Computational Geometry
Chapter 5: Polygon Triangulation

Sándor P. Fekete

Algorithms Division
Department of Computer Science
TU Braunschweig

- Ch 1: Introduction
- Ch 2: Convex hulls
- Ch 3: Closest pairs
- Ch 4: Voronoi diagrams
- Ch 5: Polygon triangulation
- Ch 6: Point triangulation
- Ch 7: Location problems
- Ch 8: Minimum area polygons

Technische Universität Braunschweig

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

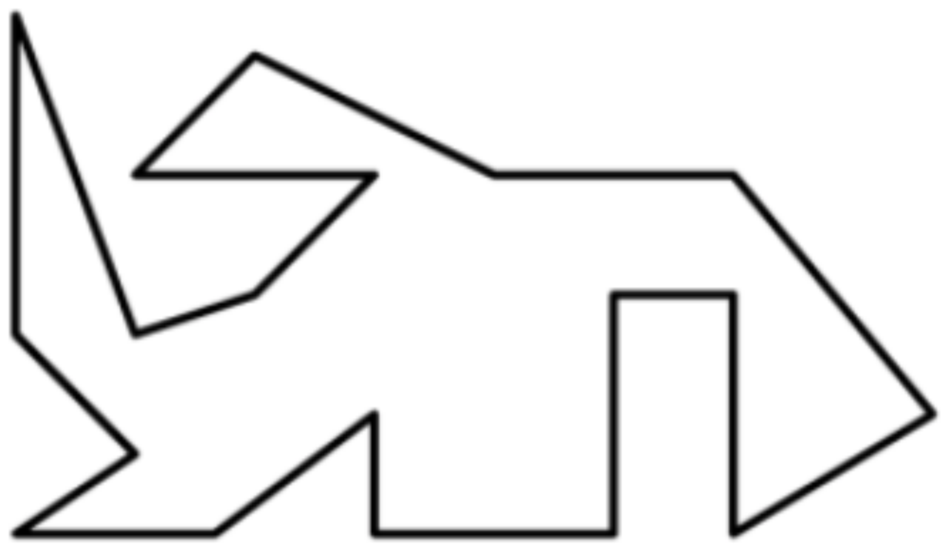
Problem 5.1

Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

Problem 5.1

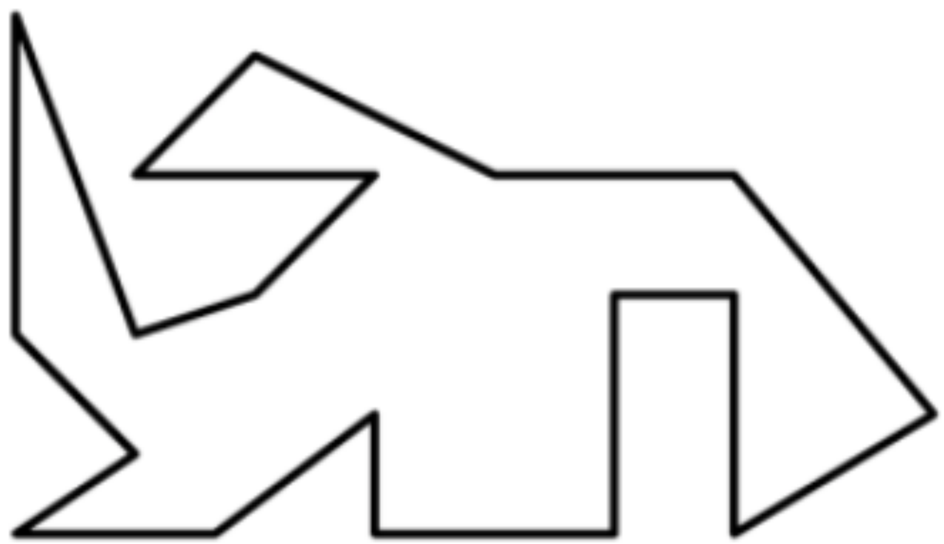
Given: A (simple) polygon P in $\mathbb{R}^2$



Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

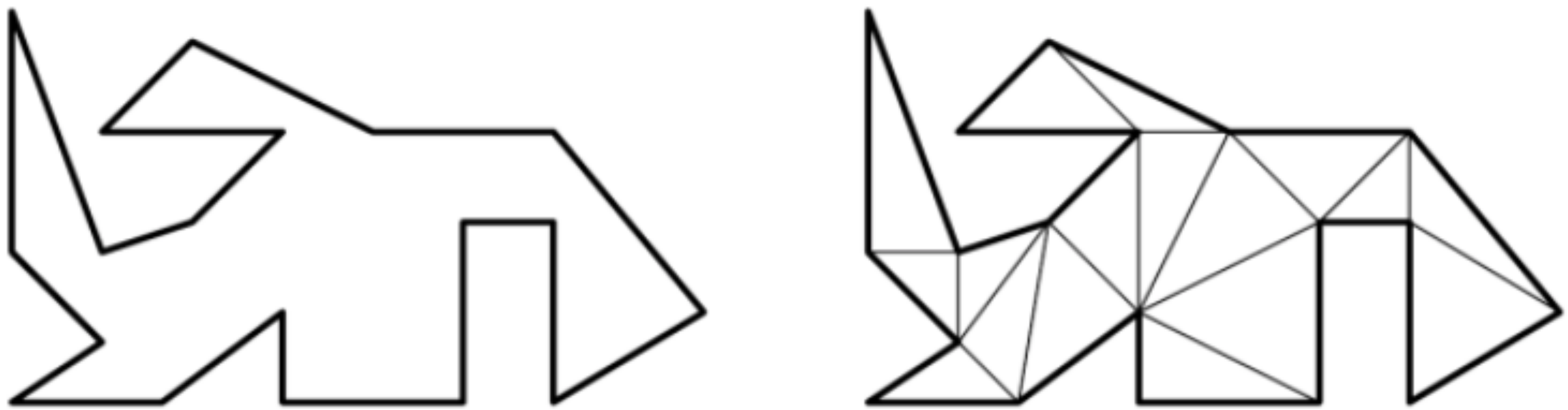
Wanted: Subdivision of P by disjoint vertex connections into triangles



Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

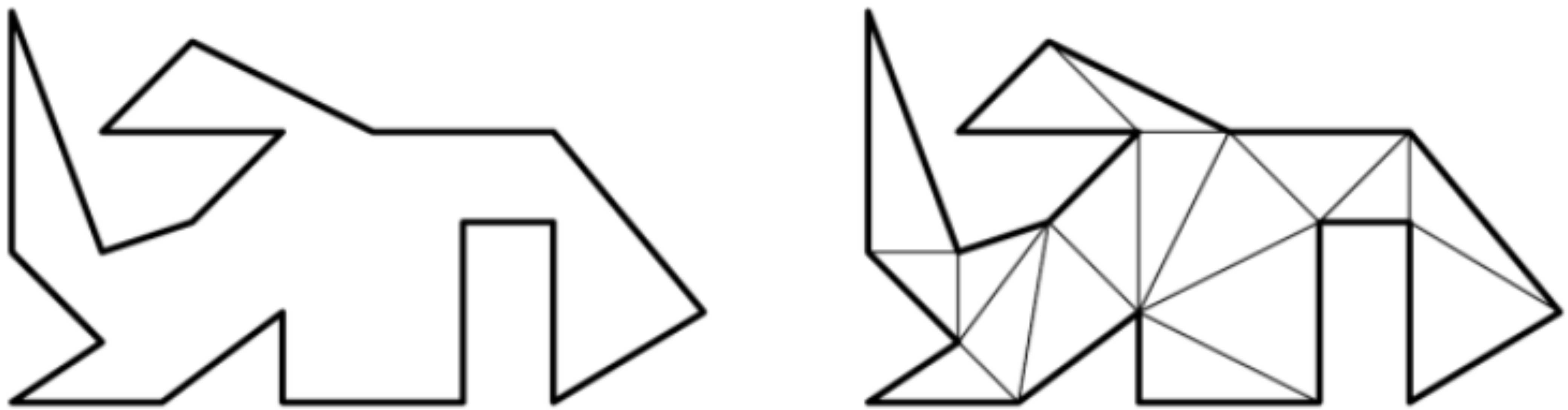
Wanted: Subdivision of P by disjoint vertex connections into triangles



Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

Wanted: Subdivision of P by disjoint vertex connections into triangles

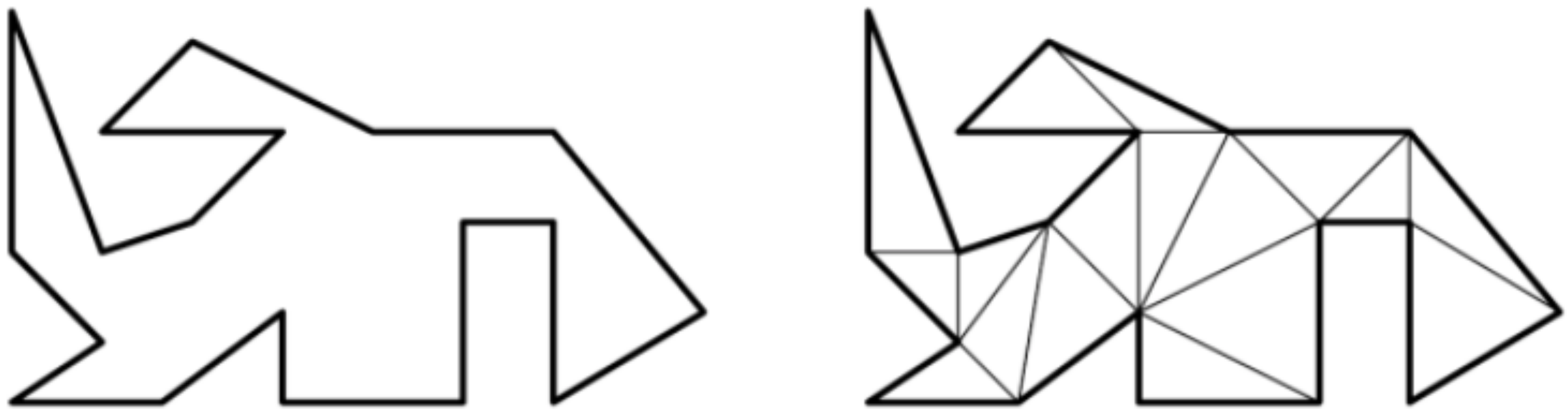


Definition 5.2

Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

Wanted: Subdivision of P by disjoint vertex connections into triangles



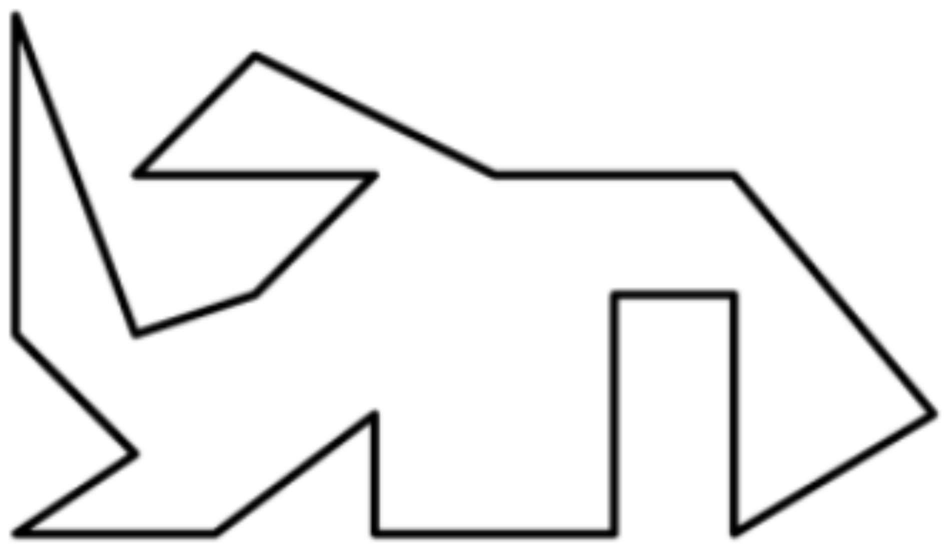
Definition 5.2

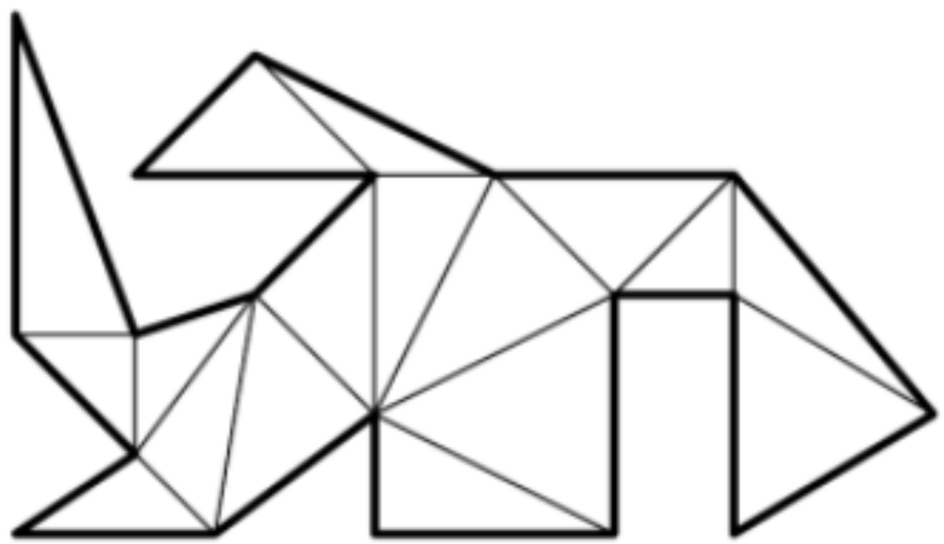
- A *diagonal* in a polygon connects two distinct vertices; its interior lies inside the polygon.

Problem 5.1

Given: A (simple) polygon P in $\mathbb{R}^2$

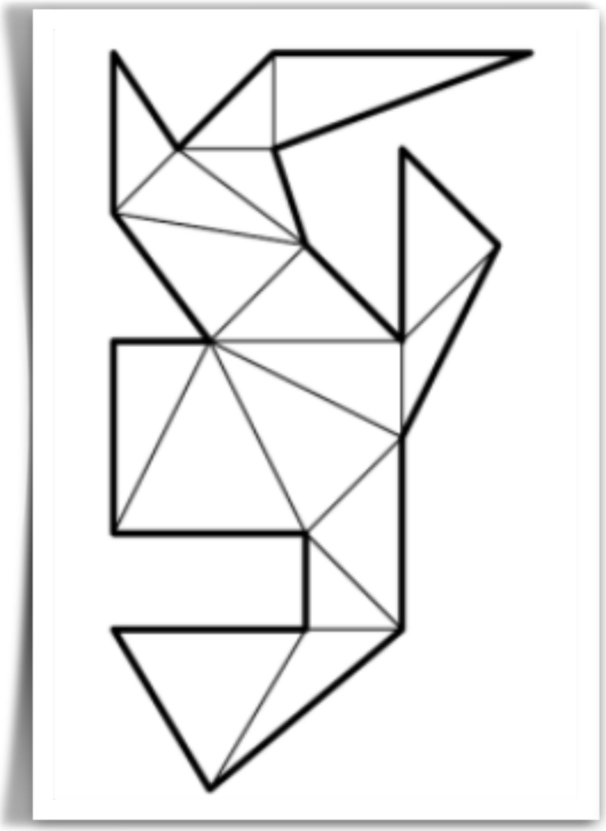
Wanted: Subdivision of P by disjoint vertex connections into triangles

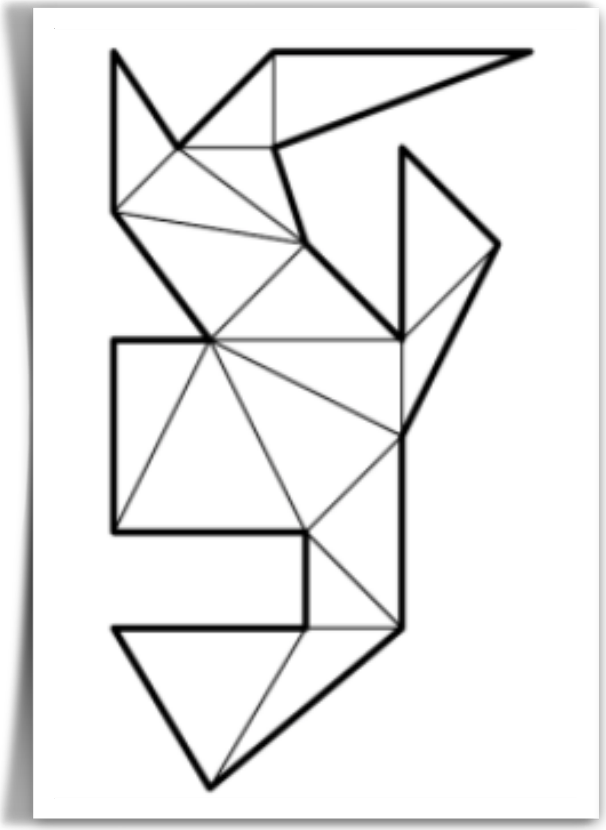


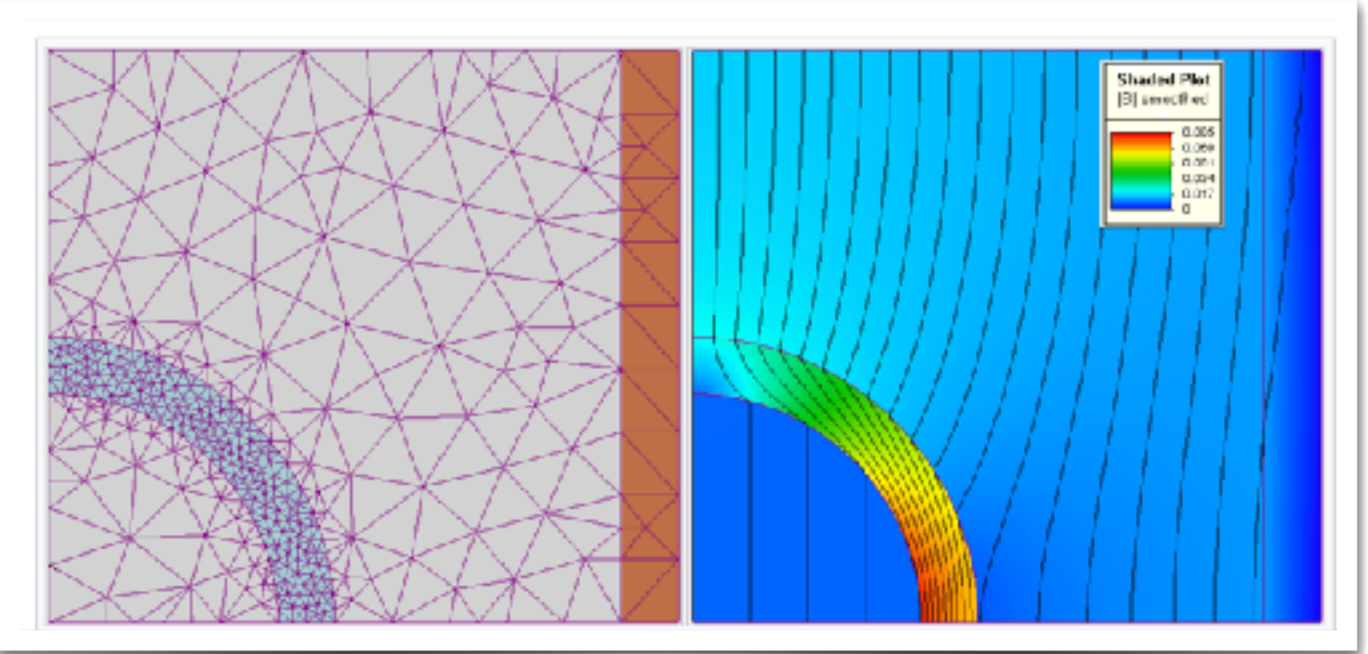


Definition 5.2

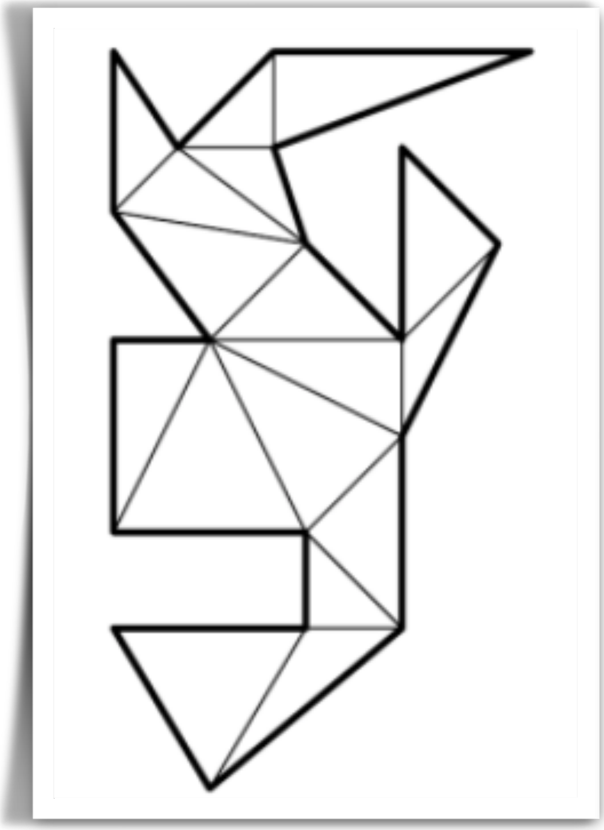
- A *diagonal* in a polygon connects two distinct vertices; its interior lies inside the polygon.
- Two different diagonals in a triangulation may share end points, but nothing else.

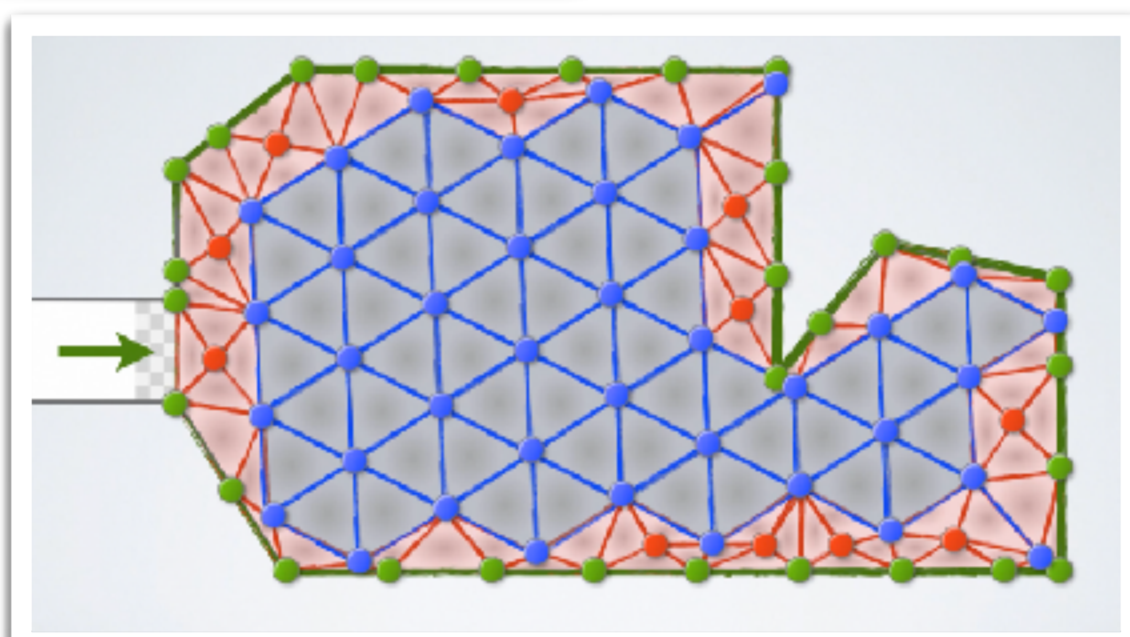


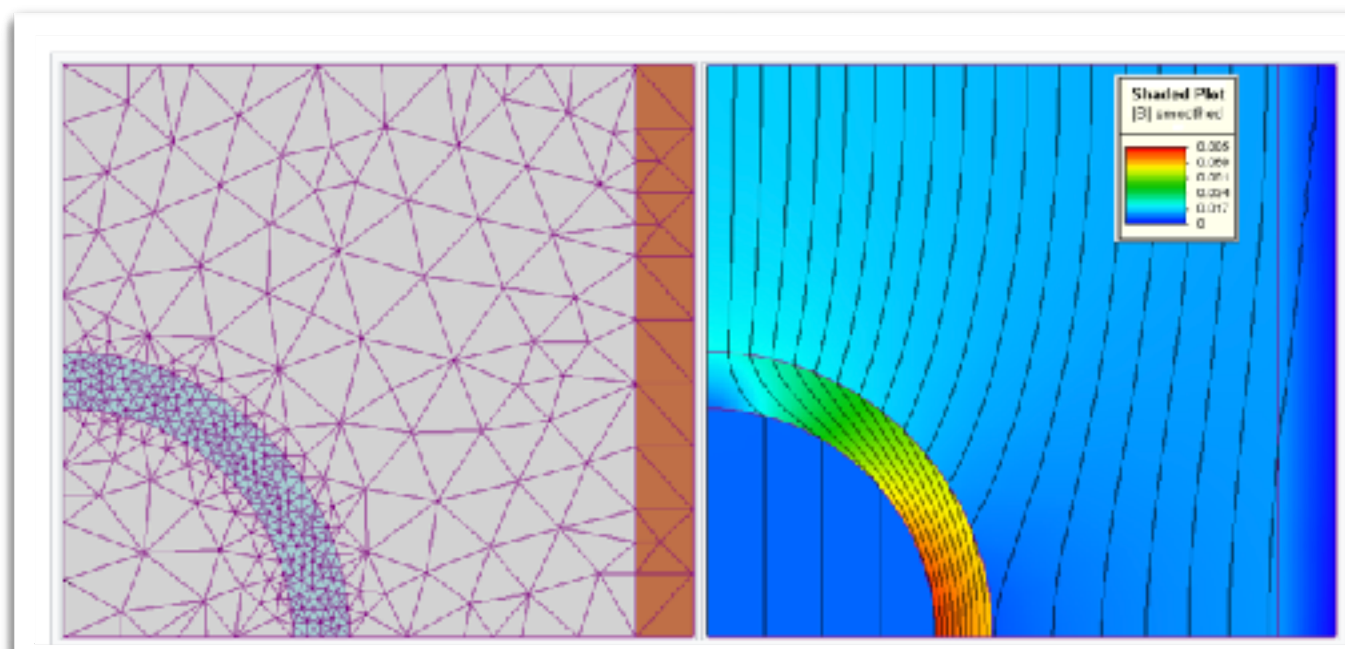


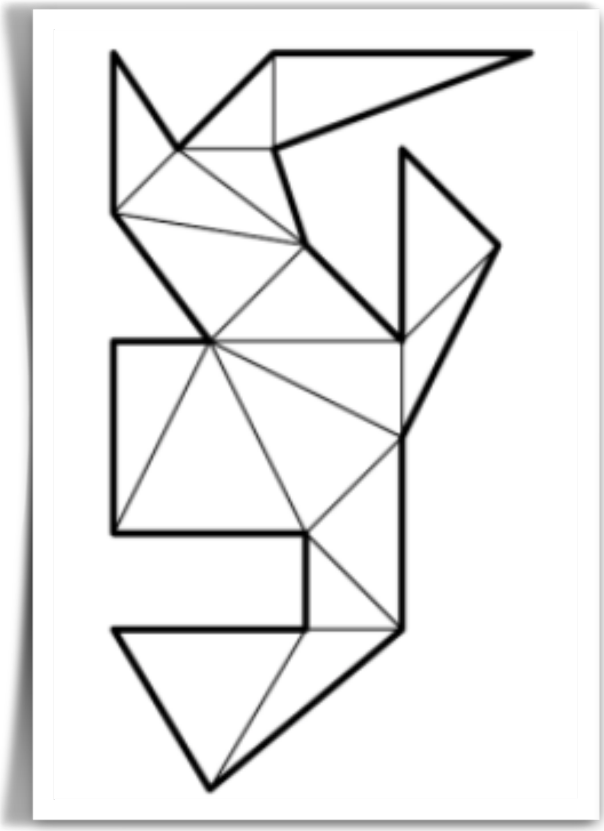


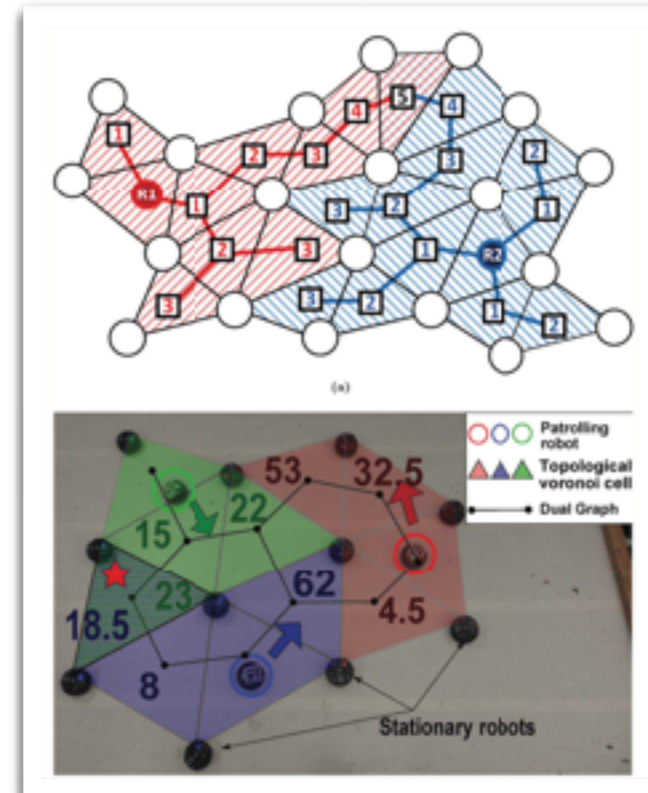
Technische Universität Braunschweig

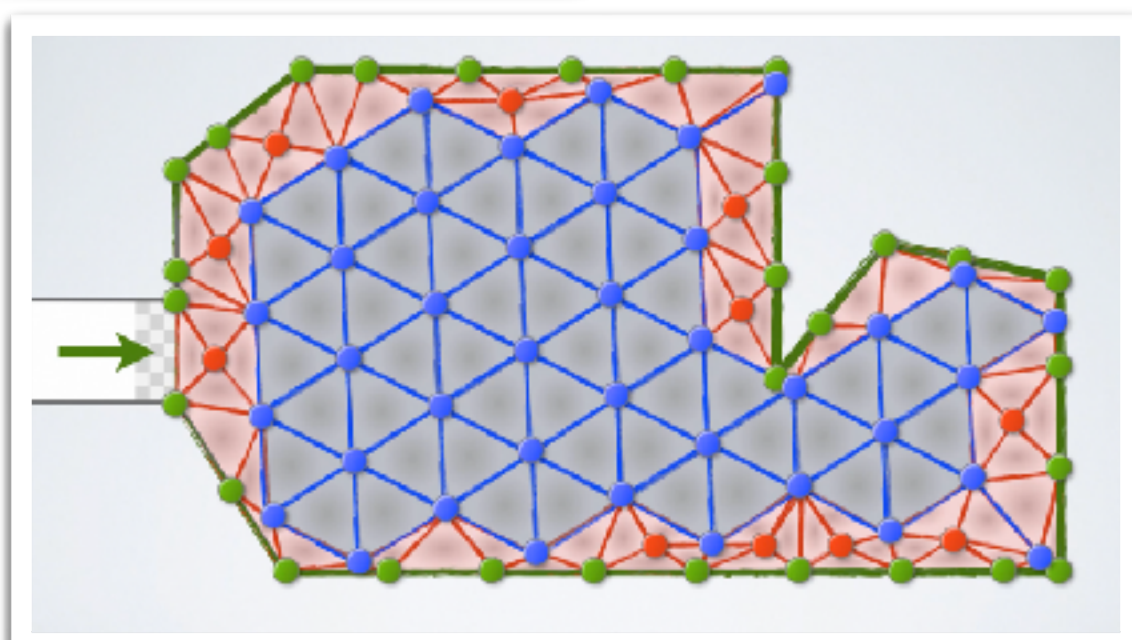


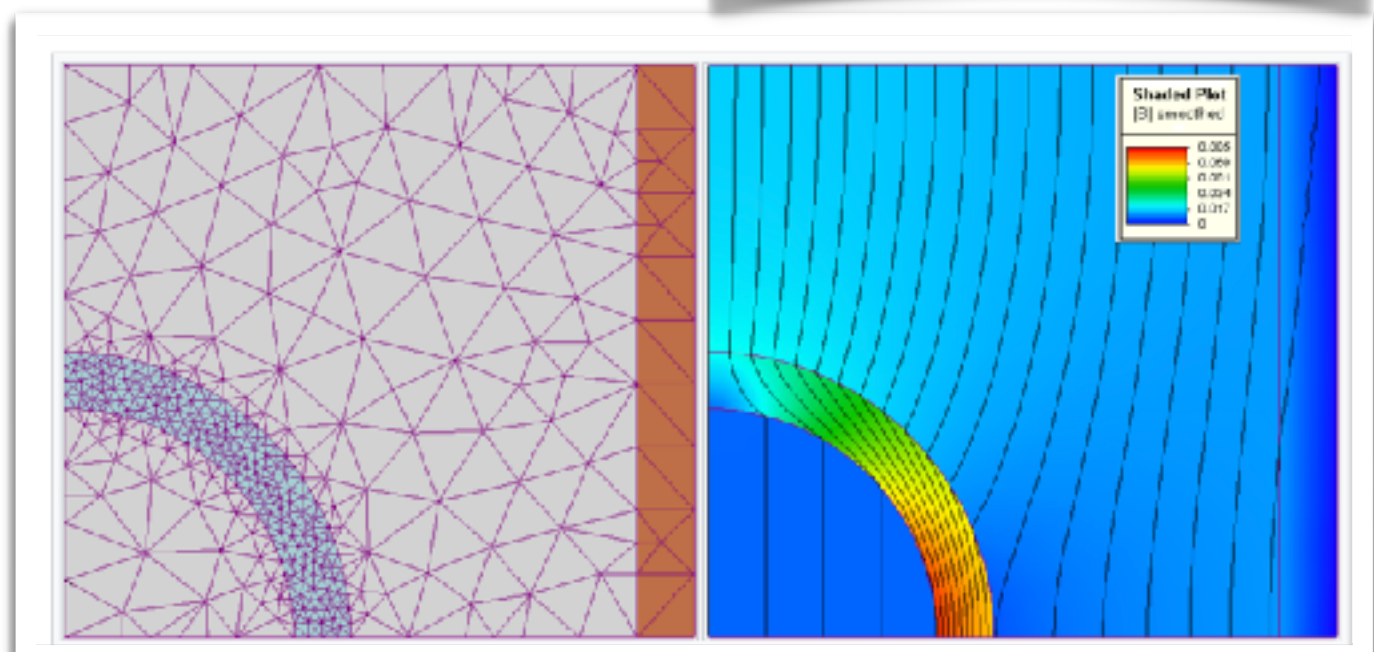


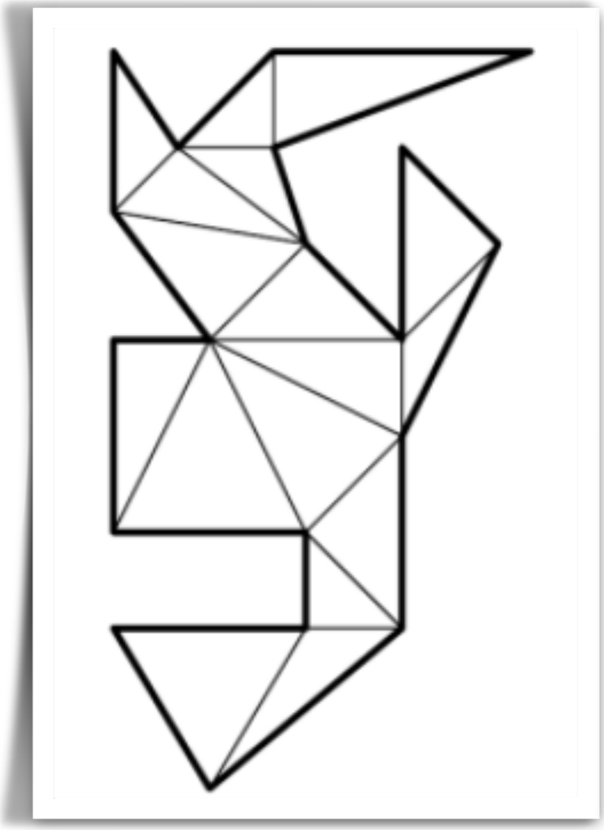

Shaded Plot



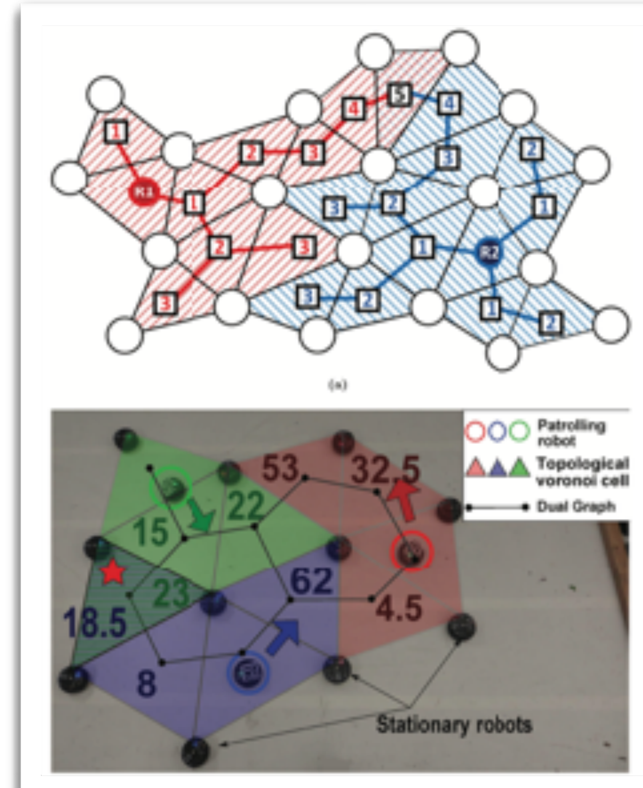

53
32.5
15
22
23
62
18.5
4.5
8
Patrolling robot
Topological voronoi cell
Dual Graph
Stationary robots

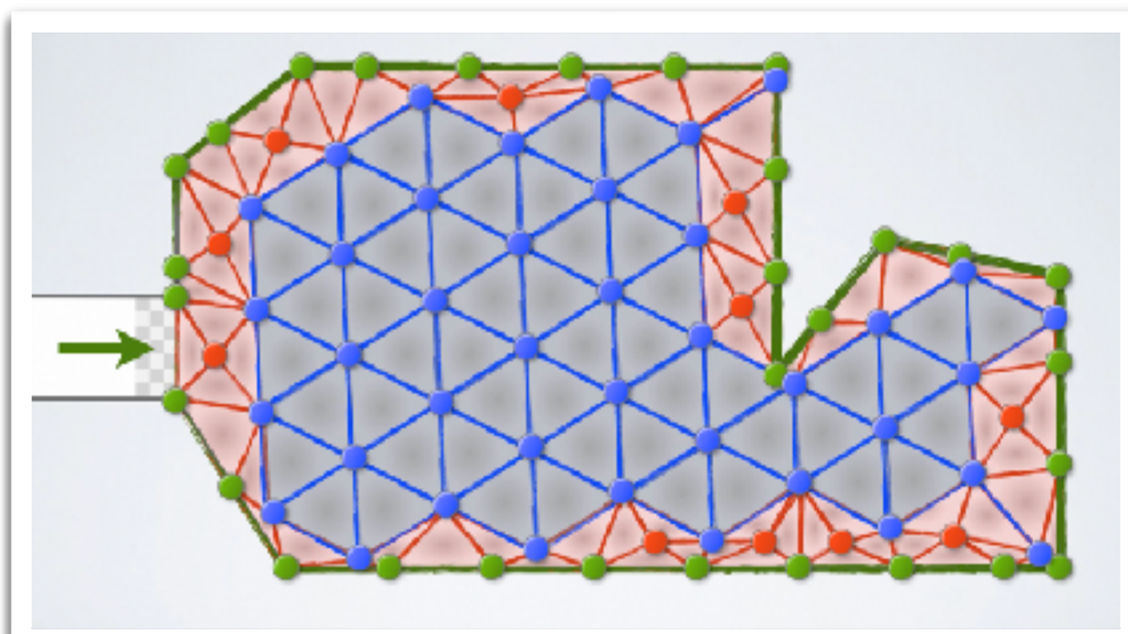


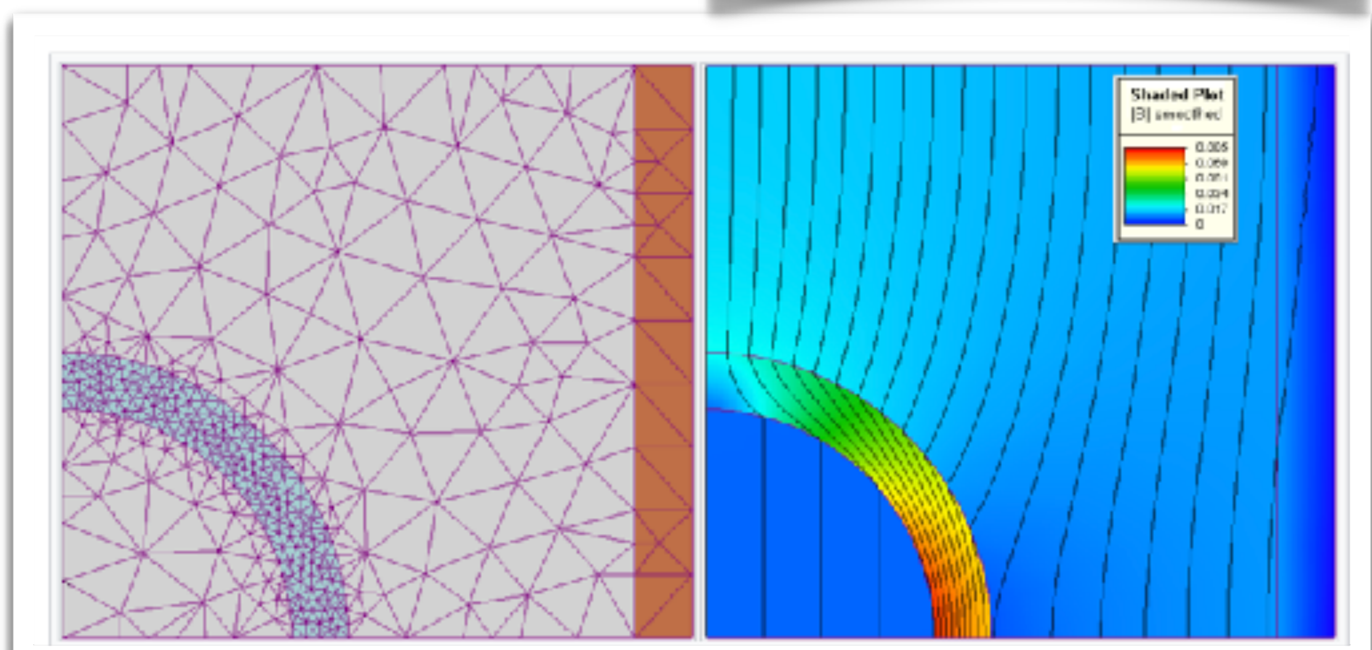

Shaded Plot

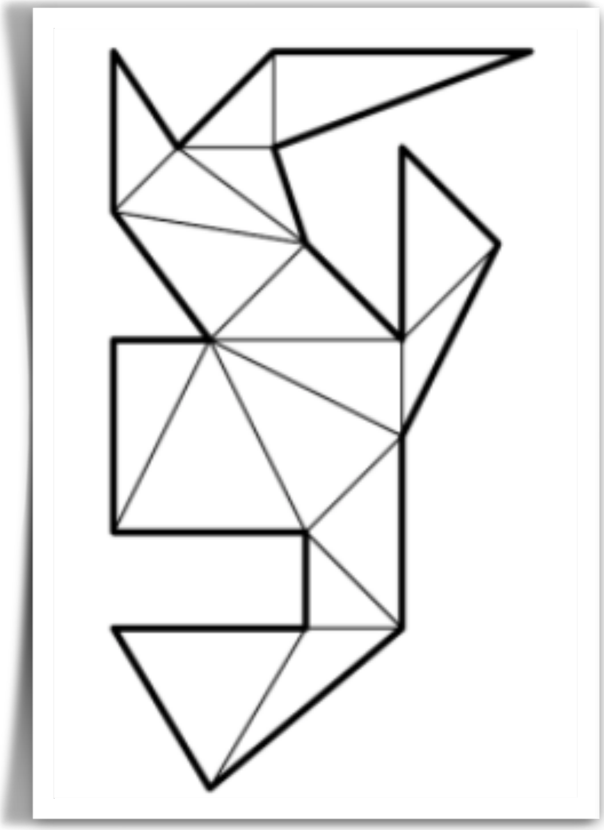


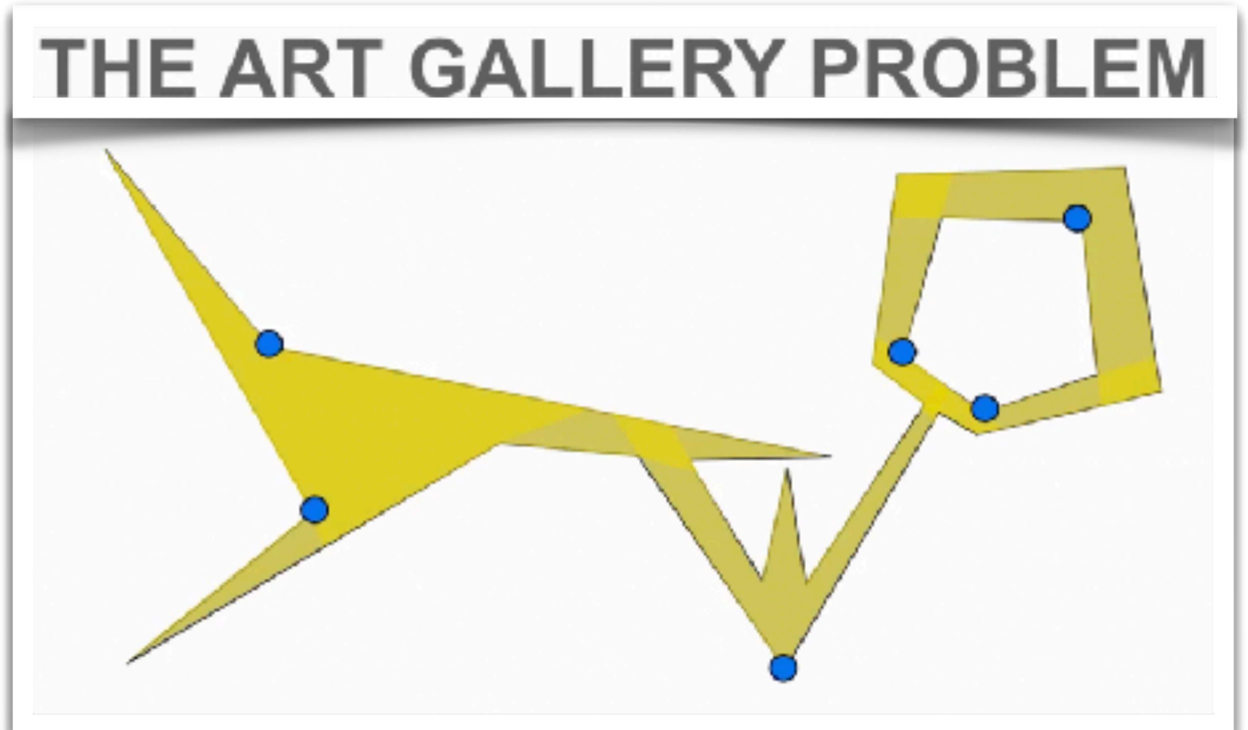
THE ART GALLERY PROBLEM

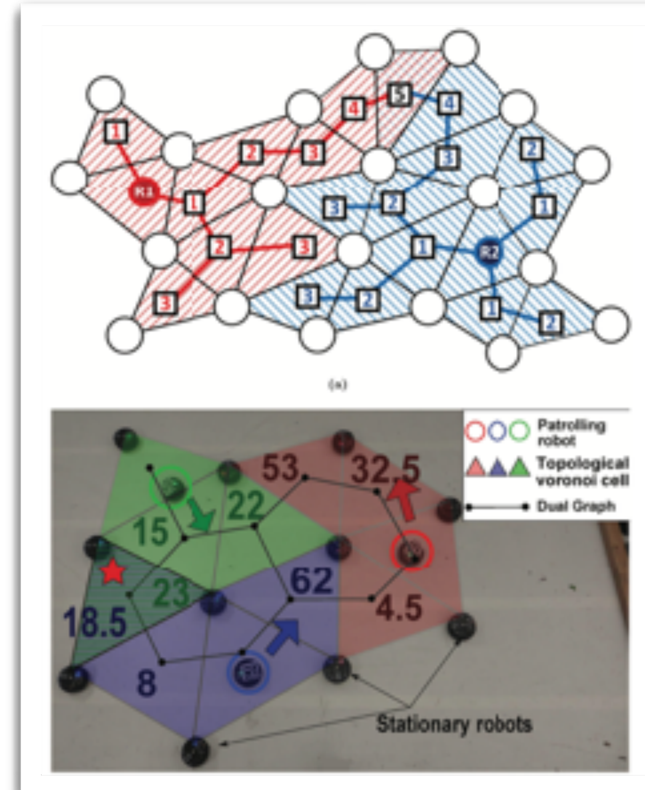


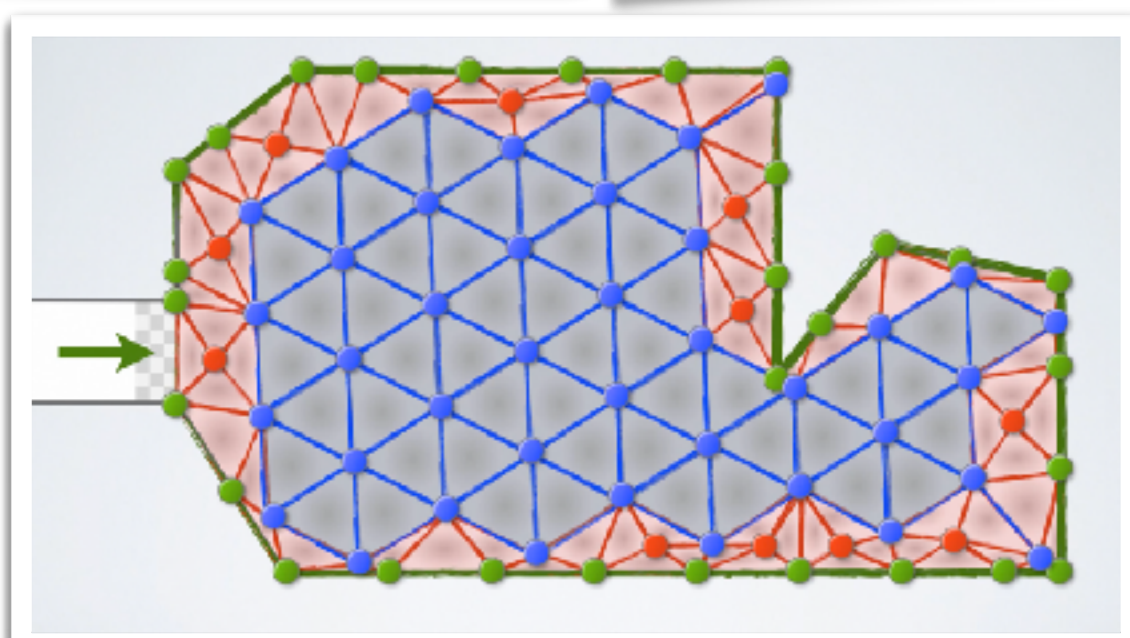


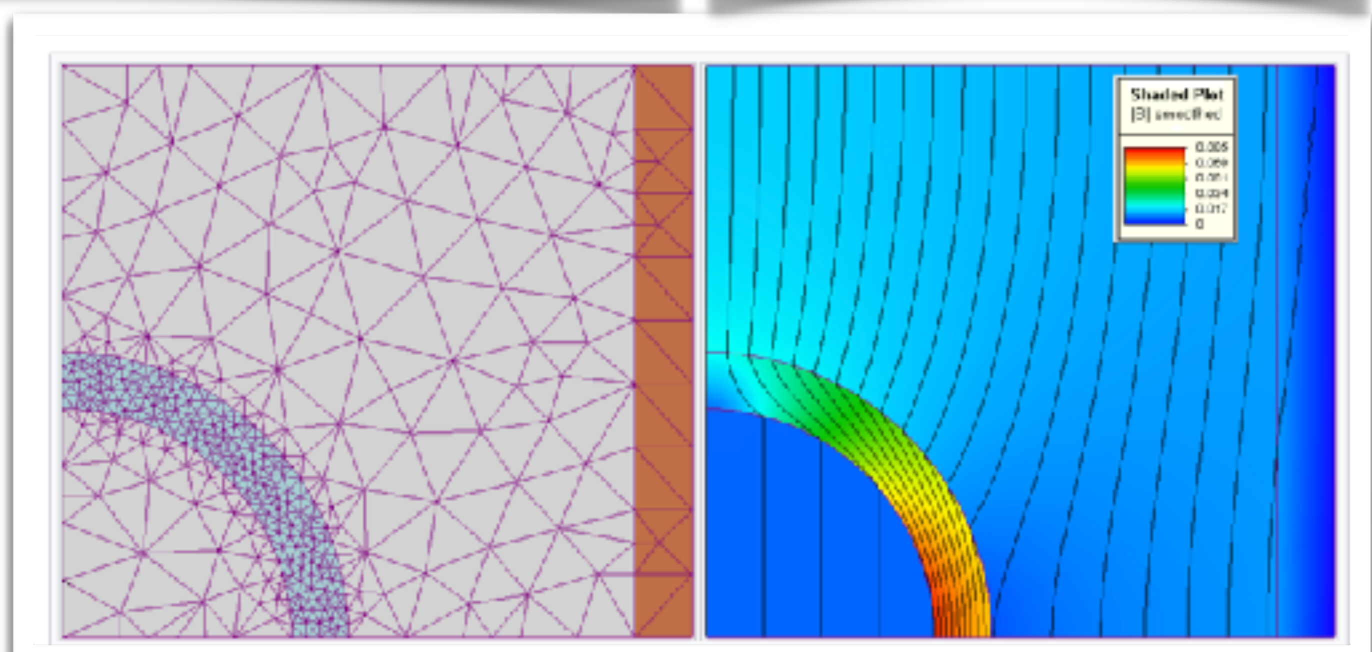





THE ART GALLERY PROBLEM


Patrolling robot
Topological voronoi cell
Dual Graph
53
32.5
22
15
23
62
18.5
4.5
8
Stationary robots




Shaded Plot

@eimnihellas

Technische Universität Braunschweig

Captain Triangular

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

Technische Universität Braunschweig

When does a polygon have a triangulation?

When does a polygon have a triangulation?

Lemma 5.3

When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.

When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.

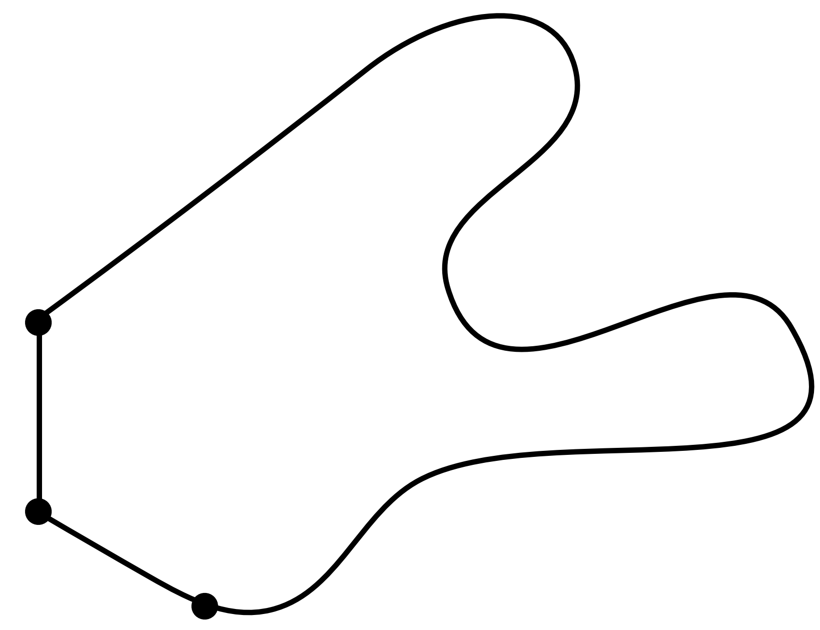
Proof:

When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.

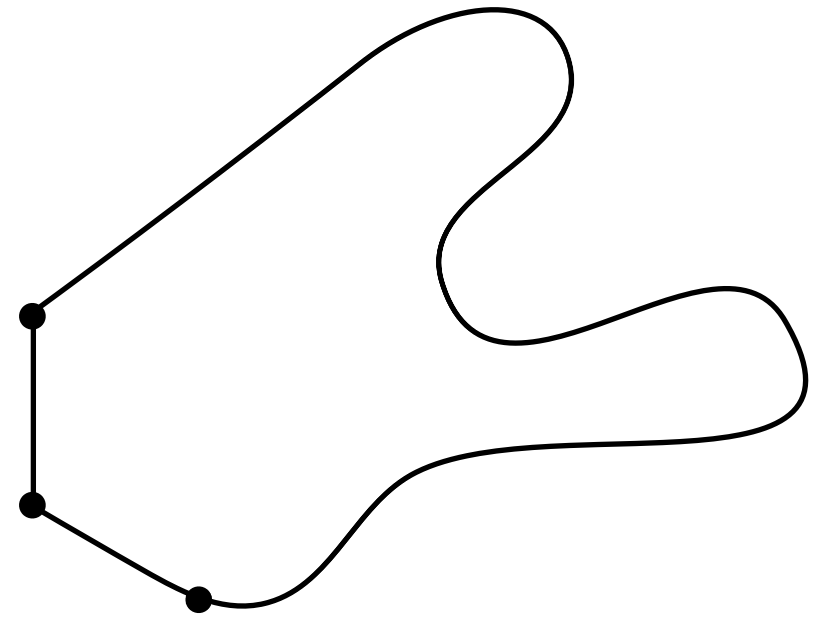
Proof:



When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.



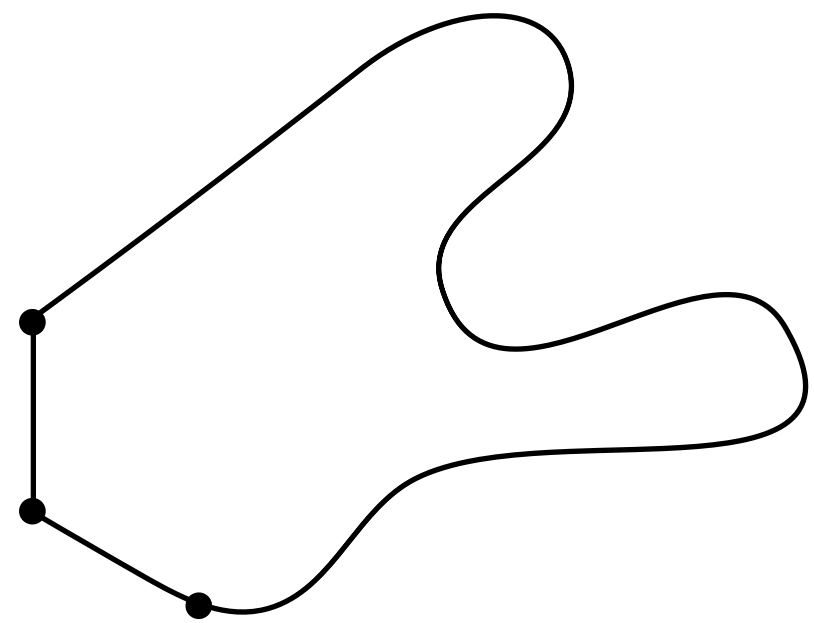
Proof:

Consider the bottommost
of all leftmost vertices.

When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.



Proof:

Consider the bottommost of all leftmost vertices.

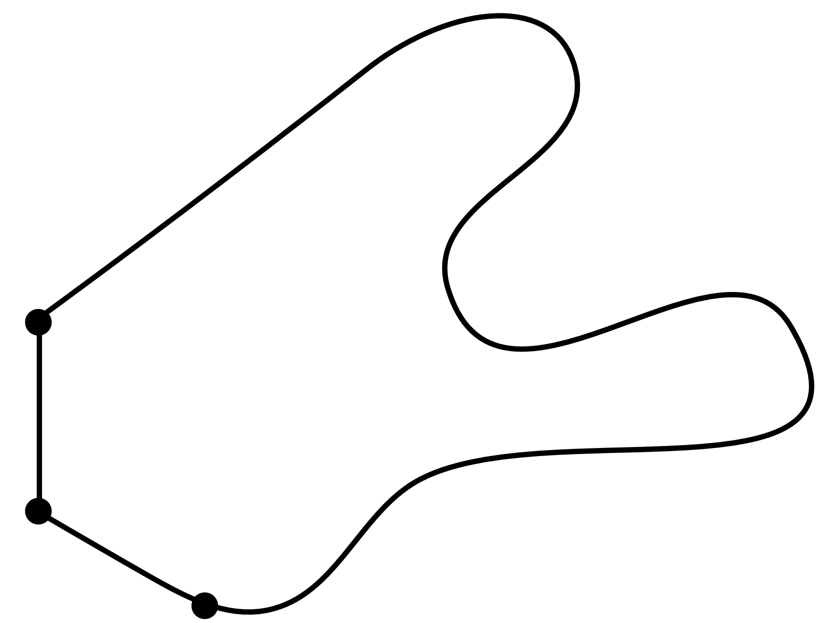
Exercise 5.4

Technische Universität Braunschweig

When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.



Proof:

Consider the bottommost
of all leftmost vertices.

Exercise 5.4

Every polygon has at least three convex vertices.

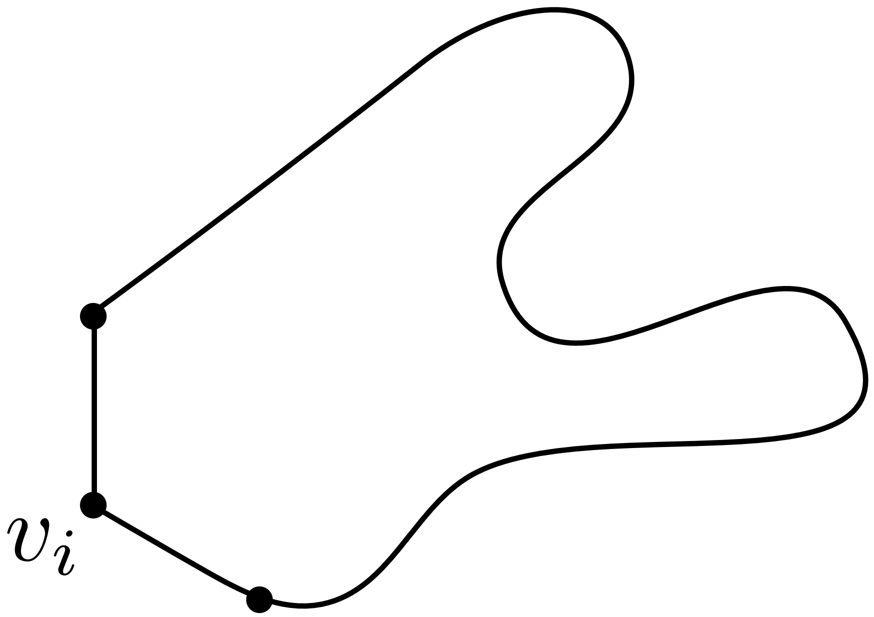
When does a polygon have a triangulation?

Lemma 5.3

Every polygon has a convex vertex.

Proof:

Consider the bottommost
of all leftmost vertices.



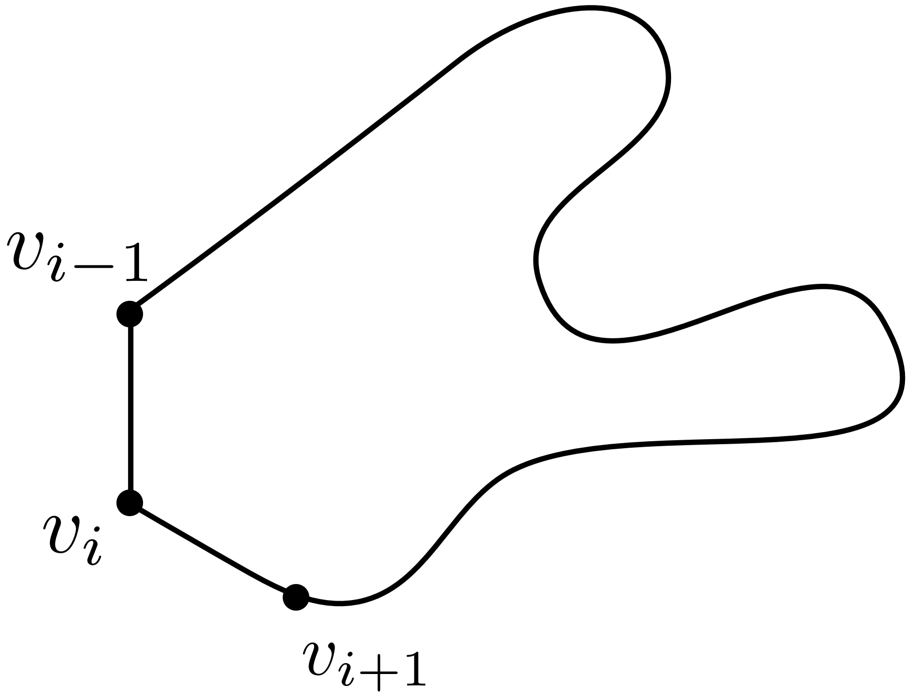
Exercise 5.4

Every polygon has at least three convex vertices.

Lemma 5.5

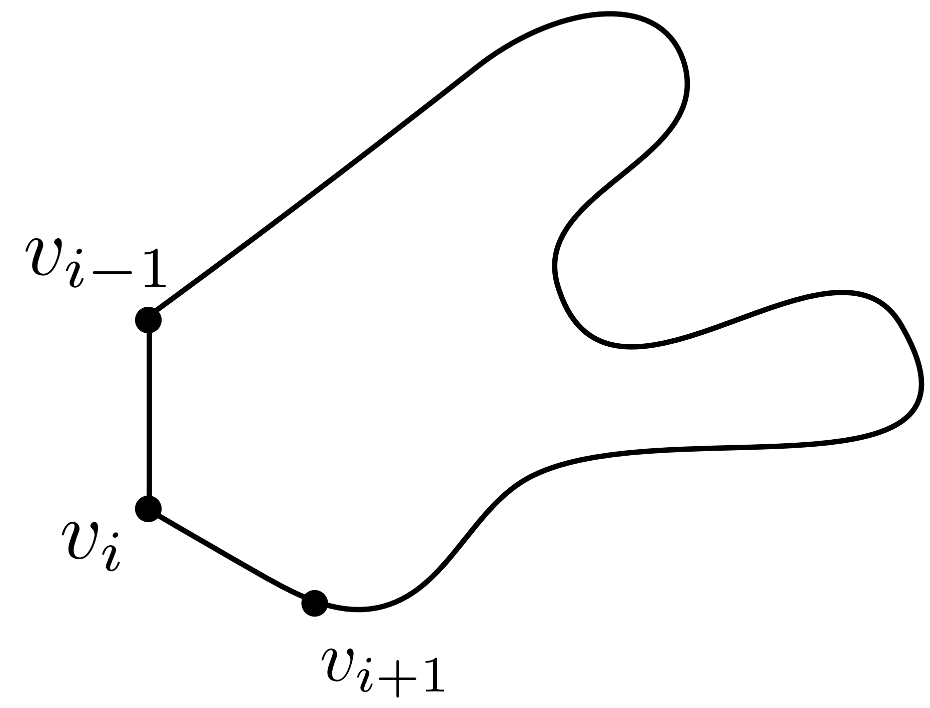
Lemma 5.5
Every polygon has a diagonal.

Lemma 5.5
Every polygon has a diagonal.



Lemma 5.5
Every polygon has a diagonal.

Proof:

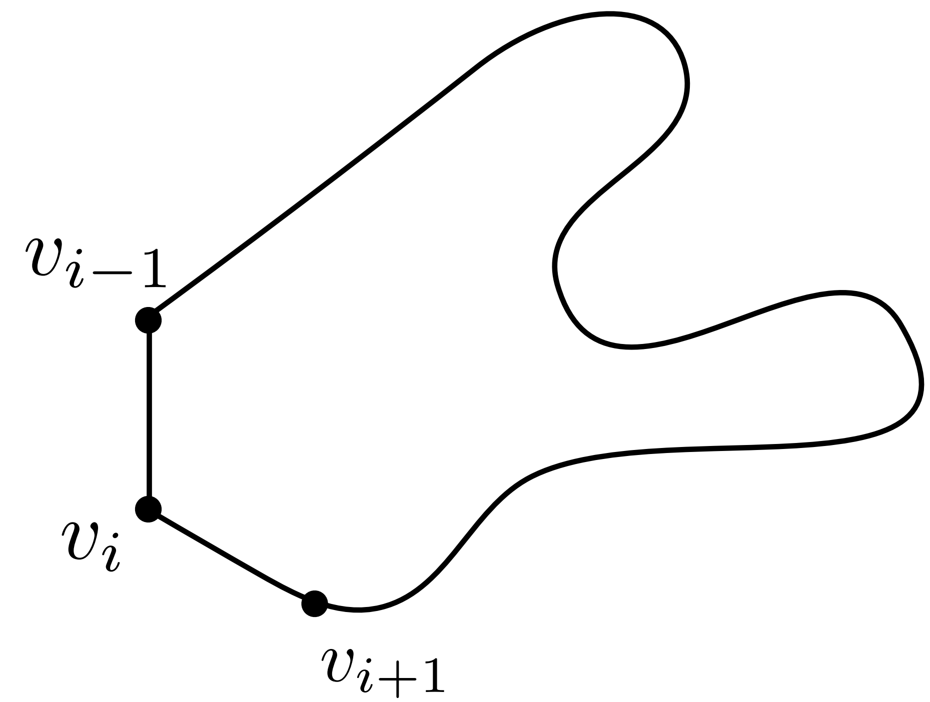


Lemma 5.5

Every polygon has a diagonal.

Proof:

Consider the connection between predecessor and successor of v_i.

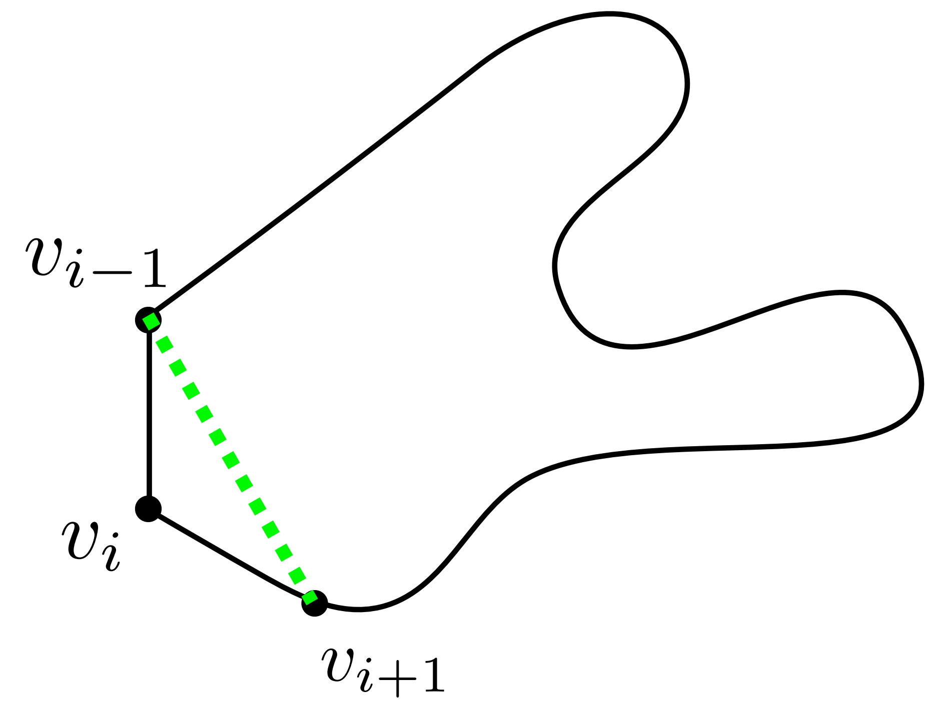


Lemma 5.5

Every polygon has a diagonal.

Proof:

Consider the connection between predecessor and successor of v_i.



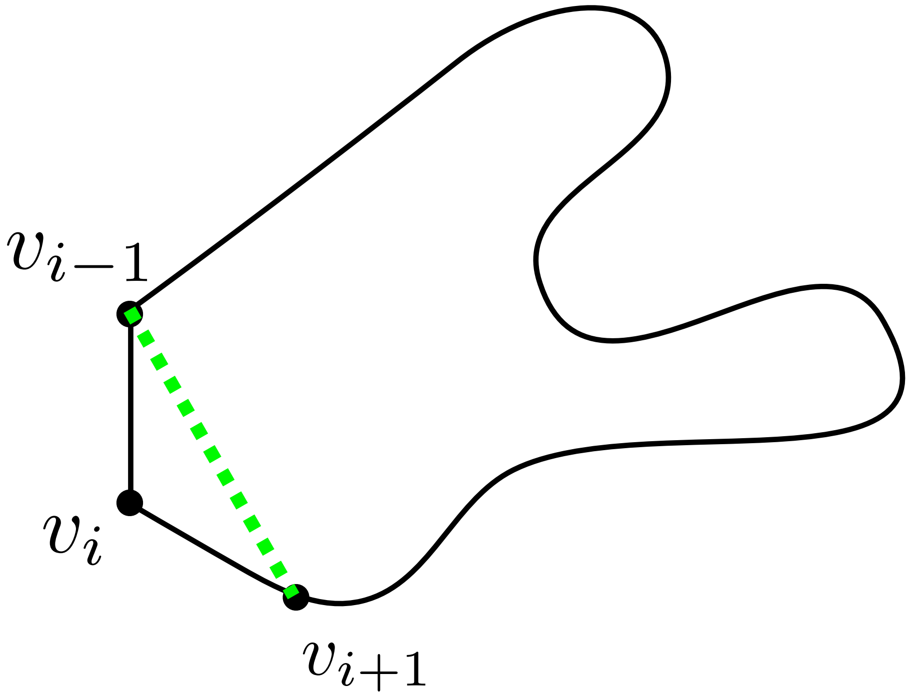
Lemma 5.5

Every polygon has a diagonal.

Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.



Technische Universität Braunschweig

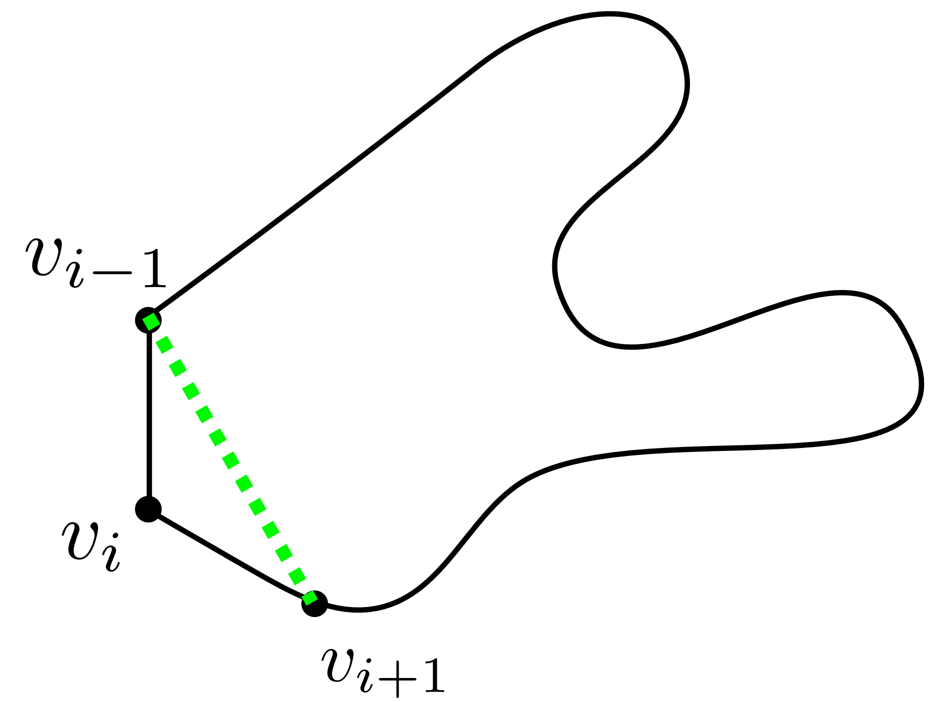
Lemma 5.5

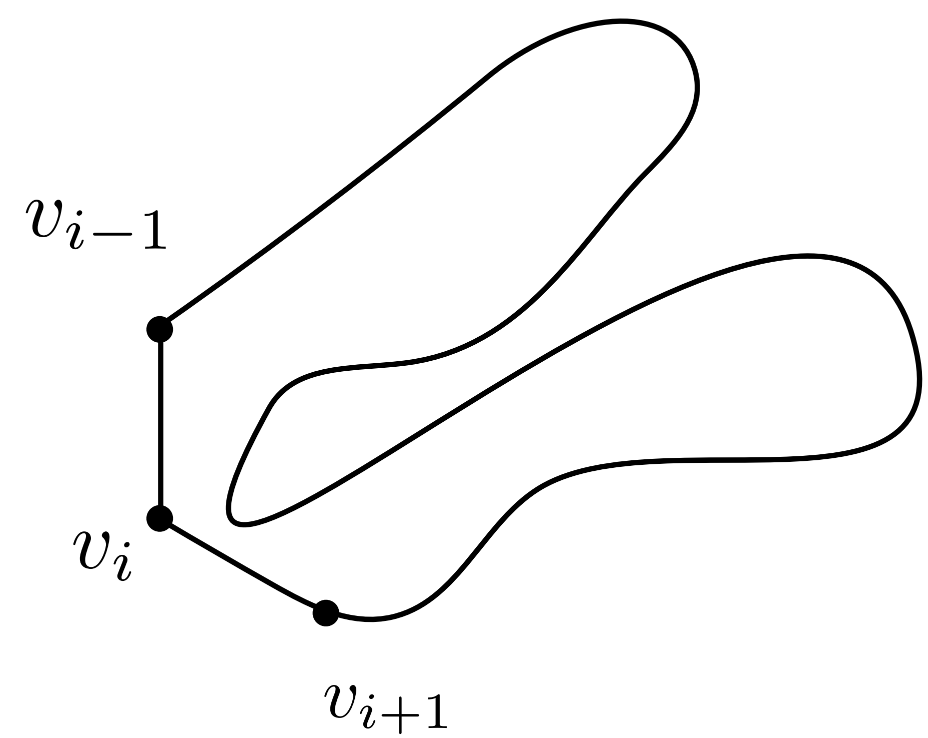
Every polygon has a diagonal.

Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.



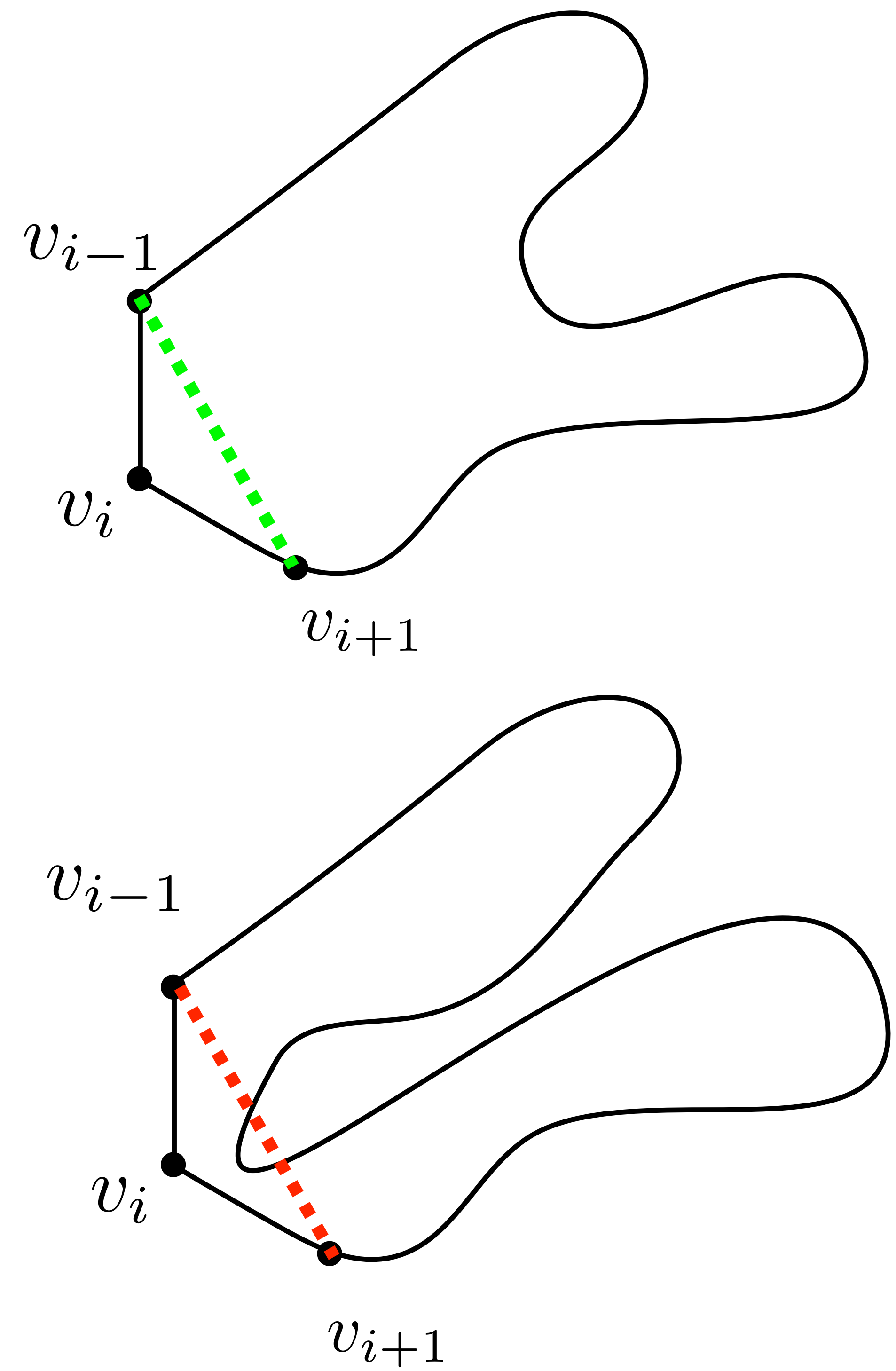


Lemma 5.5
Every polygon has a diagonal.

Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.



Lemma 5.5

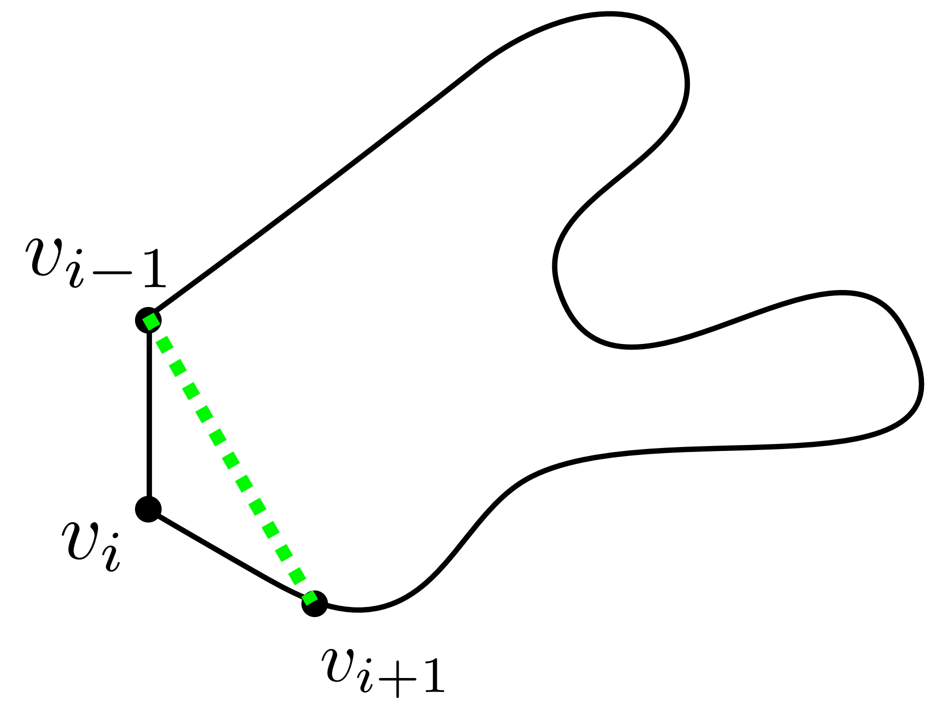
Every polygon has a diagonal.

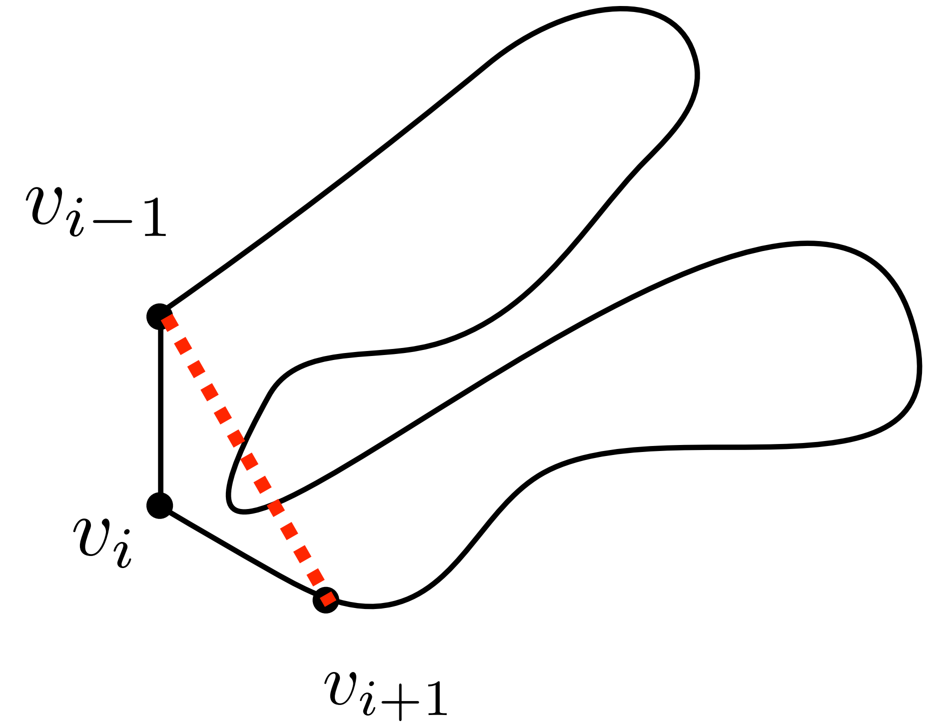
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.





Lemma 5.5

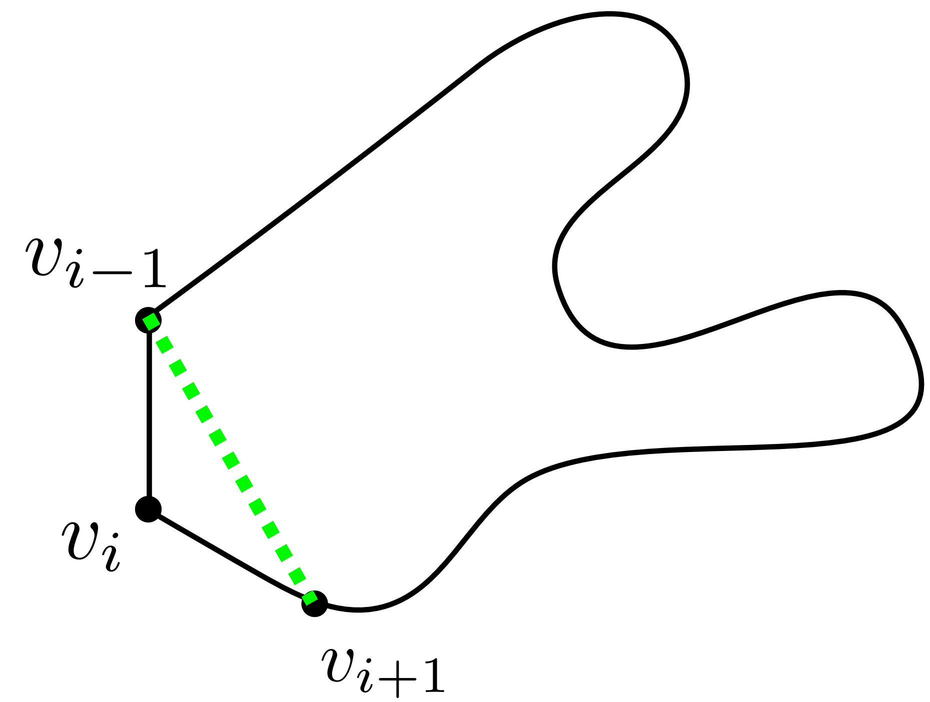
Every polygon has a diagonal.

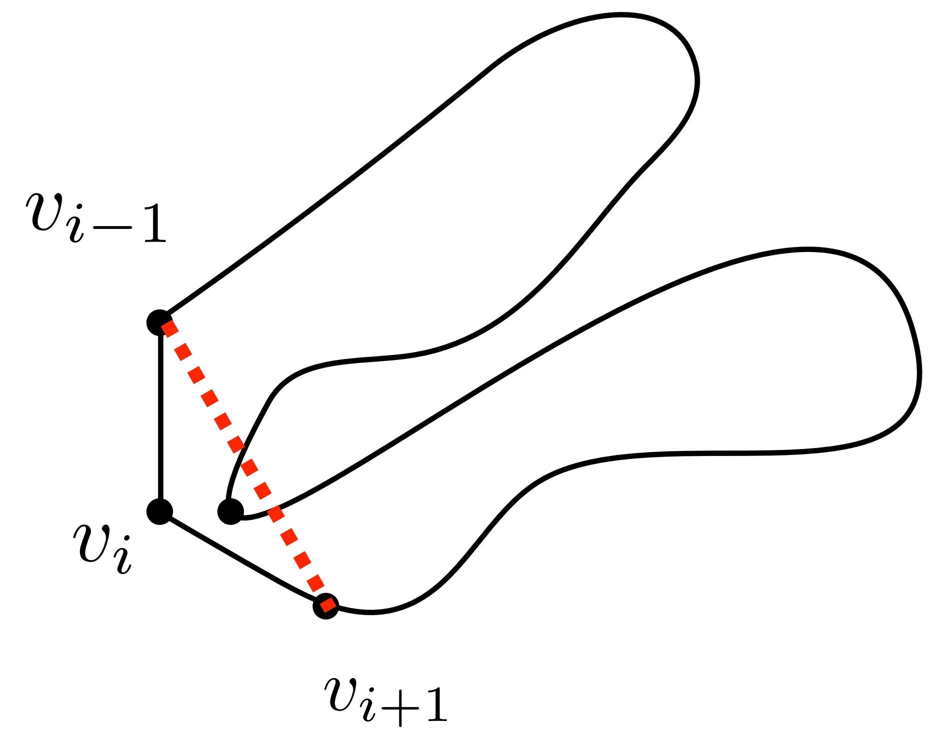
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.





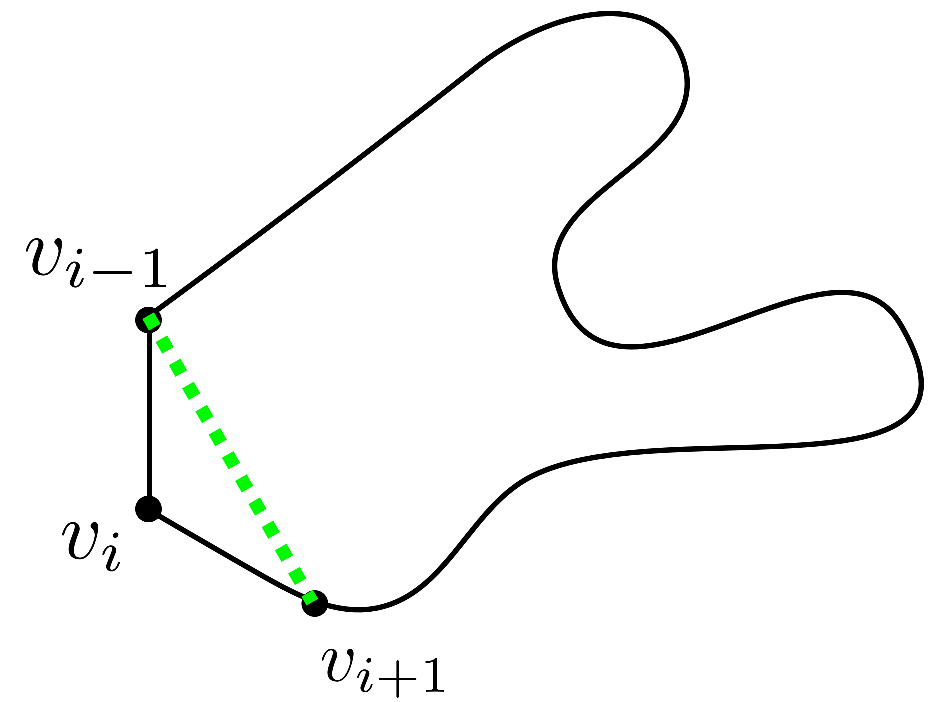
Lemma 5.5
Every polygon has a diagonal.

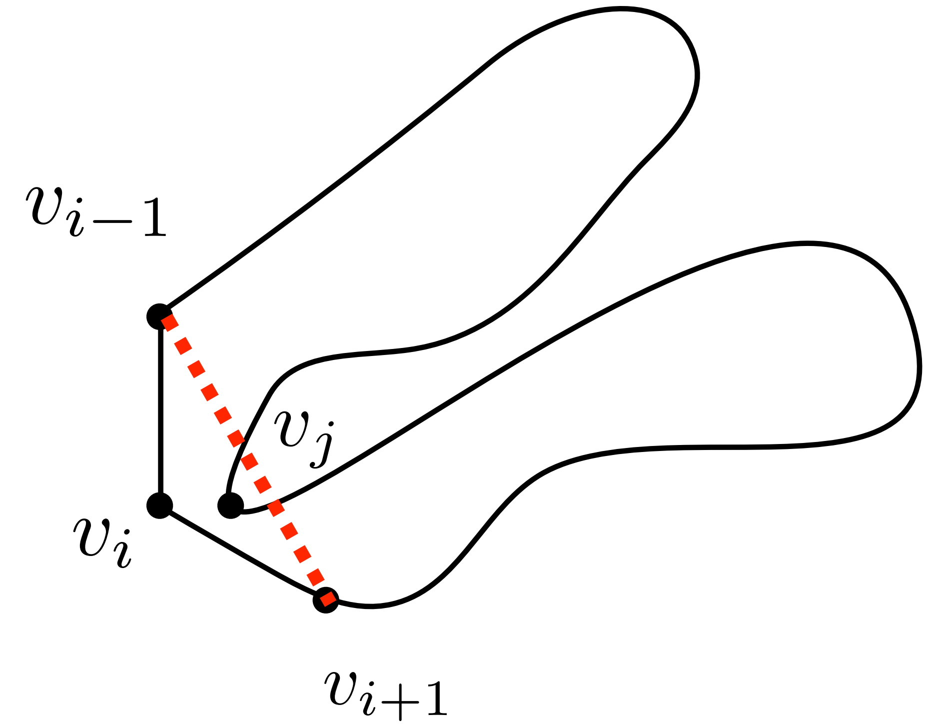
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.





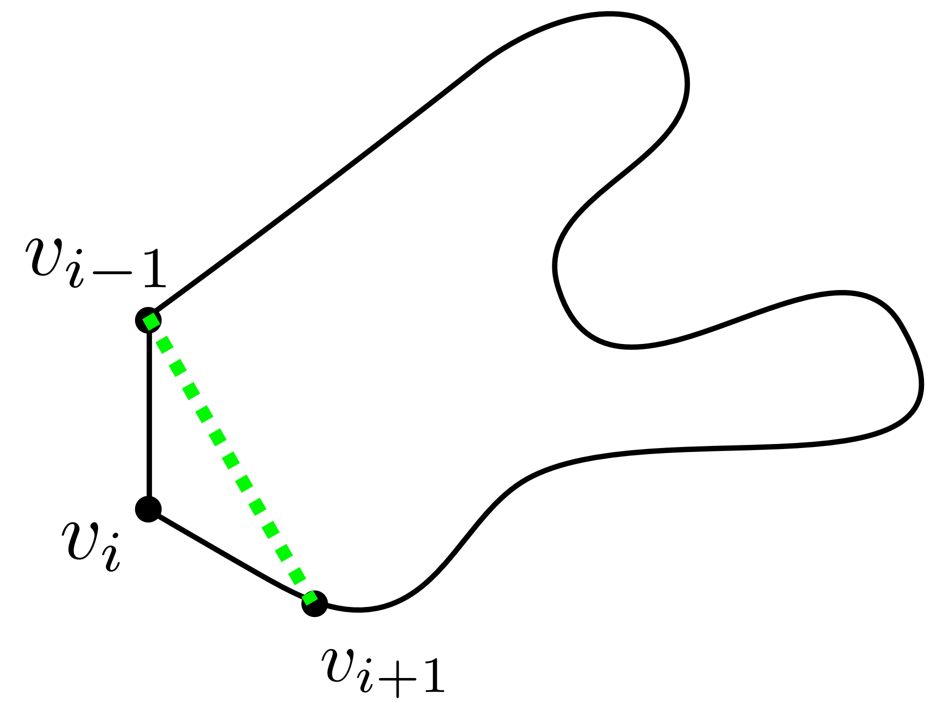
Technische Universität Braunschweig

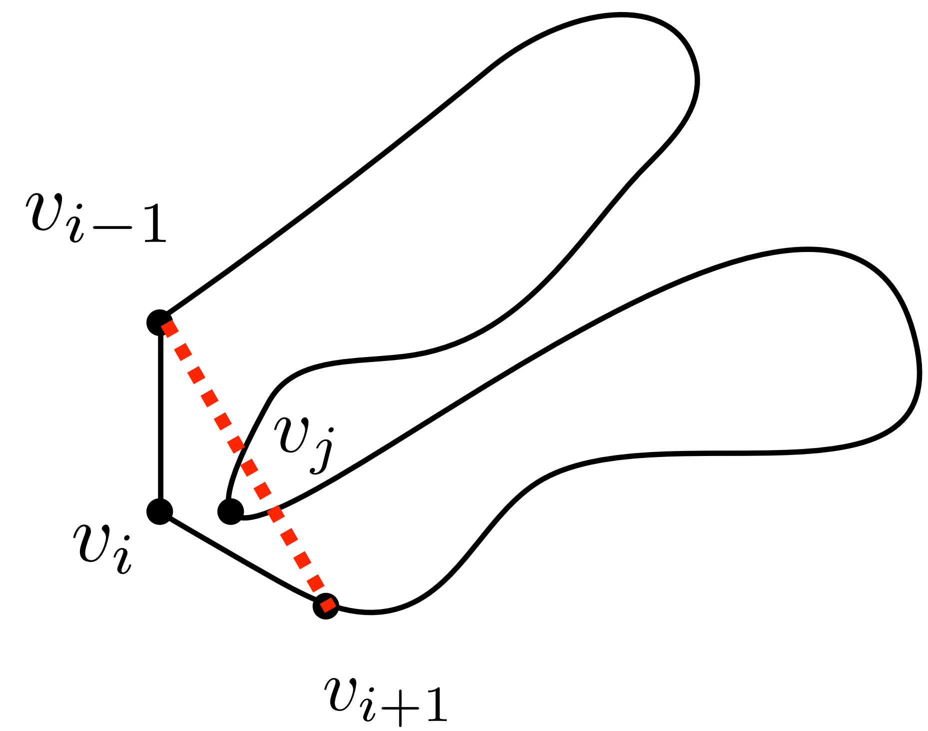
Lemma 5.5
Every polygon has a diagonal.

Proof:
Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.





Lemma 5.5

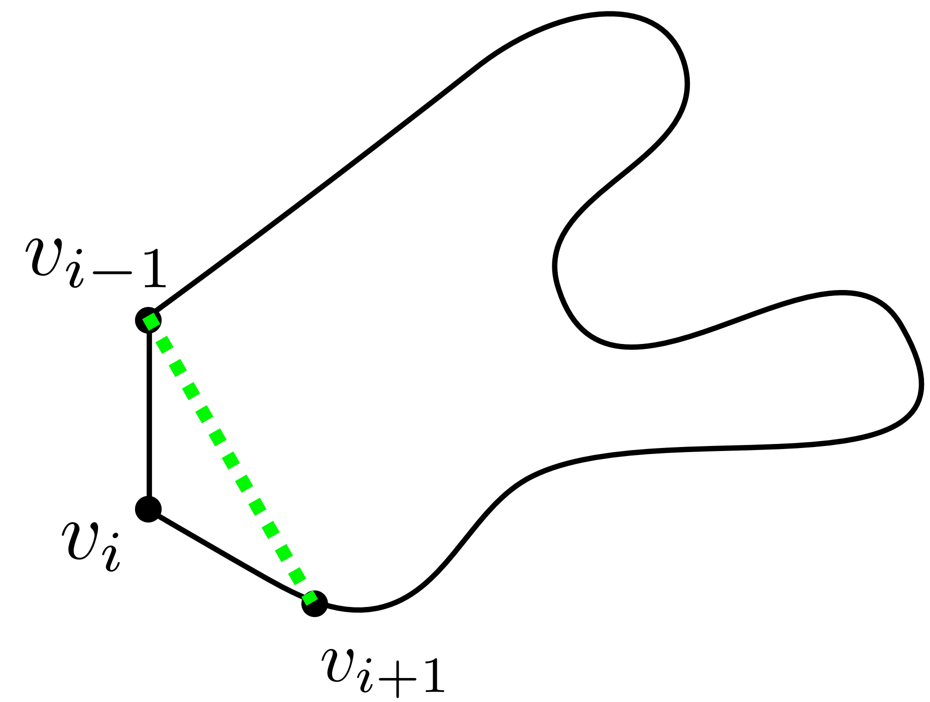
Every polygon has a diagonal.

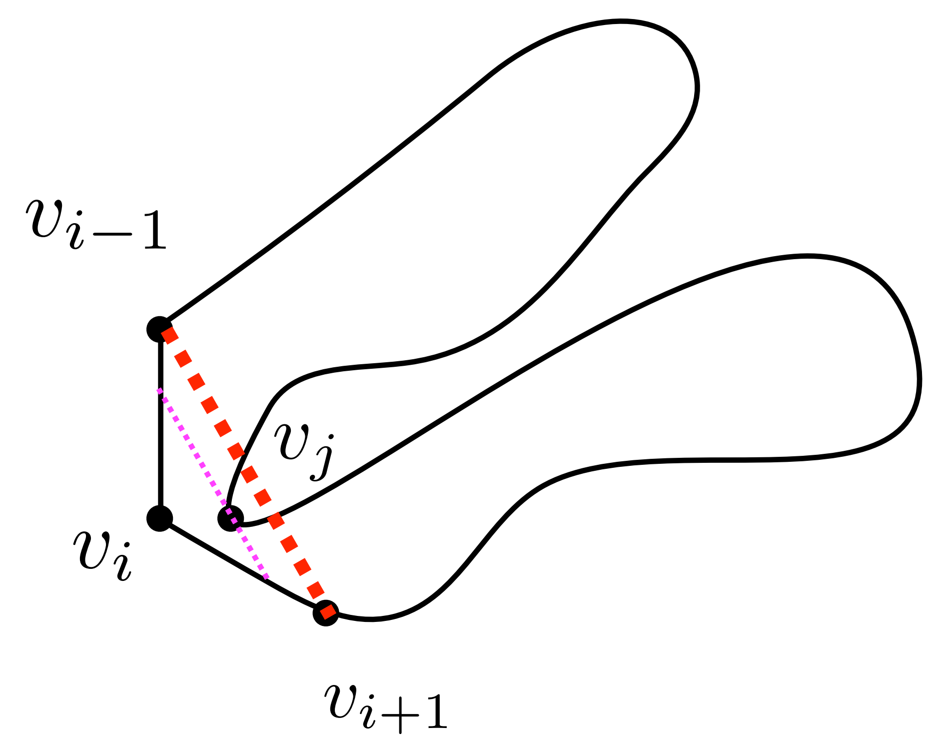
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.





Technische Universität Braunschweig

Lemma 5.5

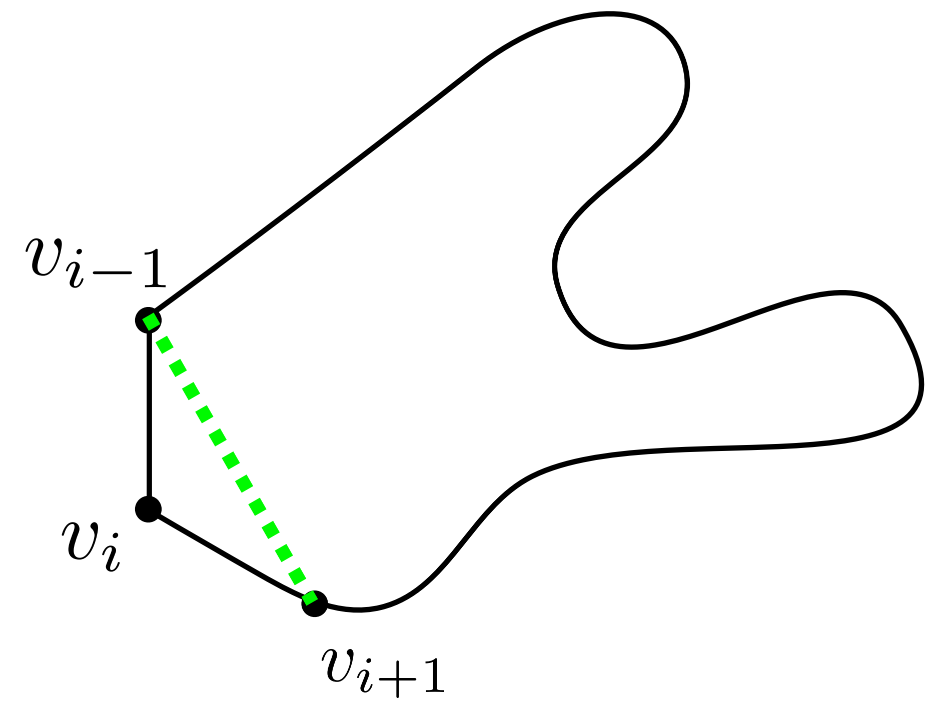
Every polygon has a diagonal.

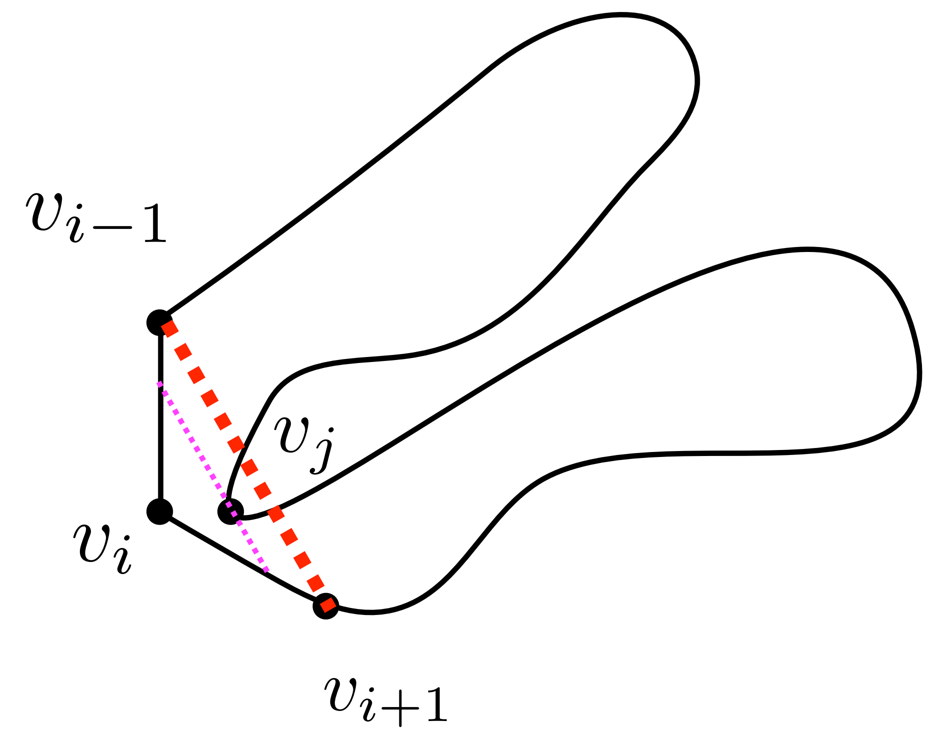
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.
The connection to v_i is a diagonal.





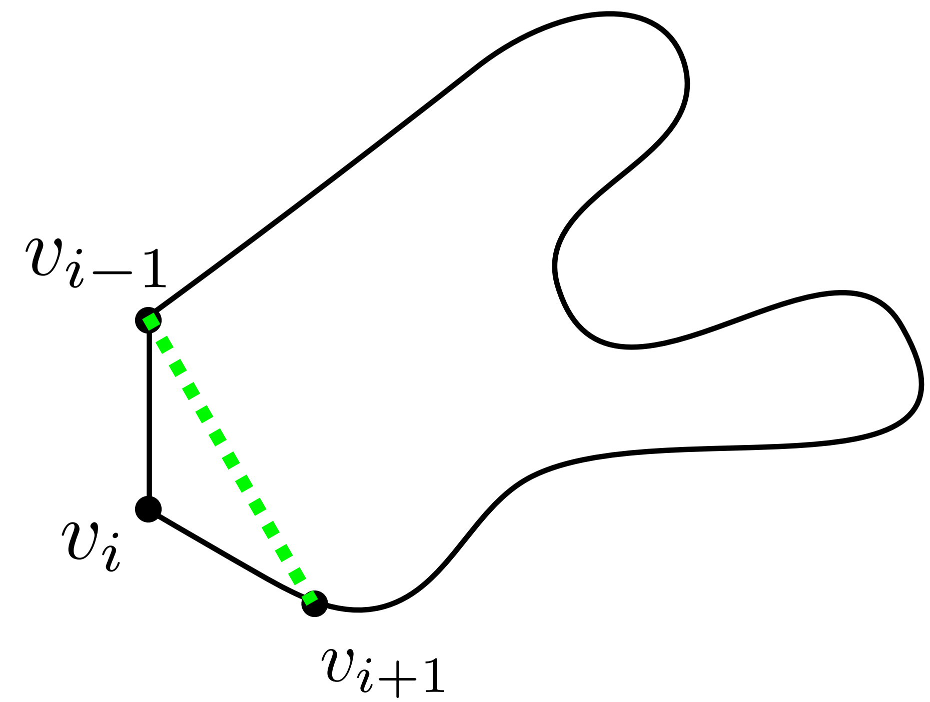
Lemma 5.5
Every polygon has a diagonal.

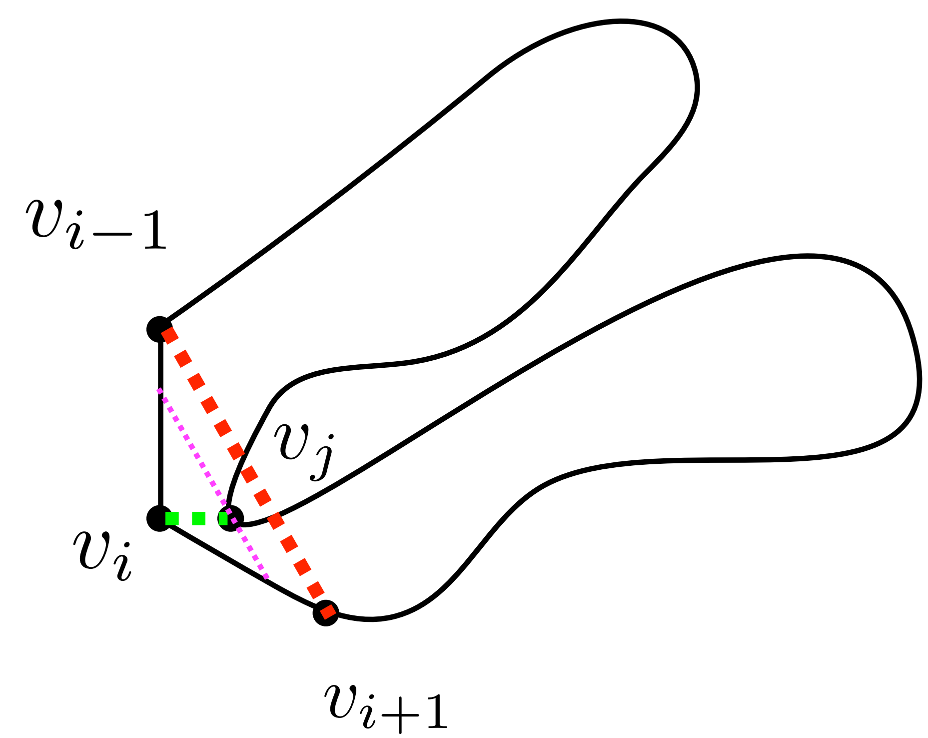
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.
The connection to v_i is a diagonal.





Lemma 5.5

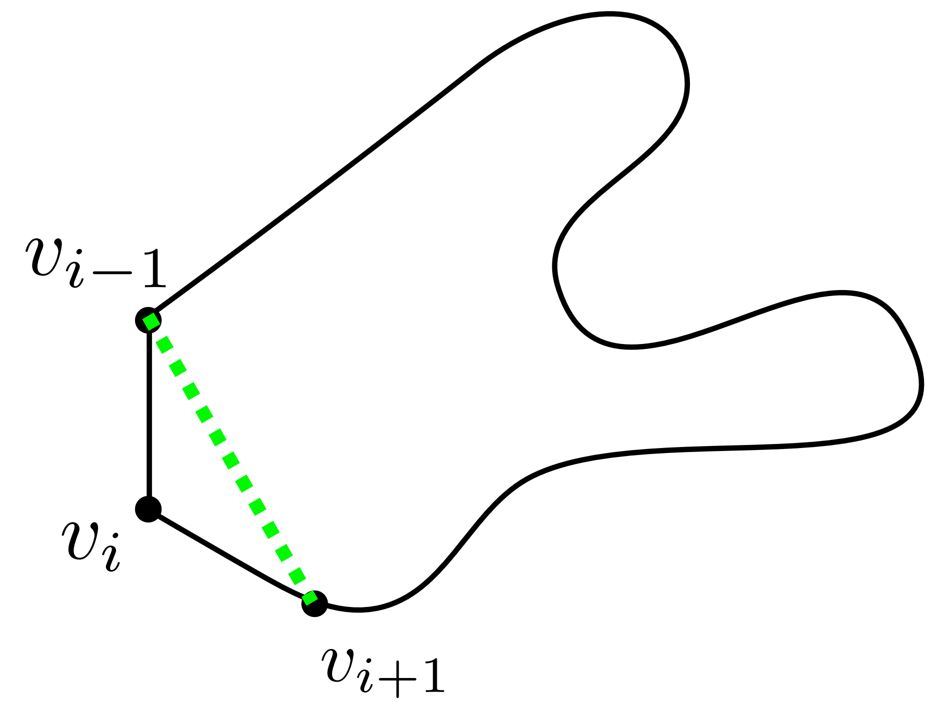
Every polygon has a diagonal.

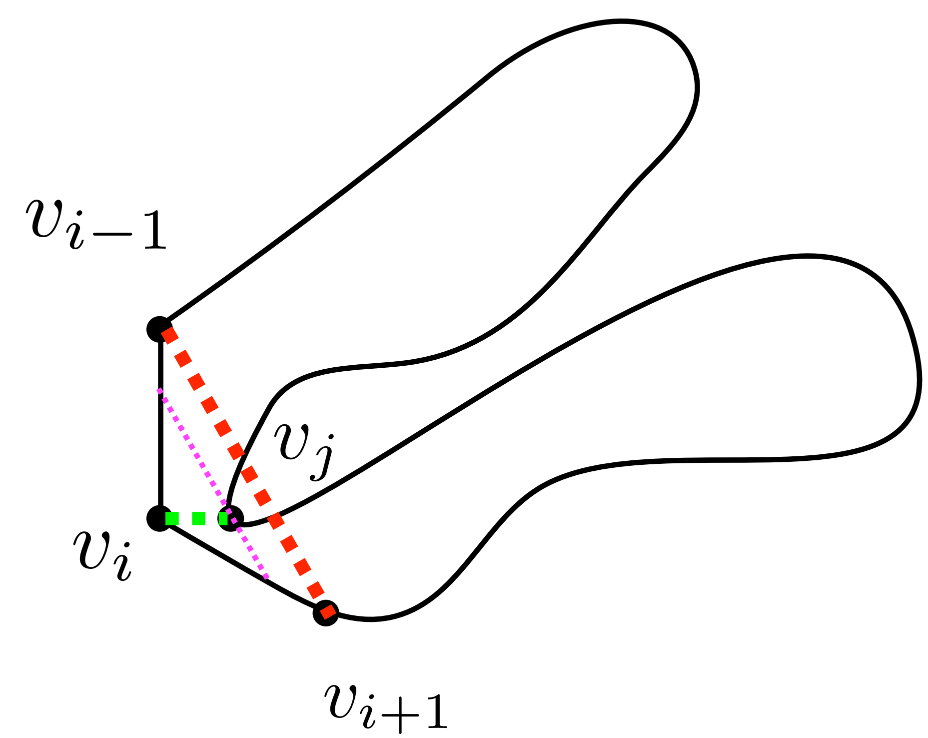
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.
The connection to v_i is a diagonal.





Theorem 5.6

Technische Universität Braunschweig

Lemma 5.5

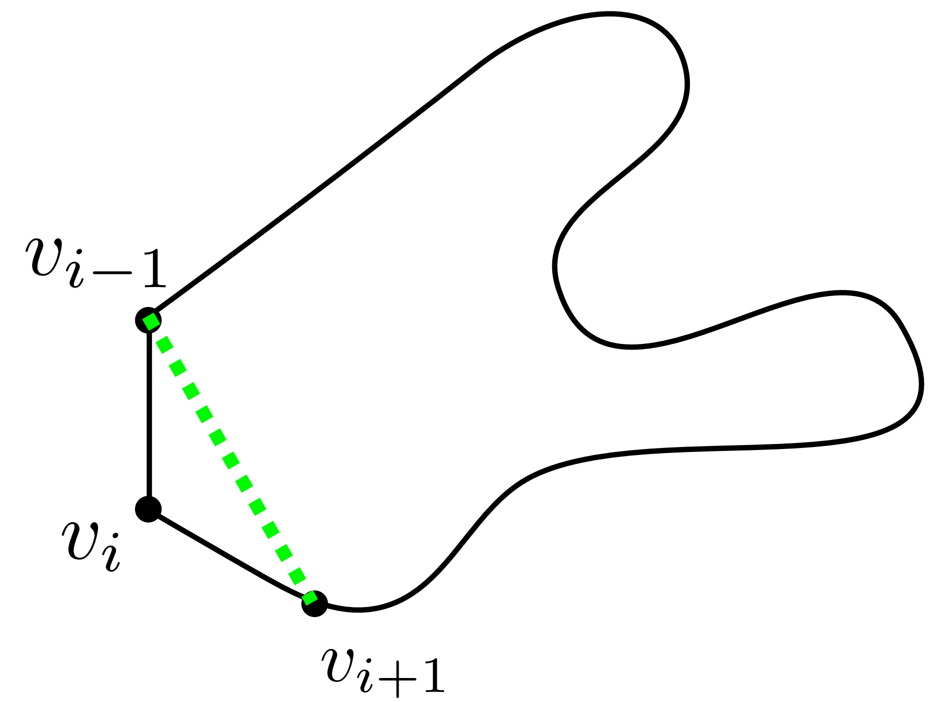
Every polygon has a diagonal.

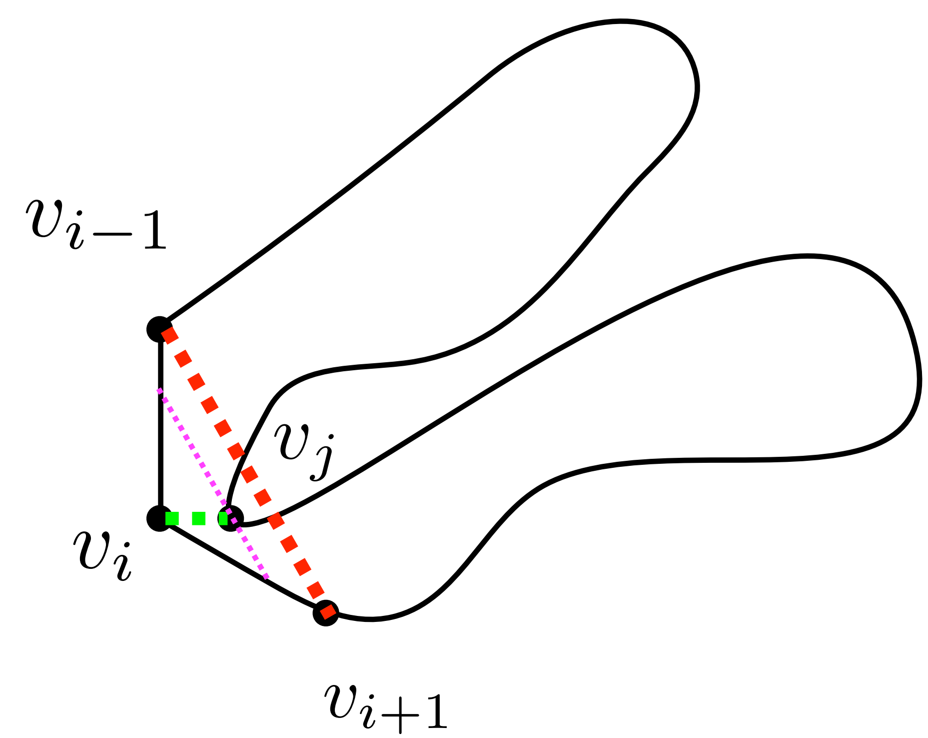
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.
The connection to v_i is a diagonal.





Theorem 5.6

Every polygon can be triangulated.

Lemma 5.5

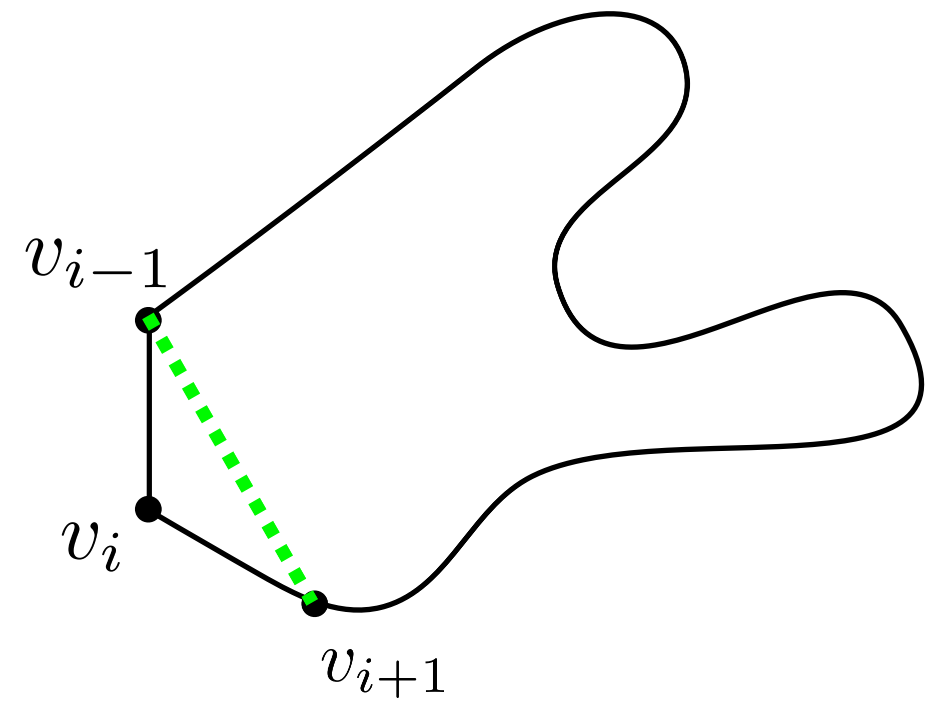
Every polygon has a diagonal.

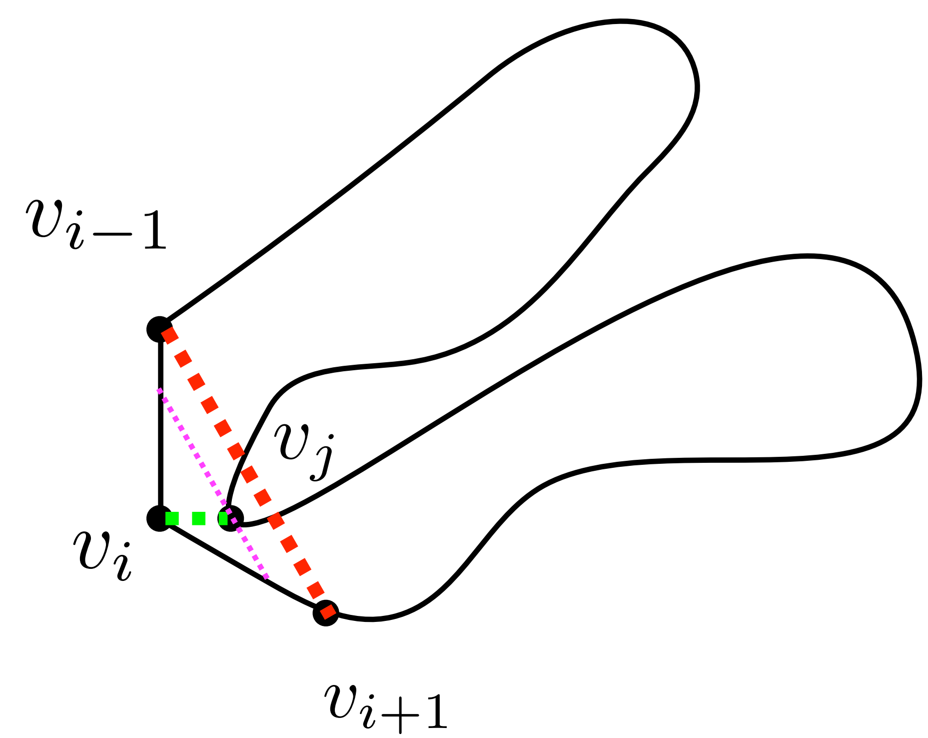
Proof:

Consider the connection between predecessor and successor of v_i.

If this is a diagonal, we are done.

Otherwise, there is at least another vertex v_j inside the triangle v_{i-1}, v_i, v_{i+1}.
Consider the one furthest from the line v_{i-1}, v_{i+1}.
The connection to v_i is a diagonal.





Theorem 5.6

Every polygon can be triangulated.

Proof: Induction.

Theorem 5.7

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

Theorem 5.7

A triangulation of a simple polygon with n vertices has $n\text{-}3$ diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n\text{-}k+1)$-gon.

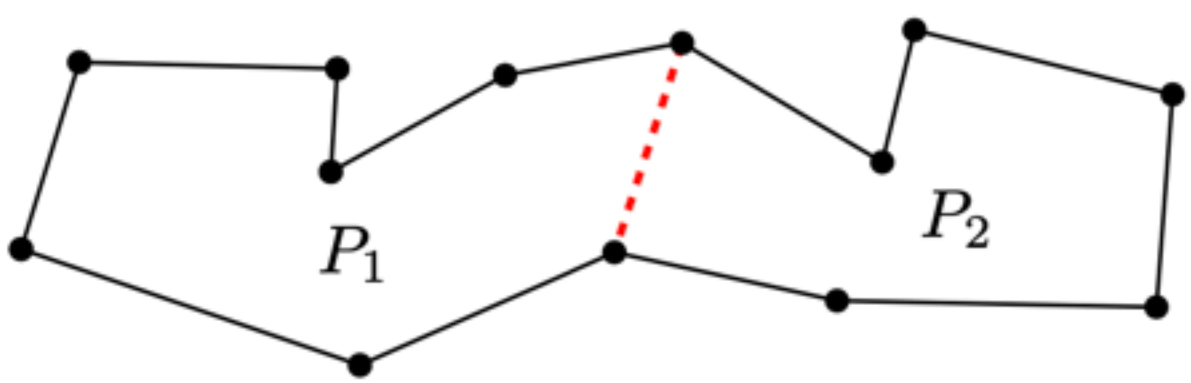
Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n$-$k+1)$-gon.



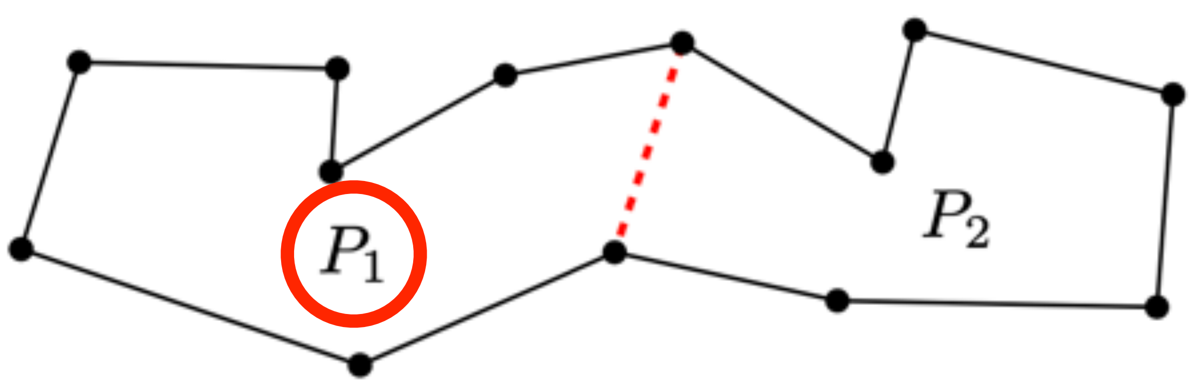
Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n\text{-}k+1)$-gon.



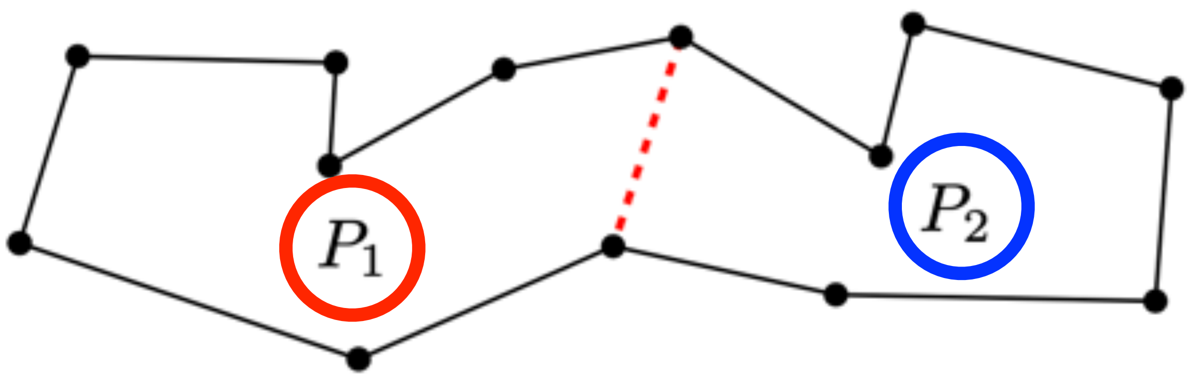
Theorem 5.7

A triangulation of a simple polygon with n vertices has $n-3$ diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.



Theorem 5.7

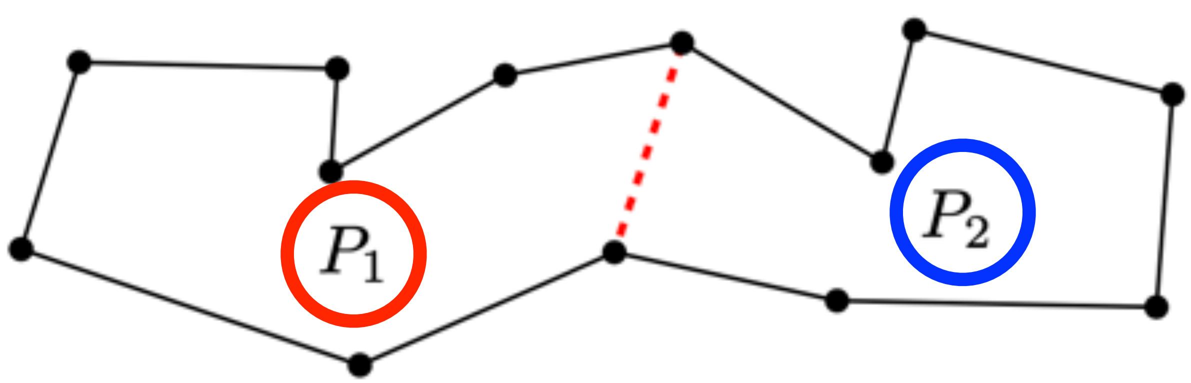
A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have



Theorem 5.7

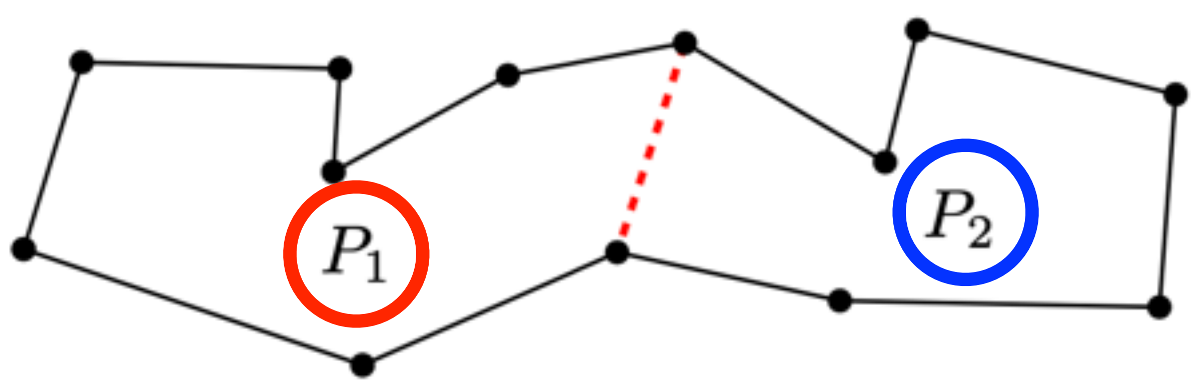
A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have



Technische Universität Braunschweig

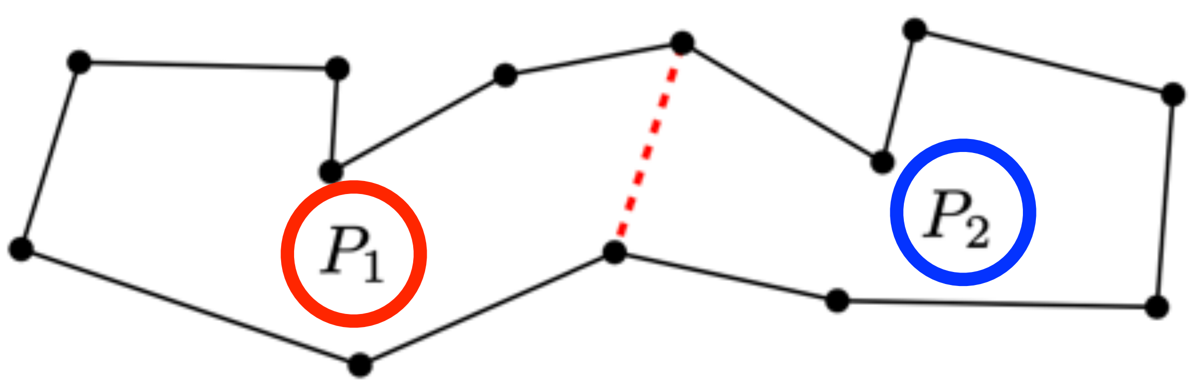
Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.
By assumption, these have

$$d_1 = k + 1 - 3$$



Technische Universität Braunschweig

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

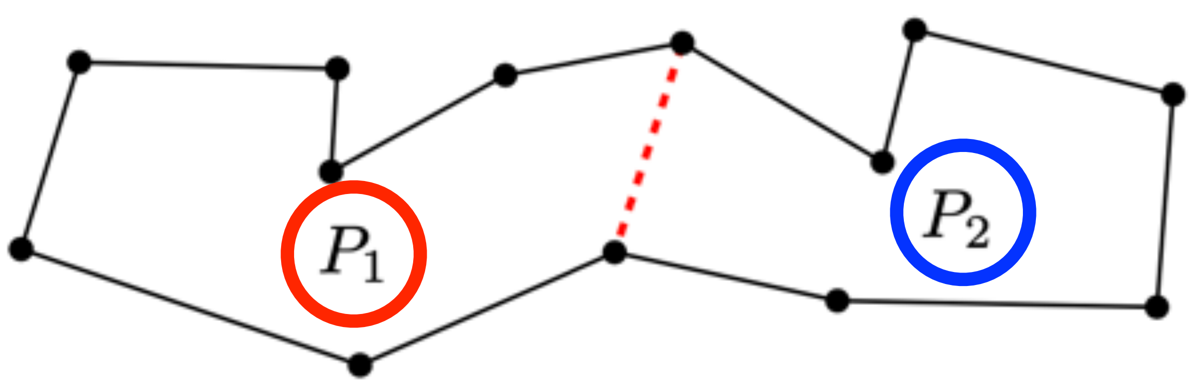
Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have

$$d_1 = k + 1 - 3$$



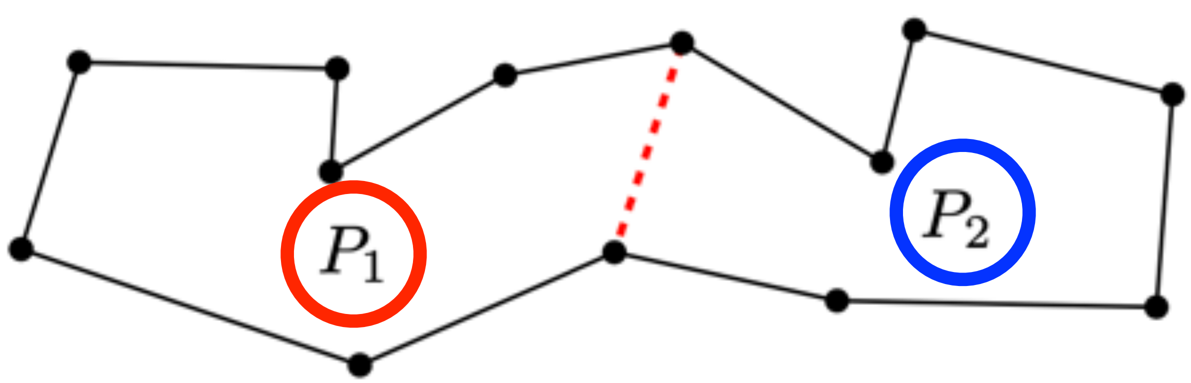
Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.
By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$



Technische Universität Braunschweig

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

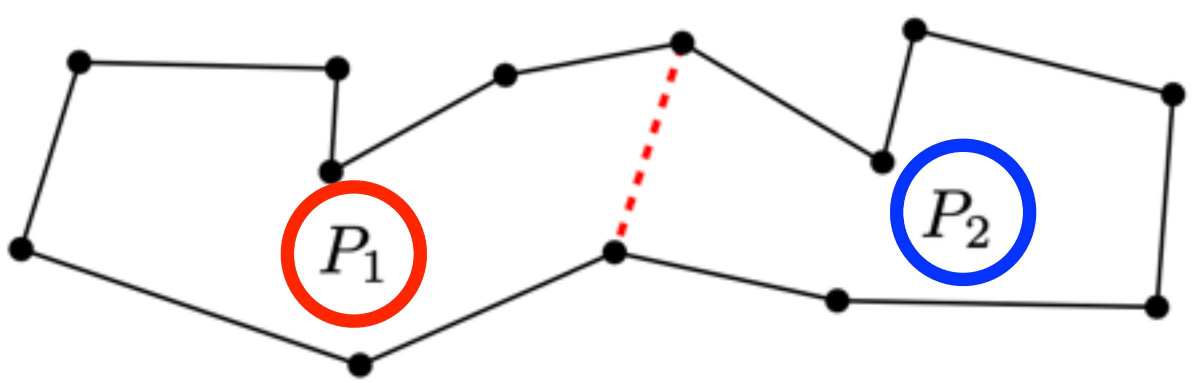
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have

$$d_1 = k + 1 - 3$$

$$d_2 = n - k + 1 - 3$$

diagonals. Now



Theorem 5.7

A triangulation of a simple polygon with n vertices has $n\text{-}3$ diagonals.

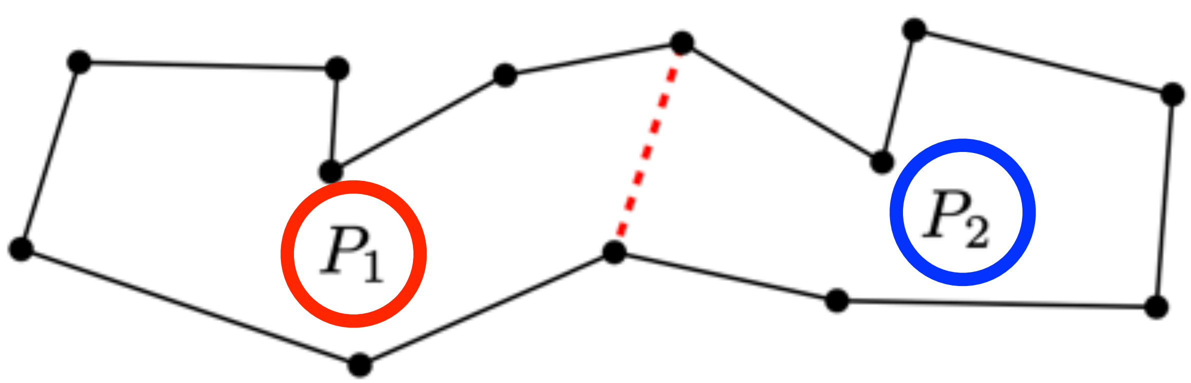
Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n\text{-}k+1)$-gon.
By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$

diagonals. Now

$$d = d_1 + d_2 + 1$$



Technische Universität Braunschweig

Theorem 5.7

A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.

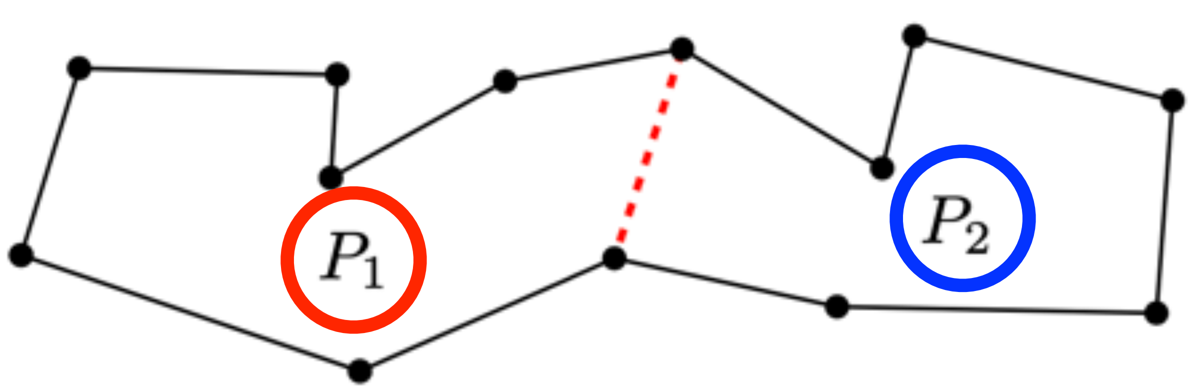
For larger n, consider a diagonal, subdividing P into a ($k+1$)-gon and an (n-$k+1$)-gon.

By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$

diagonals. Now

$$d = d_1 + d_2 + 1 = k + 1 - 3 + n - k + 1 - 3 + 1$$



Theorem 5.7

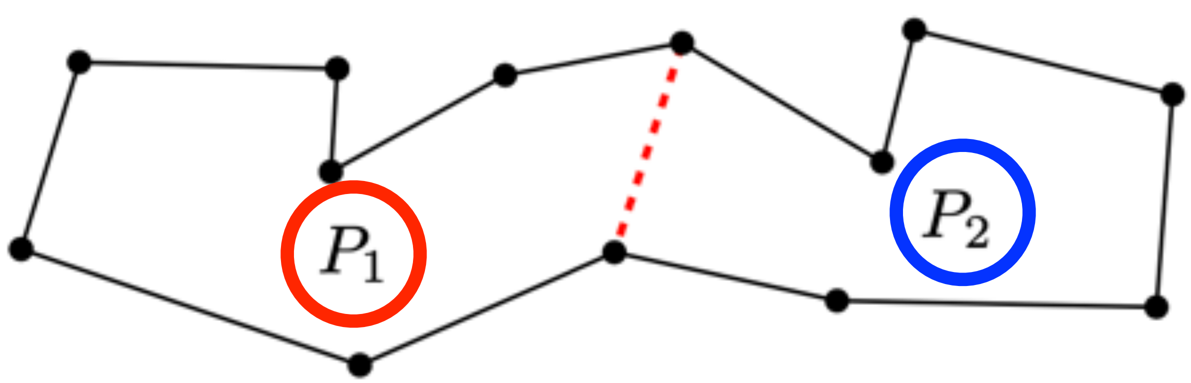
A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n$-$k+1)$-gon.
By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$

diagonals. Now $d = d_1 + d_2 + 1 = k + 1 - 3 + n - k + 1 - 3 + 1$



Theorem 5.7

A triangulation of a simple polygon with n vertices has $n-3$ diagonals.

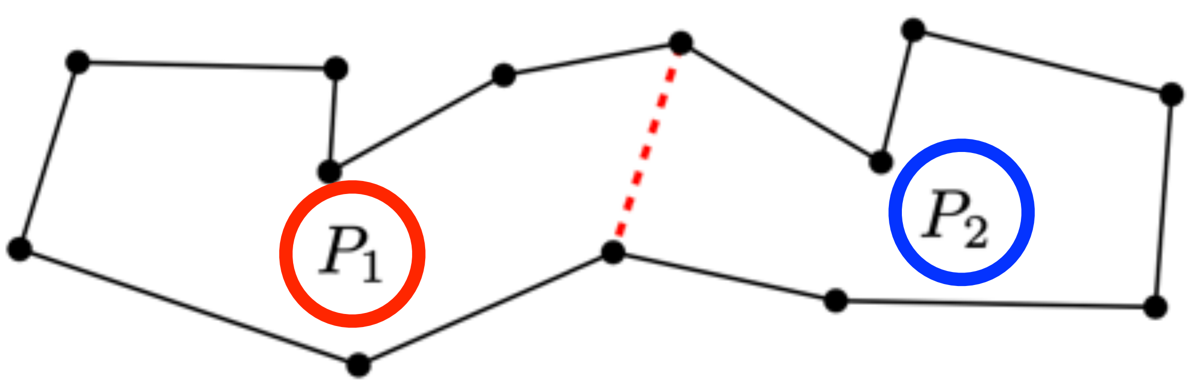
Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon. By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$

diagonals. Now $d = d_1 + d_2 + 1 = k + 1 - 3 + n - k + 1 - 3 + 1$



Theorem 5.7

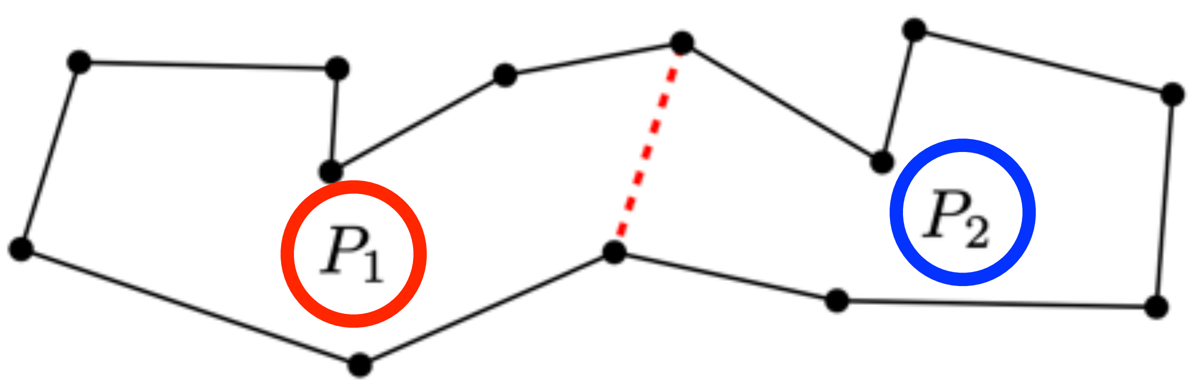
A triangulation of a simple polygon with n vertices has n-3 diagonals.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n\text{-}k+1)$-gon.
By assumption, these have

$$d_1 = k + 1 - 3$$
$$d_2 = n - k + 1 - 3$$

diagonals. Now $d = d_1 + d_2 + 1 = k + 1 - 3 + n - k + 1 - 3 + 1 = n - 3.$



Technische Universität Braunschweig

Theorem 5.8

Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

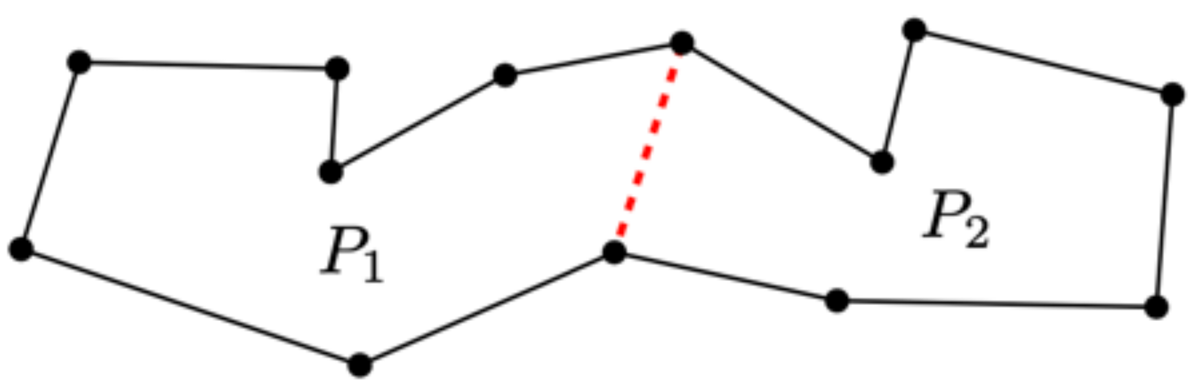
Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.



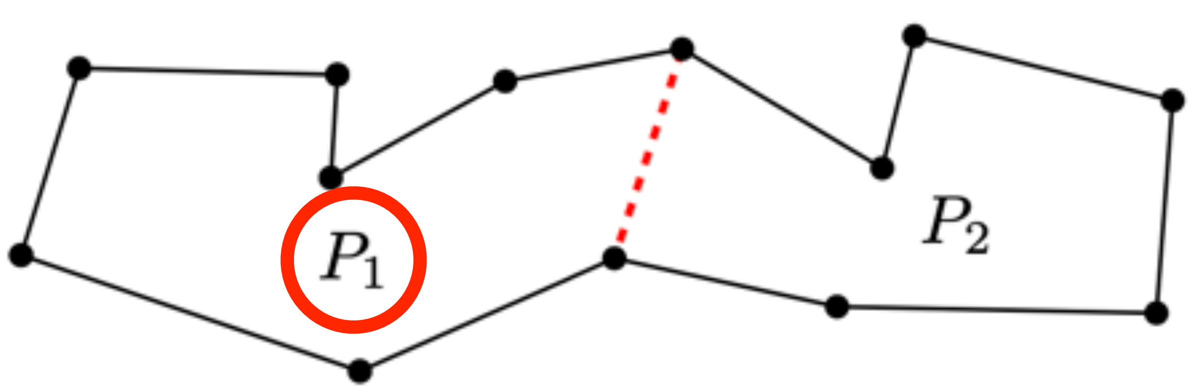
Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.



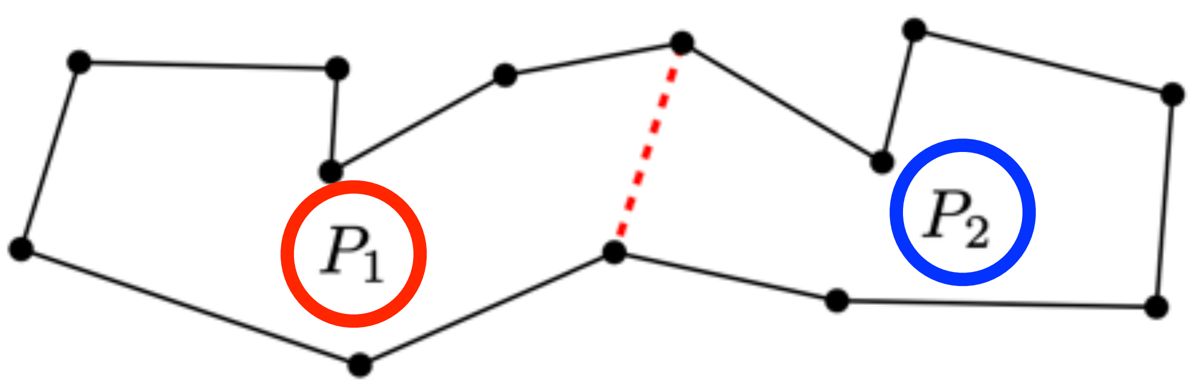
Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

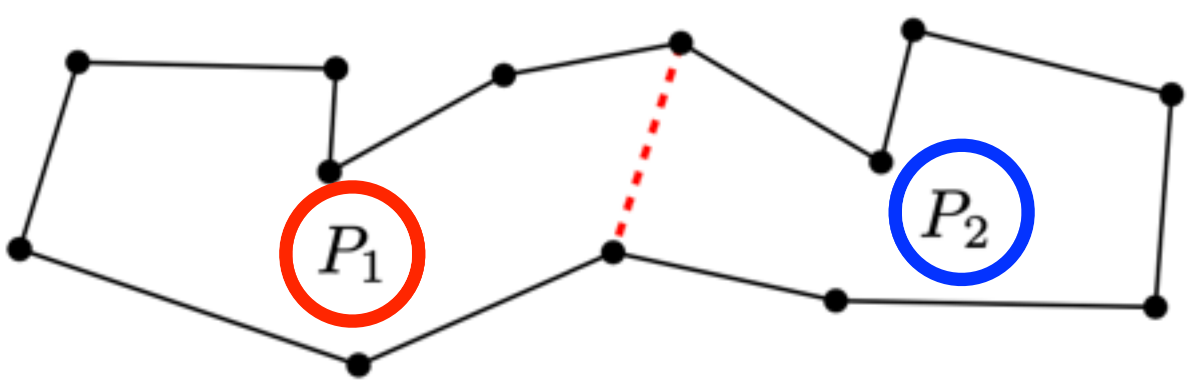


Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n$-$k+1)$-gon.
By assumption, these have



Theorem 5.8

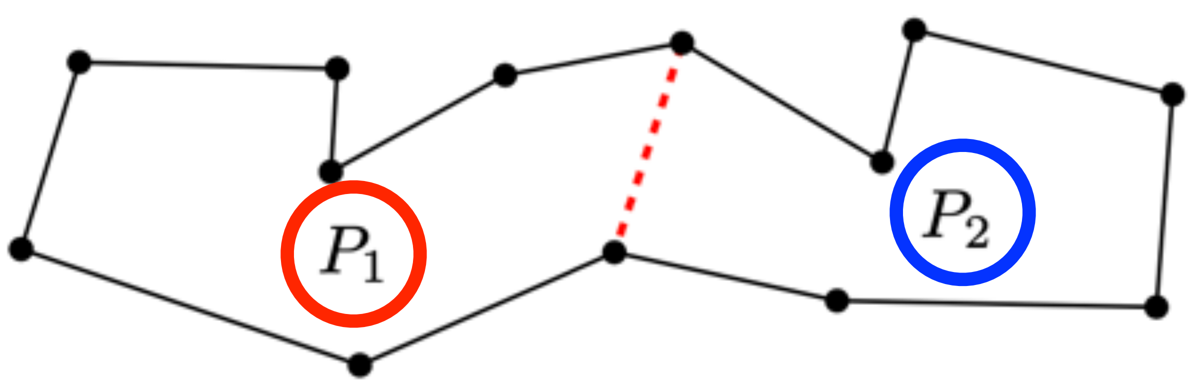
A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n$-$k+1)$-gon.

By assumption, these have



Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

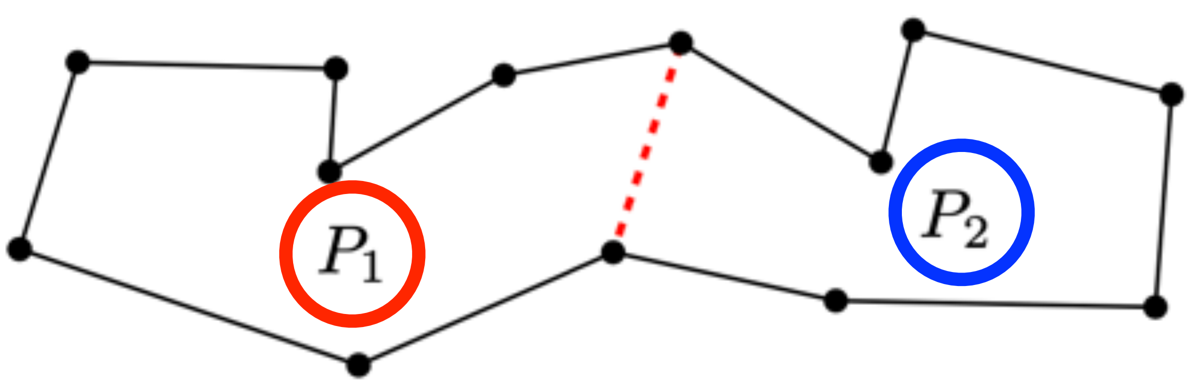
Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have

$$t_1 = k + 1 - 2$$



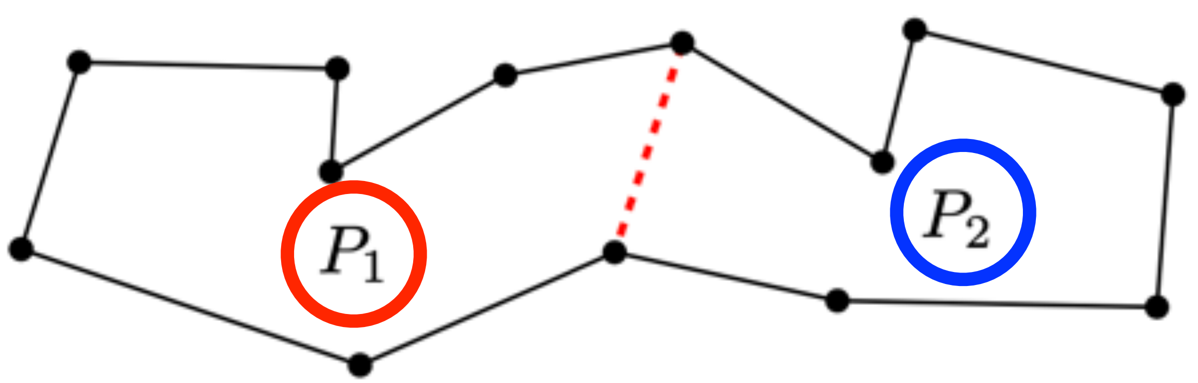
Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.
By assumption, these have

$$t_1 = k + 1 - 2$$



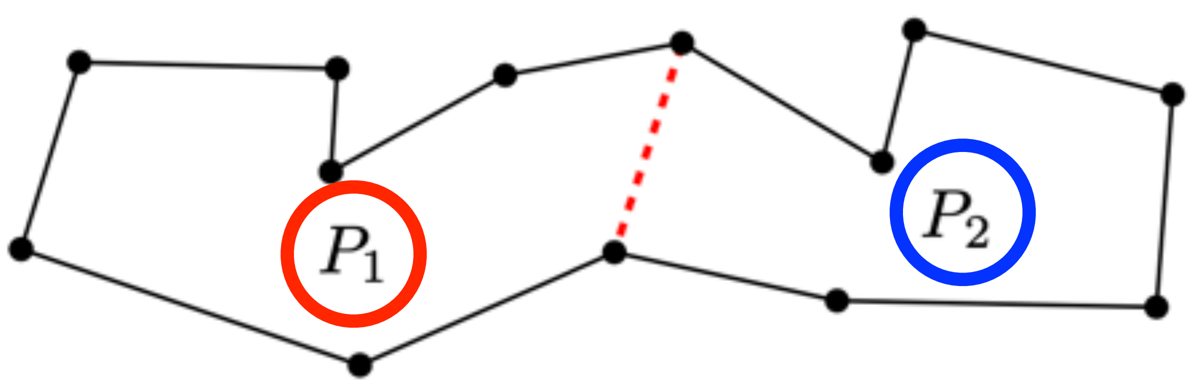
Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.
By assumption, these have

$$t_1 = k + 1 - 2$$
$$t_2 = n - k + 1 - 2$$



Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

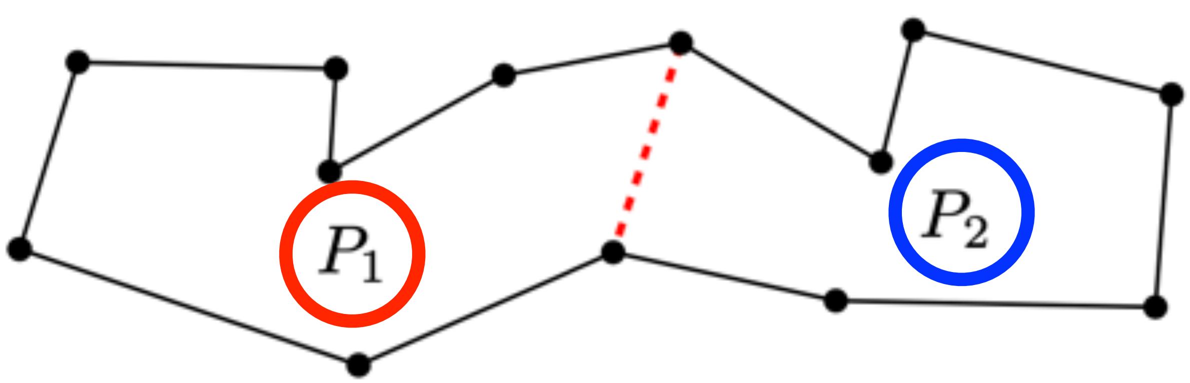
Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n$-$k+1)$-gon. By assumption, these have

$$t_1 = k + 1 - 2$$
$$t_2 = n - k + 1 - 2$$

diagonals. Now



Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n\text{-}k+1)$-gon.

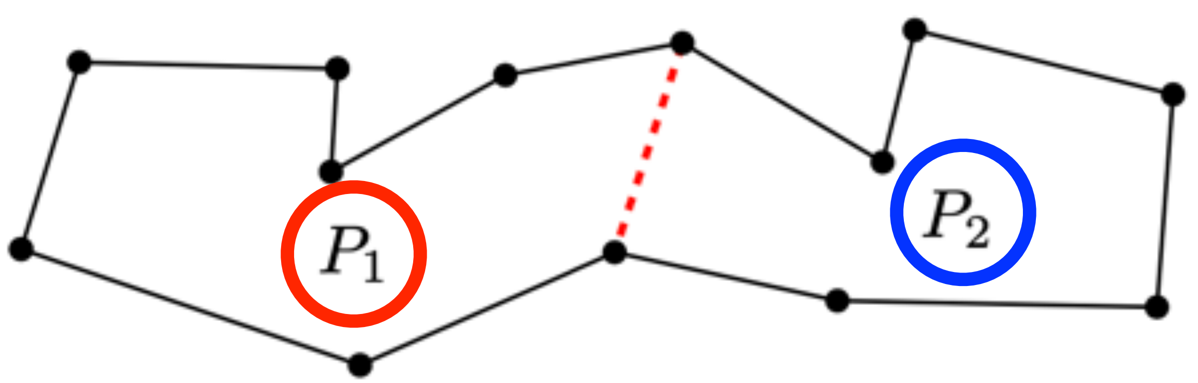
By assumption, these have

$$t_1 = k + 1 - 2$$

$$t_2 = n - k + 1 - 2$$

diagonals. Now

$$t = t_1 + t_2$$



Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

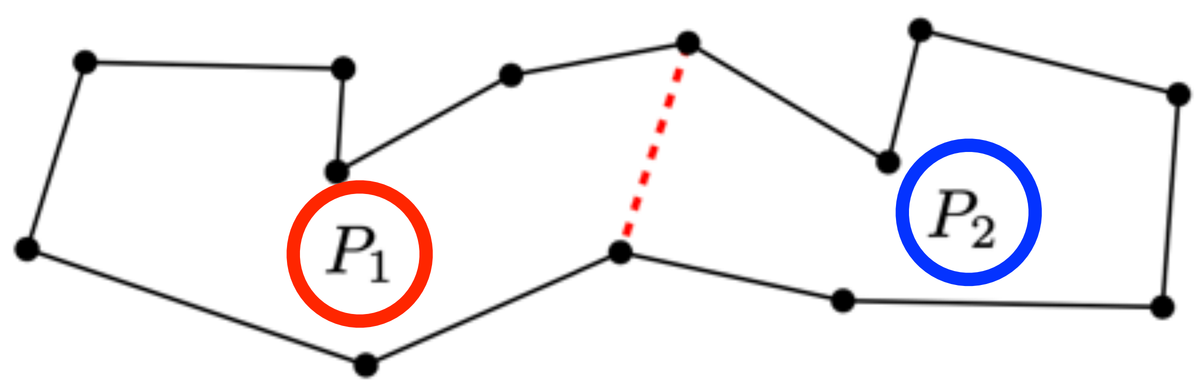
By assumption, these have

$$t_1 = k + 1 - 2$$

$$t_2 = n - k + 1 - 2$$

diagonals. Now

$$t = t_1 + t_2 = k + 1 - 2 + n - k + 1 - 2$$



Theorem 5.8

A triangulation of a simple polygon with n vertices has n-2 triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

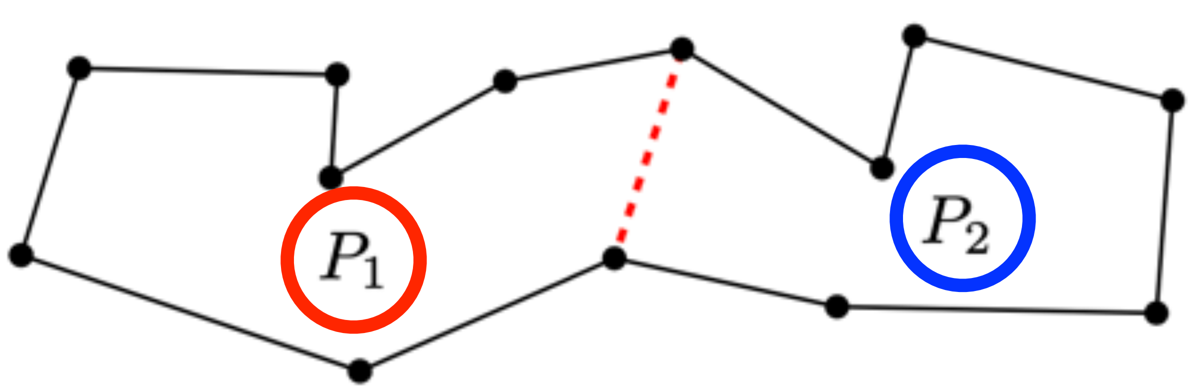
By assumption, these have

$$t_1 = k + 1 - 2$$

$$t_2 = n - k + 1 - 2$$

diagonals. Now

$$t = t_1 + t_2 = k + 1 - 2 + n - k + 1 - 2$$



Technische Universität Braunschweig

Theorem 5.8

A triangulation of a simple polygon with n vertices has $n-2$ triangles.

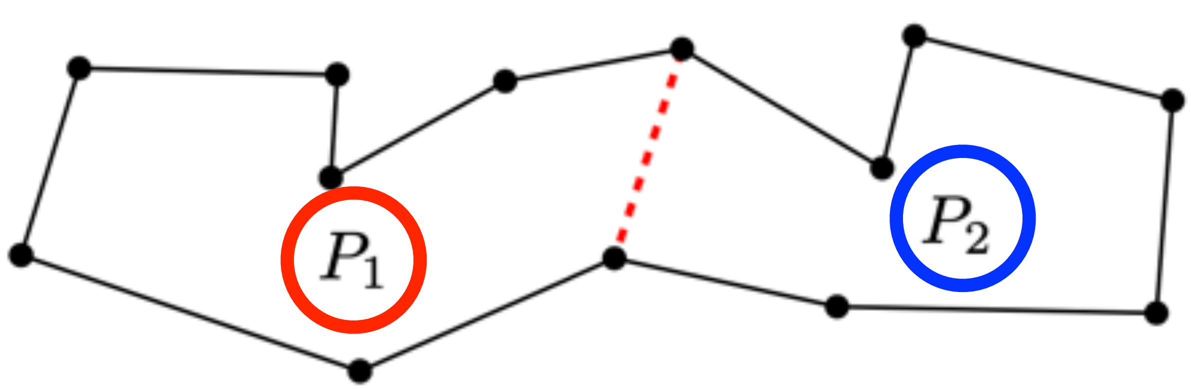
Proof:

Induction over n: The claim is clear for $n=3$.
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.
By assumption, these have

$$t_1 = k+1-2$$
$$t_2 = n-k+1-2$$

diagonals. Now

$$t = t_1 + t_2 = k+1-2 + n-k+1-2$$



Theorem 5.8

A triangulation of a simple polygon with n vertices has $n\text{-}2$ triangles.

Proof:

Induction over n: The claim is clear for $n=3$.

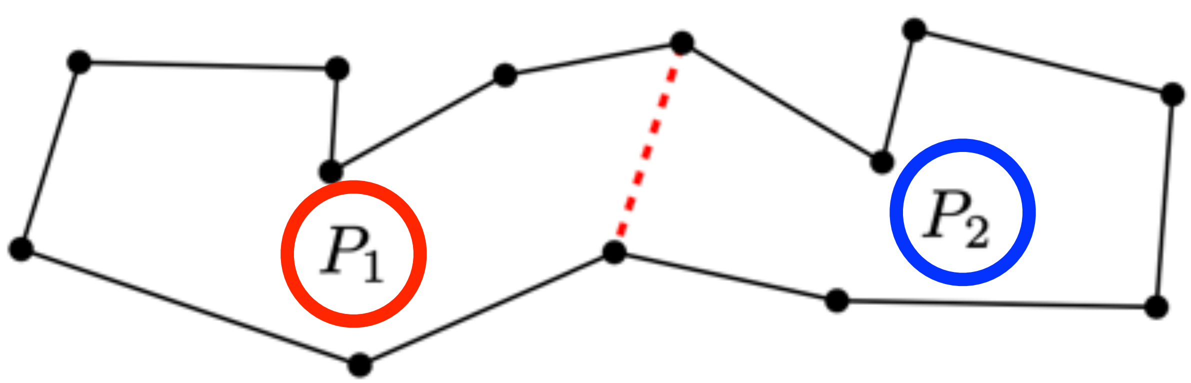
For larger n, consider a diagonal, subdividing P into a $(k+1)$-gon and an $(n-k+1)$-gon.

By assumption, these have

$$t_1 = k+1-2$$
$$t_2 = n-k+1-2$$

diagonals. Now

$$t = t_1 + t_2 = k+1-2 + n-k+1-2 = n-2.$$



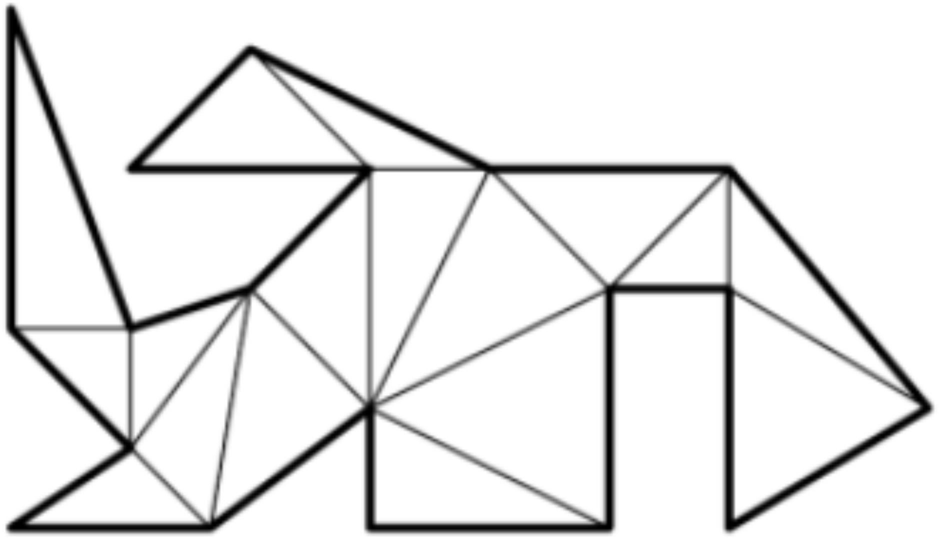
Theorem 5.9

Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

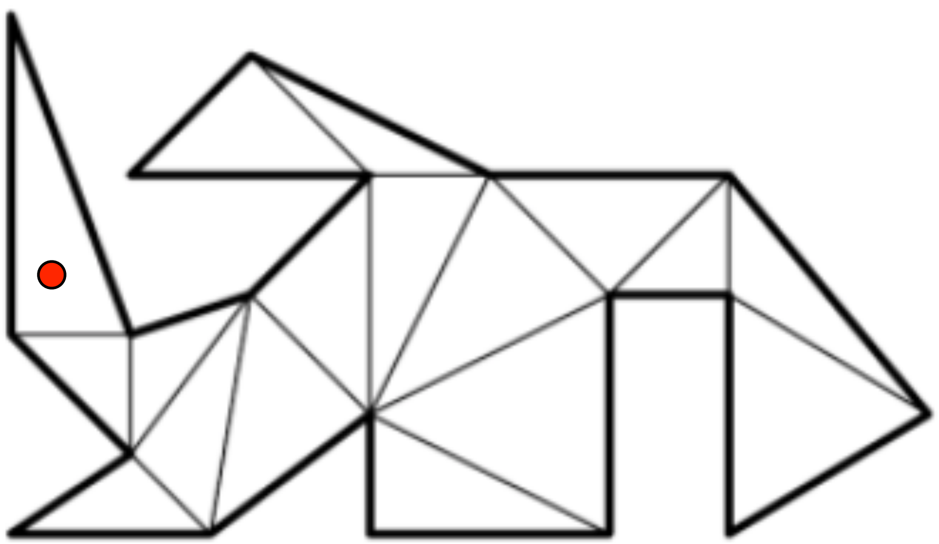
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



Theorem 5.9

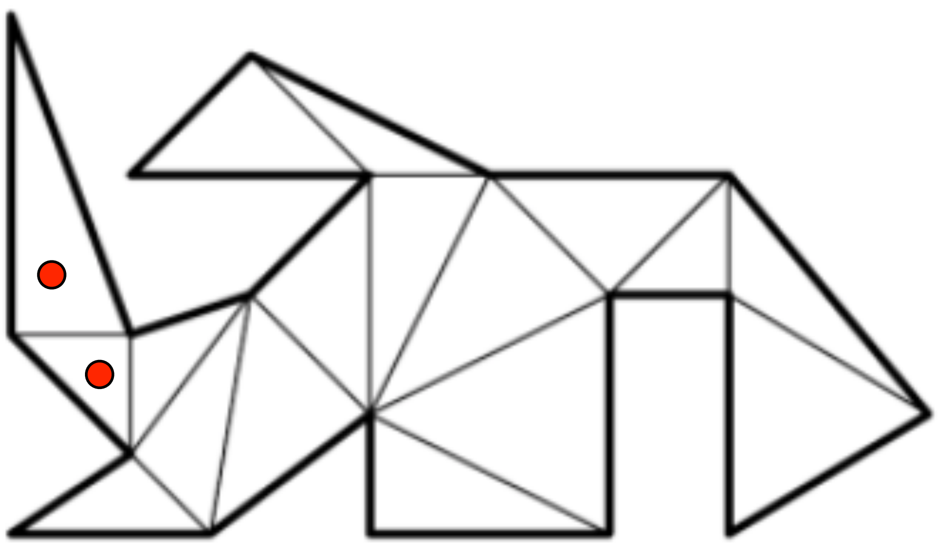
The dual graph of a triangulation of a simple polygon is a tree.



Technische Universität Braunschweig

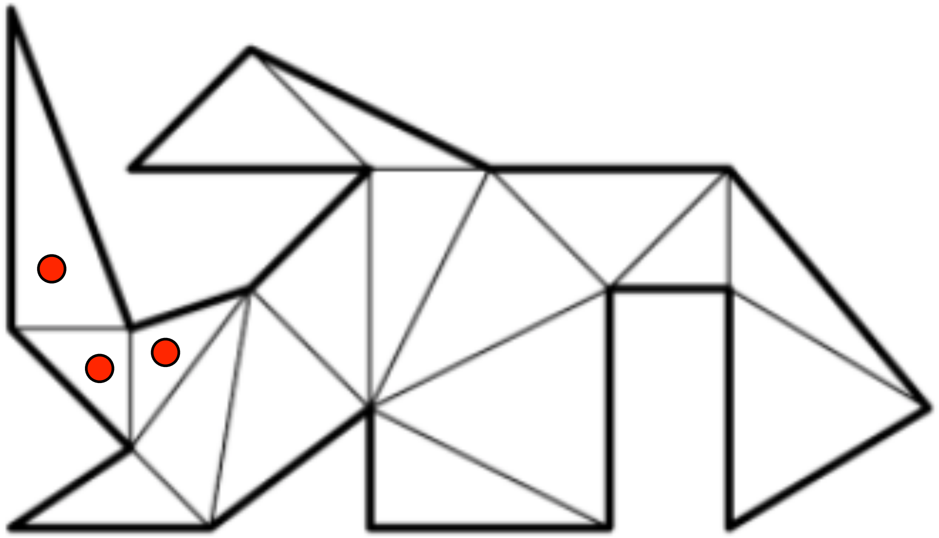
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



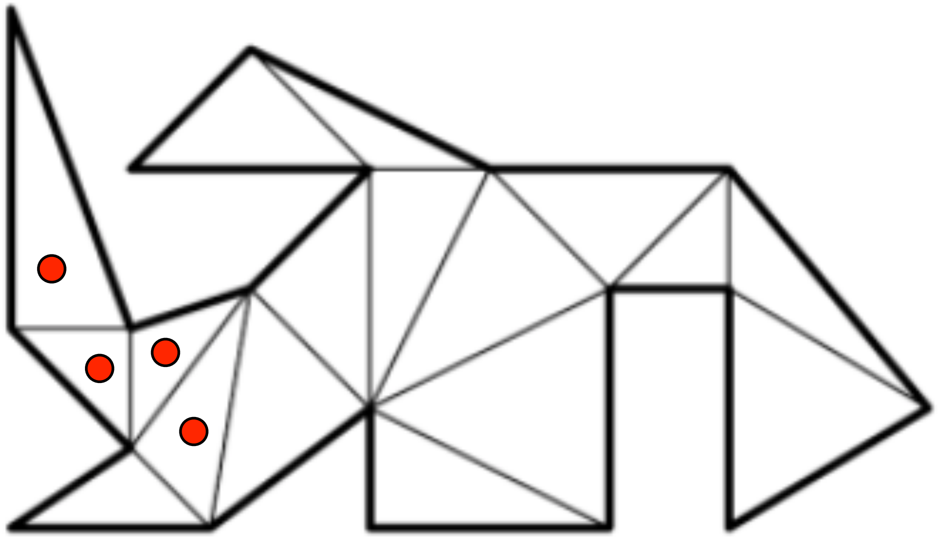
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



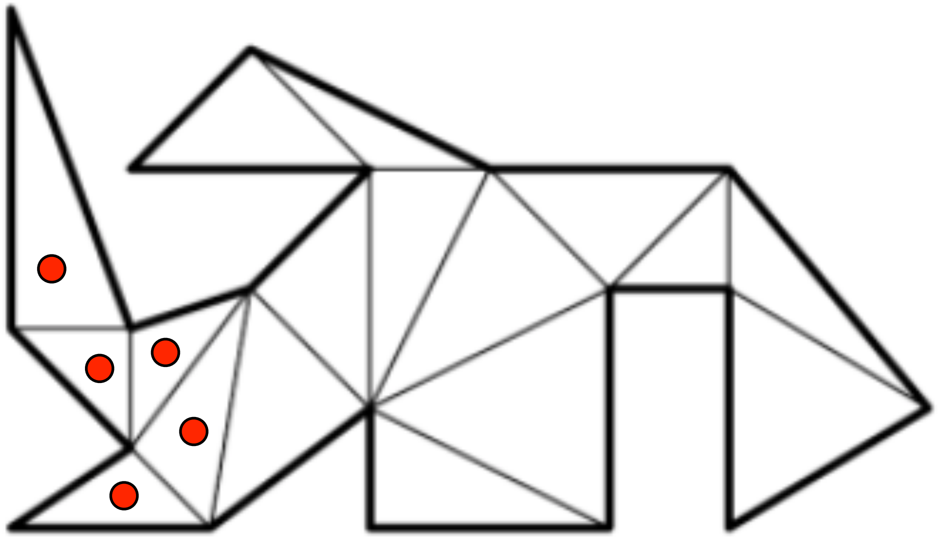
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



Theorem 5.9

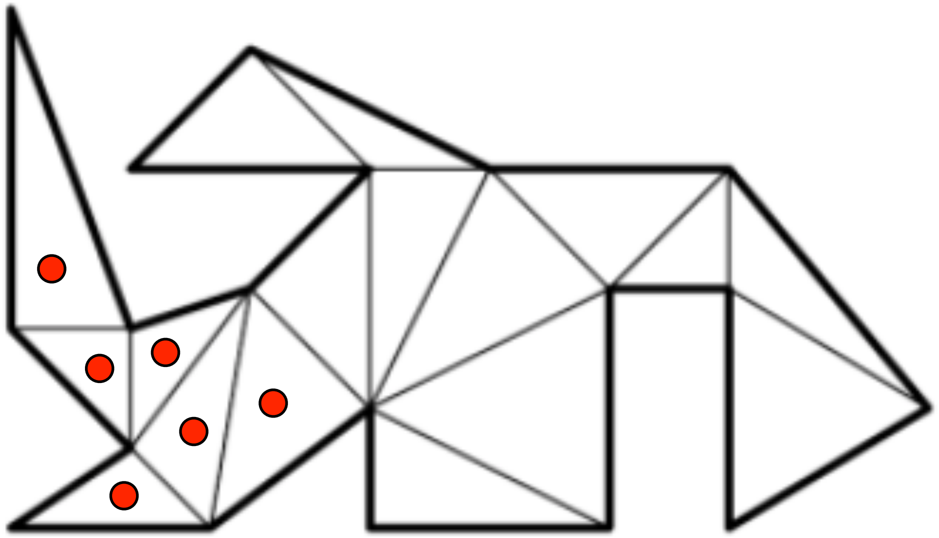
The dual graph of a triangulation of a simple polygon is a tree.



Technische Universität Braunschweig

Theorem 5.9

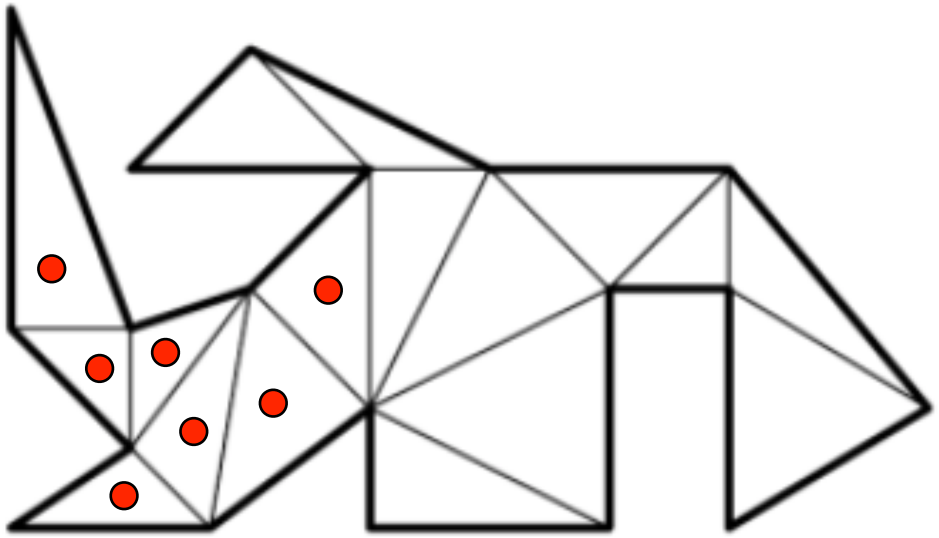
The dual graph of a triangulation of a simple polygon is a tree.



Technische Universität Braunschweig

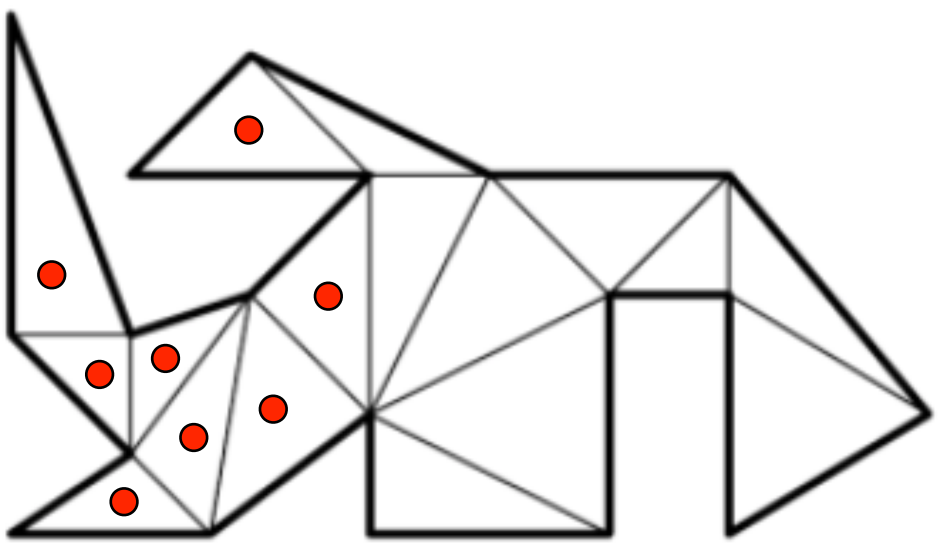
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



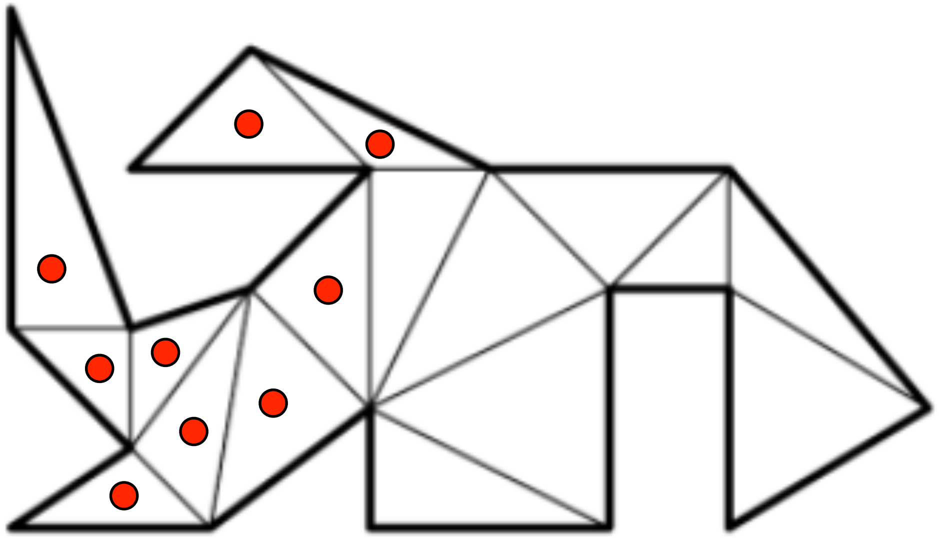
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



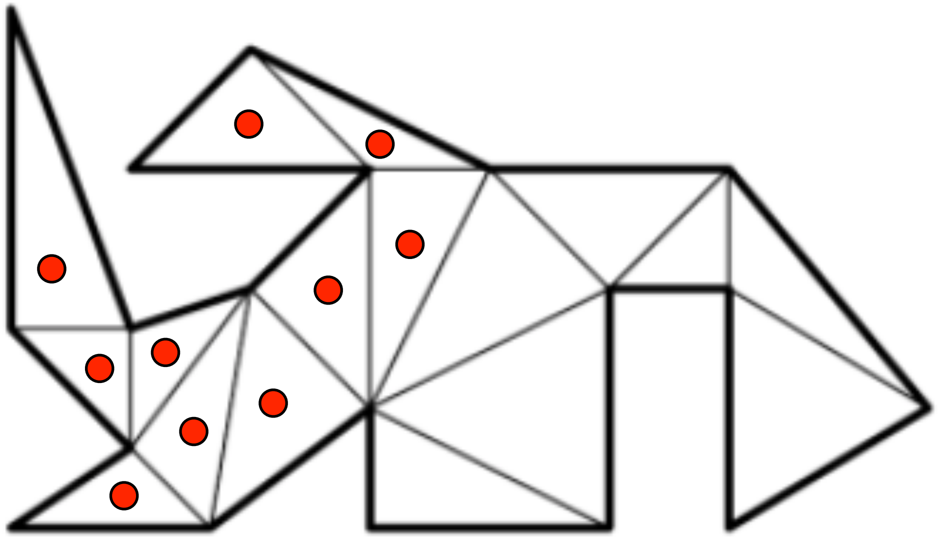
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



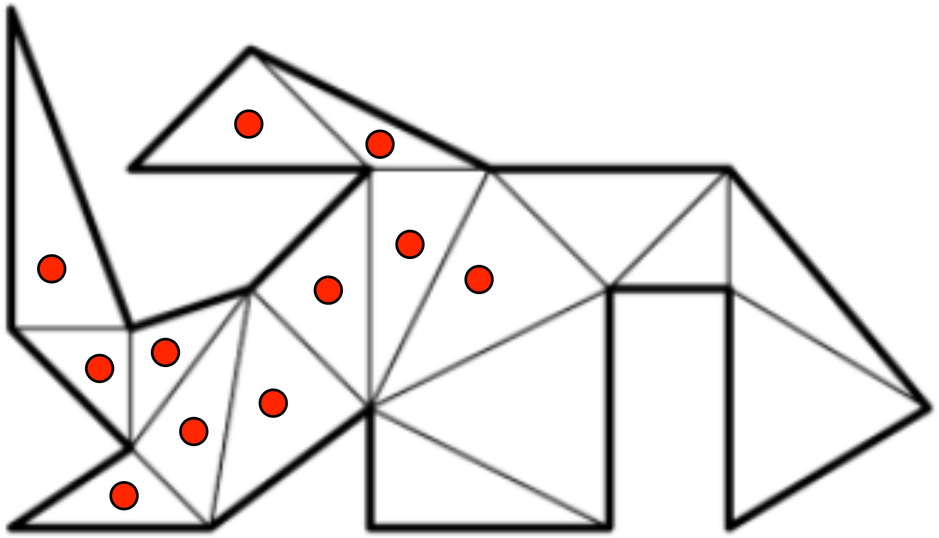
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



Theorem 5.9

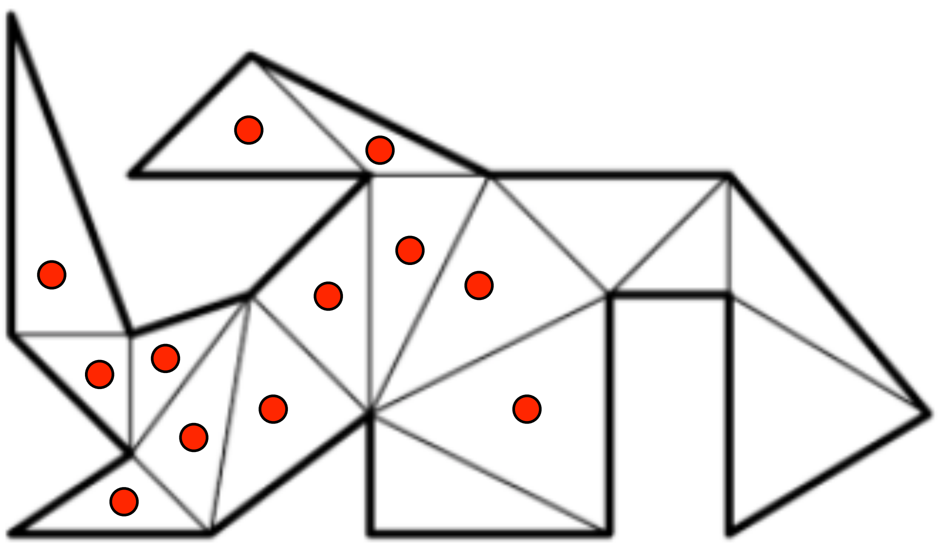
The dual graph of a triangulation of a simple polygon is a tree.



Technische Universität Braunschweig

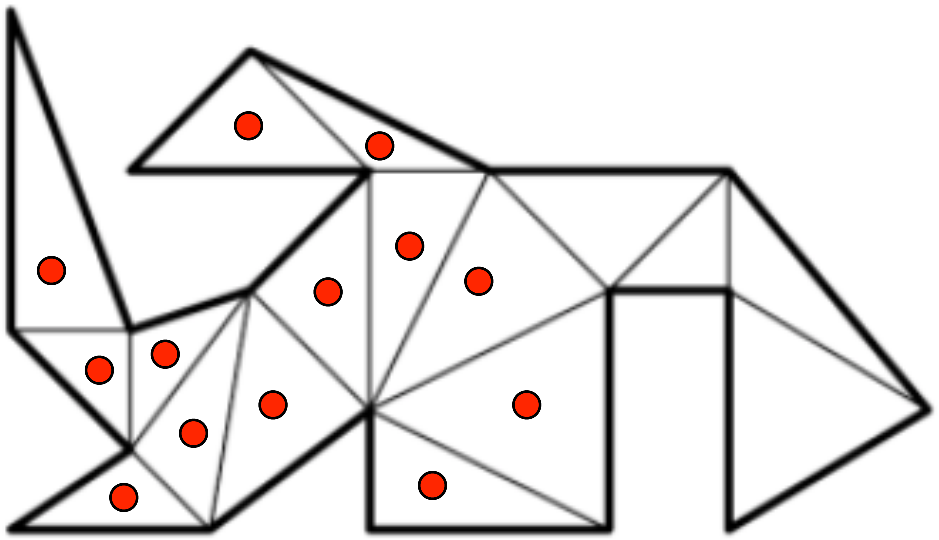
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



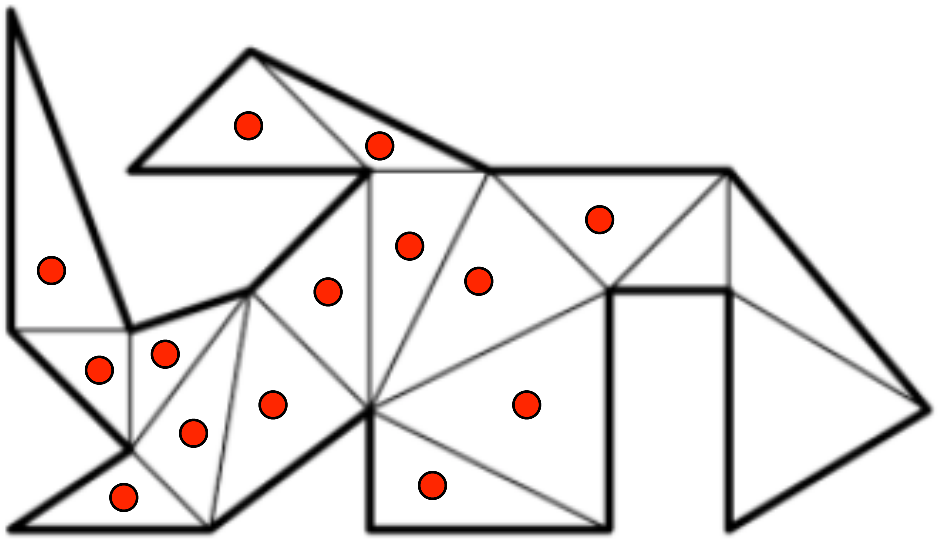
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



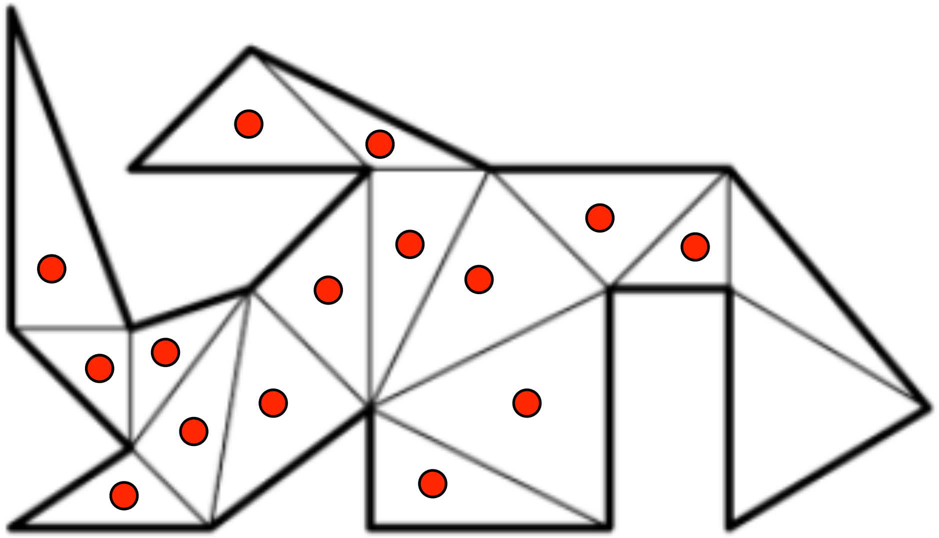
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



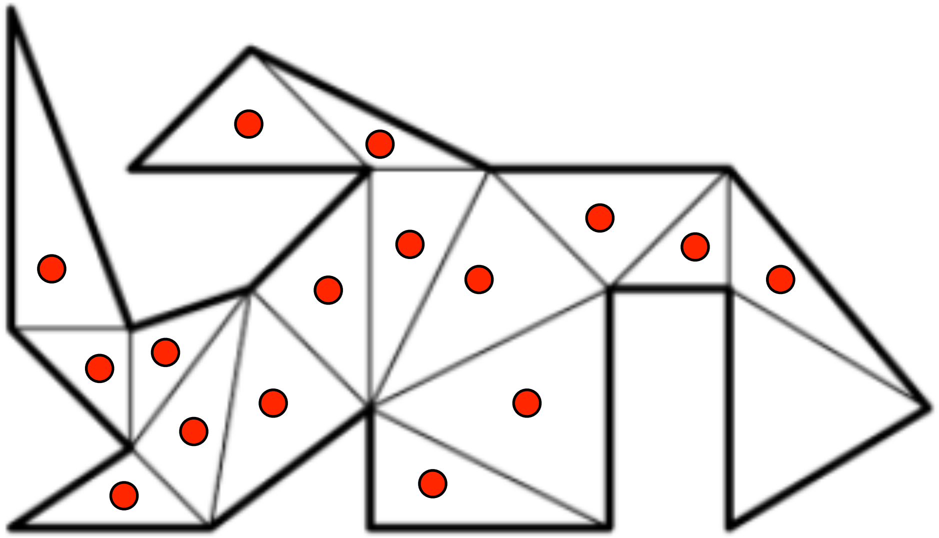
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



Theorem 5.9

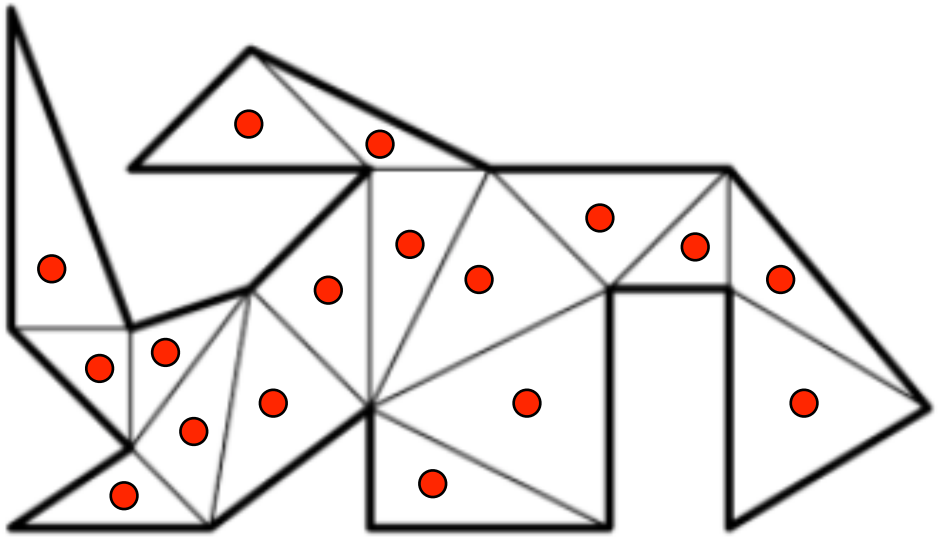
The dual graph of a triangulation of a simple polygon is a tree.



Technische Universität Braunschweig

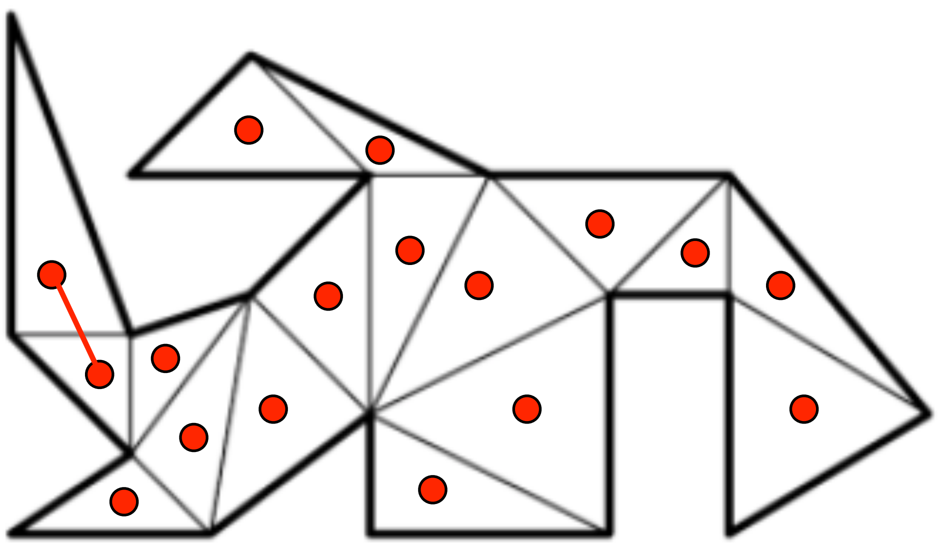
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



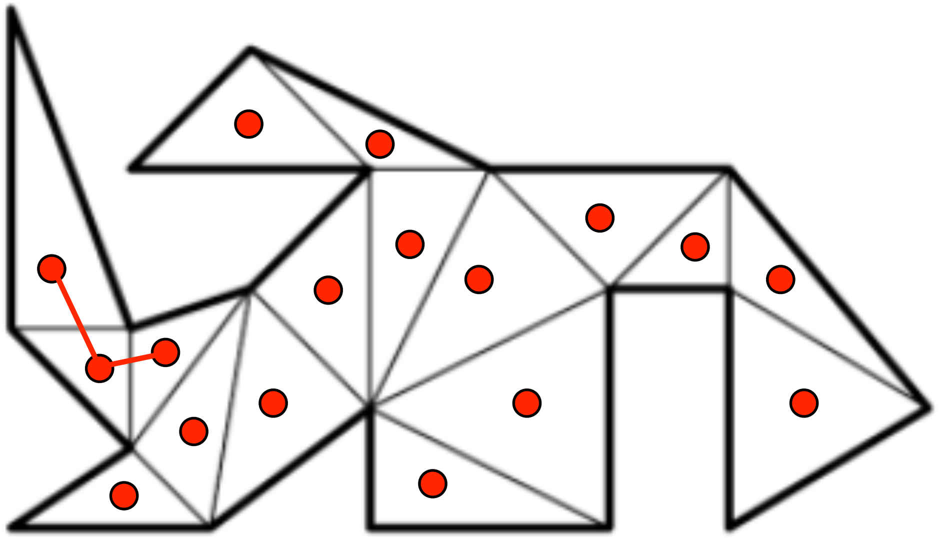
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



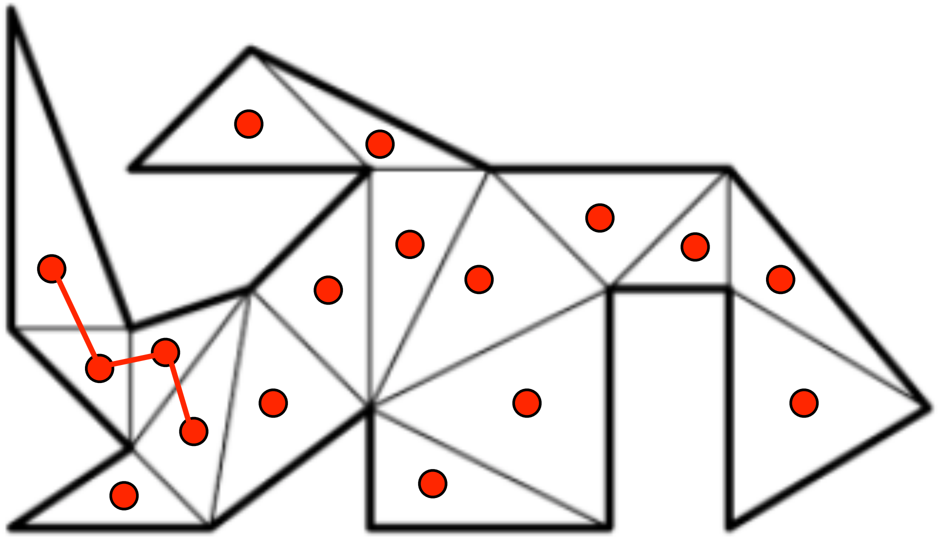
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



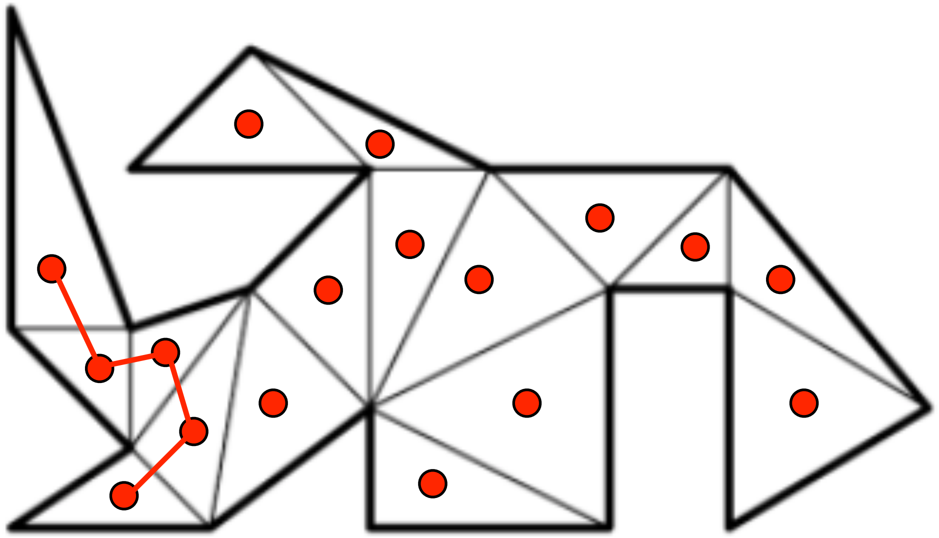
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



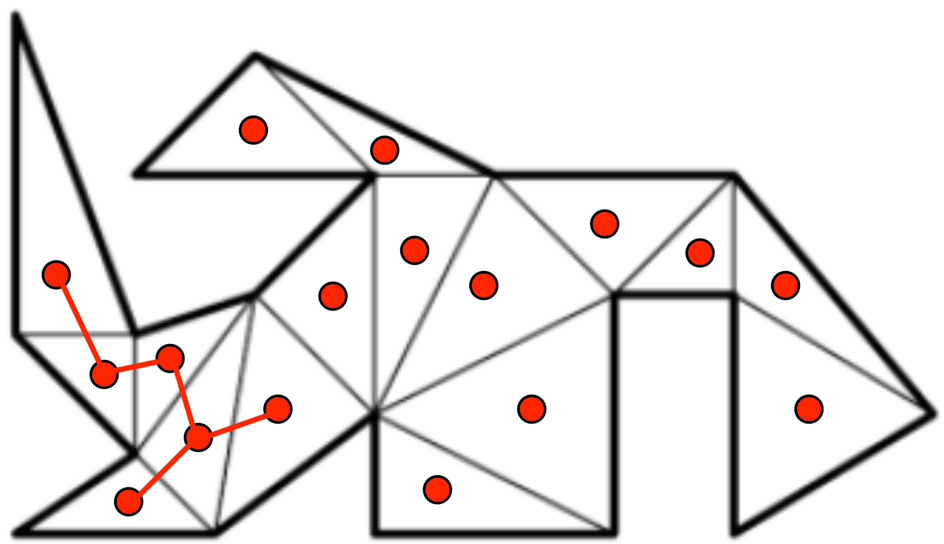
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



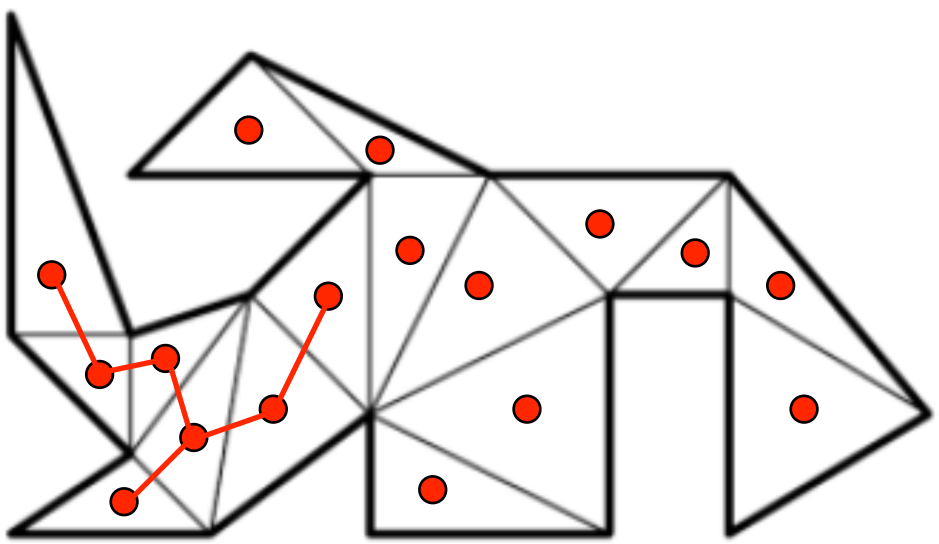
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



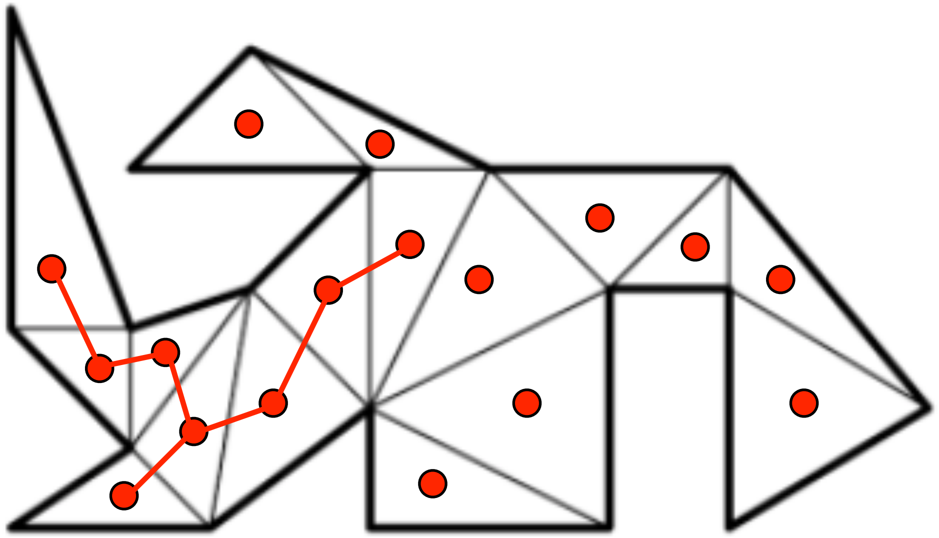
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



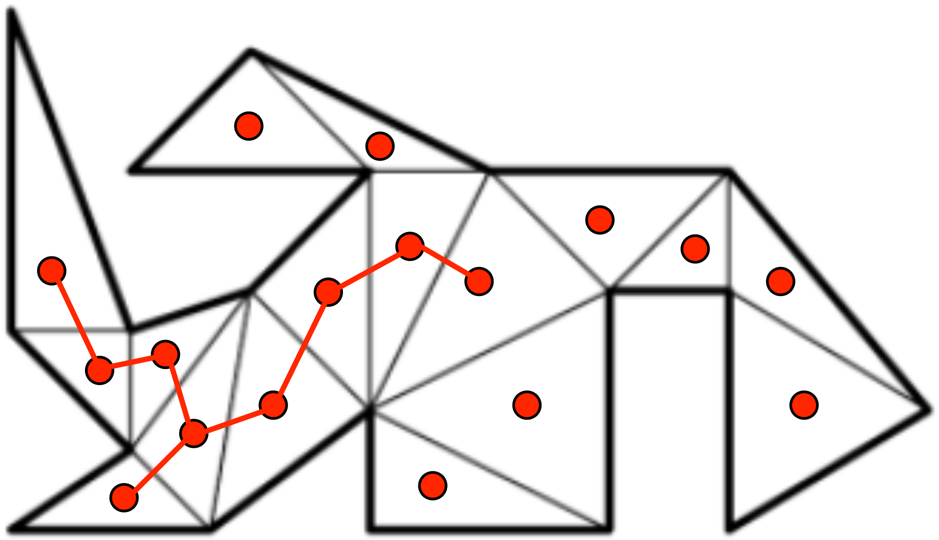
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



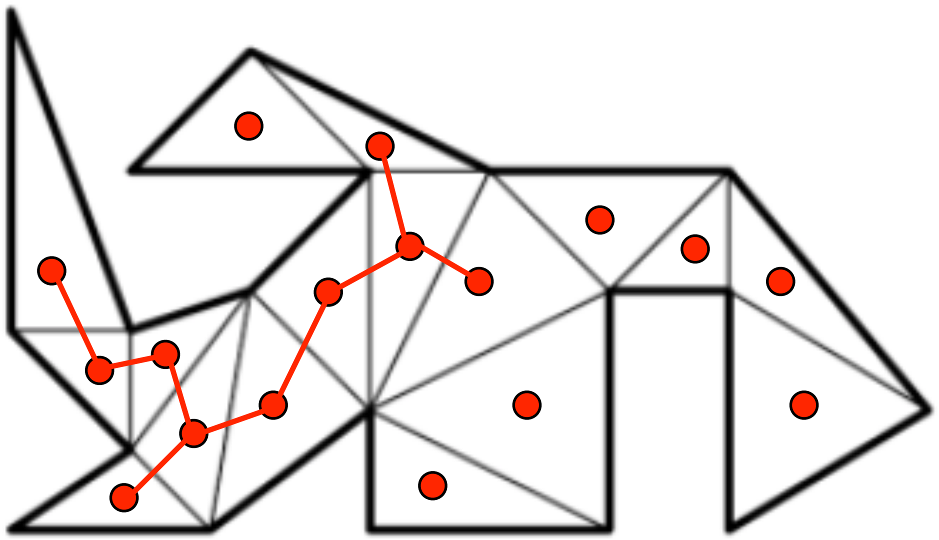
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



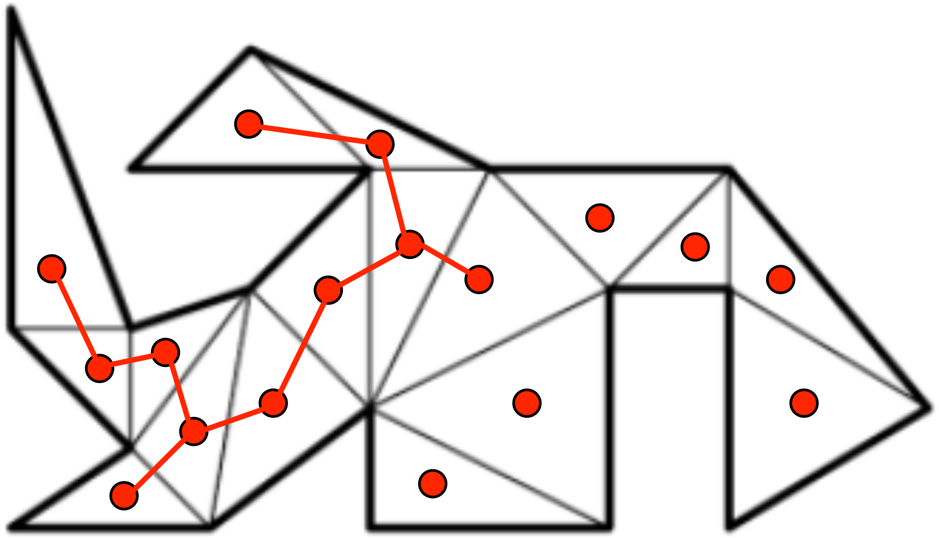
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



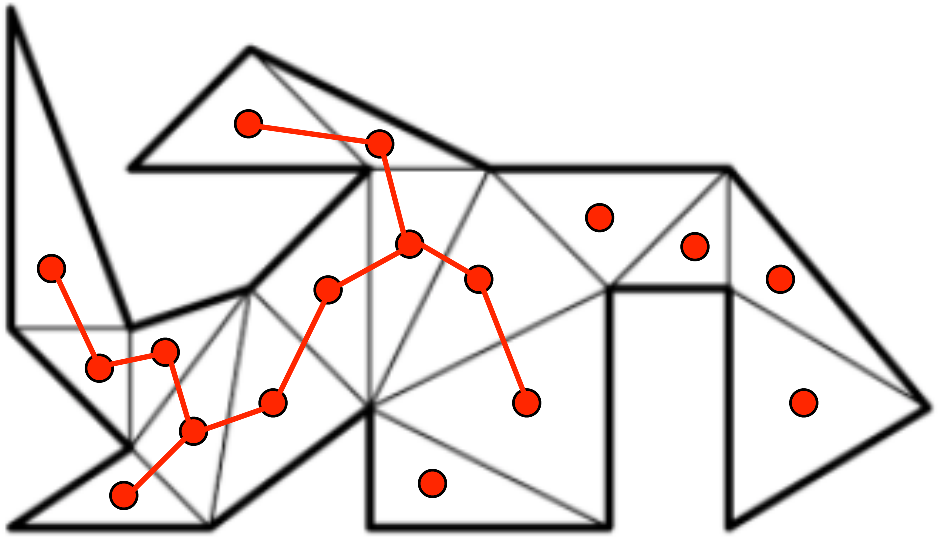
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



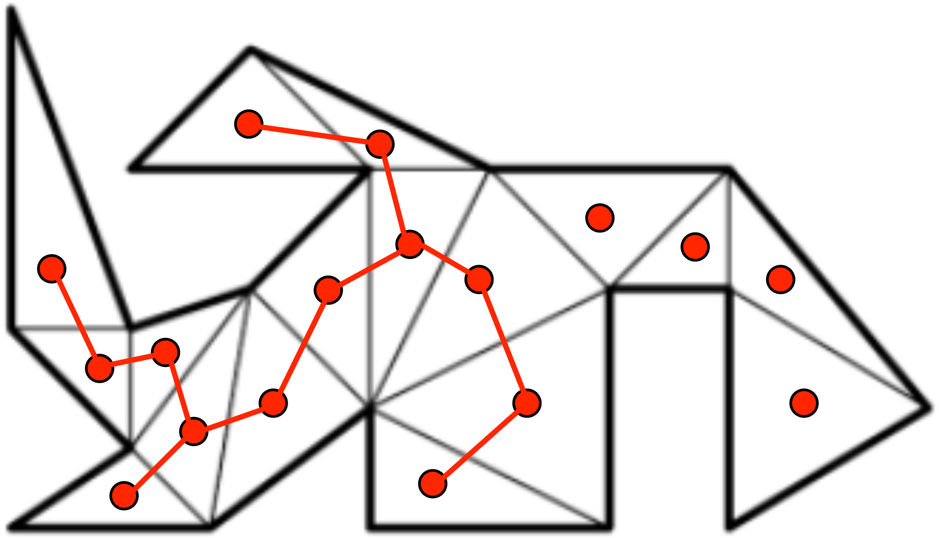
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



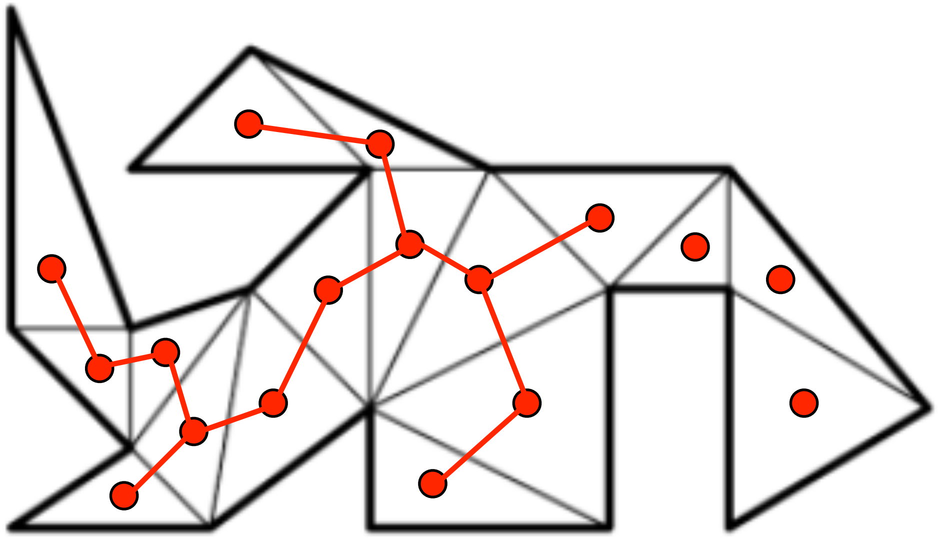
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



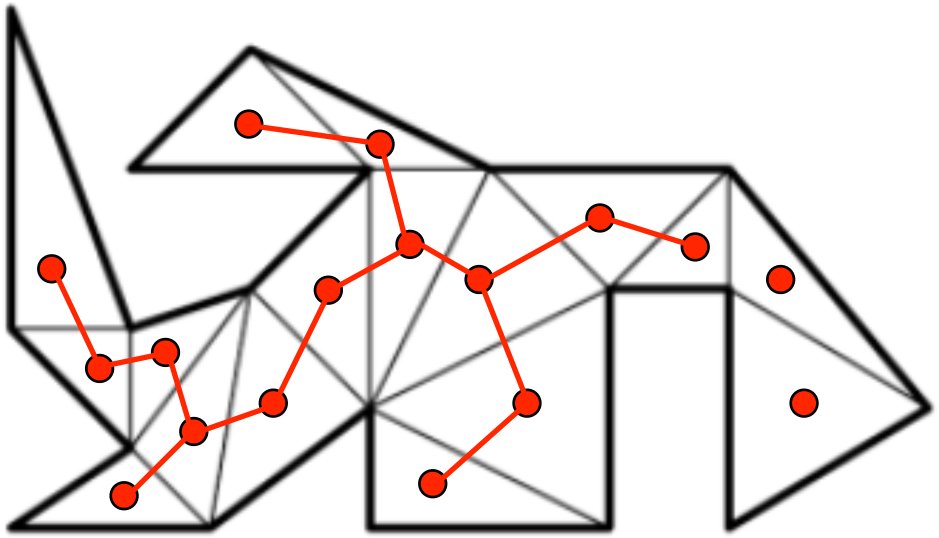
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



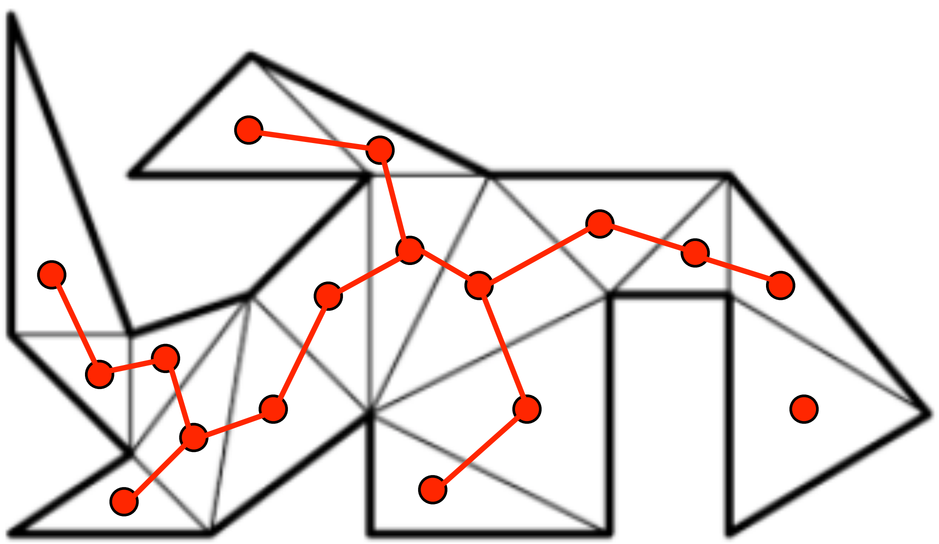
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



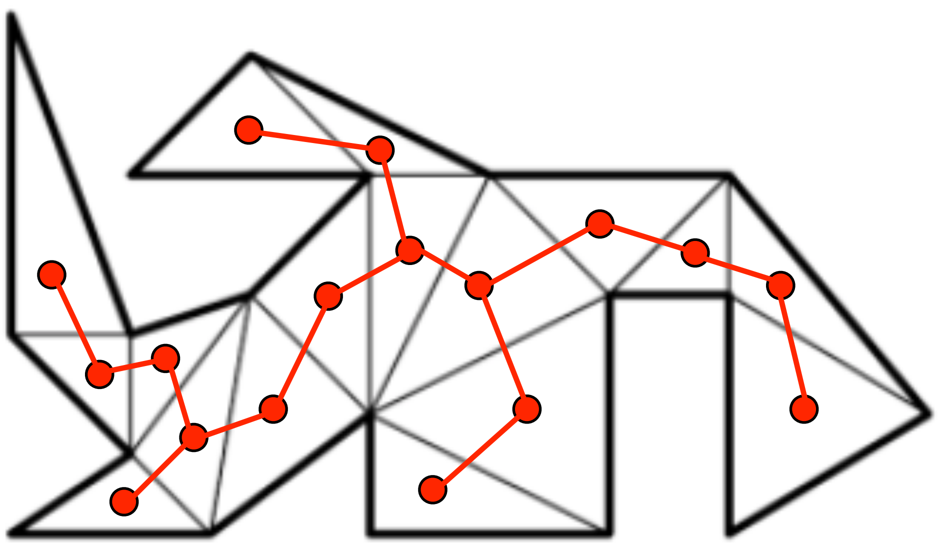
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.



Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

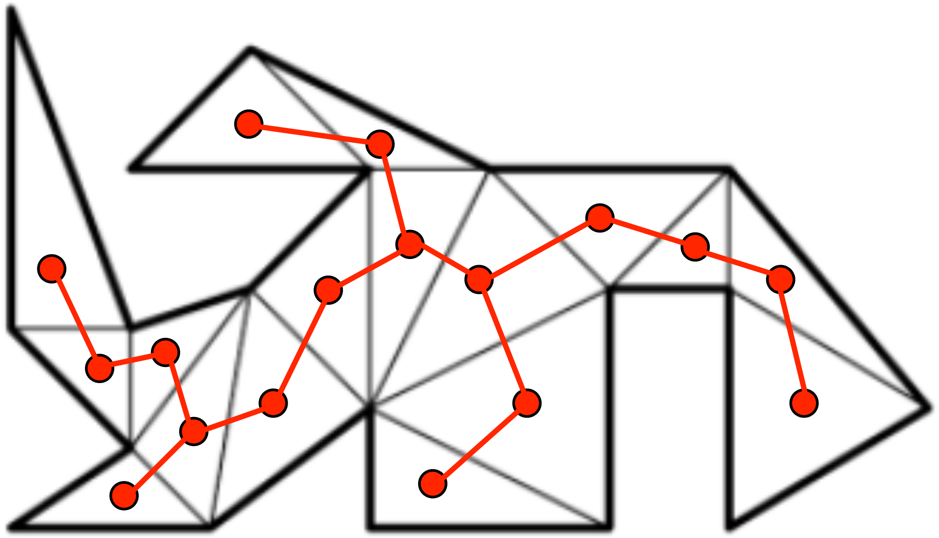


Technische Universität Braunschweig

Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

Proof:

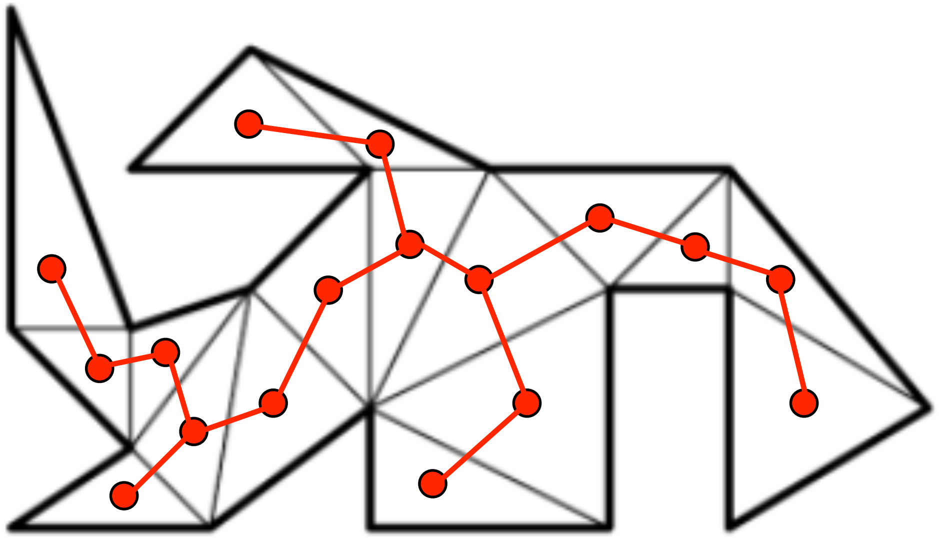


Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.



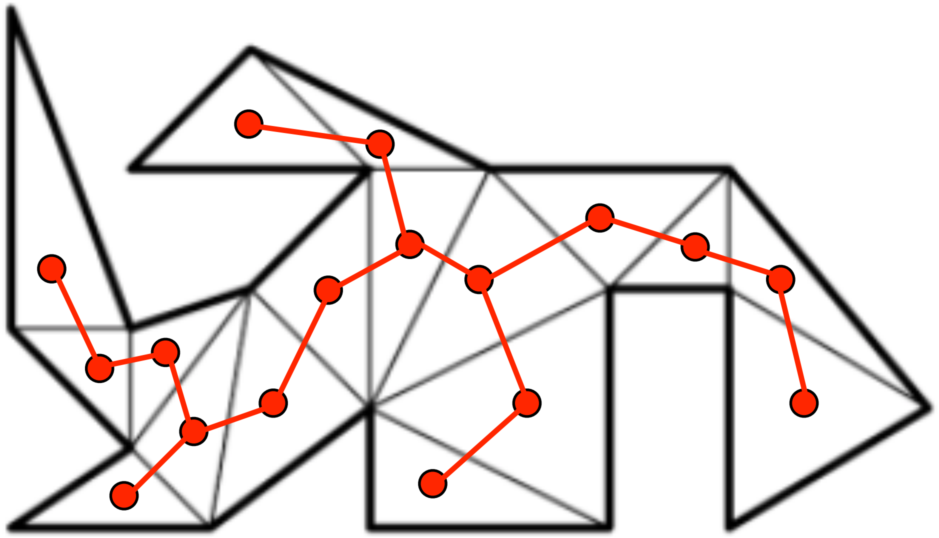
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.



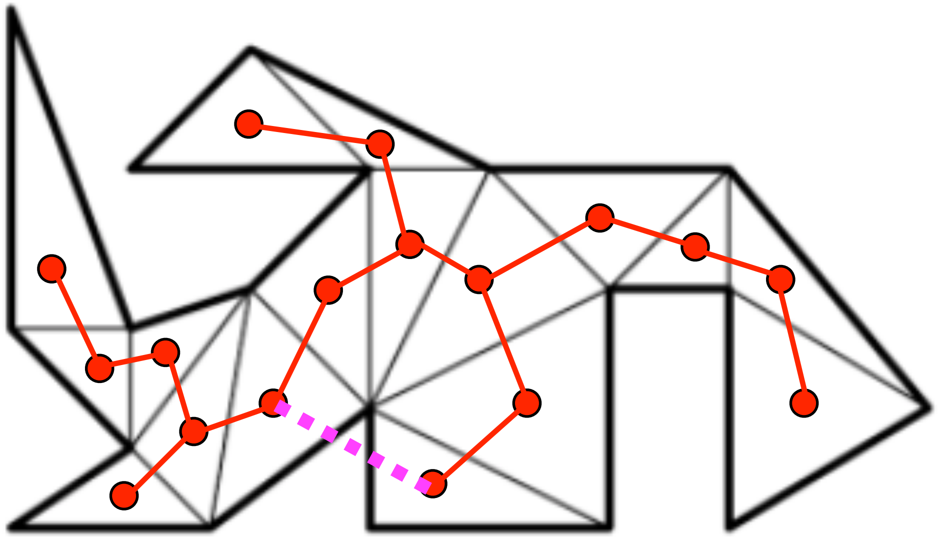
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.



Theorem 5.9

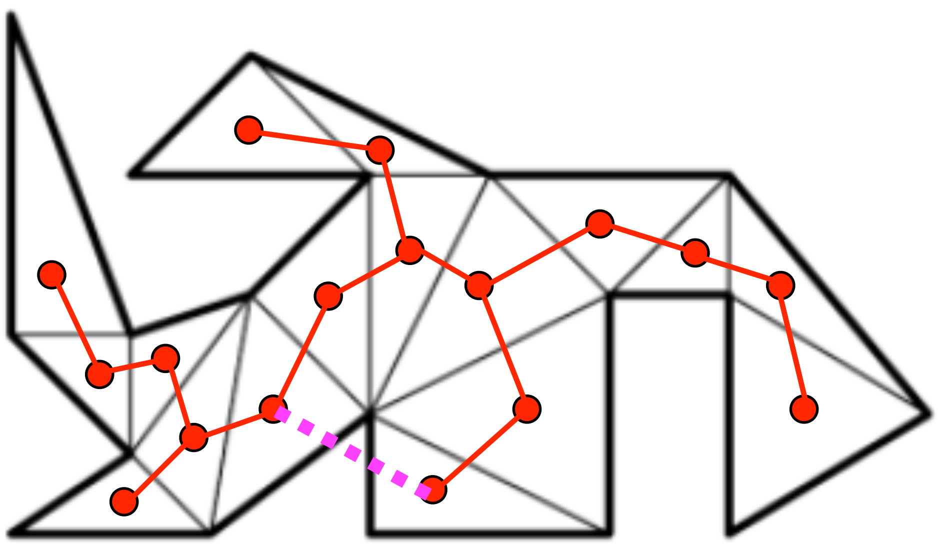
The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.

Note that each dual edge corresponds to



Technische Universität Braunschweig

Theorem 5.9

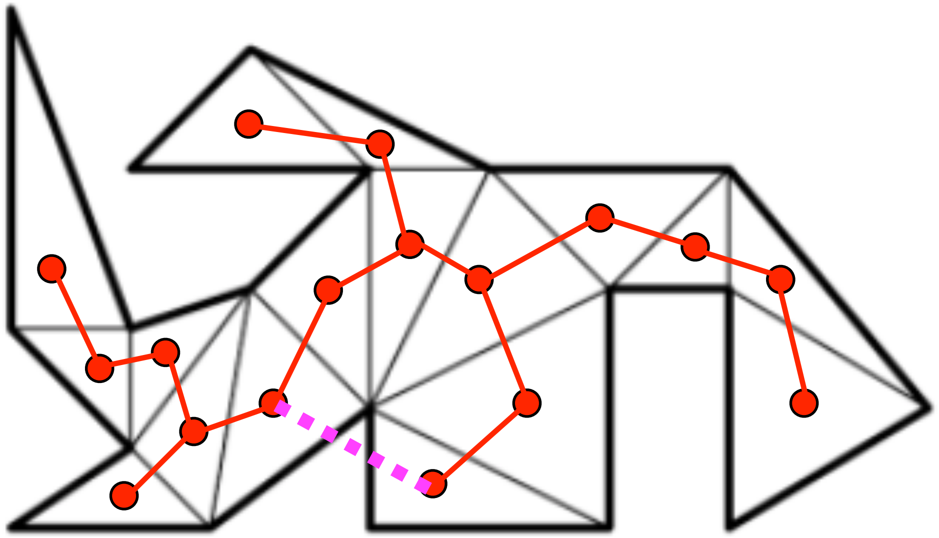
The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.

Note that each dual edge corresponds to a path of interior polygon points.



Theorem 5.9

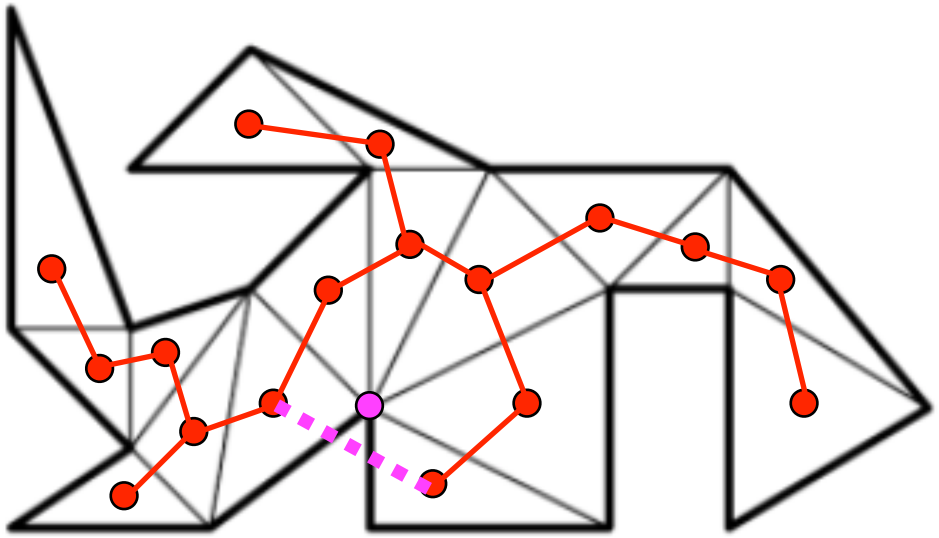
The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.

Note that each dual edge corresponds to a path of interior polygon points.



Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

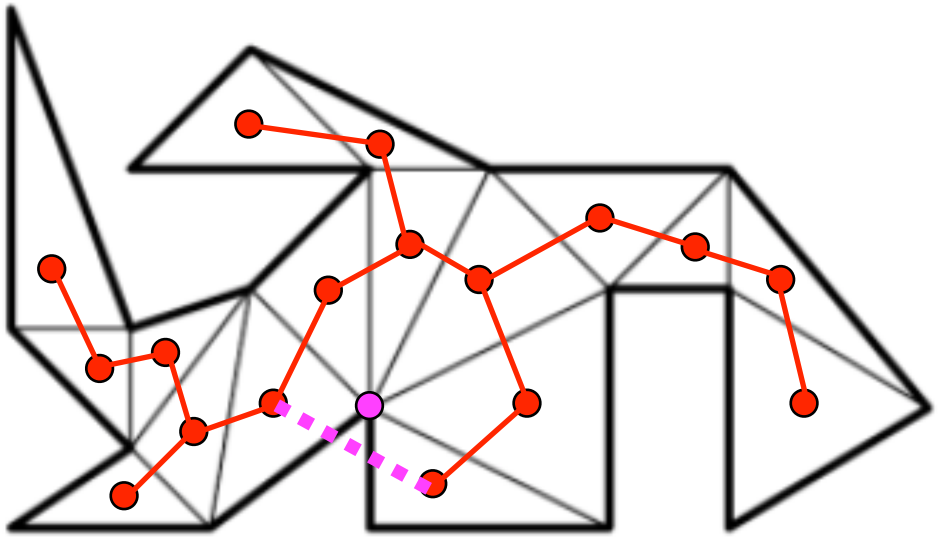
Proof:

The dual graph is trivially connected.

Assume it contains a cycle.

Note that each dual edge corresponds to a path of interior polygon points.

Therefore, a cycle corresponds to a closed path of interior points that surrounds a boundary vertex – a contradiction to a simple polygon having a connected boundary.



Technische Universität Braunschweig

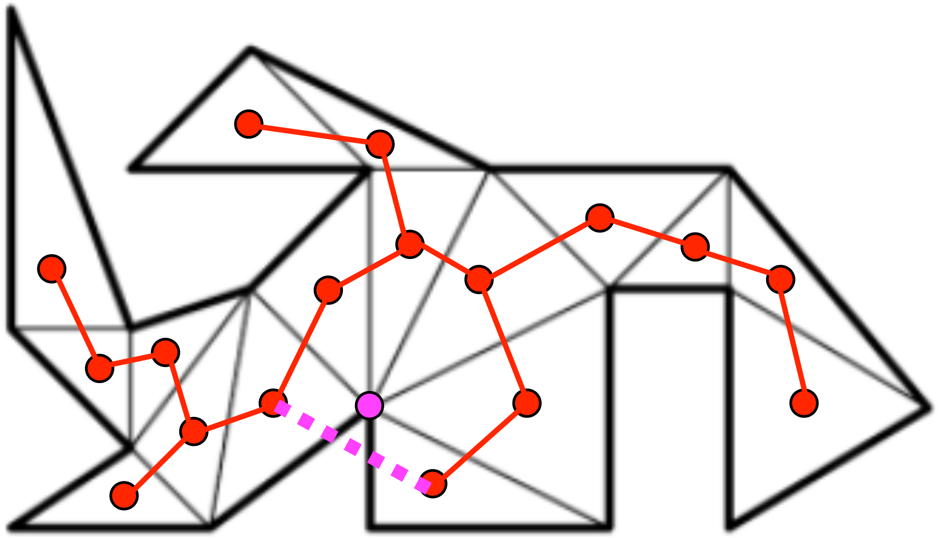
Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

Proof:

The dual graph is trivially connected.

Assume it contains a cycle.



Note that each dual edge corresponds to a path of interior polygon points.

Therefore, a cycle corresponds to a closed path of interior points that surrounds a boundary vertex - a contradiction to a simple polygon having a connected boundary.

Corollary 5.10

Theorem 5.9

The dual graph of a triangulation of a simple polygon is a tree.

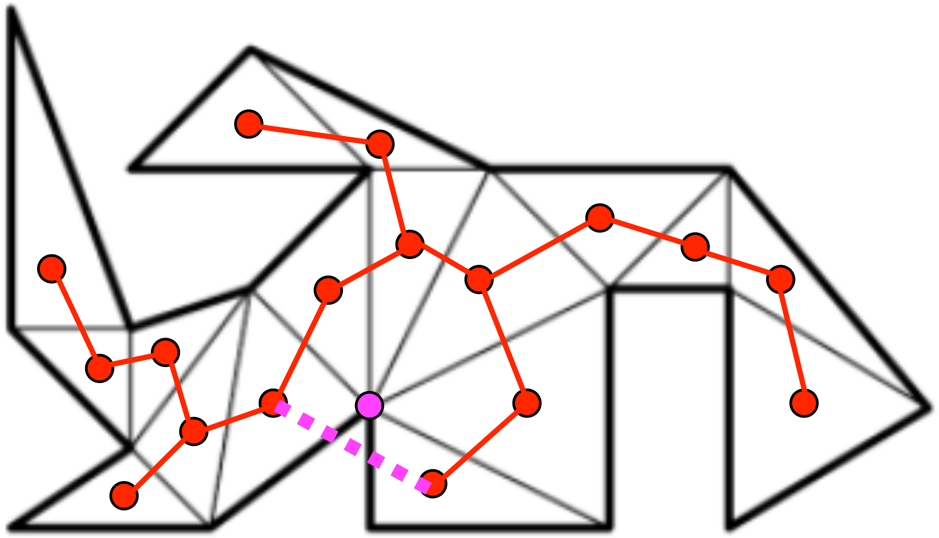
Proof:

The dual graph is trivially connected.

Assume it contains a cycle.

Note that each dual edge corresponds to a path of interior polygon points.

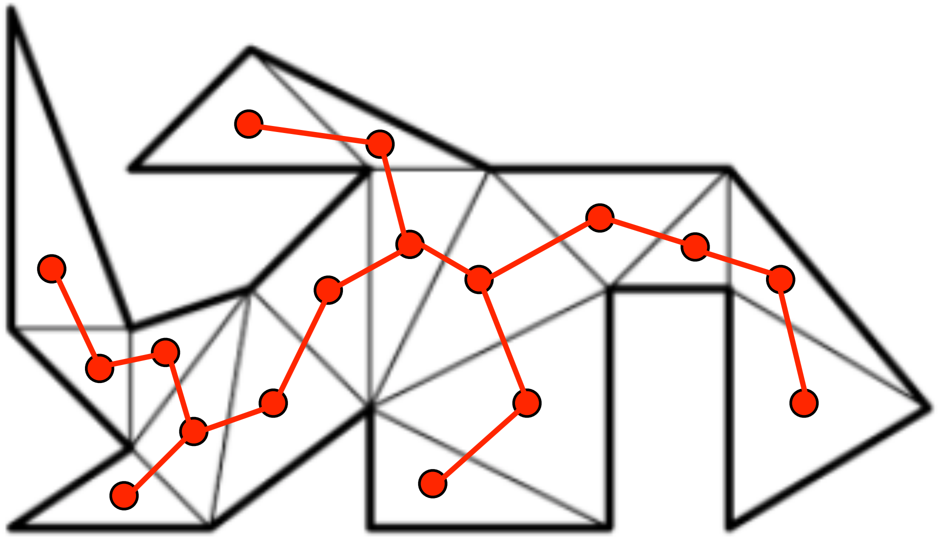
Therefore, a cycle corresponds to a closed path of interior points that surrounds a boundary vertex – a contradiction to a simple polygon having a connected boundary.



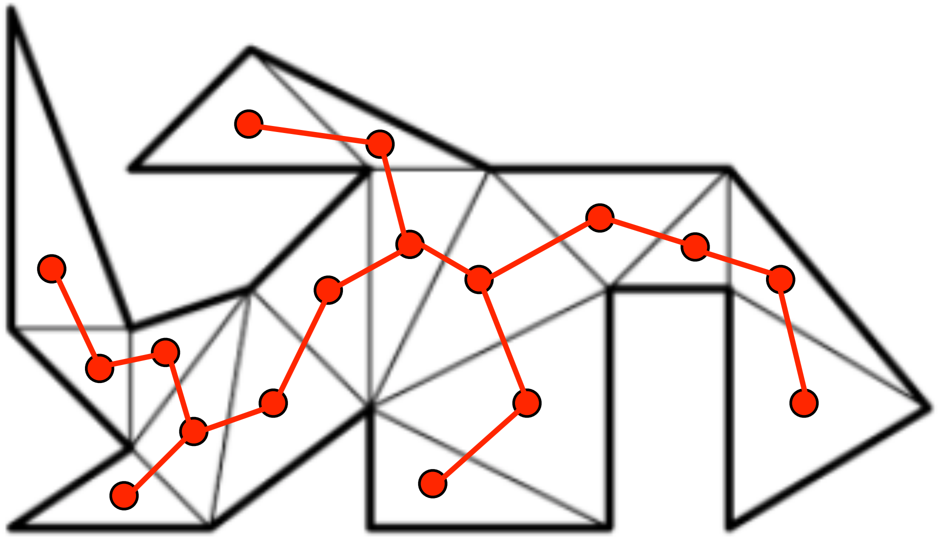
Corollary 5.10

Every triangulation of a simple polygon has two „ears“ - leaves in the dual.

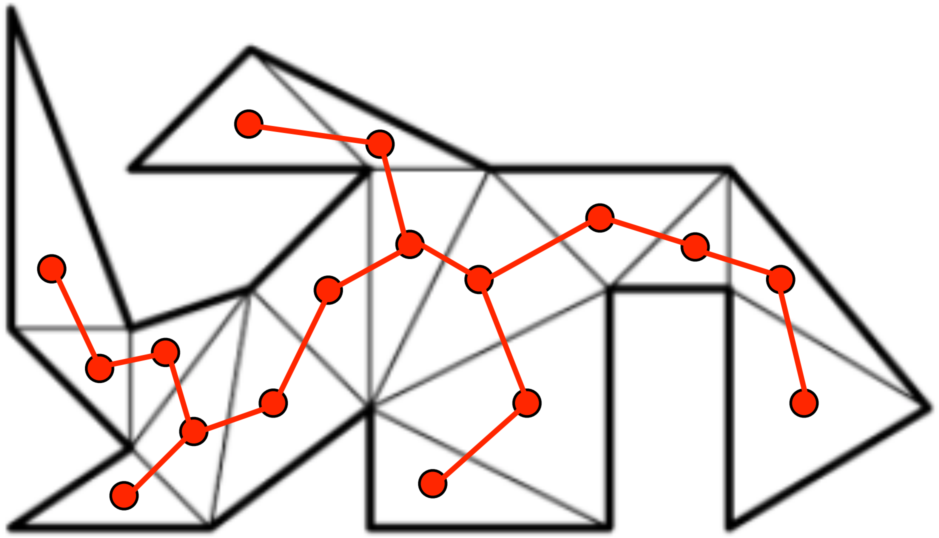
1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation



Input:

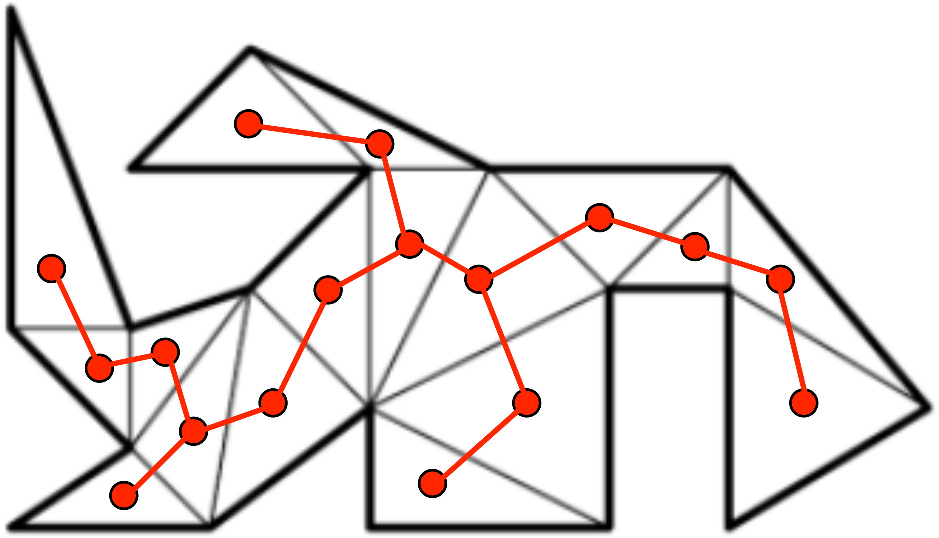


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.



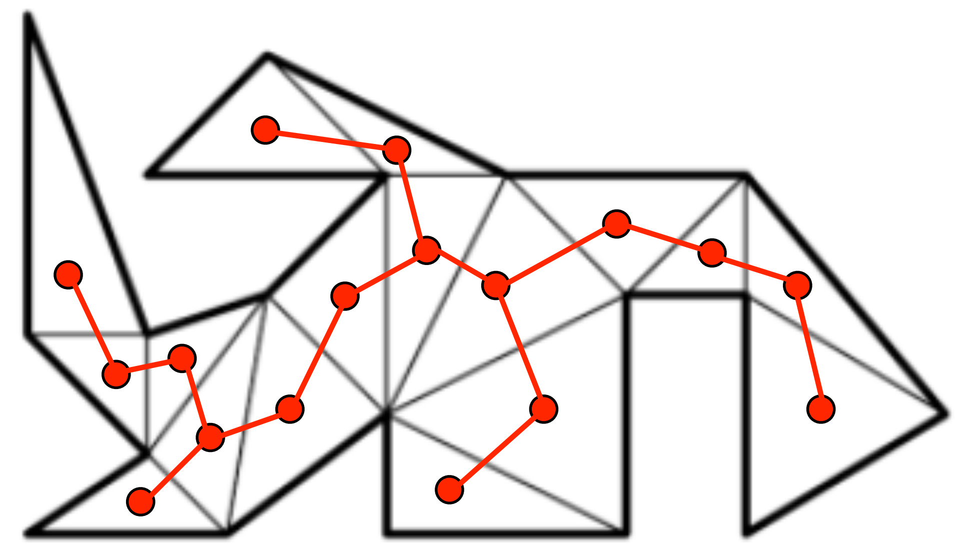
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output:



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

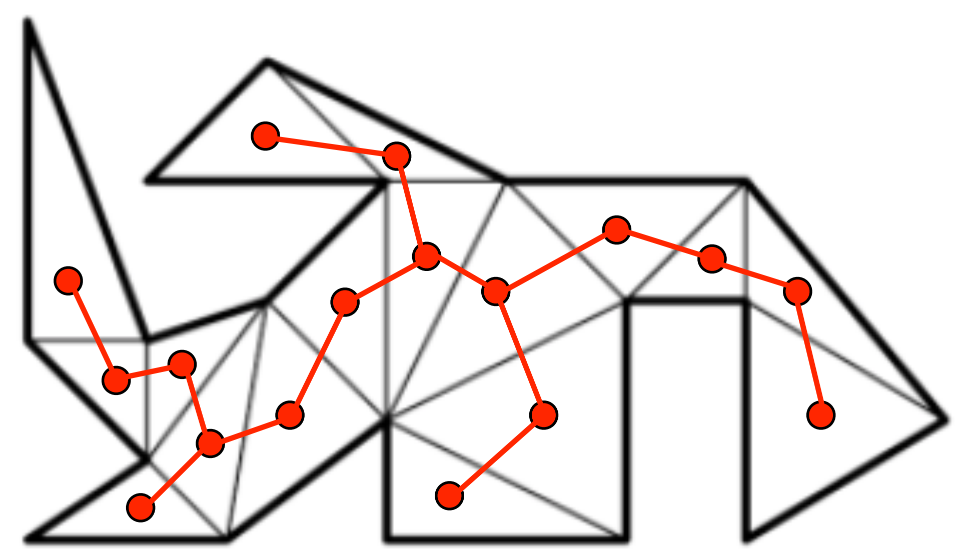
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:



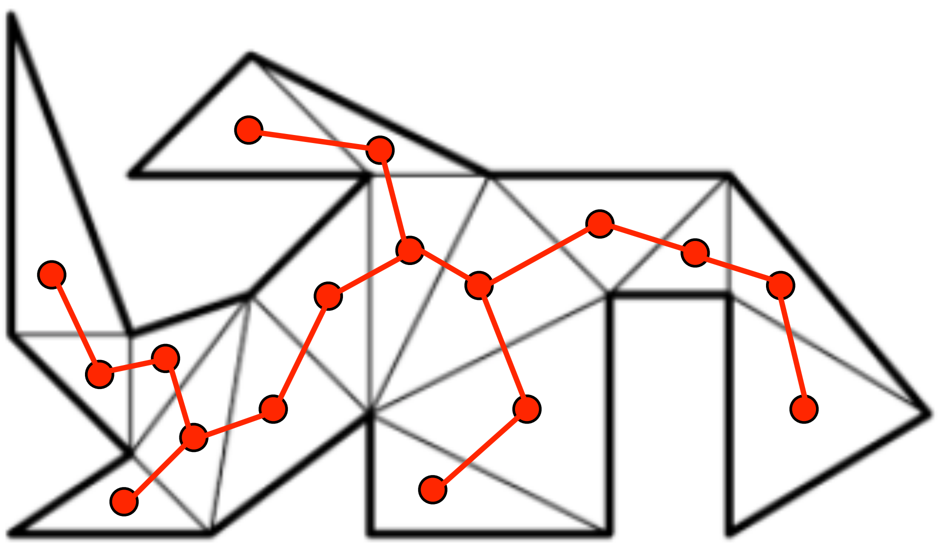
Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.

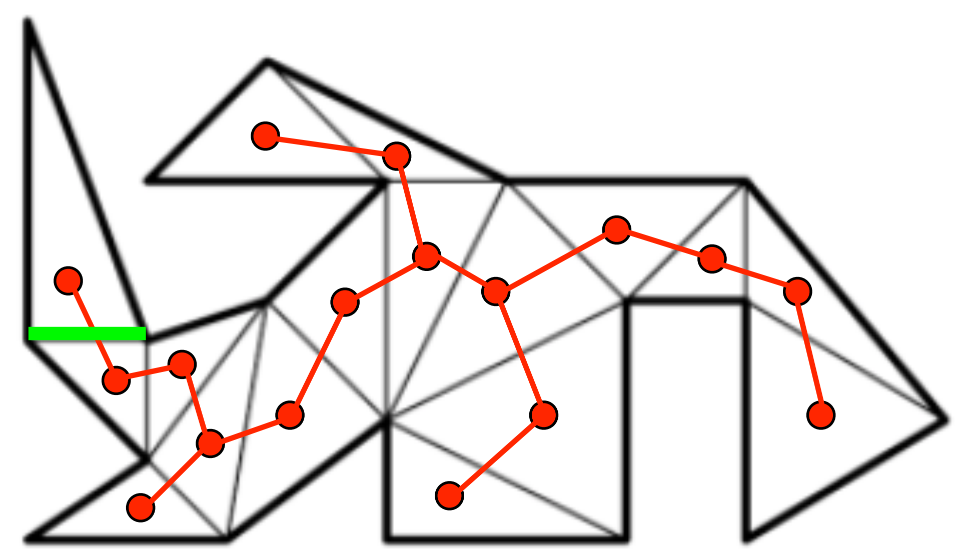


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.

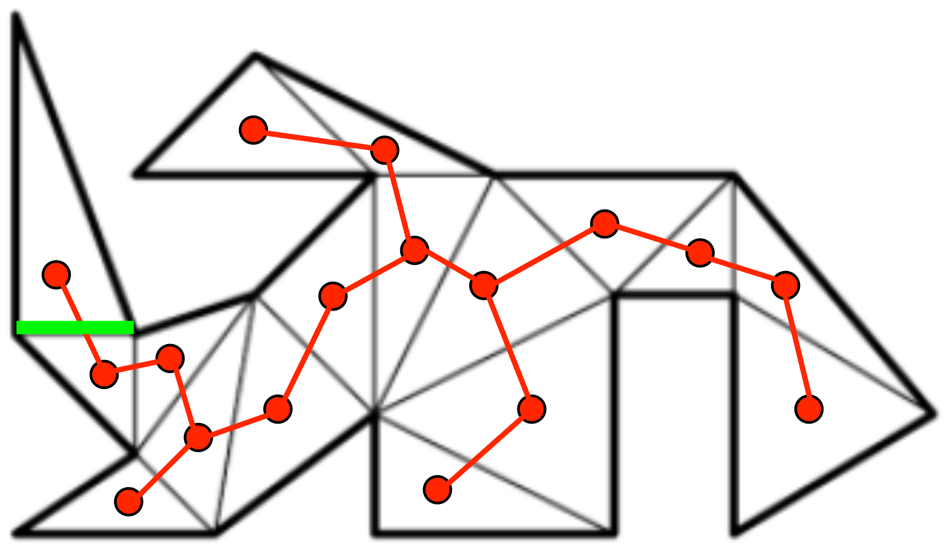


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.

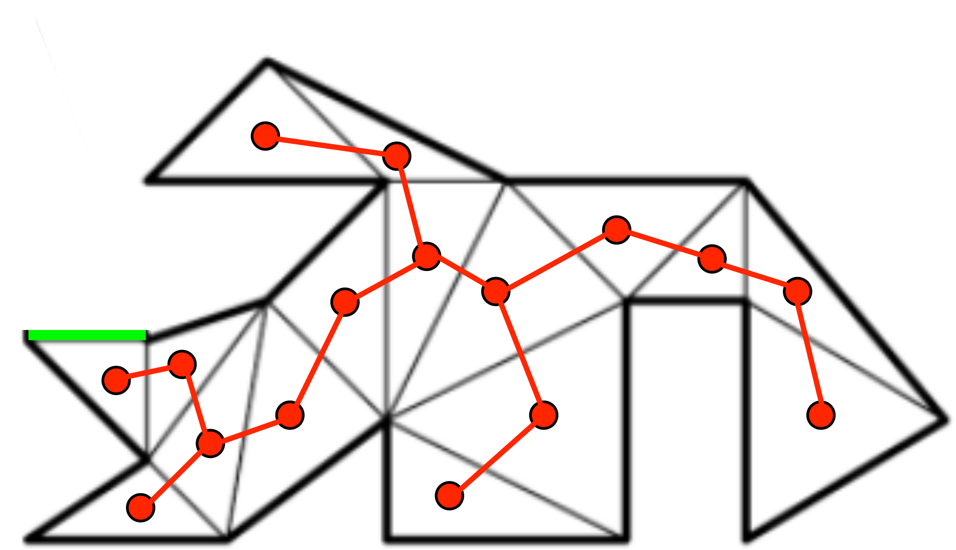


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, v_iv_j, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.

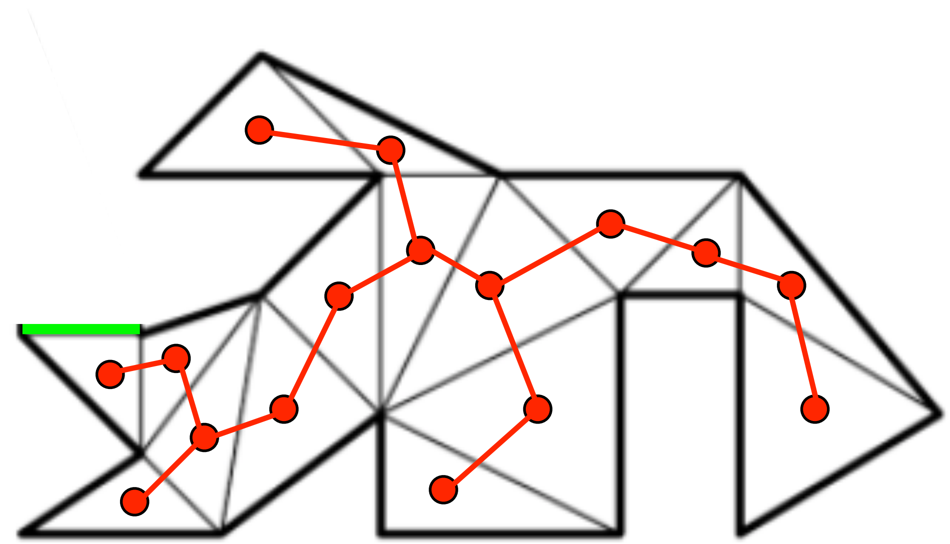


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.

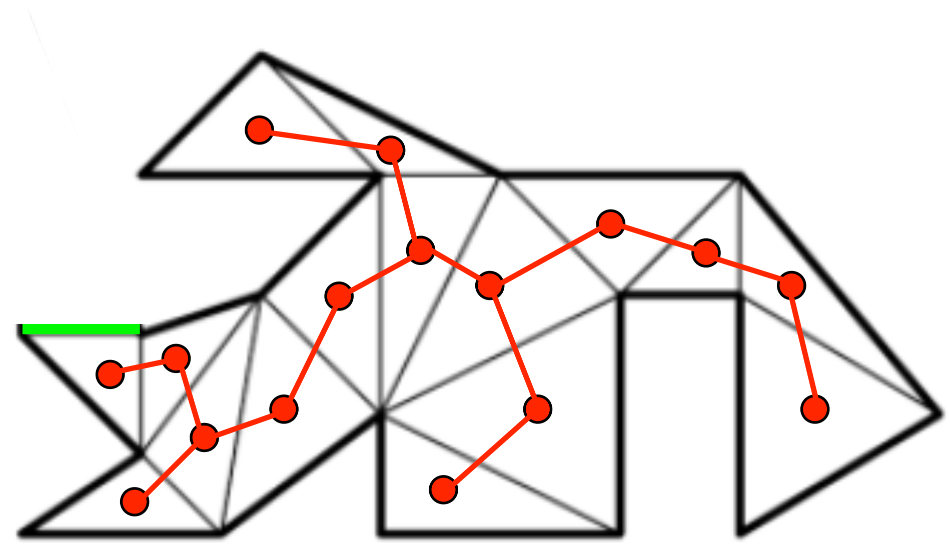


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

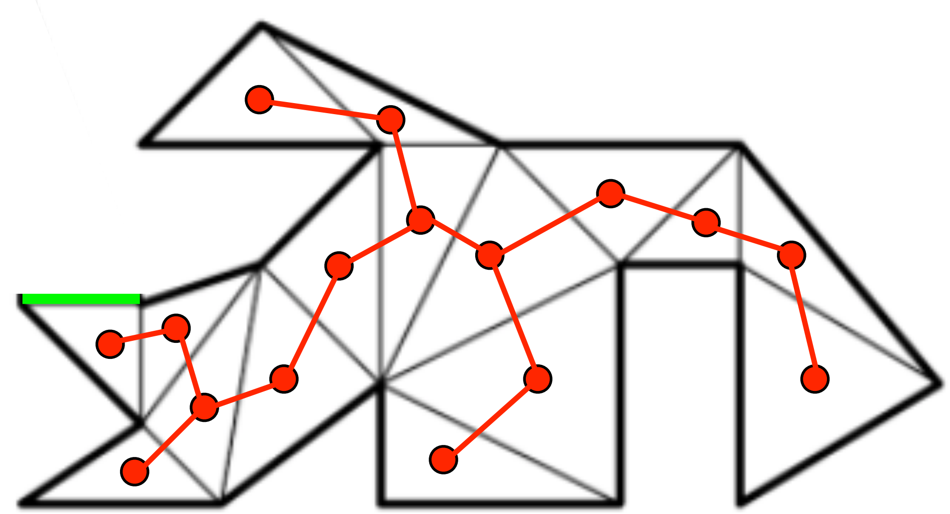
Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

- Checking convexity:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

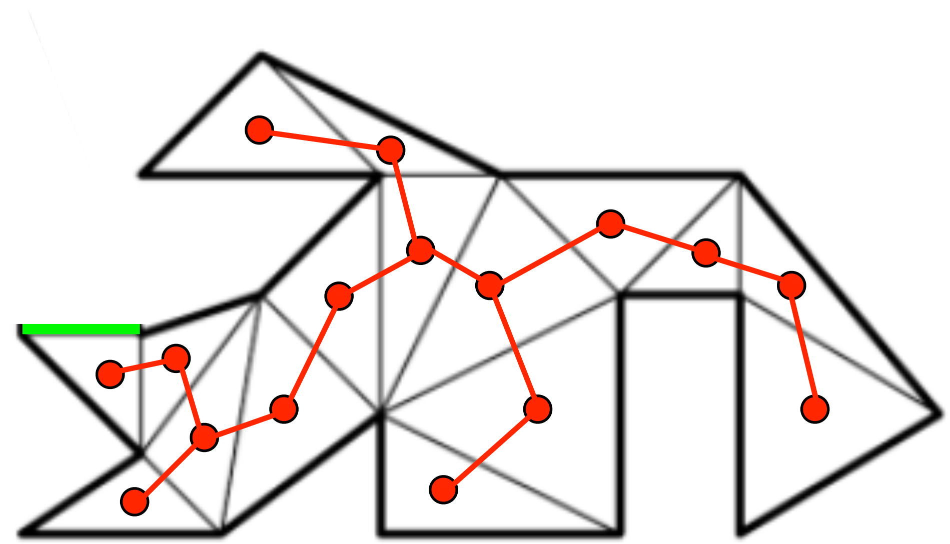
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.

Runtime:

- Checking convexity: $O(1)$



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

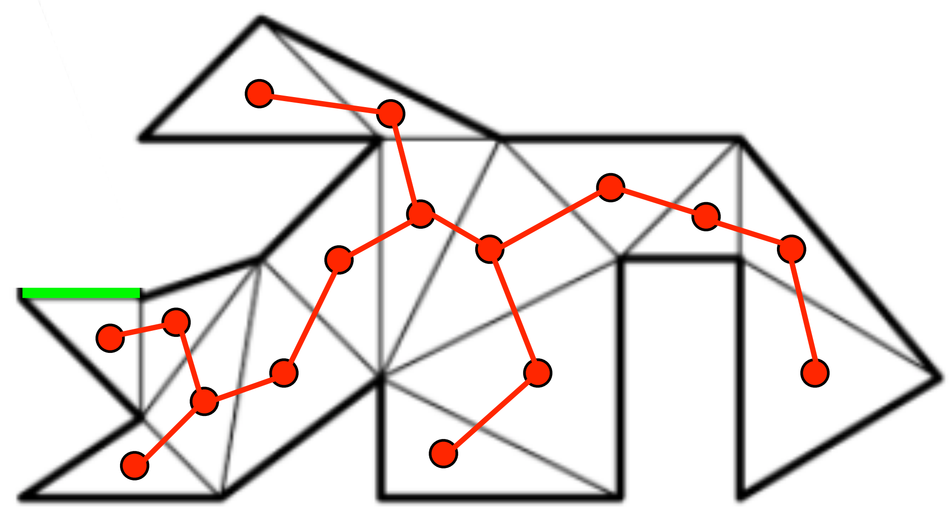
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.

Runtime:

- Checking convexity: $O(1)$
- Checking whether vertex is and ear:

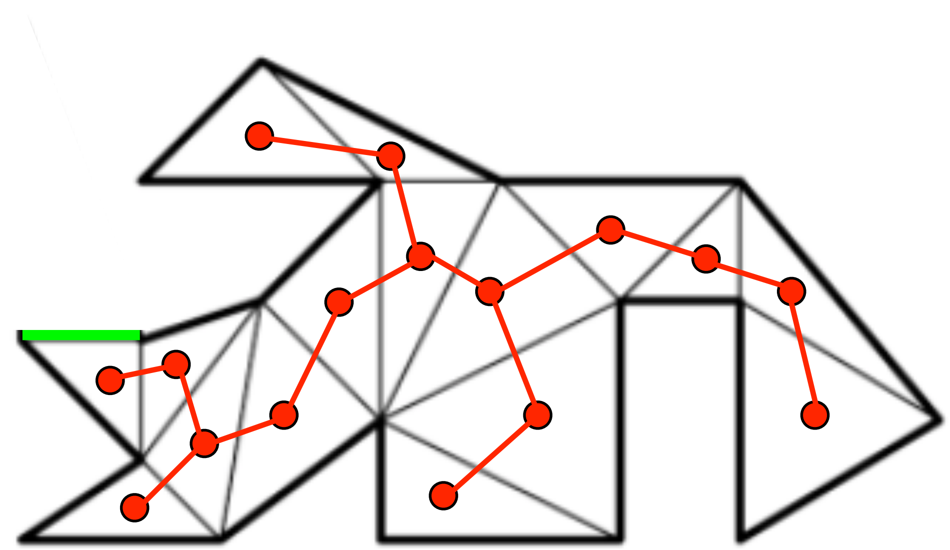


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

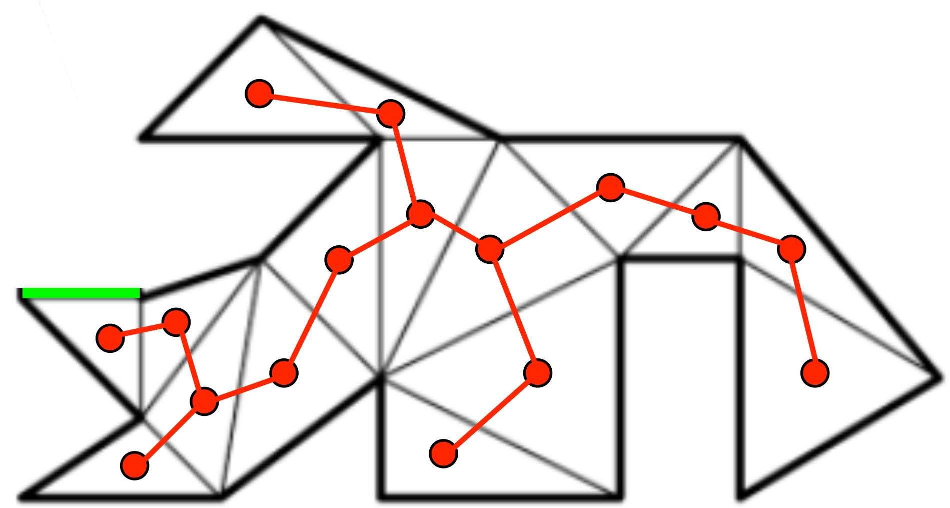
- Checking convexity: $O(1)$
- Checking whether vertex is and ear: $O(n)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

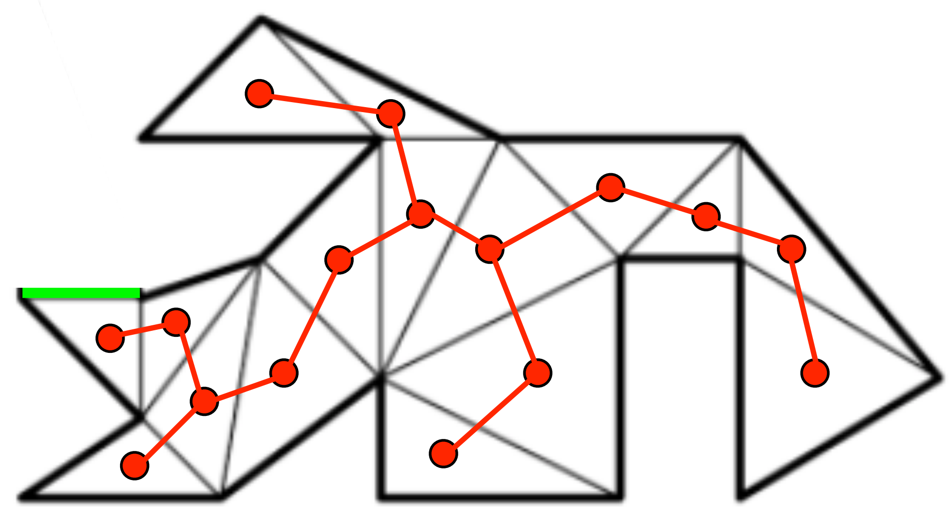
- Checking convexity: $O(1)$
- Checking whether vertex is and ear: $O(n)$
- Finding an ear:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

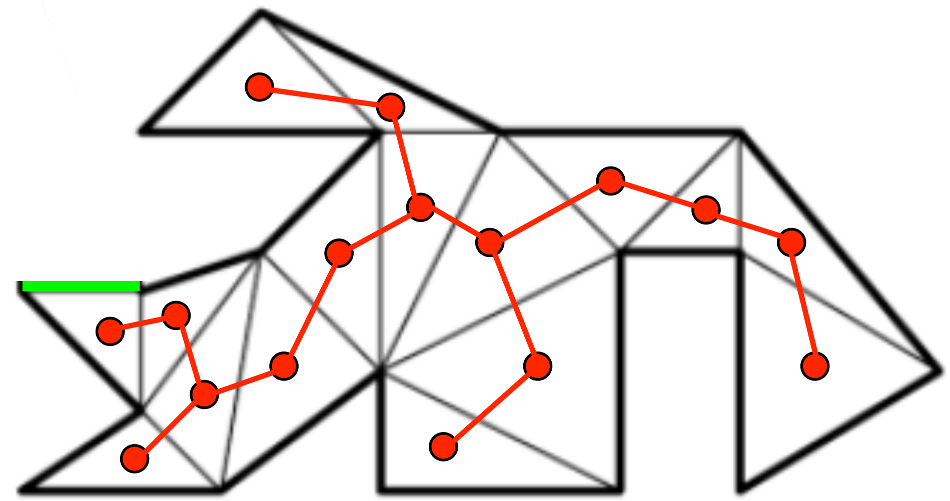
- Checking convexity: $O(1)$
- Checking whether vertex is and ear: $O(n)$
- Finding an ear: $O(n^2)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



Runtime:

- Checking convexity: $O(1)$
- Checking whether vertex is and ear: $O(n)$
- Finding an ear: $O(n^2)$

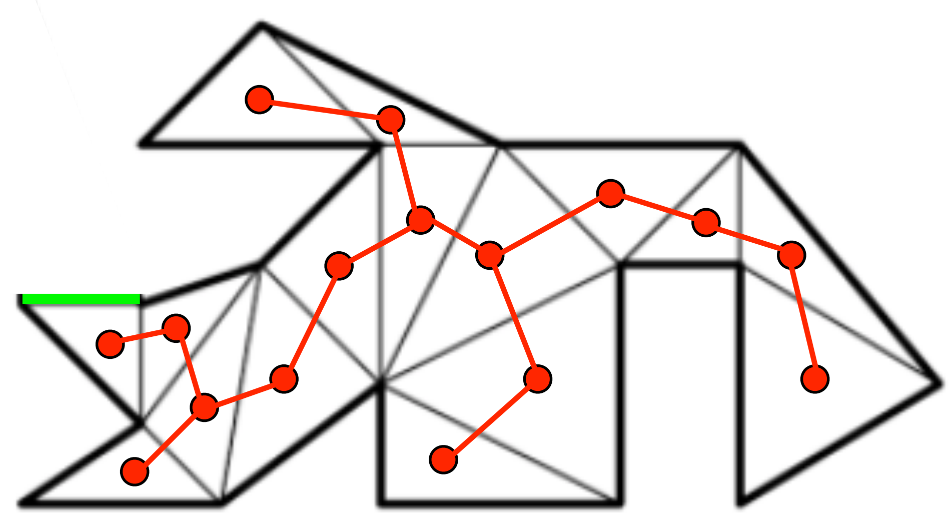
Total:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find an ear.
- Remove it.
- Update polygon.



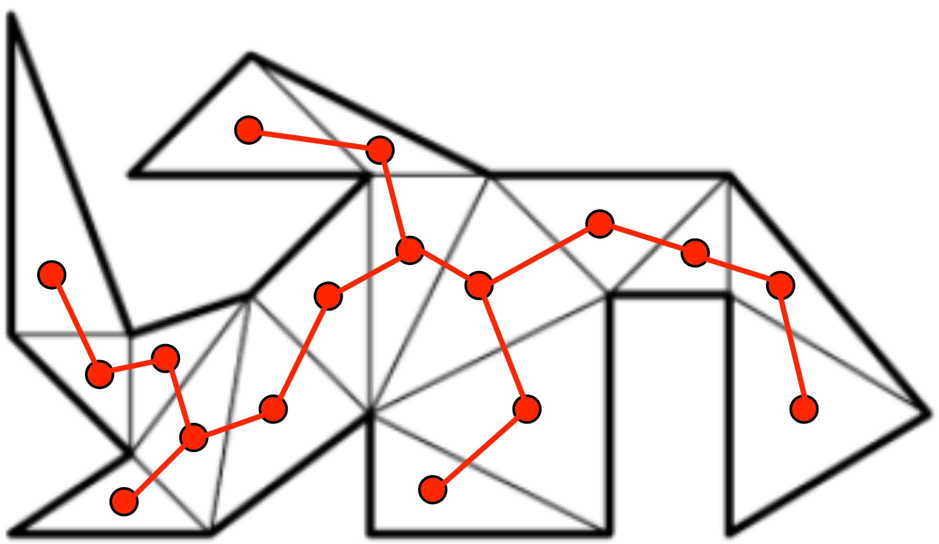
Runtime:

- Checking convexity: $O(1)$
- Checking whether vertex is and ear: $O(n)$
- Finding an ear: $O(n^2)$

Total: $O(n^3)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.



Input: $v_1, \ldots, v_n$, sorted list of t

Output: List of internal diagonals

ELSEVIER

Pattern Recognition Letters
Volume 14, Issue 9, September 1993, Pages 719-722

Slicing an ear using prune-and-search

Hossam ElGindy, Hazel Everett, Godfried Toussaint

Show more

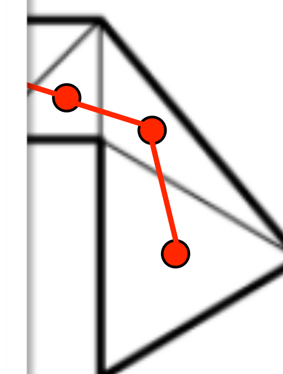
Share Cite

https://doi.org/10.1016/0167-8655(93)90141-Y Get rights and content

Abstract

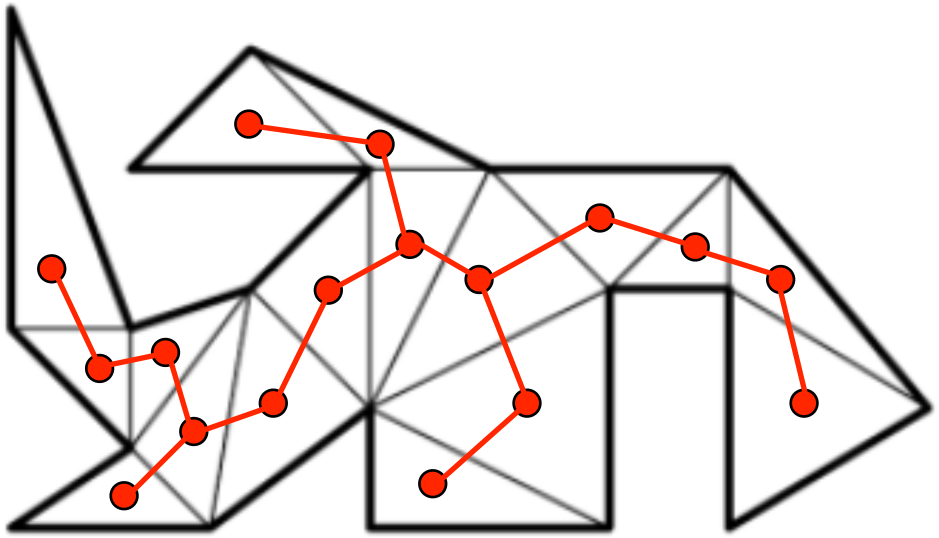
It is well known that a *diagonal* of a simple polygon *P* can be found in linear time with a simple and practically efficient algorithm. An *ear* of *P* is a triangle such that one of its edges is a diagonal of *P* and the remaining two edges are edges of *P*. An ear of *P* can easily be found by first triangulating *P* and subsequently searching the triangulation. However, although a polygon can be triangulated in linear time, such a procedure is conceptually difficult and not practically efficient. In this note we show that an *ear* of *P* can be found in linear time with a simple, practically efficient algorithm that does not require pre-triangulating *P*.

P.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

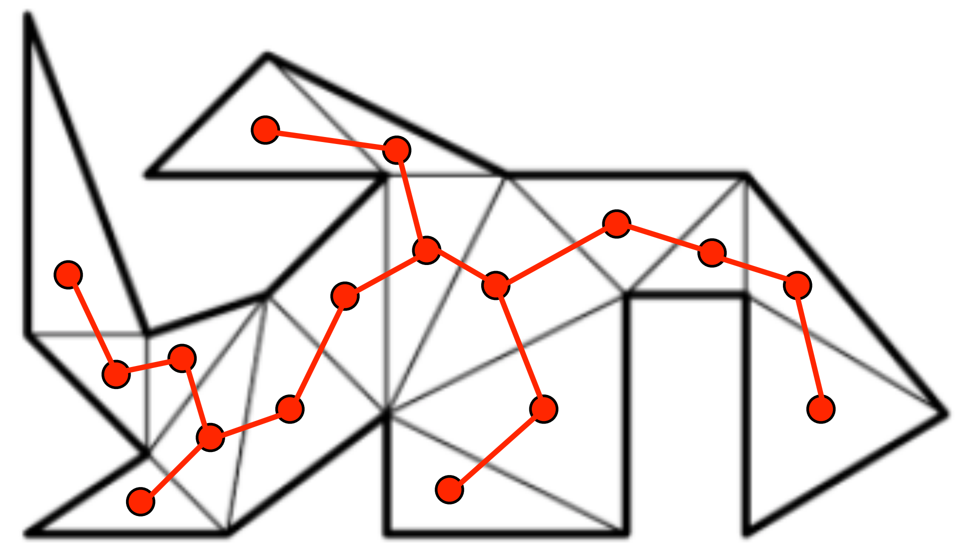
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

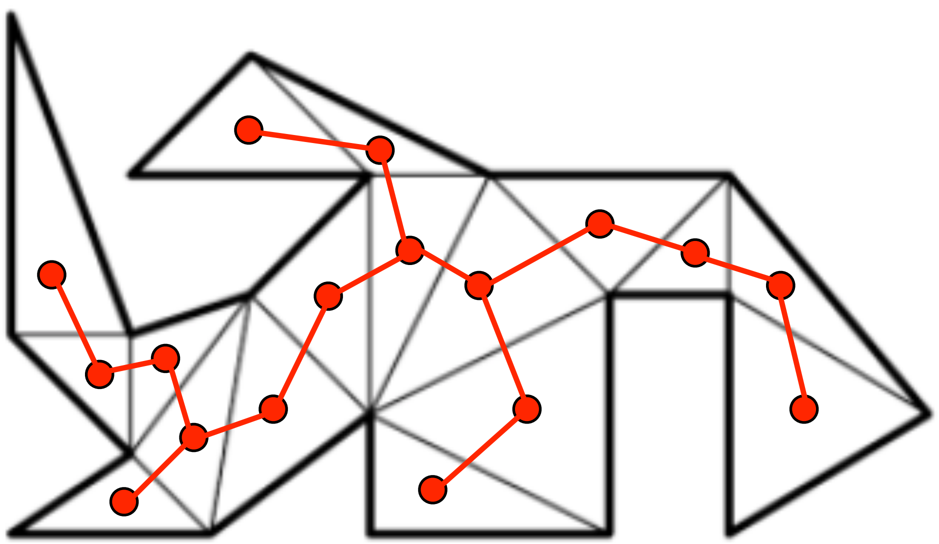


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, v_iv_j, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)



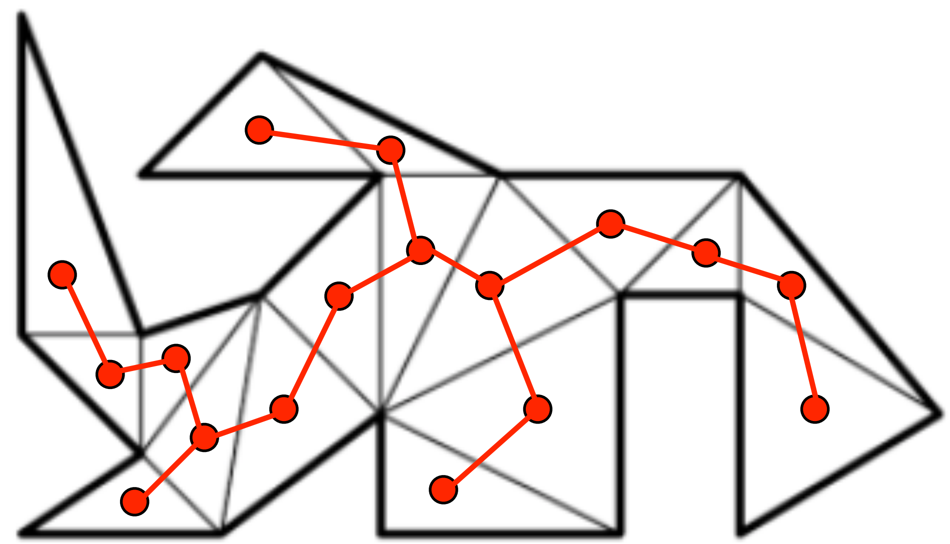
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)

- Find all convex vertices.



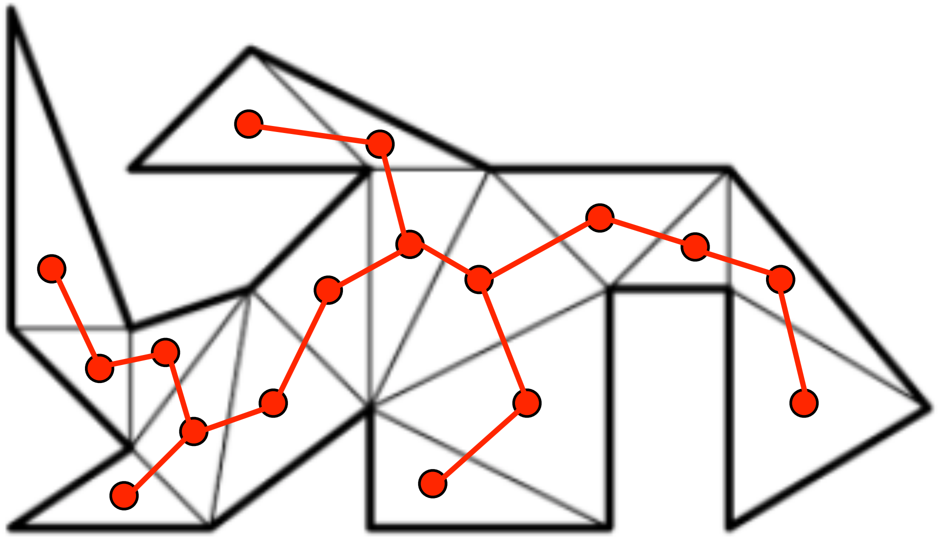
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$



Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

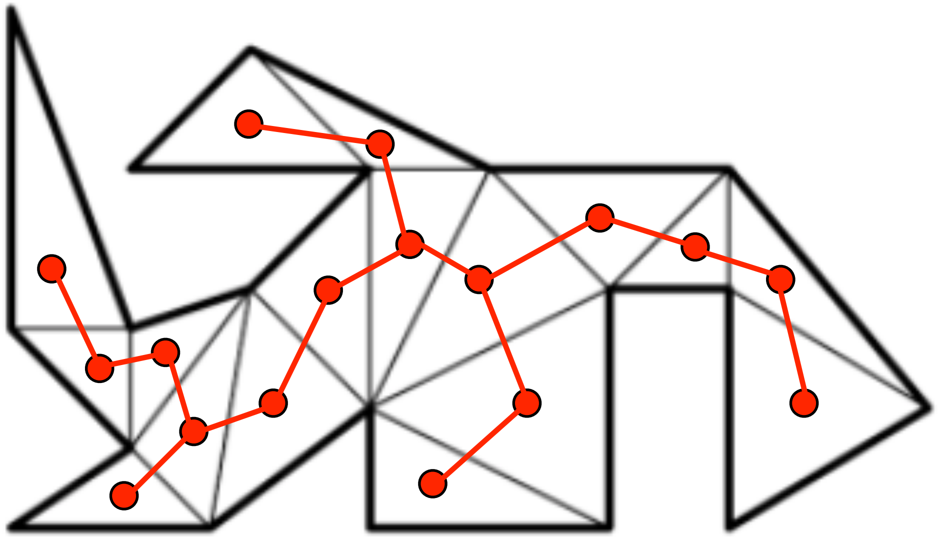
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)

- Find all convex vertices.
- Identify all ears.

$O(n)$



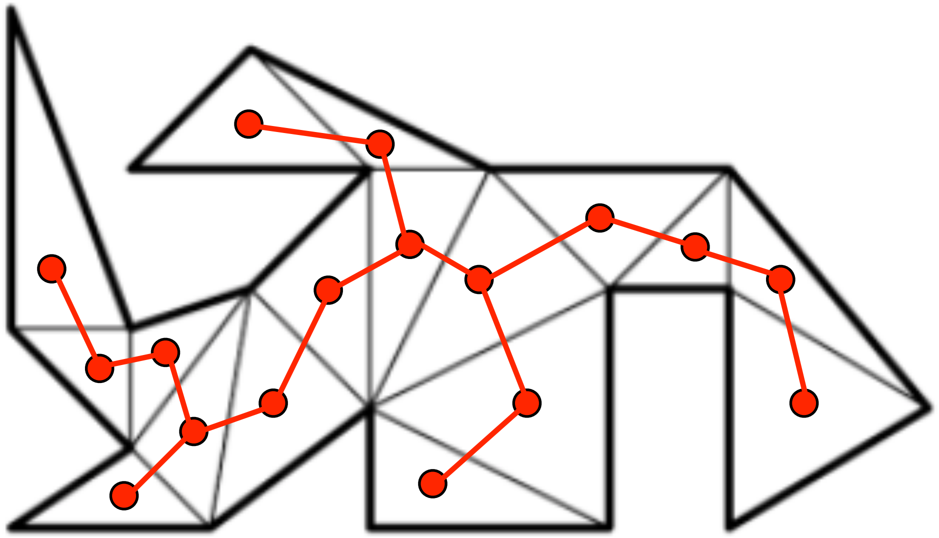
Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$



Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

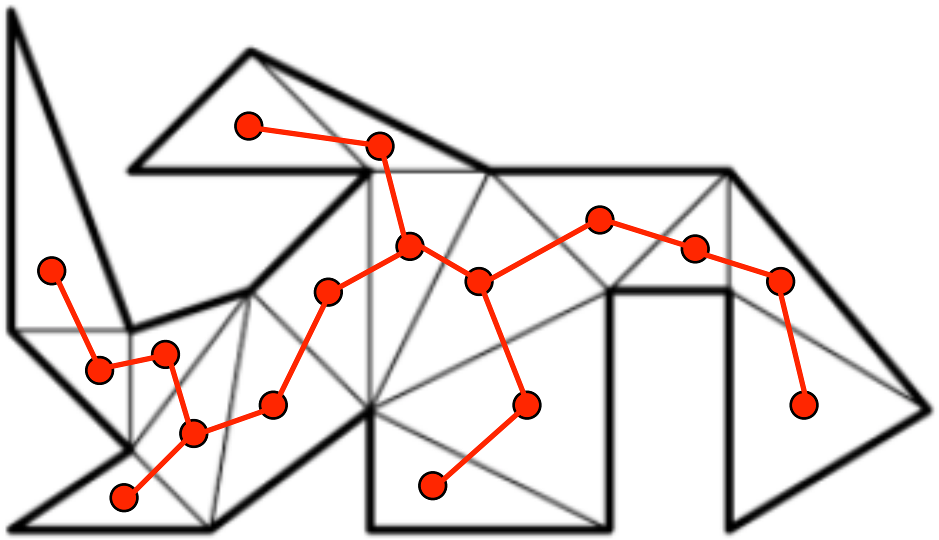
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

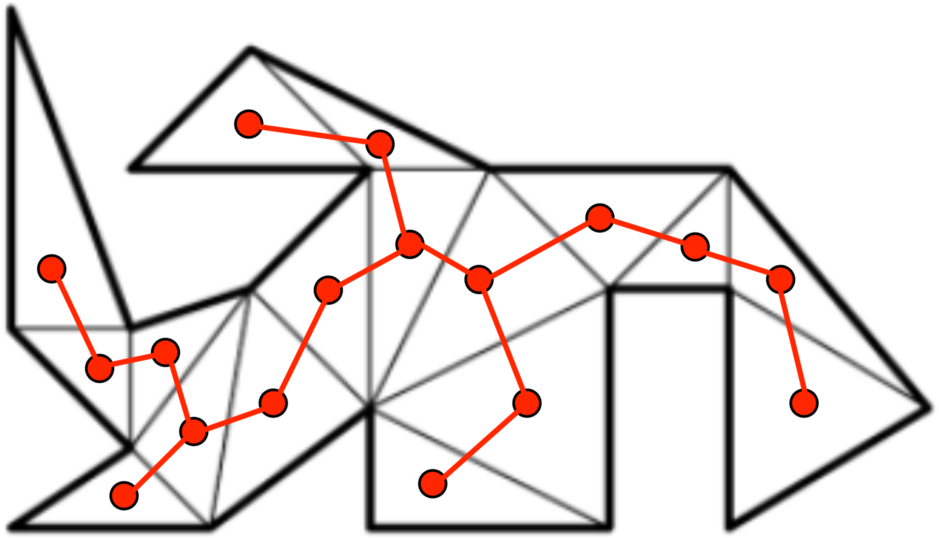
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

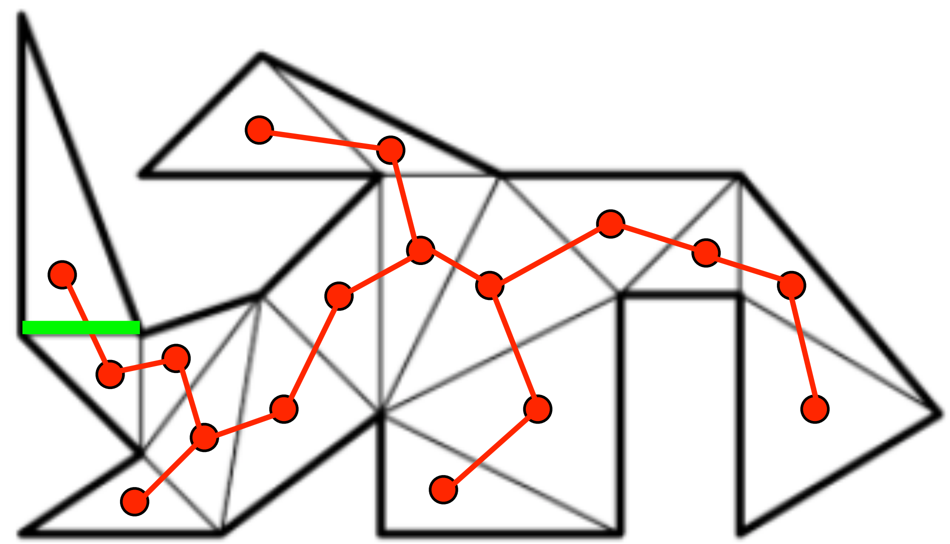
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

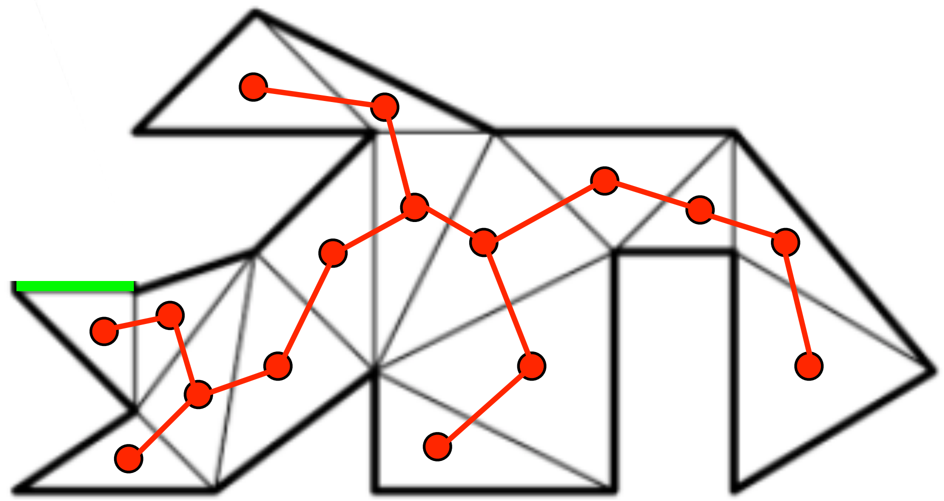
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

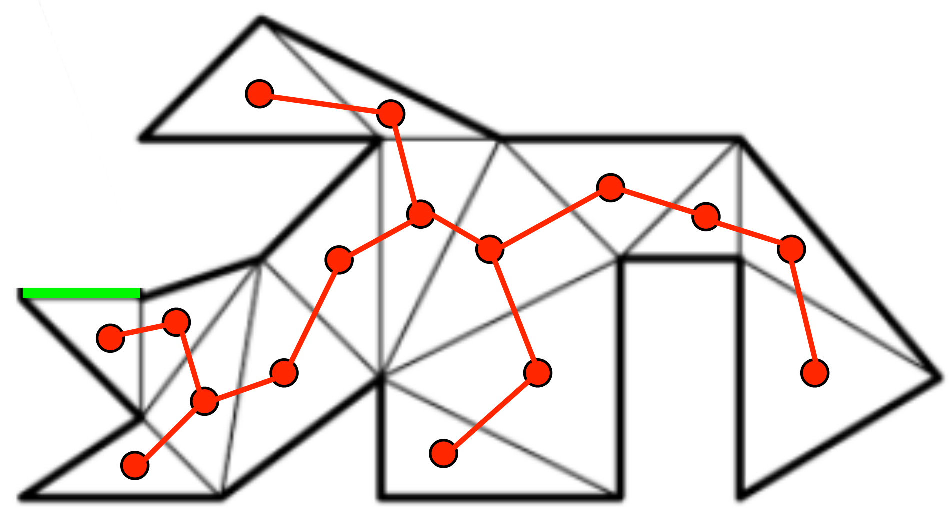
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

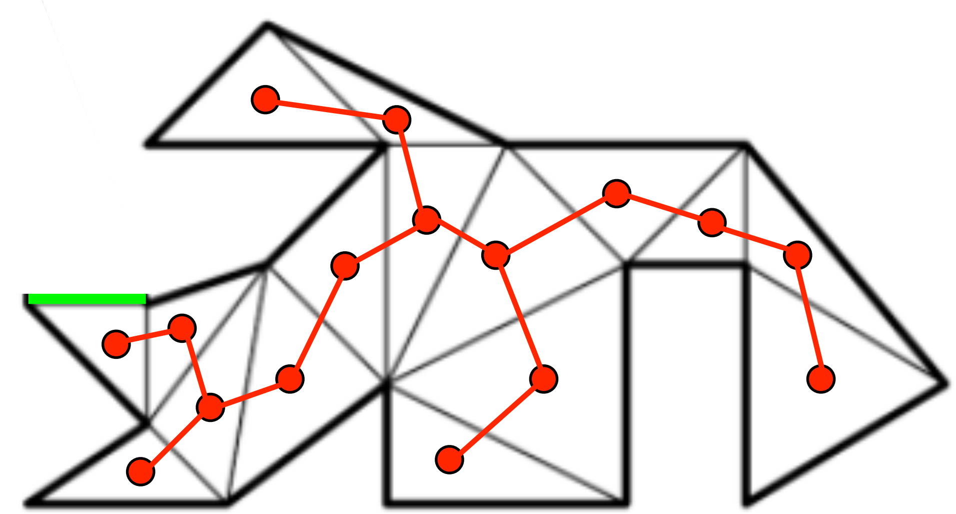
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity.



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

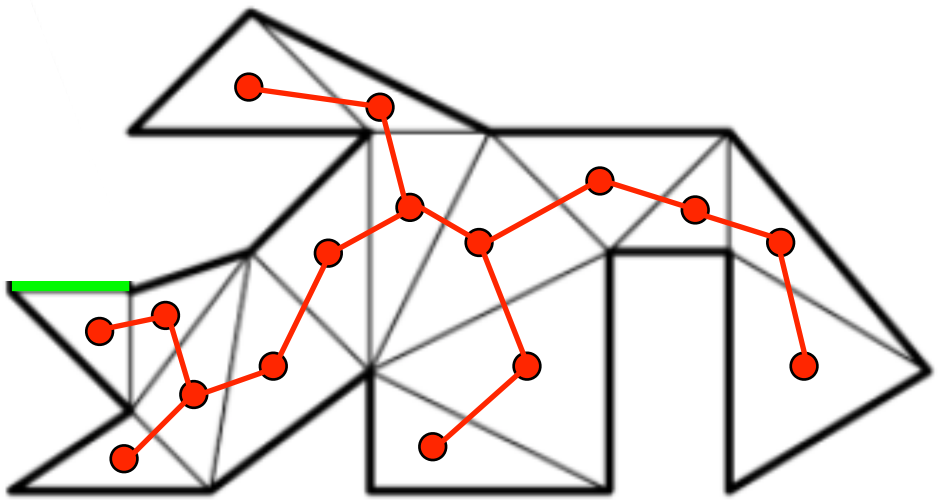
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity. $O(1)$



Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

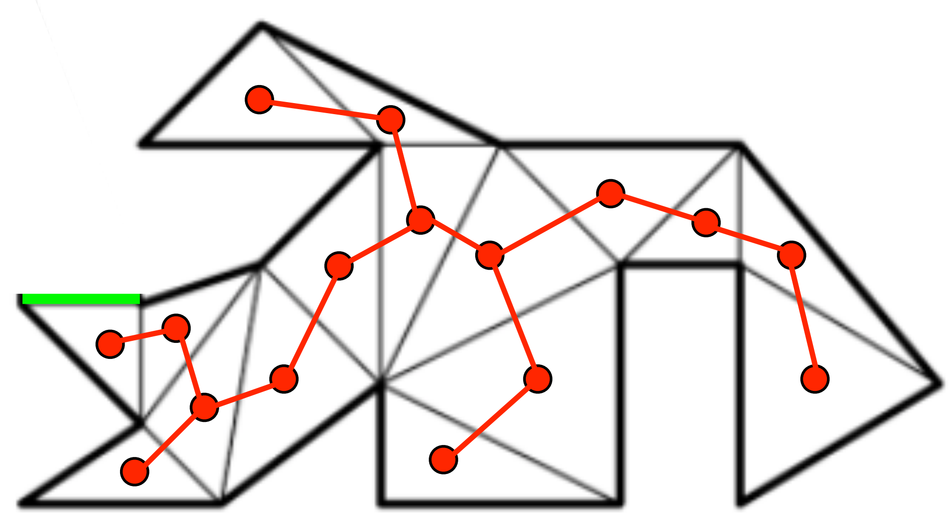
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity. $O(1)$
- Update ears.



Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

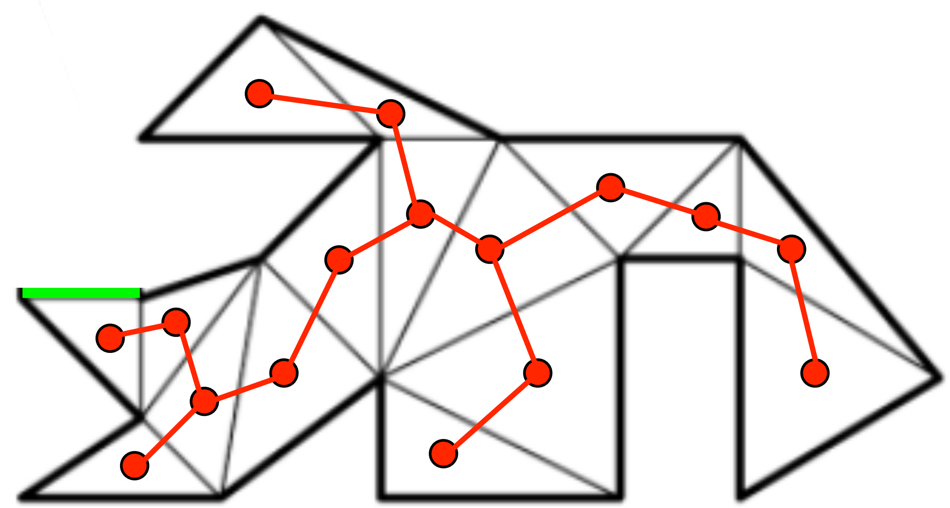
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity. $O(1)$
- Update ears. $O(n)$



Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, v_iv_j, determining a triangulation of P.

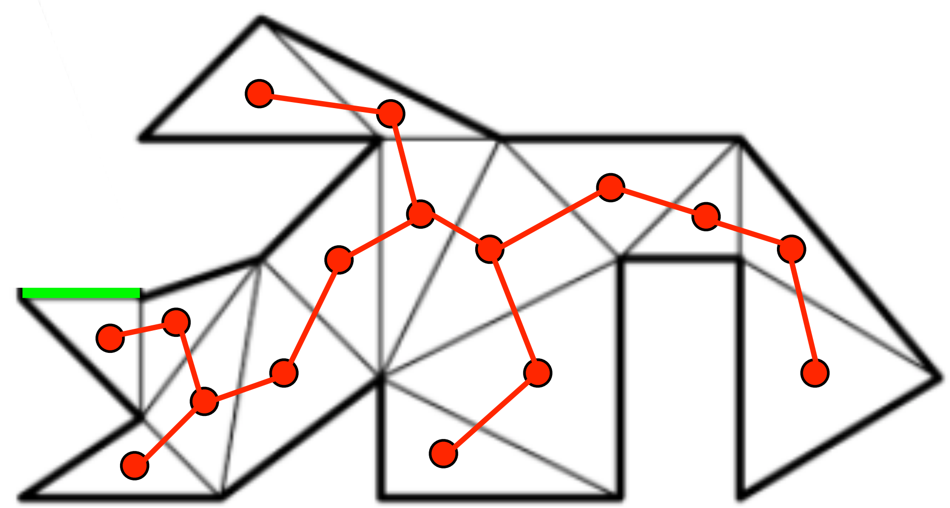
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity. $O(1)$
- Update ears. $O(n)$



Total:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

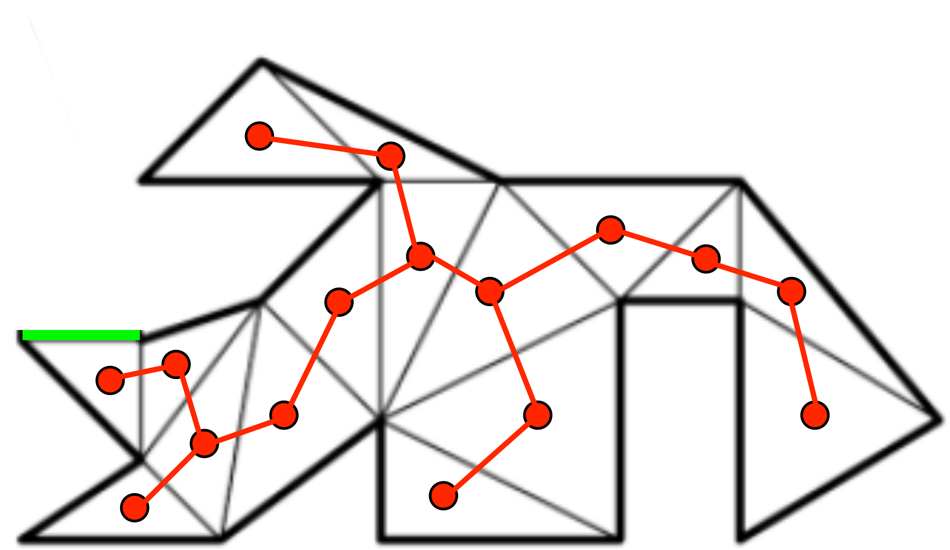
Algorithm 5.11

Initialize: (once)

- Find all convex vertices. $O(n)$
- Identify all ears. $O(n^2)$

Update: (n times)

- Remove ear. $O(1)$
- Update convexity. $O(1)$
- Update ears. $O(n)$



Total: $O(n^2)$

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

$O(n^4)$

Input:

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output:

$O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

$O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

$O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.

$$O(n^4)$$

Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Testing a chord:

$$O(n^4)$$

Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Testing a chord:
- Checking all chords:

$$O(n^4)$$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Testing a chord: $O(n)$
- Checking all chords:

$O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Testing a chord: $O(n)$
- Checking all chords:

Total: $O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Testing a chord: $O(n)$
- Checking all chords: $O(n^3)$

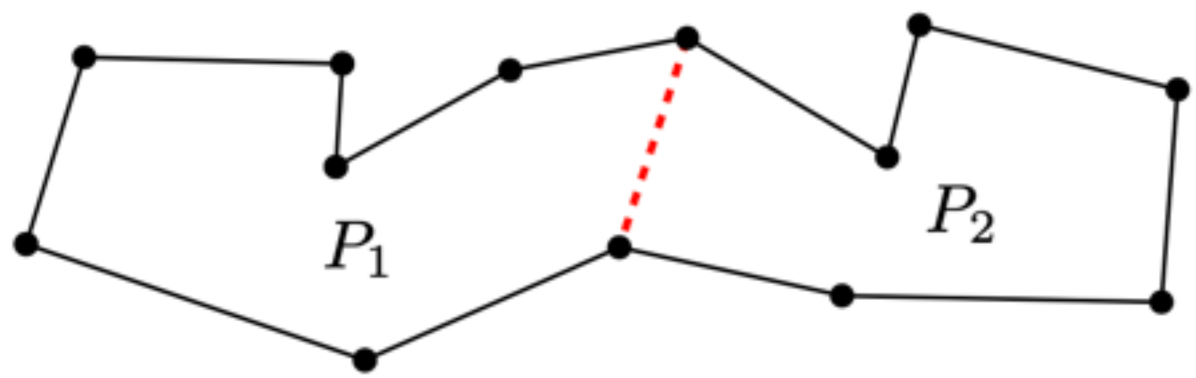
Total: $O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Idea:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.



Runtime:

- Testing a chord: $O(n)$
- Checking all chords: $O(n^3)$

Total: $O(n^4)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Consider a convex edge:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

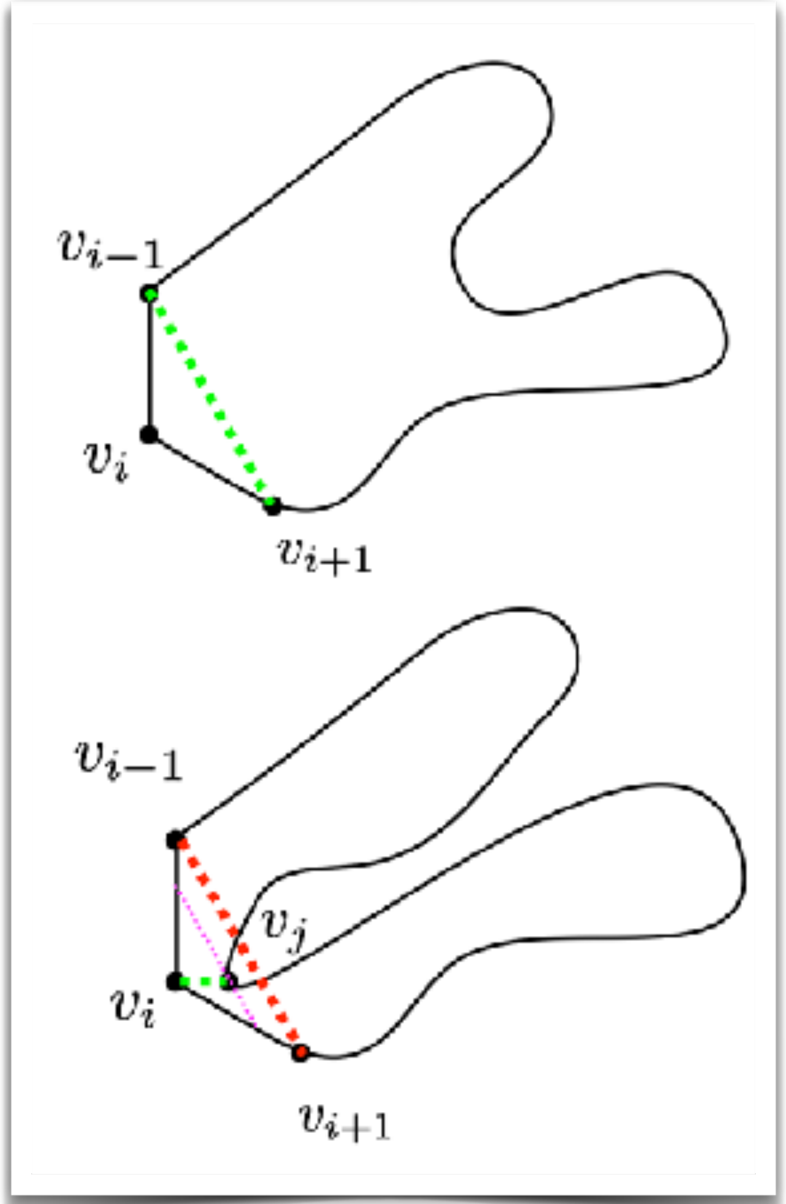
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Consider a convex edge:

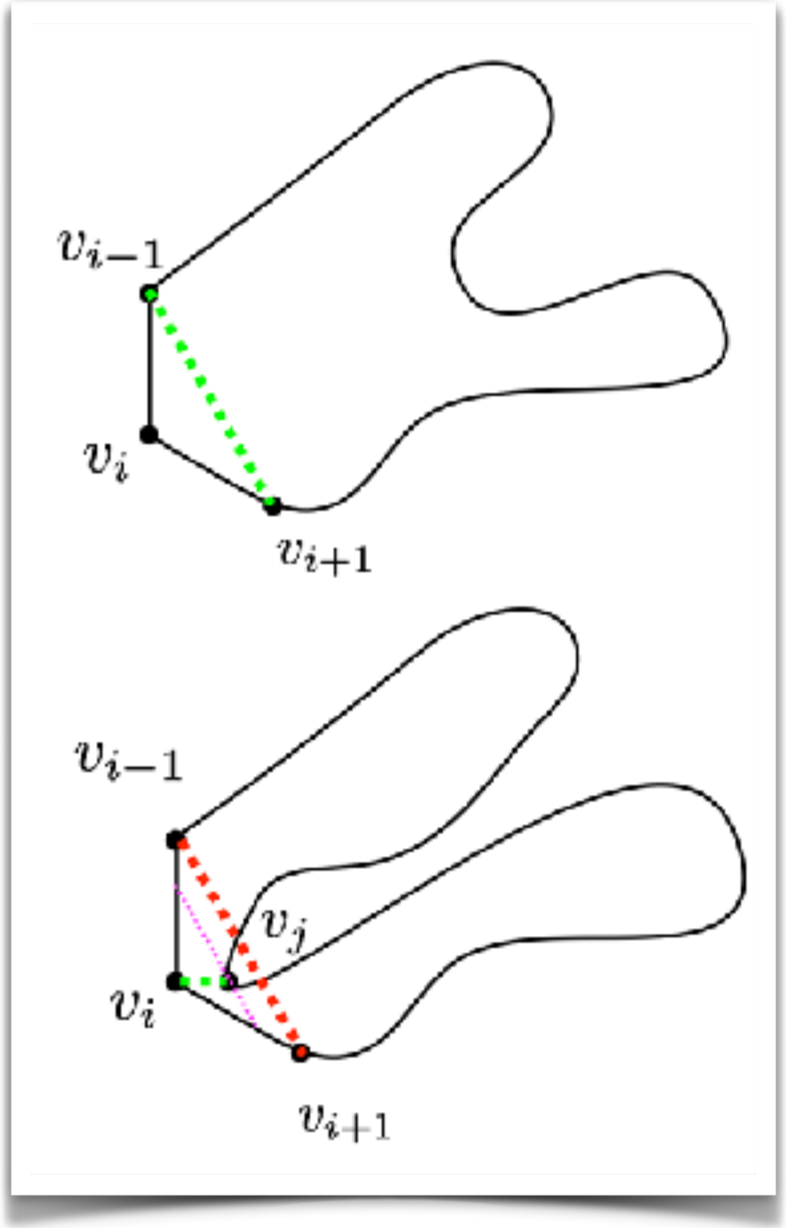


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.



Runtime:

- Consider a convex edge: $O(1)$

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

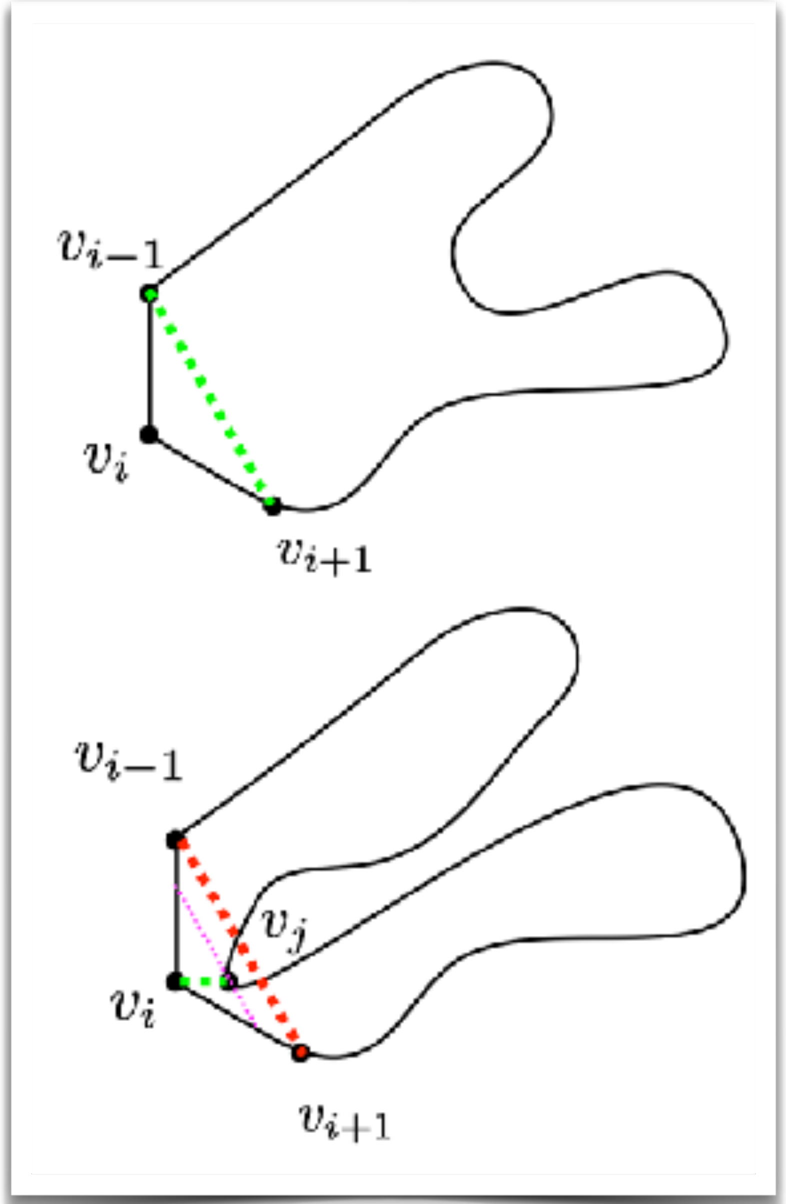
Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.

Runtime:

- Consider a convex edge: $O(1)$
- Check connection:

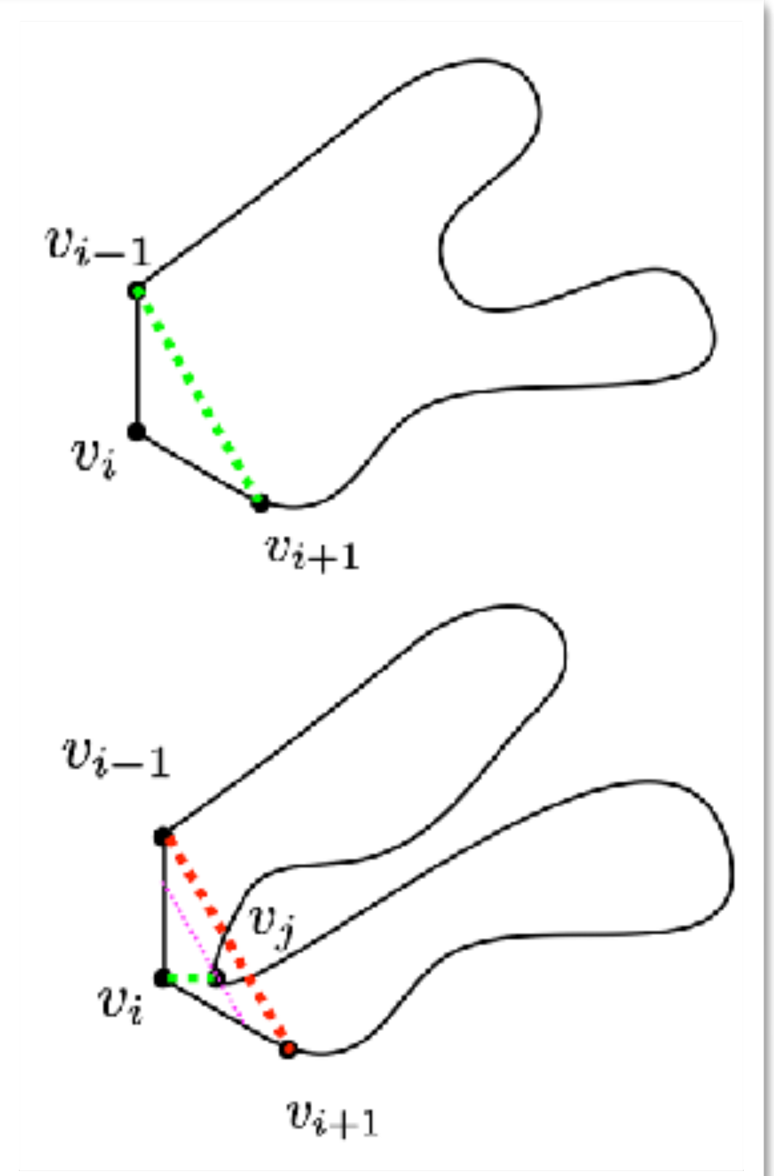


Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, v_iv_j, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.



Runtime:

- Consider a convex edge: $O(1)$
- Check connection: $O(n)$

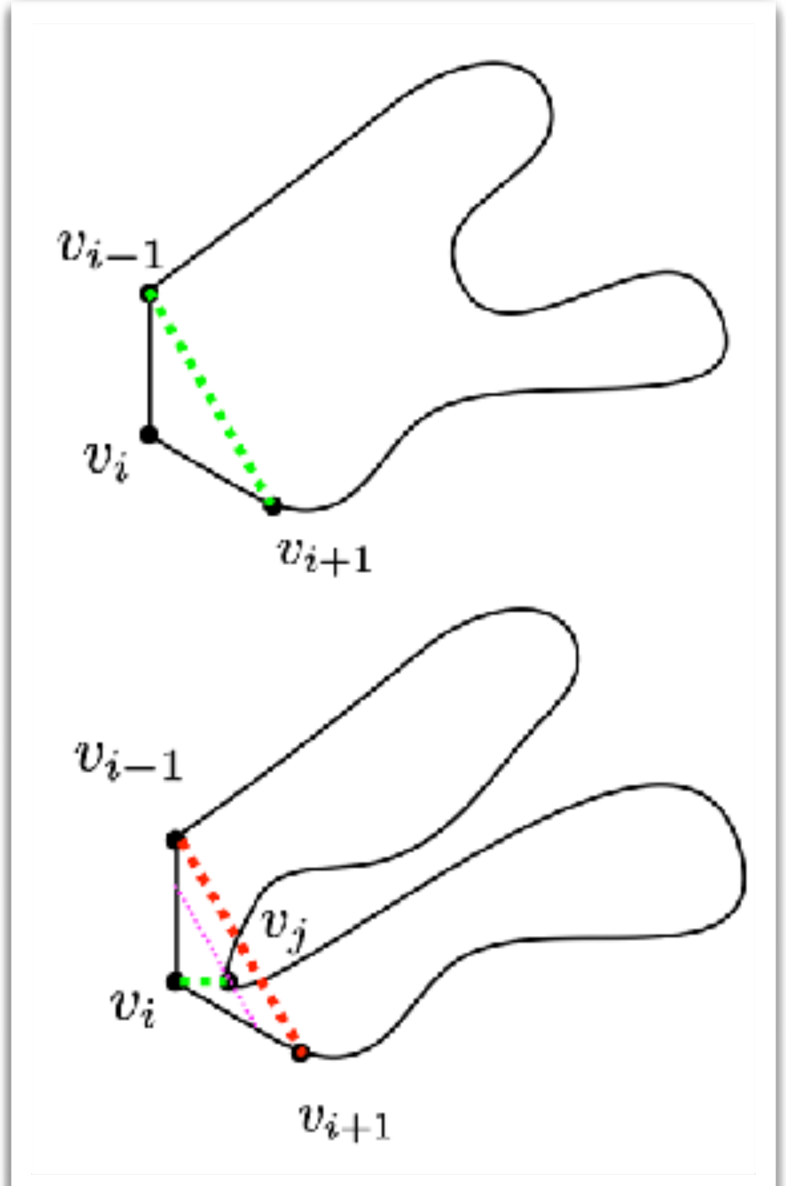
Technische Universität Braunschweig

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.



Runtime:

- Consider a convex edge: $O(1)$
- Check connection: $O(n)$

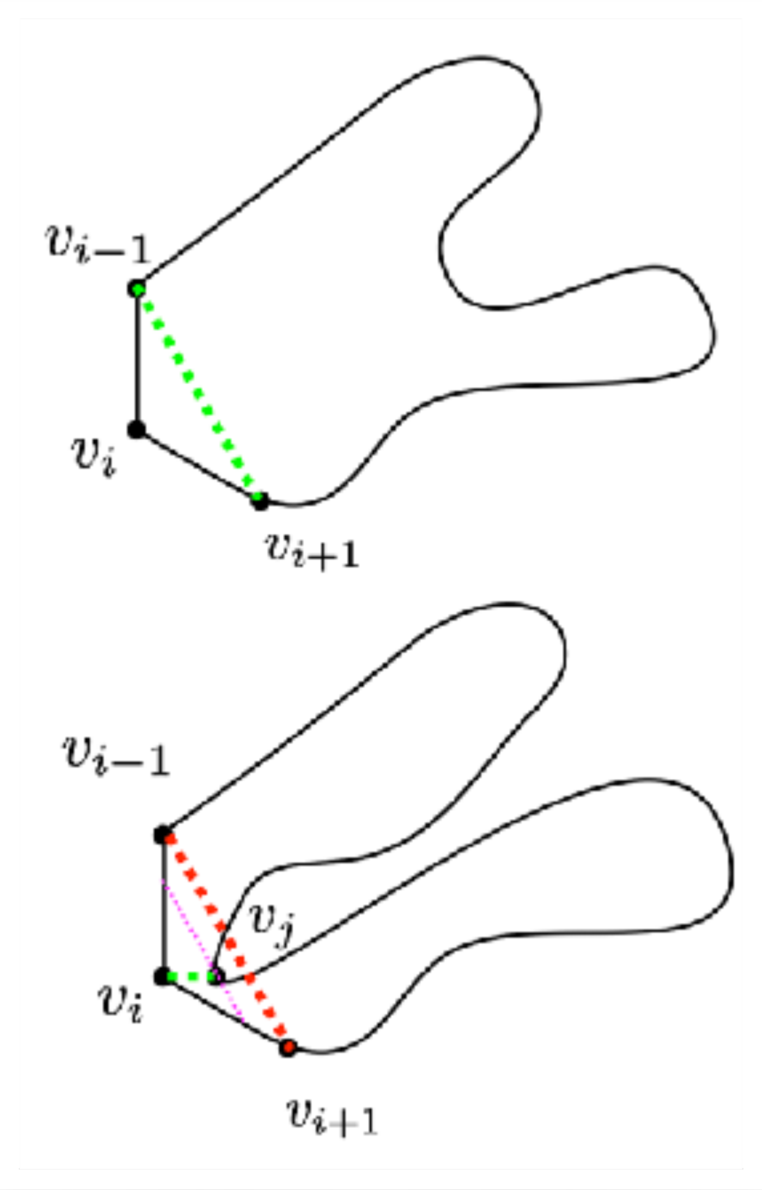
Total:

Input: $v_1, \ldots, v_n$, sorted list of the vertices of a simple polygon P.

Output: List of internal diagonals of P, $v_i v_j$, determining a triangulation of P.

Algorithm 5.12:

- Find a diagonal.
- Insert it.
- Recurse over subpolygons.



Runtime:

- Consider a convex edge: $O(1)$
- Check connection: $O(n)$

Total: $O(n^2)$

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

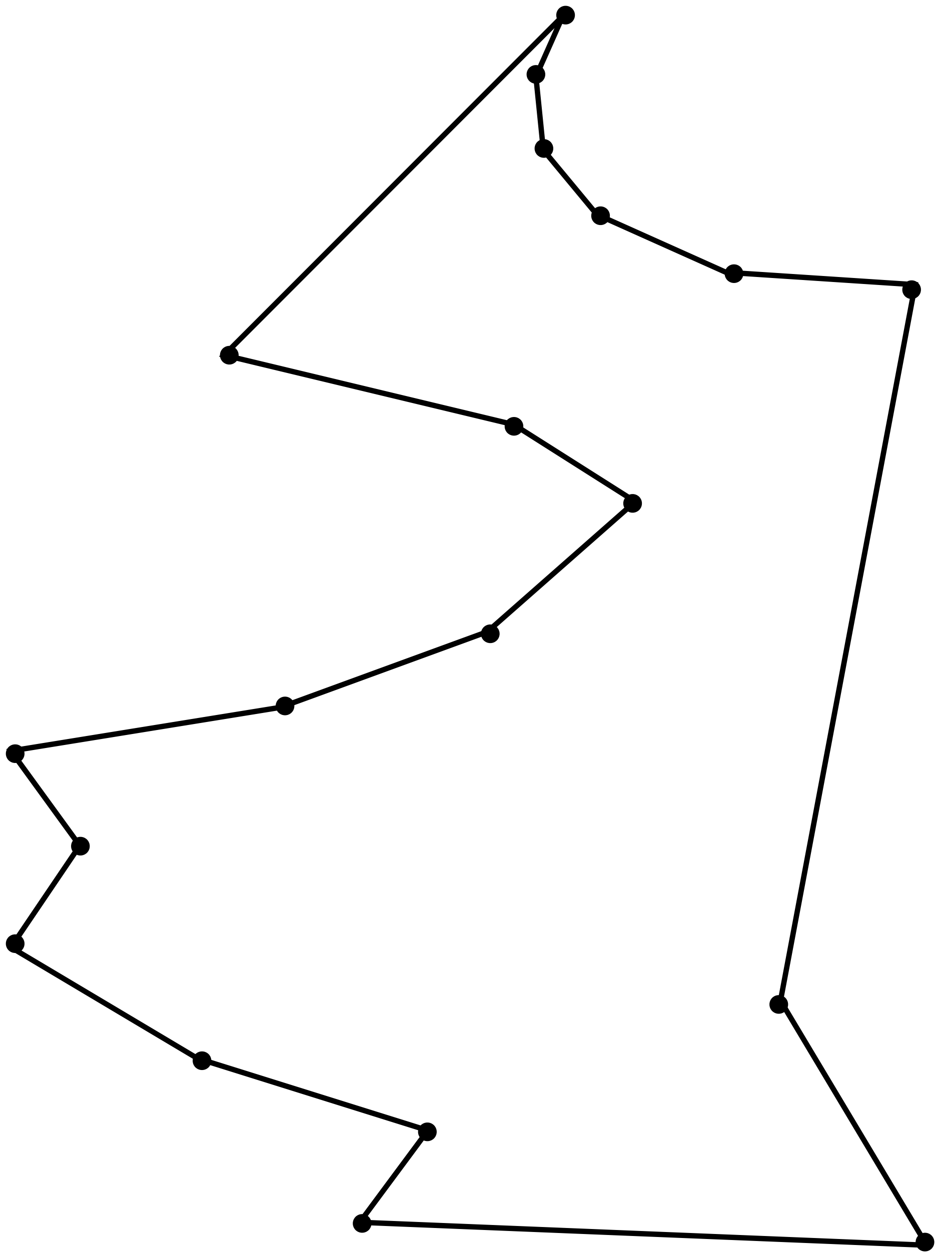
Definition 5.13

Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

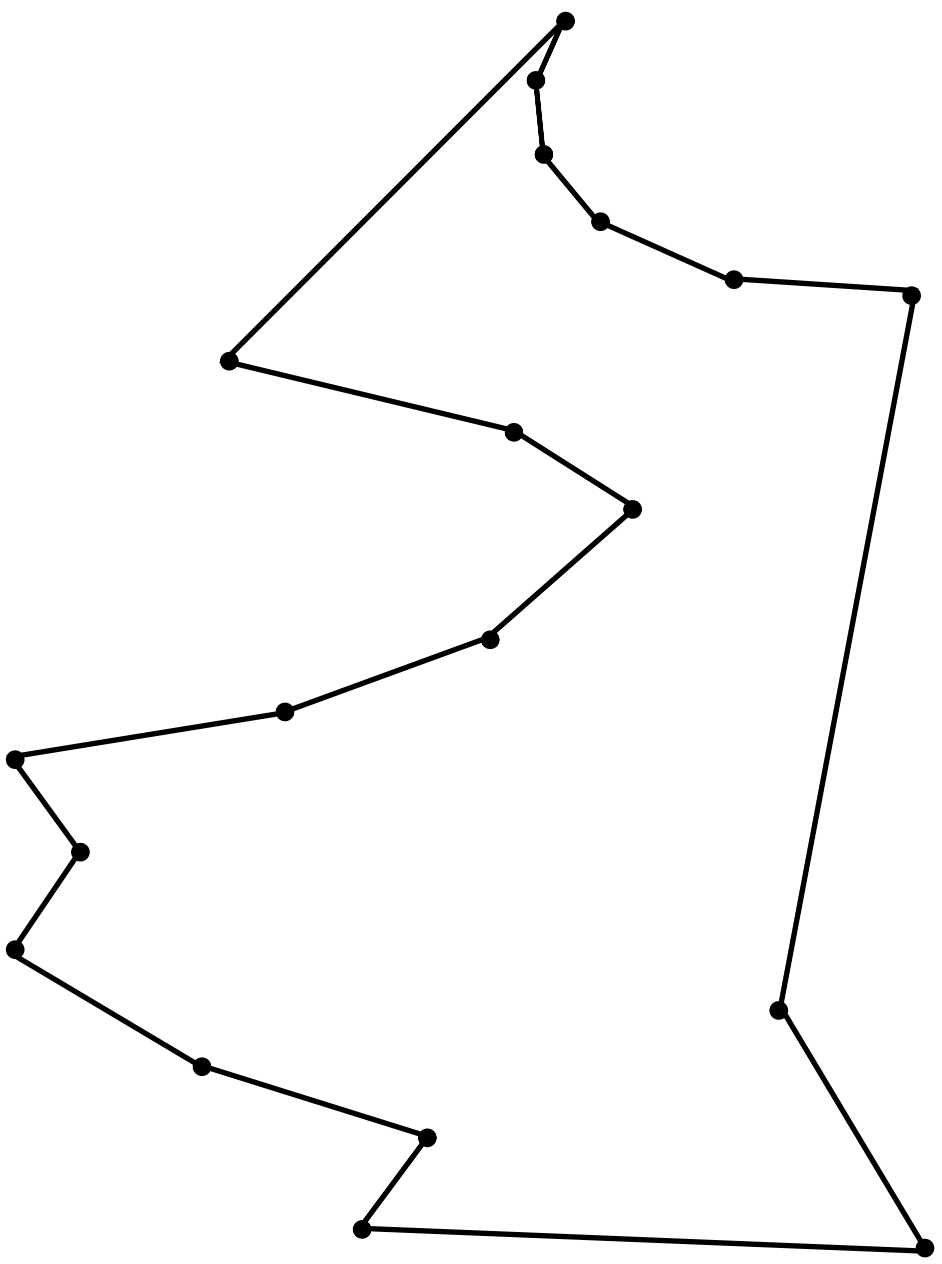
Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.



Definition 5.13

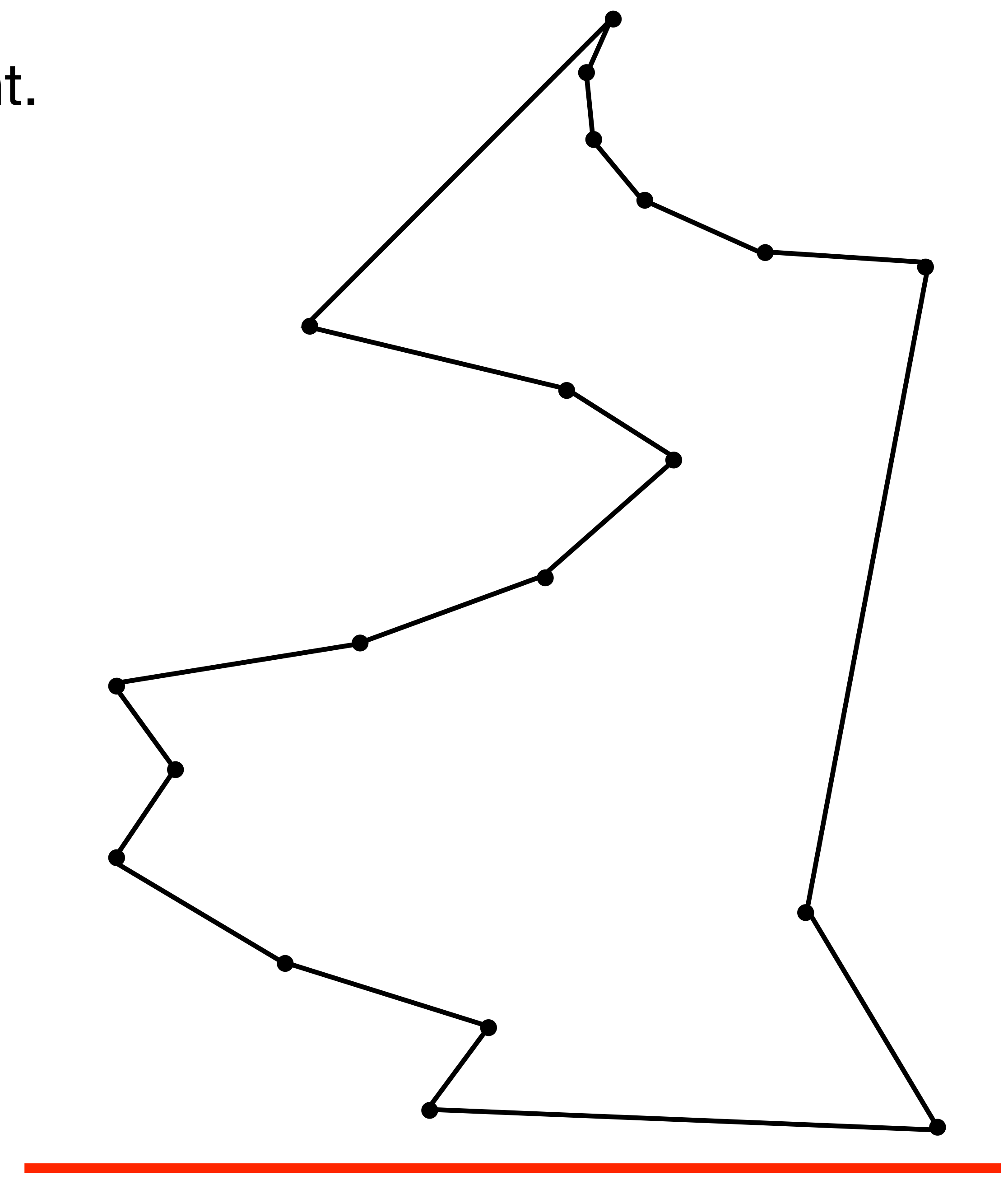
A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.



Technische Universität Braunschweig

Definition 5.13

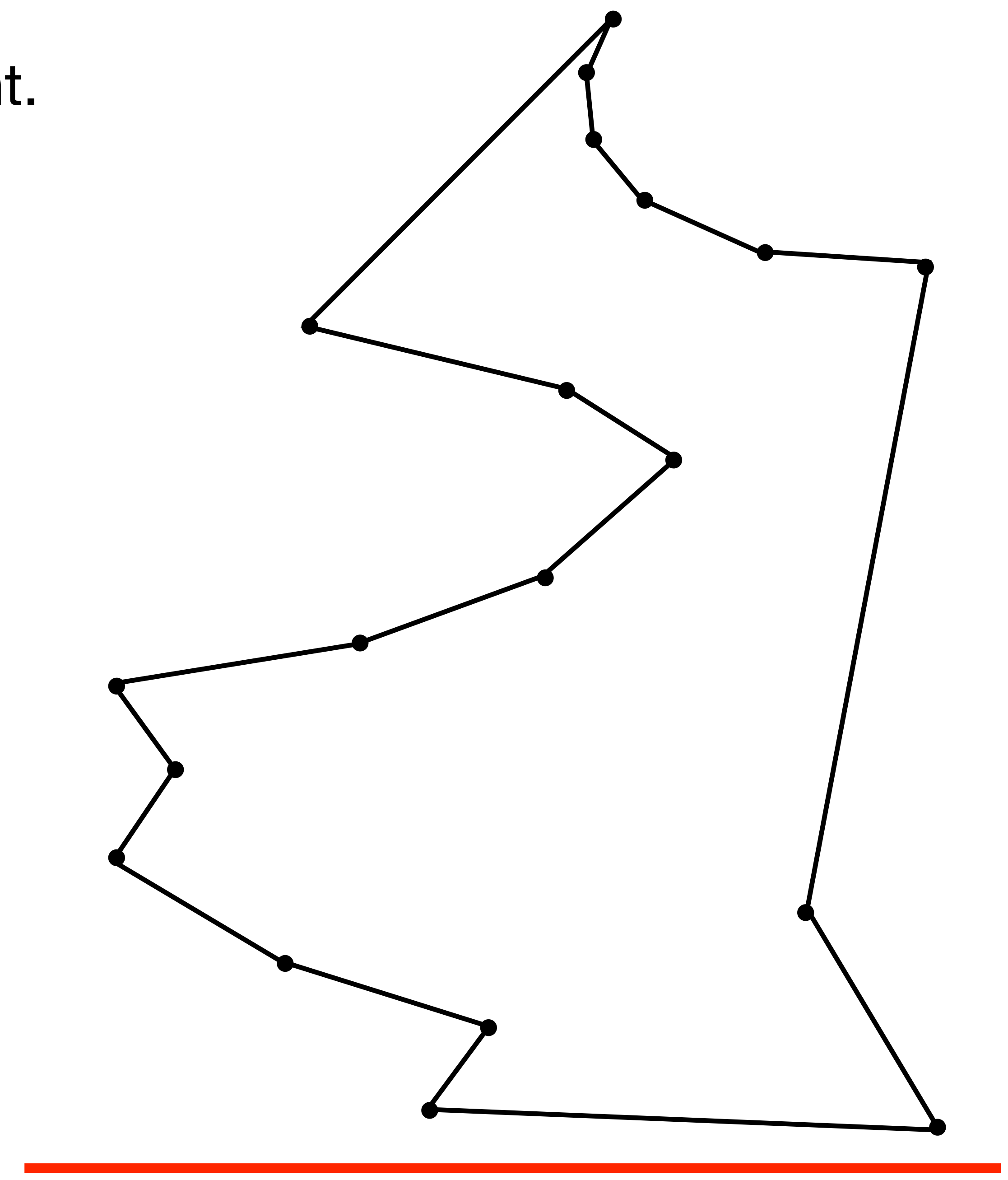
A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.



Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

Observation 5.14

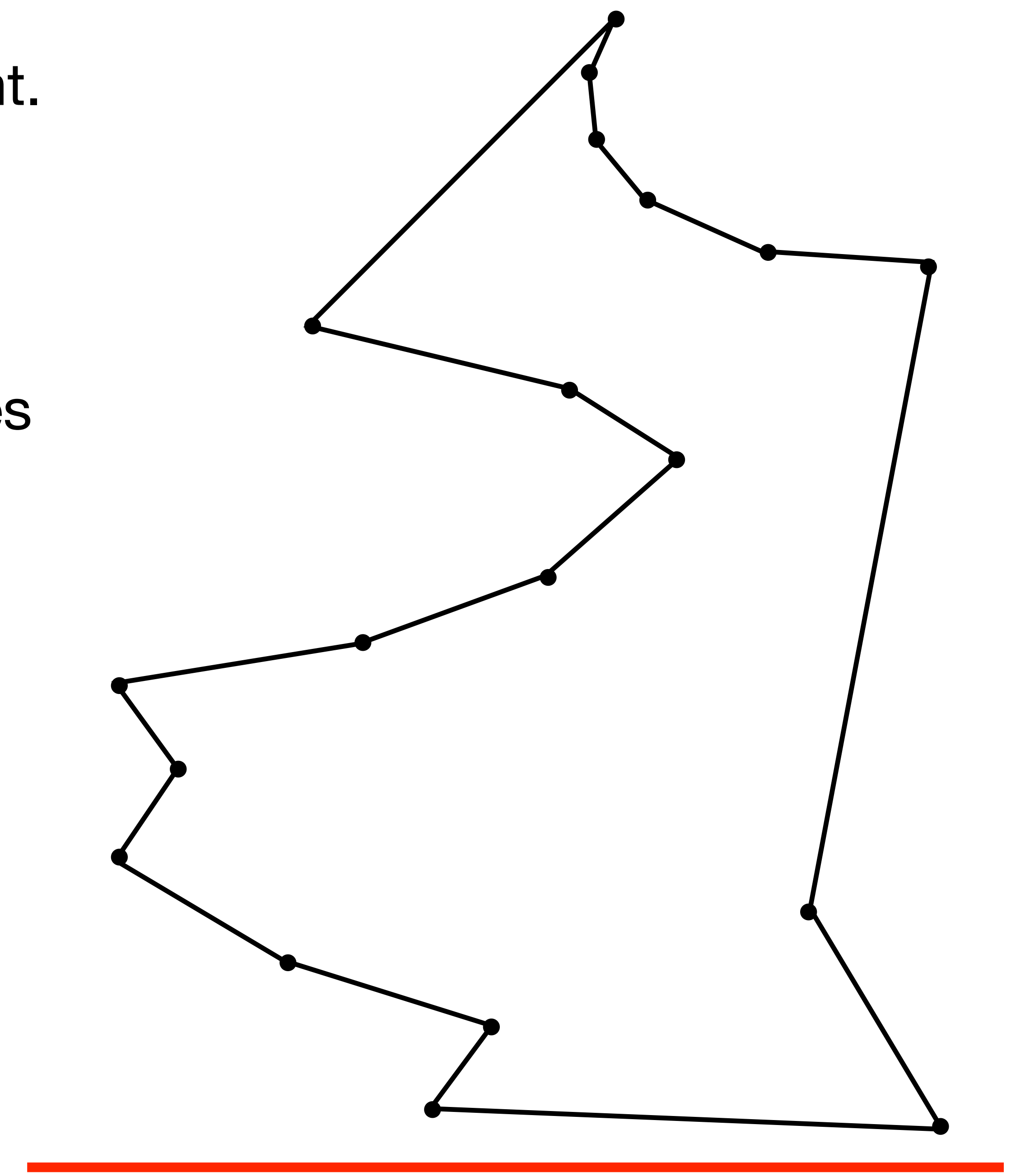


Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

Observation 5.14

A polygon P is monotone, if and only if it does not have a *cusp:* a reflex vertex v such that the vertices before and after are both above or both below v.

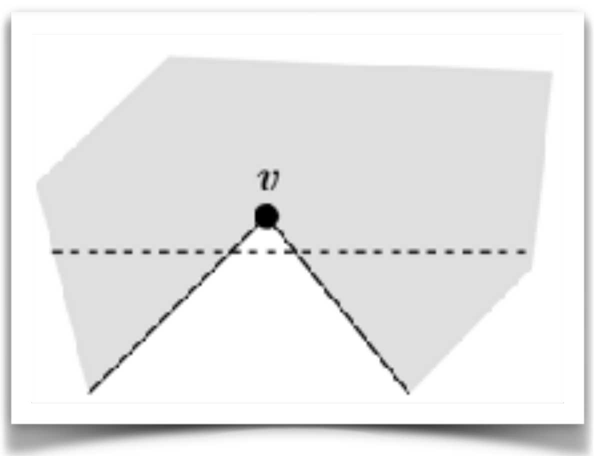


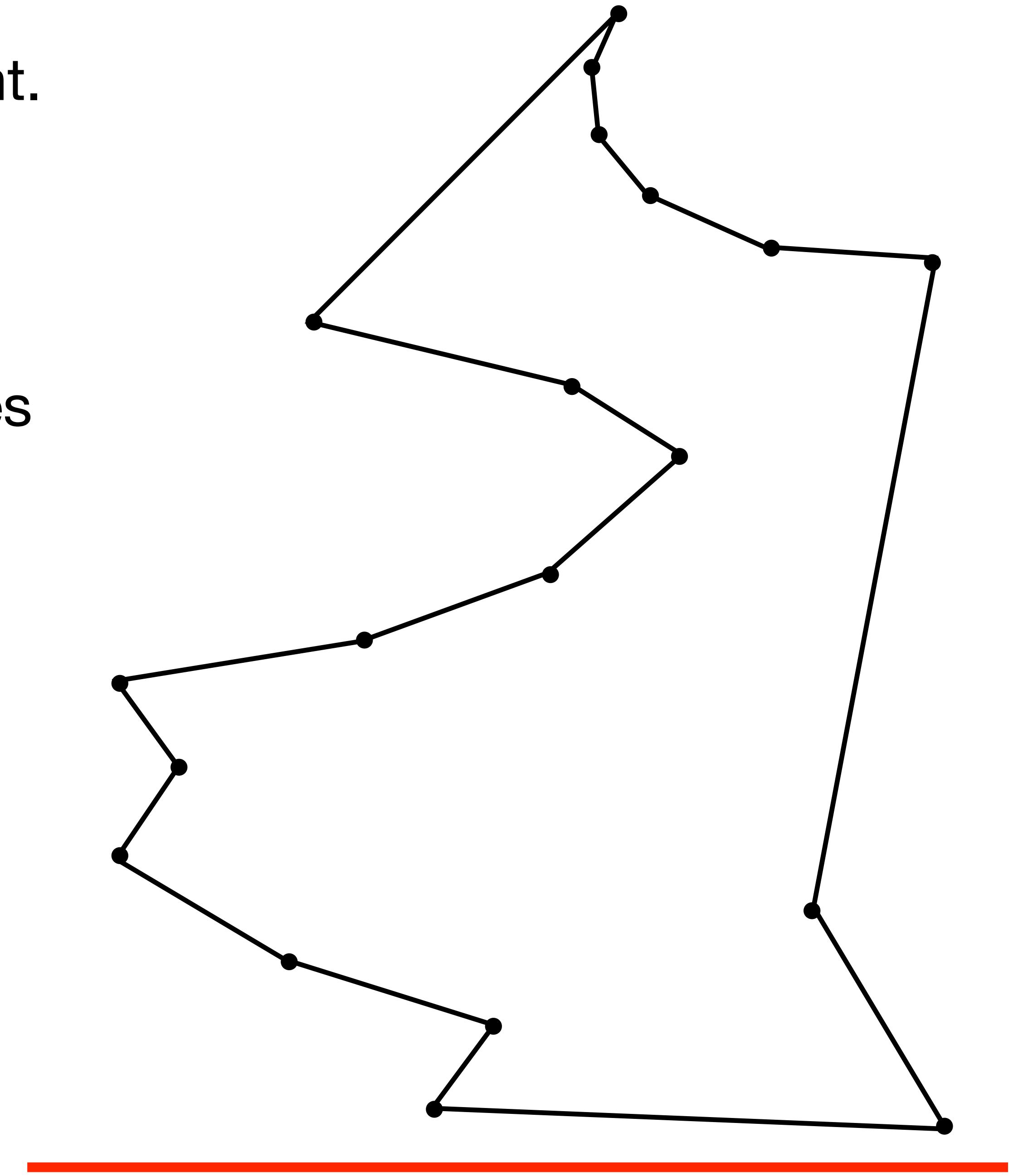
Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

Observation 5.14

A polygon P is monotone, if and only if it does not have a *cusp:* a reflex vertex v such that the vertices before and after are both above or both below v.



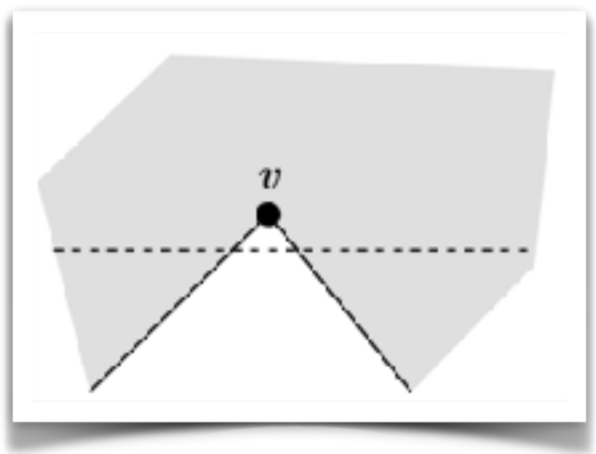


Definition 5.13

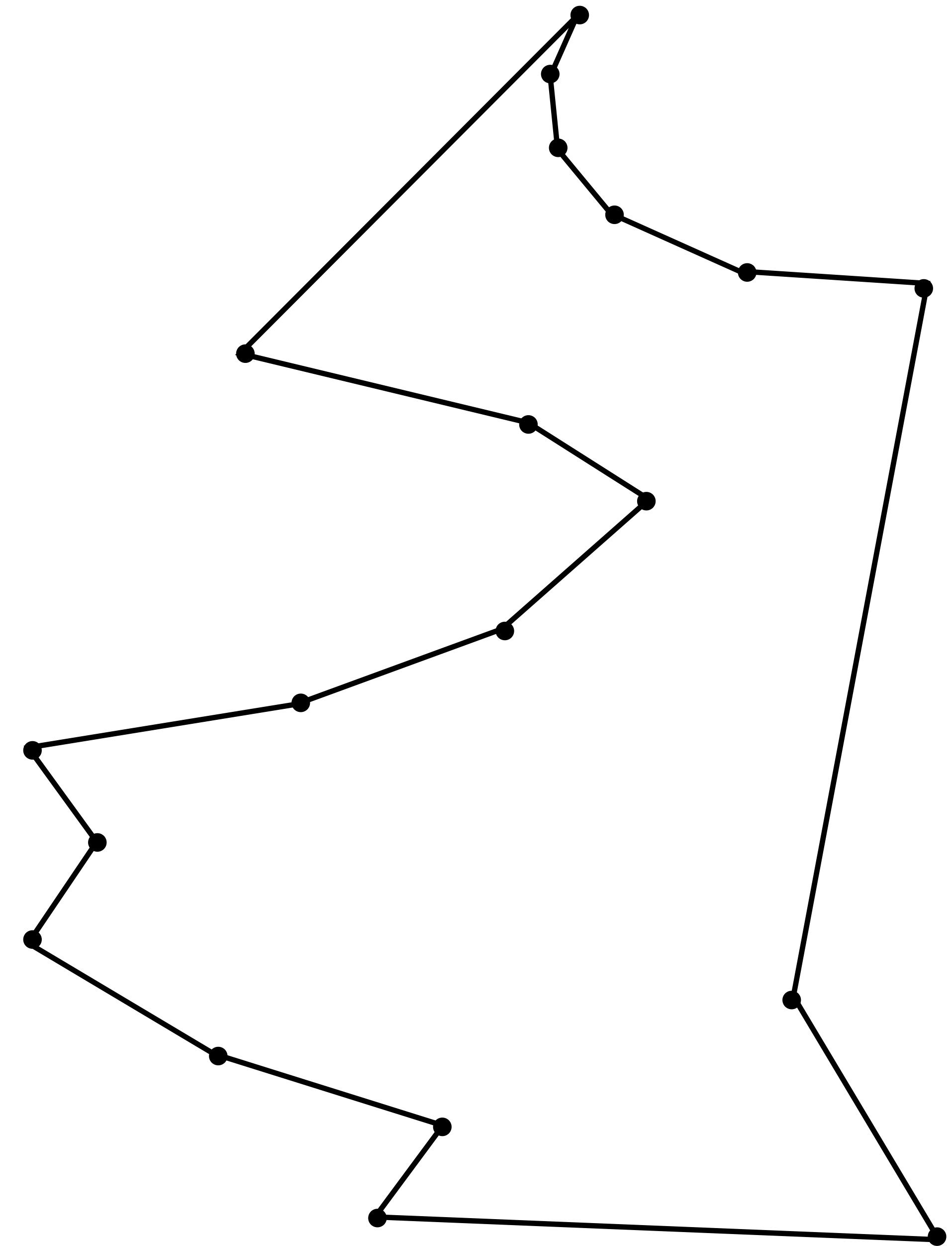
A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

Observation 5.14

A polygon P is monotone, if and only if it does not have a *cusp:* a reflex vertex v such that the vertices before and after are both above or both below v.



Corollary 5.15

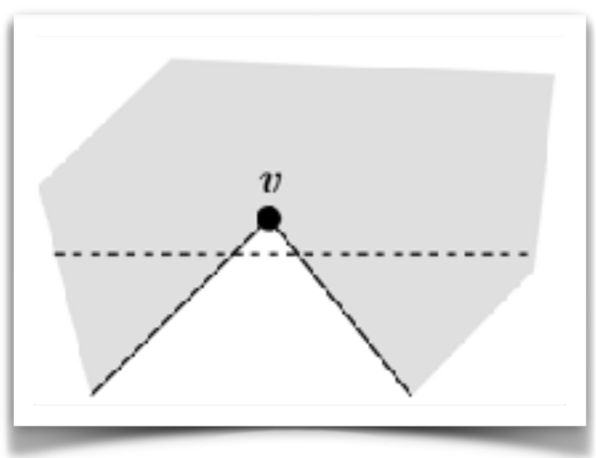


Definition 5.13

A polygon P is *monotone (in y-direction),* if every x-parallel line intersects it in a connected segment.

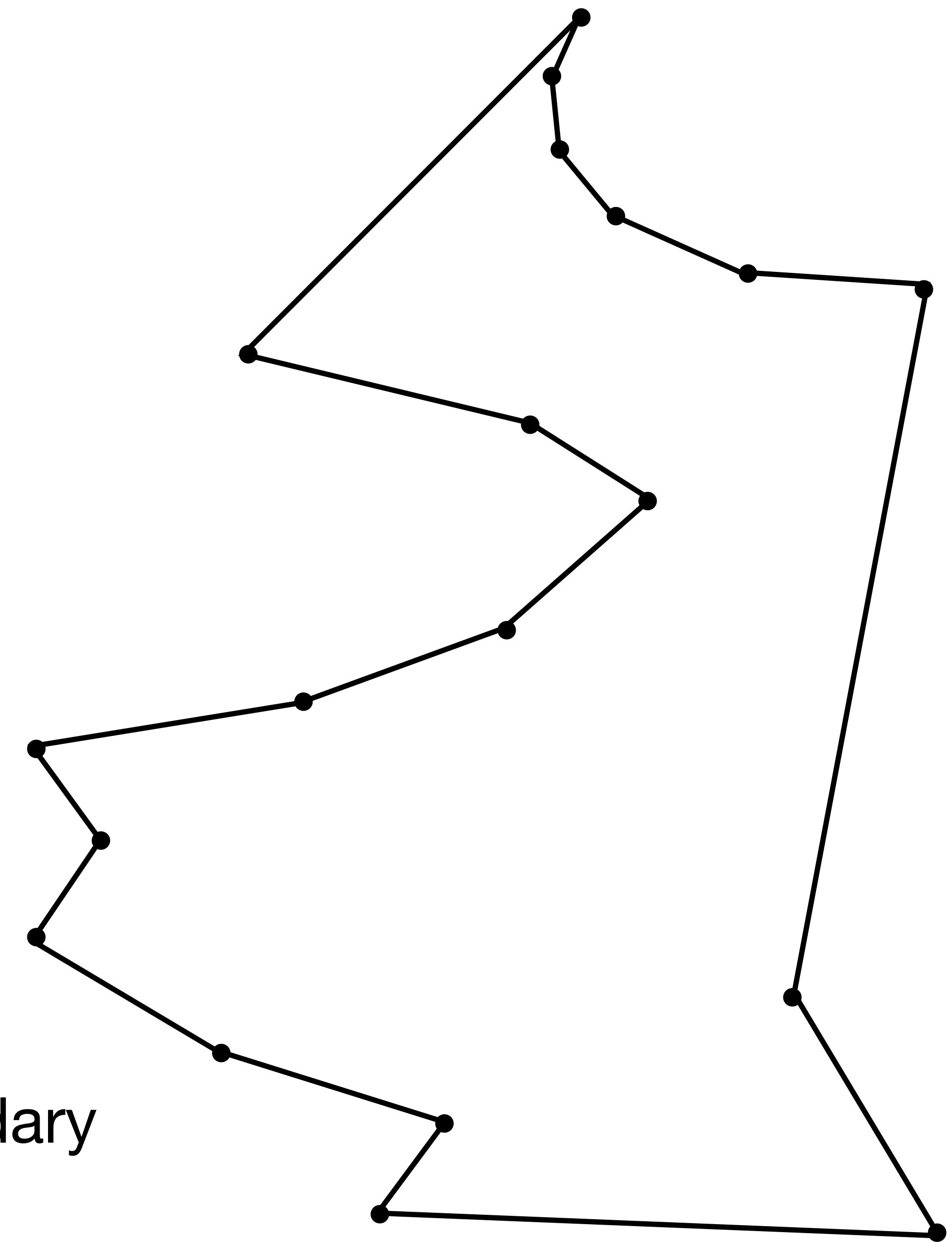
Observation 5.14

A polygon P is monotone, if and only if it does not have a *cusp:* a reflex vertex v such that the vertices before and after are both above or both below v.



Corollary 5.15

A polygon P is *monotone (in y-direction),* iff its boundary consists of two y-monotone chains.



Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

Technische Universität Braunschweig

Algorithm 5.16

Algorithm 5.16

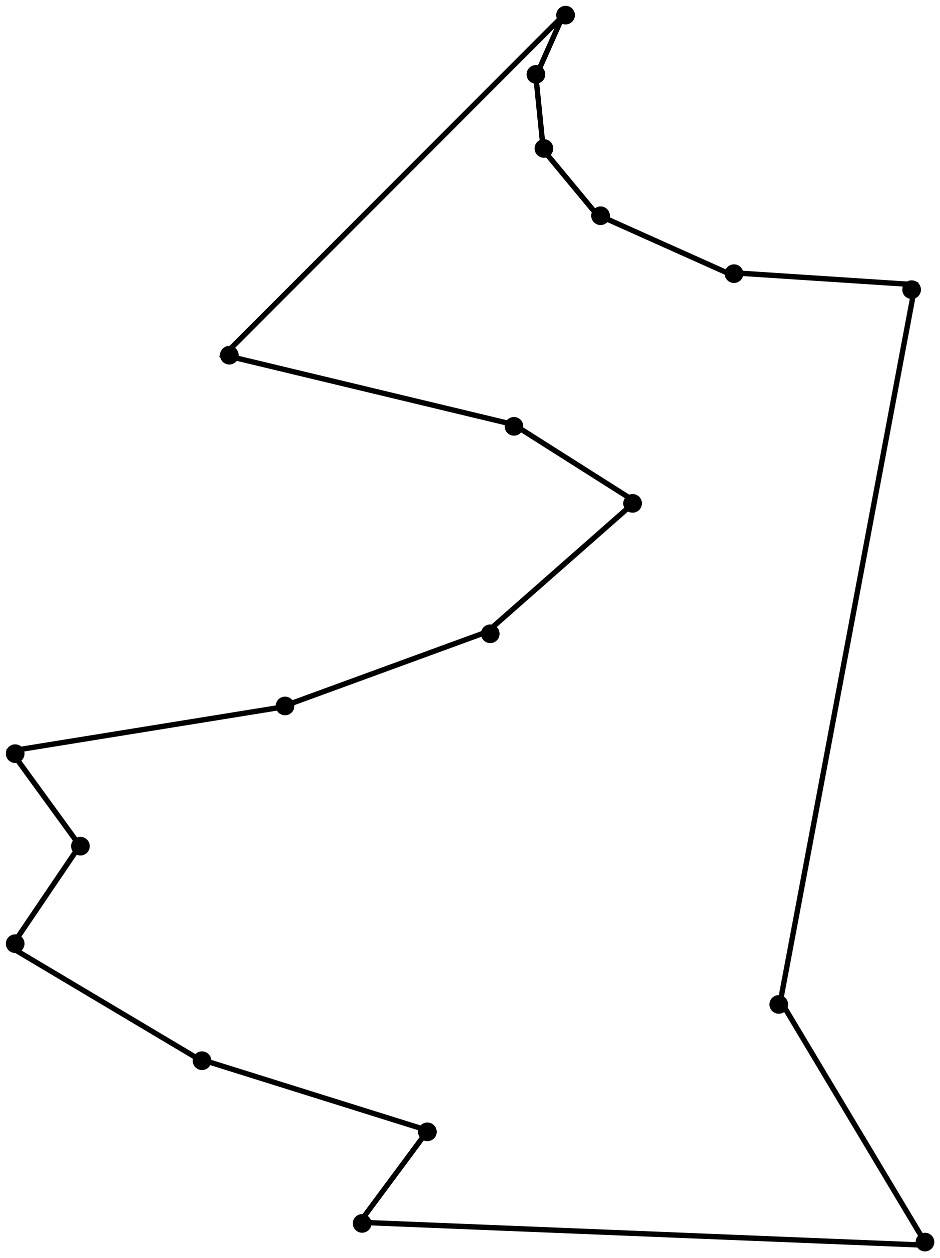
Input:

Algorithm 5.16

Input: A y-monotone polygon P

Algorithm 5.16

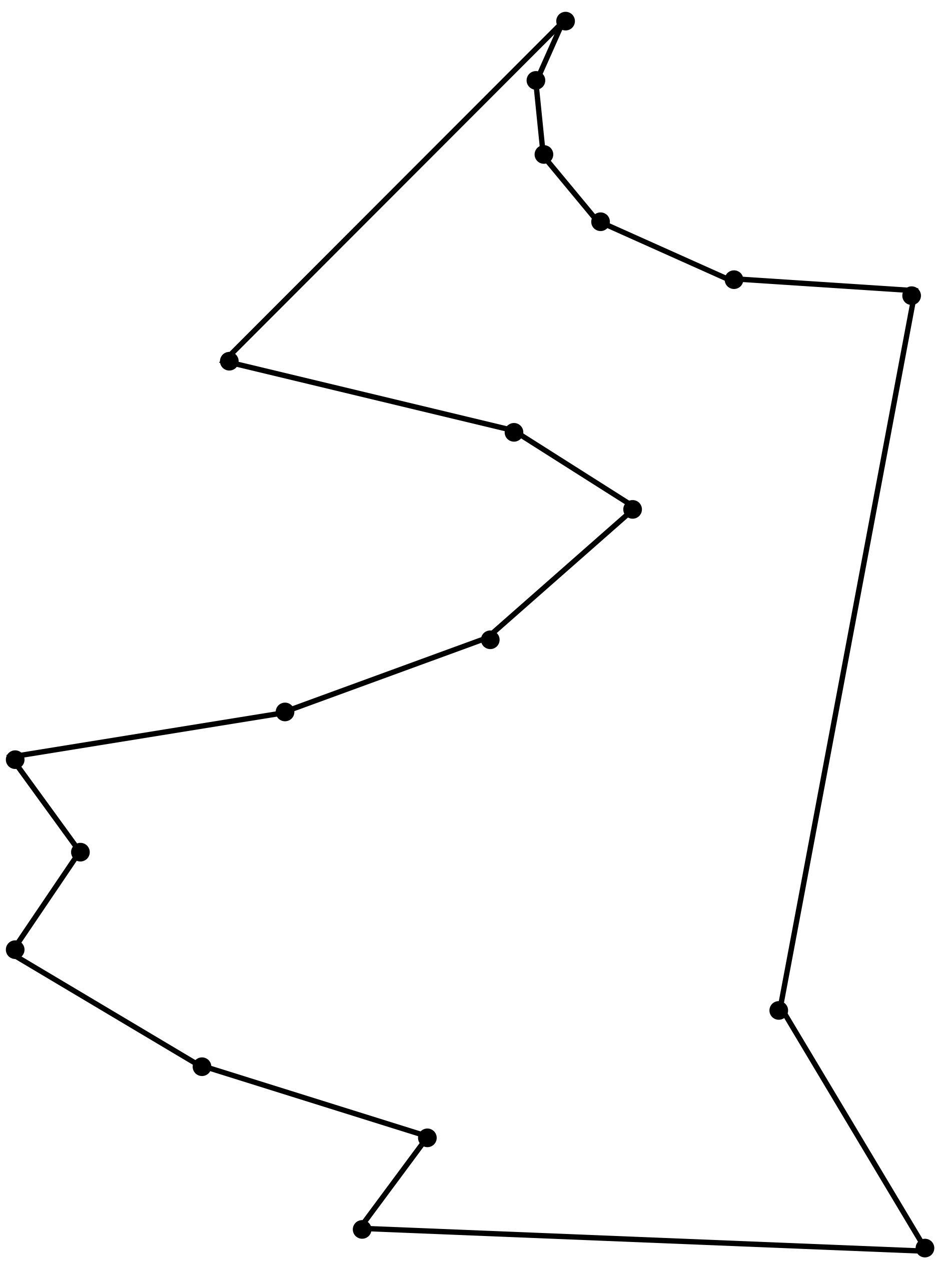
Input: A y-monotone polygon P



Algorithm 5.16

Input: A y-monotone polygon P

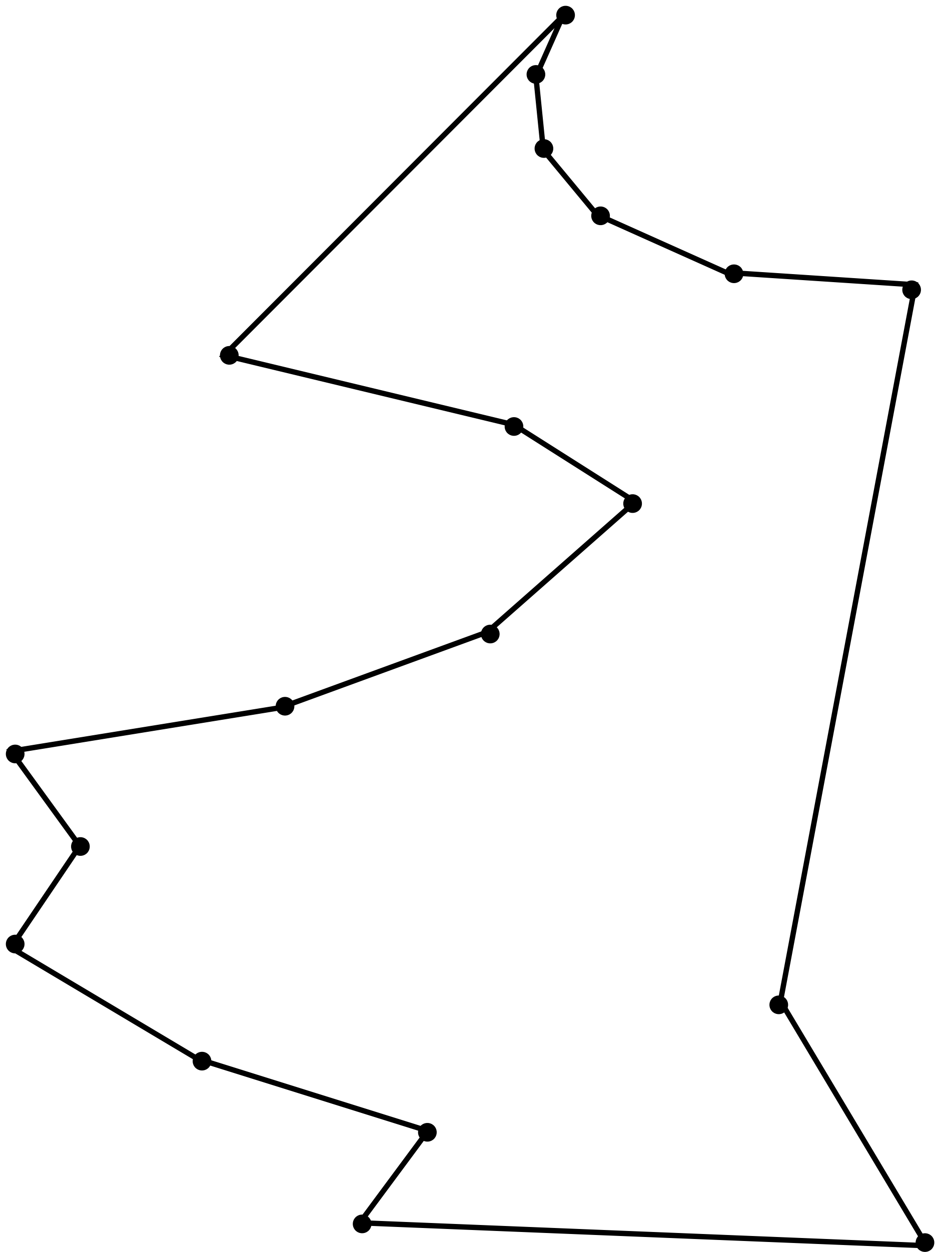
Output:



Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

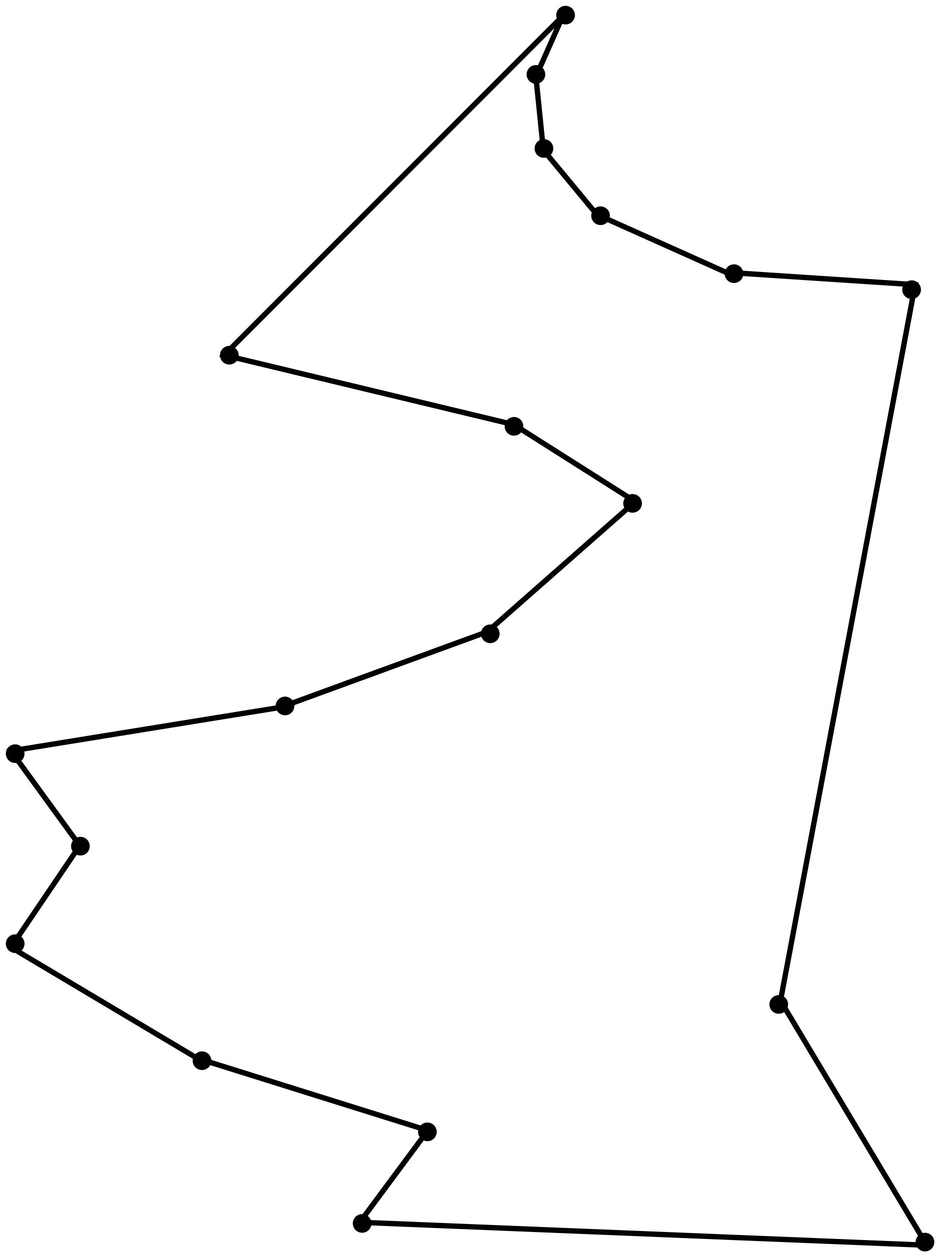


Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate

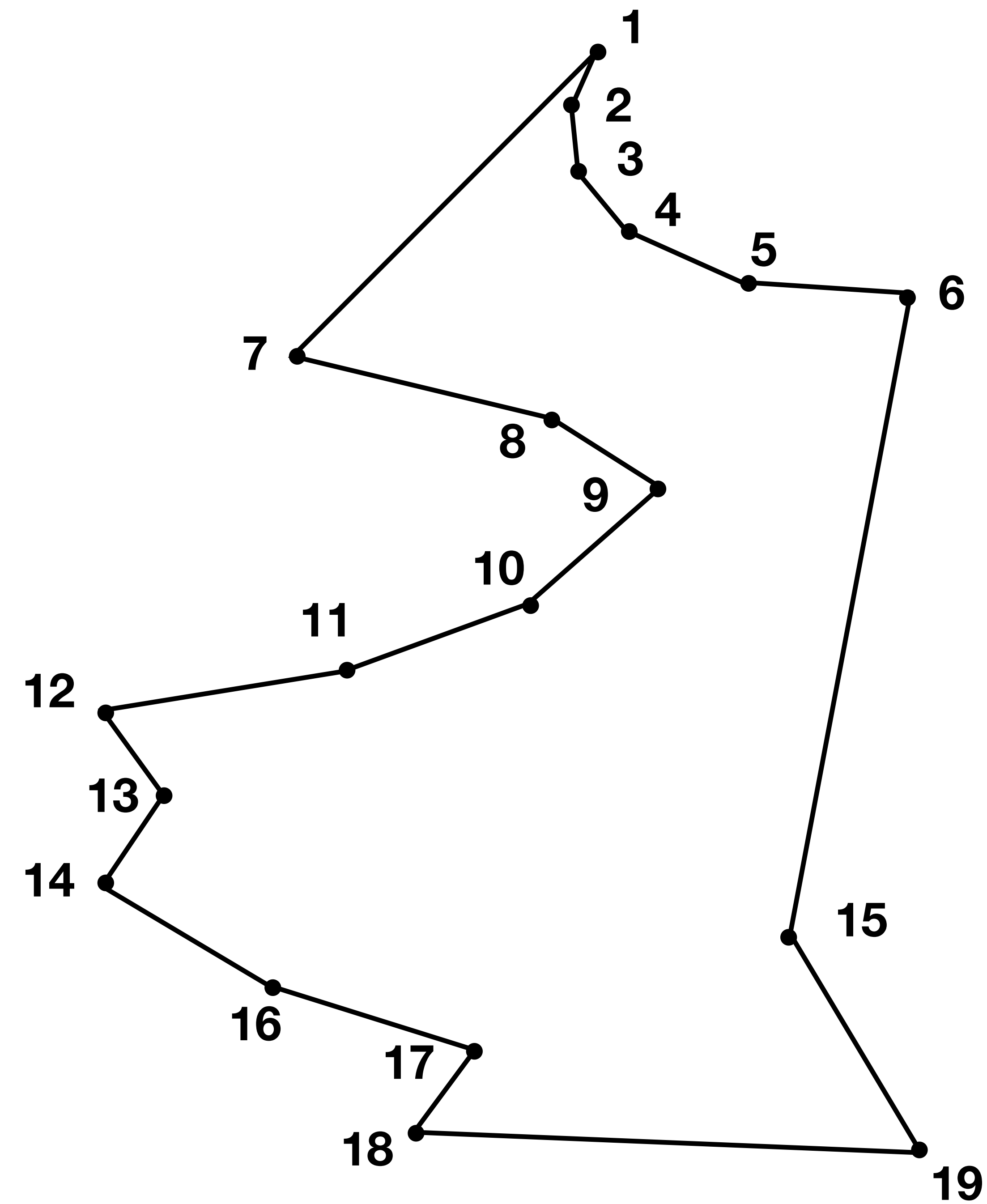


Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate

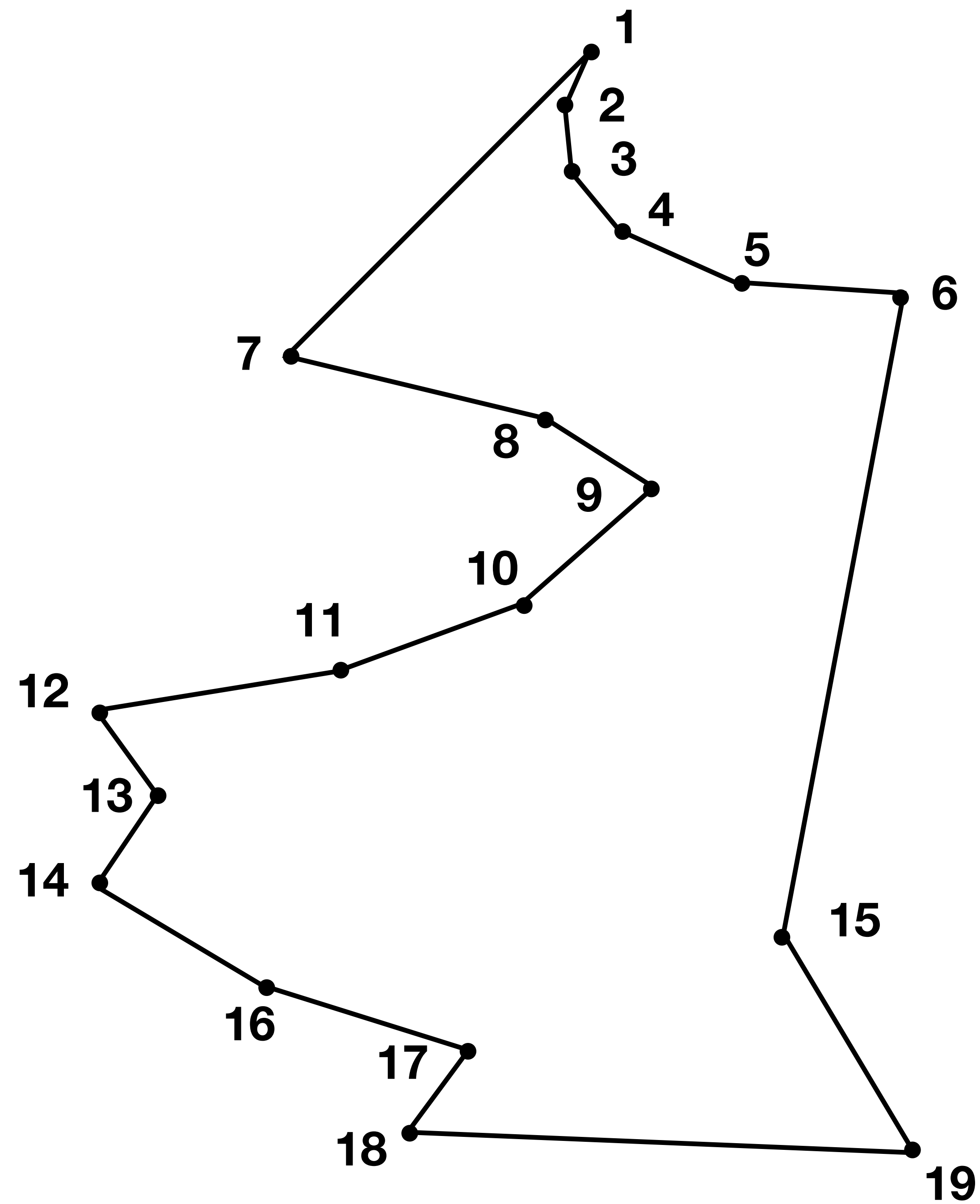


Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed

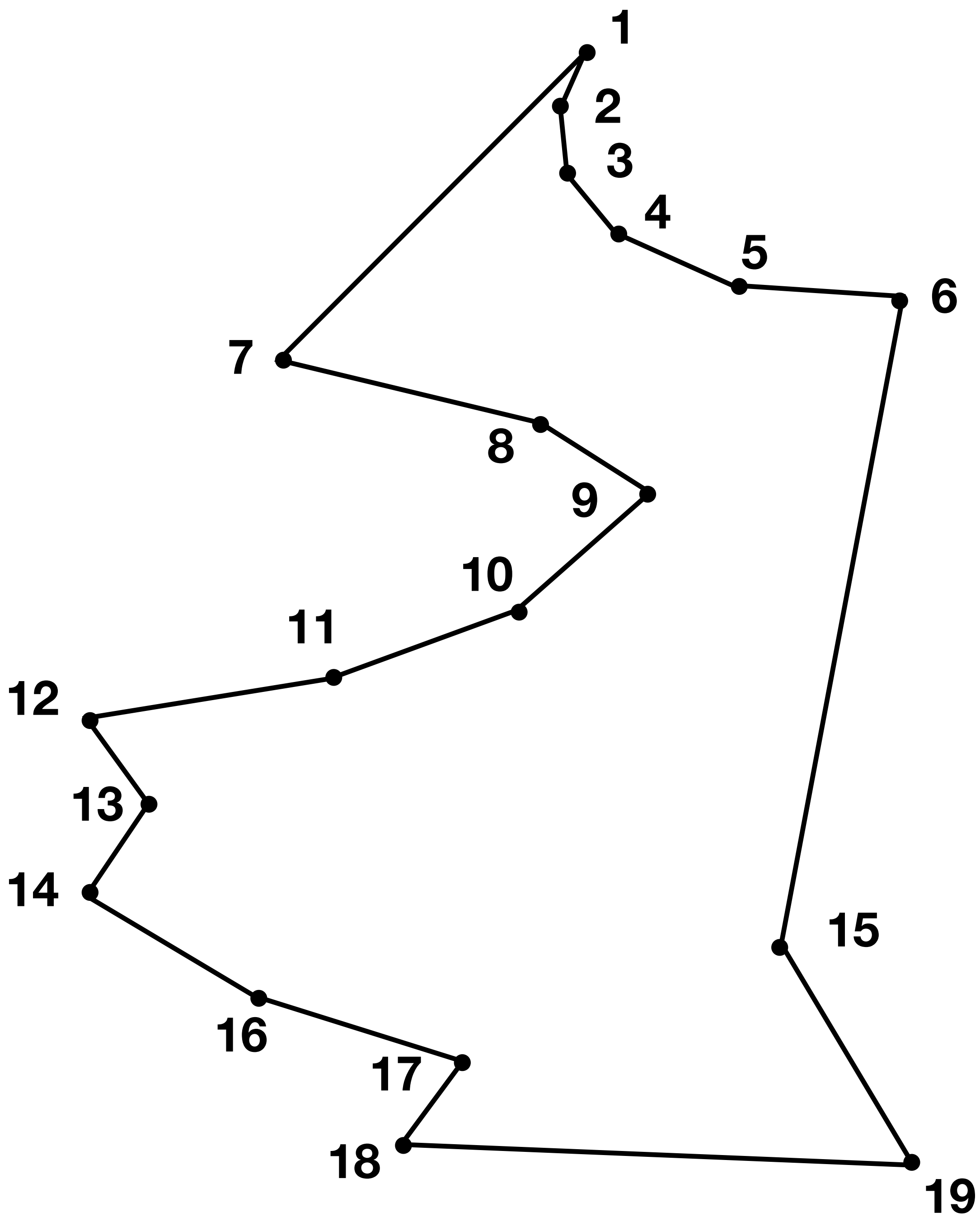


Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon



Technische Universität Braunschweig

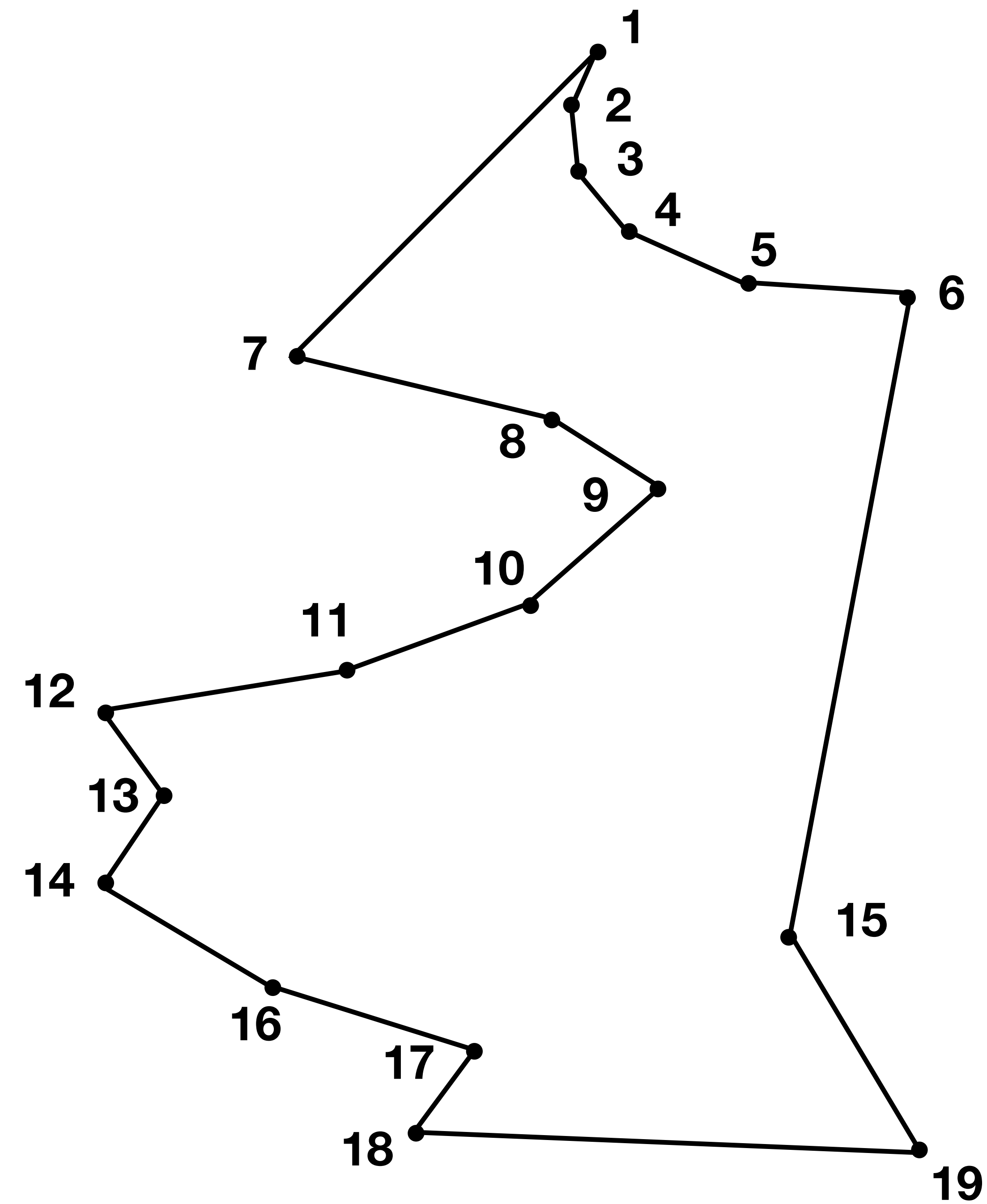
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon

Queue:



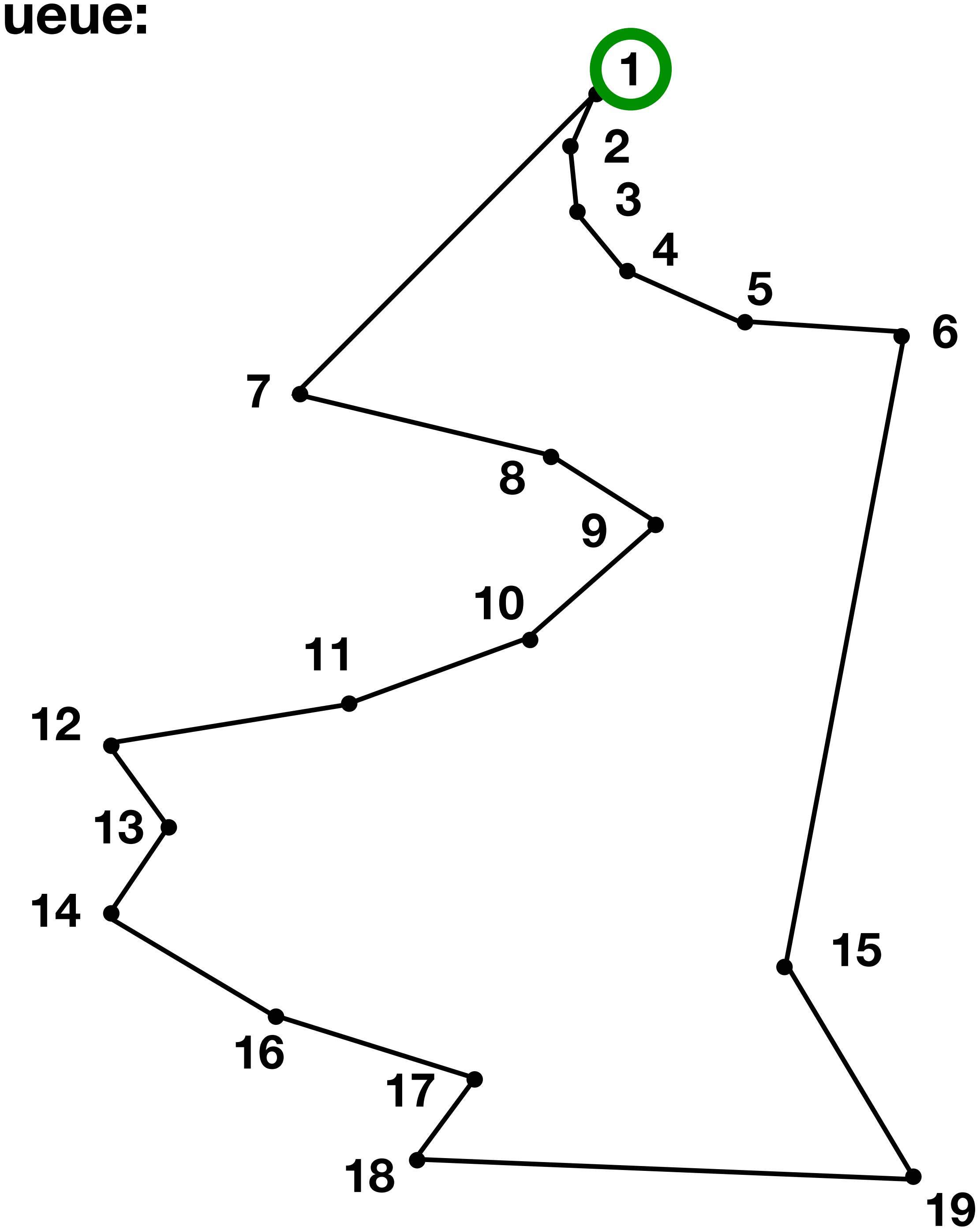
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon

Queue:



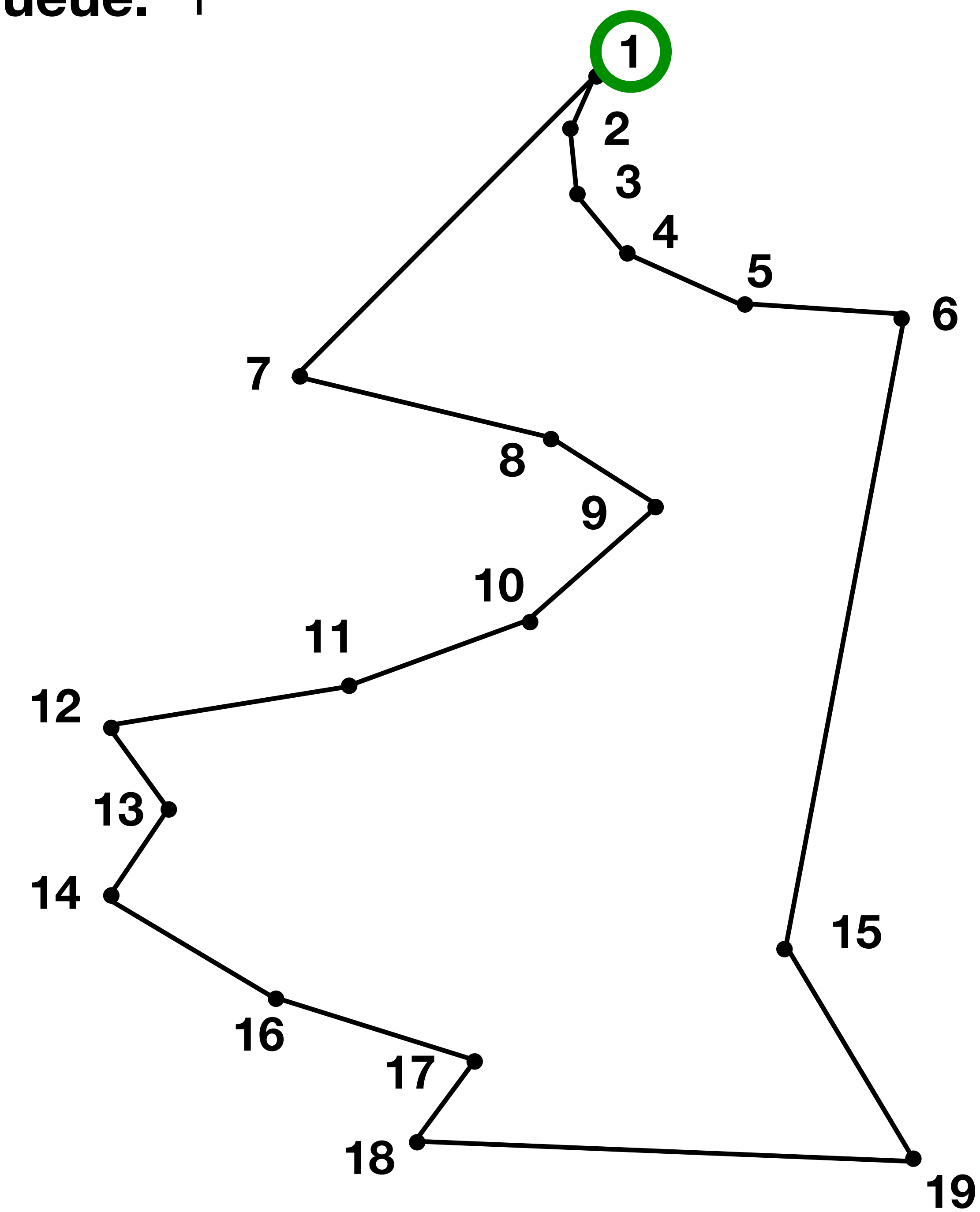
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon

Queue: 1



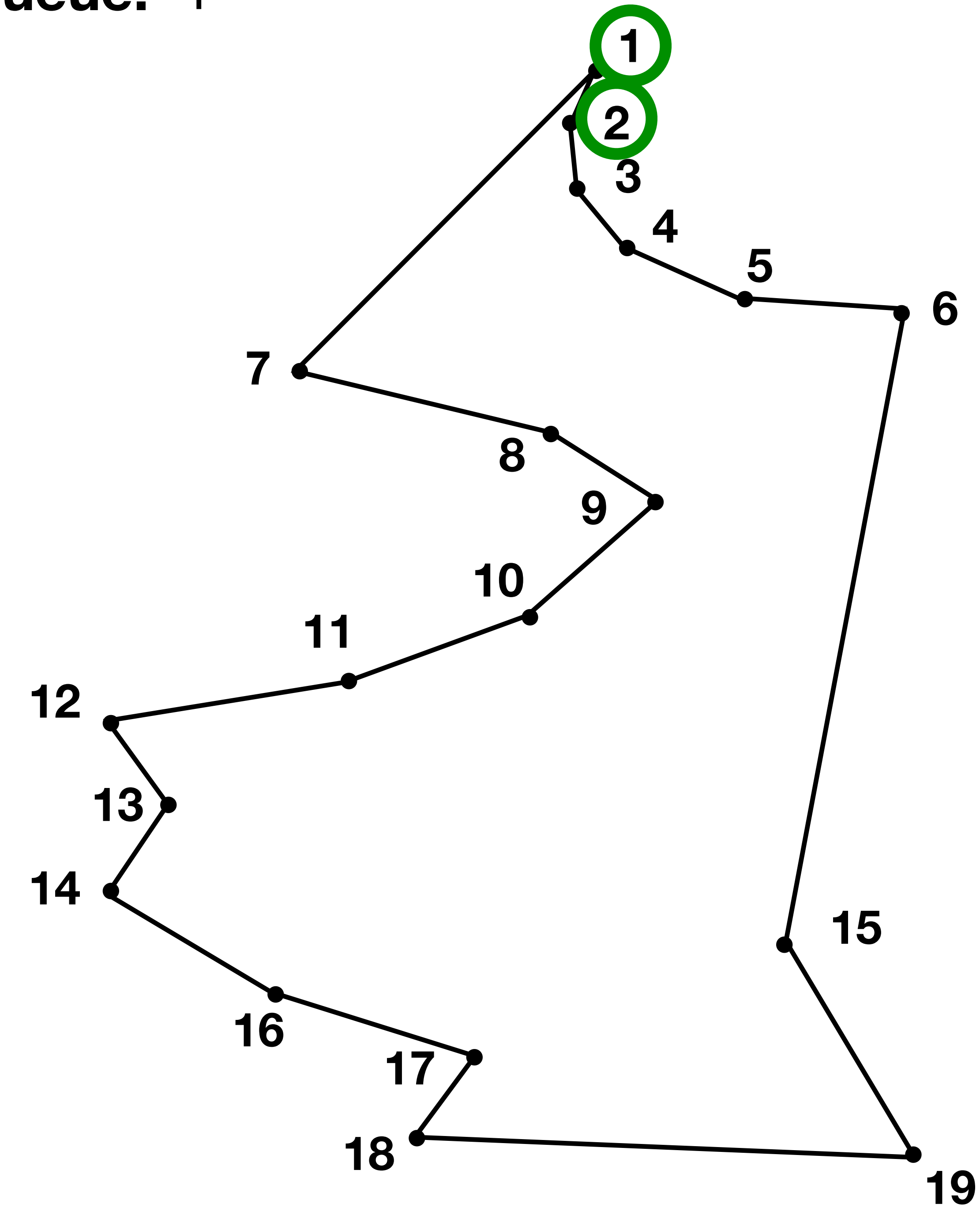
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon

Queue: 1



Technische Universität Braunschweig

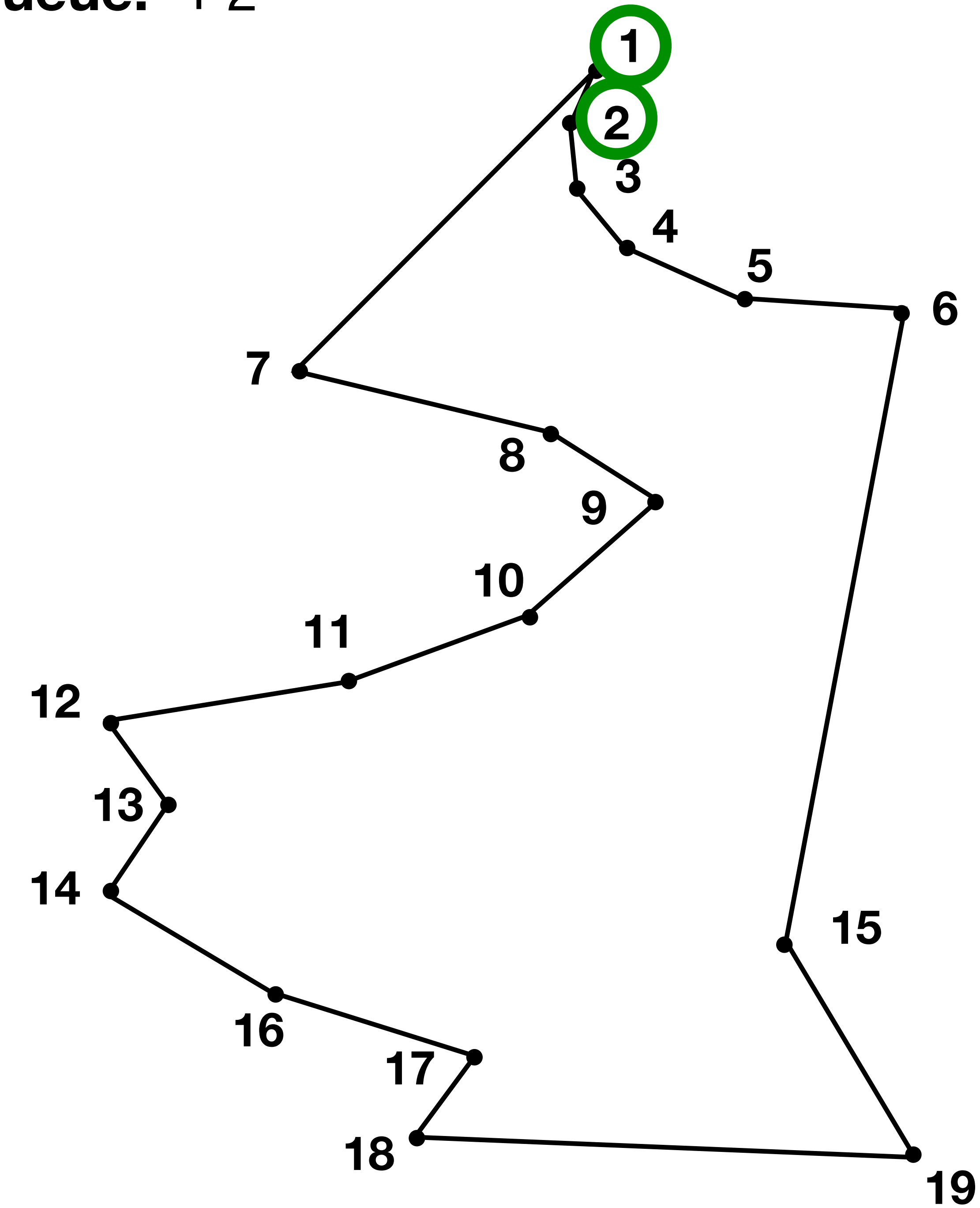
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon

Queue: 1 2



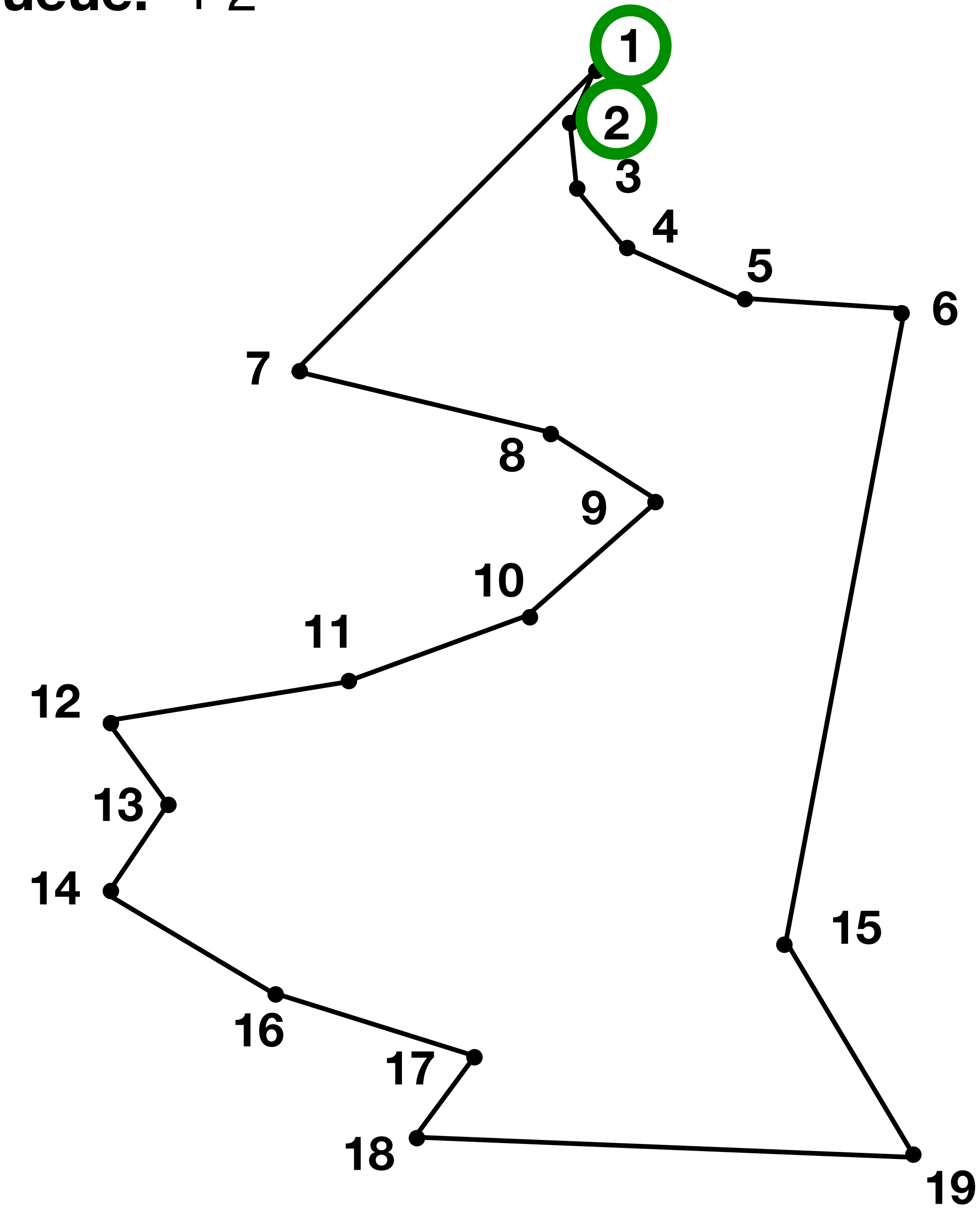
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex

Queue: 1 2



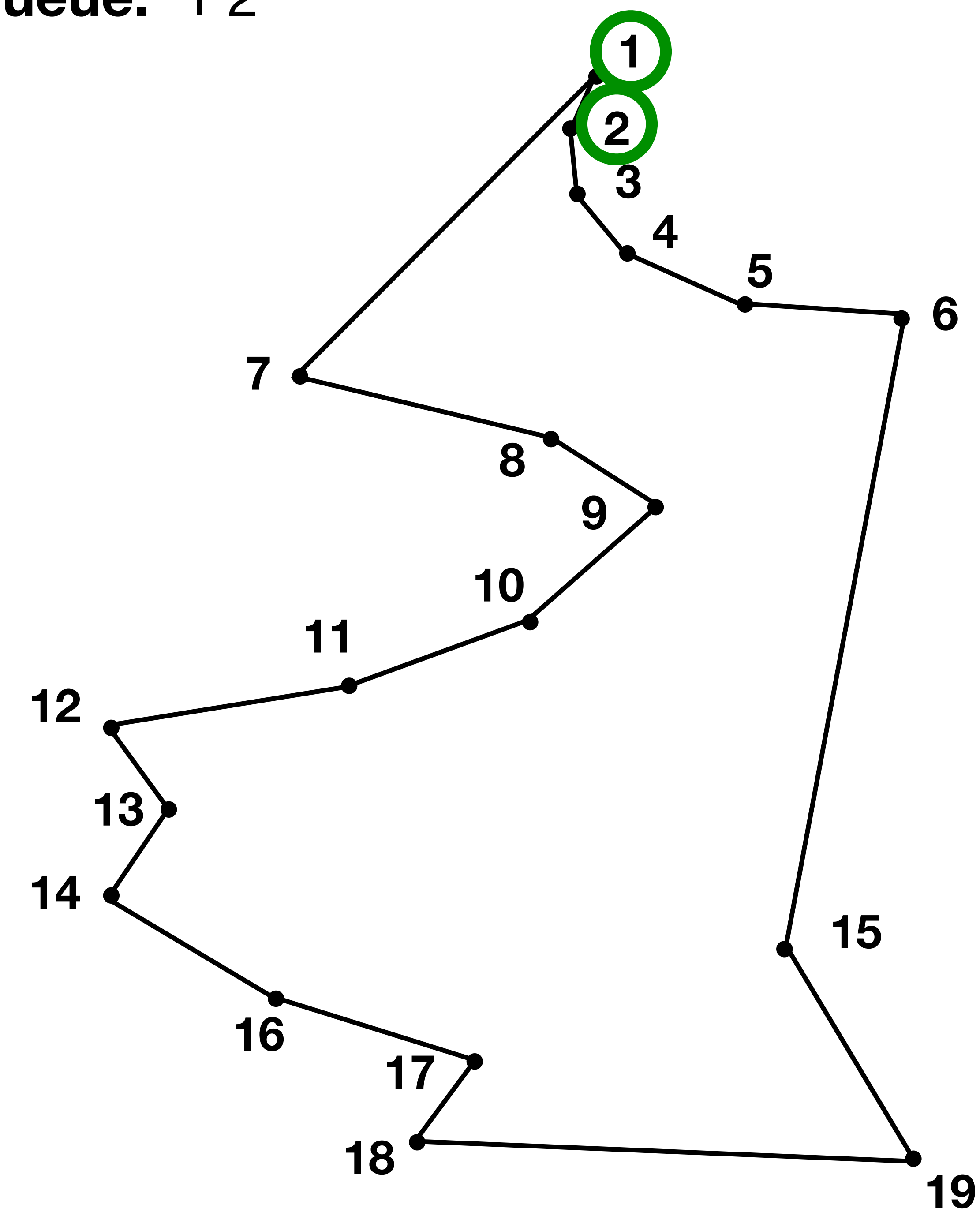
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:

Queue: 1 2



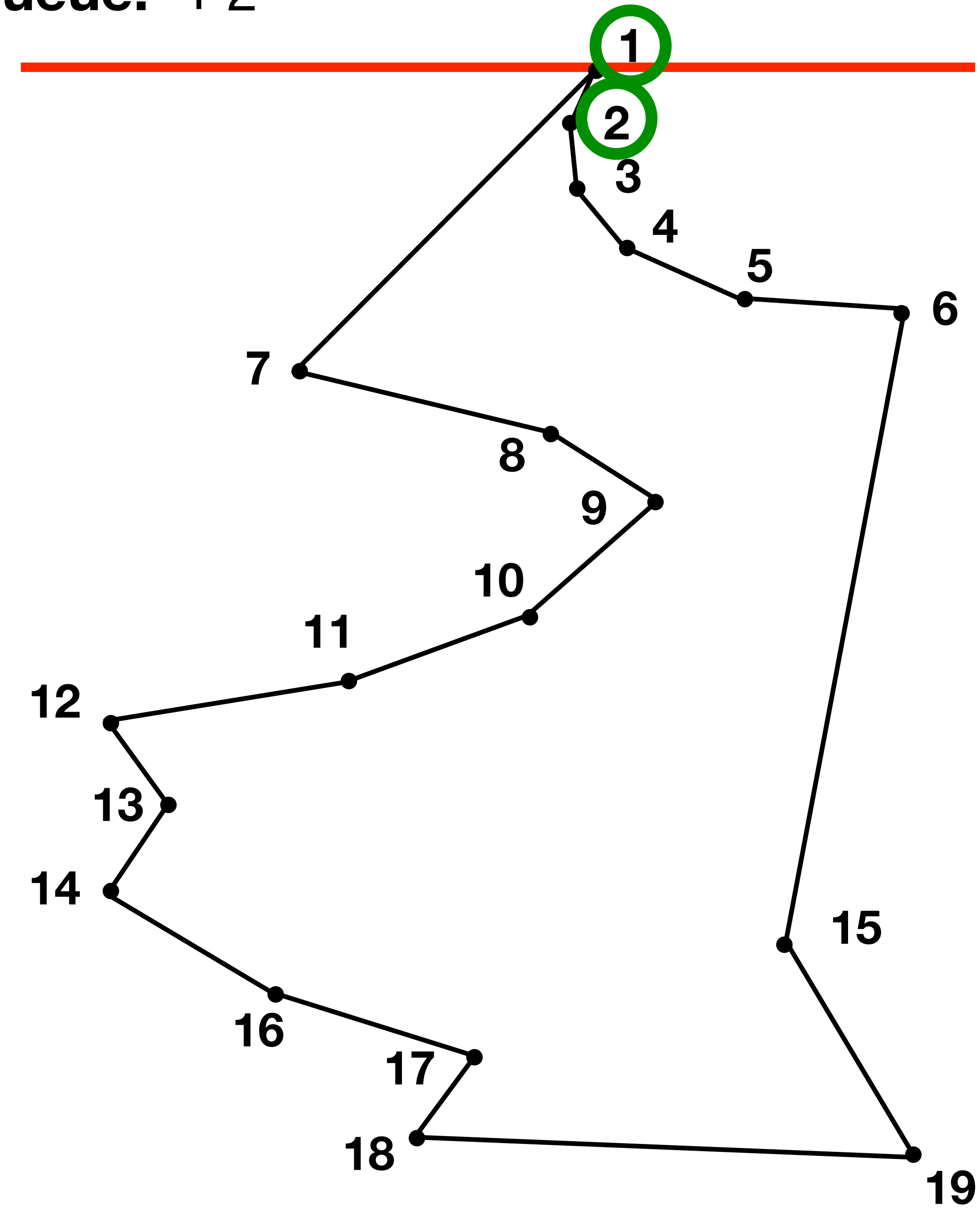
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:

Queue: 1 2



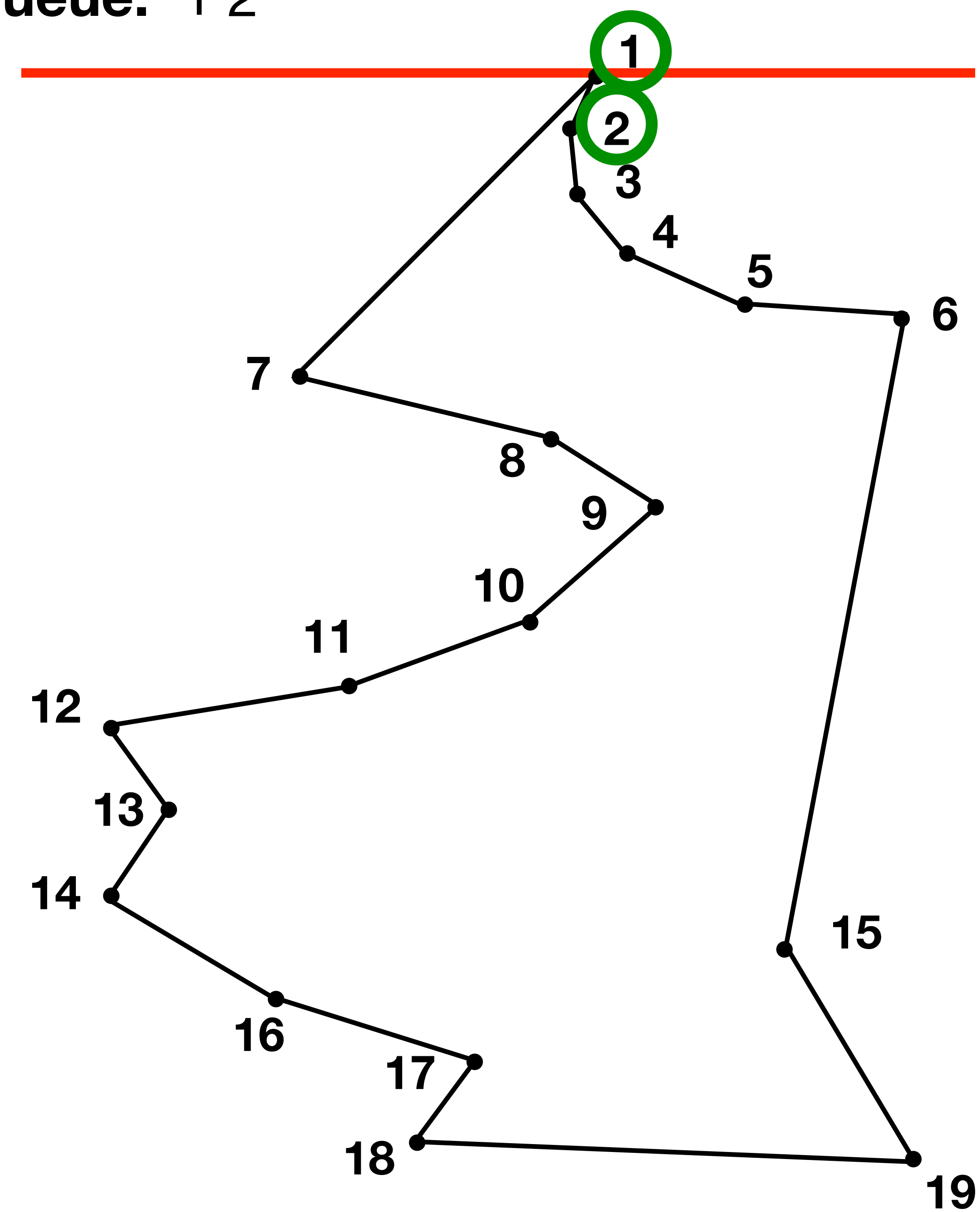
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:

Queue: 1 2



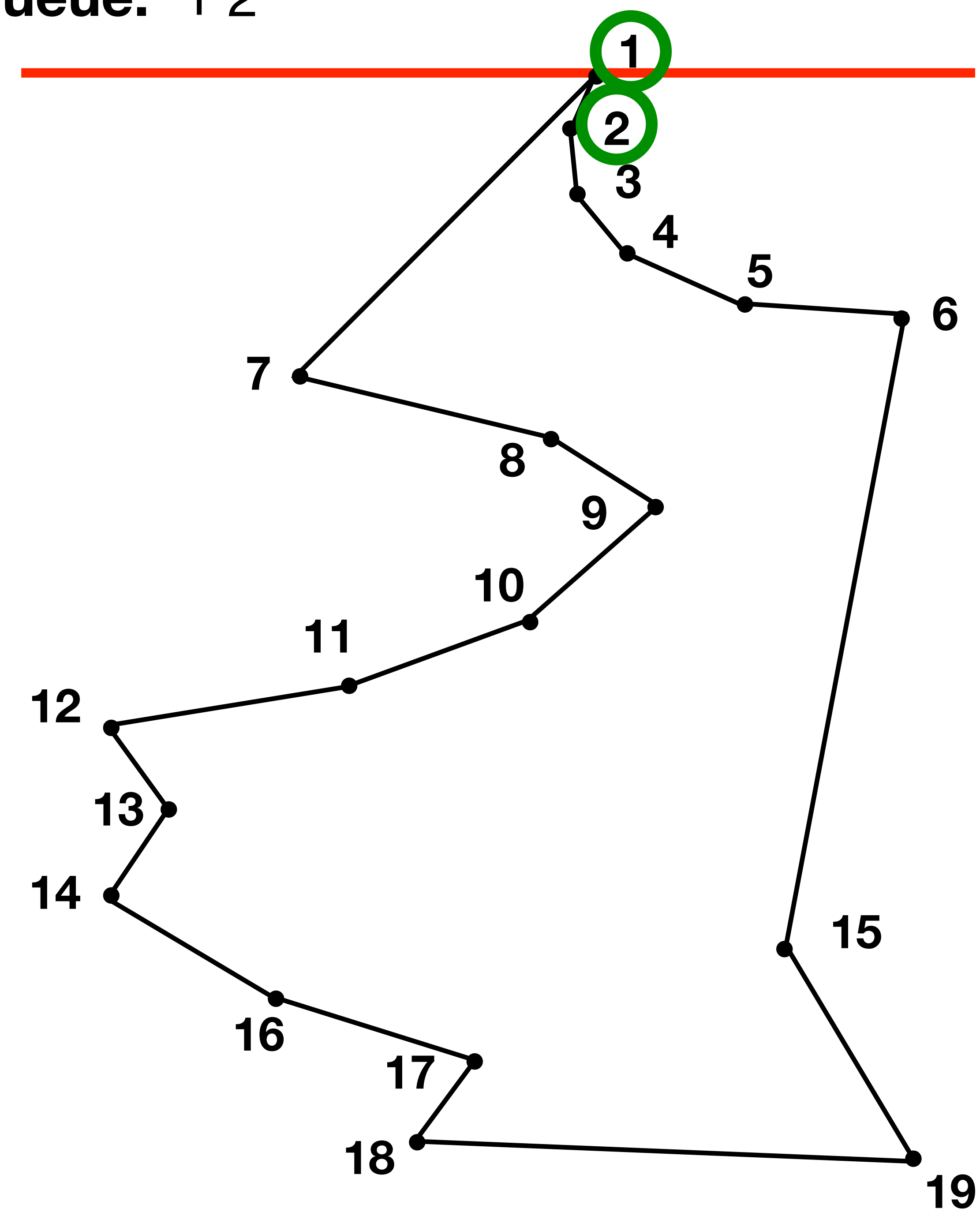
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue

Queue: 1 2



Algorithm 5.16

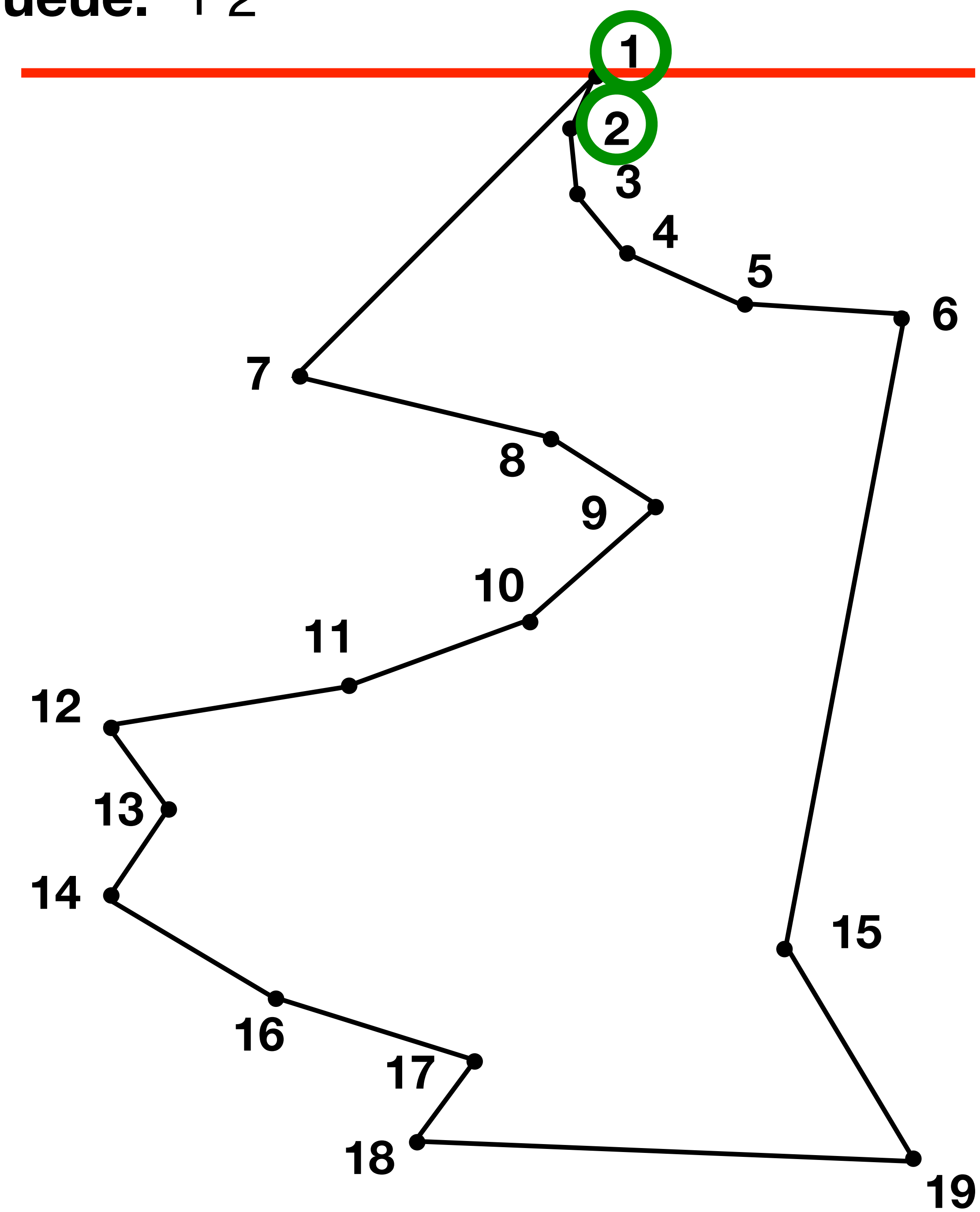
Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue

Chain

Queue: 1 2



Algorithm 5.16

Input: A y-monotone polygon P

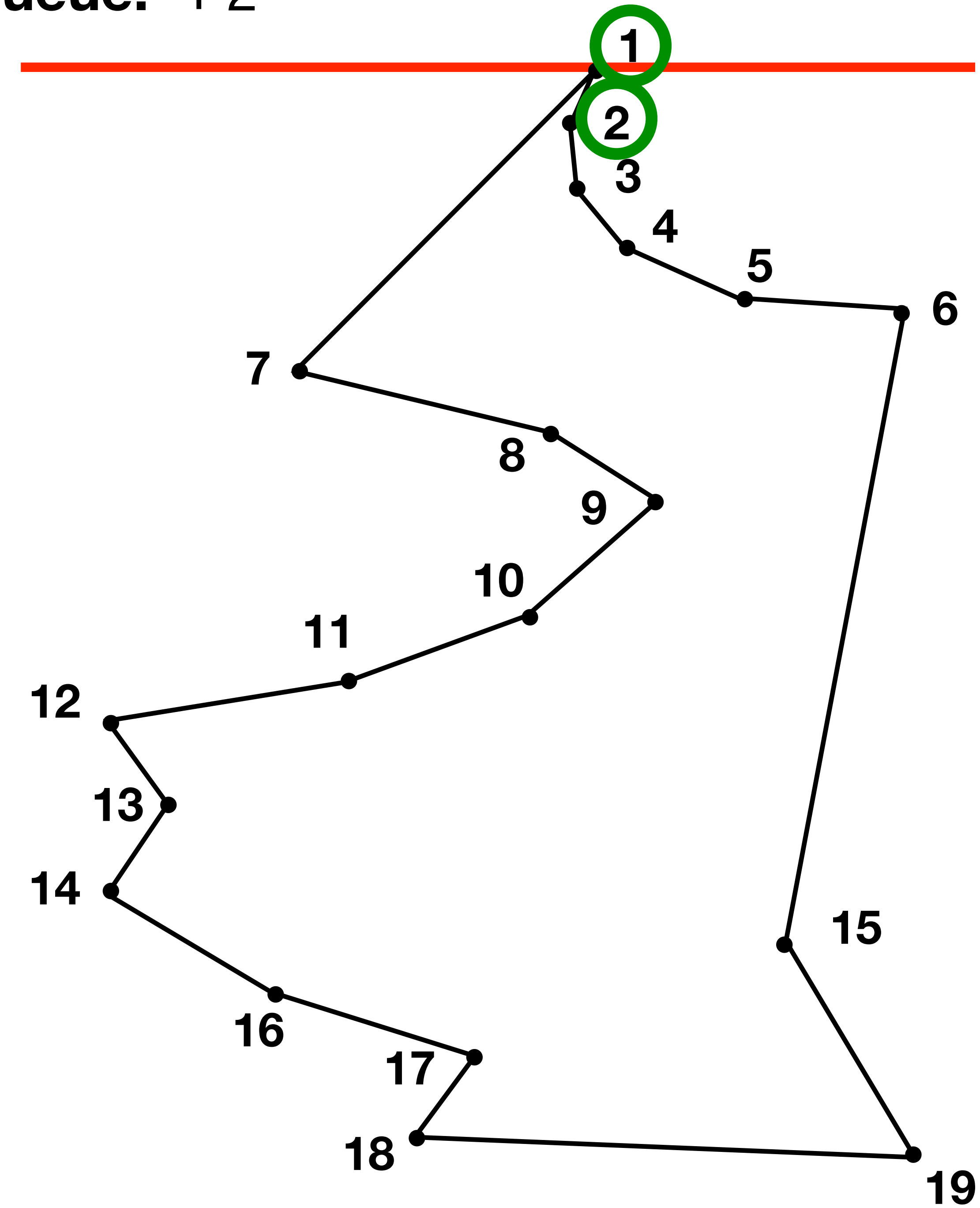
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue

   3.2 If same chain, predecessor reflex:

Chain

Queue: 1 2



Algorithm 5.16

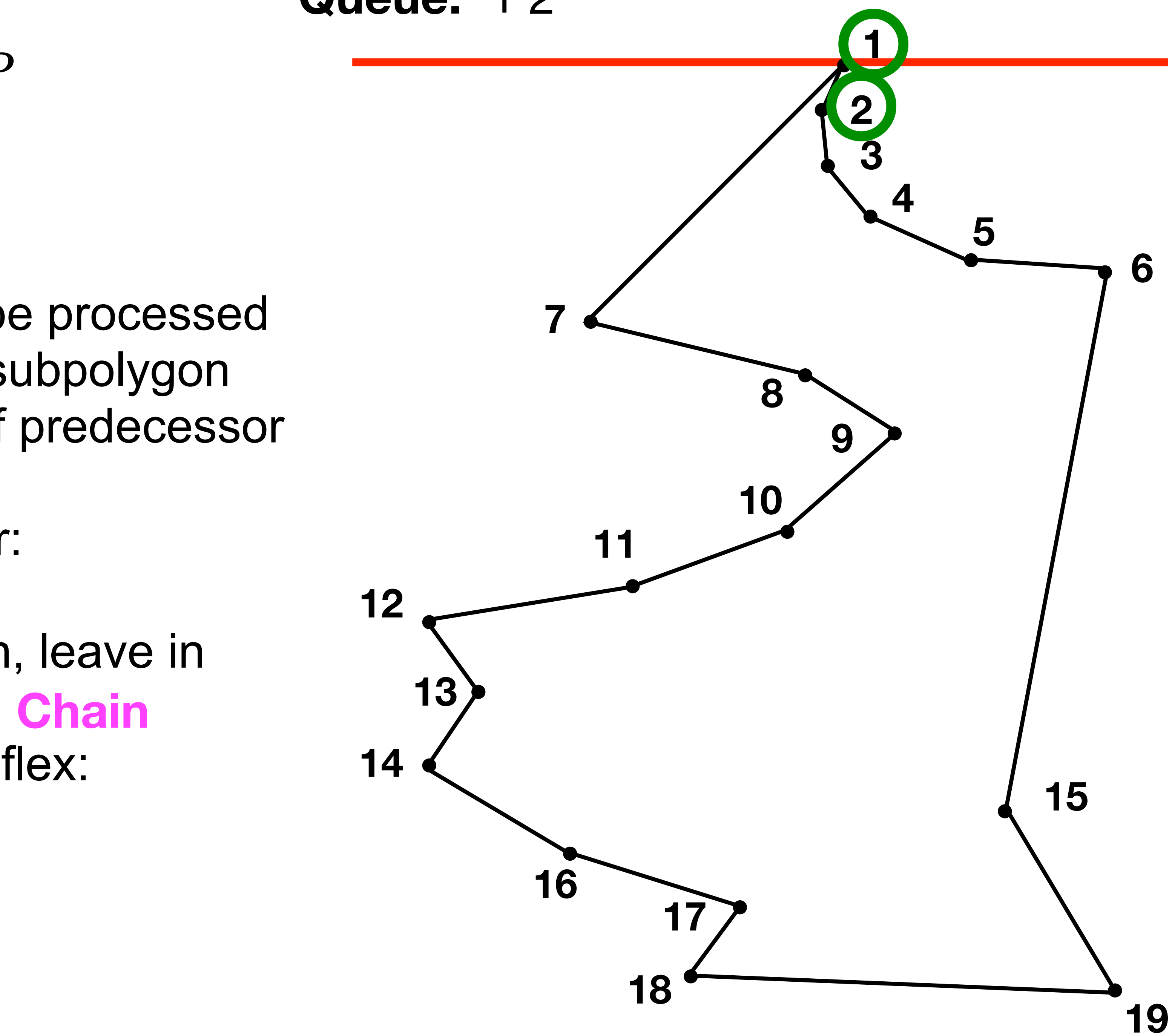
Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue

   3.2 If same chain, predecessor reflex:
   - Add to queue

Queue: 1 2



Algorithm 5.16

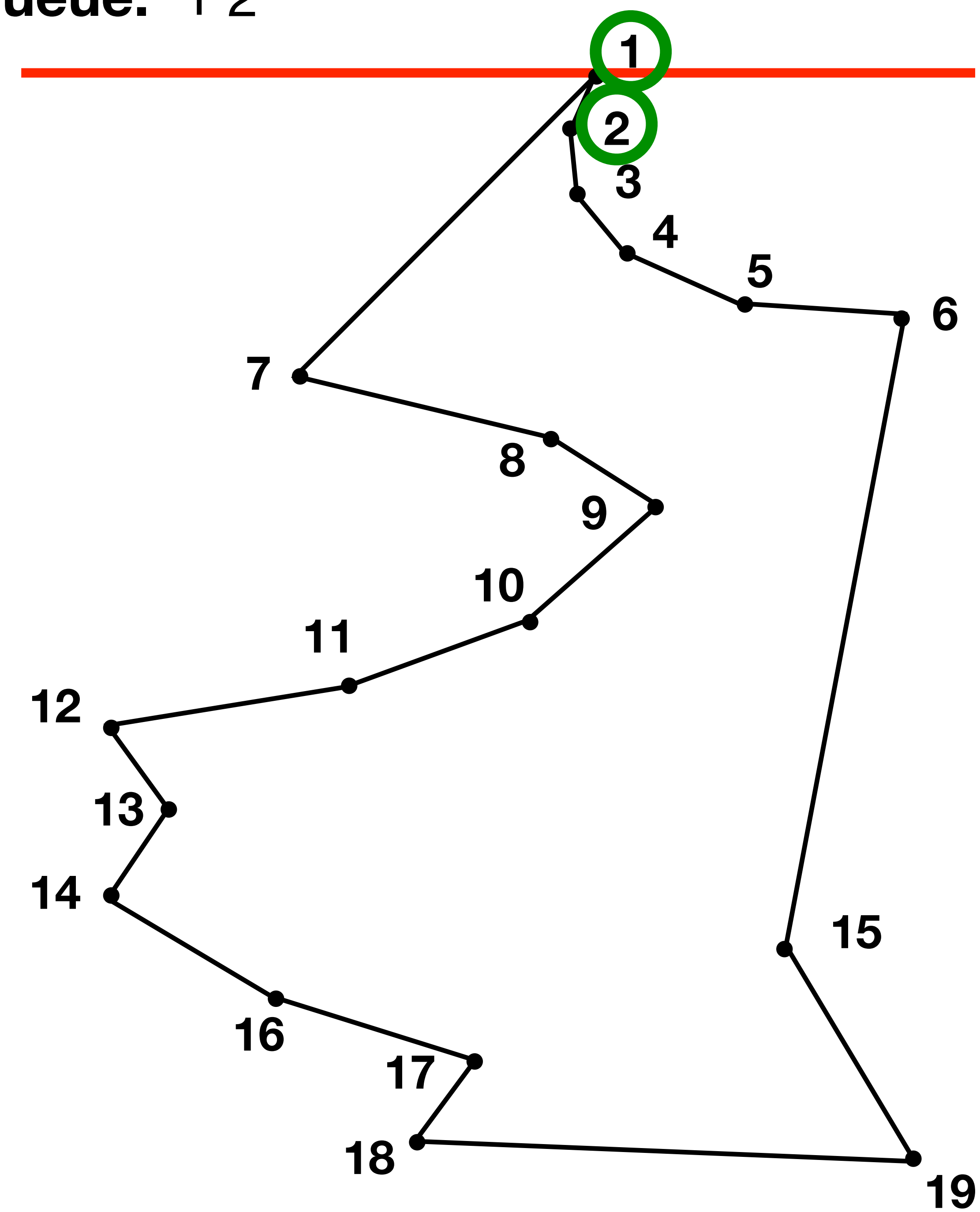
Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

Queue: 1 2



Algorithm 5.16

Input: A y-monotone polygon P

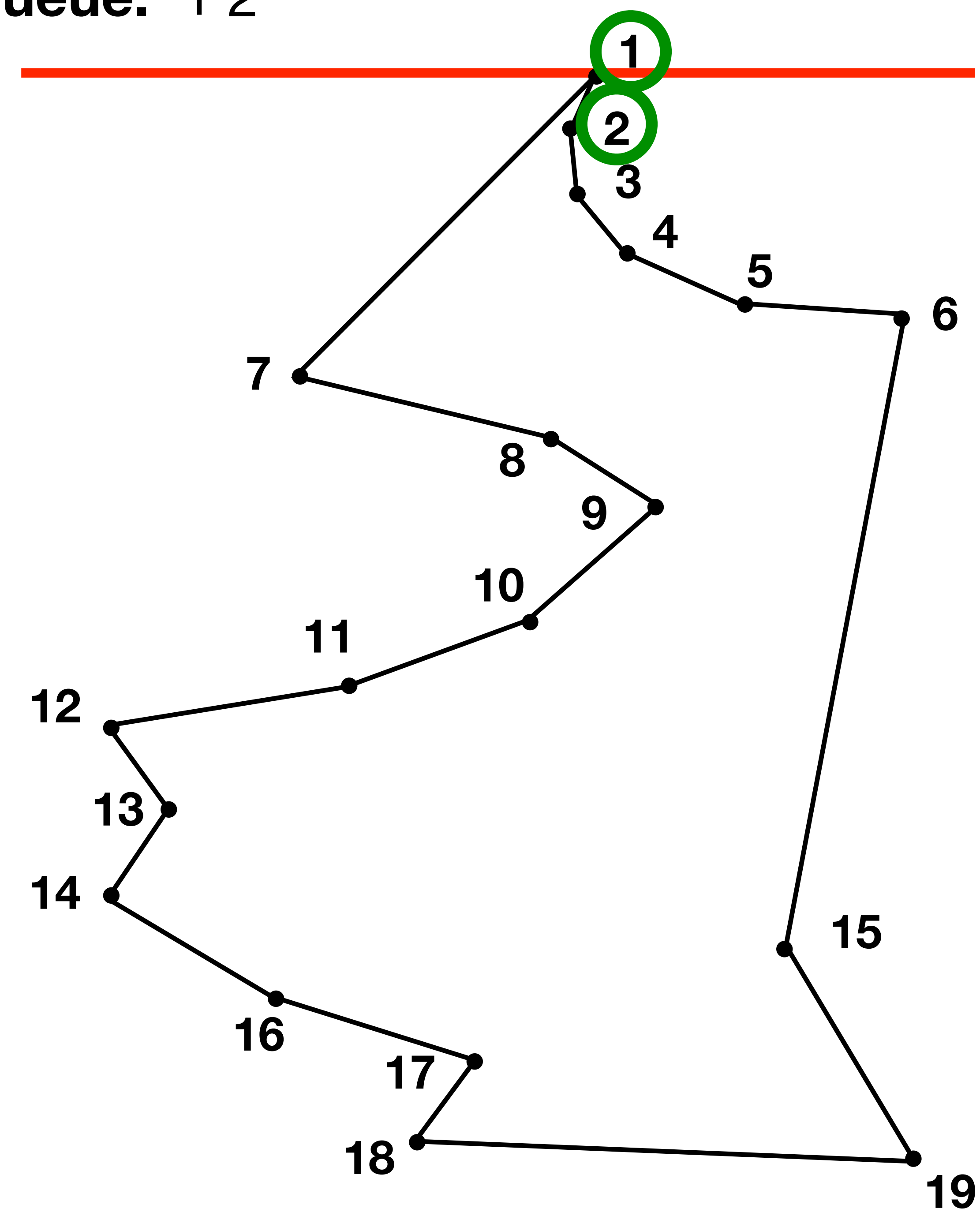
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:

Queue: 1 2



Algorithm 5.16

Input: A y-monotone polygon P

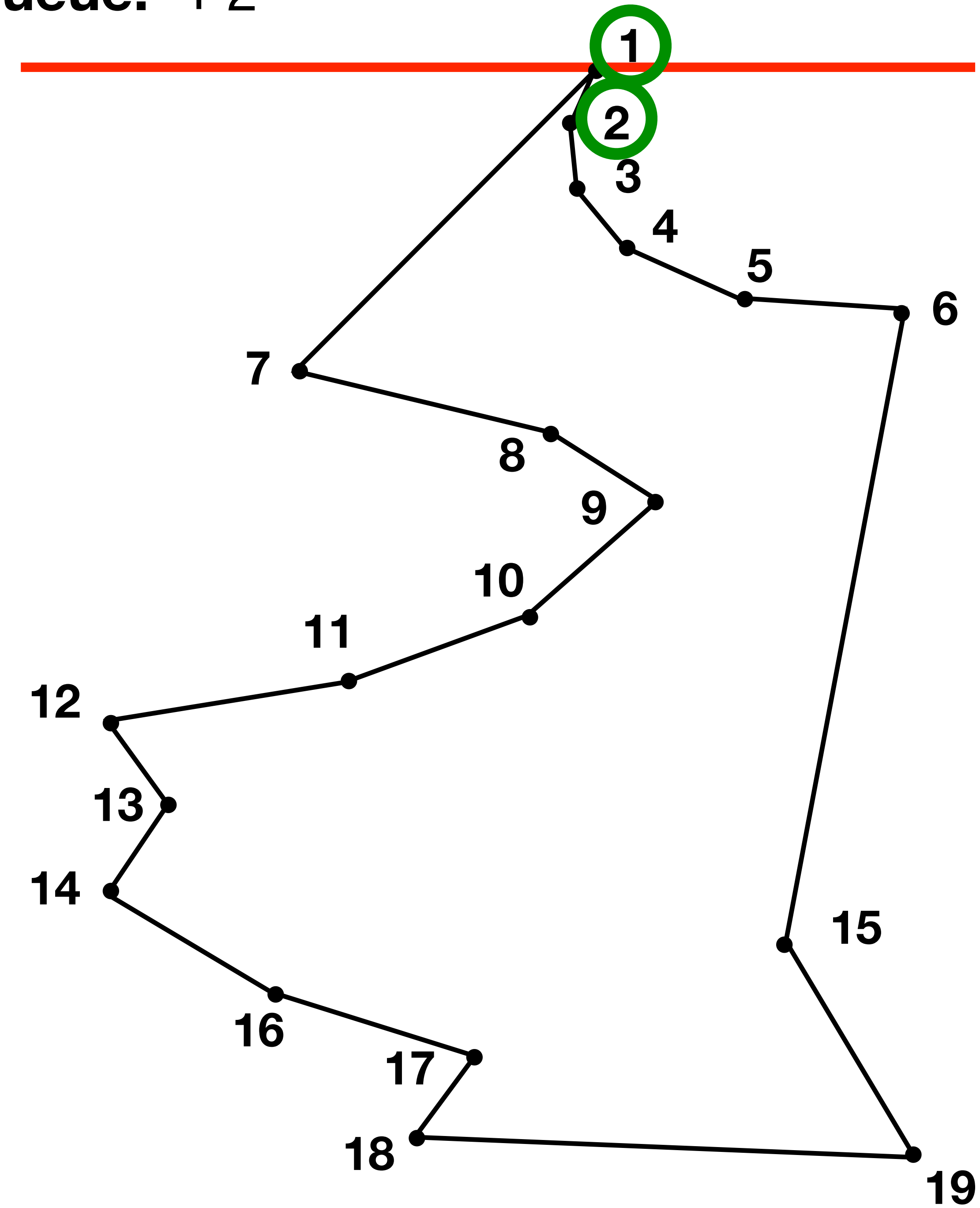
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update

Queue: 1 2



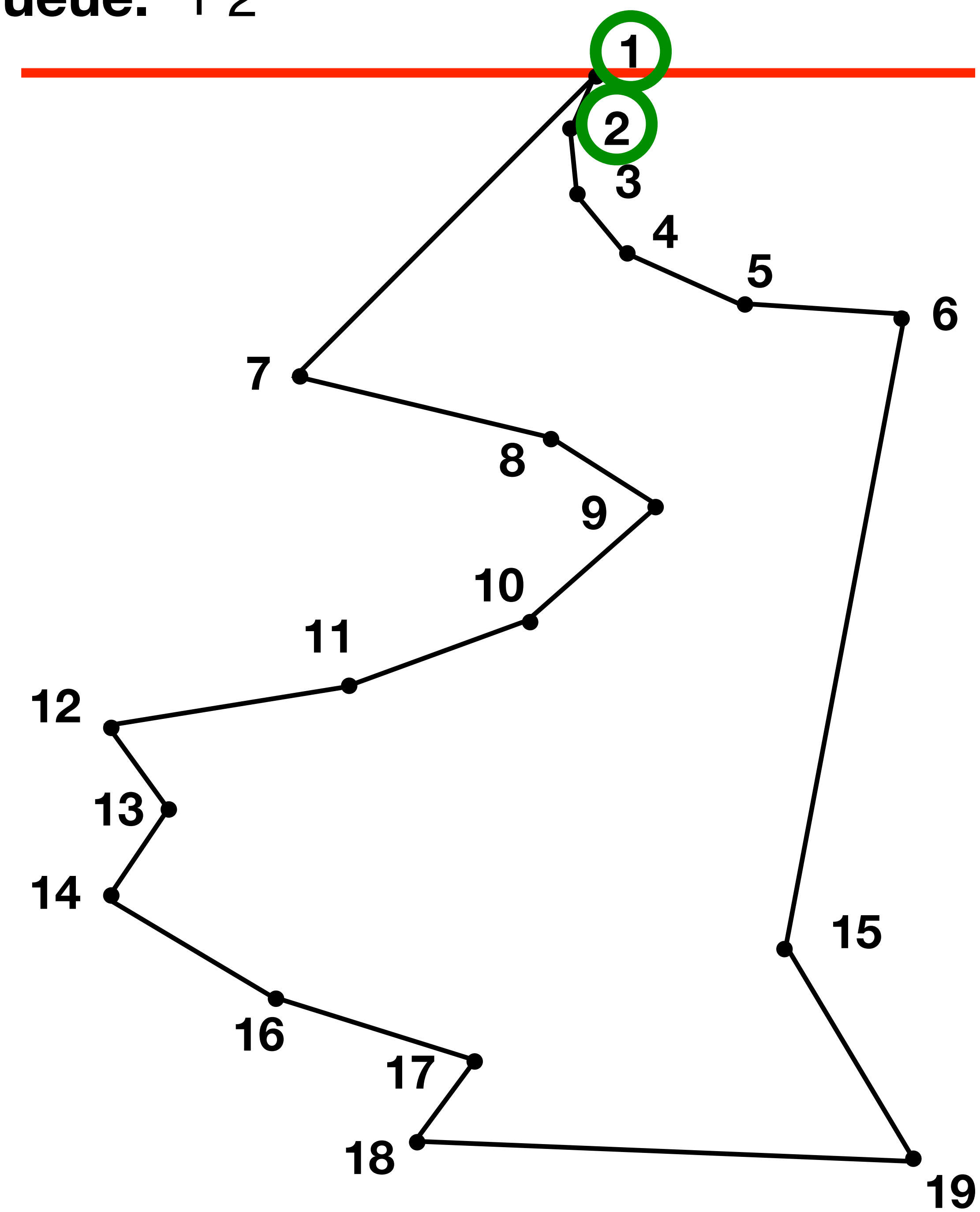
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2



Algorithm 5.16

Input: A y-monotone polygon P

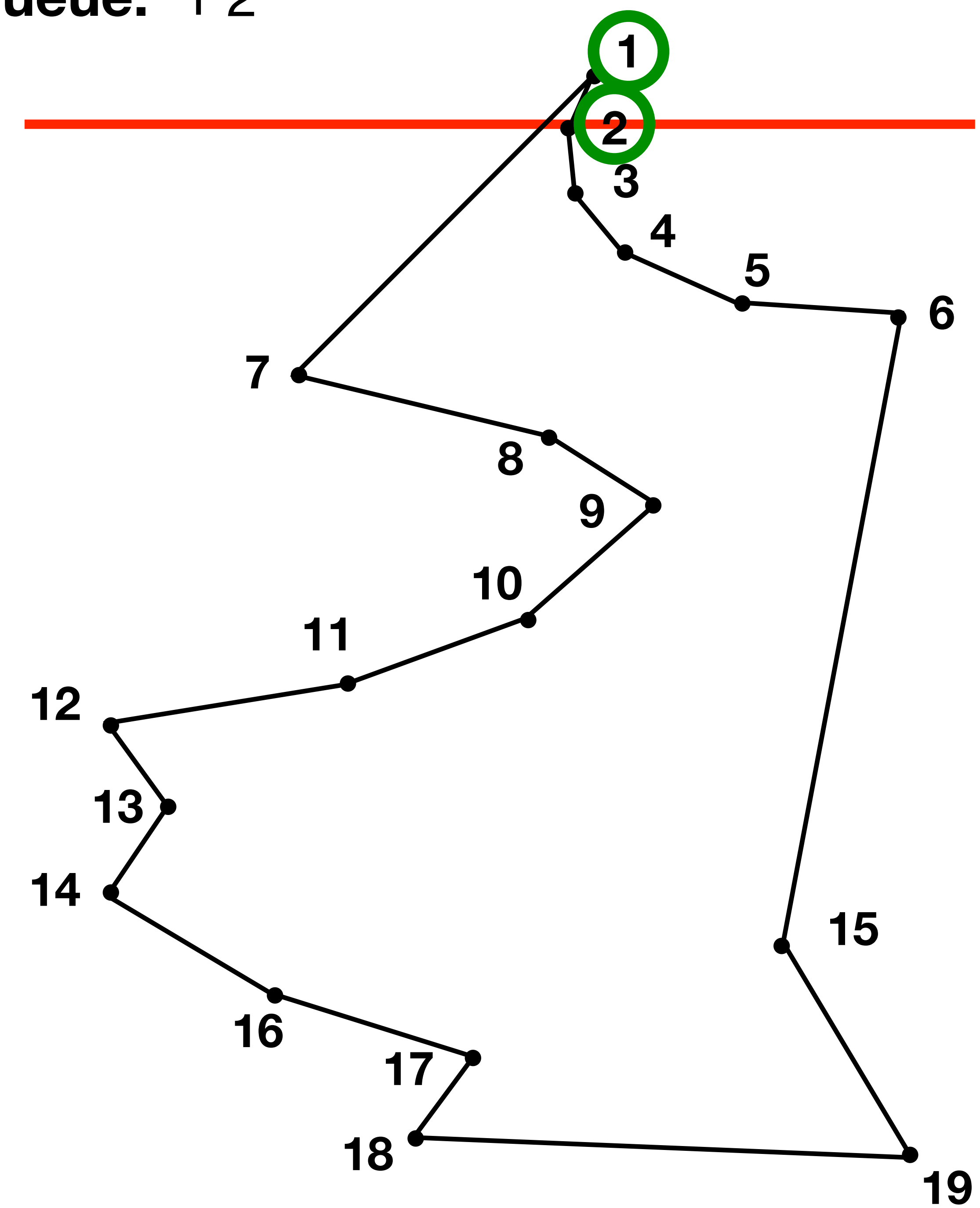
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2



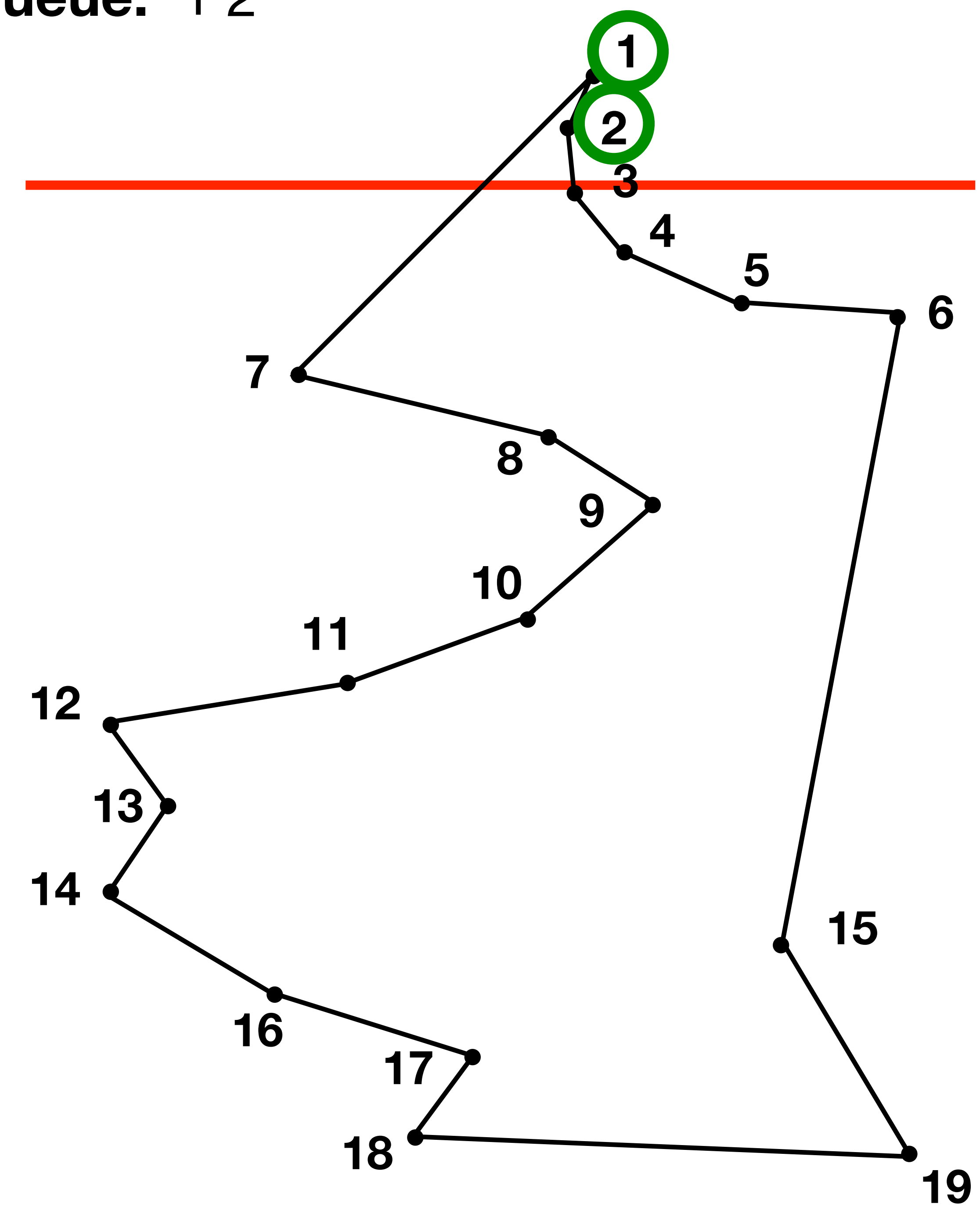
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2



Technische Universität Braunschweig

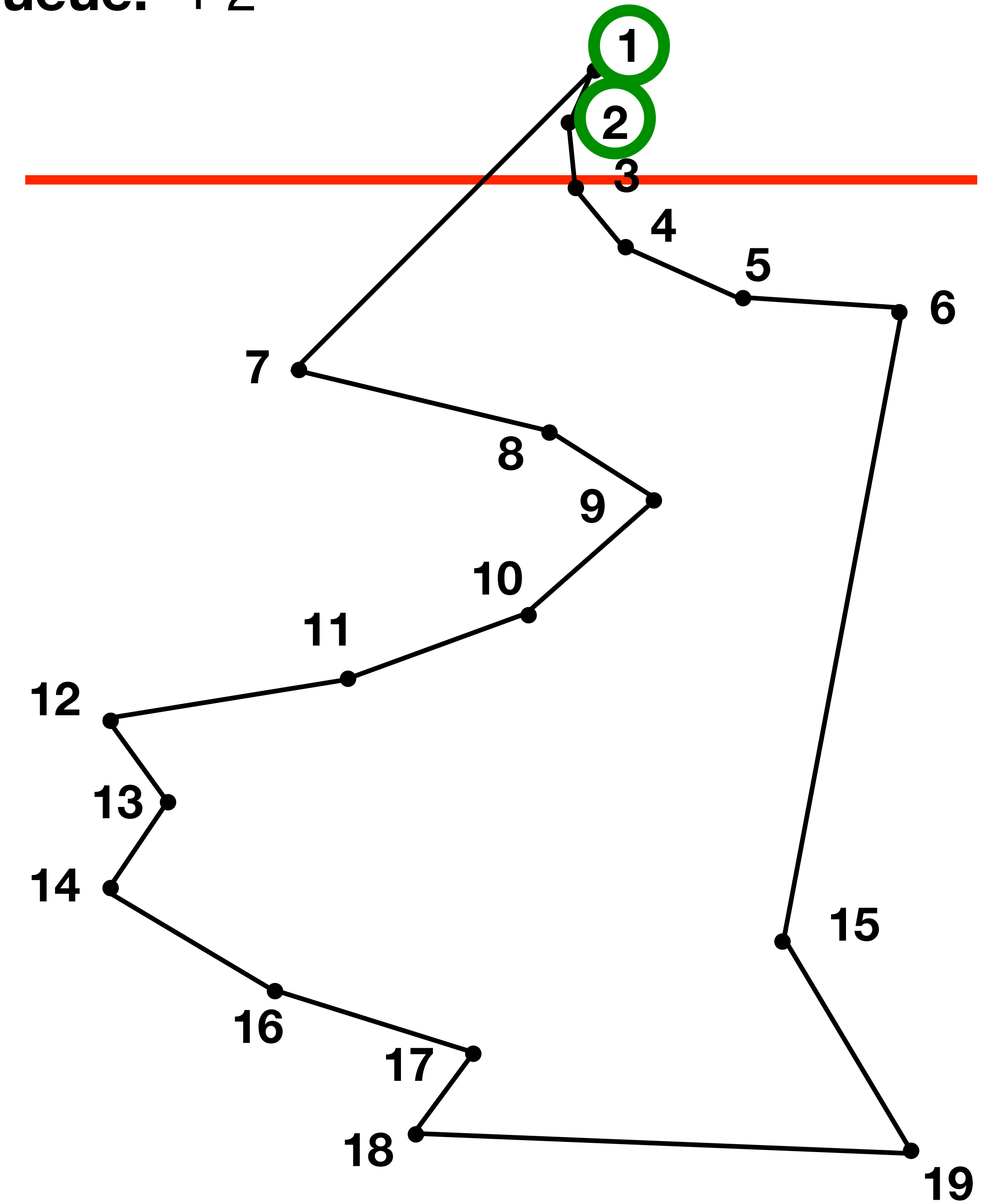
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2



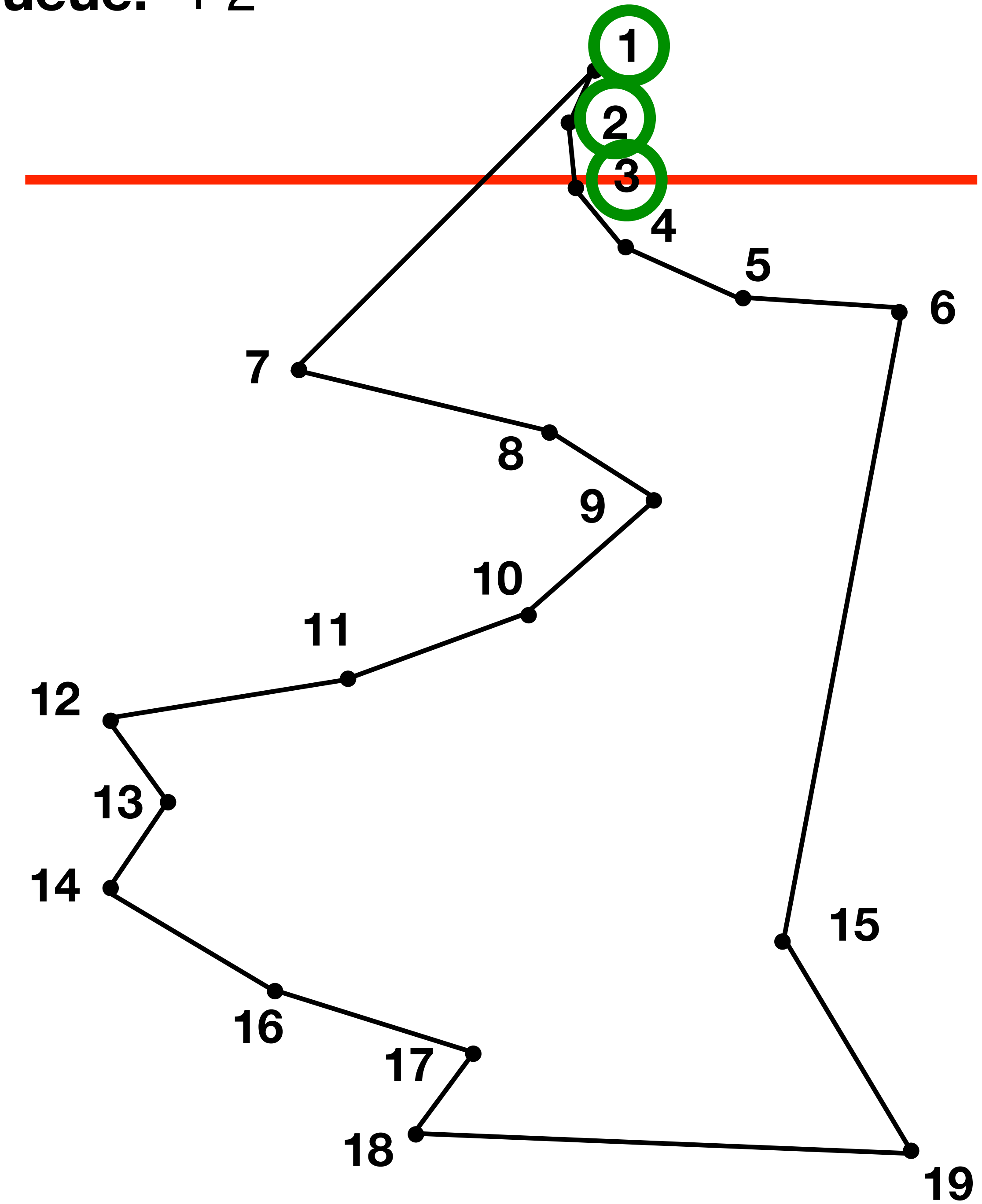
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2



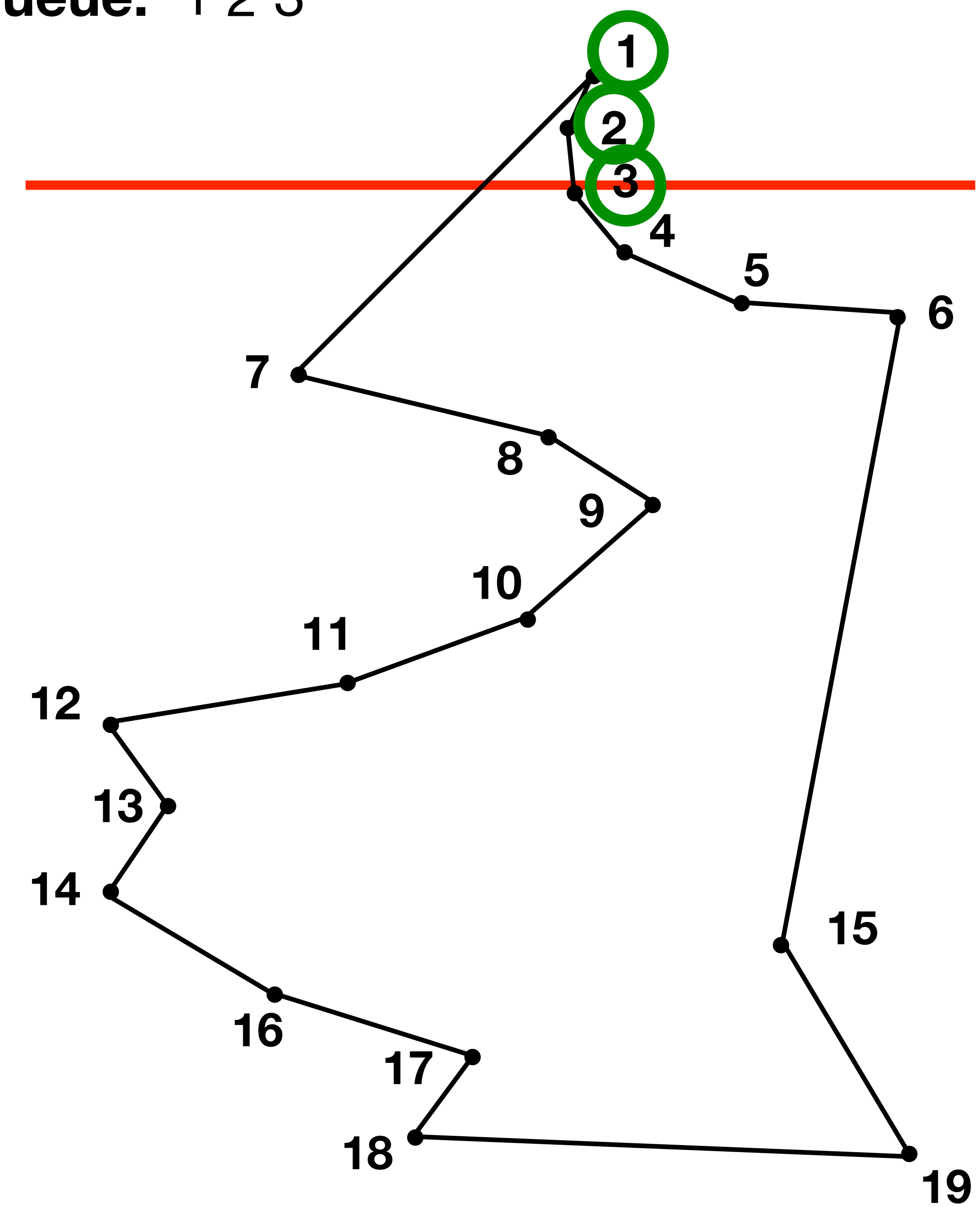
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3



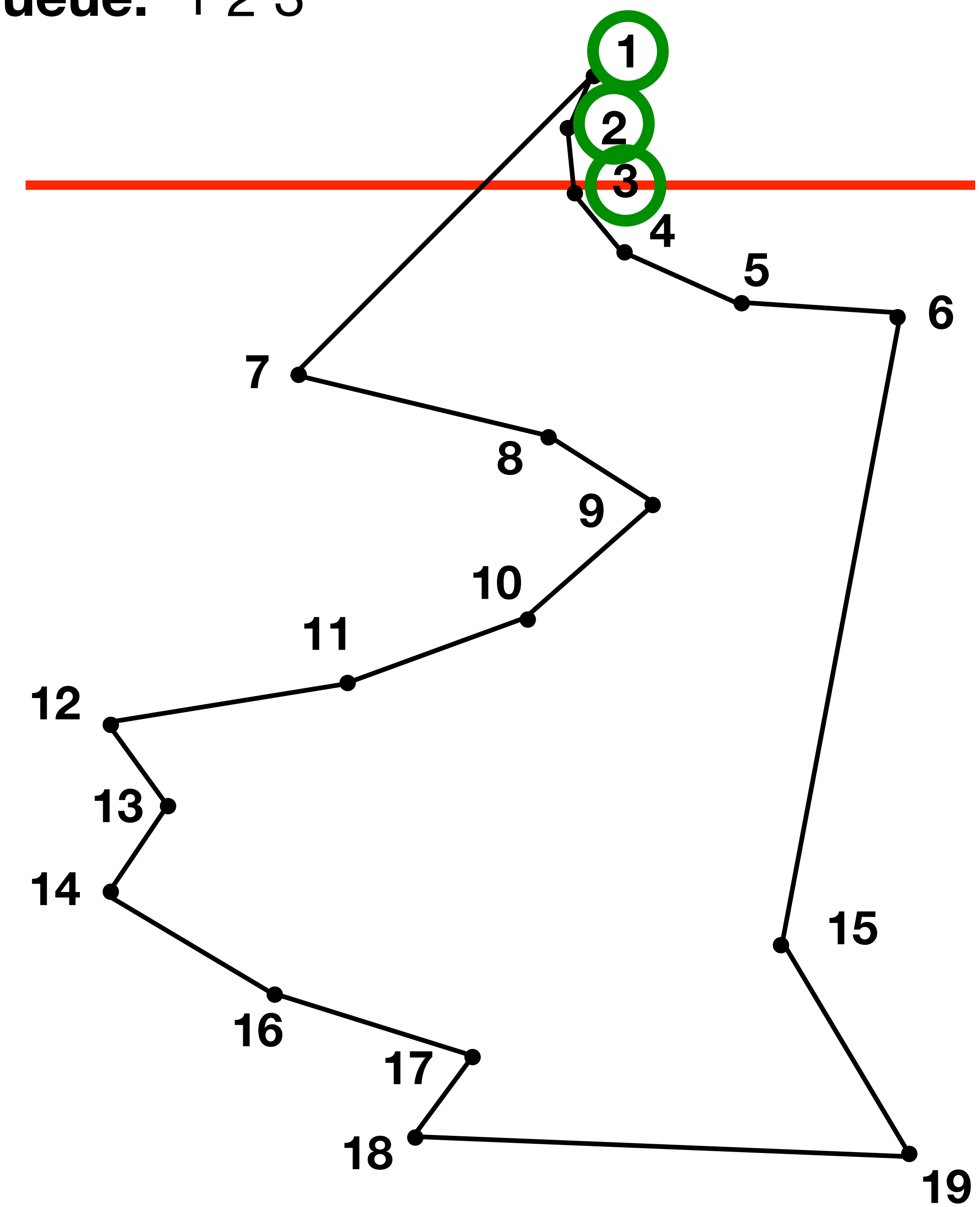
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3



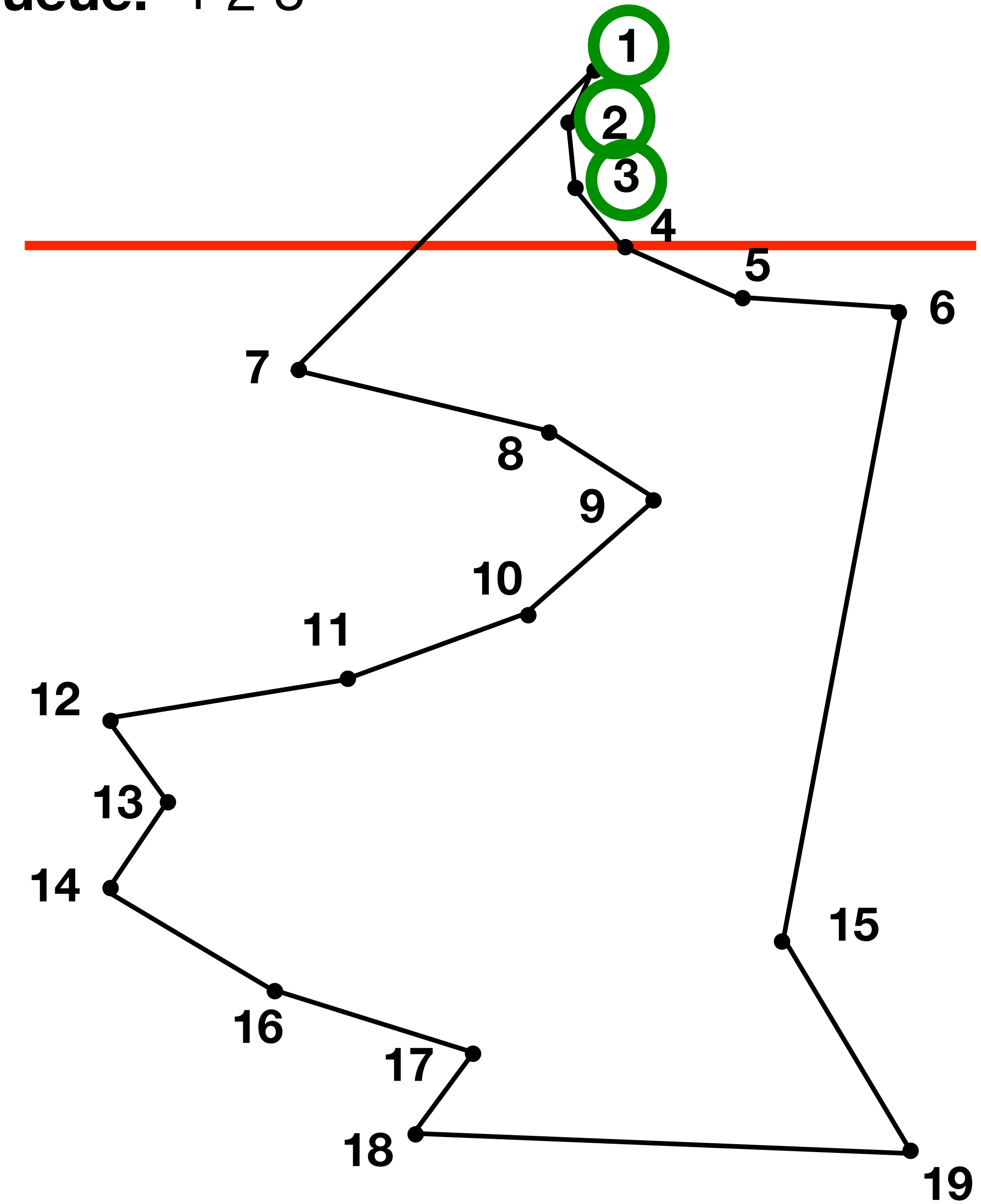
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3



Algorithm 5.16

Input: A y-monotone polygon P

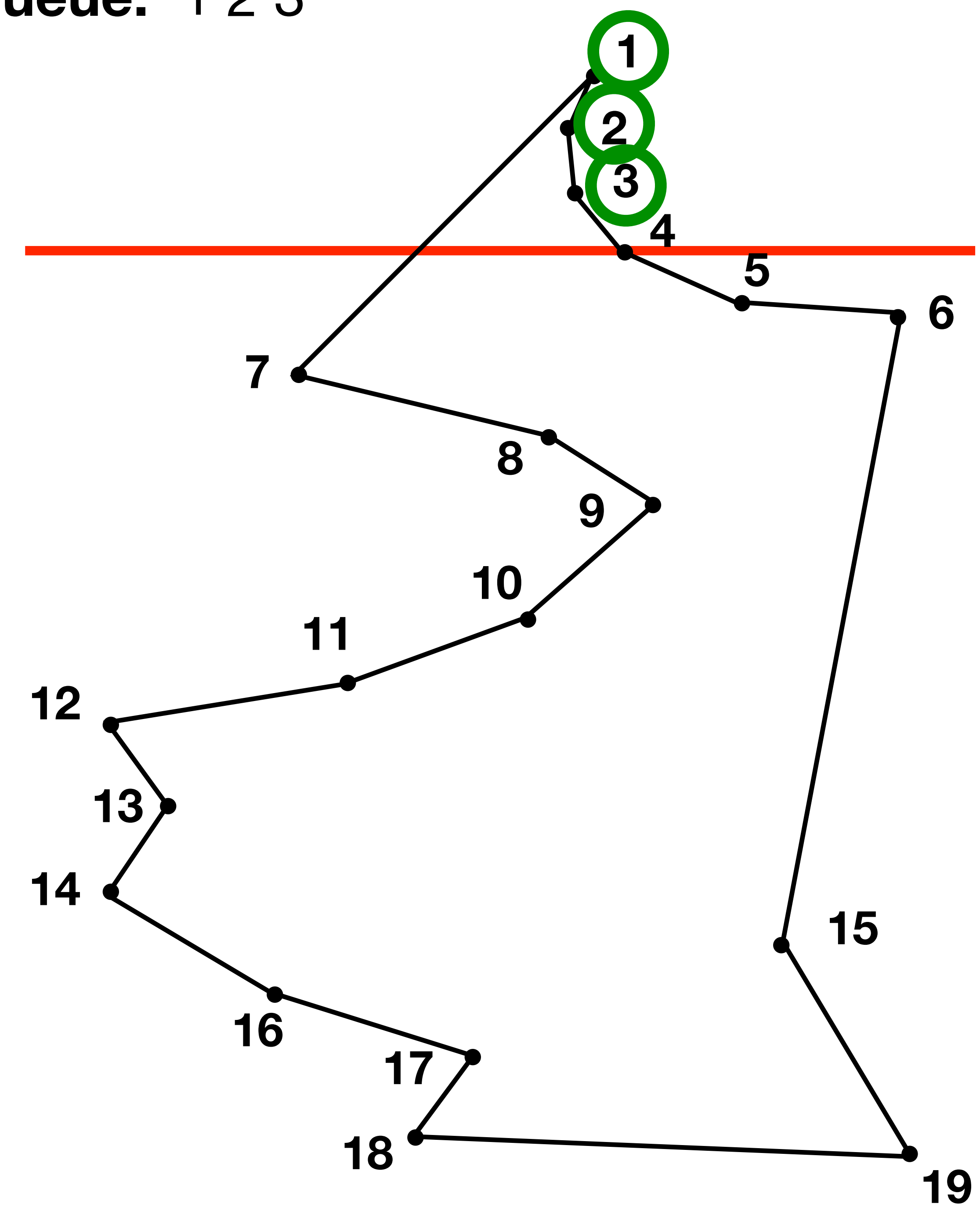
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2 3



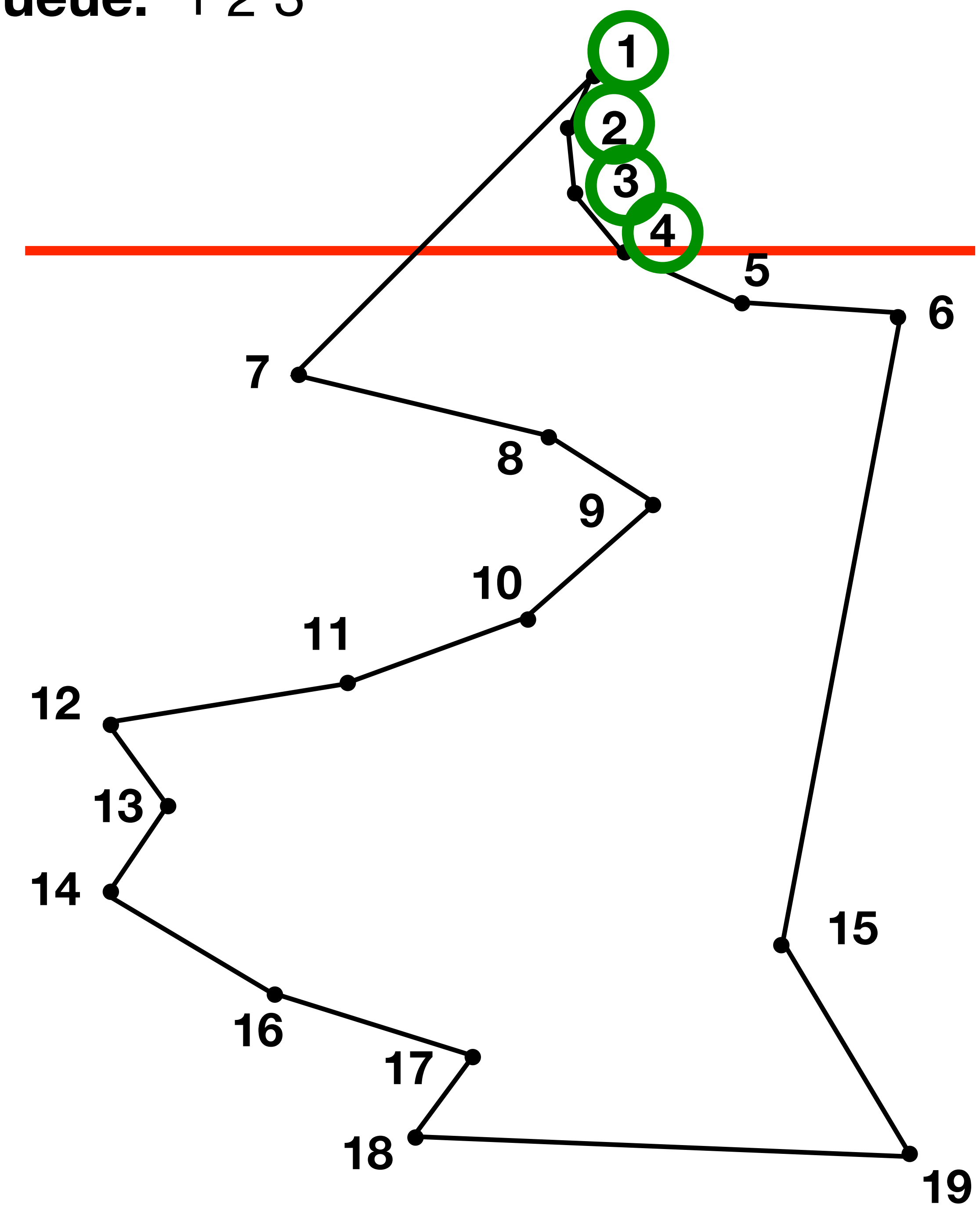
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3



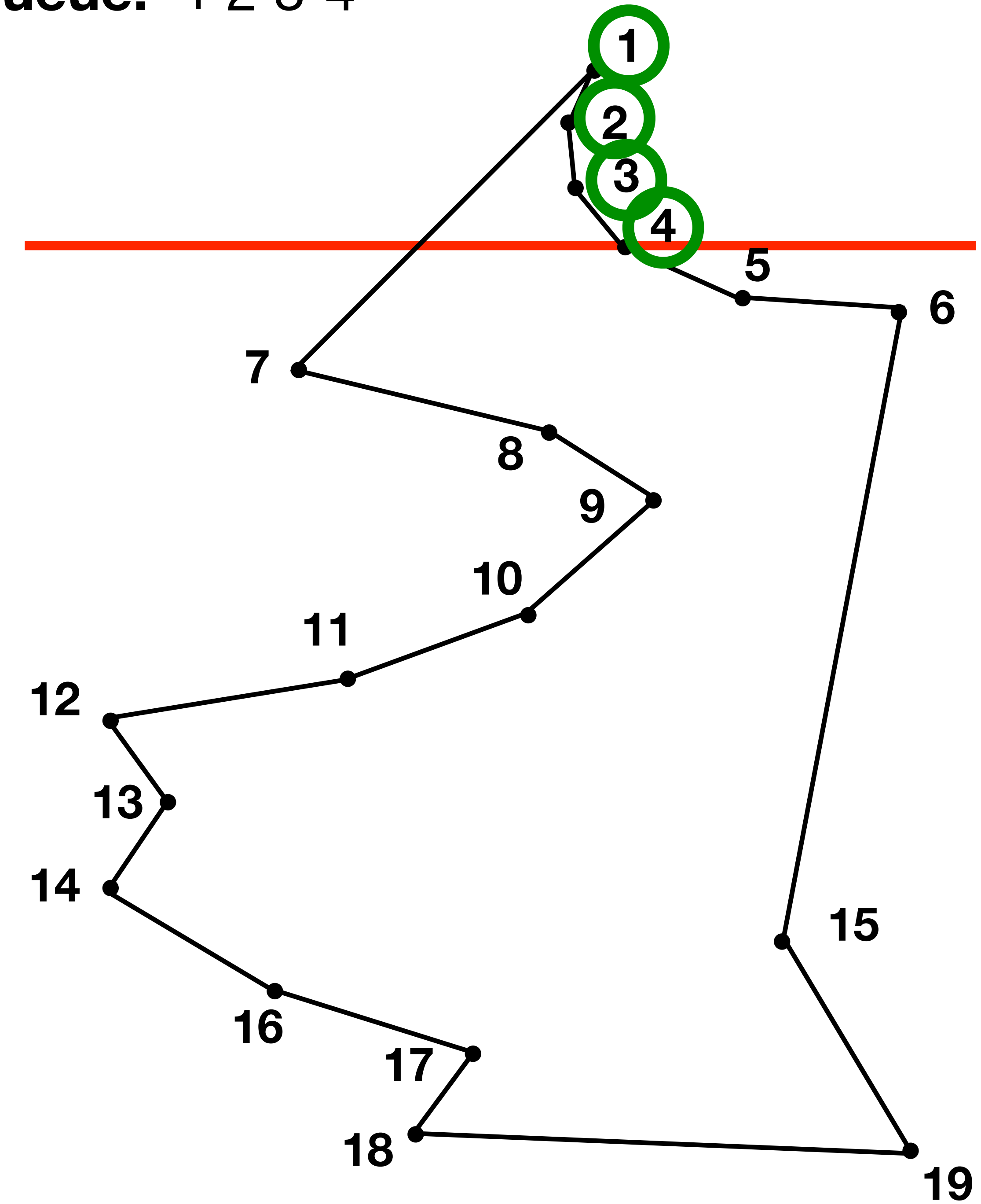
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 1 2 3 4



Algorithm 5.16

Input: A y-monotone polygon P

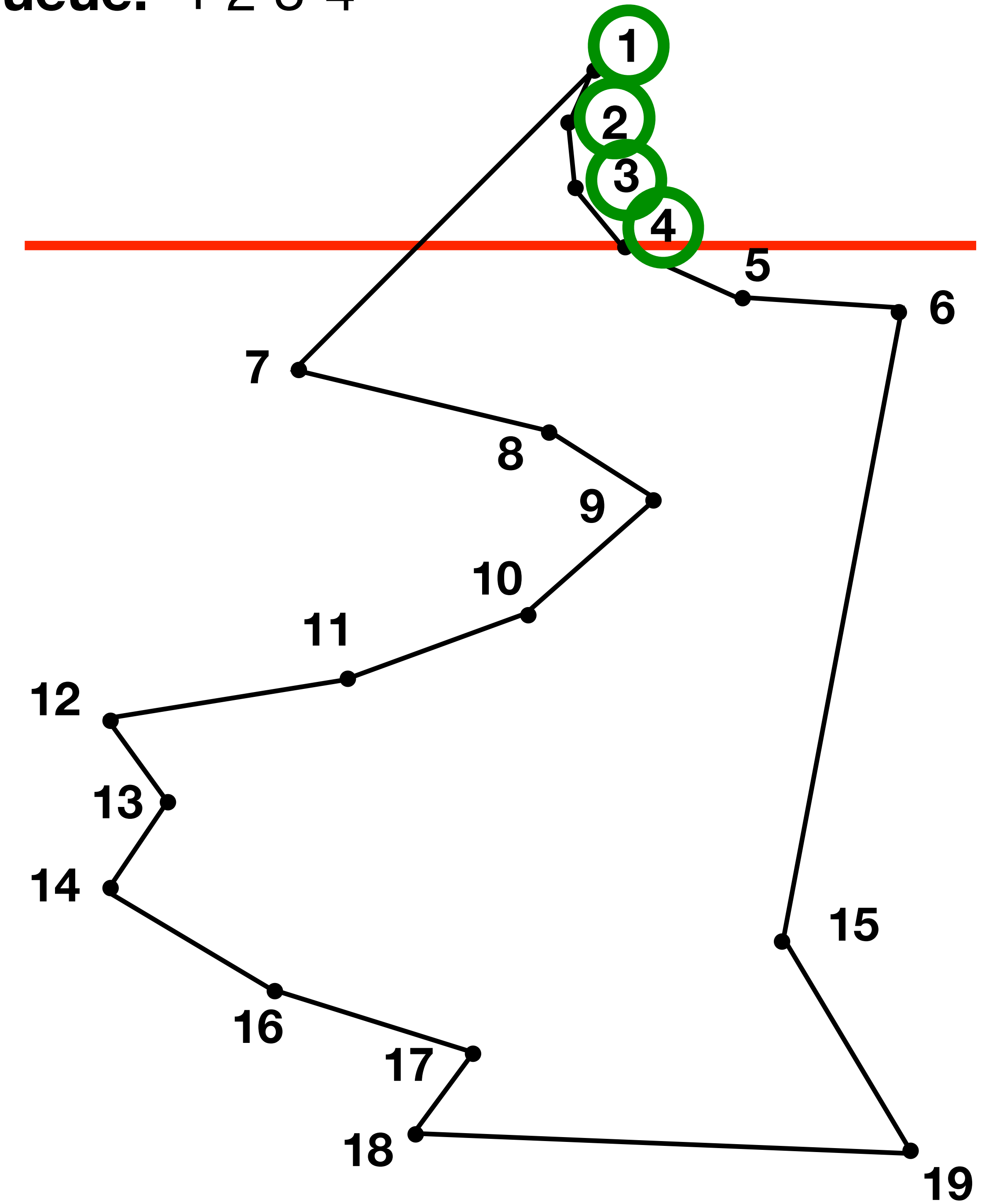
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2 3 4



Algorithm 5.16

Input: A y-monotone polygon P

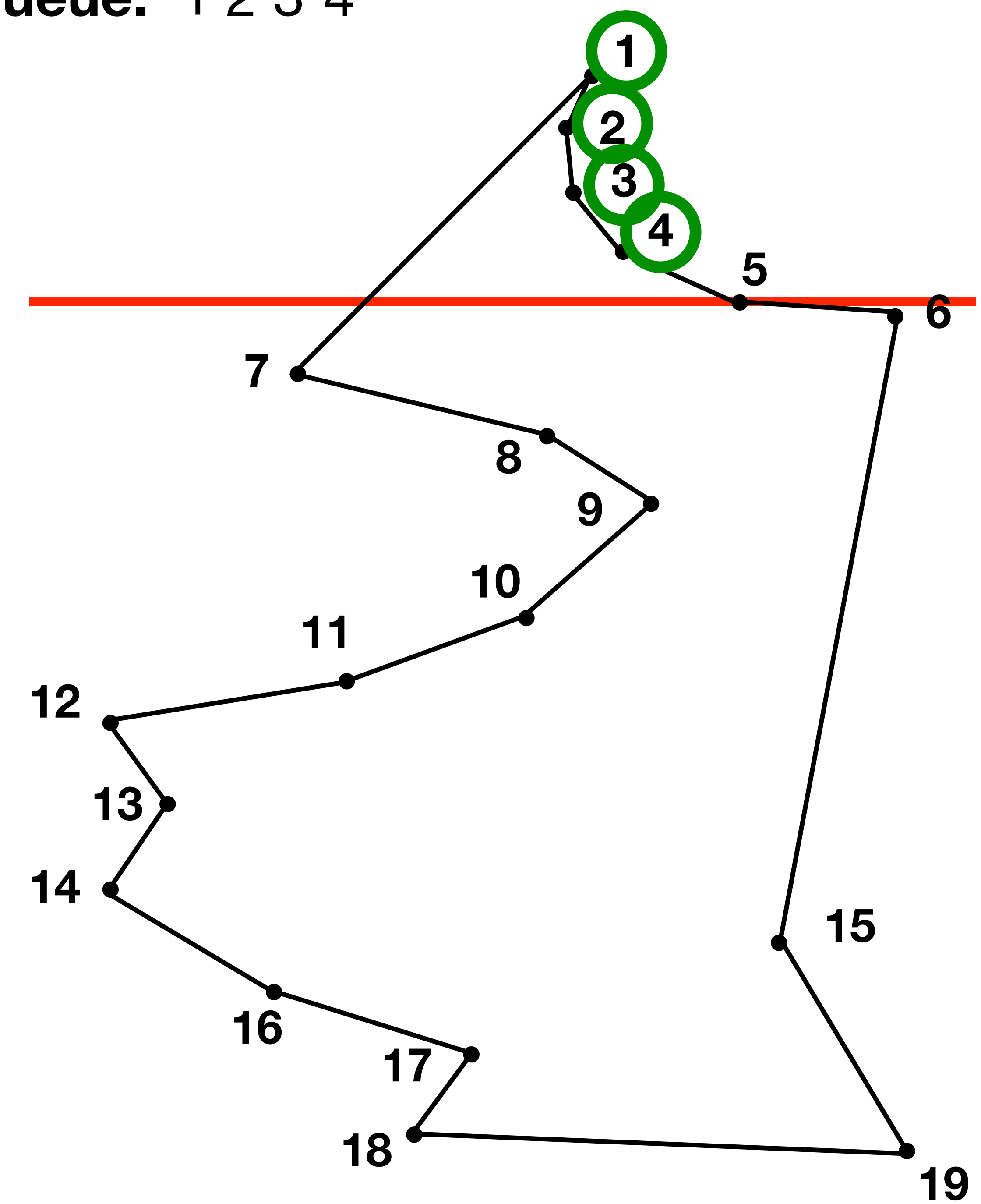
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2 3 4



Algorithm 5.16

Input: A y-monotone polygon P

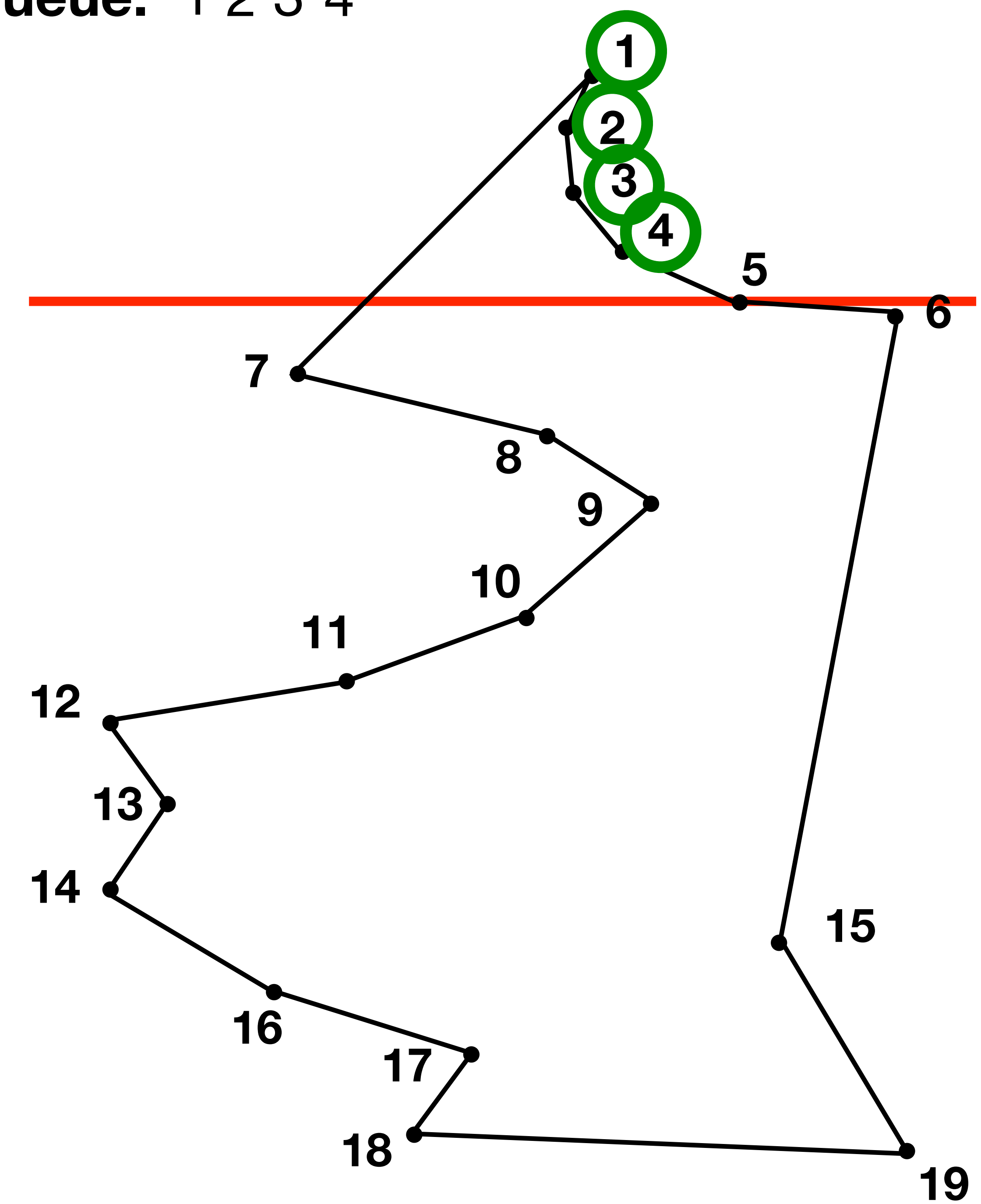
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 1 2 3 4



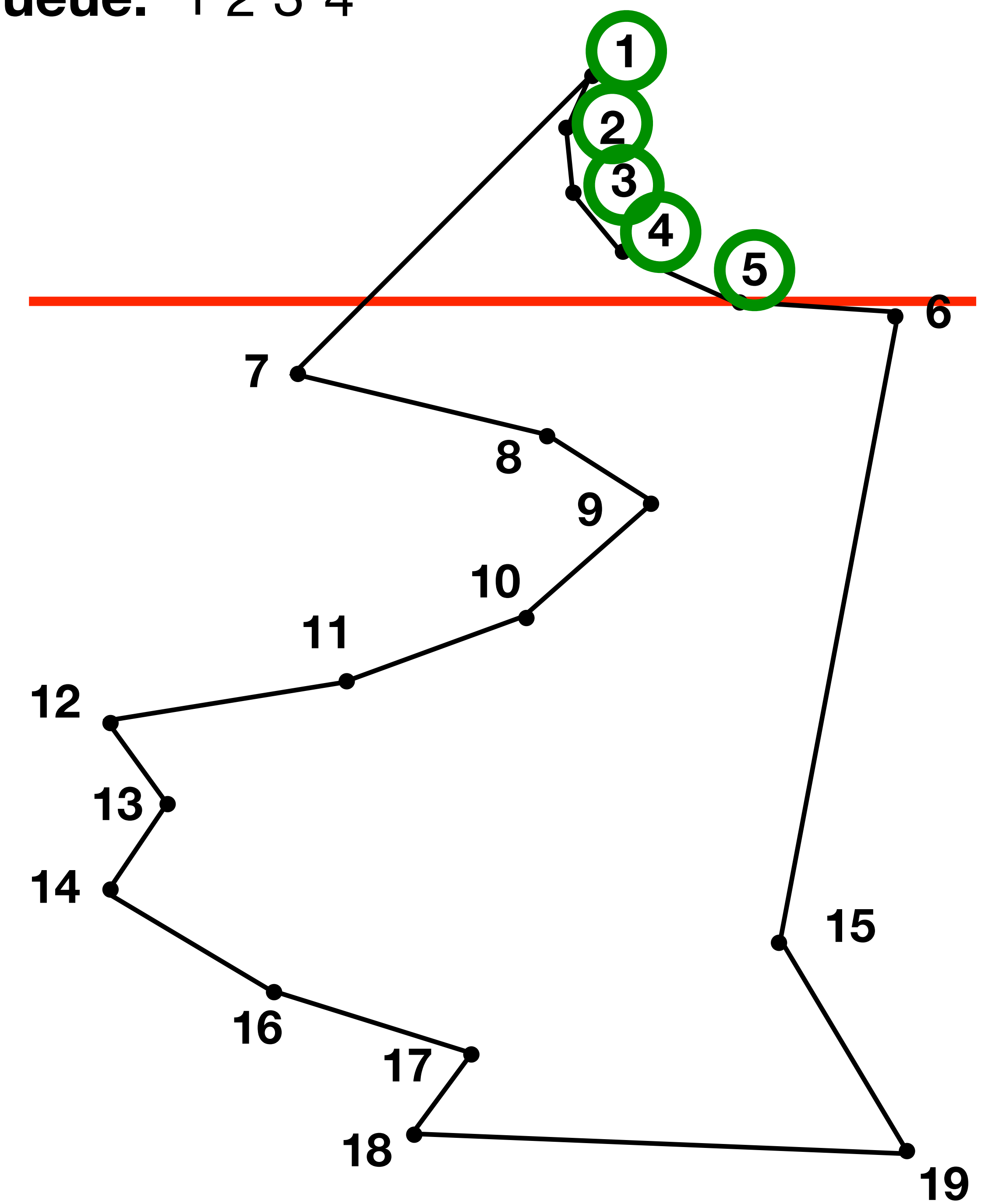
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 1 2 3 4



Algorithm 5.16

Input: A y-monotone polygon P

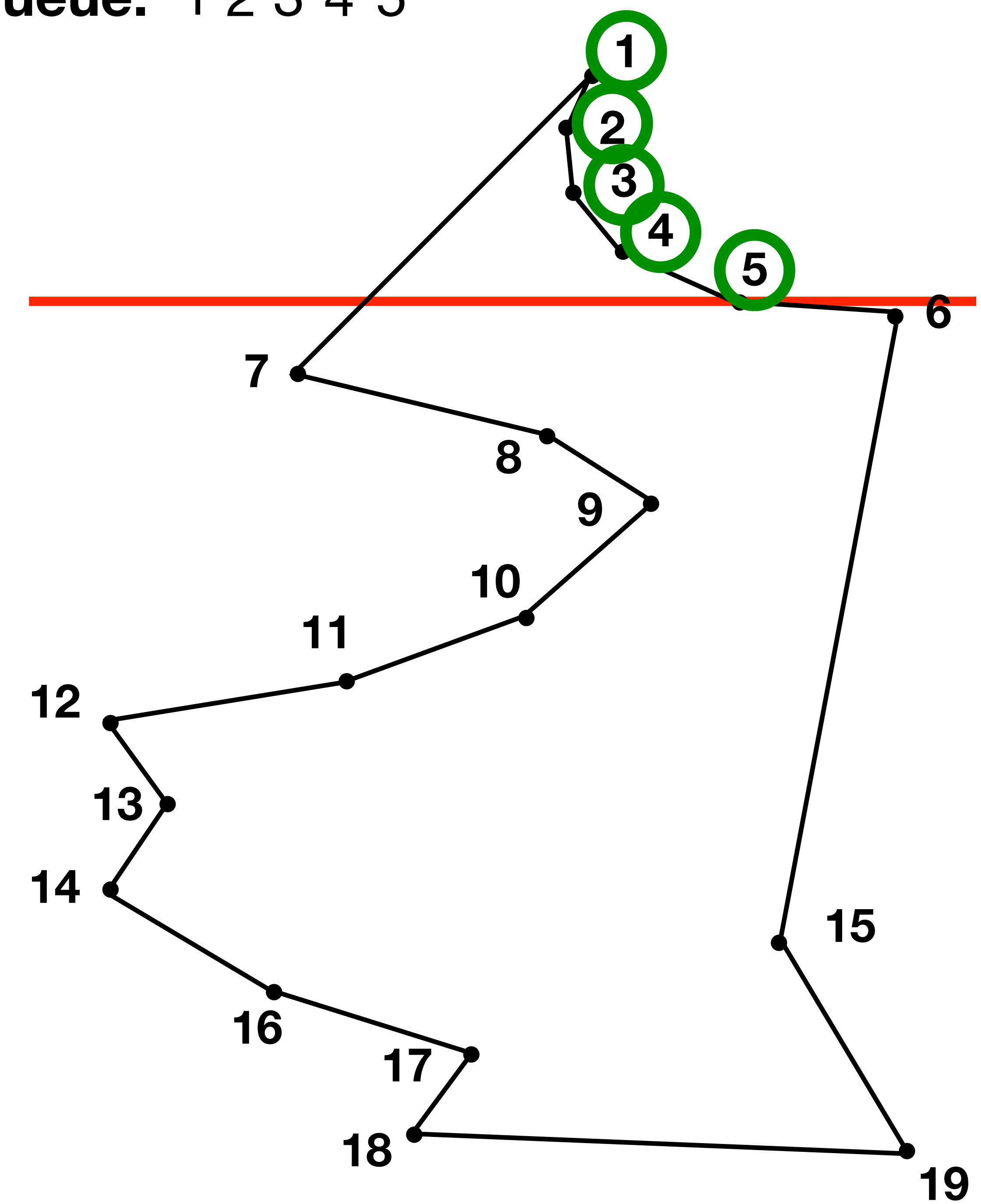
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5



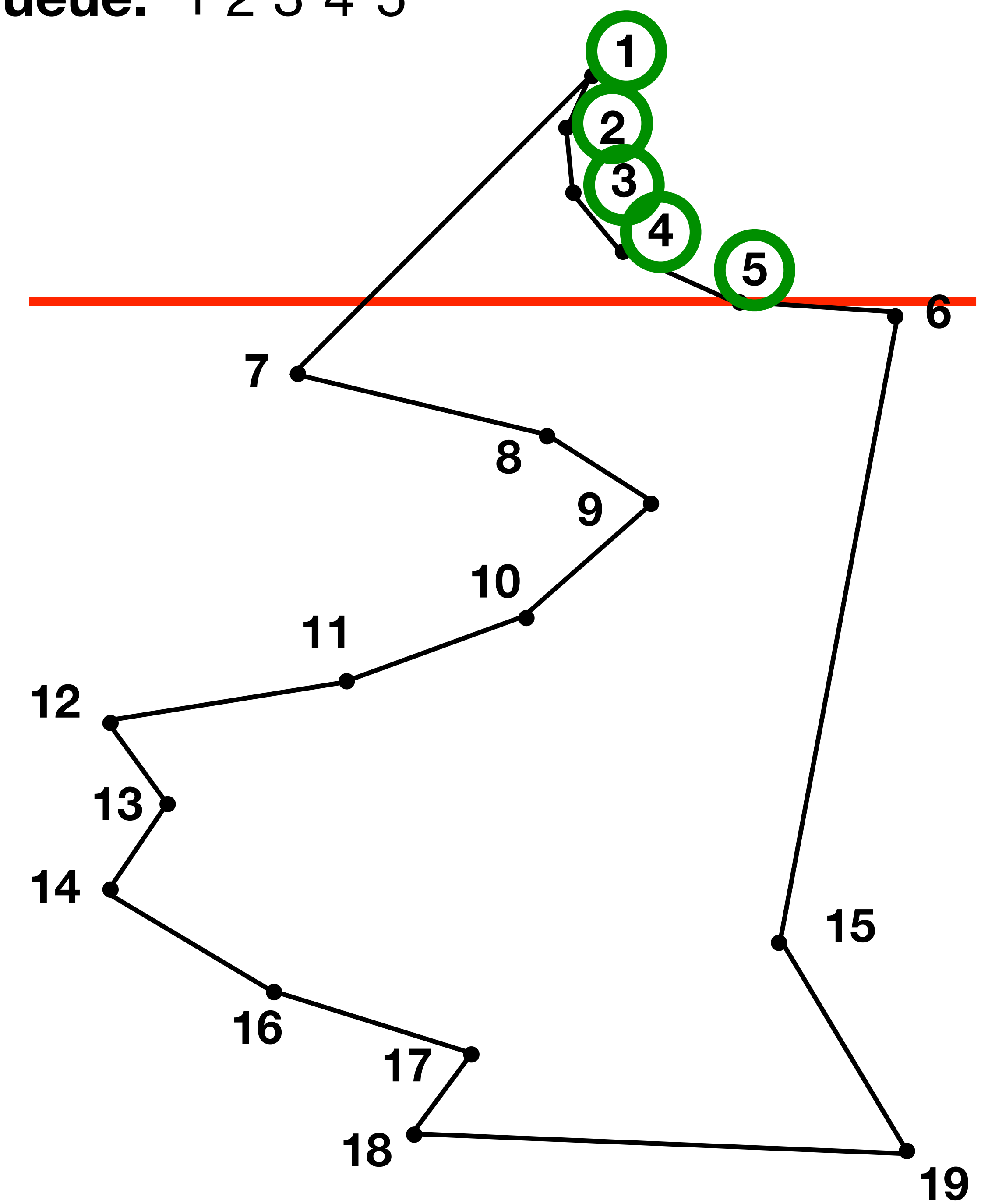
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5



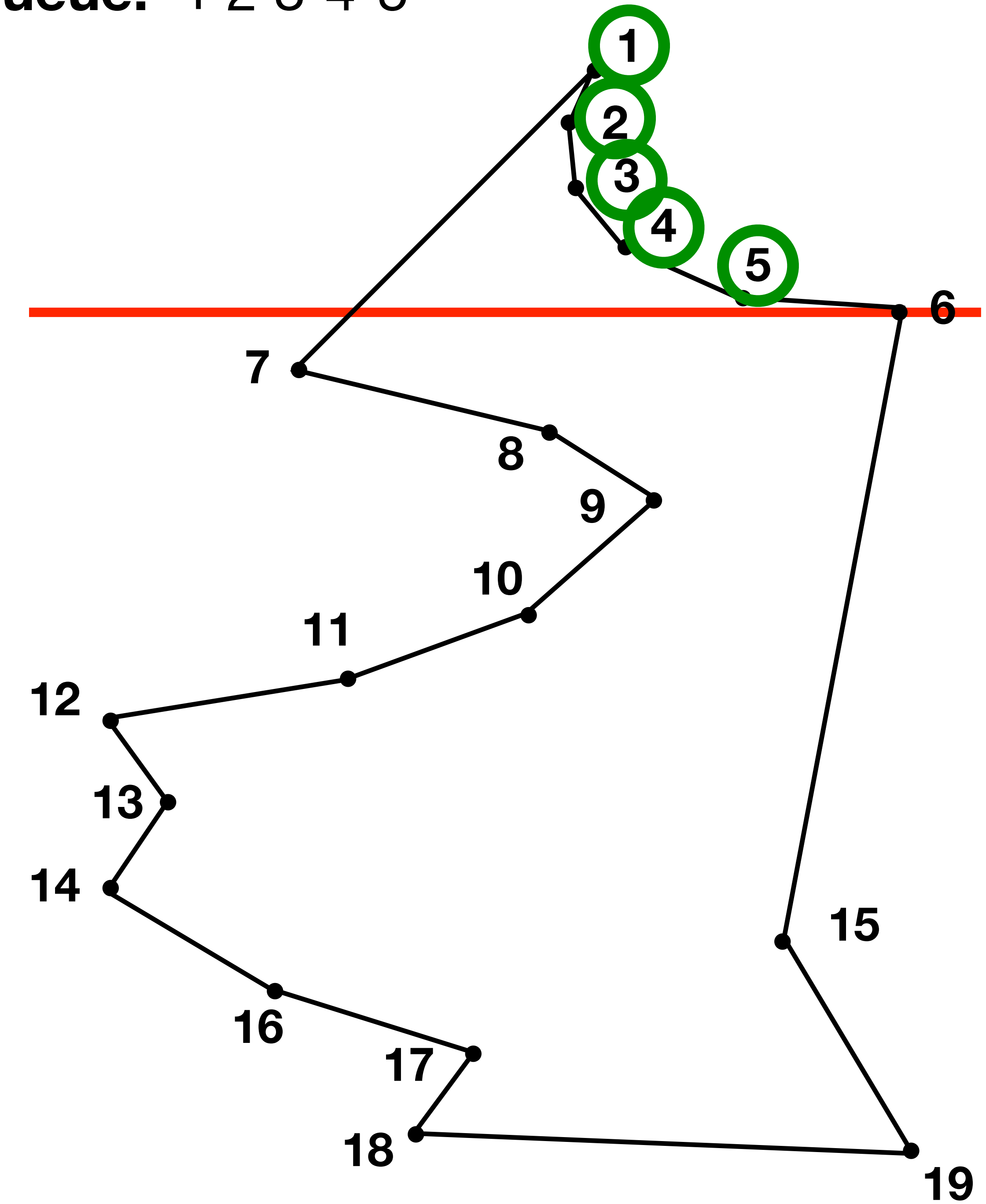
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5



Algorithm 5.16

Input: A y-monotone polygon P

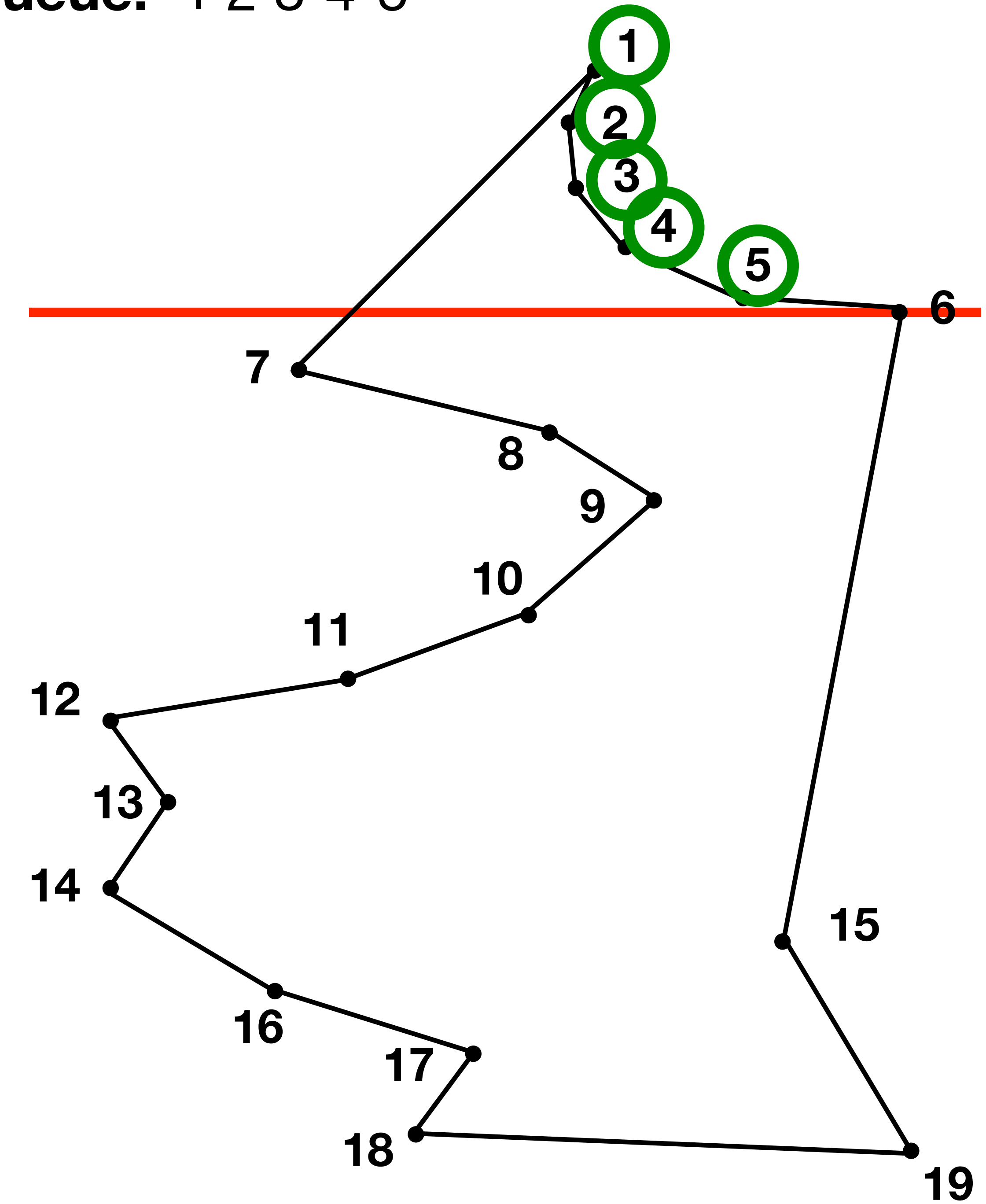
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 1 2 3 4 5



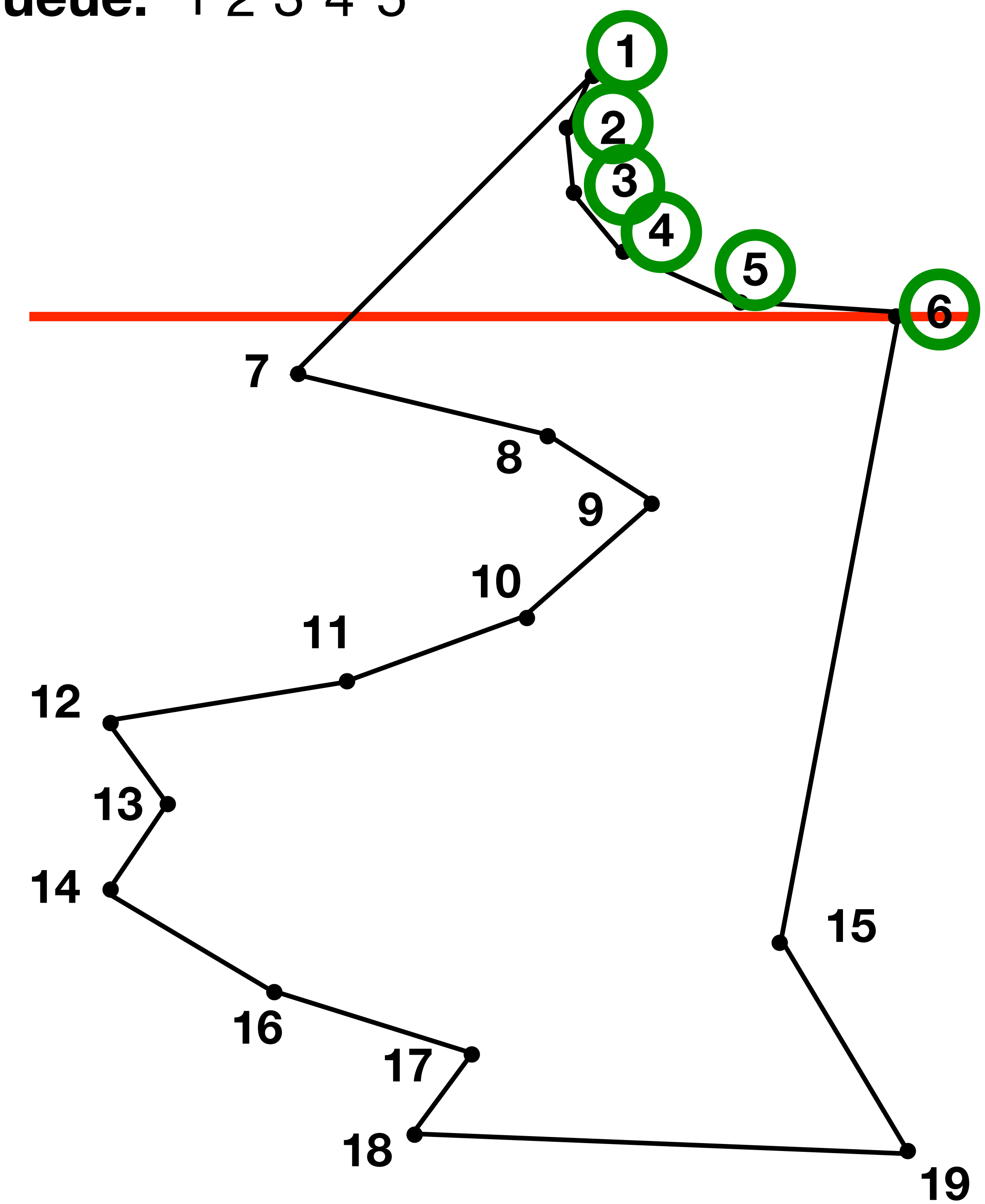
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5



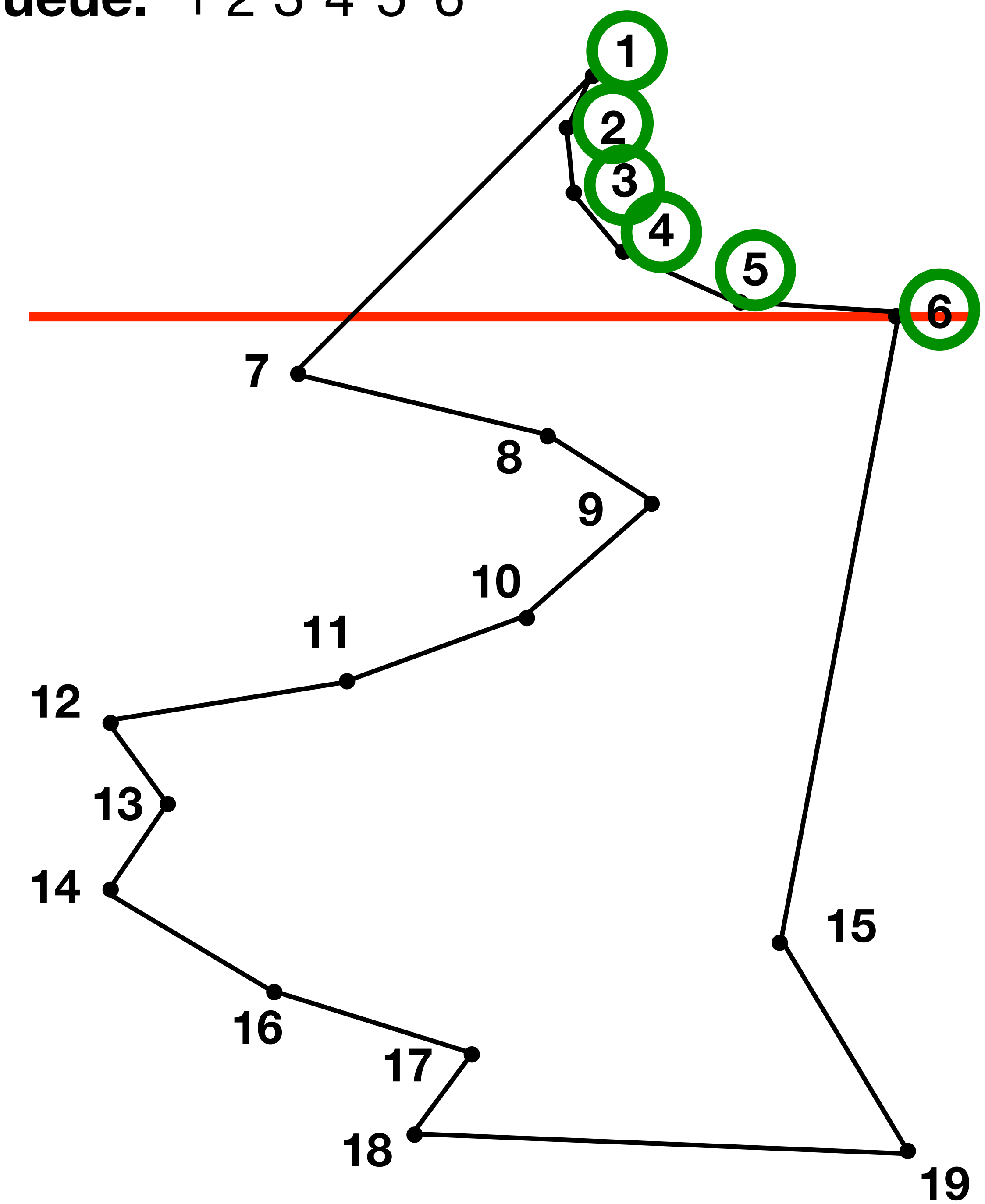
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 1 2 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

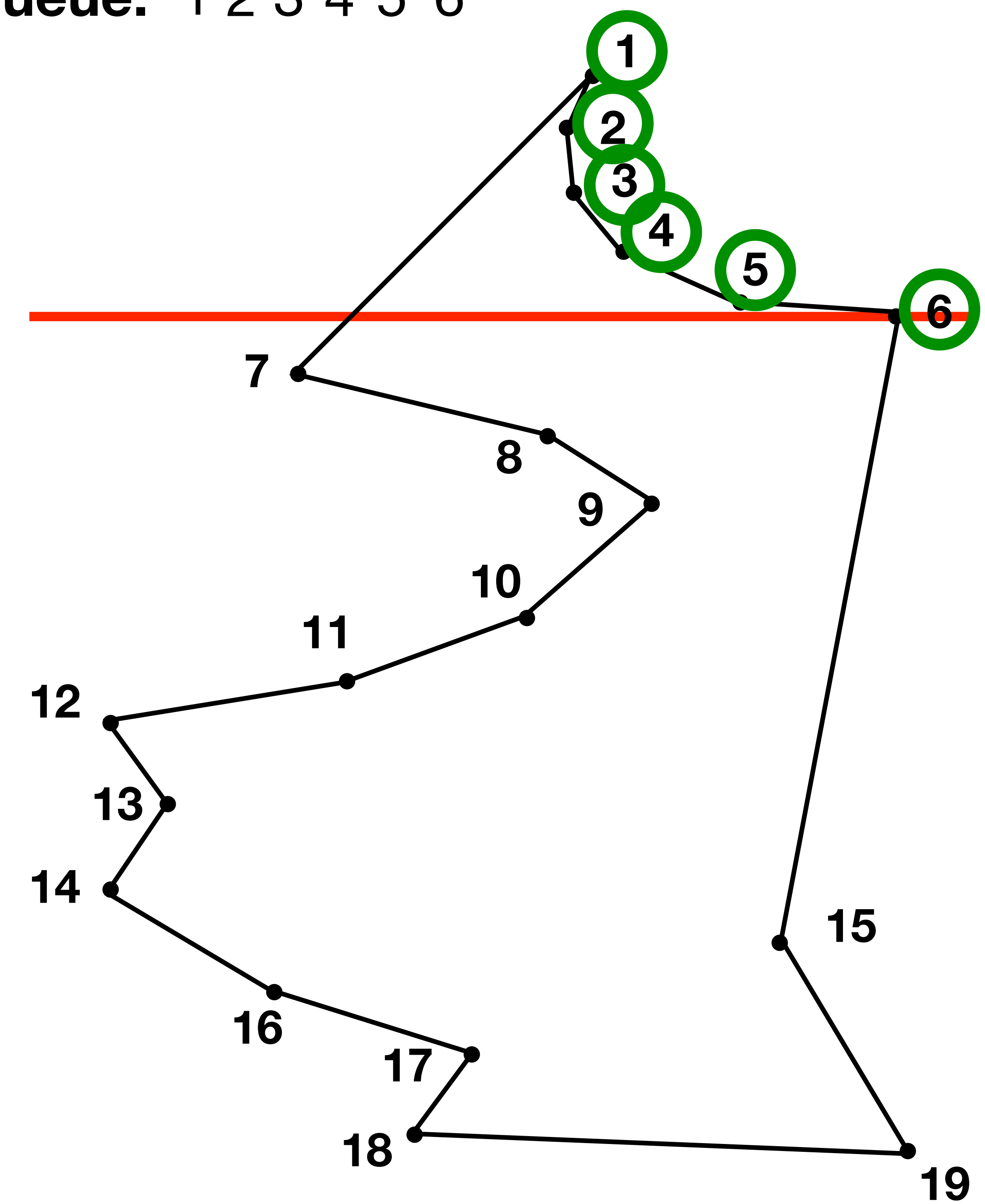
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5 6



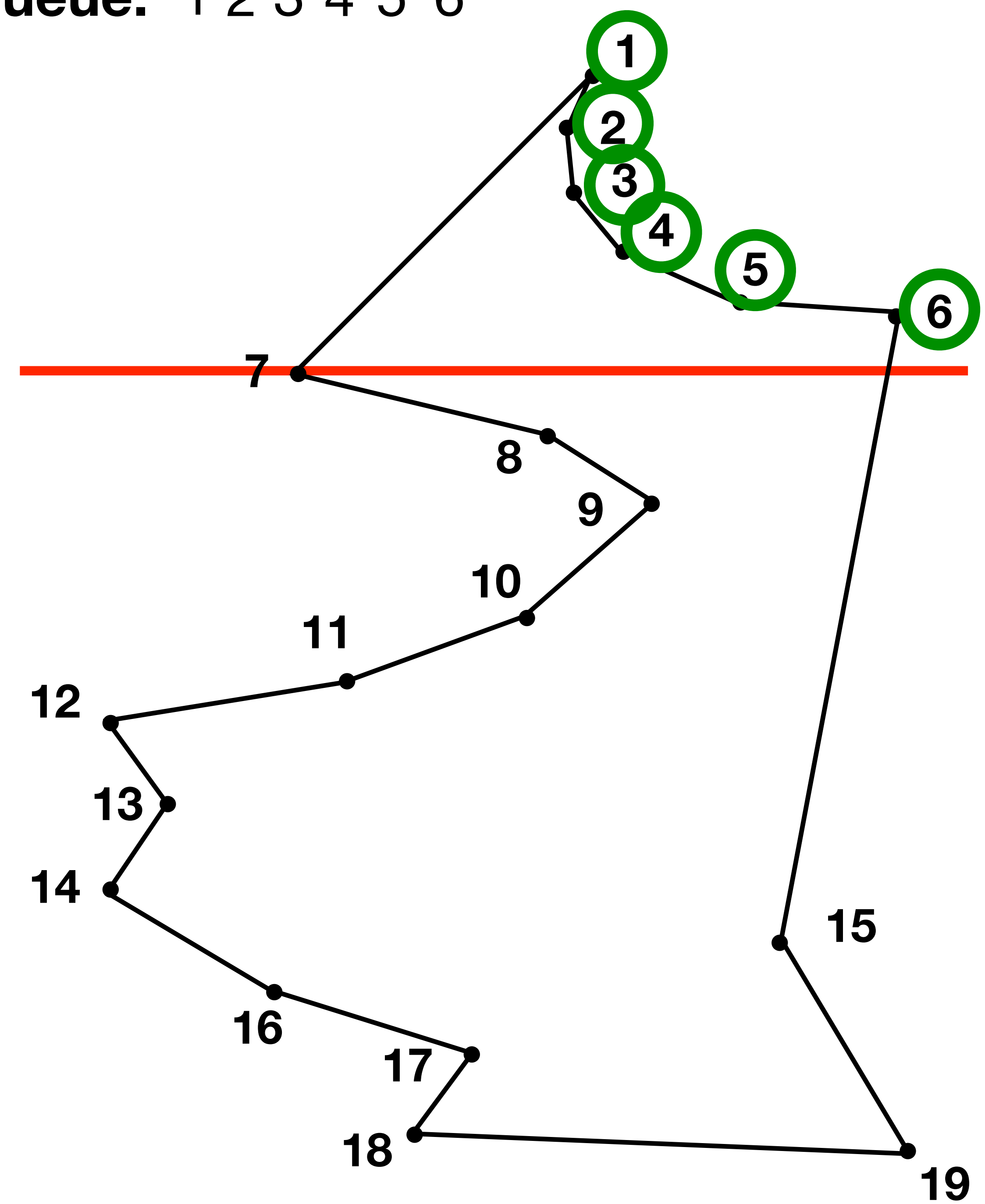
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5 6



Technische Universität Braunschweig

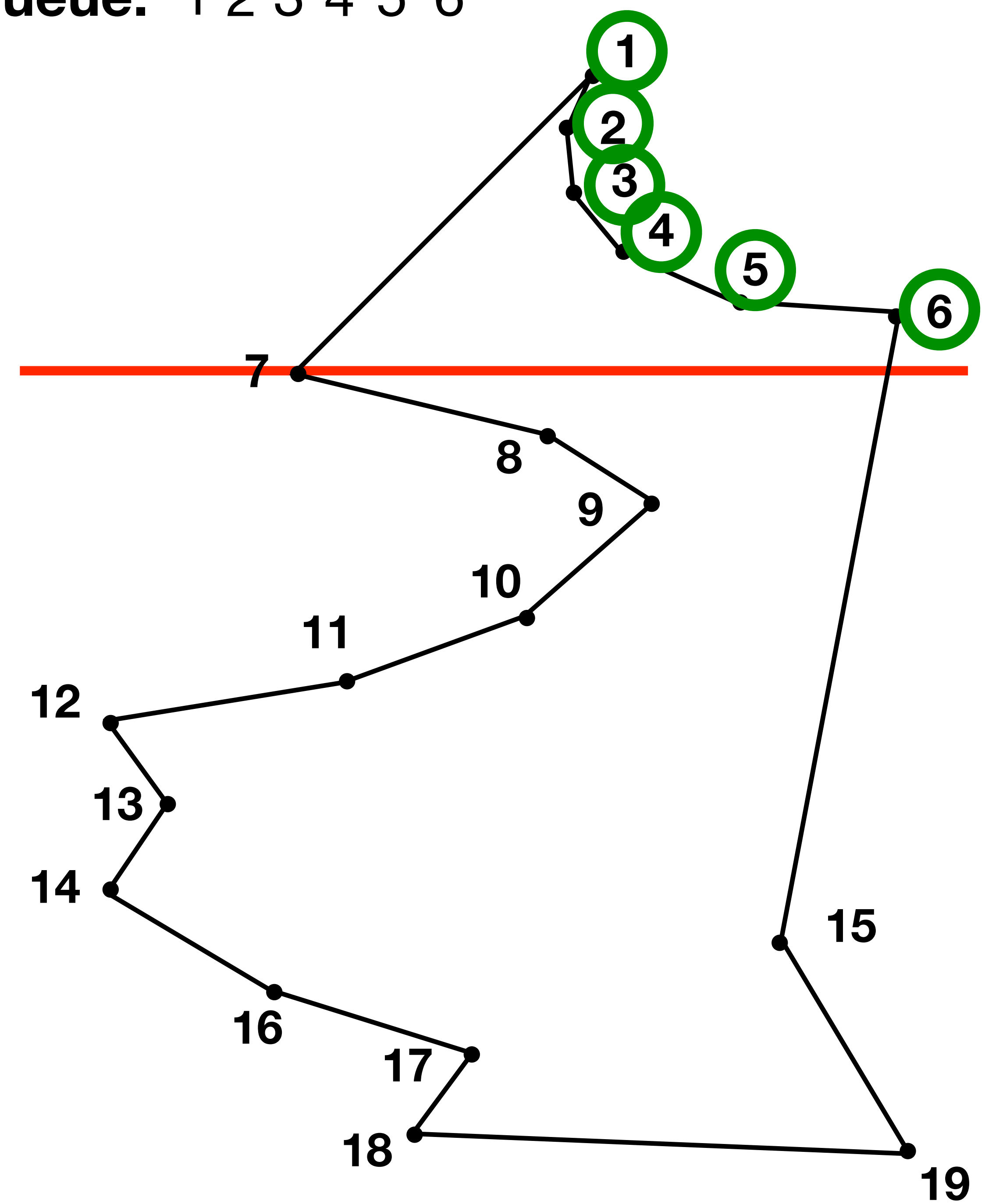
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 1 2 3 4 5 6



Technische Universität Braunschweig

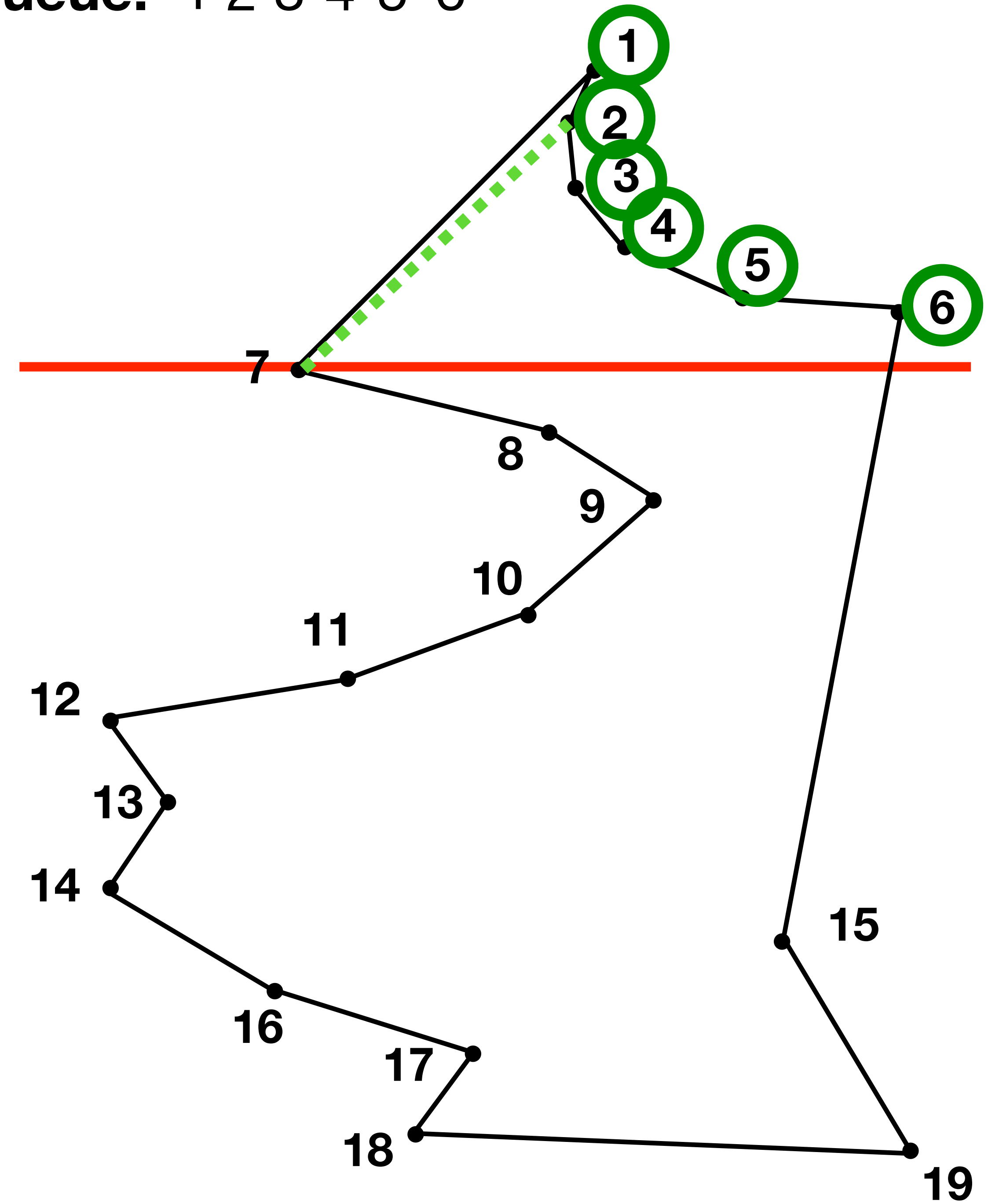
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 1 2 3 4 5 6



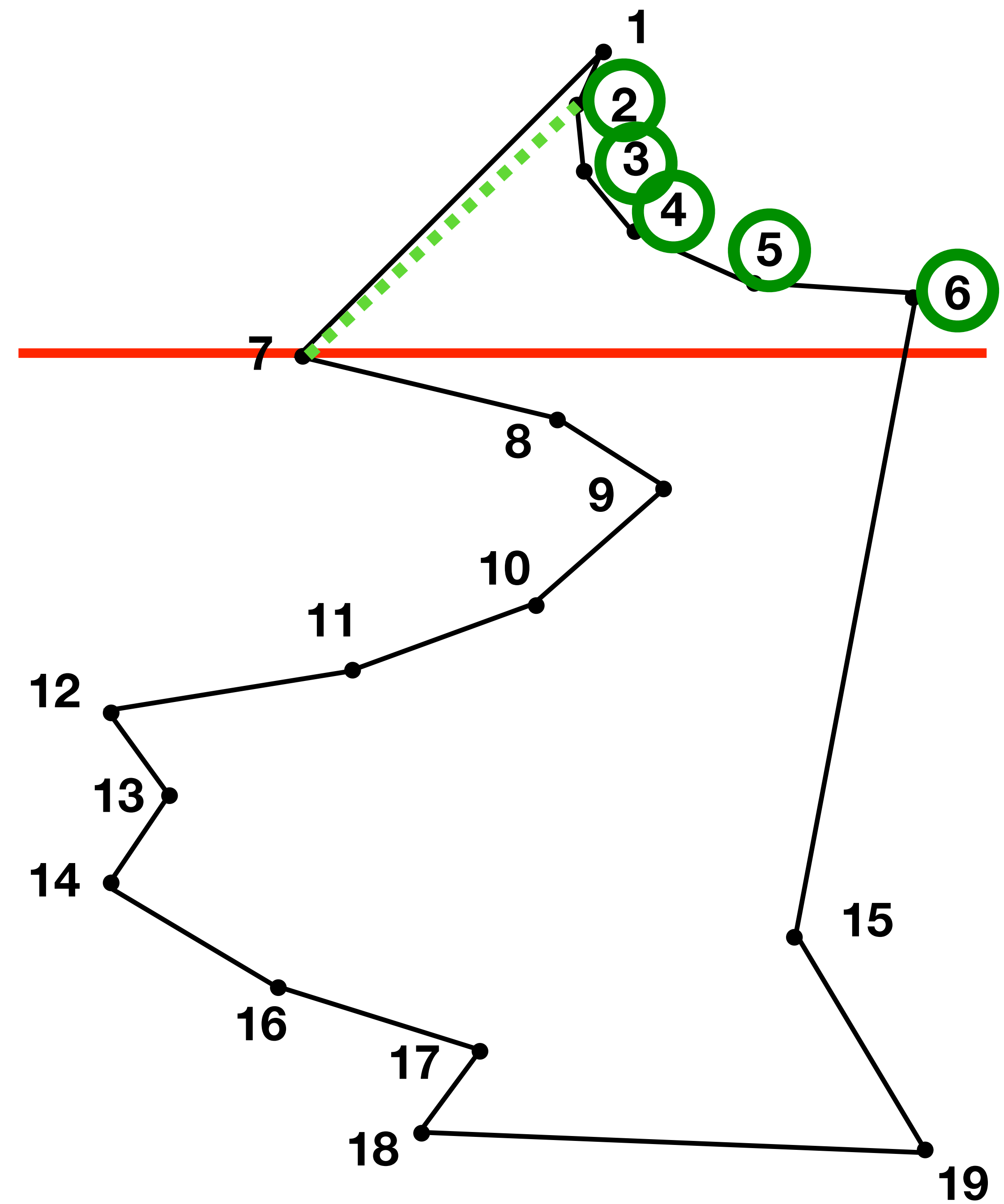
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 1 2 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

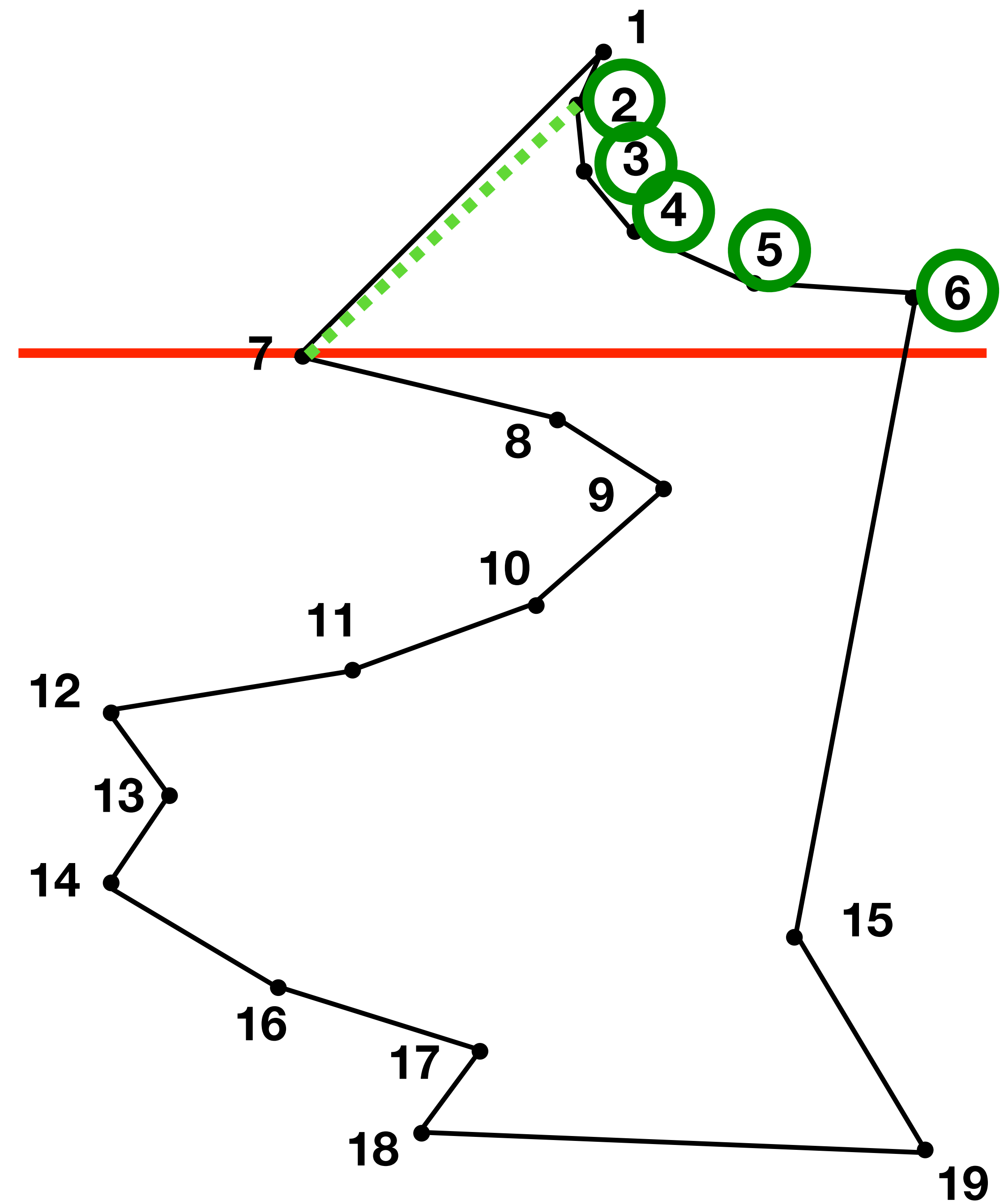
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 2 3 4 5 6



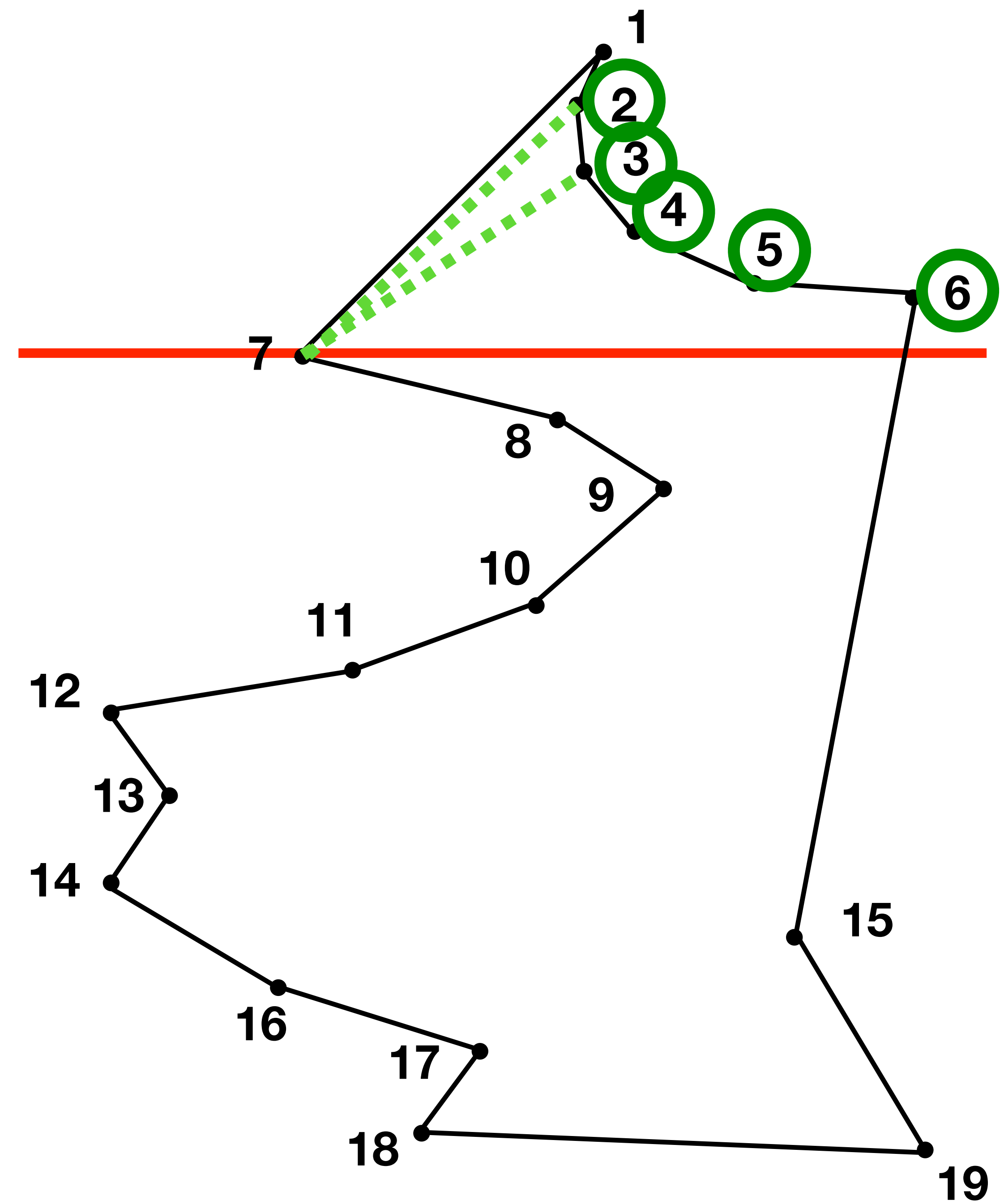
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 2 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

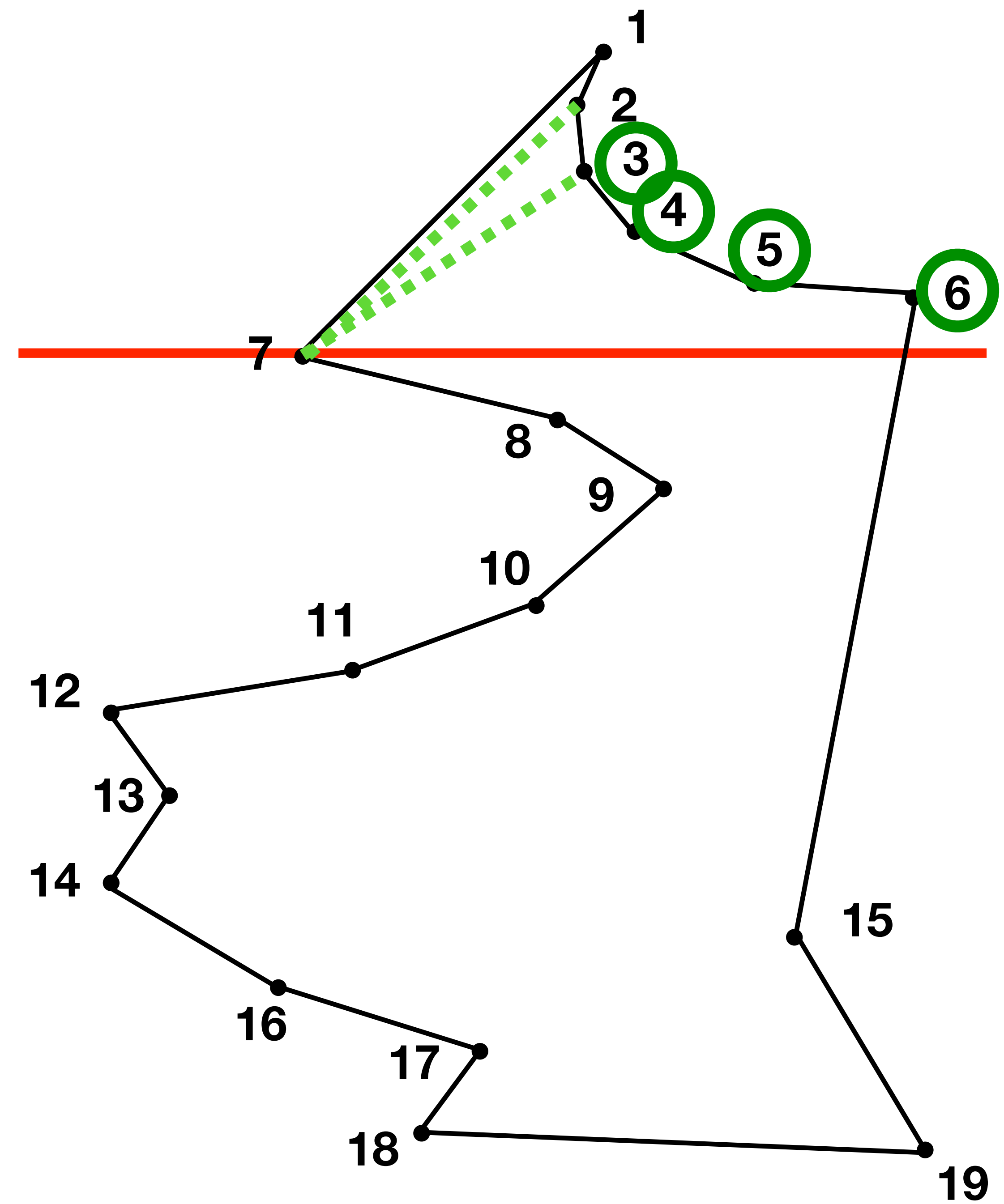
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 2 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

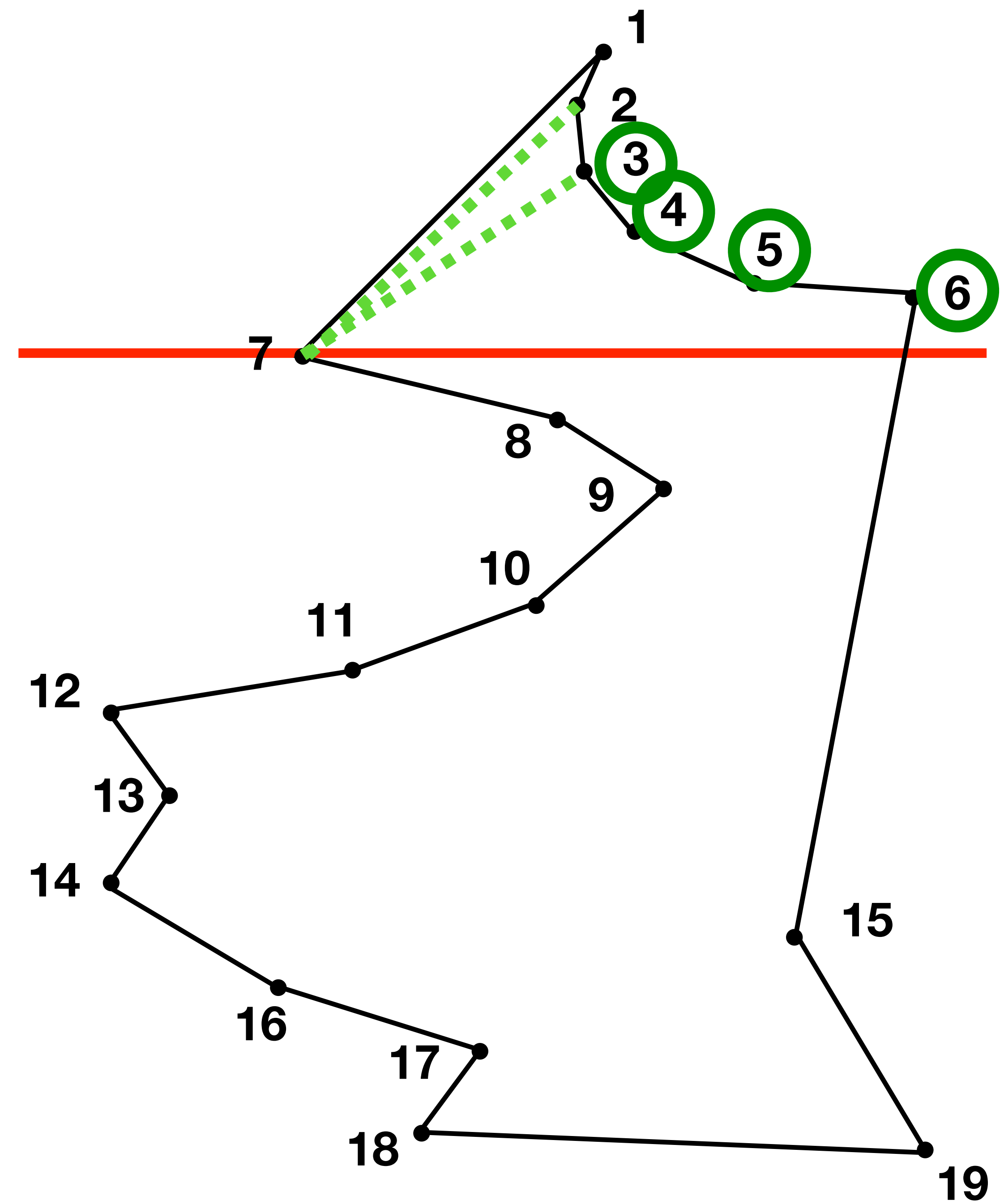
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 3 4 5 6



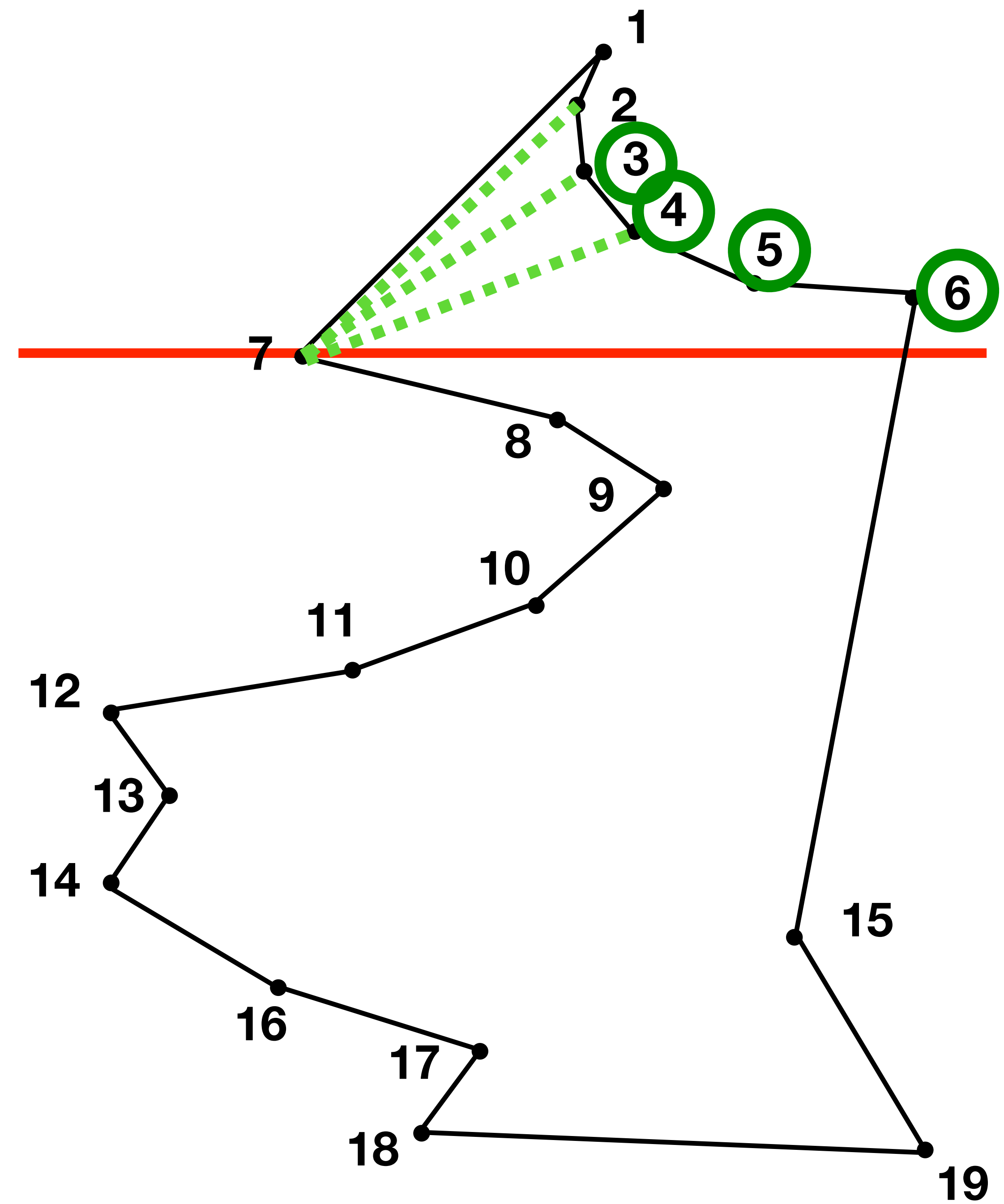
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

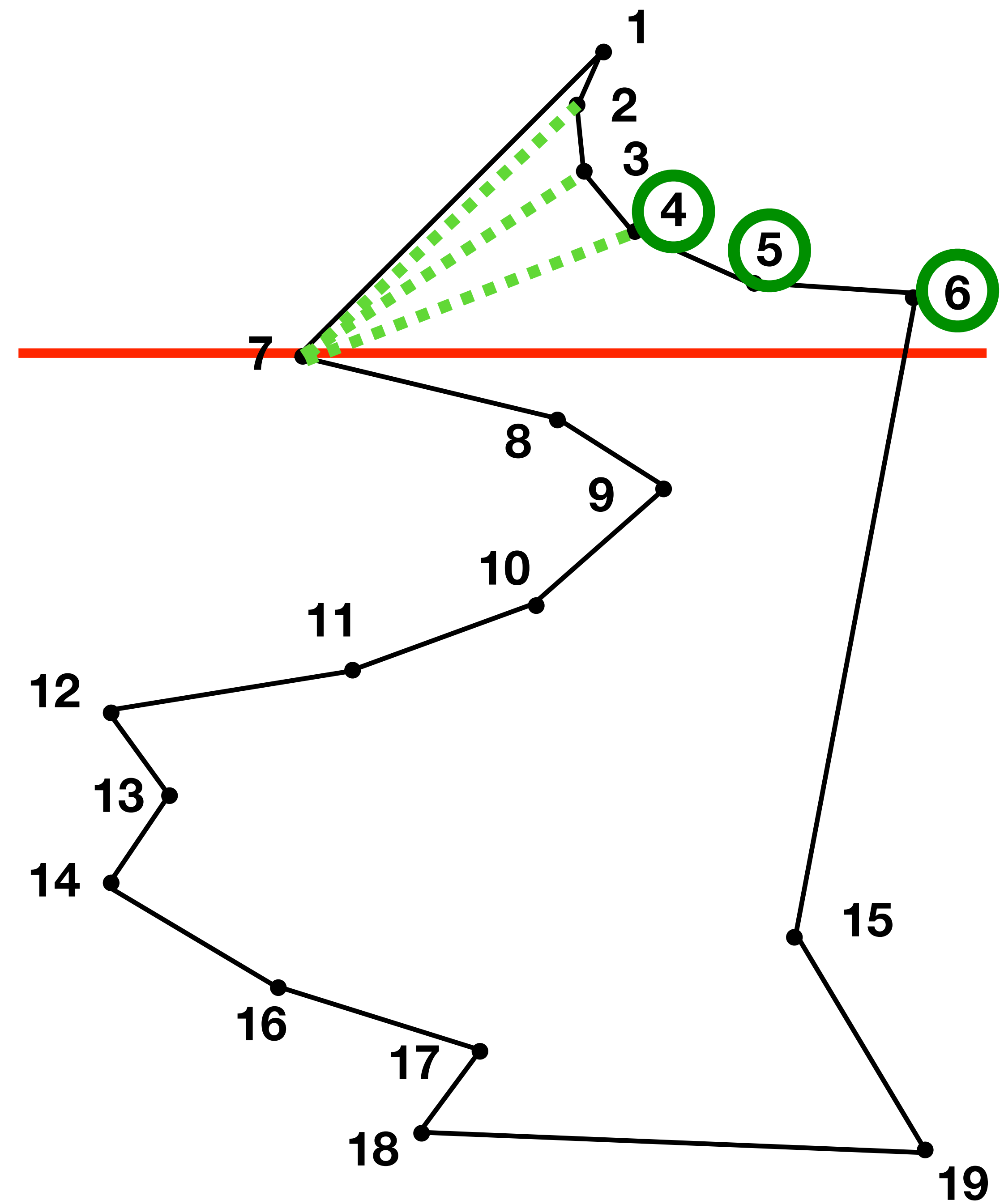
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 3 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

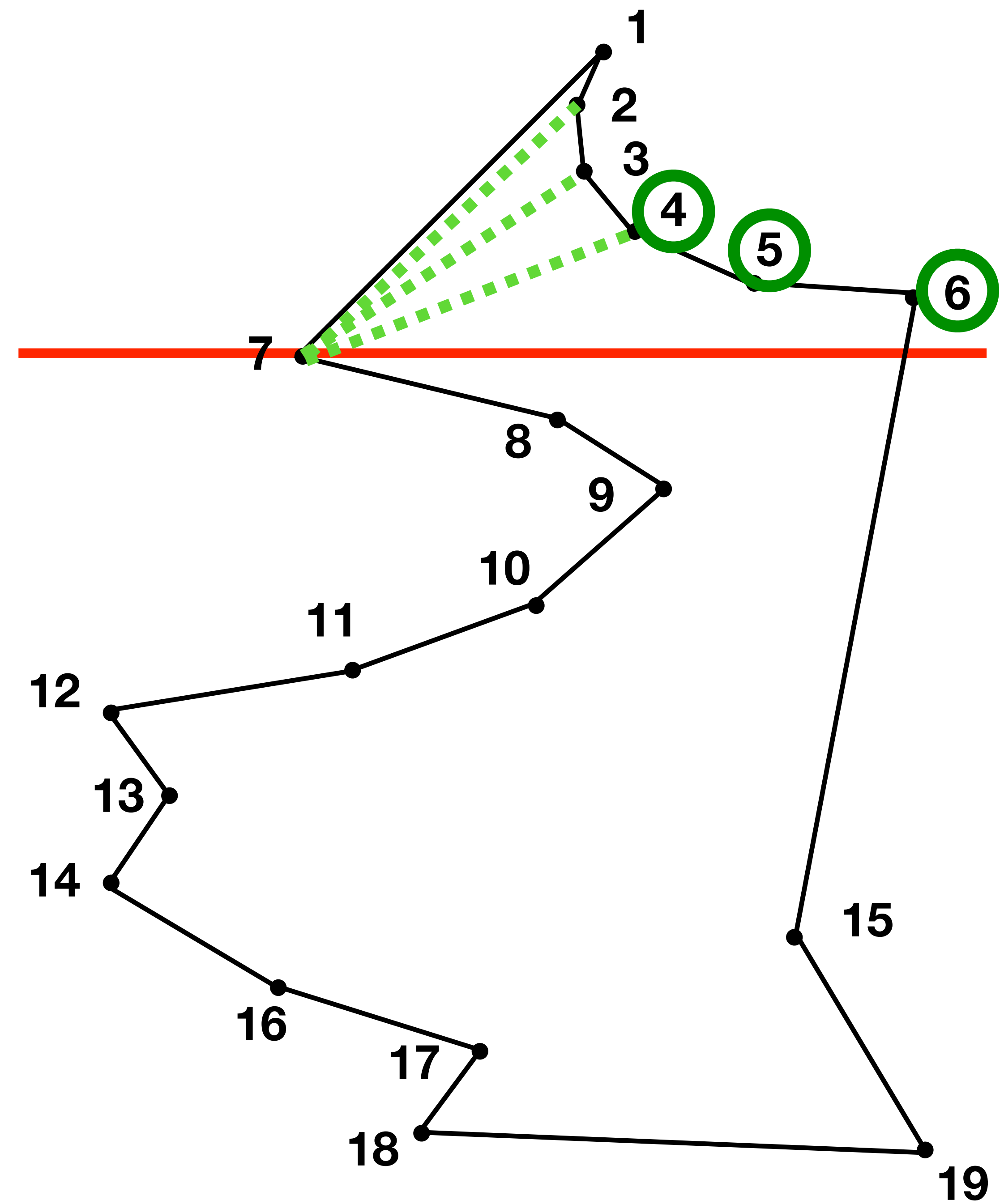
1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue
   - 3.2 If same chain, predecessor reflex:
     - Add to queue
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update

Chain

Add

Ear

Queue: 4 5 6



Algorithm 5.16

Input: A y-monotone polygon P

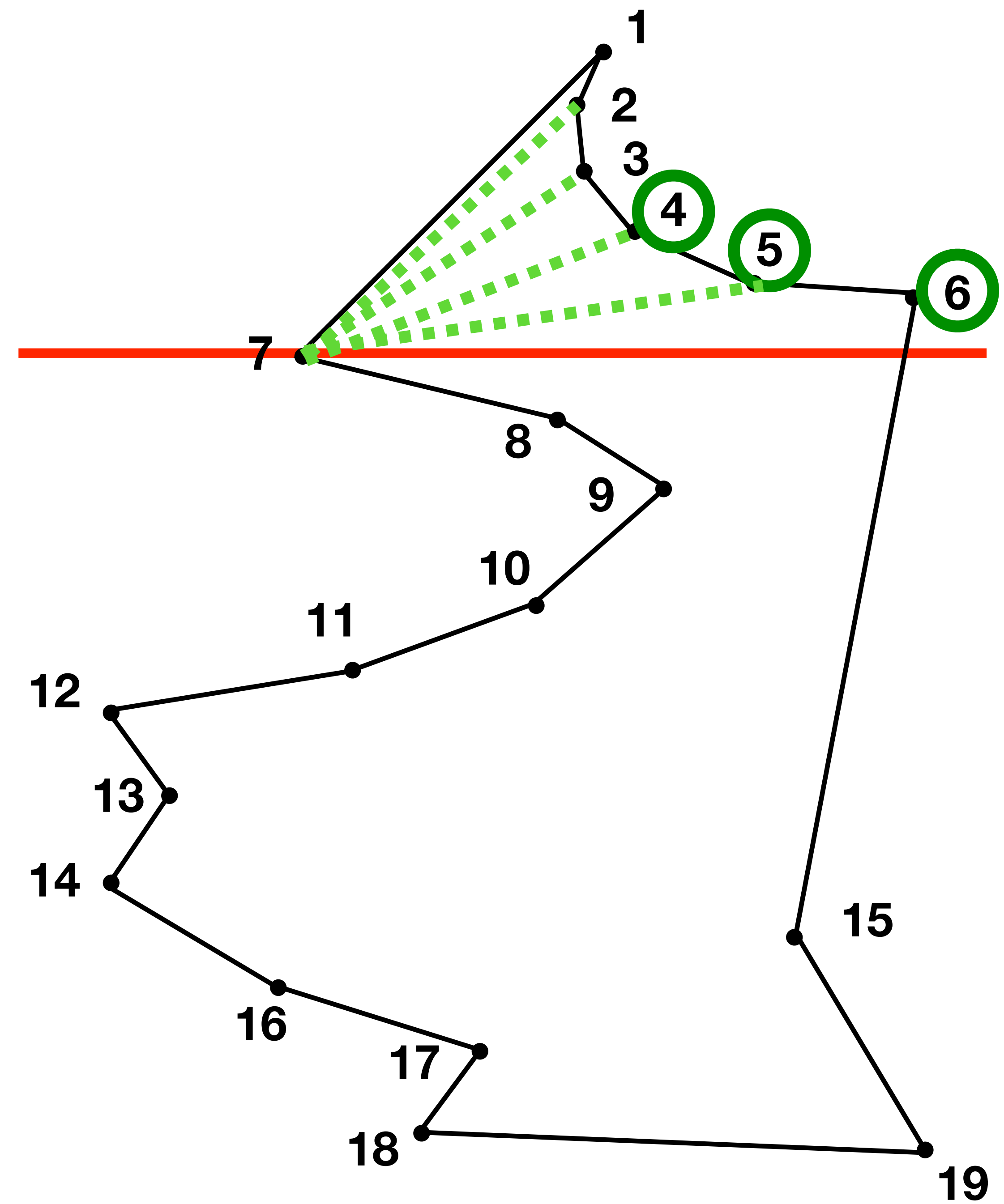
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 4 5 6



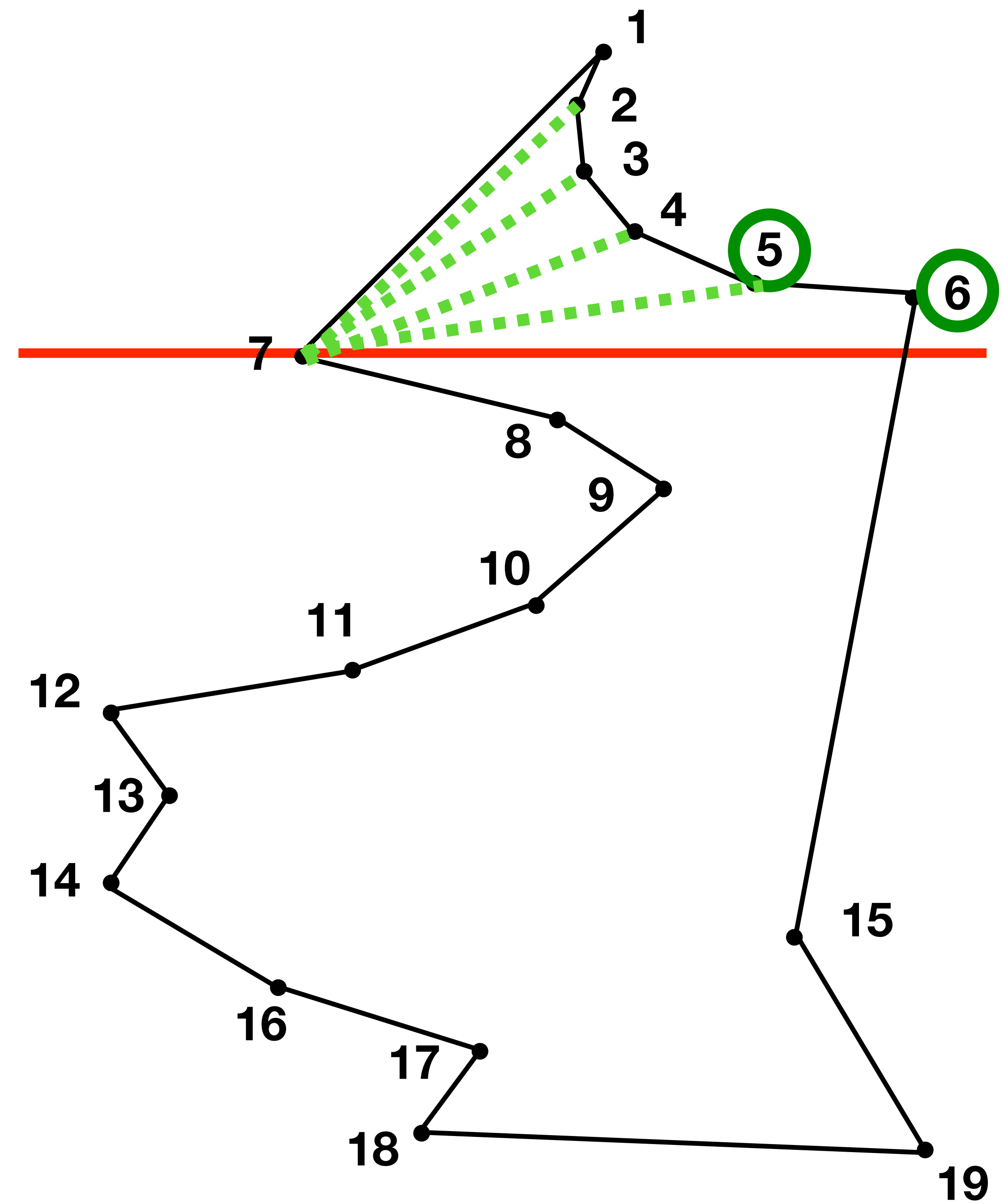
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 4 5 6



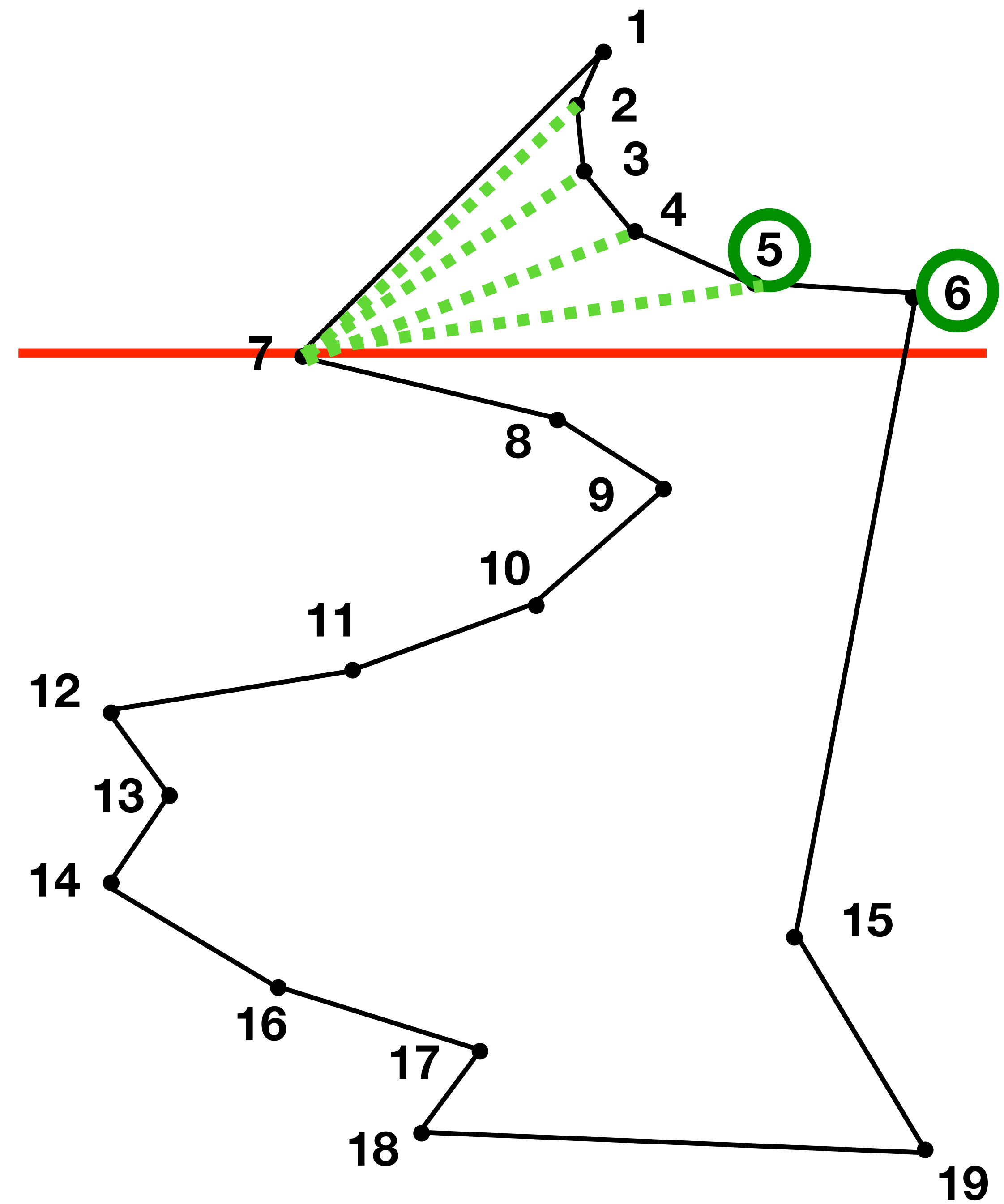
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 5 6



Algorithm 5.16

Input: A y-monotone polygon P

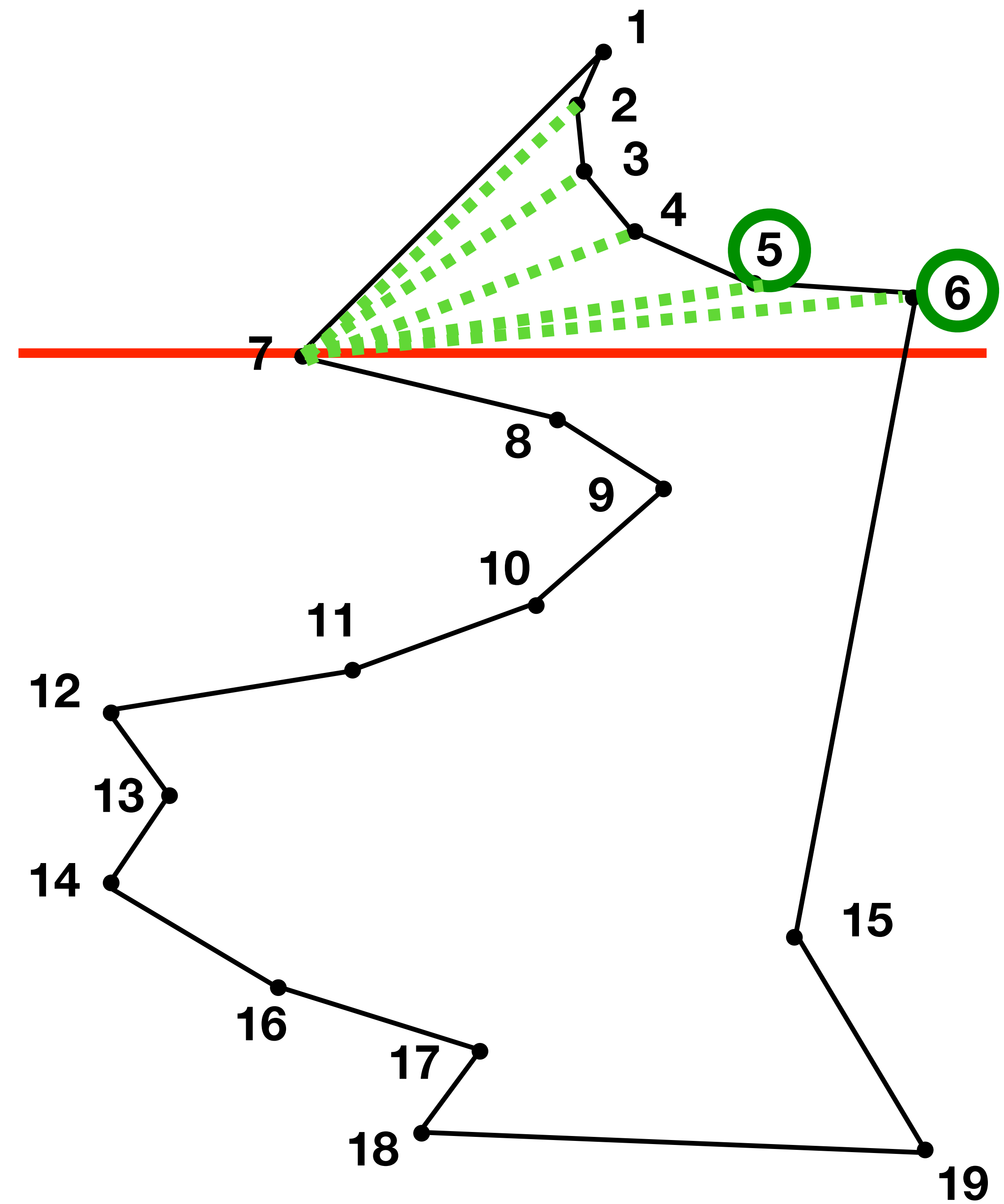
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 5 6



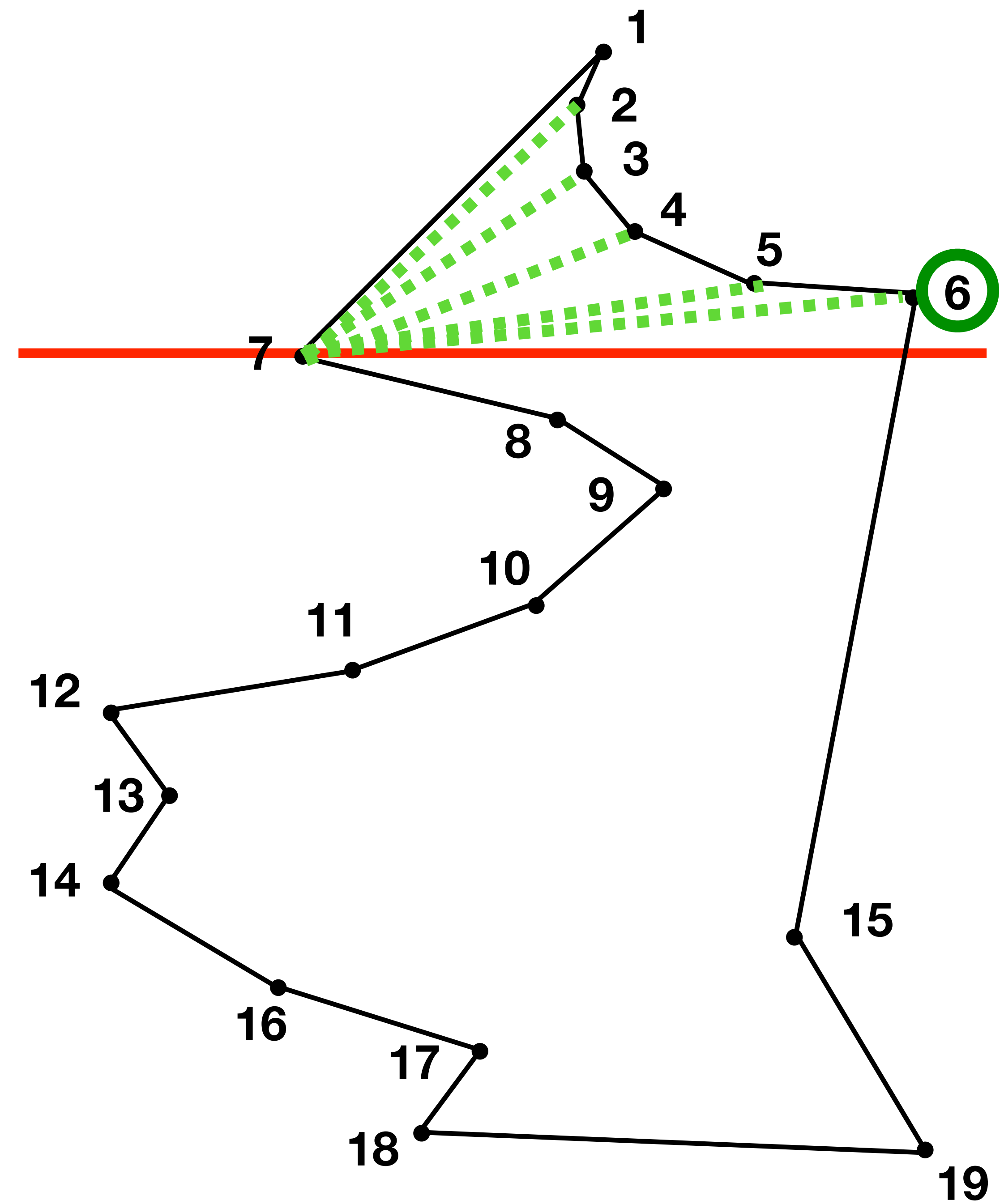
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 5 6



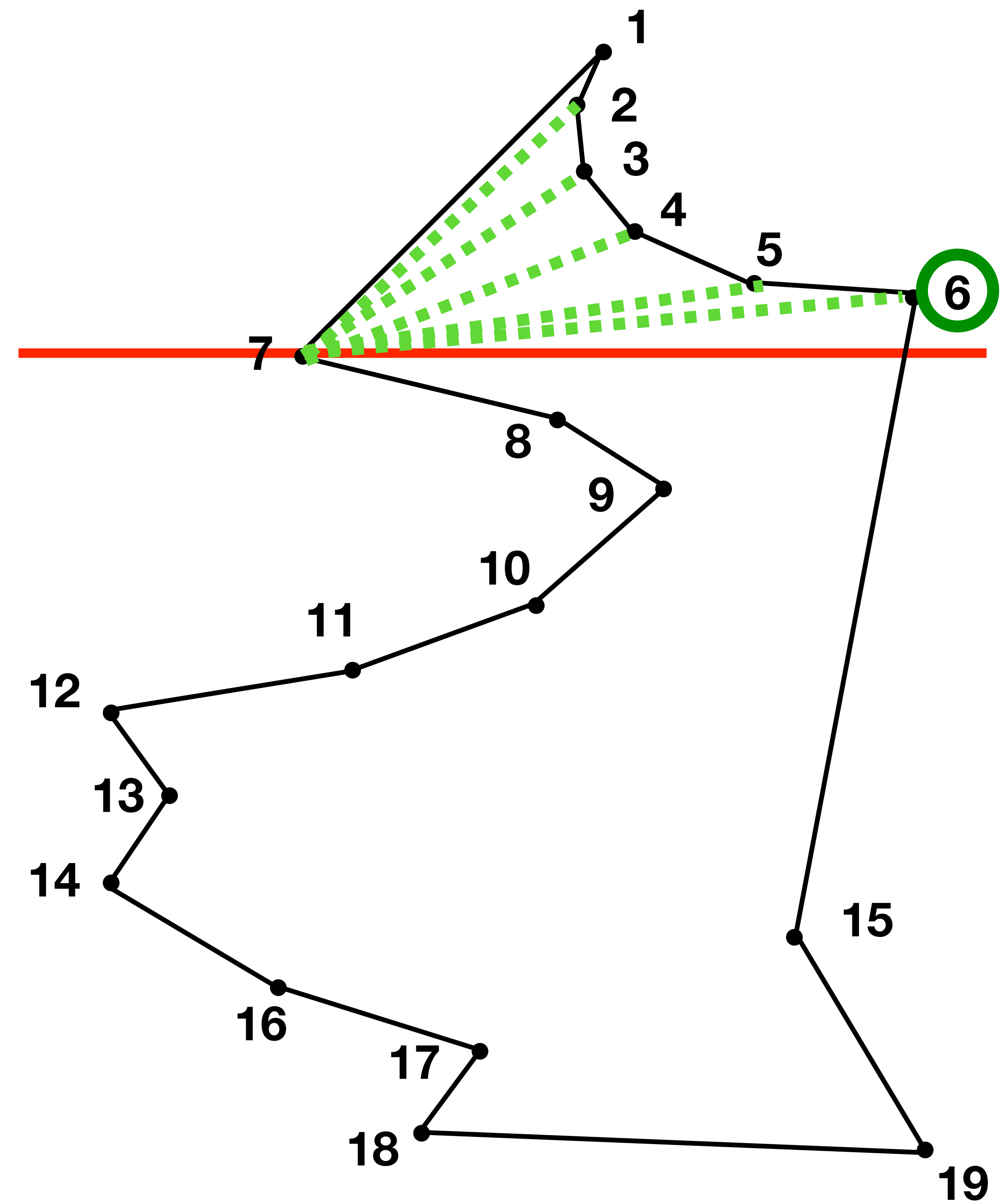
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6



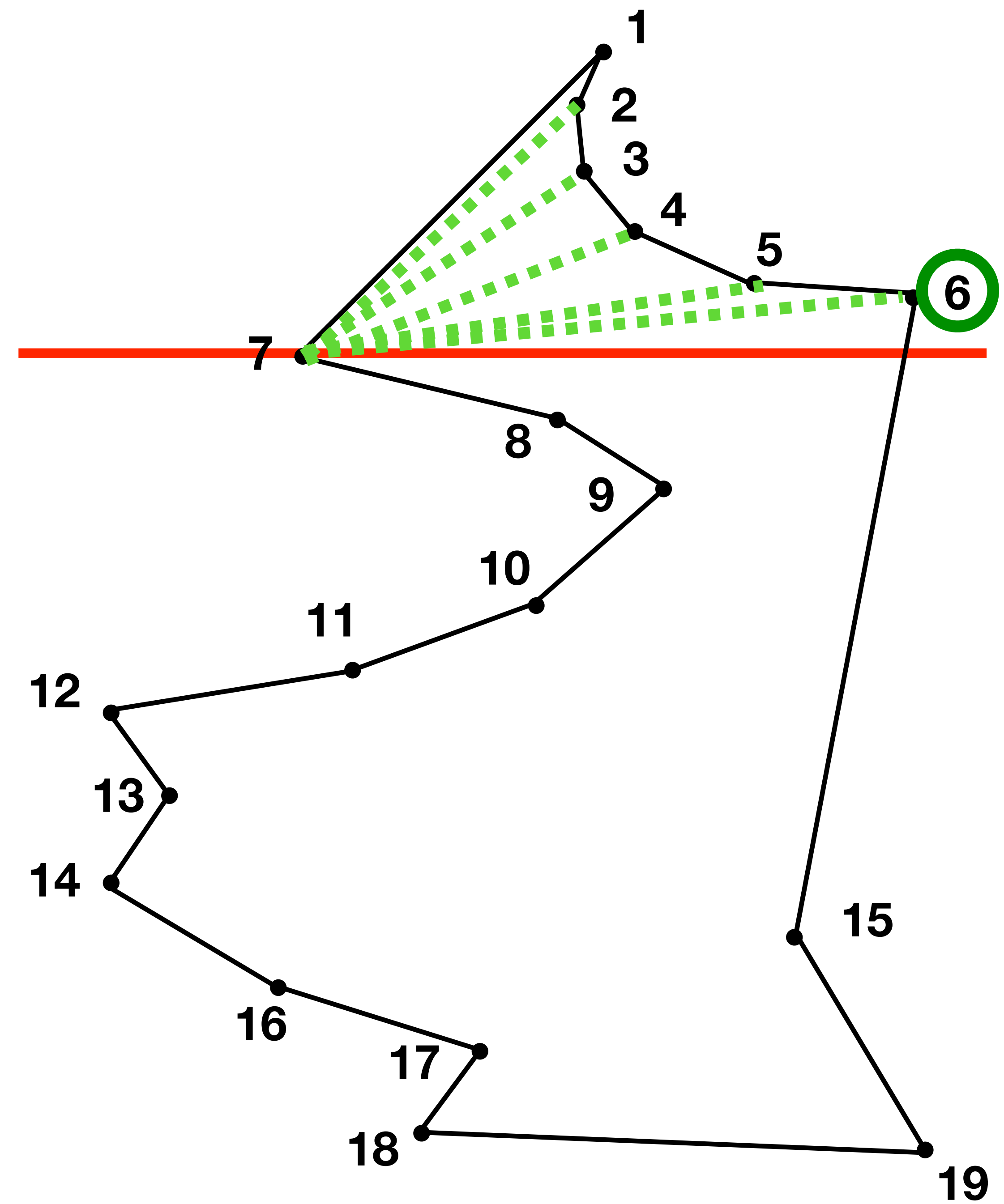
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6



Algorithm 5.16

Input: A y-monotone polygon P

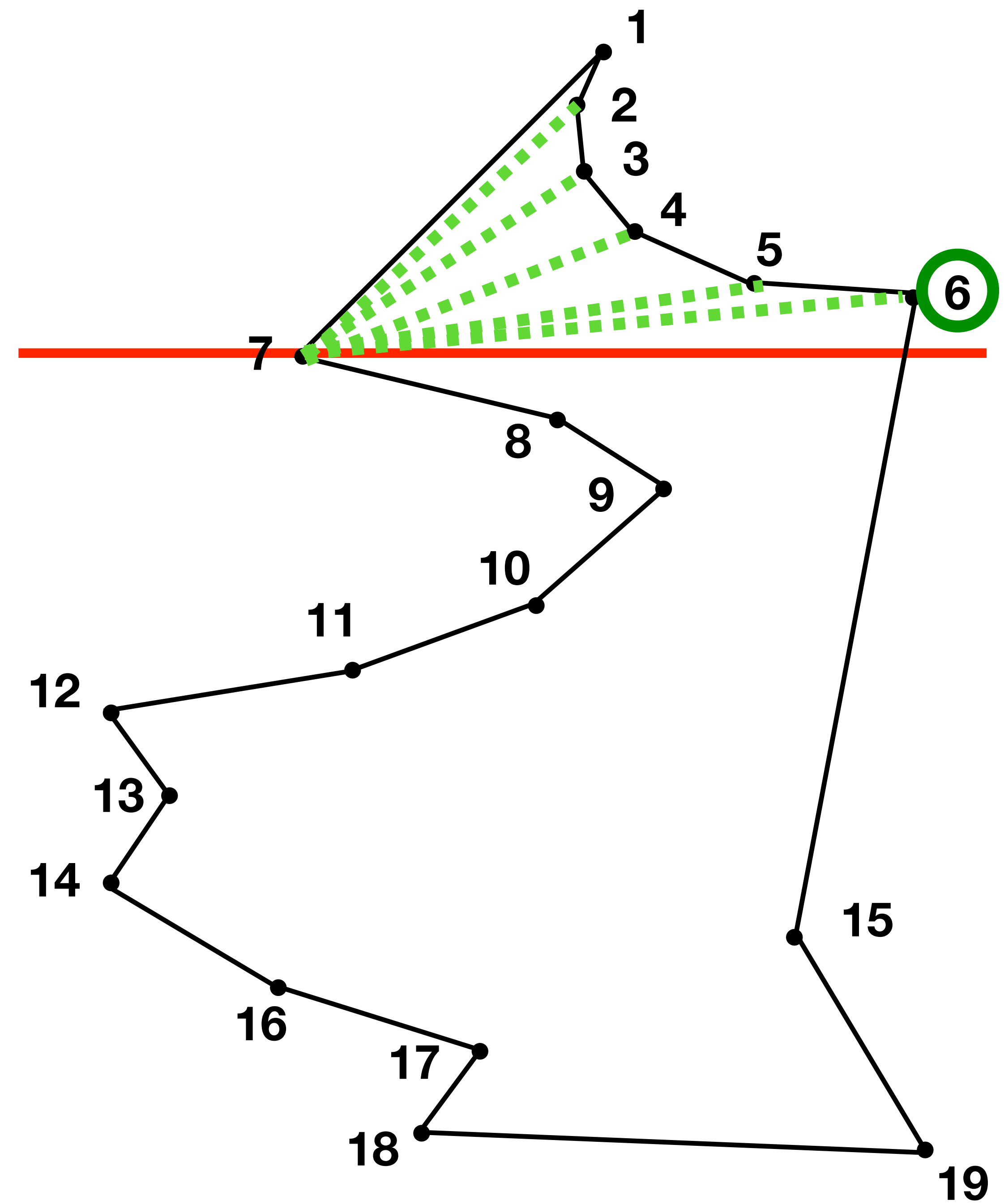
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 7



Algorithm 5.16

Input: A y-monotone polygon P

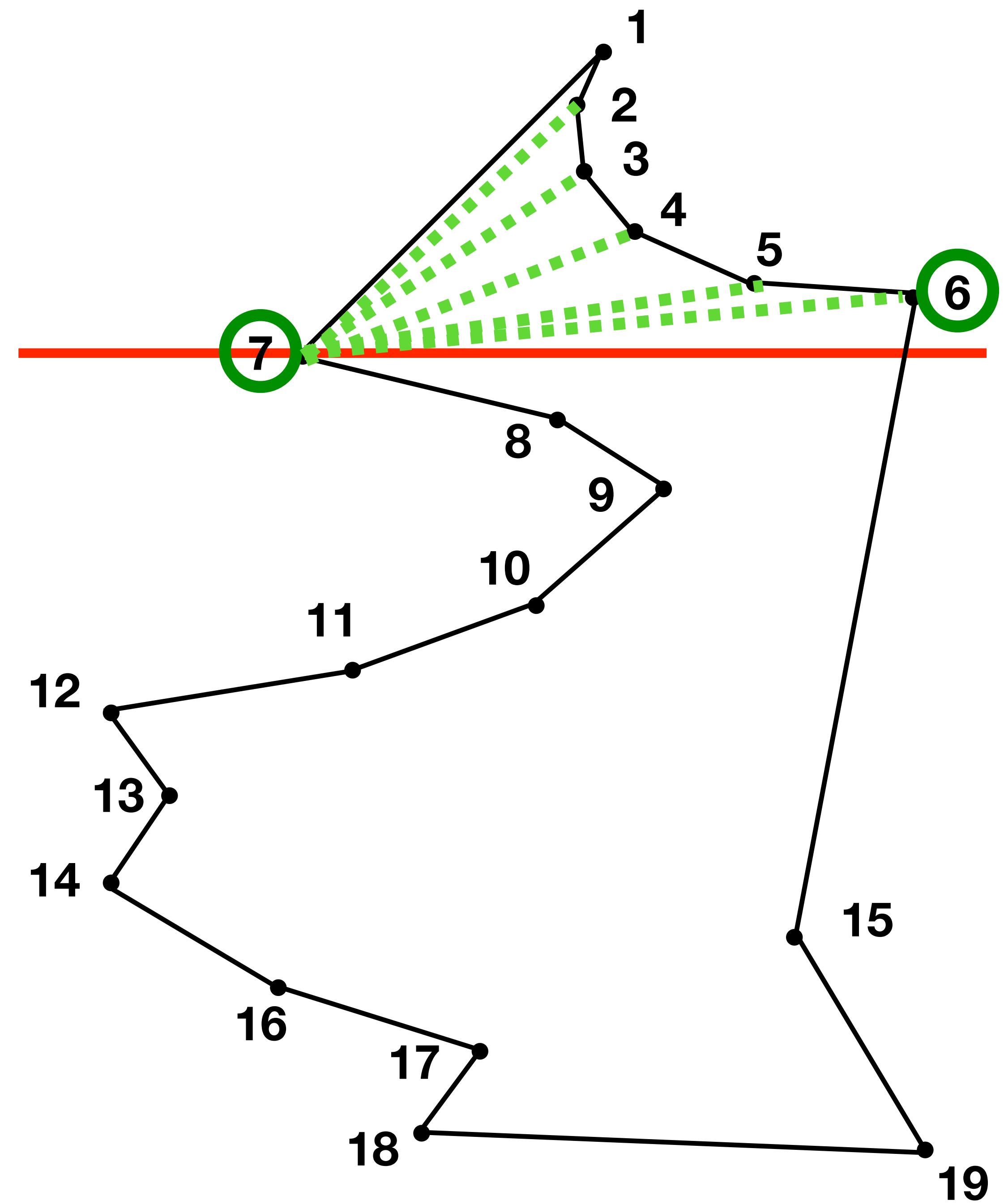
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 7



Algorithm 5.16

Input: A y-monotone polygon P

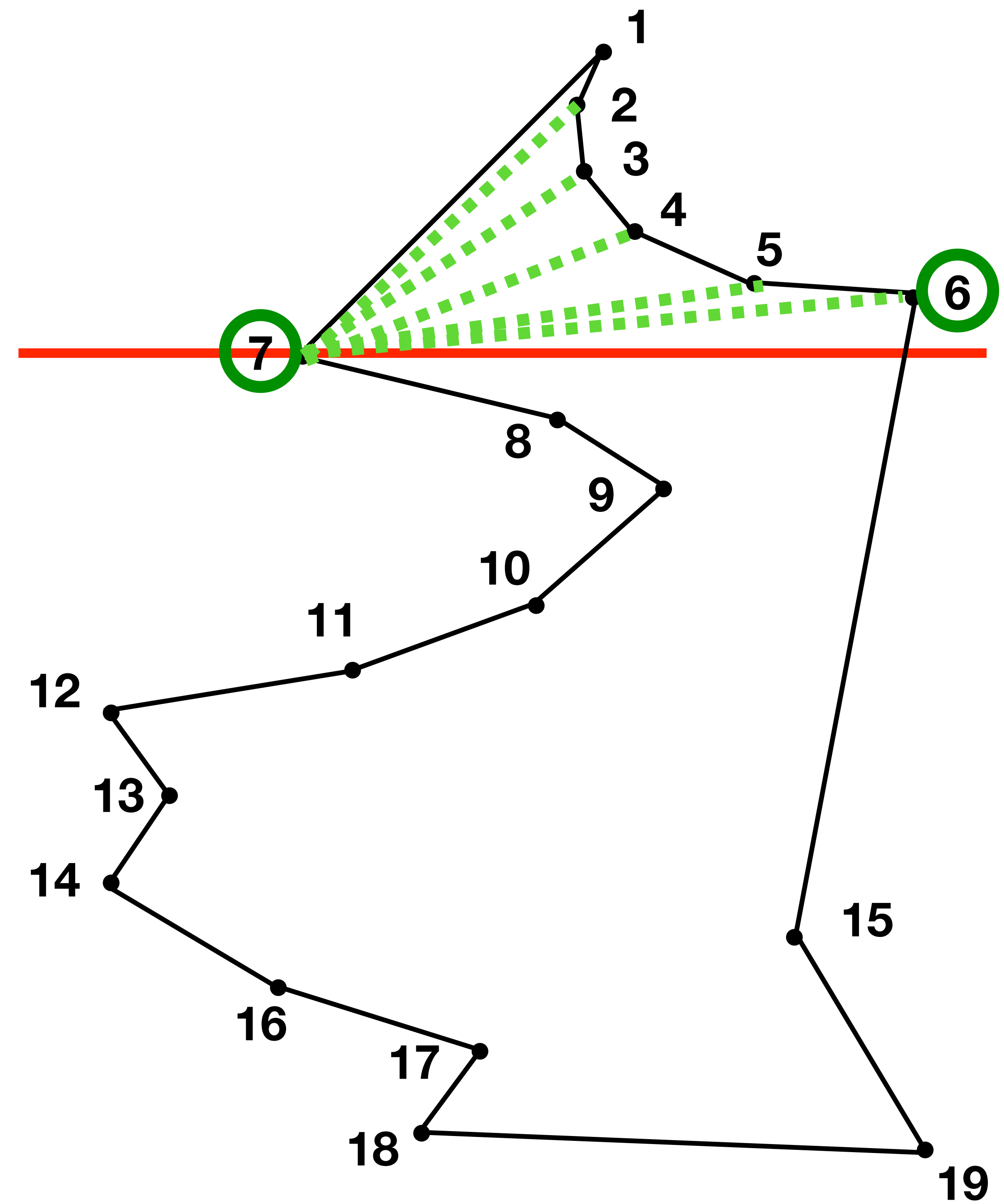
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 7



Technische Universität Braunschweig

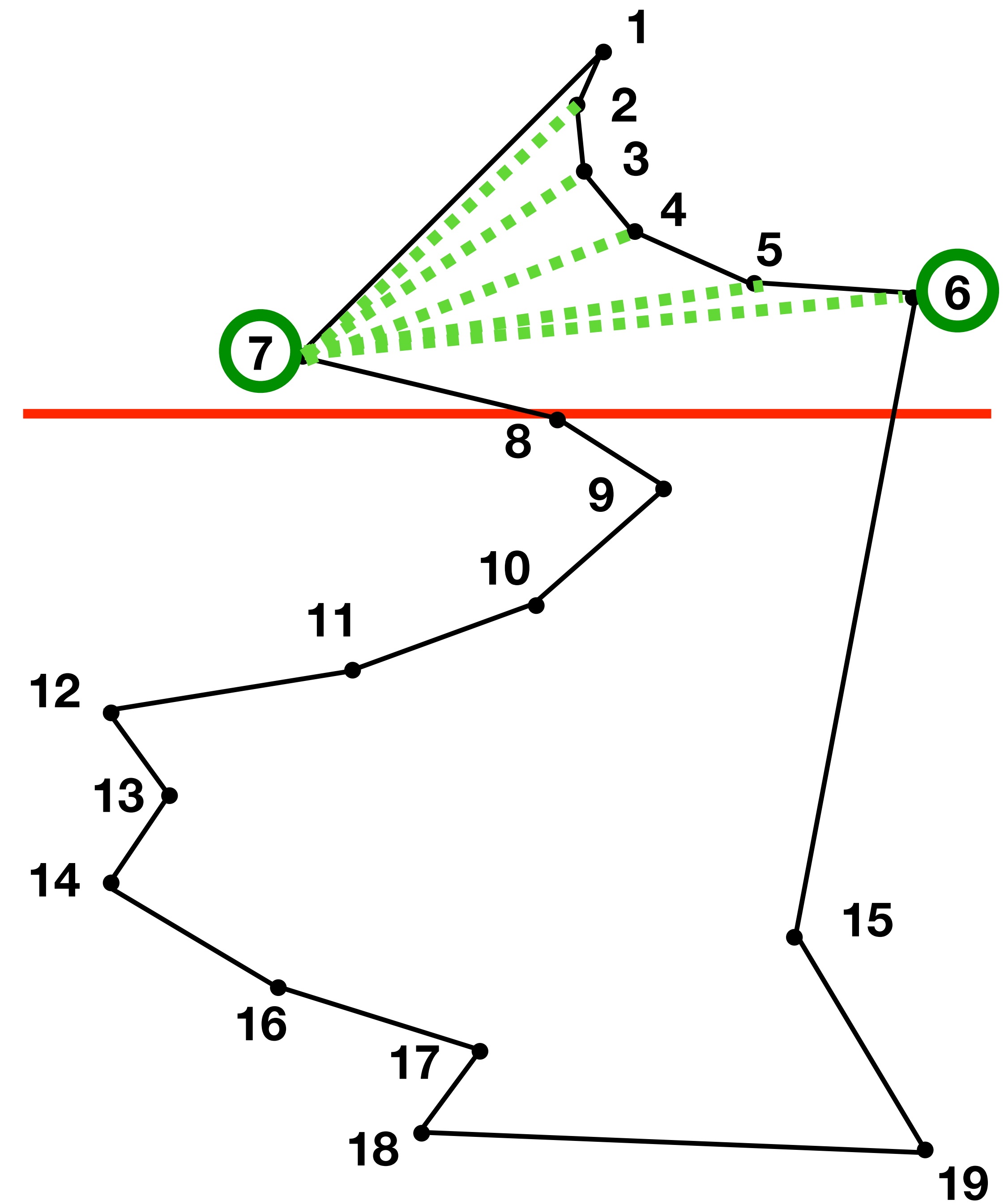
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 7



Algorithm 5.16

Input: A y-monotone polygon P

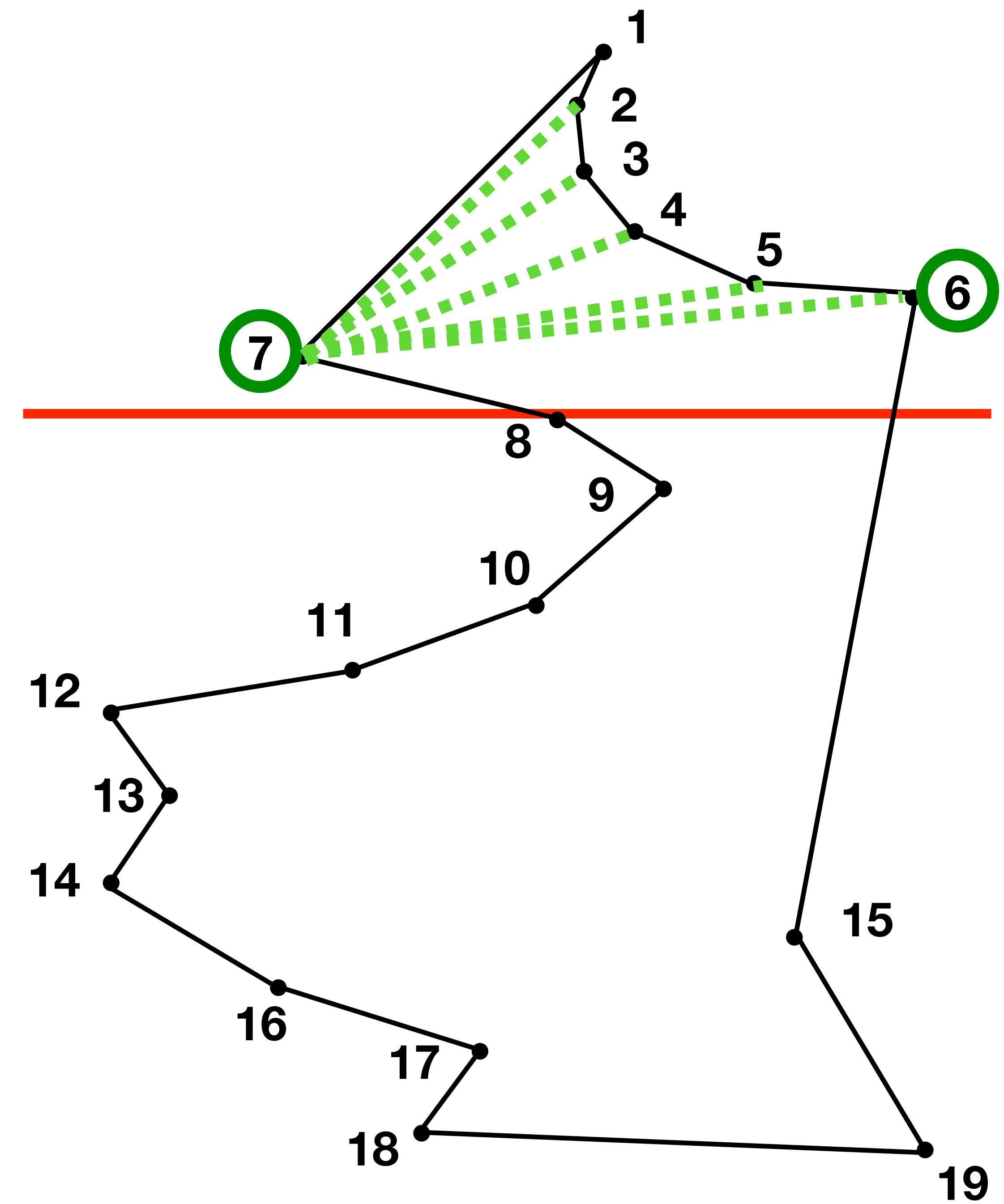
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 7



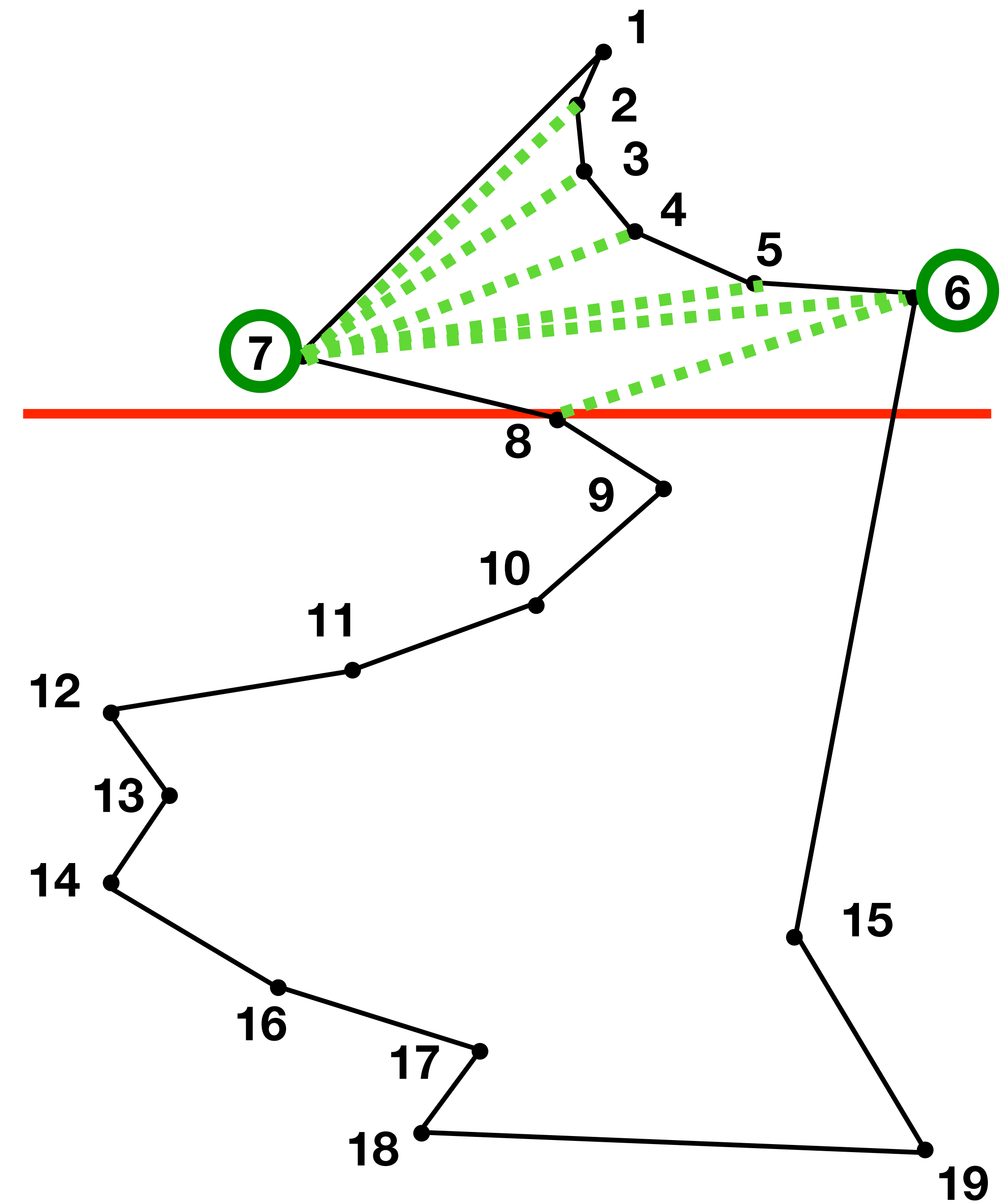
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 7



Algorithm 5.16

Input: A y-monotone polygon P

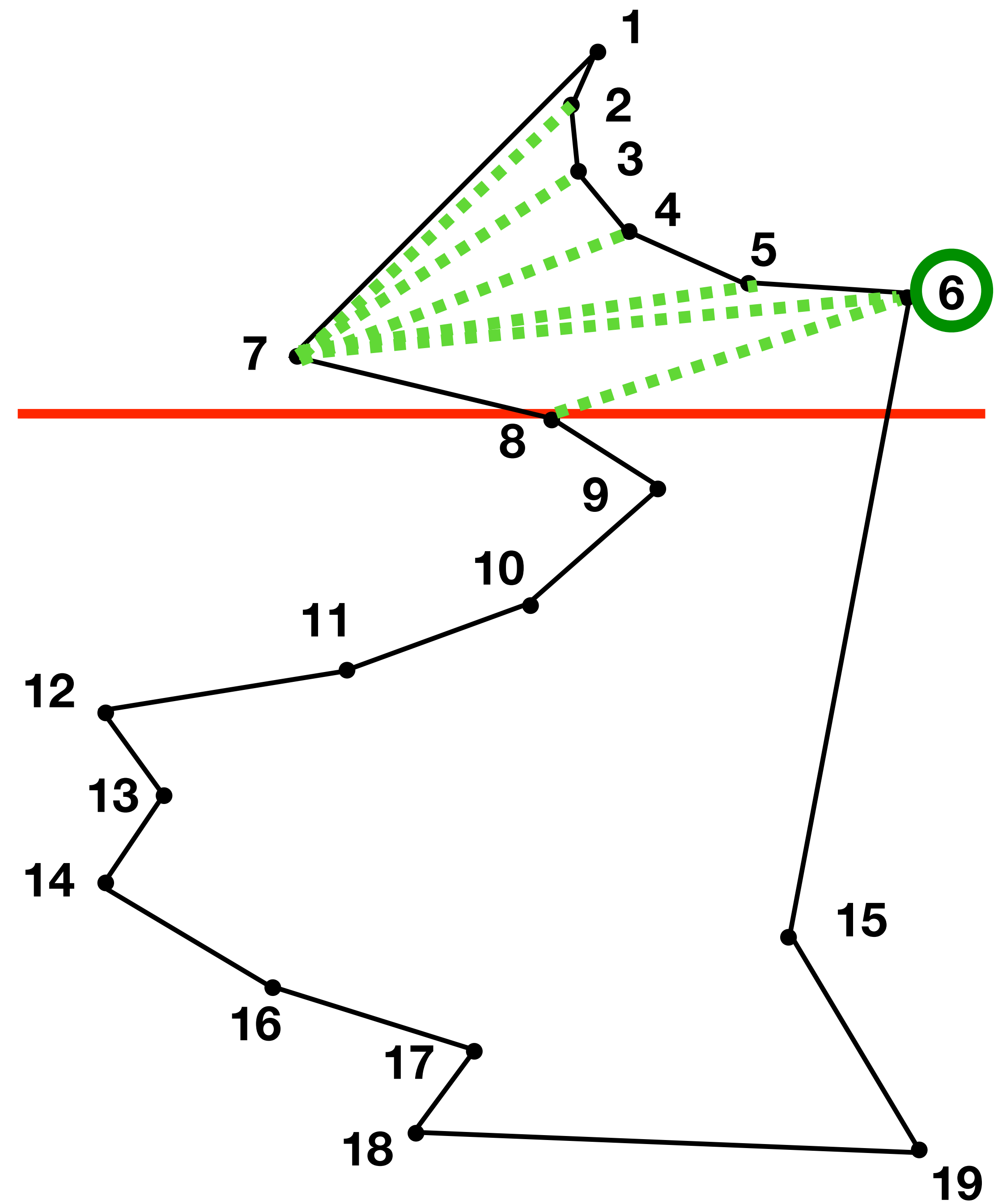
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 6 7



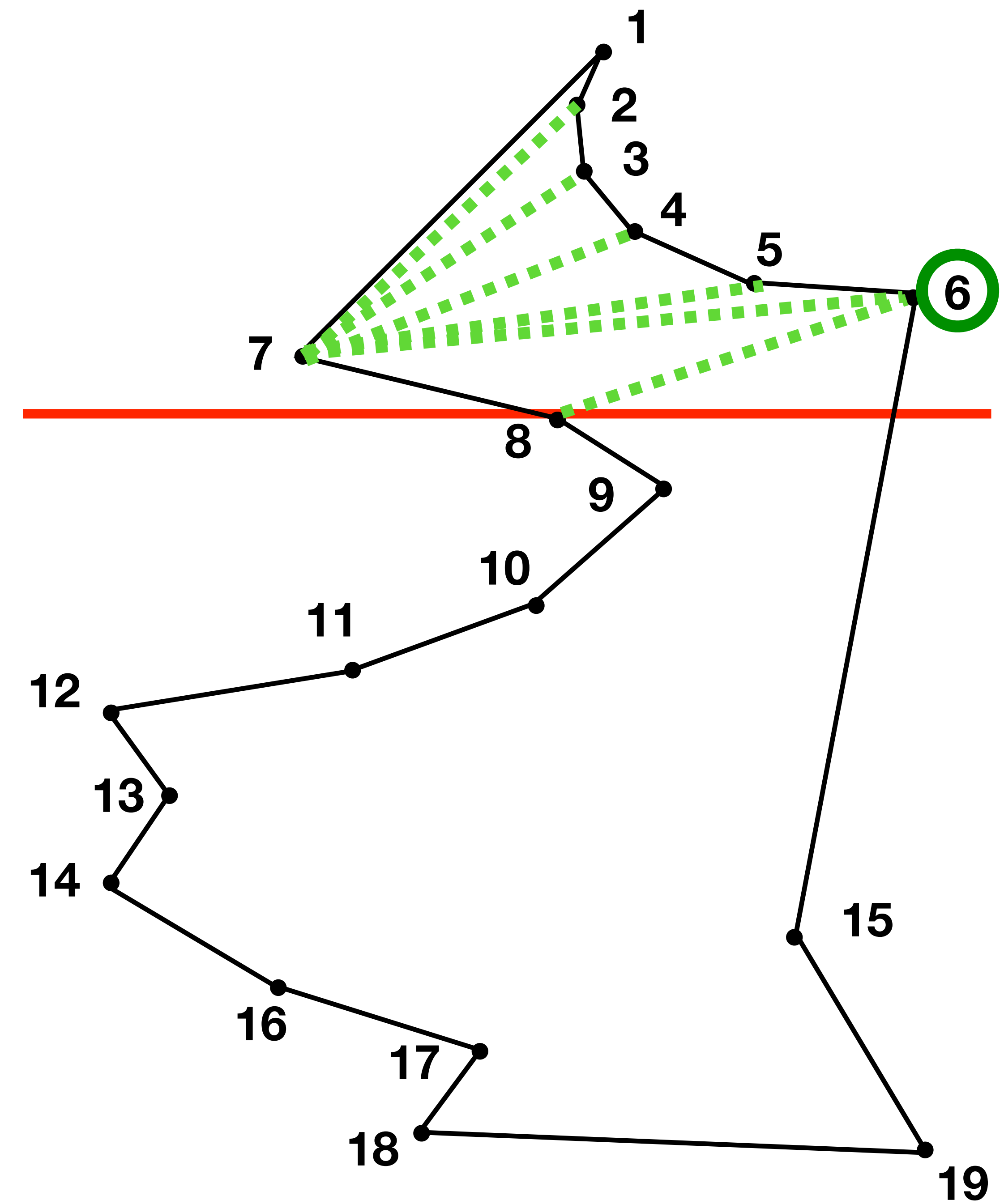
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6



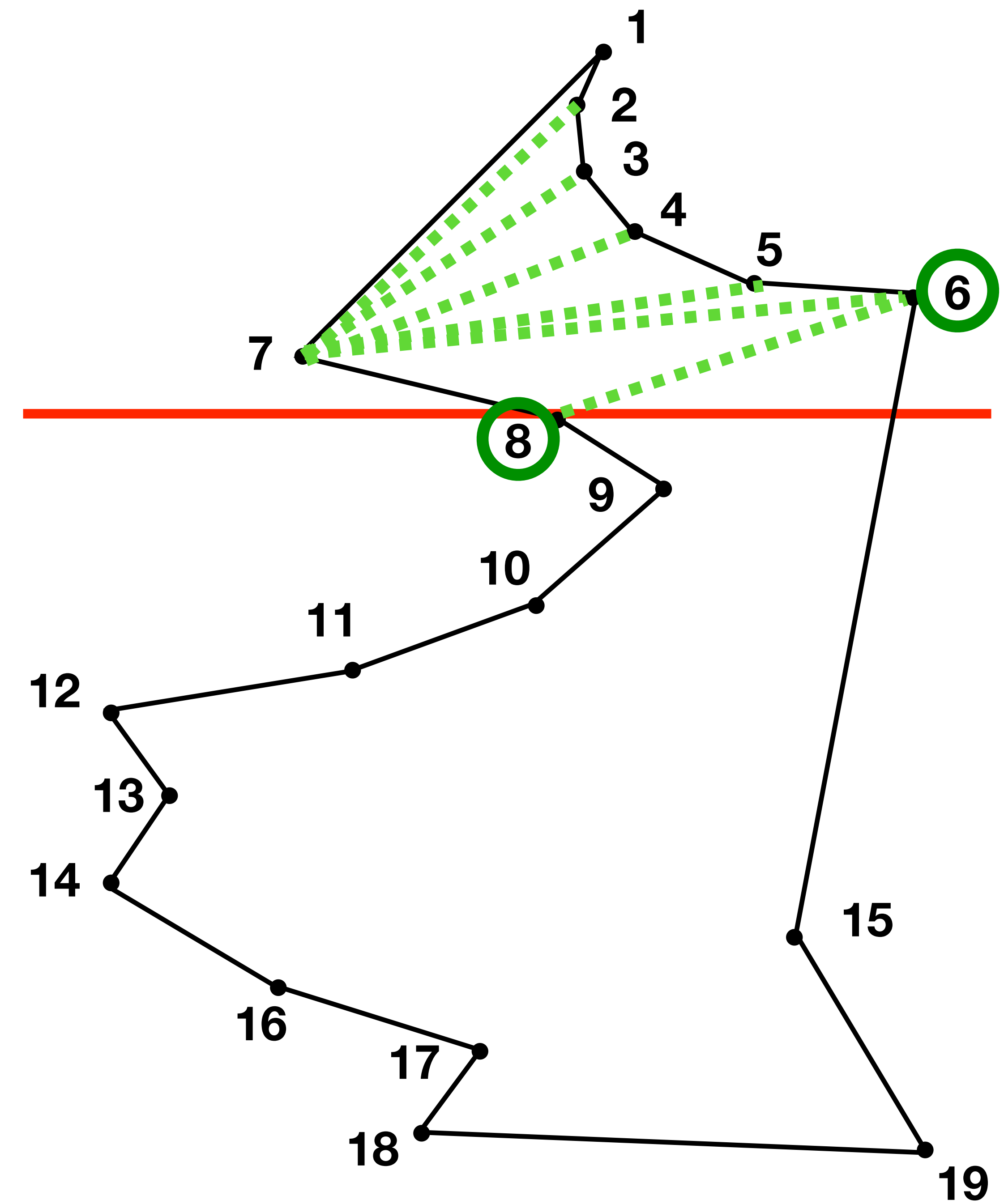
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6



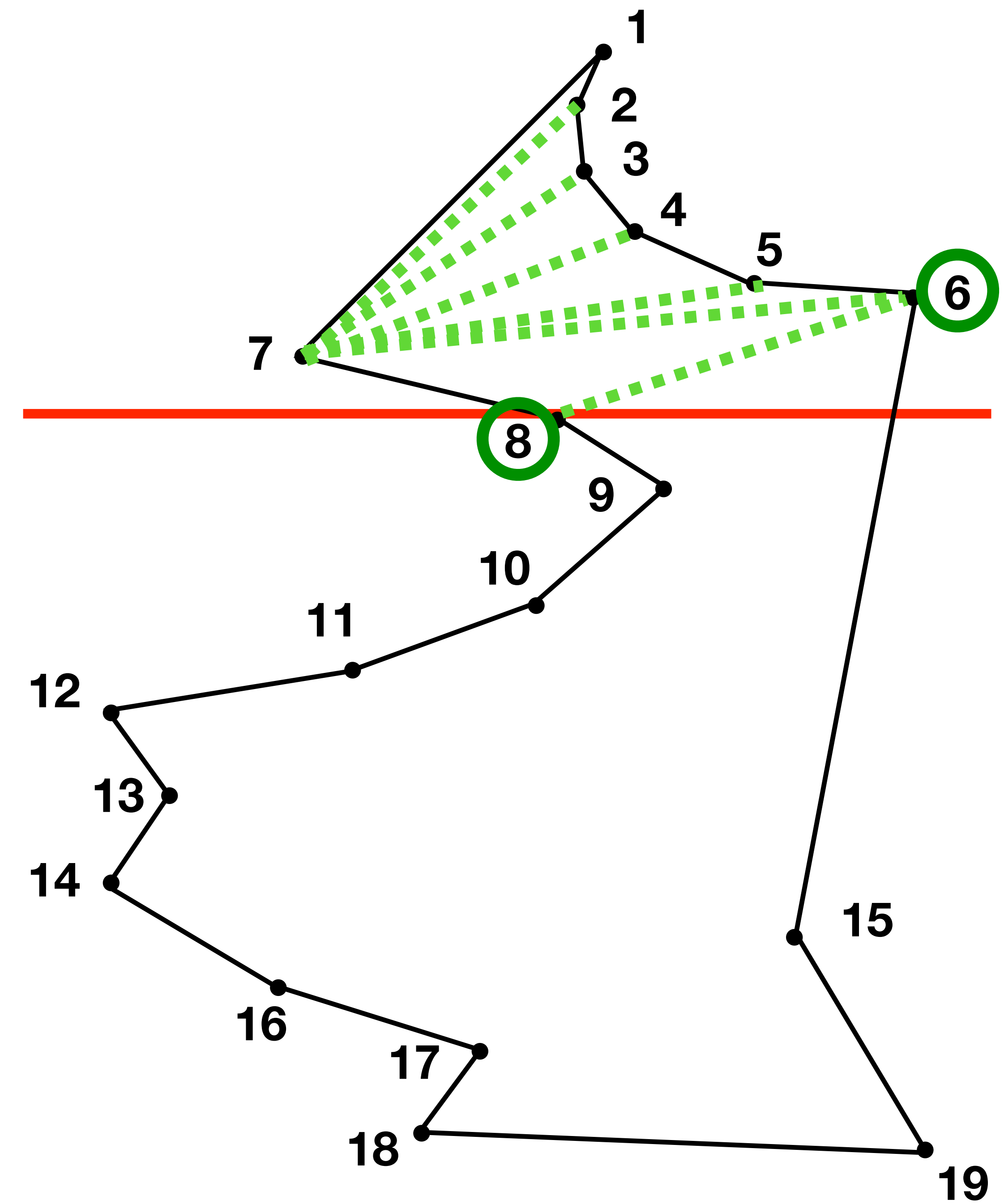
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 8



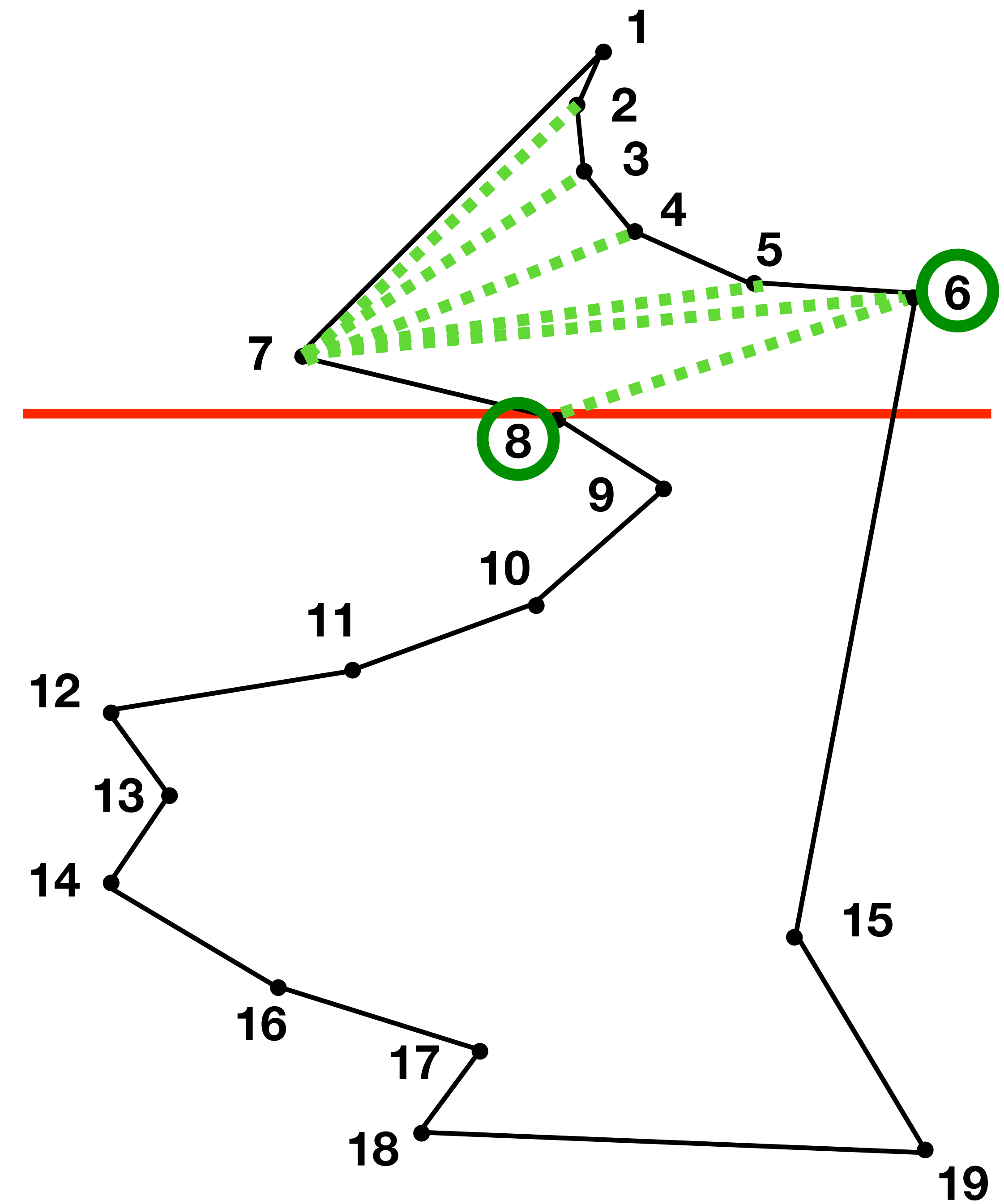
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 8



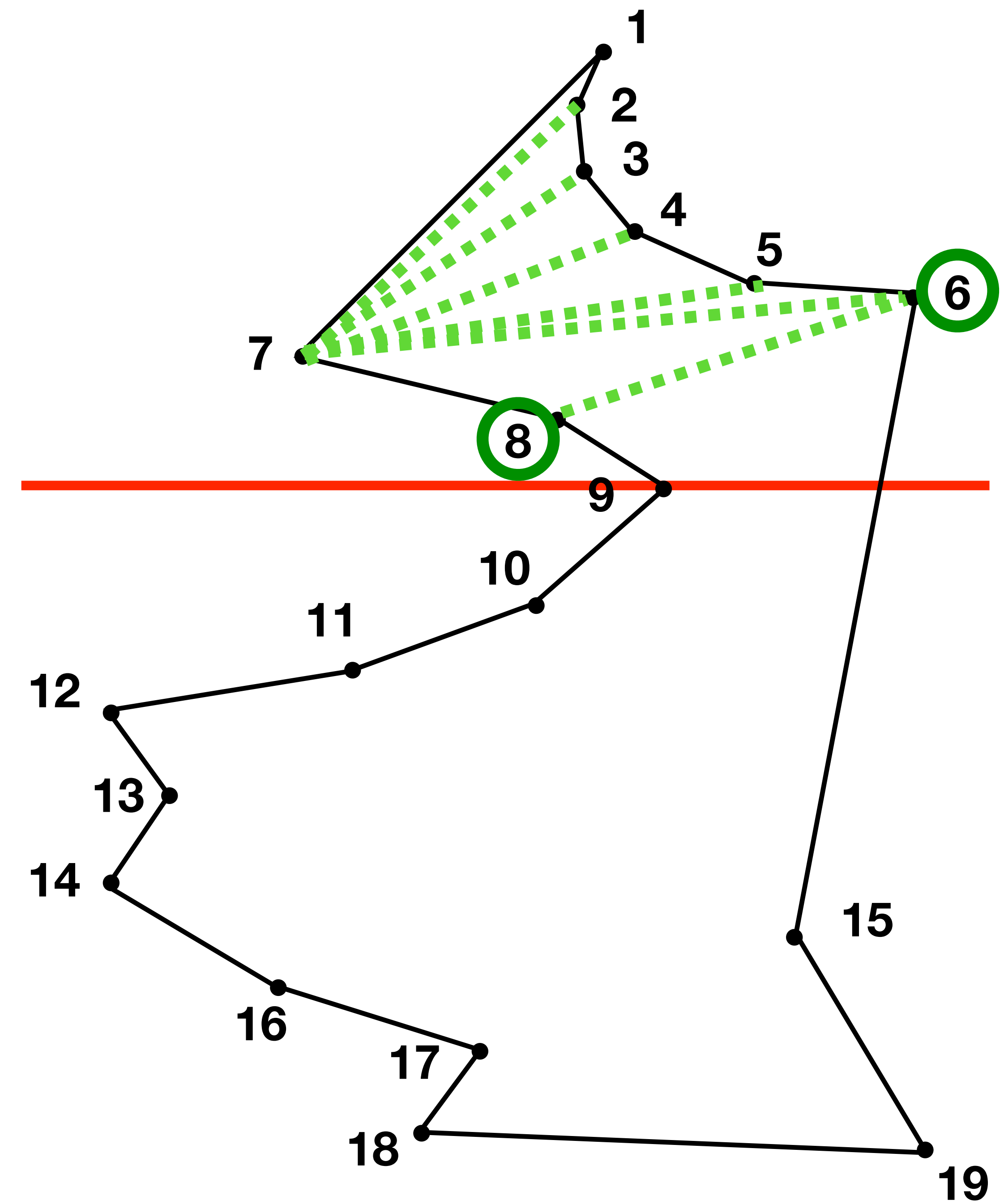
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 8



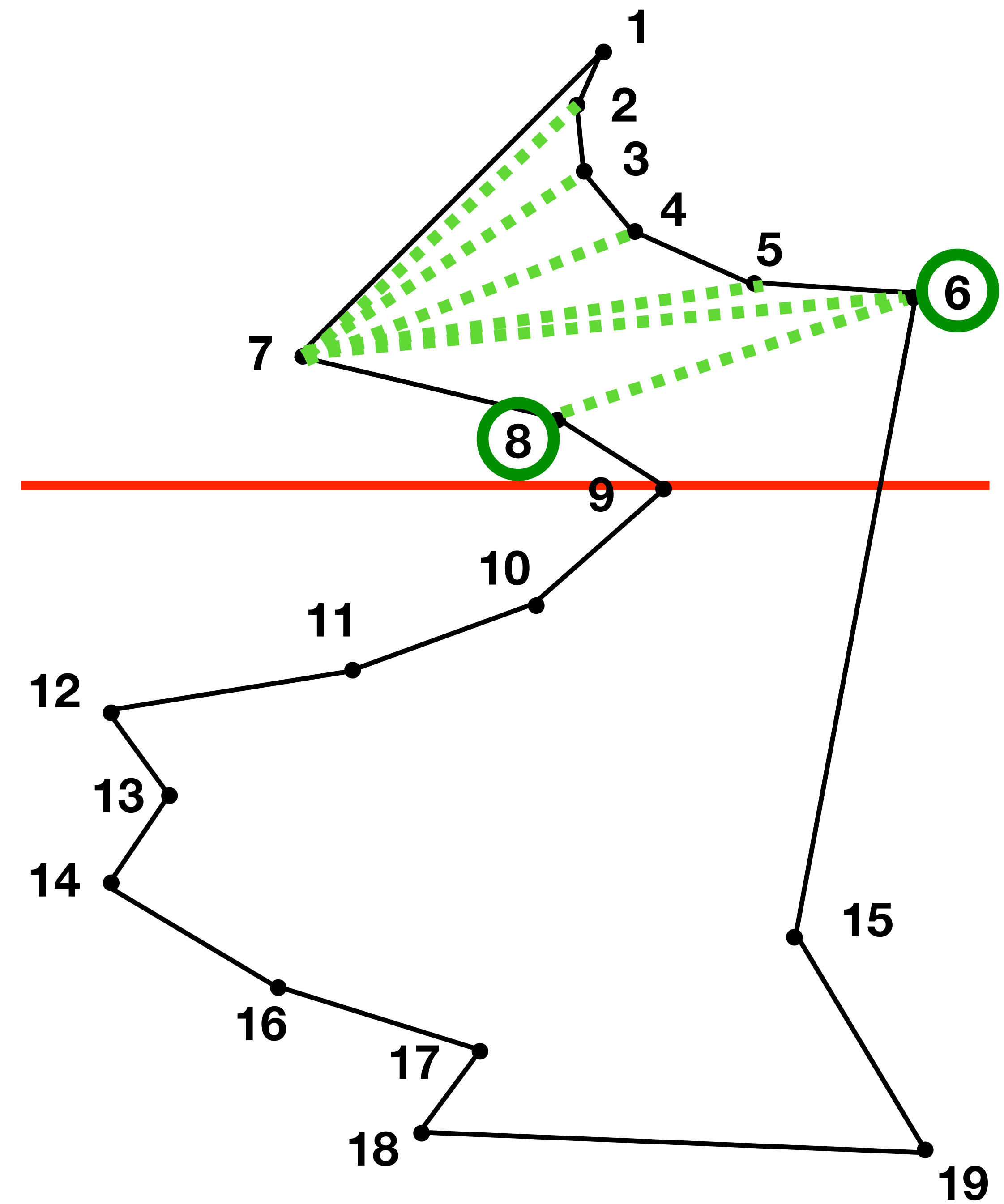
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
    - First two vertices of unfinished subpolygon
    - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
    - 3.1 If opposite chain:
        - Connect to all vertices in chain, leave in queue **Chain**
    - 3.2 If same chain, predecessor reflex:
        - Add to queue **Add**
    - 3.3 If same chain, predecessor convex:
        - Ear! Connect and update **Ear**

Queue: 6 8



Algorithm 5.16

Input: A y-monotone polygon P

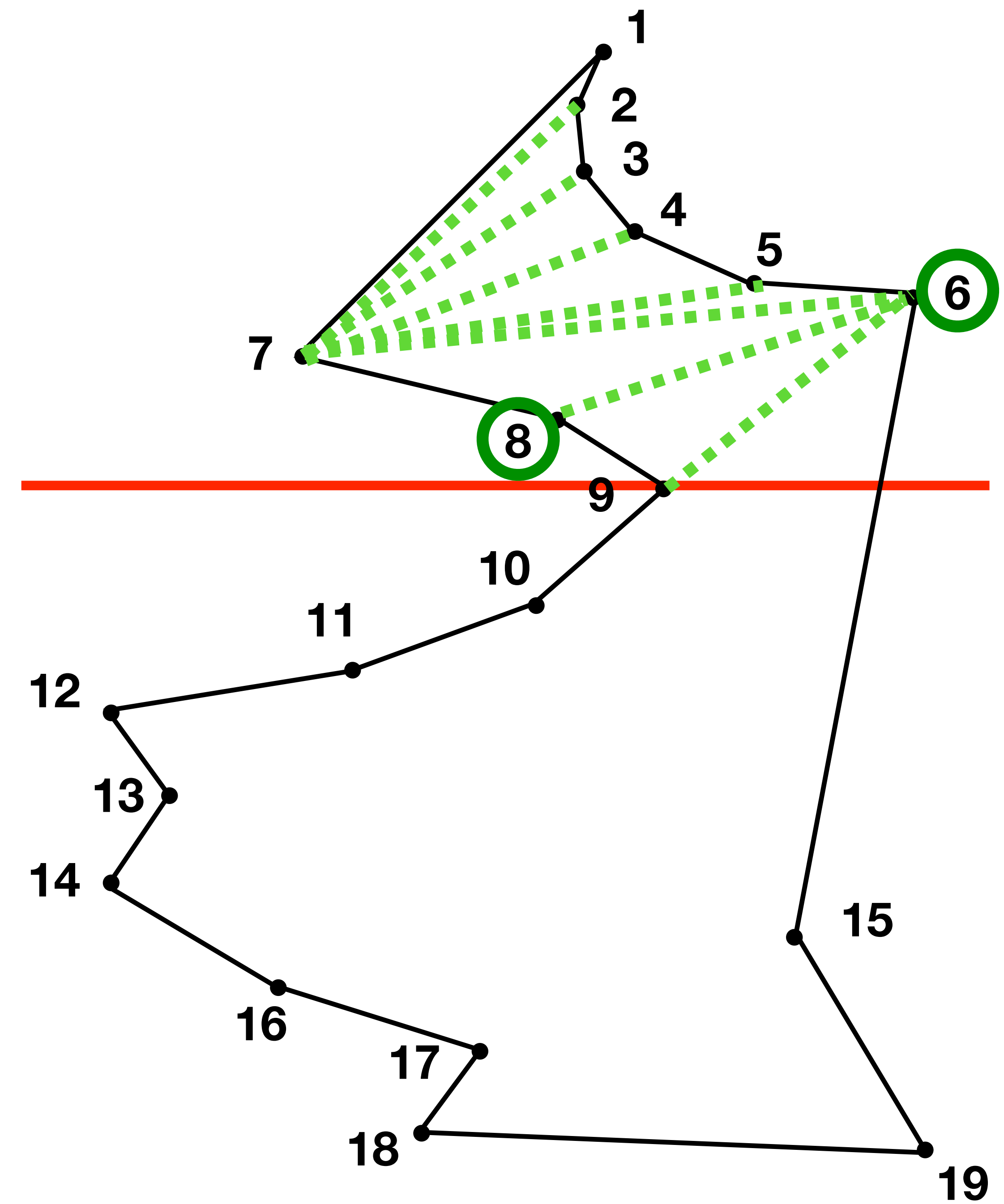
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 8



Algorithm 5.16

Input: A y-monotone polygon P

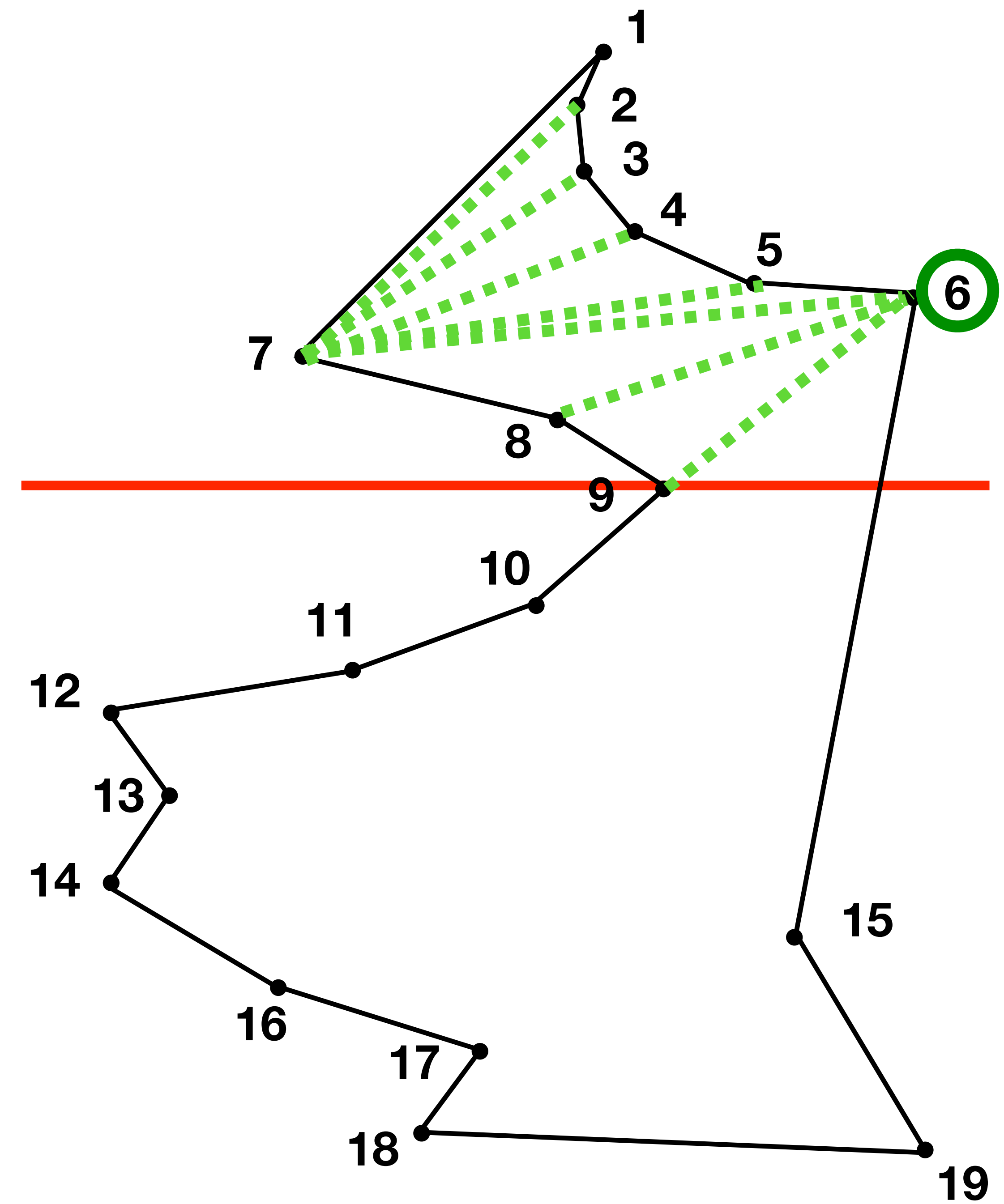
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 8



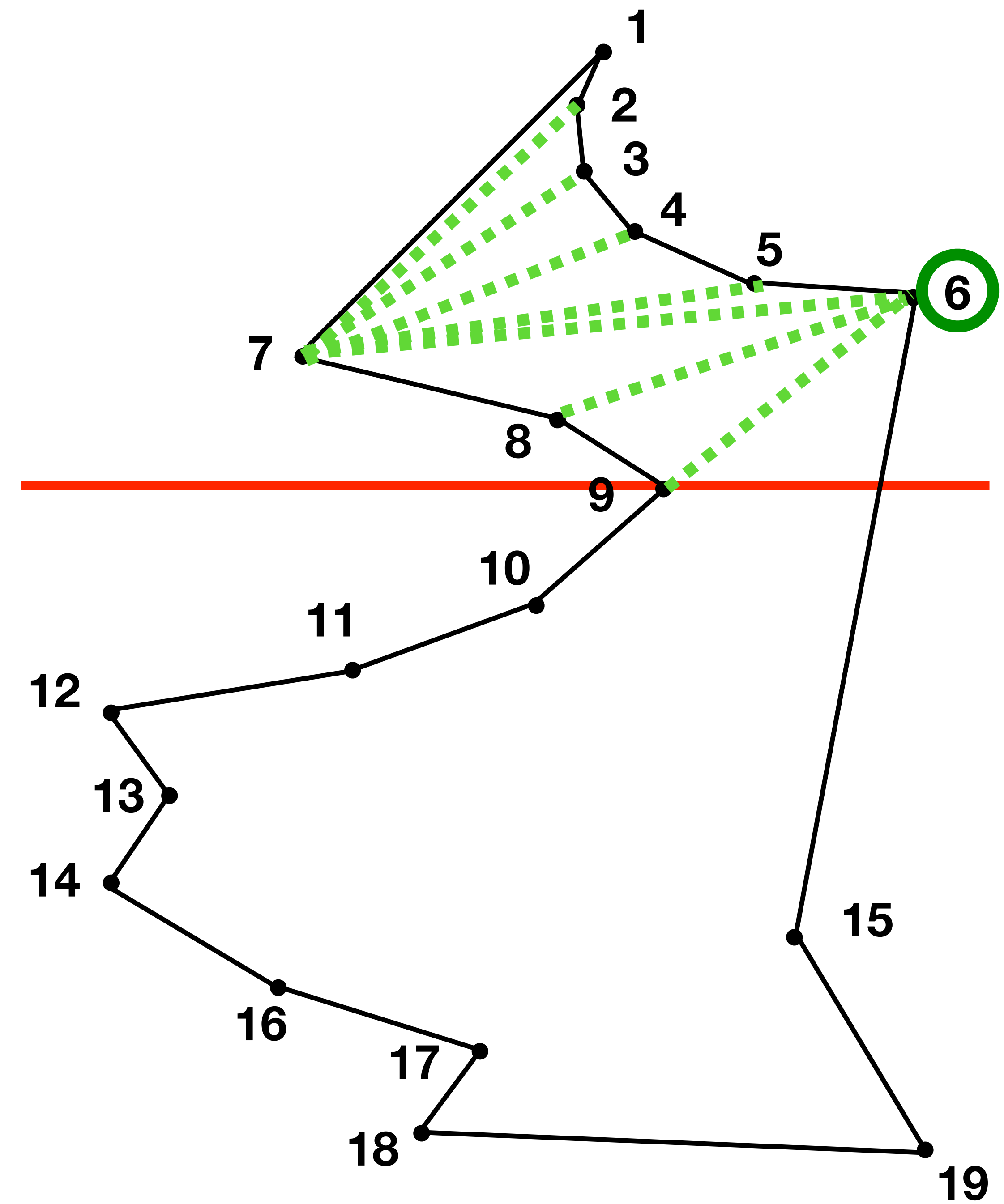
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6



Algorithm 5.16

Input: A y-monotone polygon P

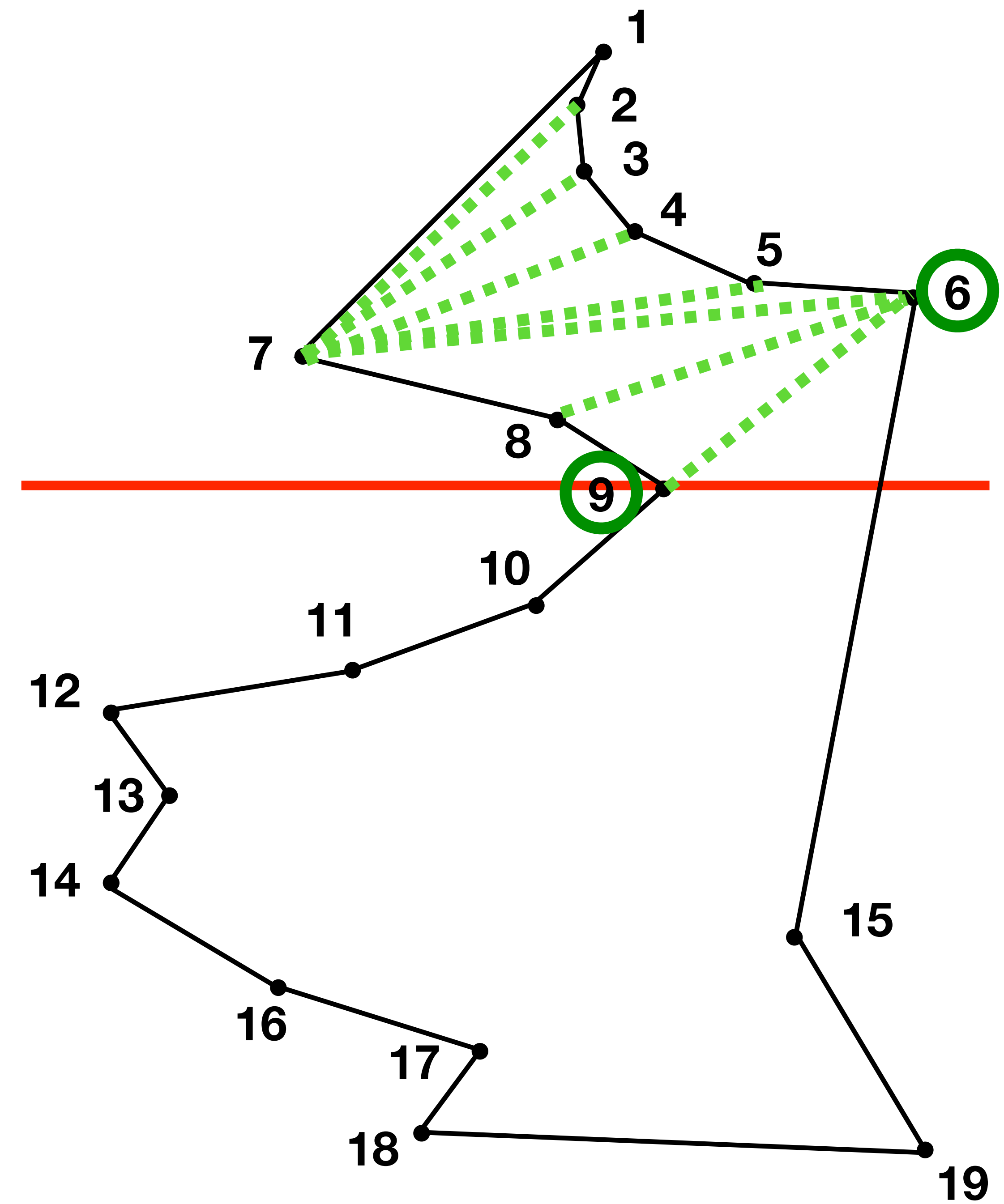
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6



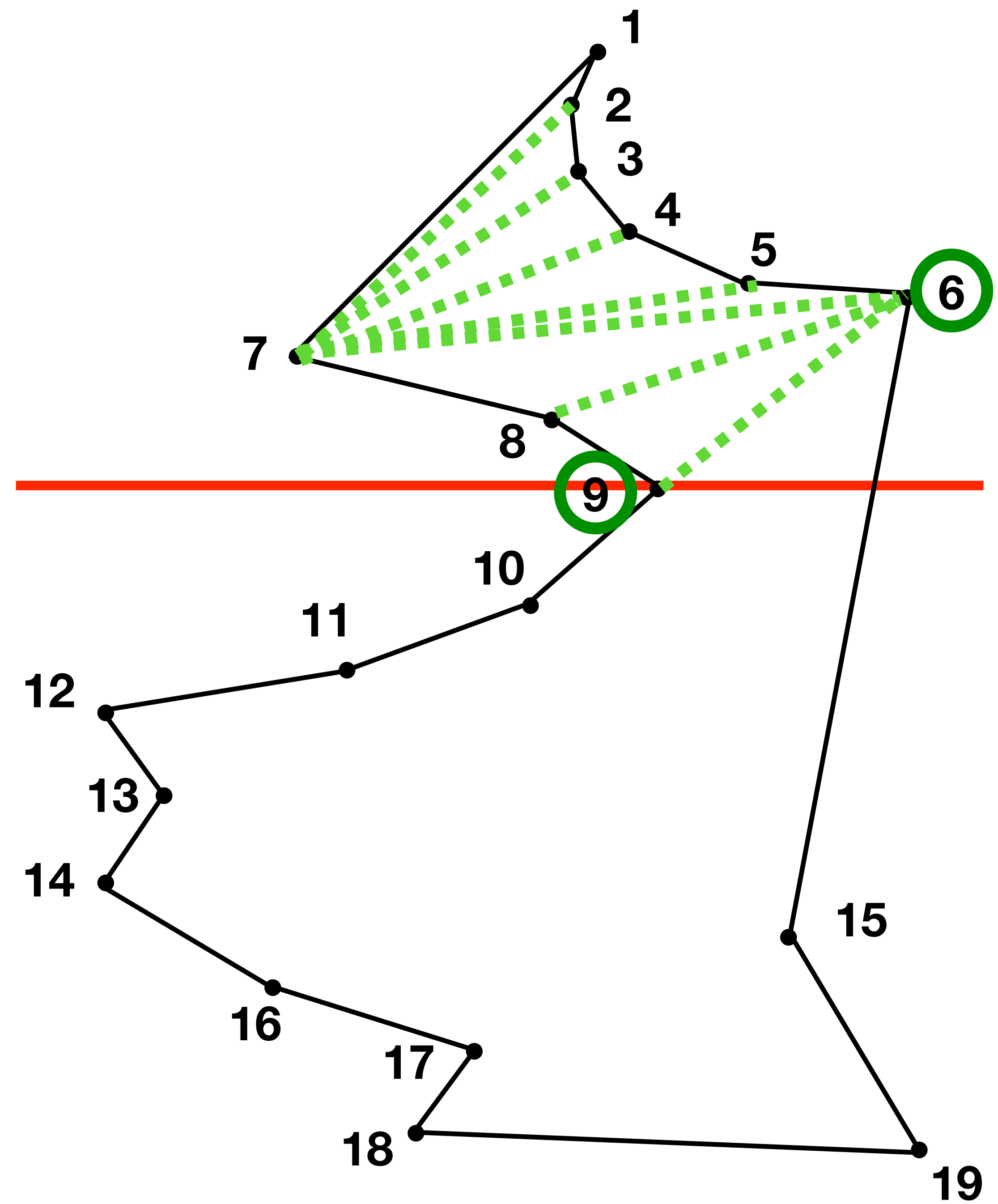
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 9



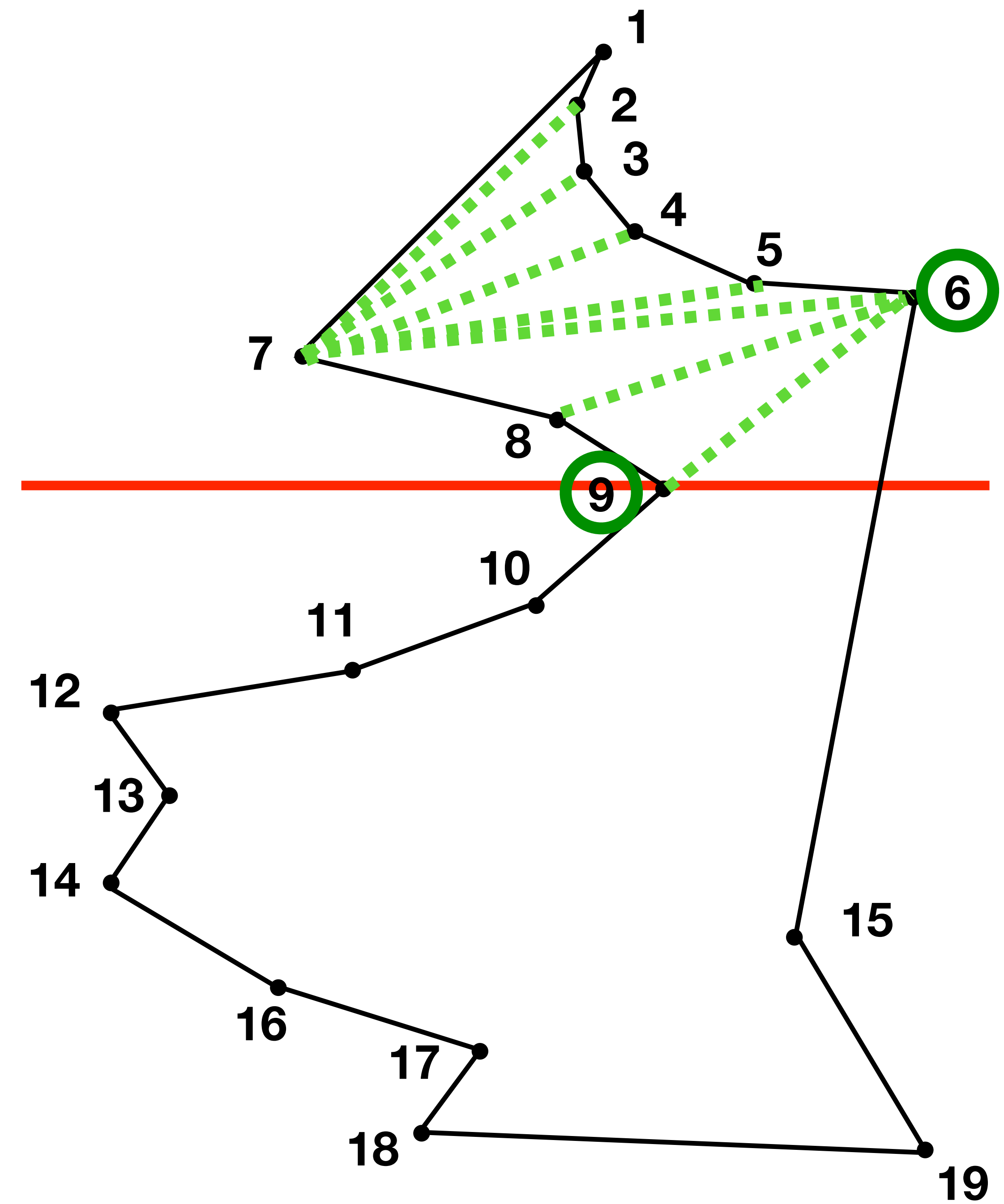
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 9



Algorithm 5.16

Input: A y-monotone polygon P

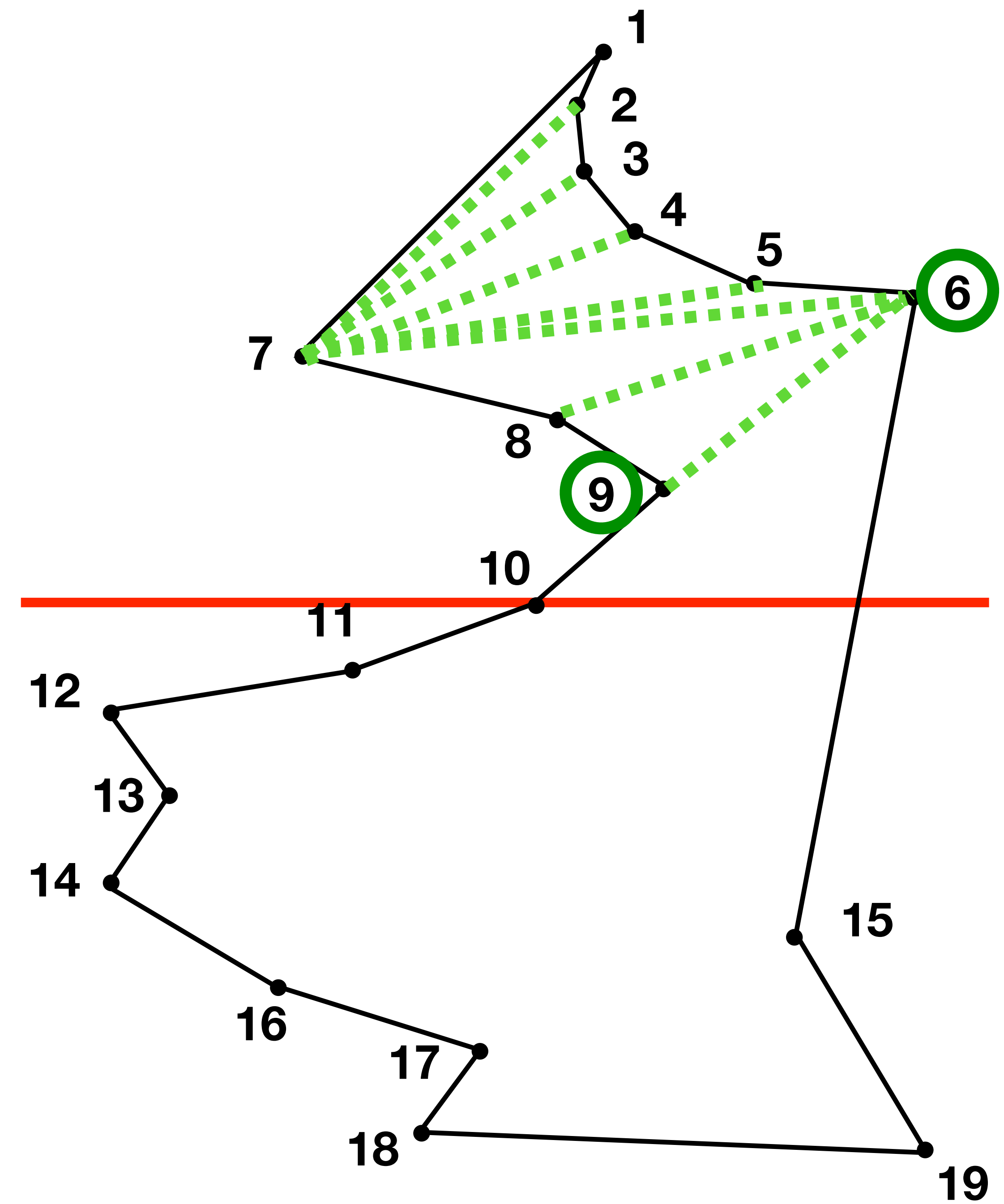
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 9



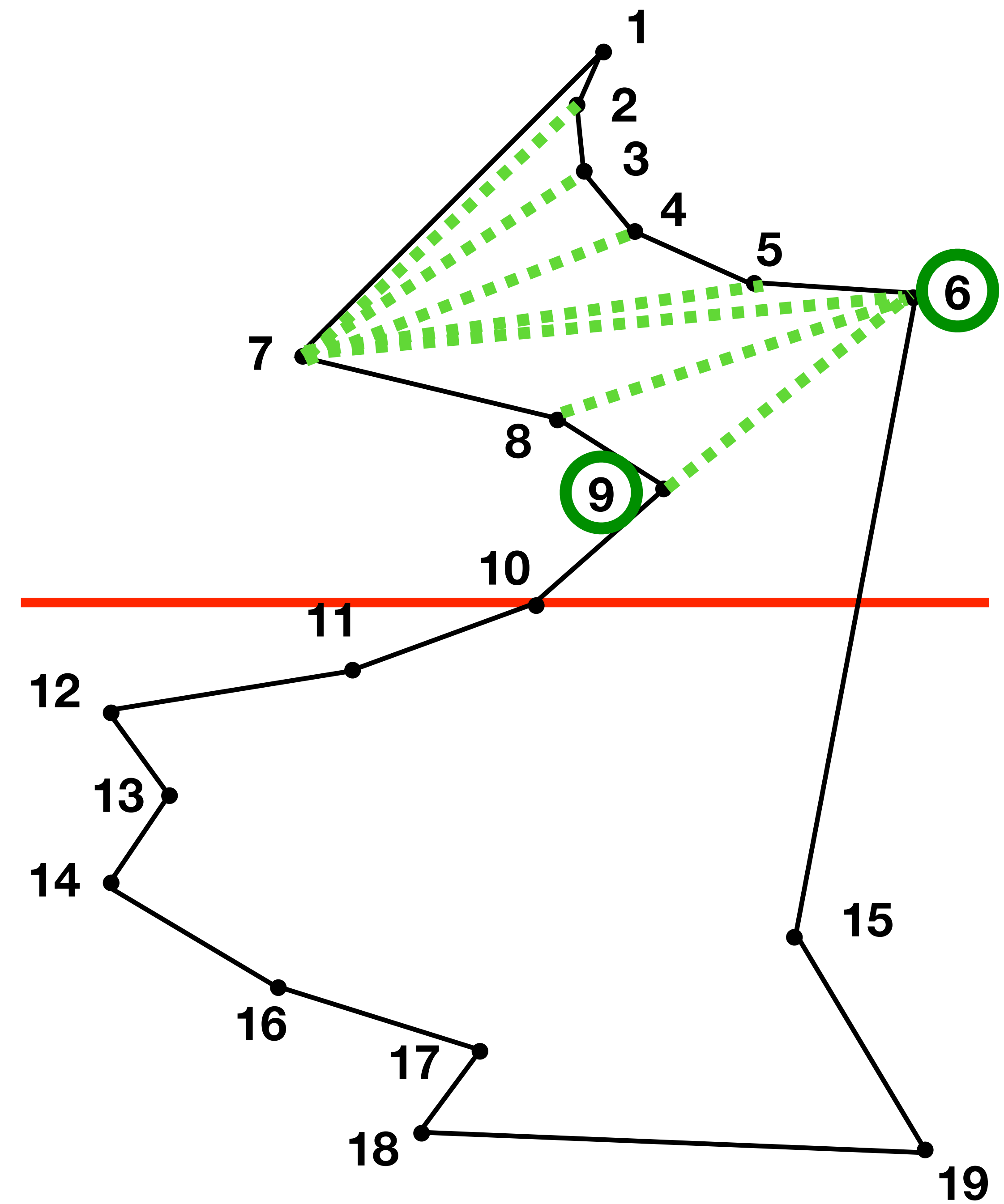
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 9



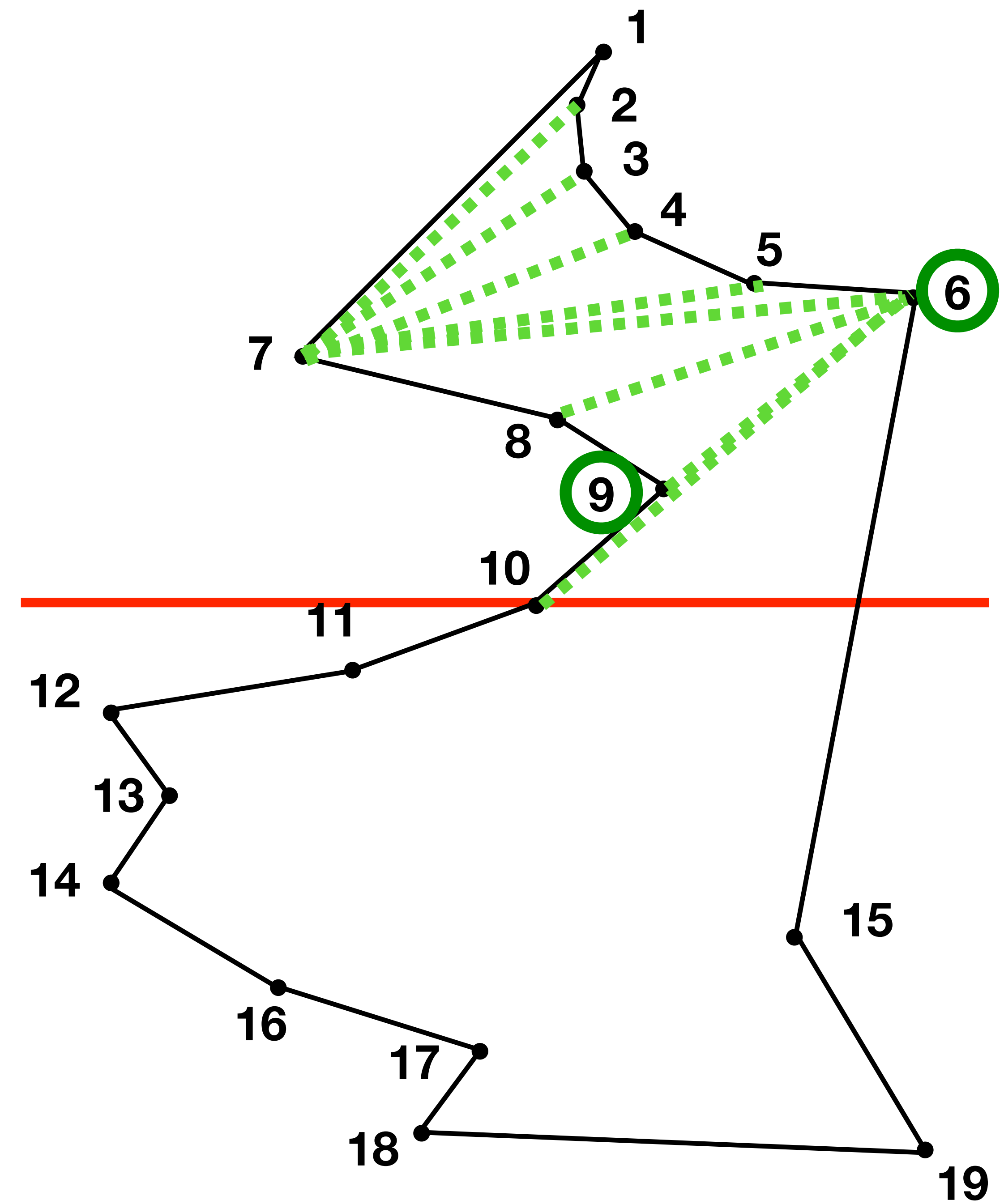
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 9



Algorithm 5.16

Input: A y-monotone polygon P

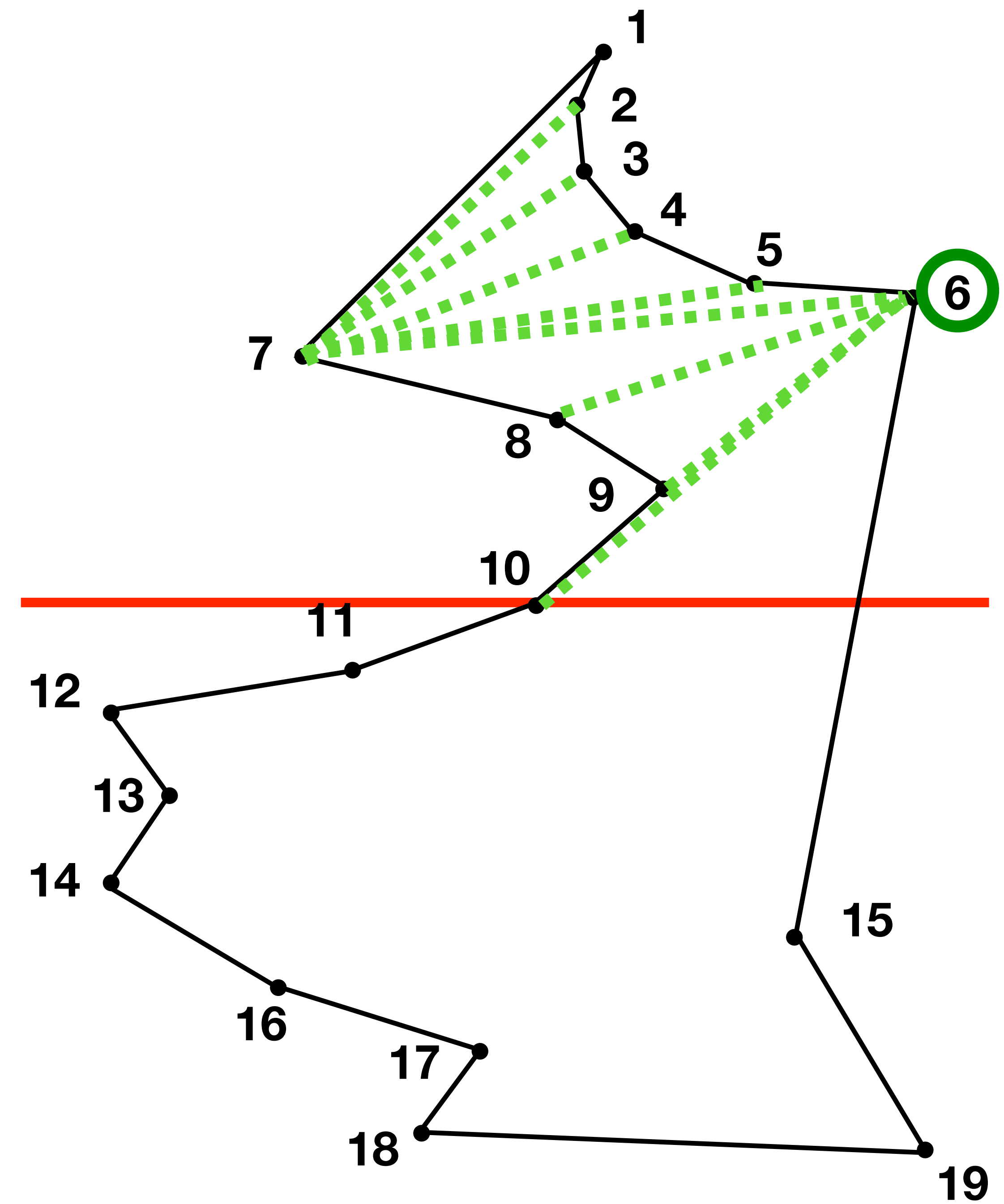
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 9



Algorithm 5.16

Input: A y-monotone polygon P

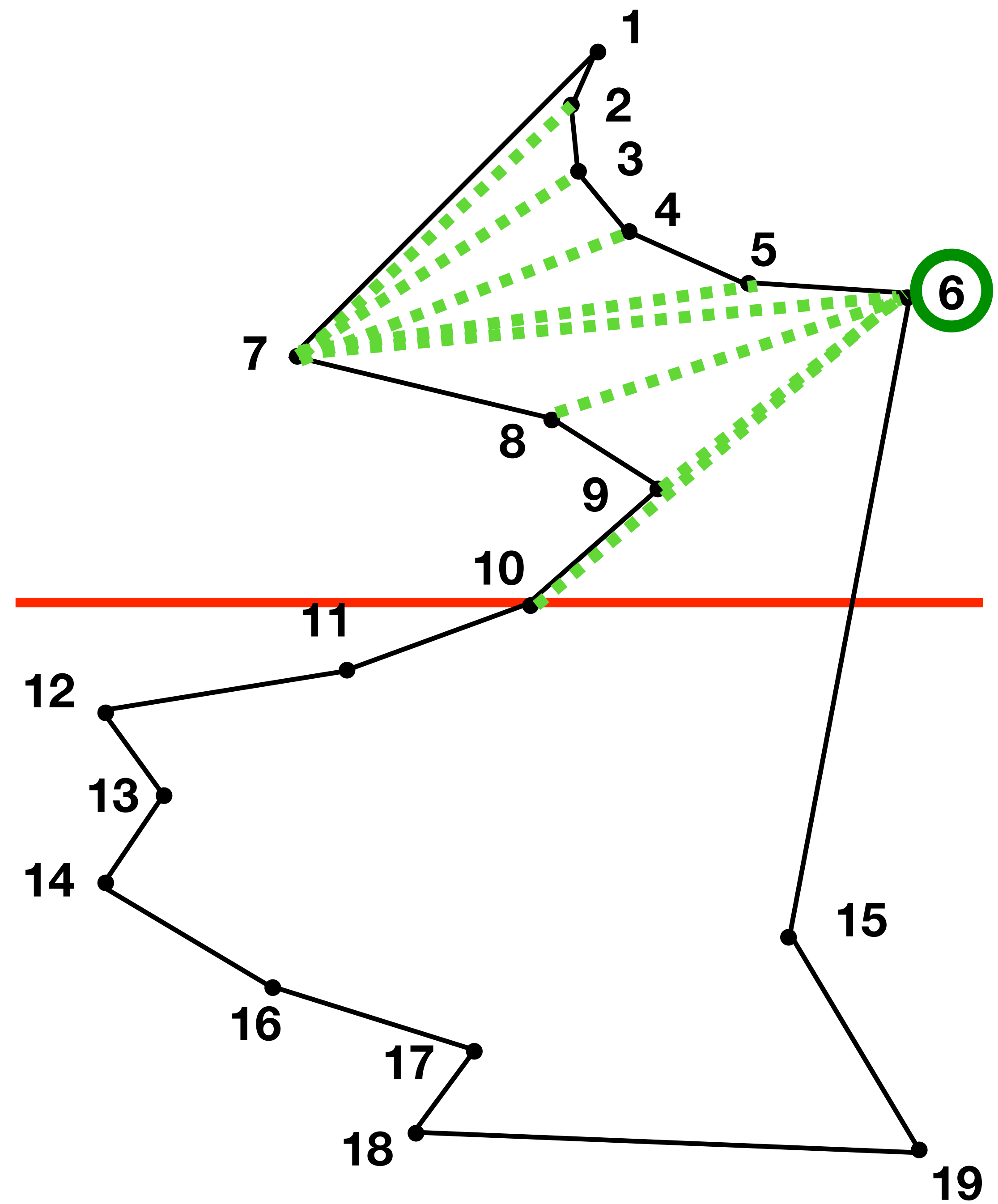
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 6



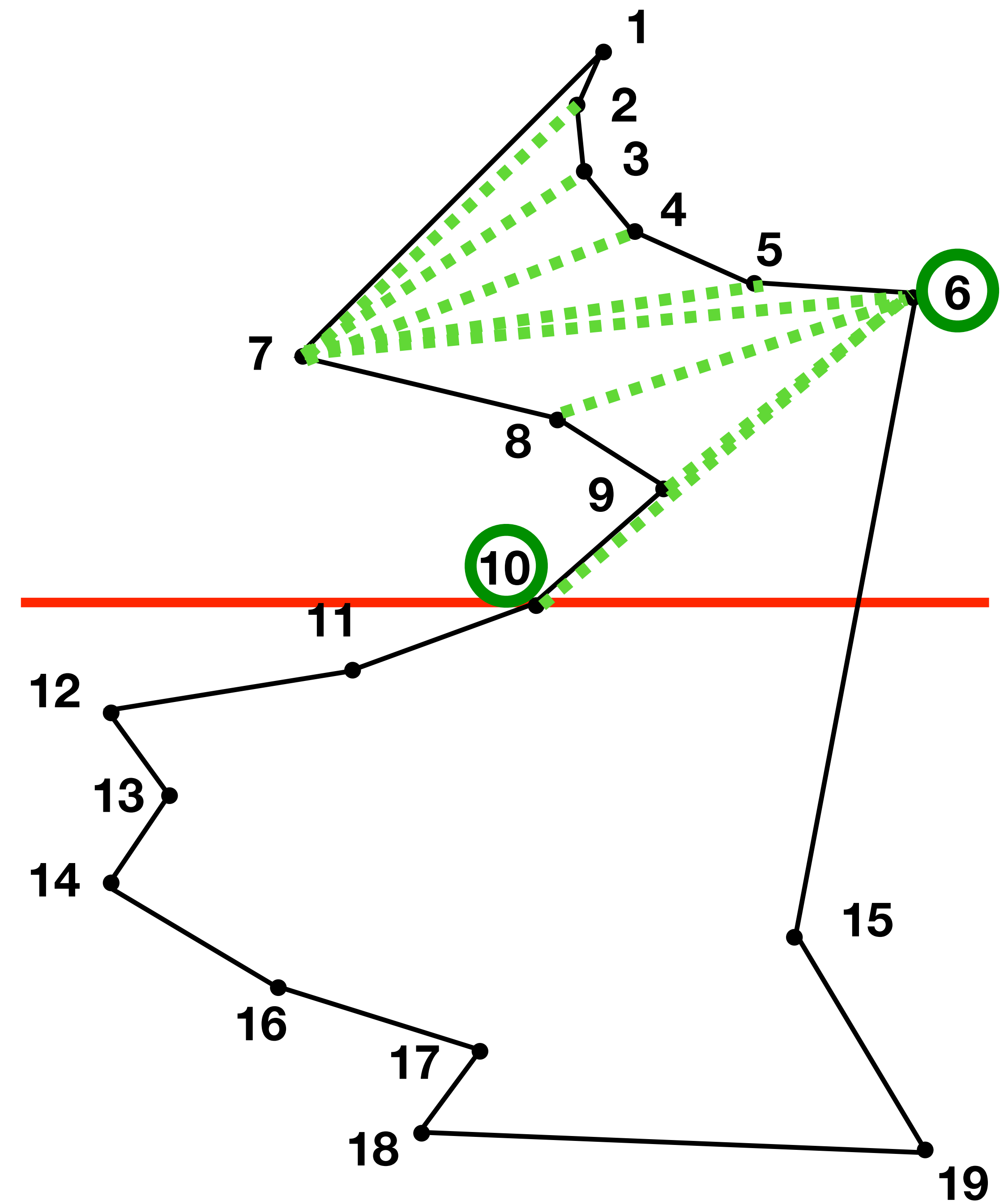
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6



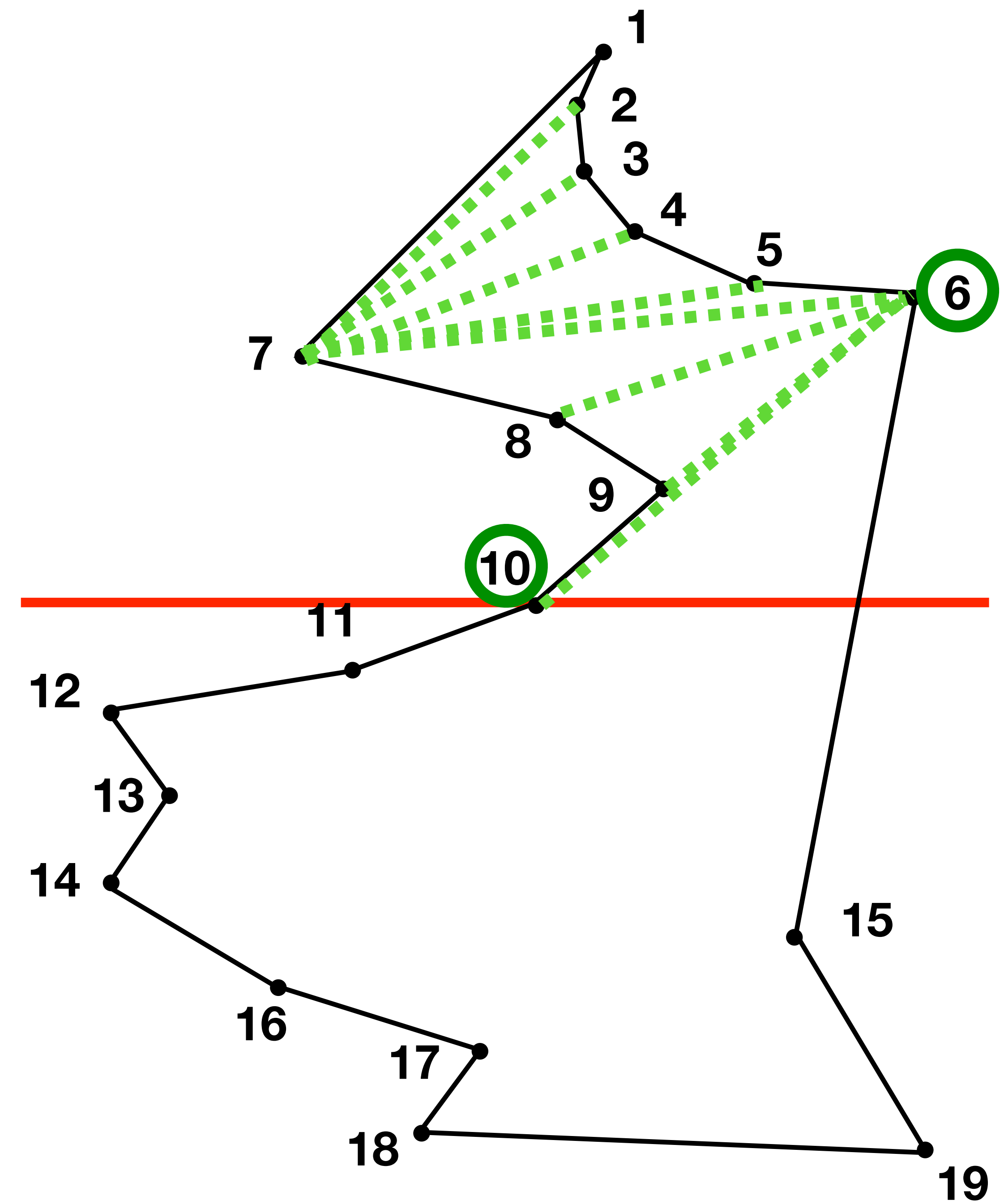
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10



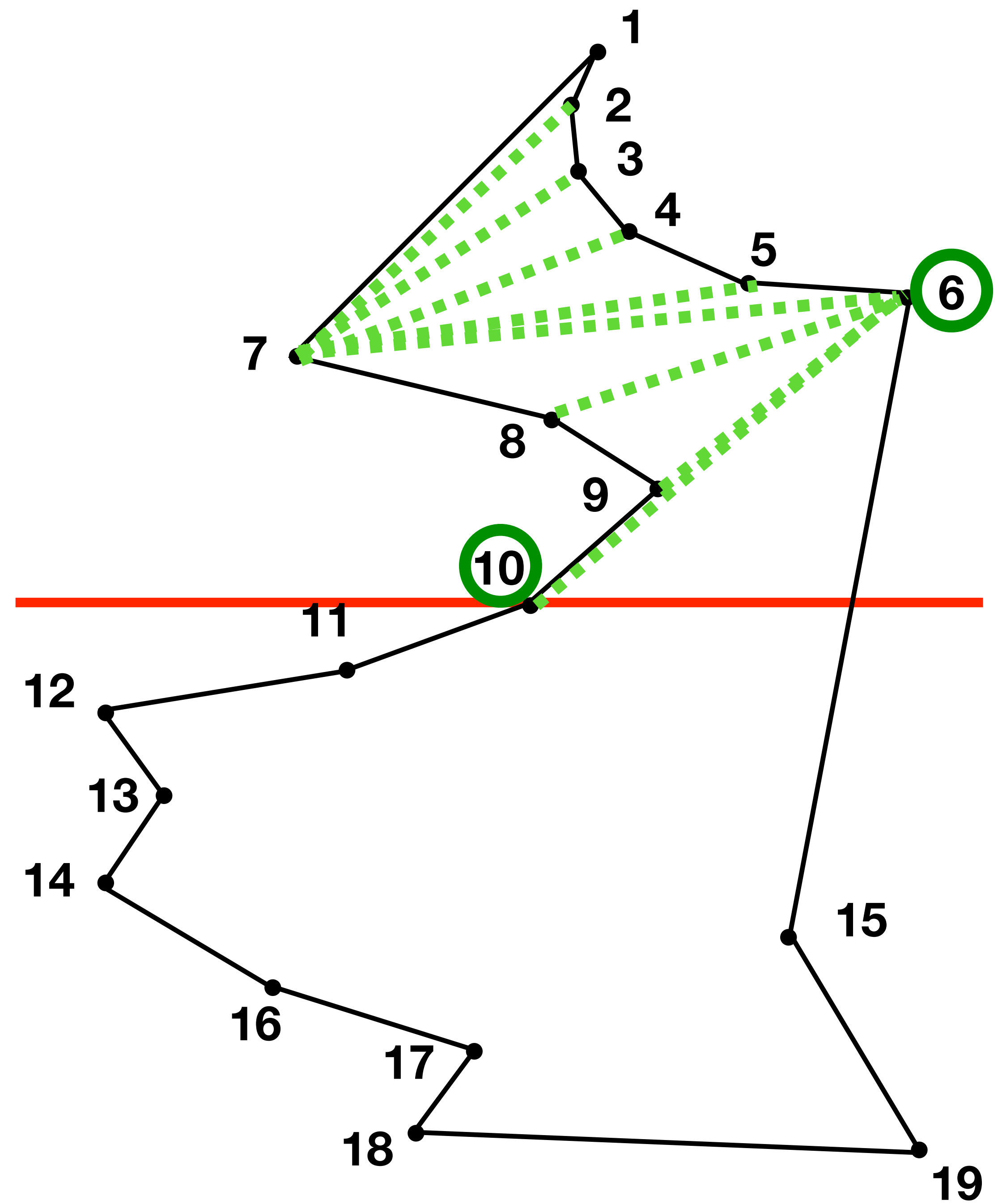
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

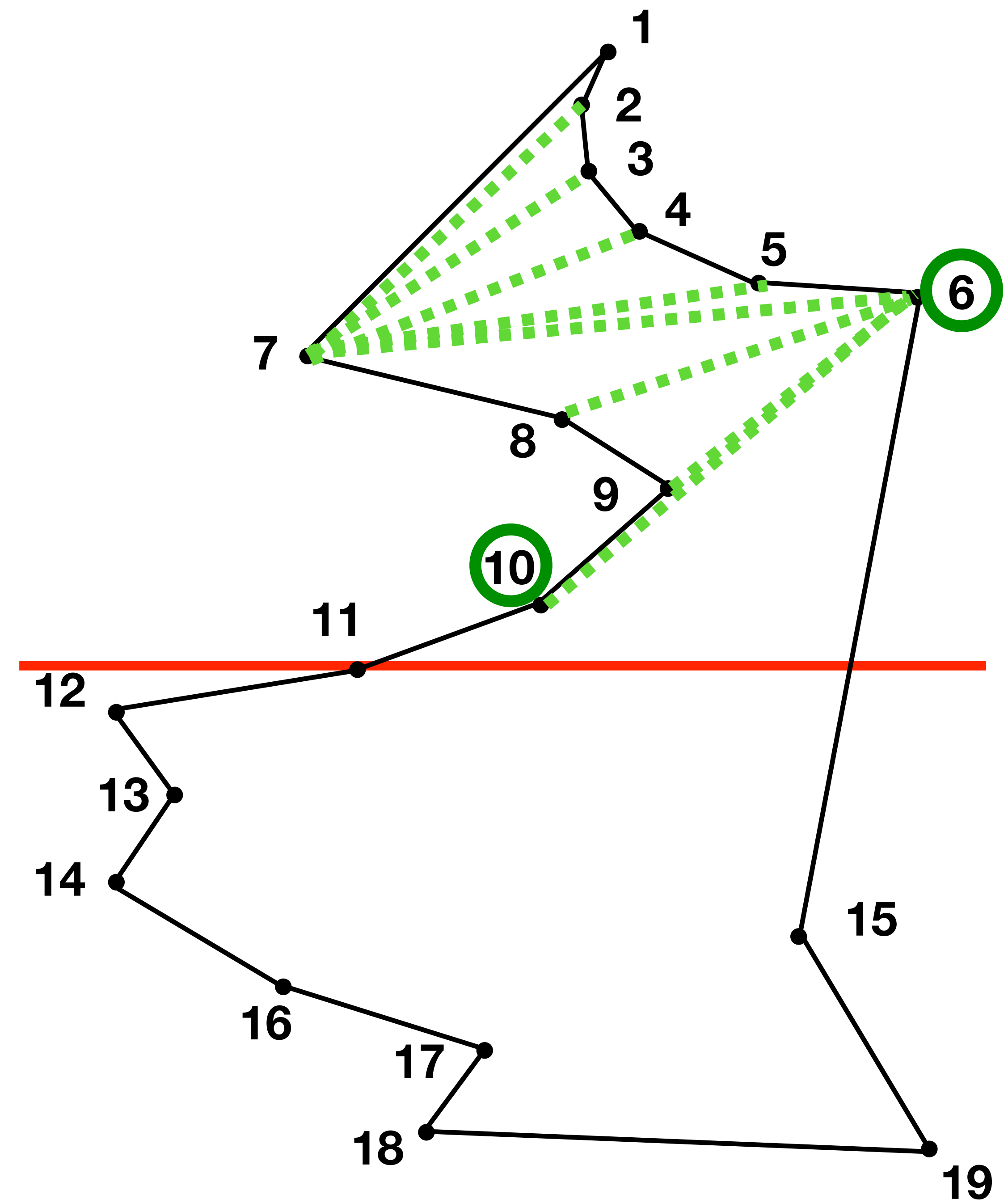
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



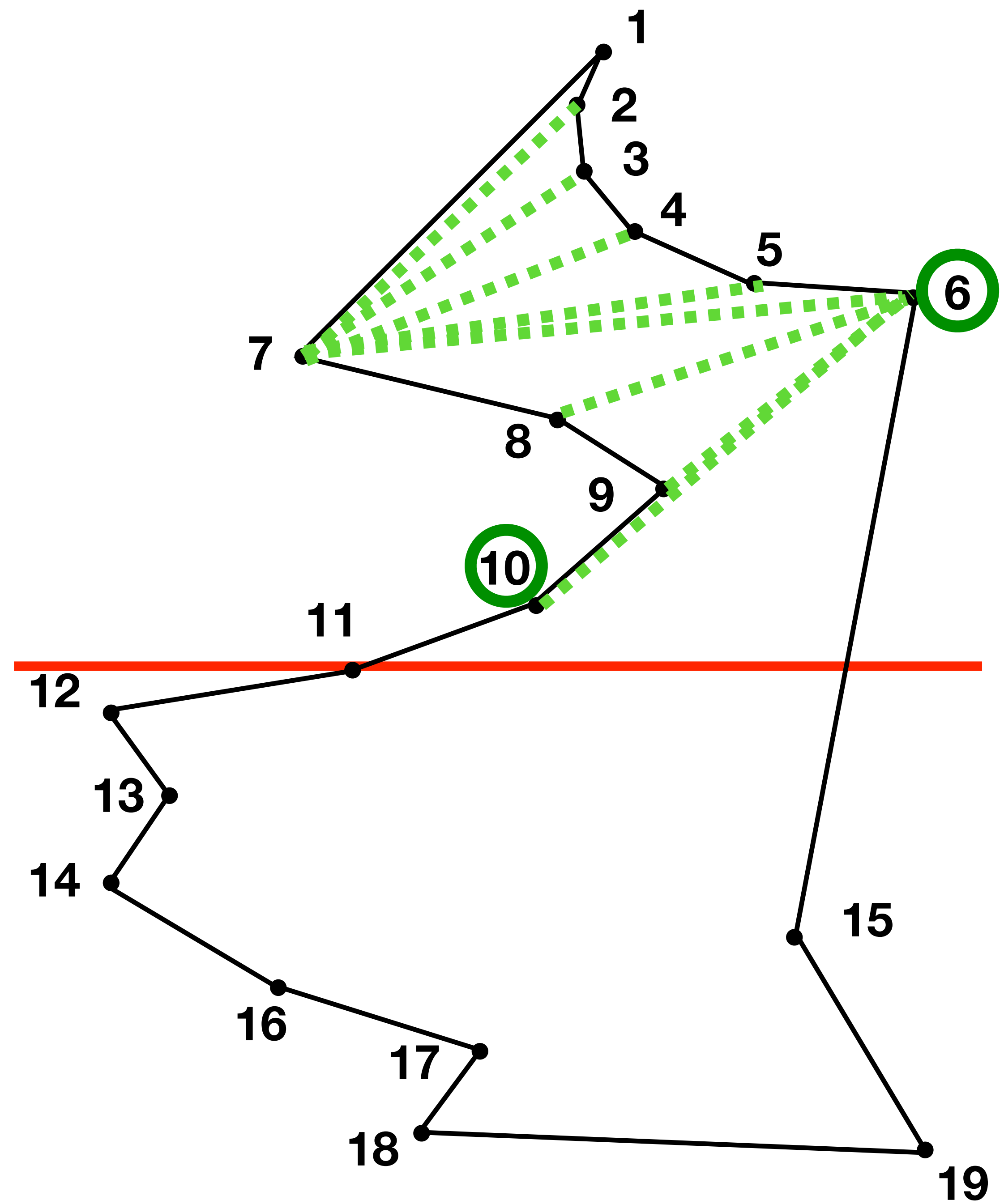
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

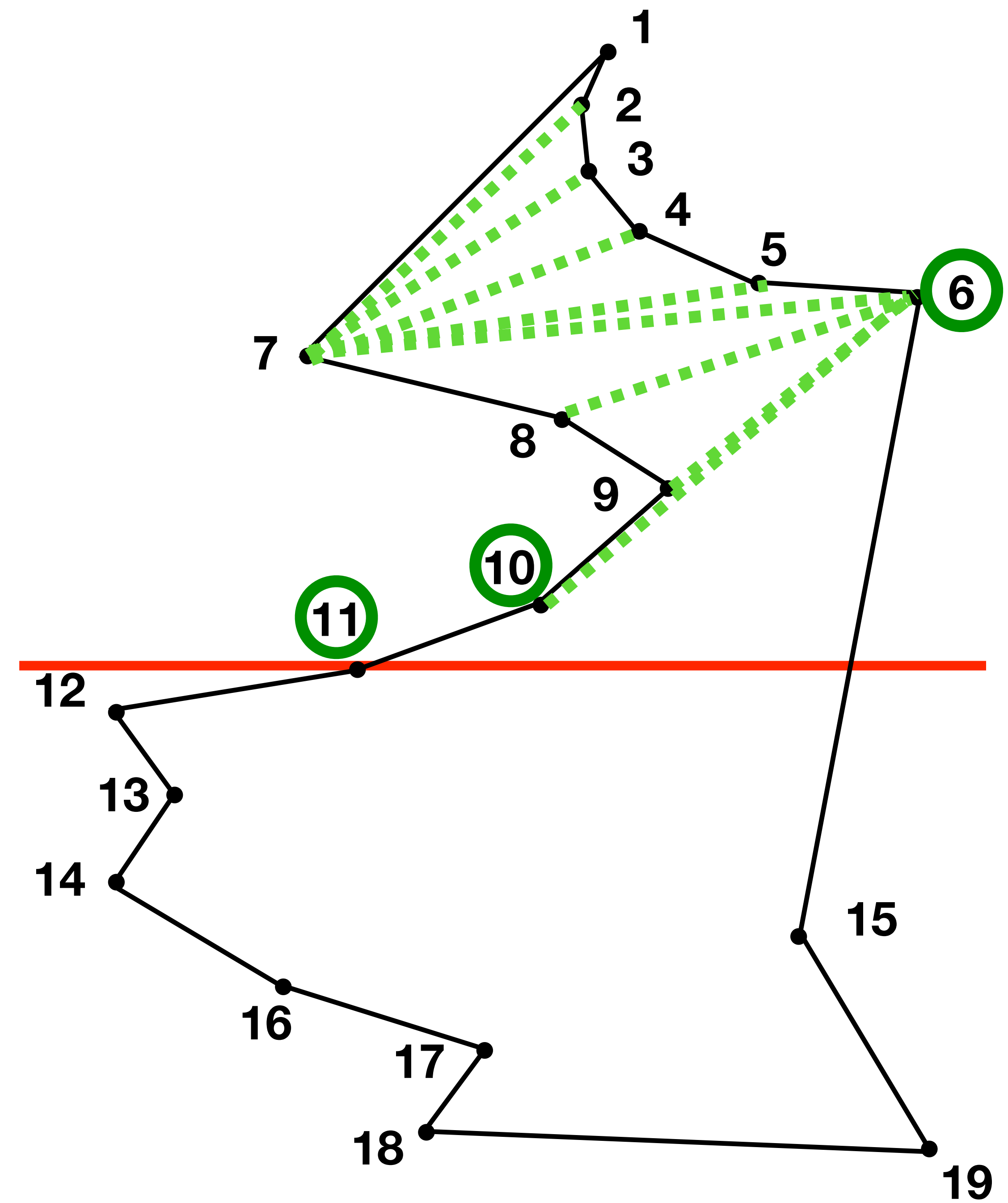
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

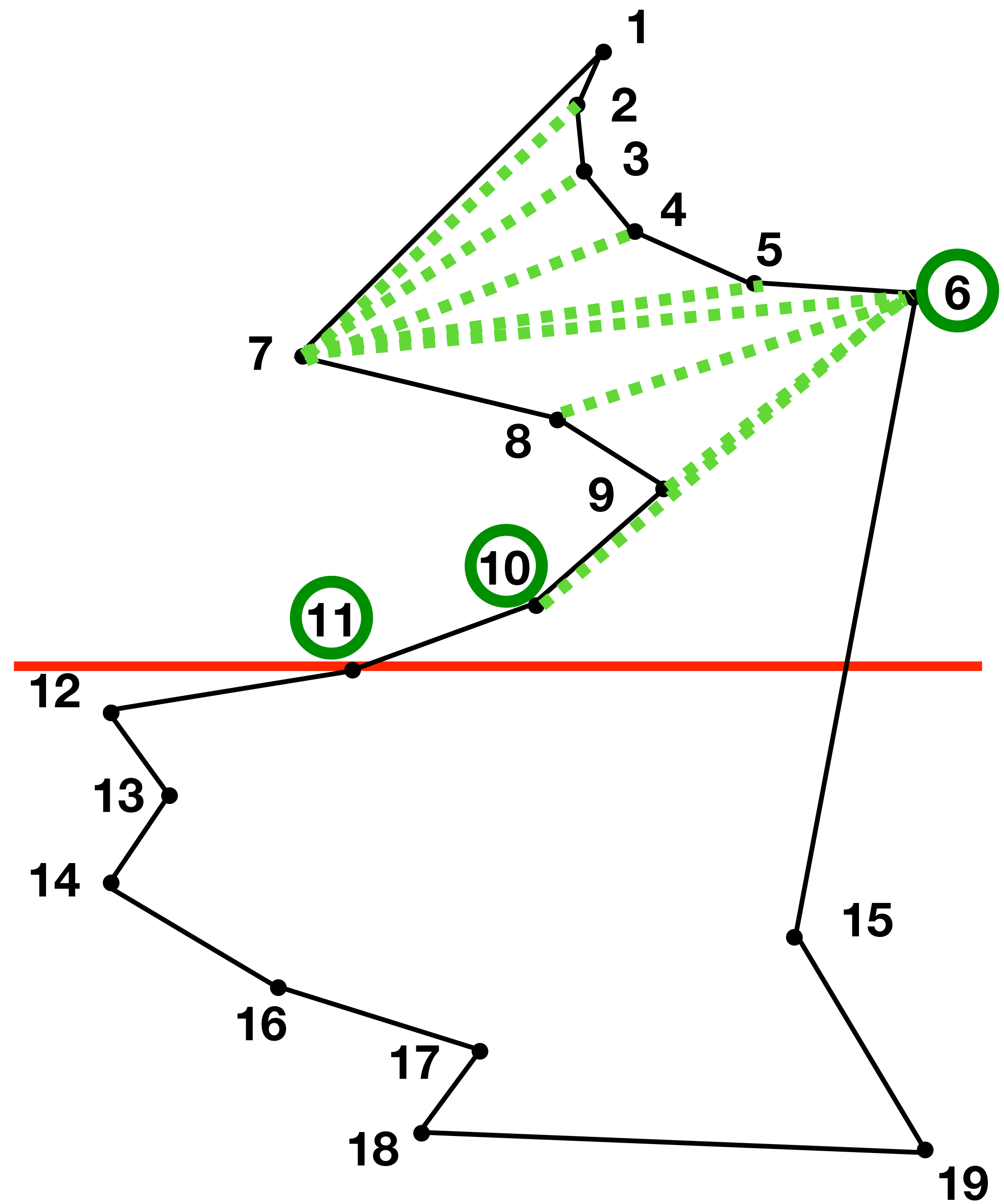
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — Chain

   3.2 If same chain, predecessor reflex:
   - Add to queue — Add

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — Ear

Queue: 6 10 11



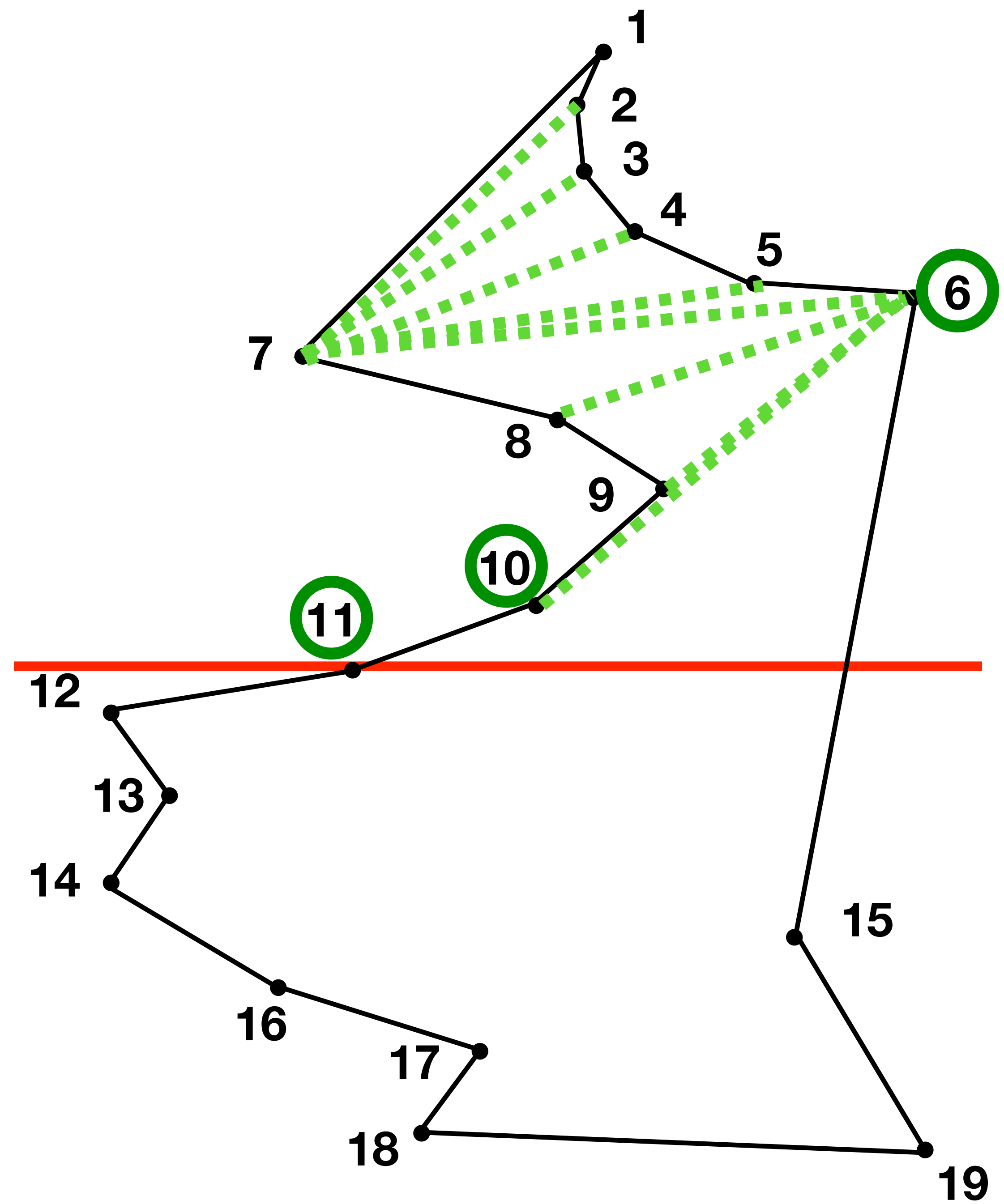
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 11



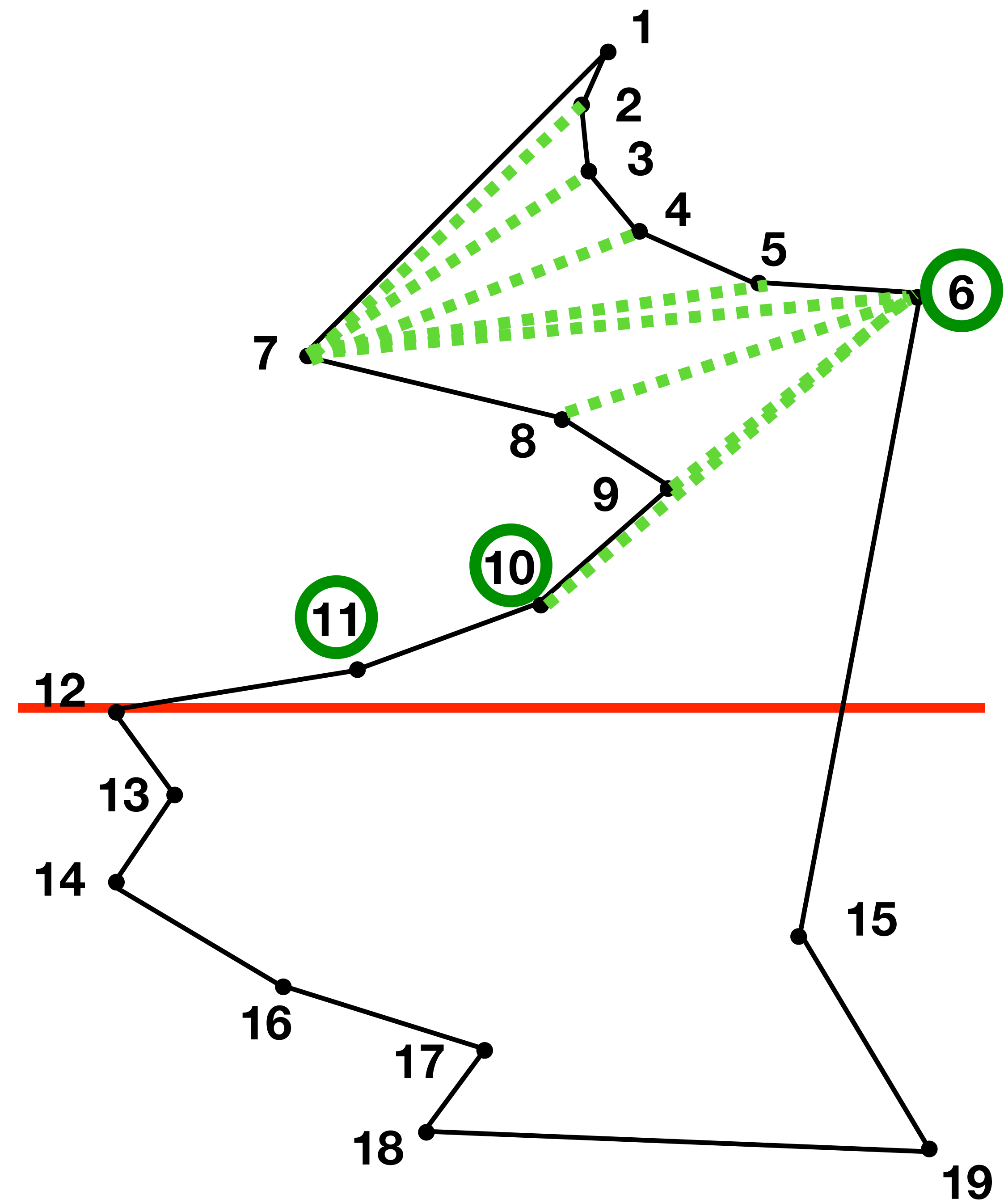
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 11



Algorithm 5.16

Input: A y-monotone polygon P

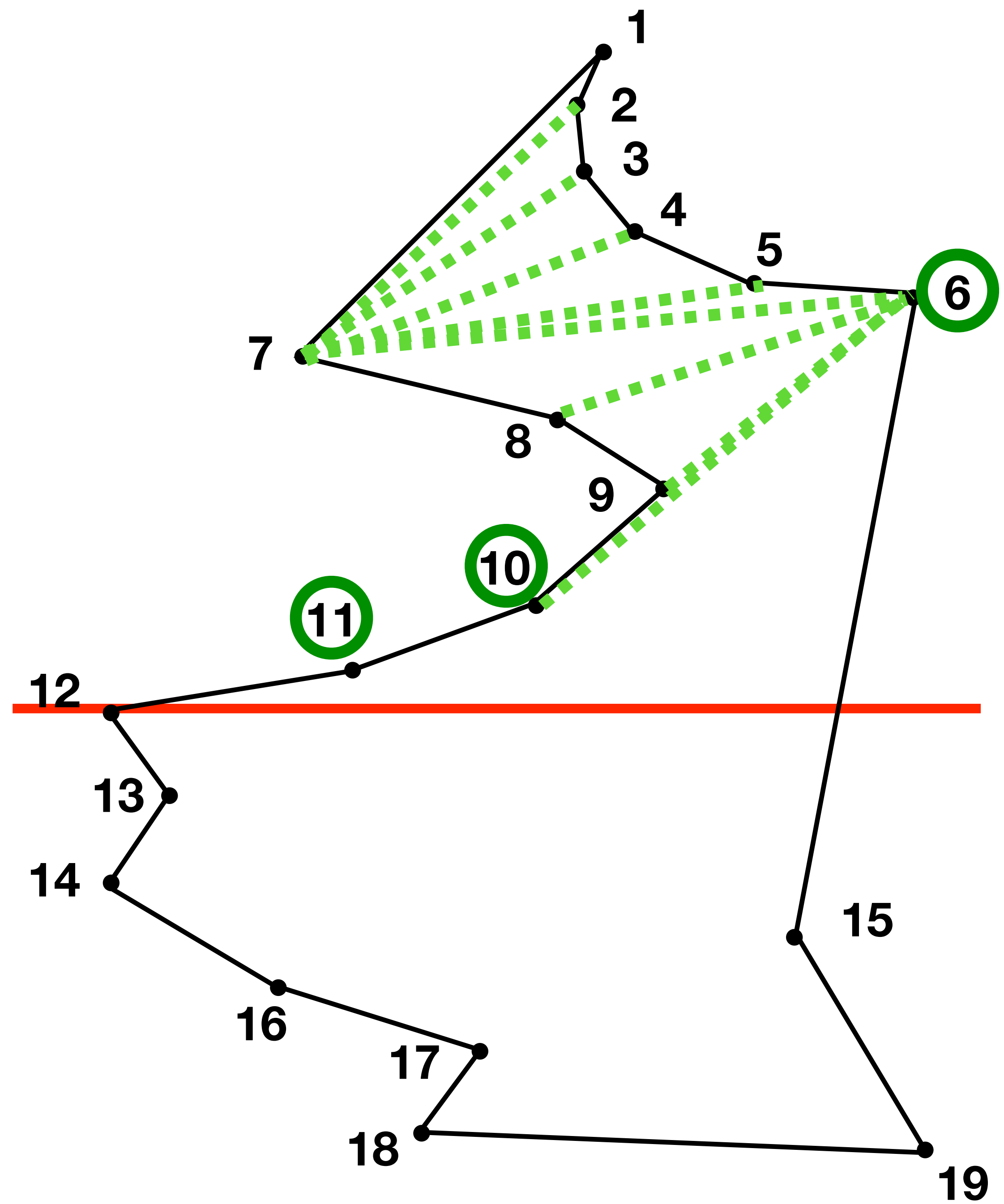
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11



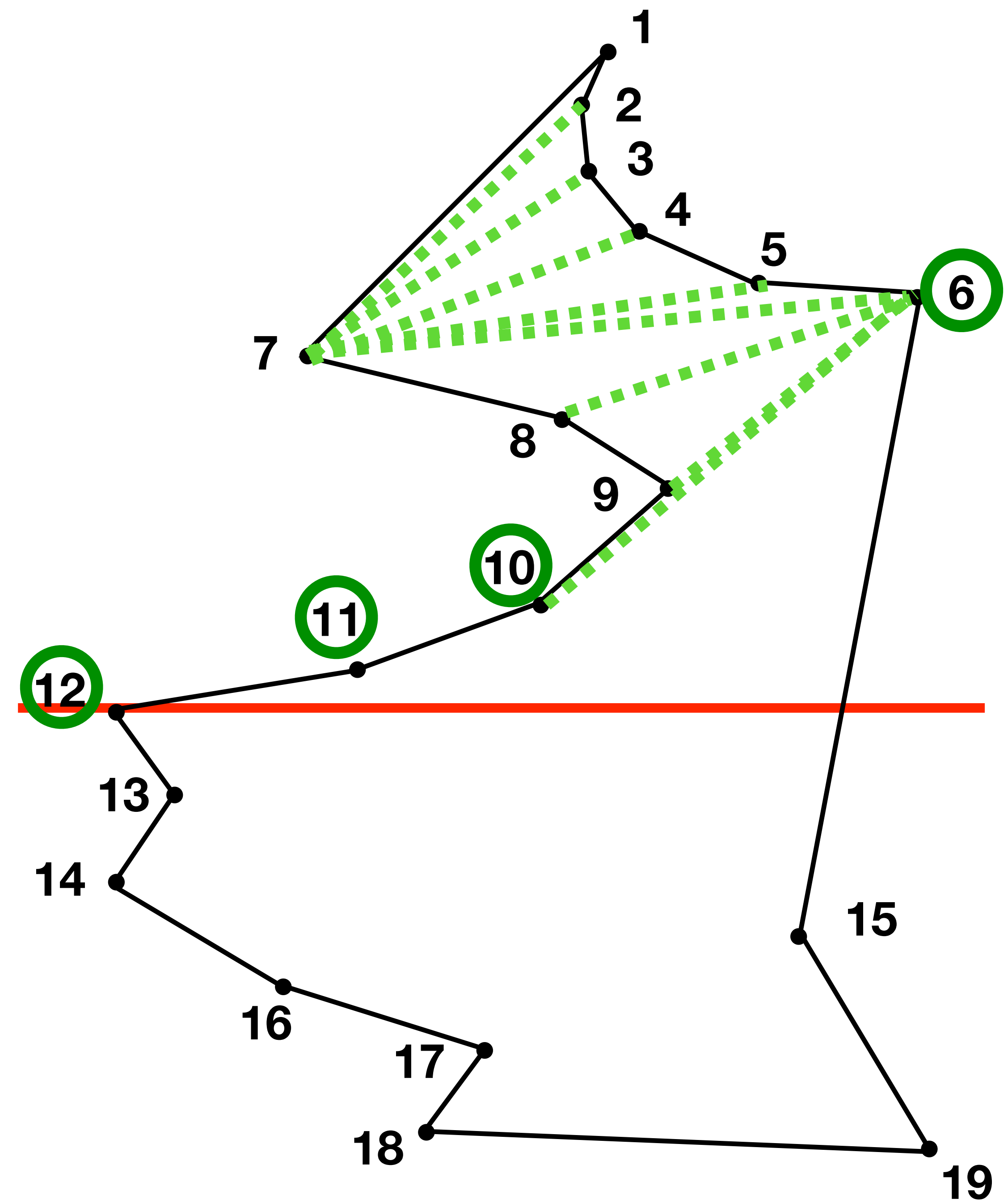
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 11



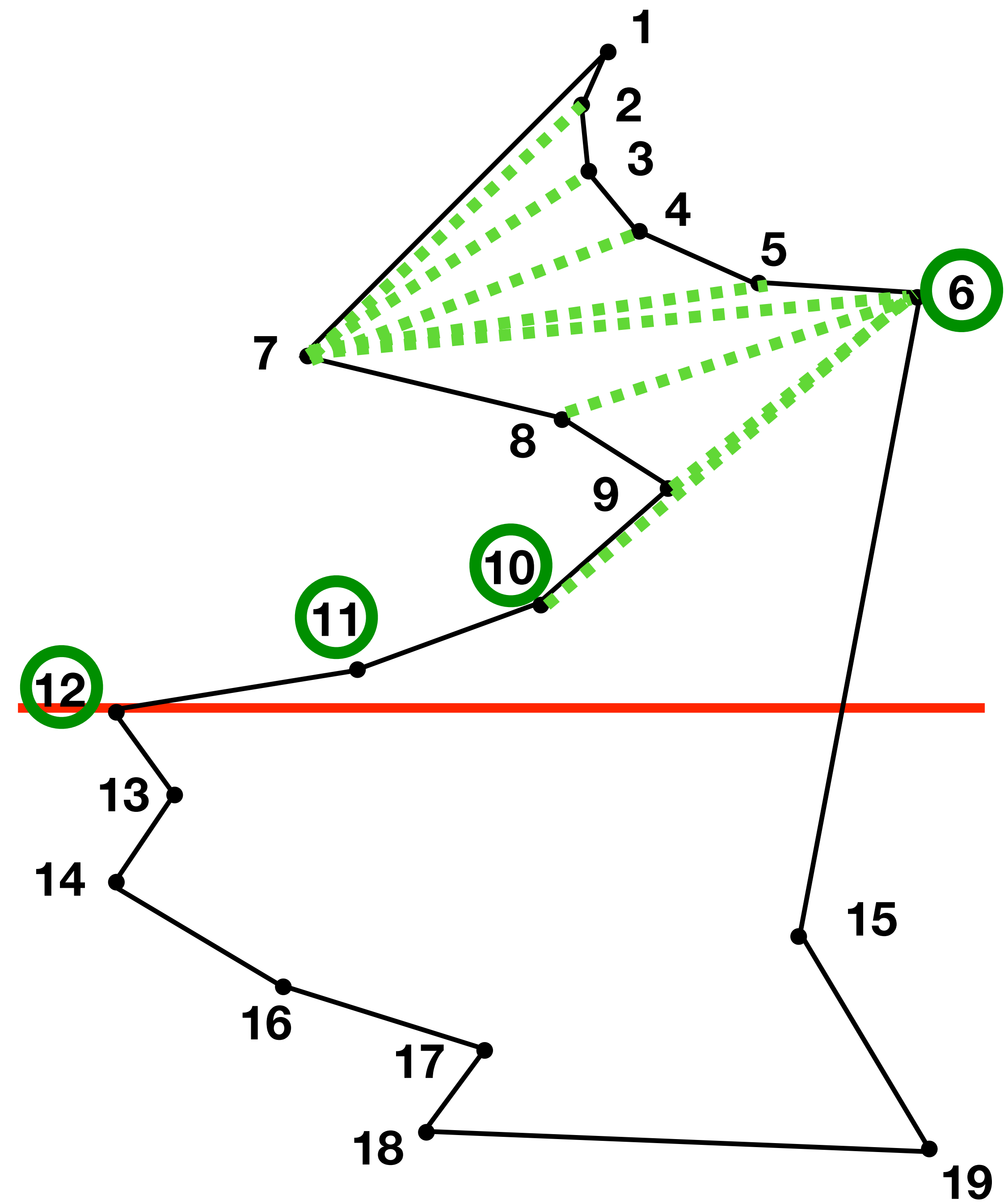
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10 11 12



Algorithm 5.16

Input: A y-monotone polygon P

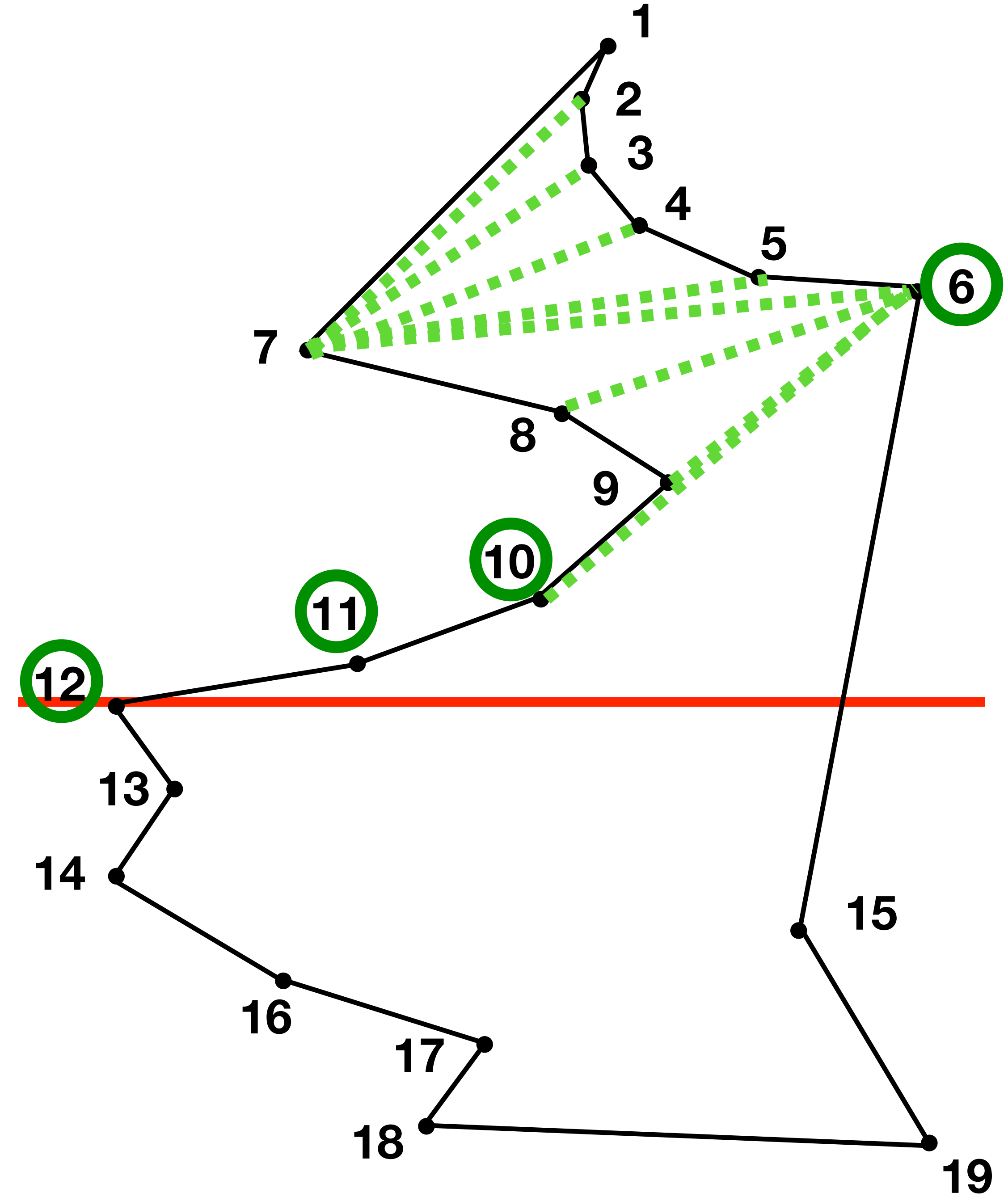
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11 12



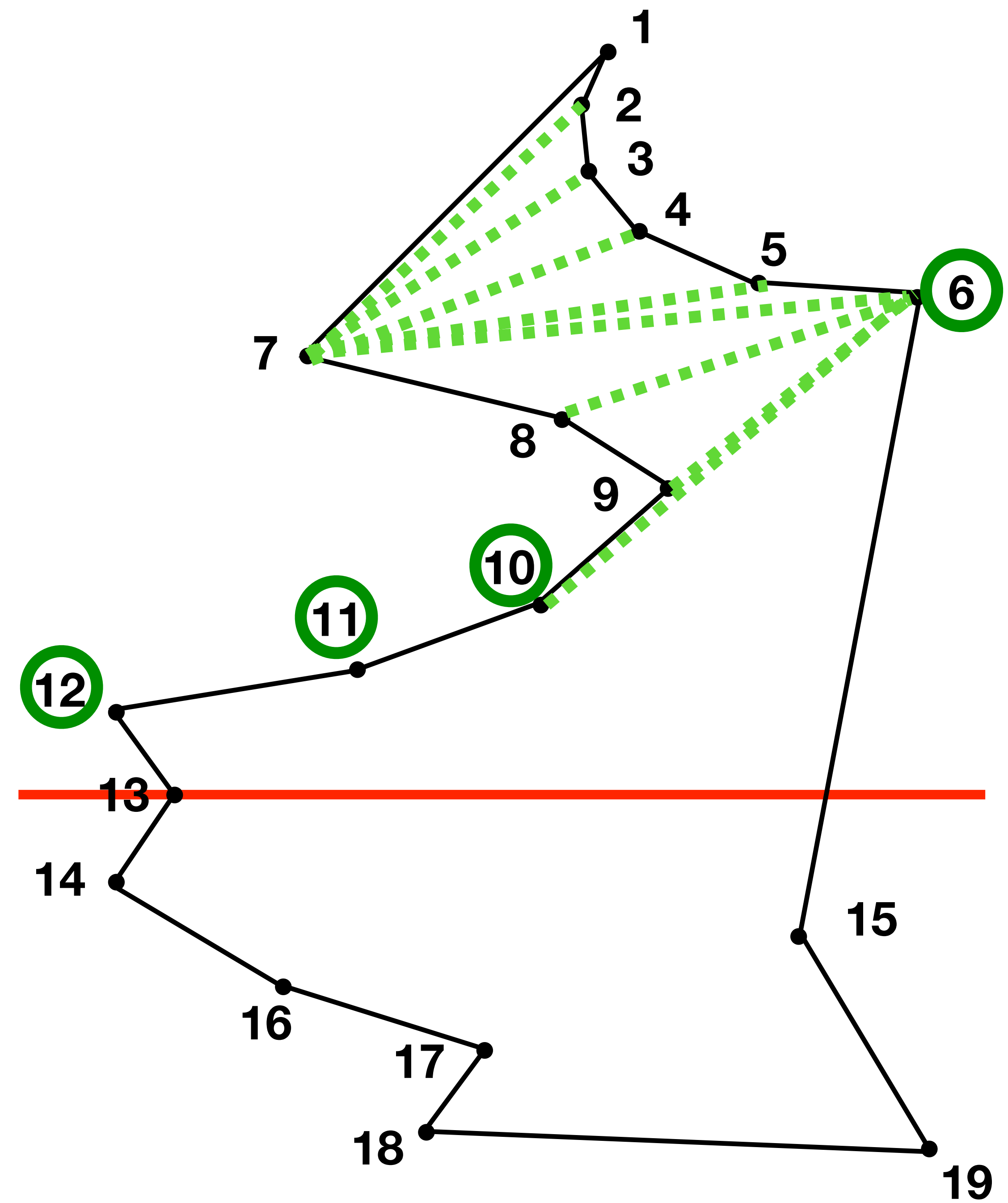
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 11 12



Algorithm 5.16

Input: A y-monotone polygon P

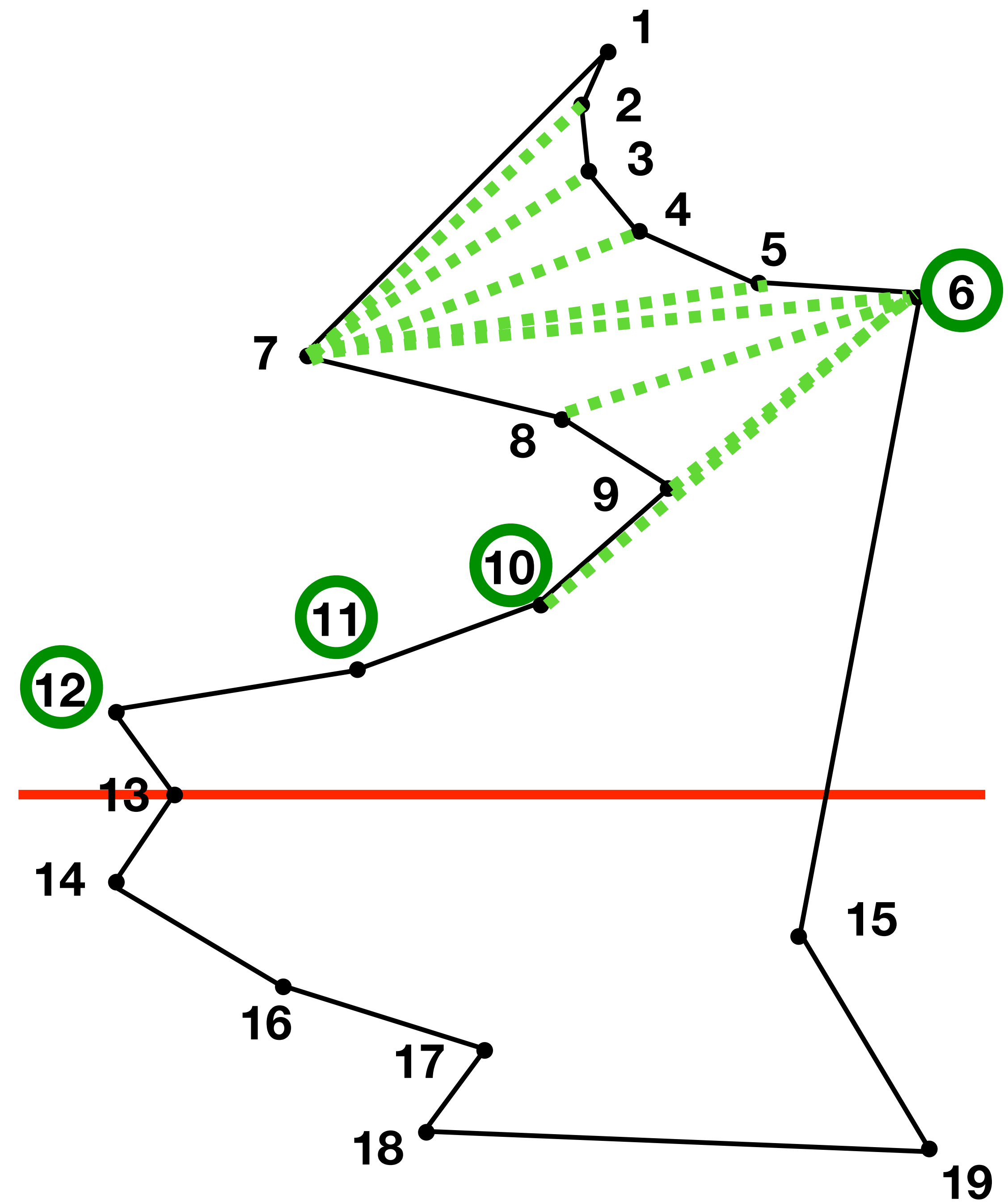
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11 12



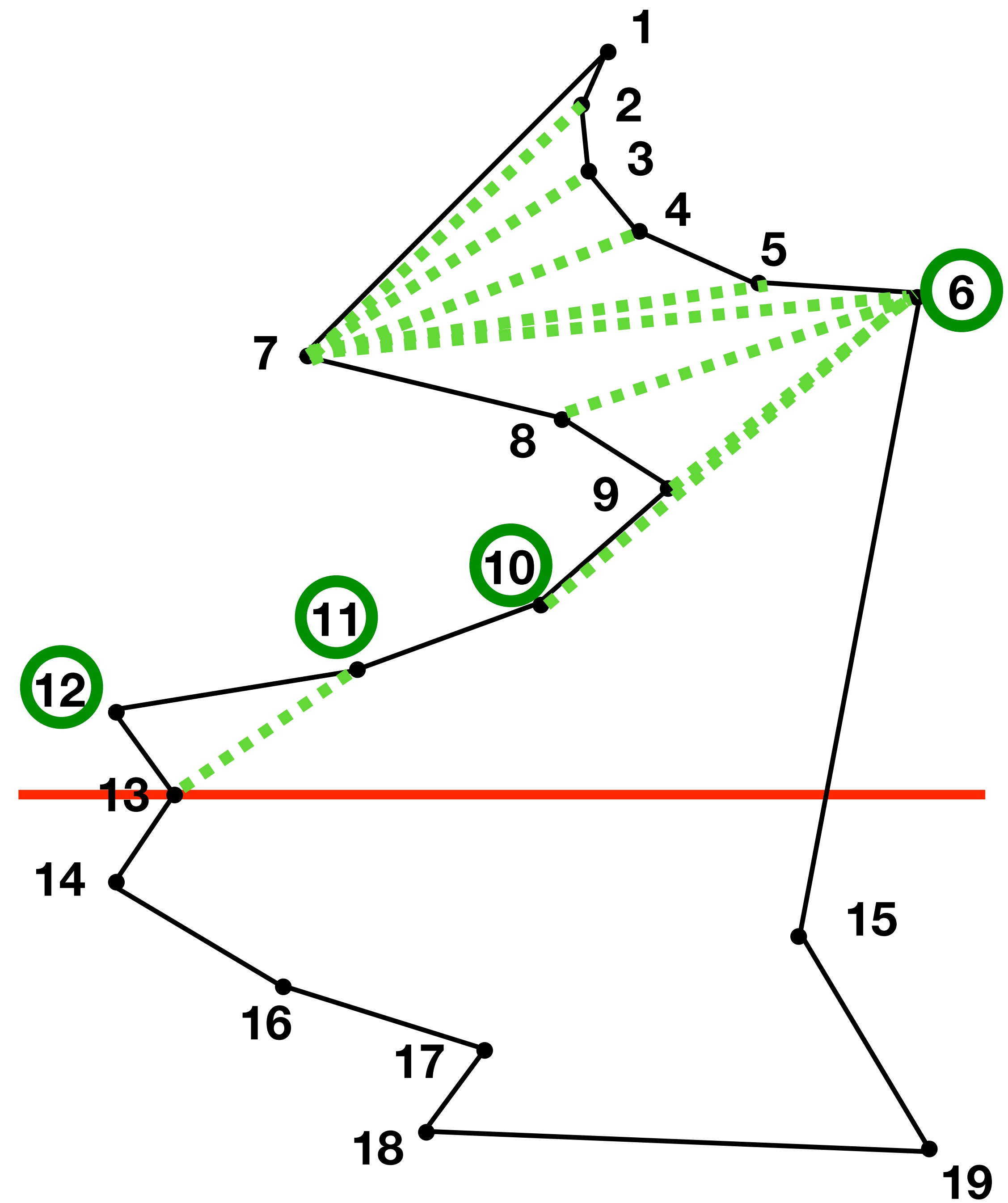
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 11 12



Algorithm 5.16

Input: A y-monotone polygon P

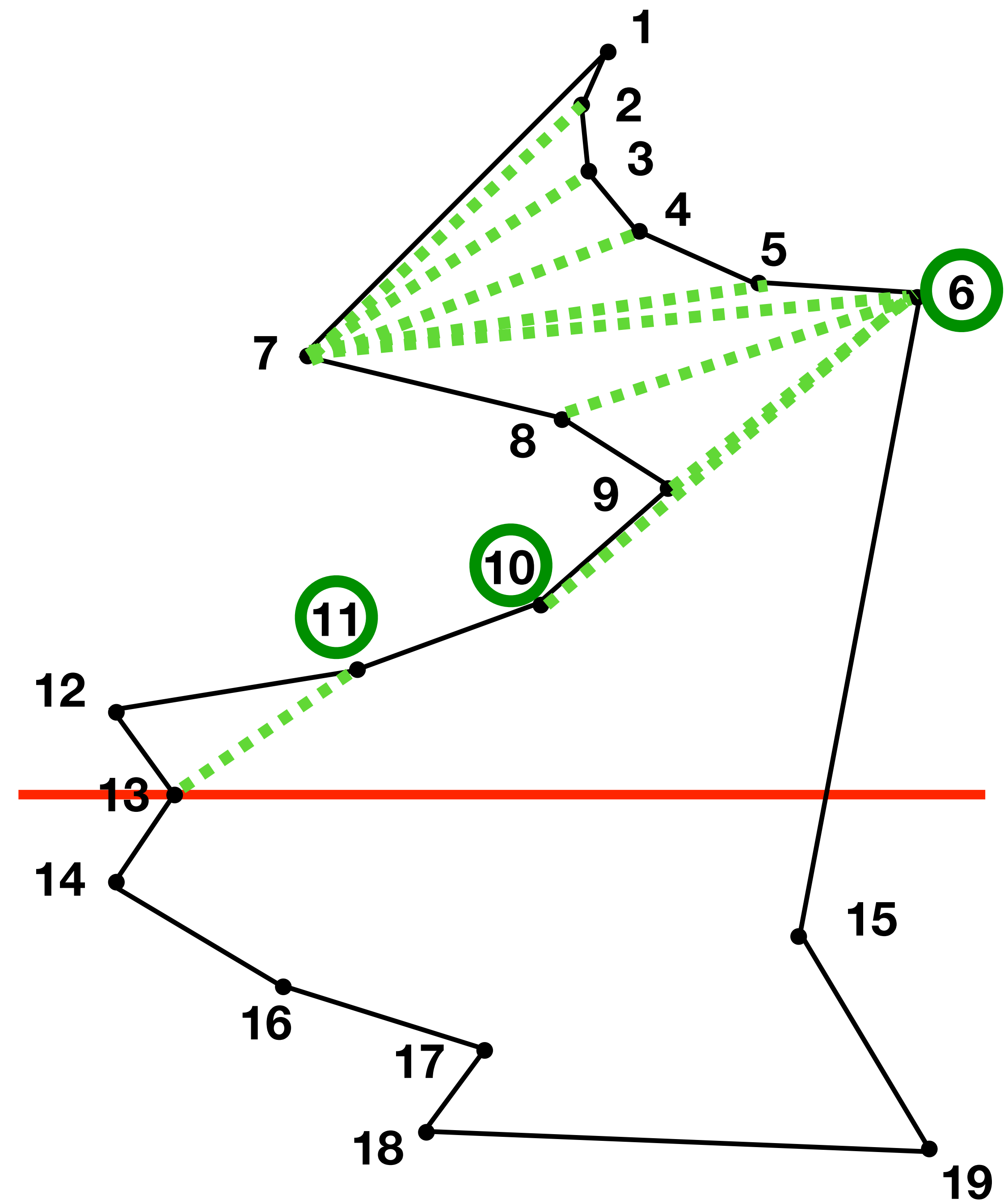
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11 12



Algorithm 5.16

Input: A y-monotone polygon P

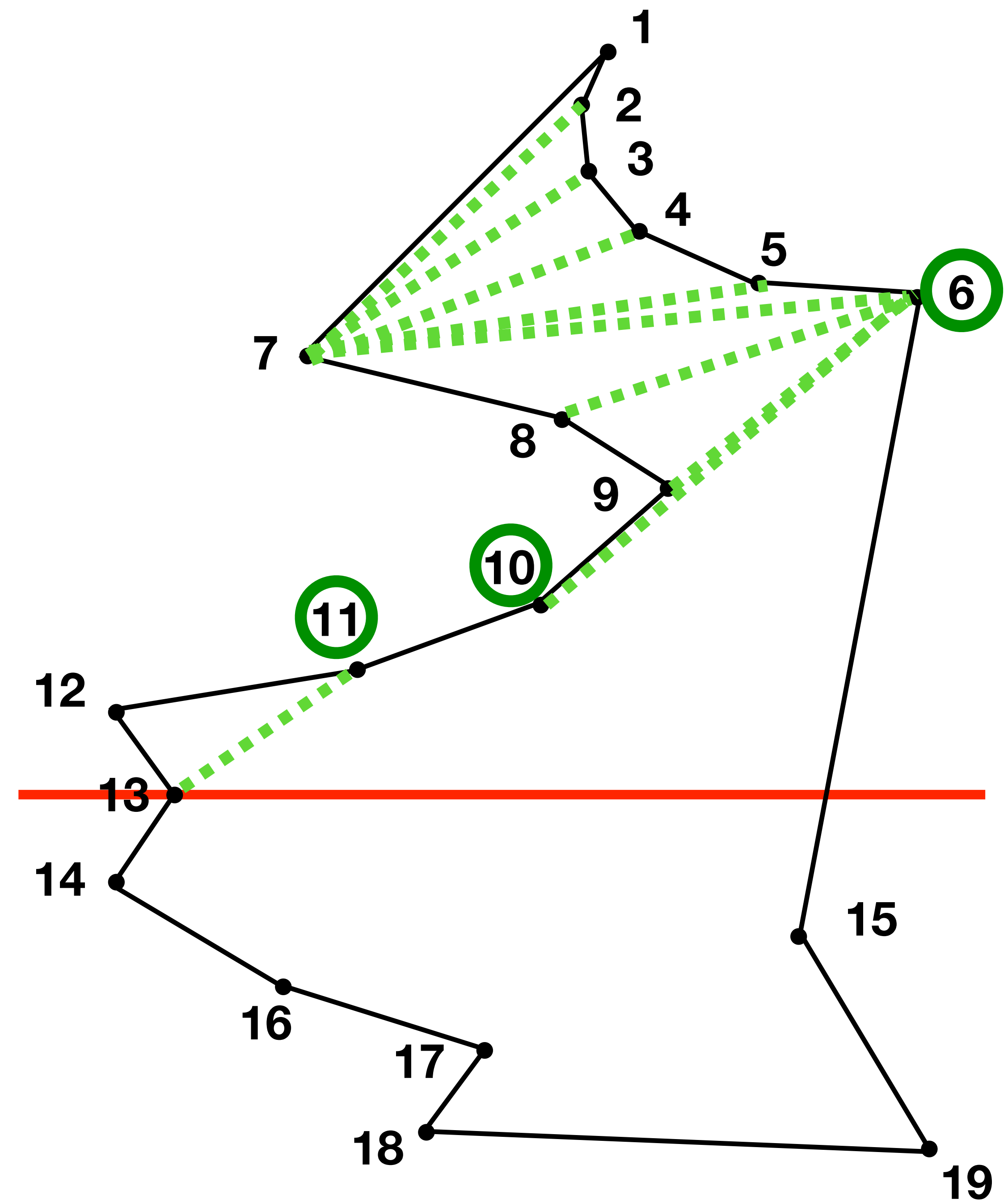
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — Chain

   3.2 If same chain, predecessor reflex:
   - Add to queue — Add

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — Ear

Queue: 6 10 11



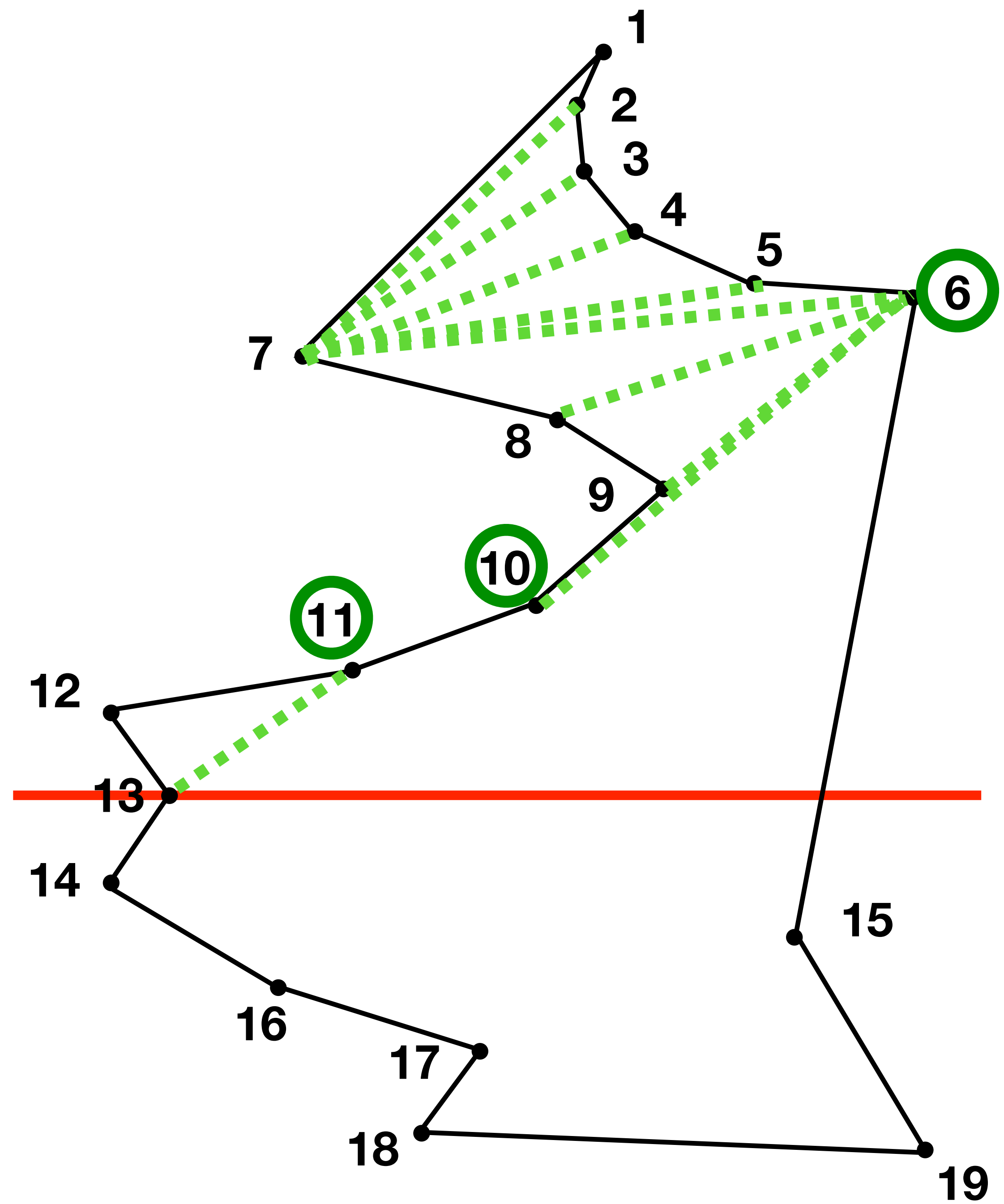
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10 11



Algorithm 5.16

Input: A y-monotone polygon P

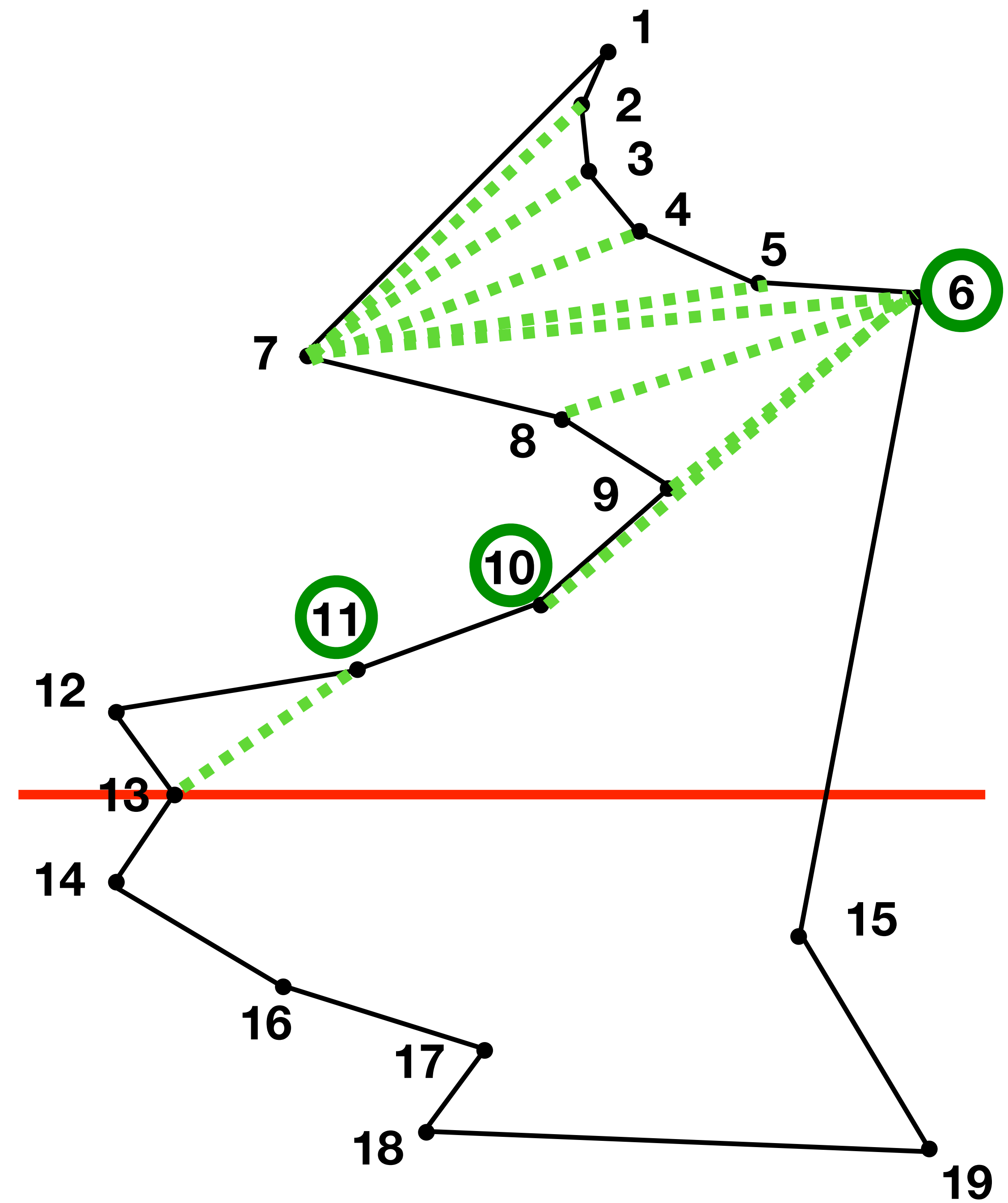
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11



Algorithm 5.16

Input: A y-monotone polygon P

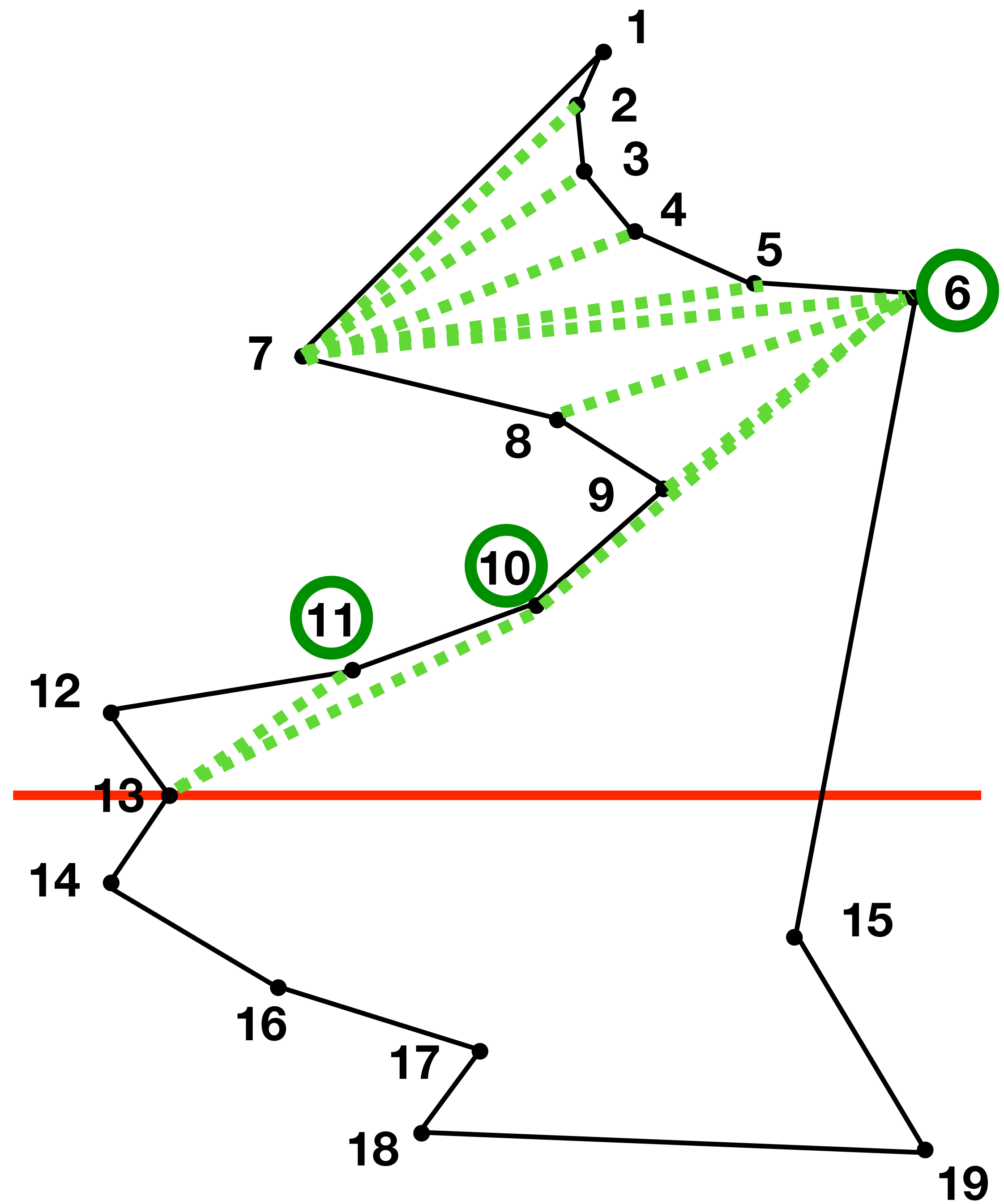
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 11



Algorithm 5.16

Input: A y-monotone polygon P

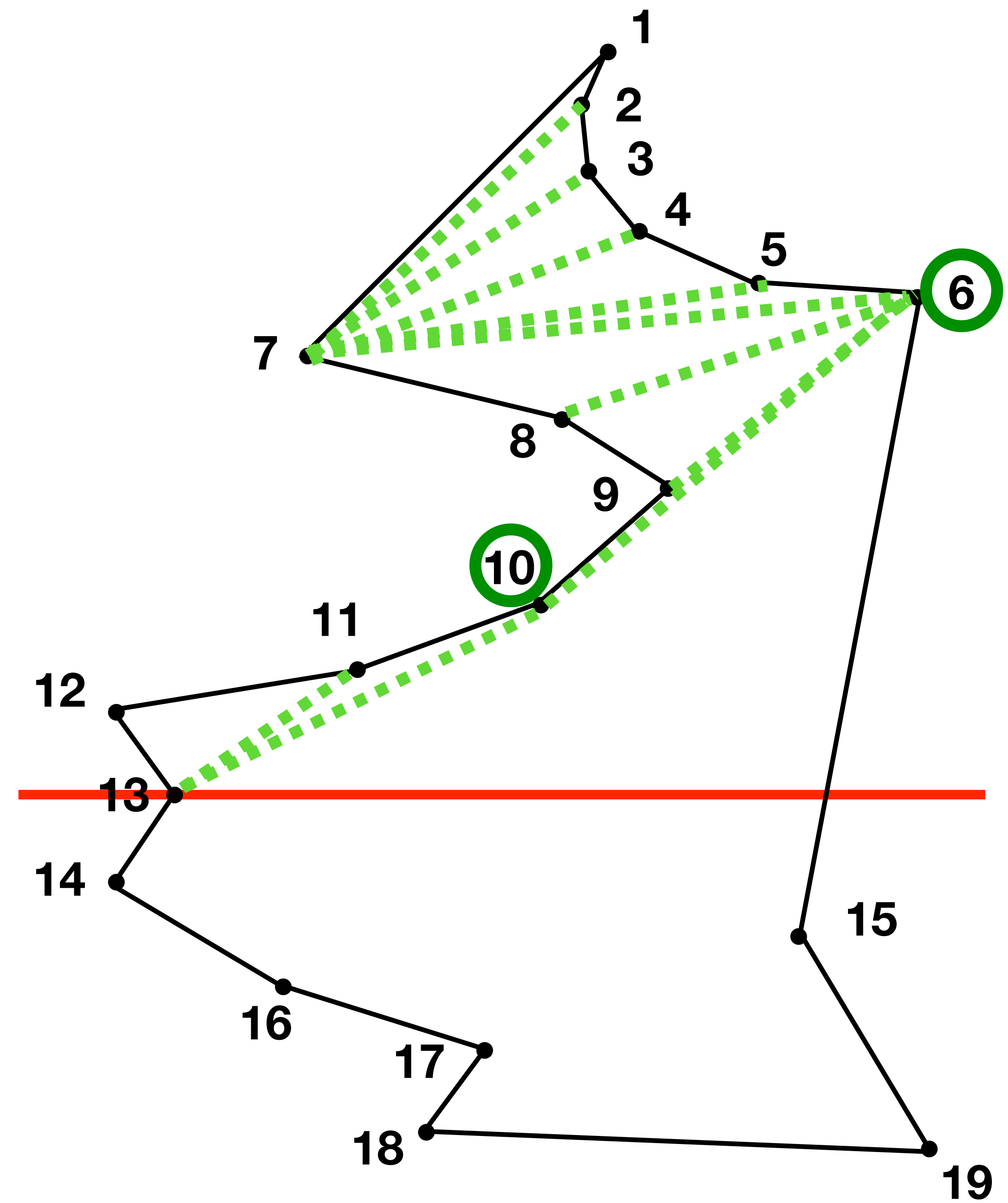
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 6 10 11



Algorithm 5.16

Input: A y-monotone polygon P

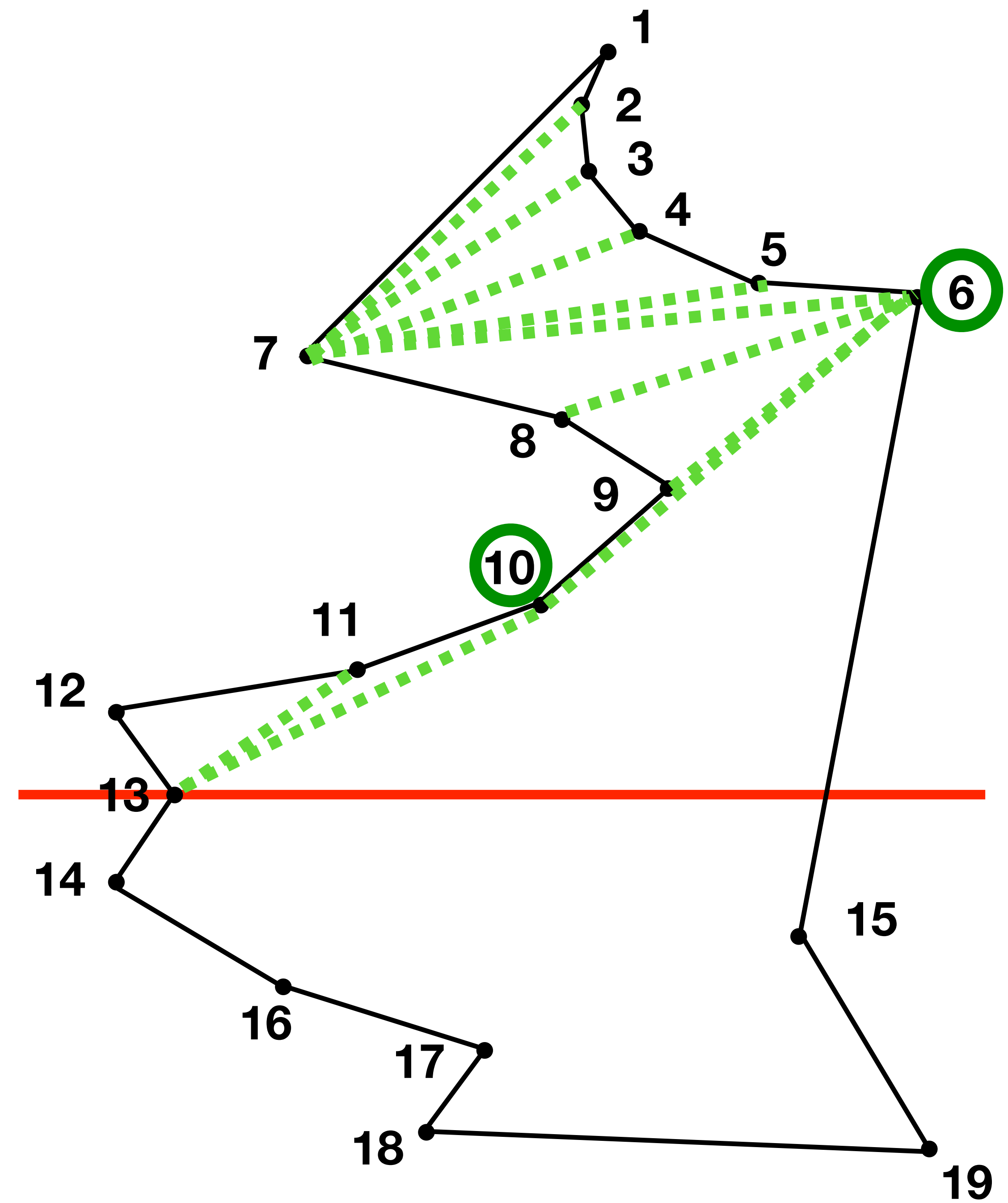
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



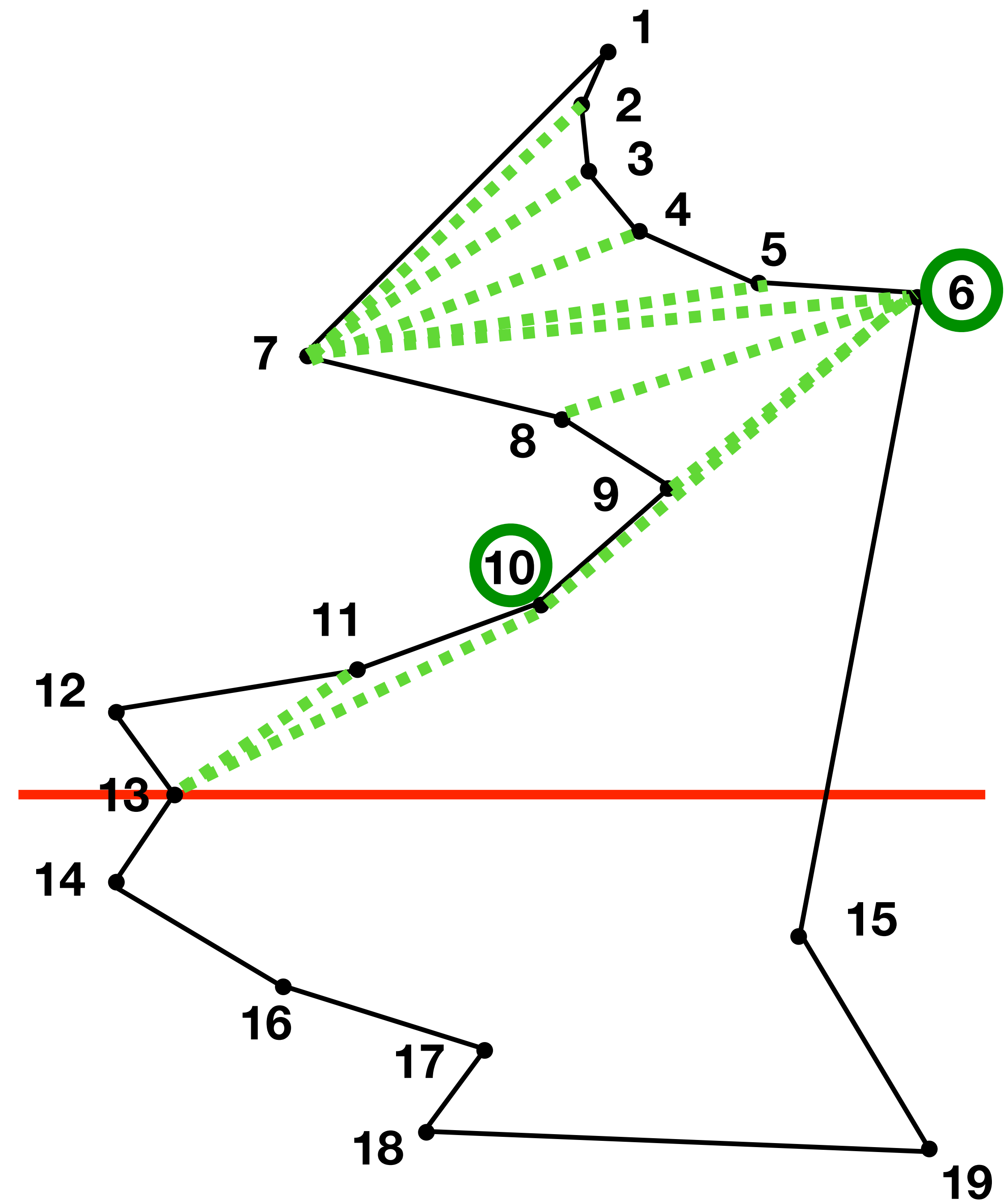
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

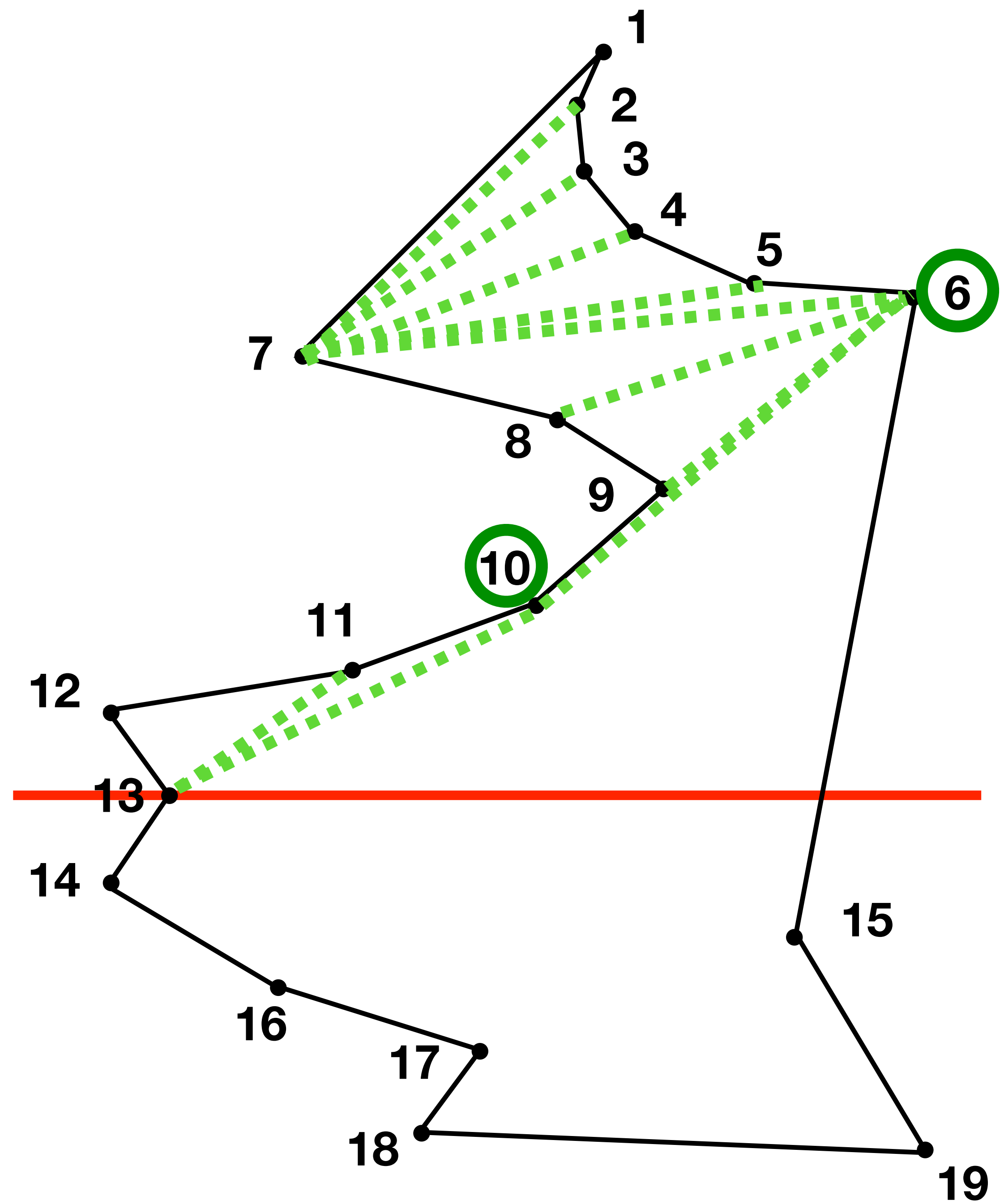
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

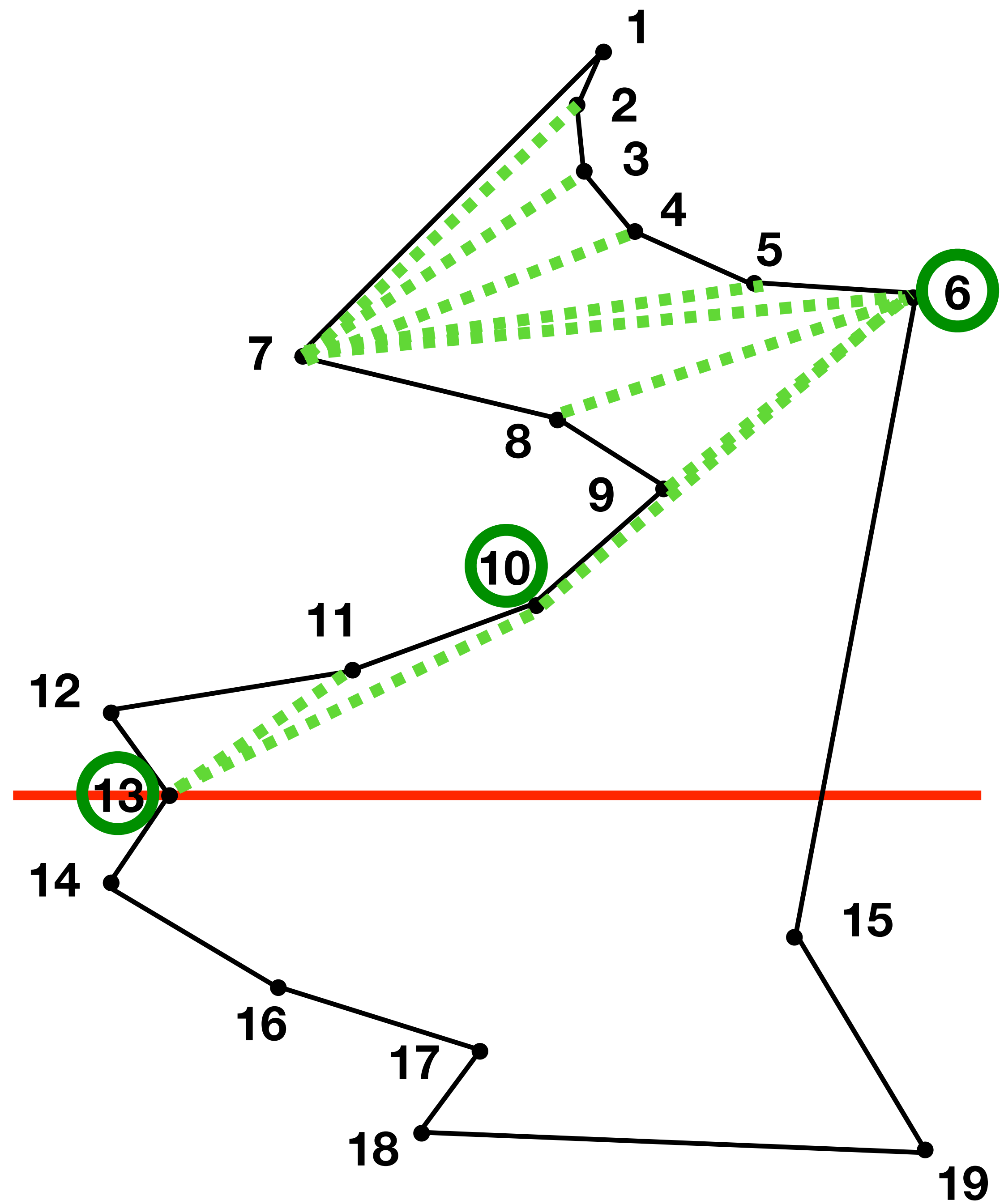
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

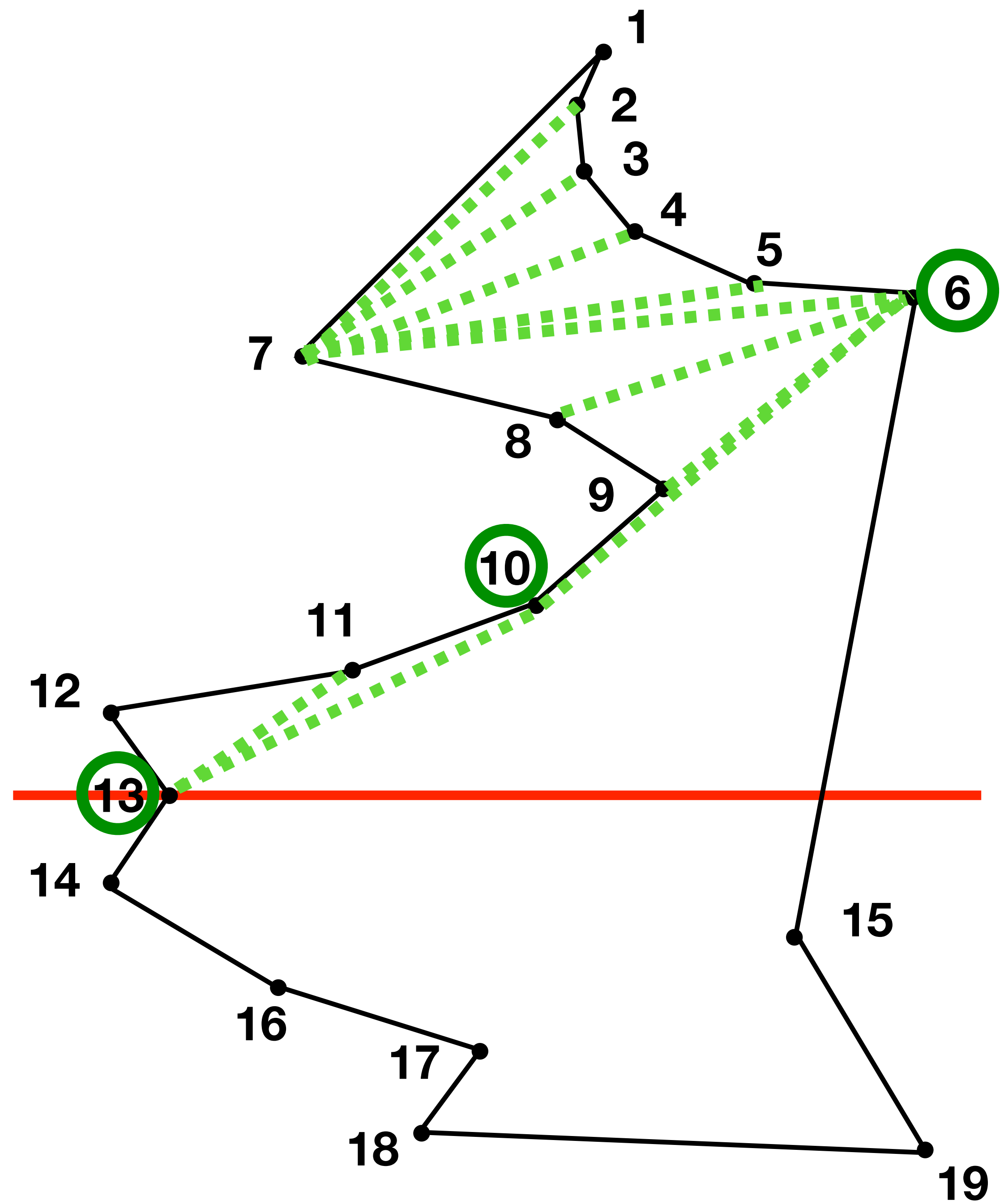
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 13



Algorithm 5.16

Input: A y-monotone polygon P

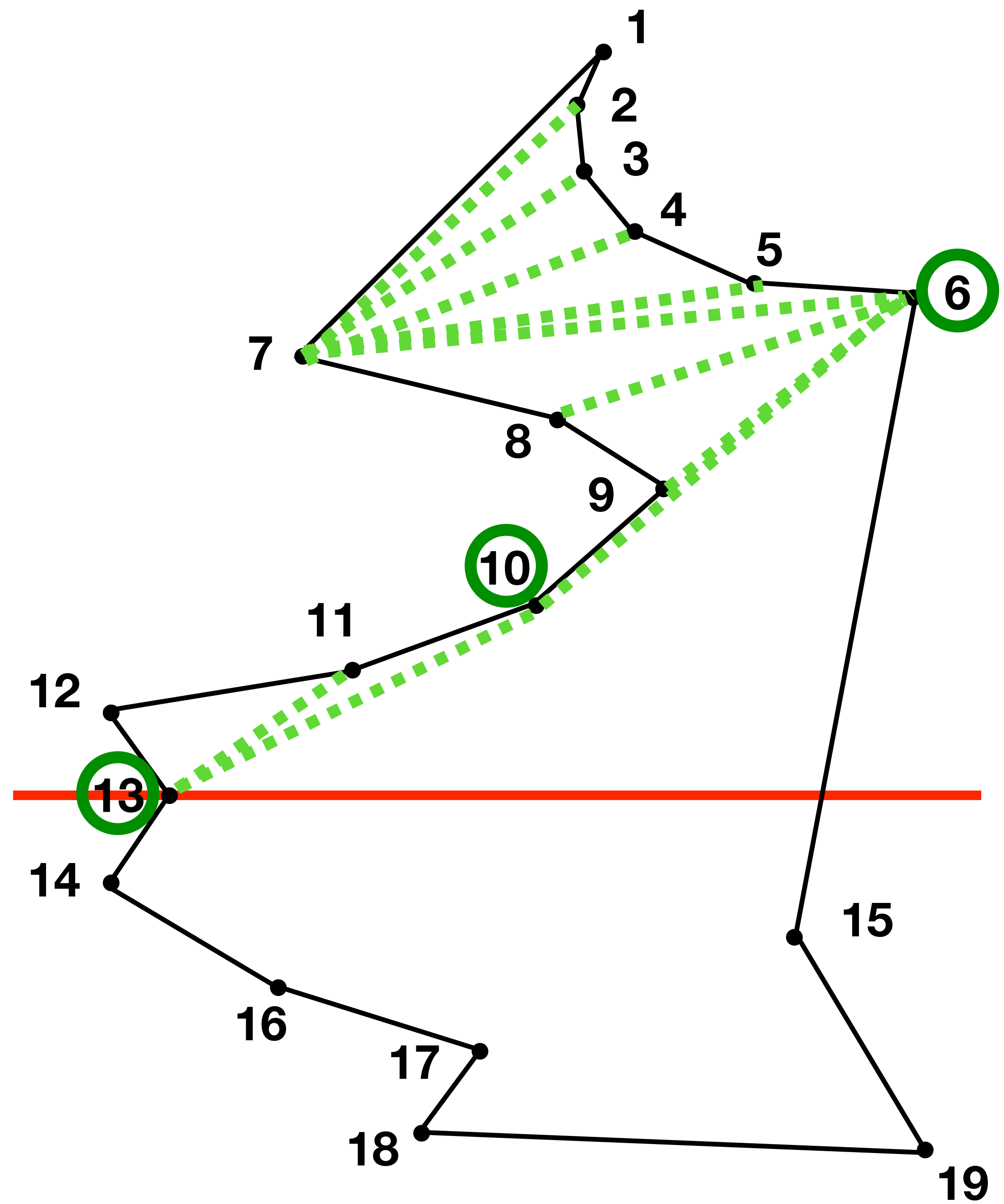
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 13



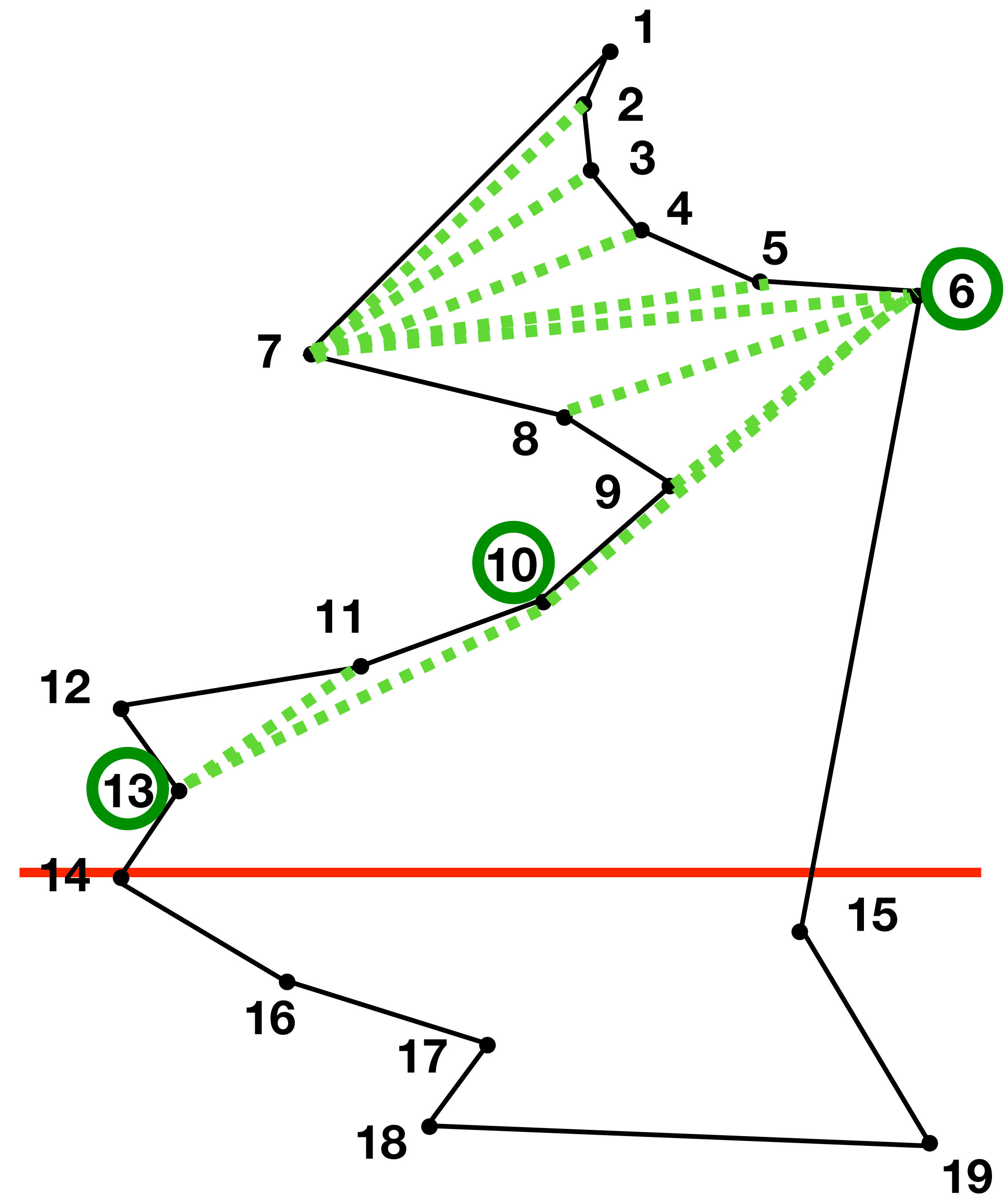
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 13



Algorithm 5.16

Input: A y-monotone polygon P

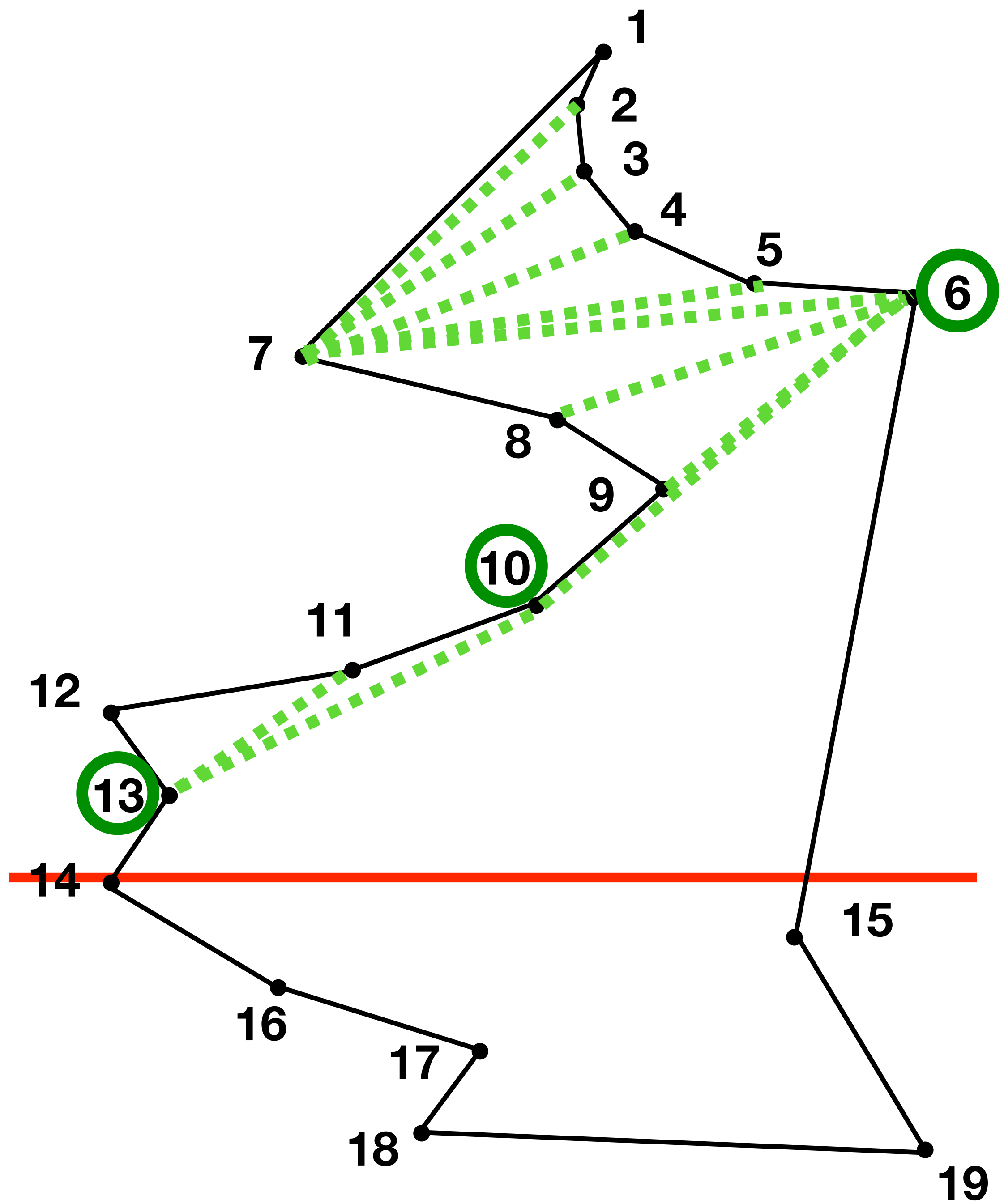
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 13



Algorithm 5.16

Input: A y-monotone polygon P

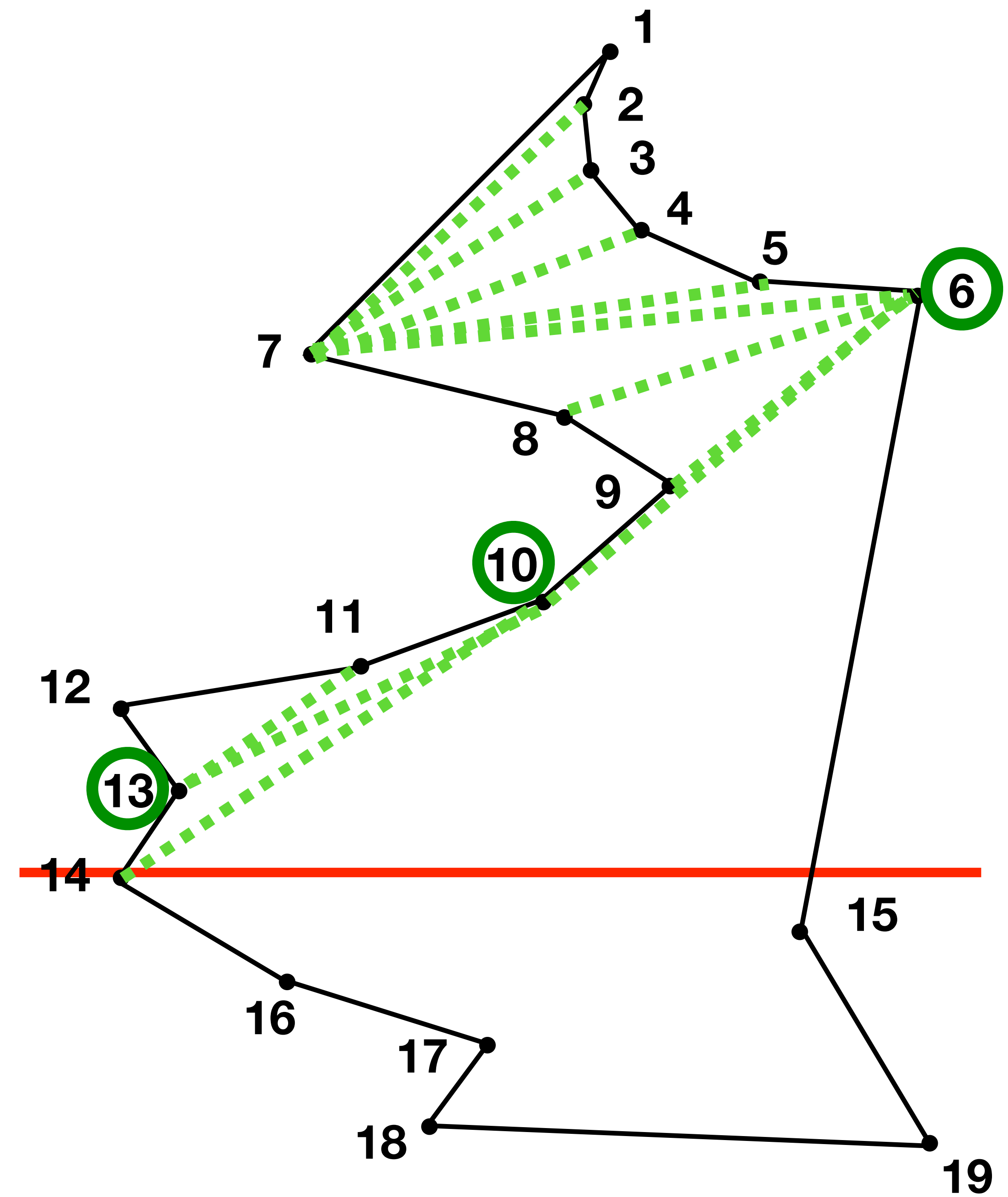
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 6 10 13



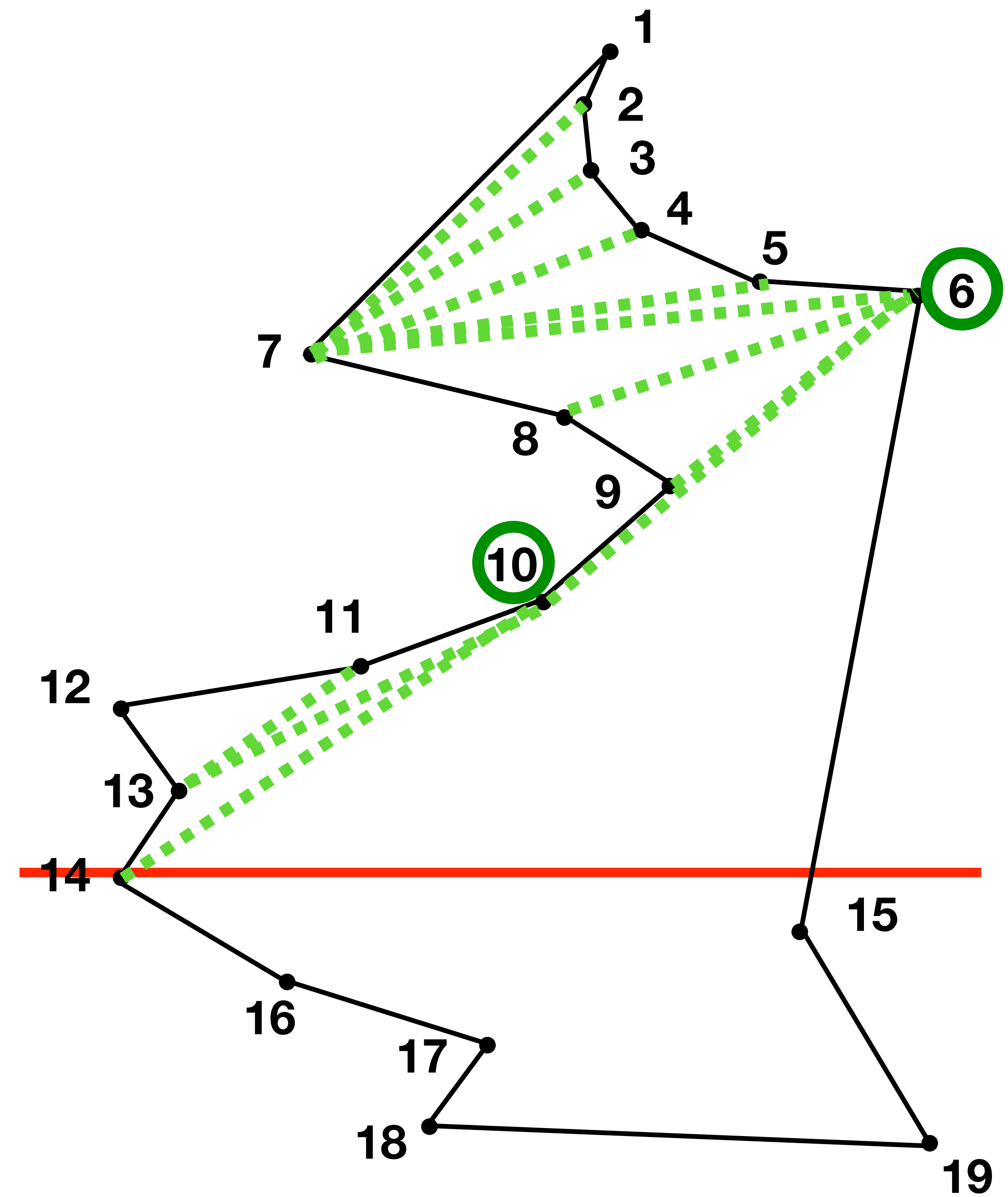
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10 13



Algorithm 5.16

Input: A y-monotone polygon P

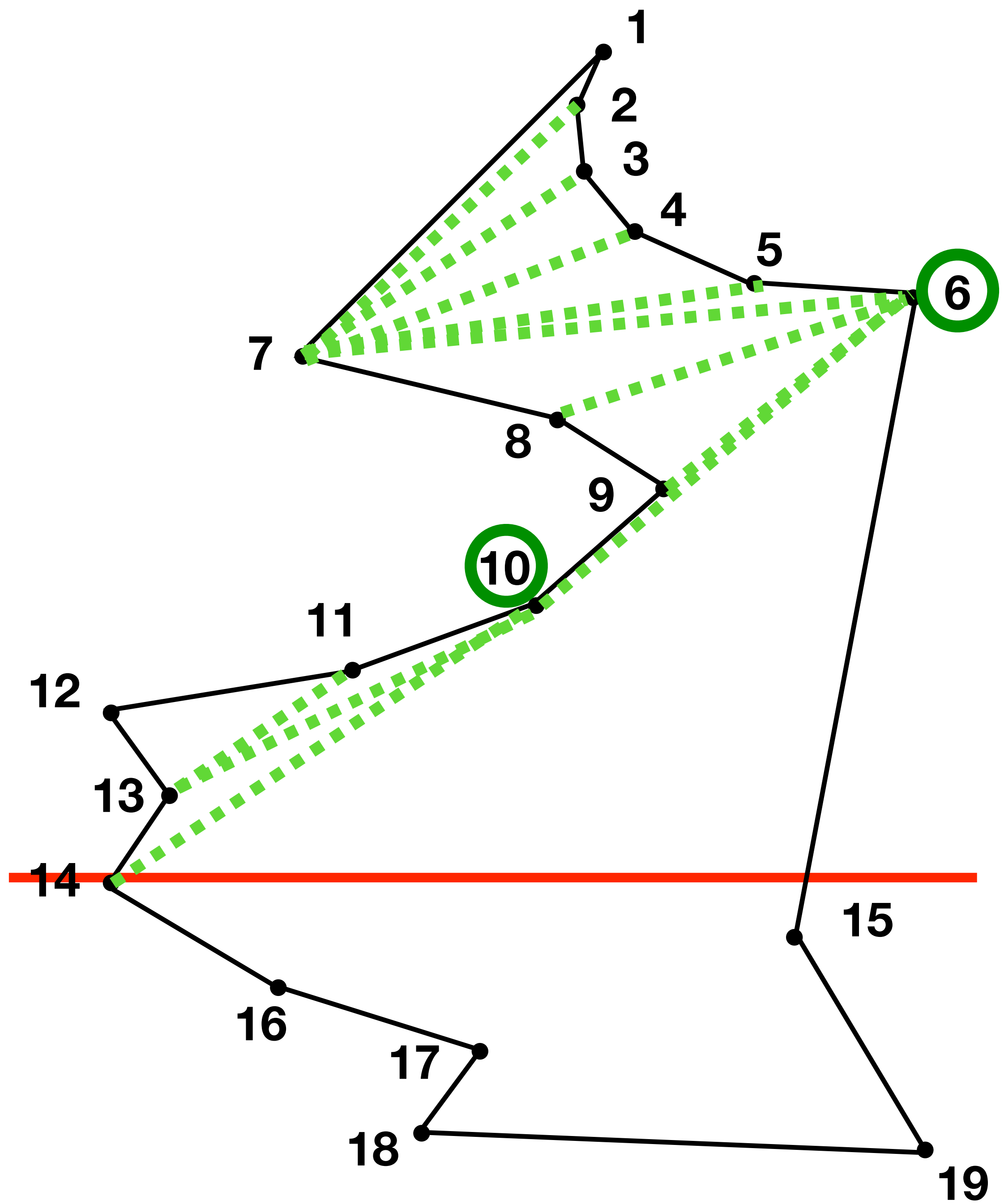
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



Technische Universität Braunschweig

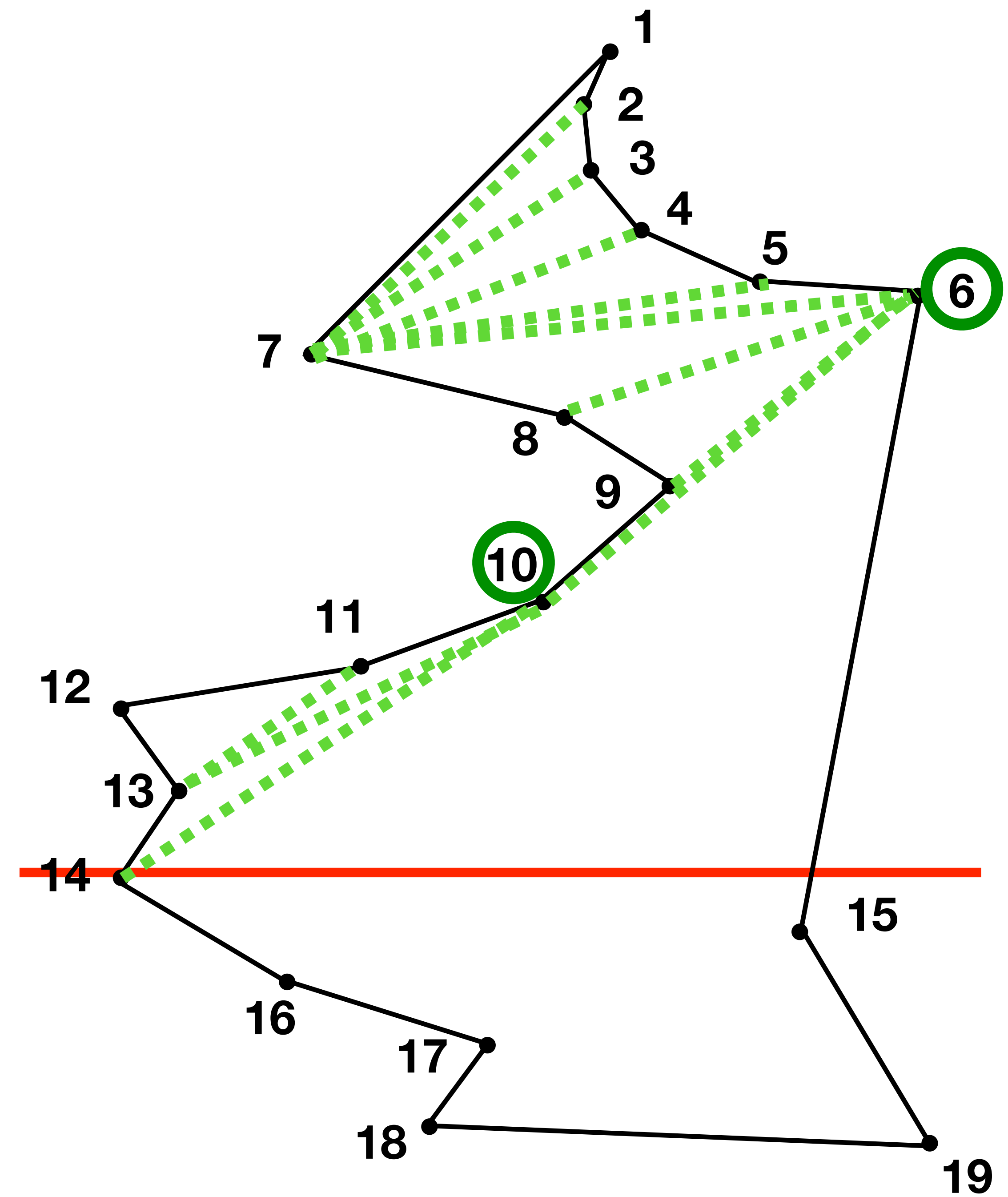
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10



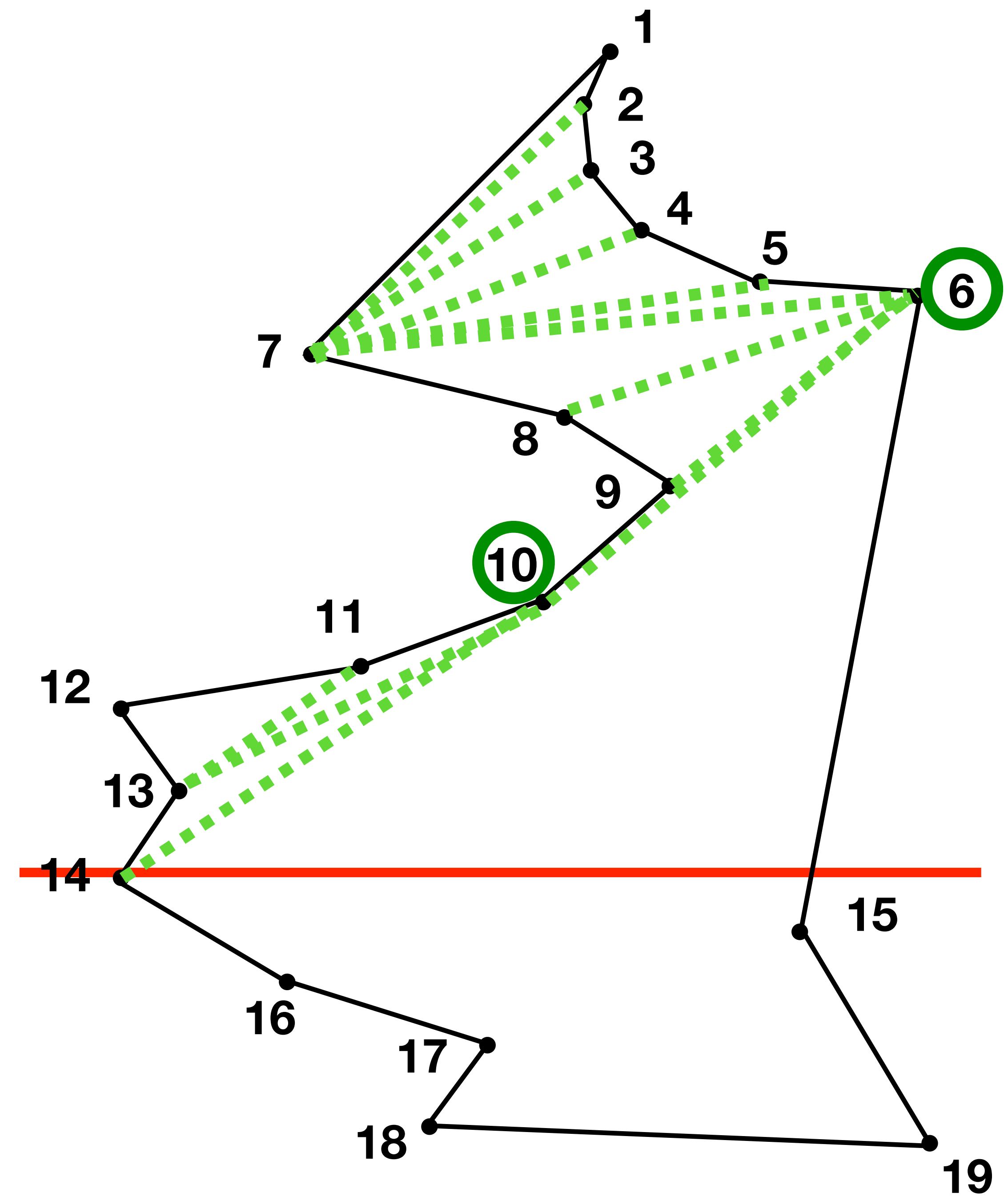
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10



Algorithm 5.16

Input: A y-monotone polygon P

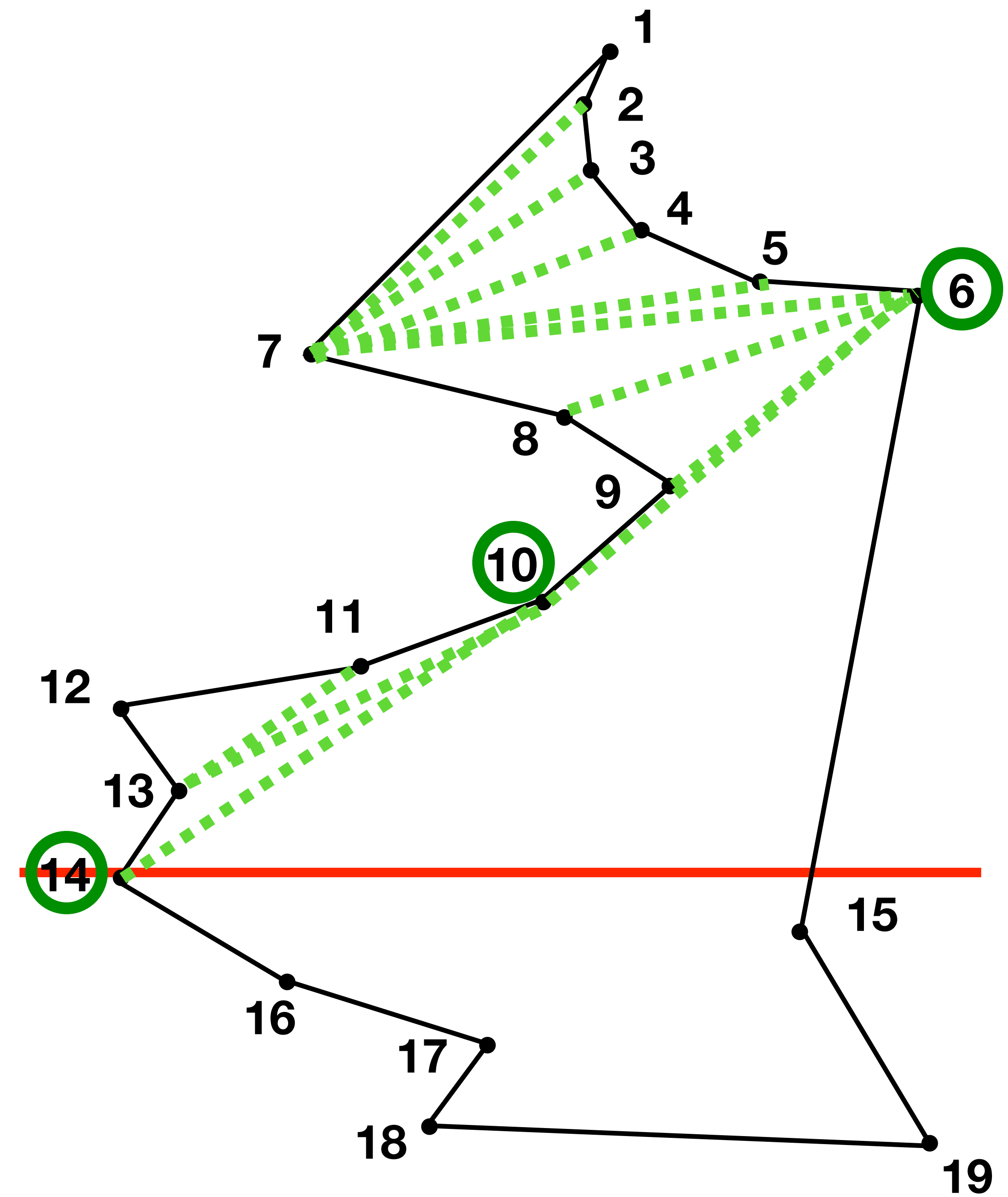
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10



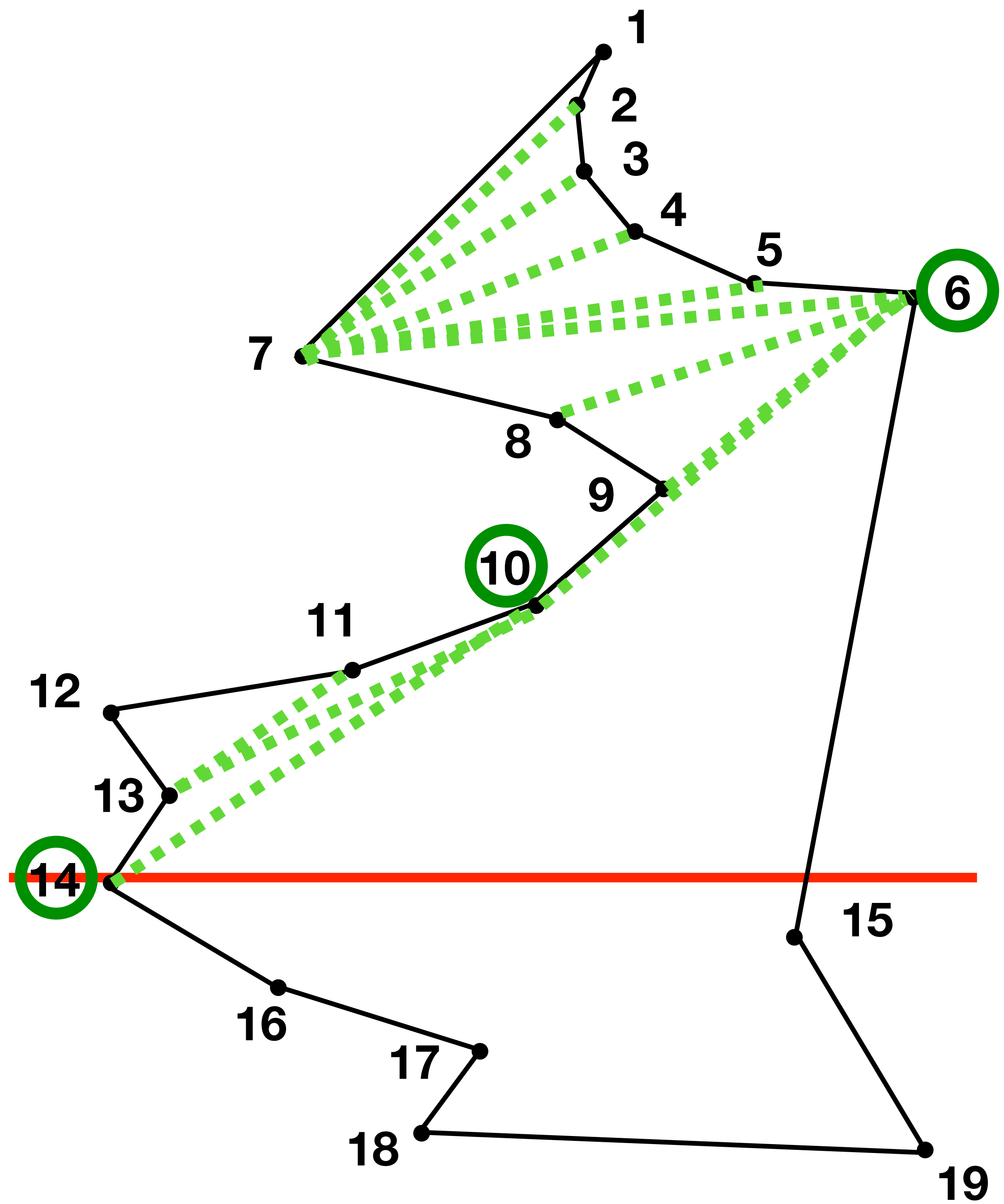
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 14



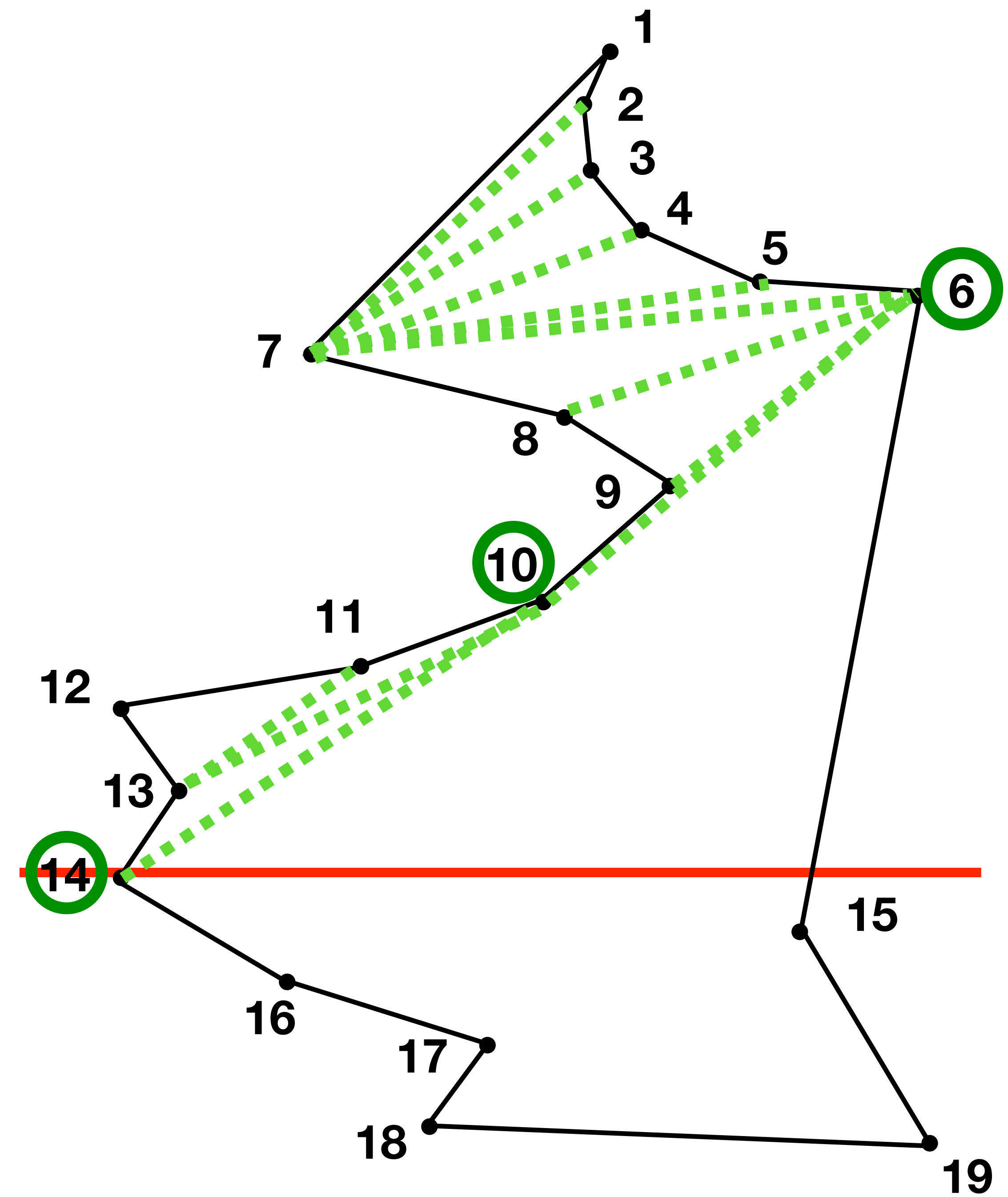
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 6 10 14



Algorithm 5.16

Input: A y-monotone polygon P

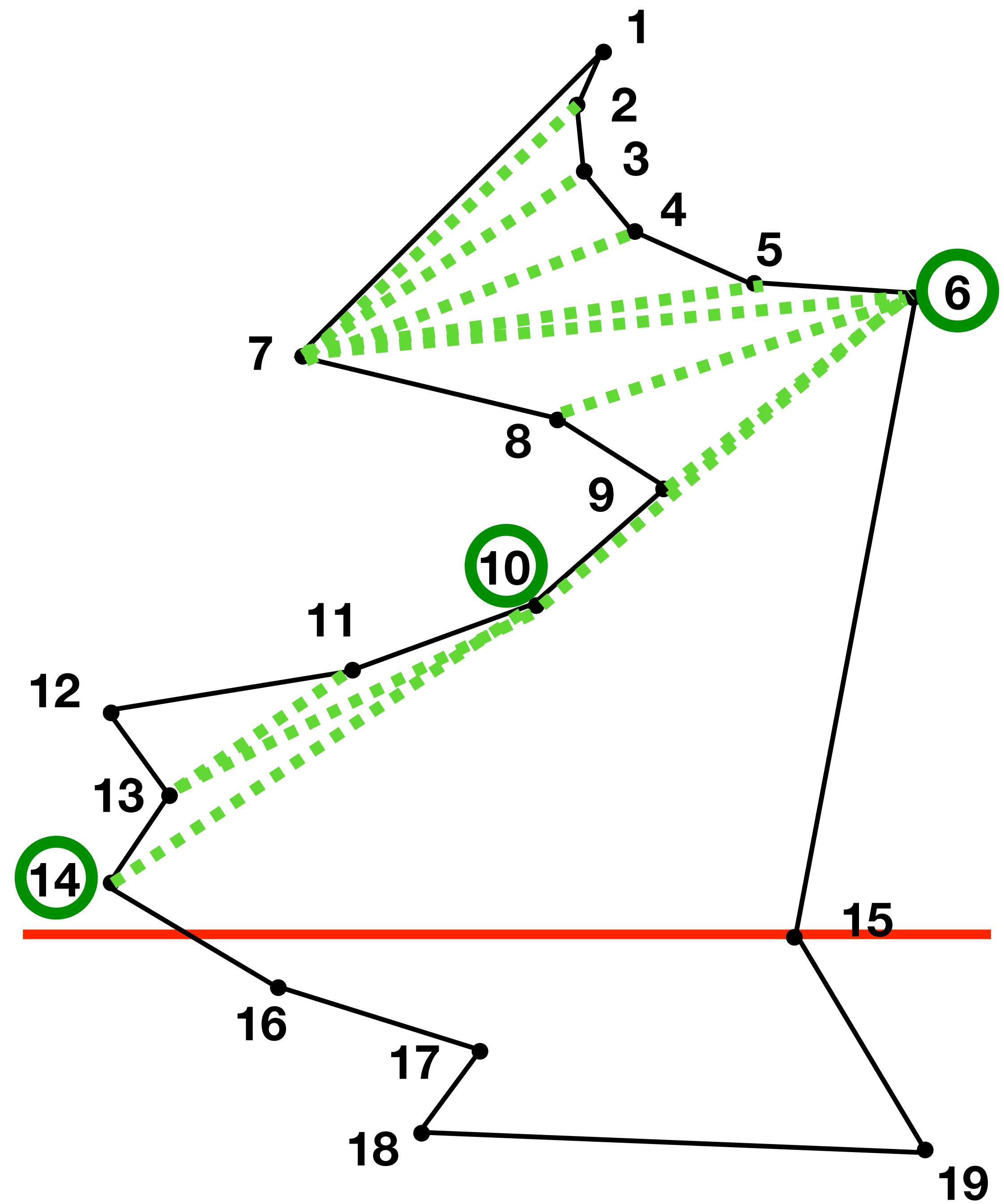
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 14



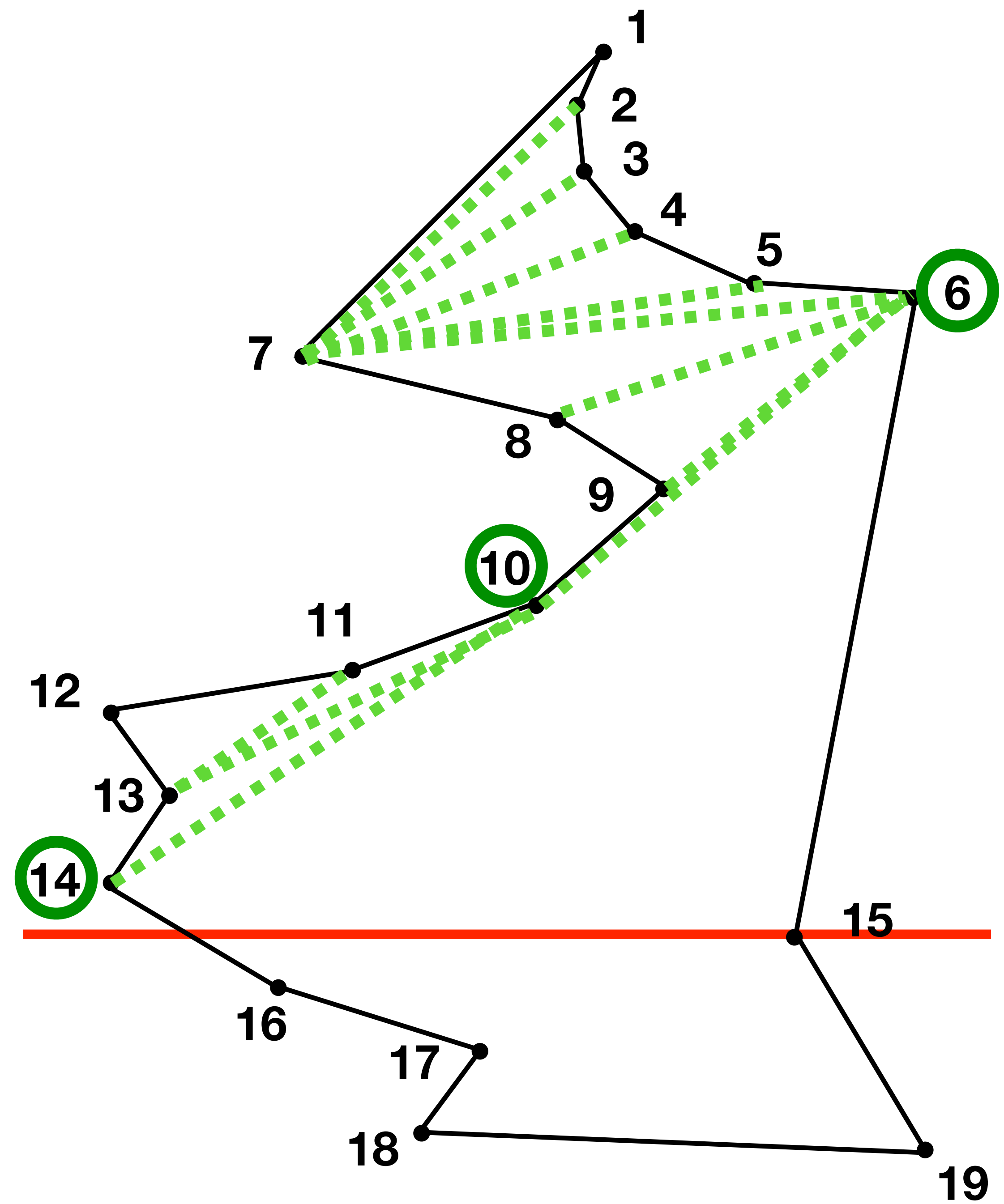
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10 14



Algorithm 5.16

Input: A y-monotone polygon P

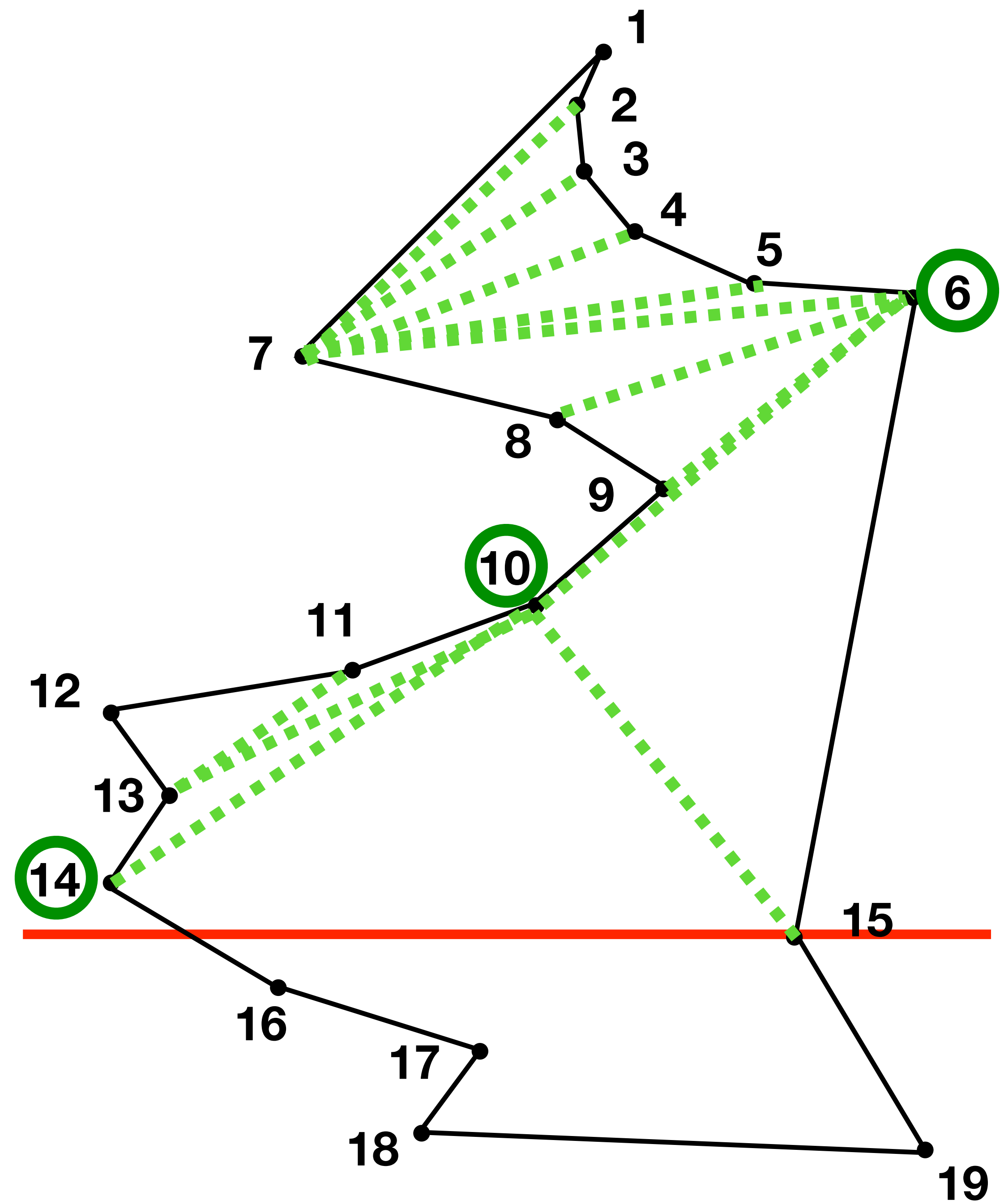
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 6 10 14



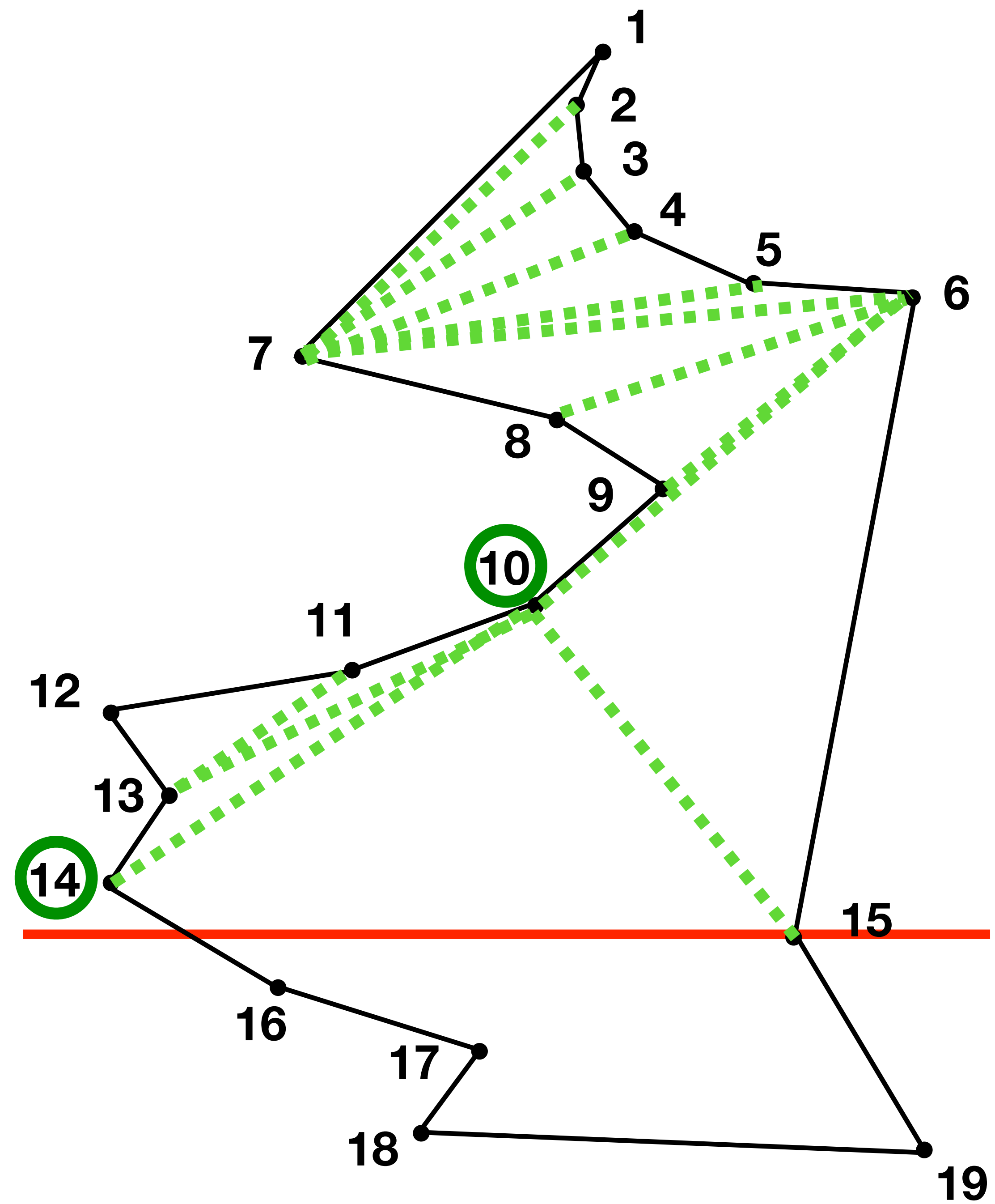
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 6 10 14



Technische Universität Braunschweig

Algorithm 5.16

Input: A y-monotone polygon P

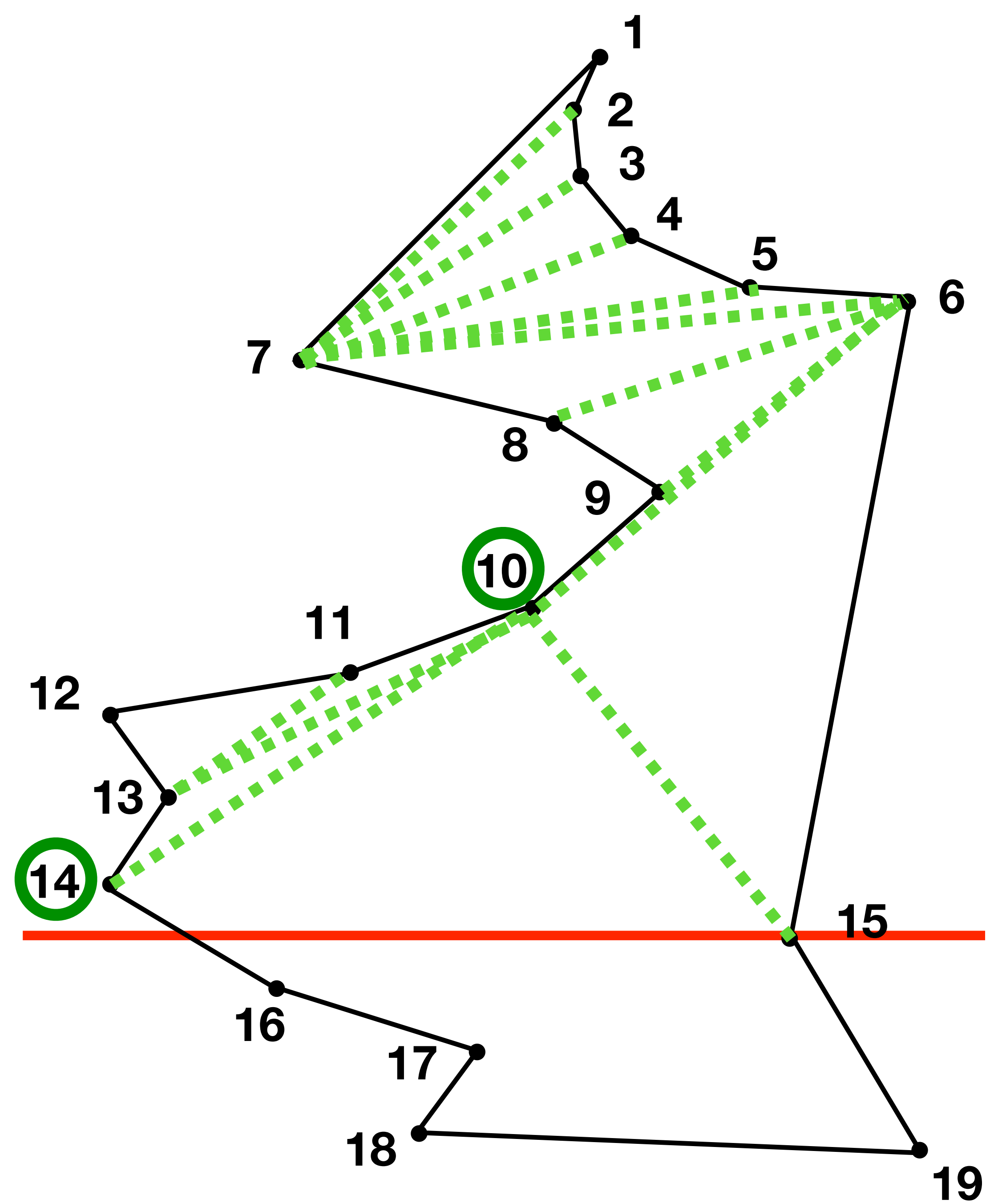
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 10 14



Algorithm 5.16

Input: A y-monotone polygon P

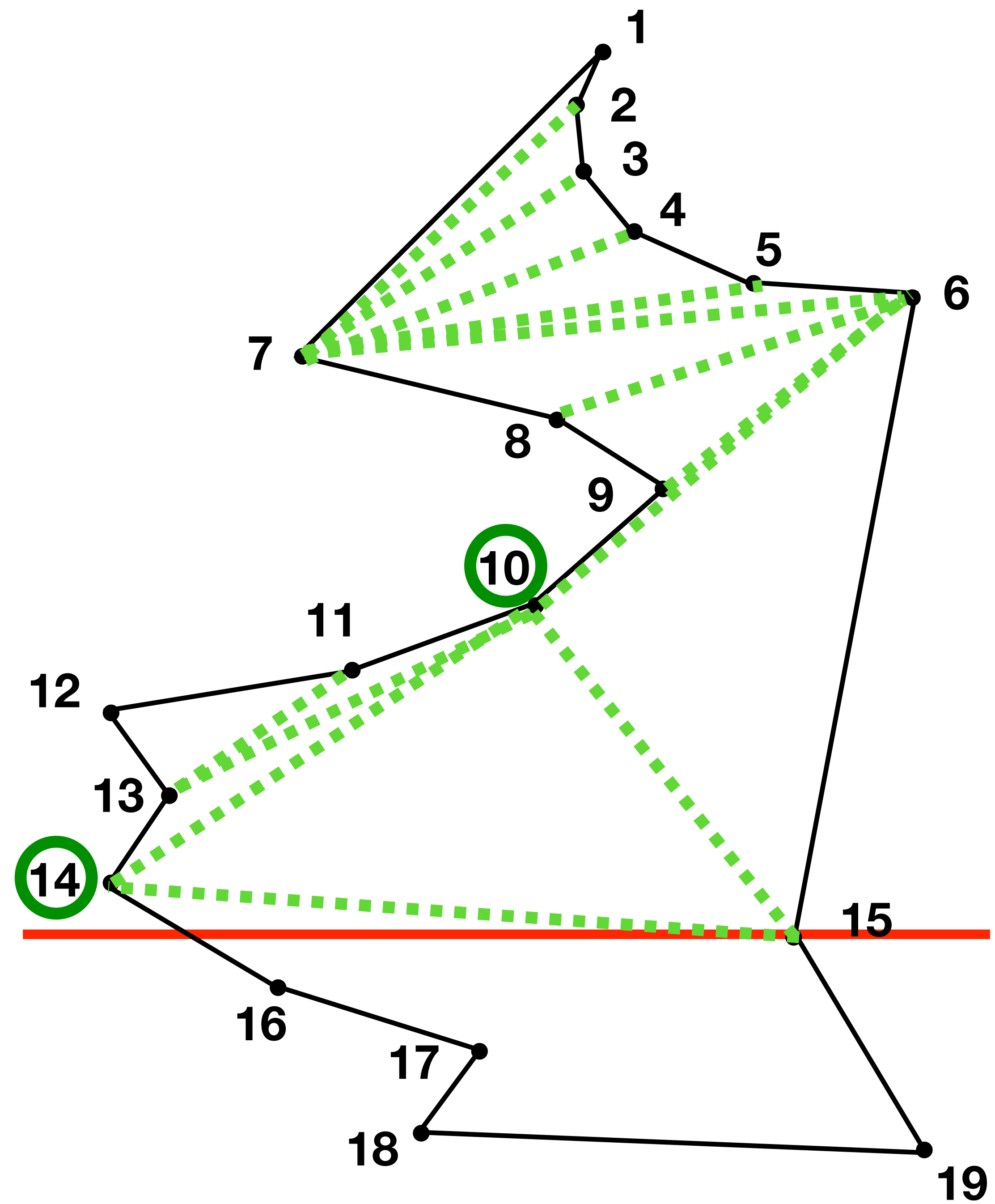
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 10 14



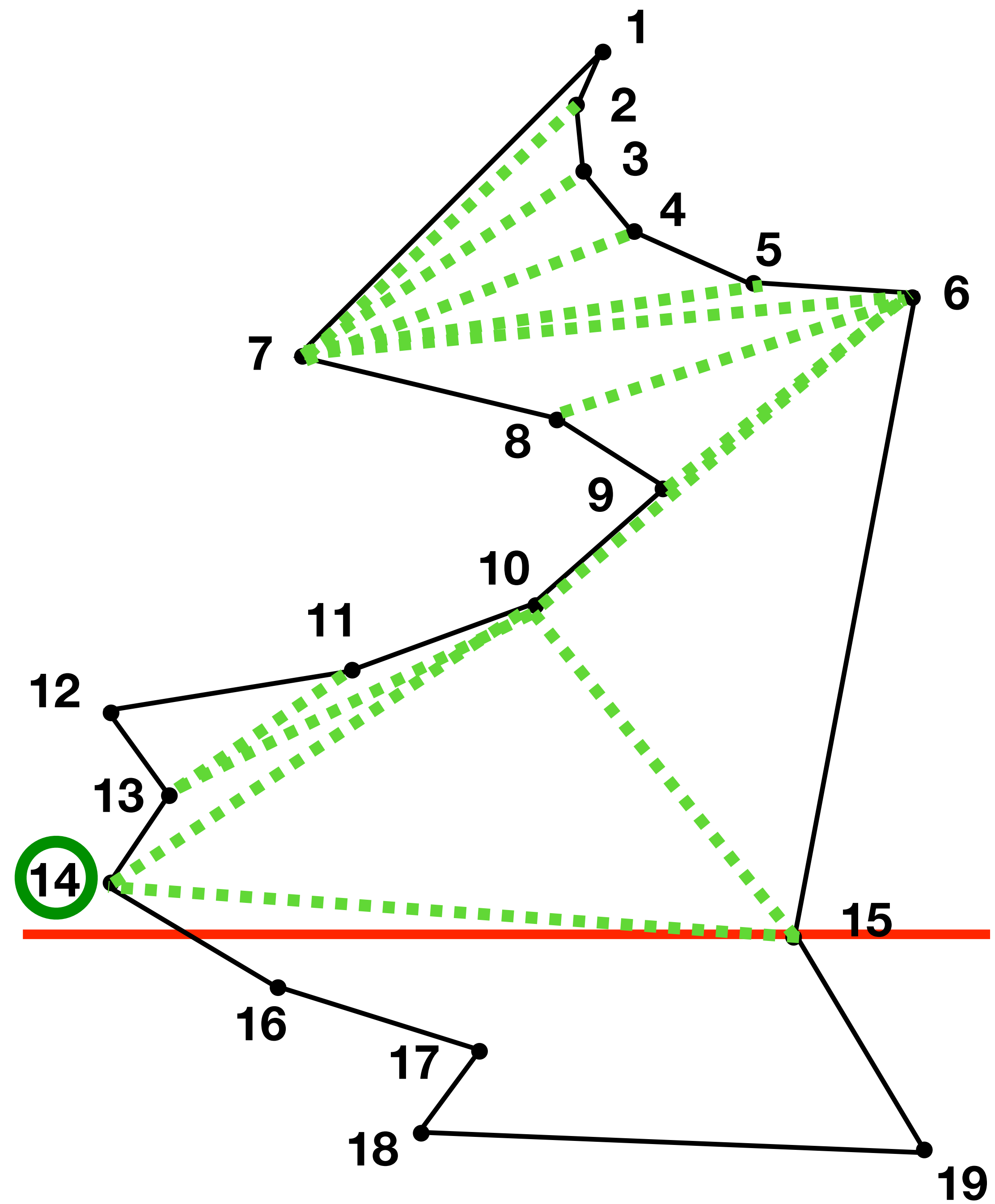
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 10 14



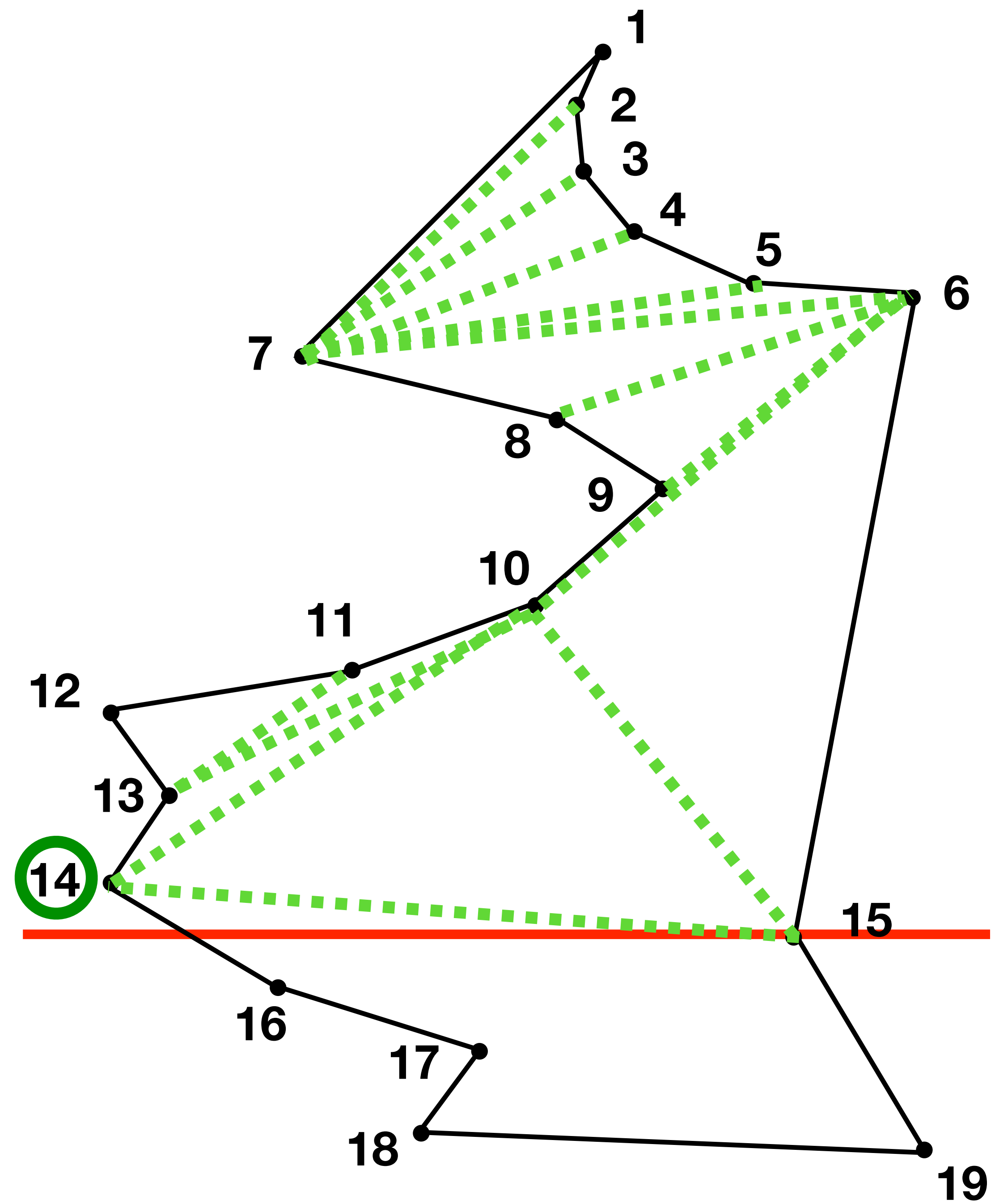
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 14



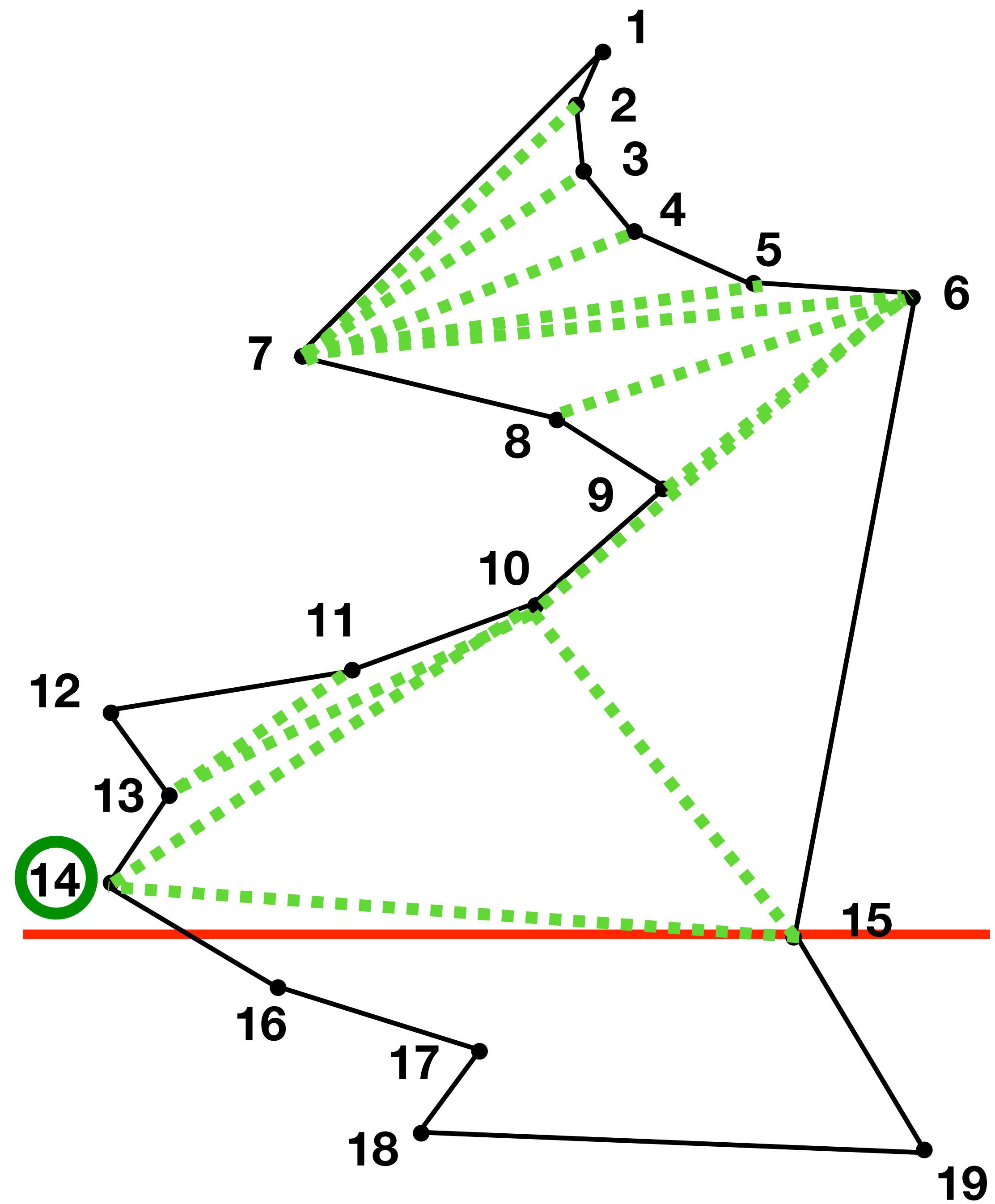
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 14



Algorithm 5.16

Input: A y-monotone polygon P

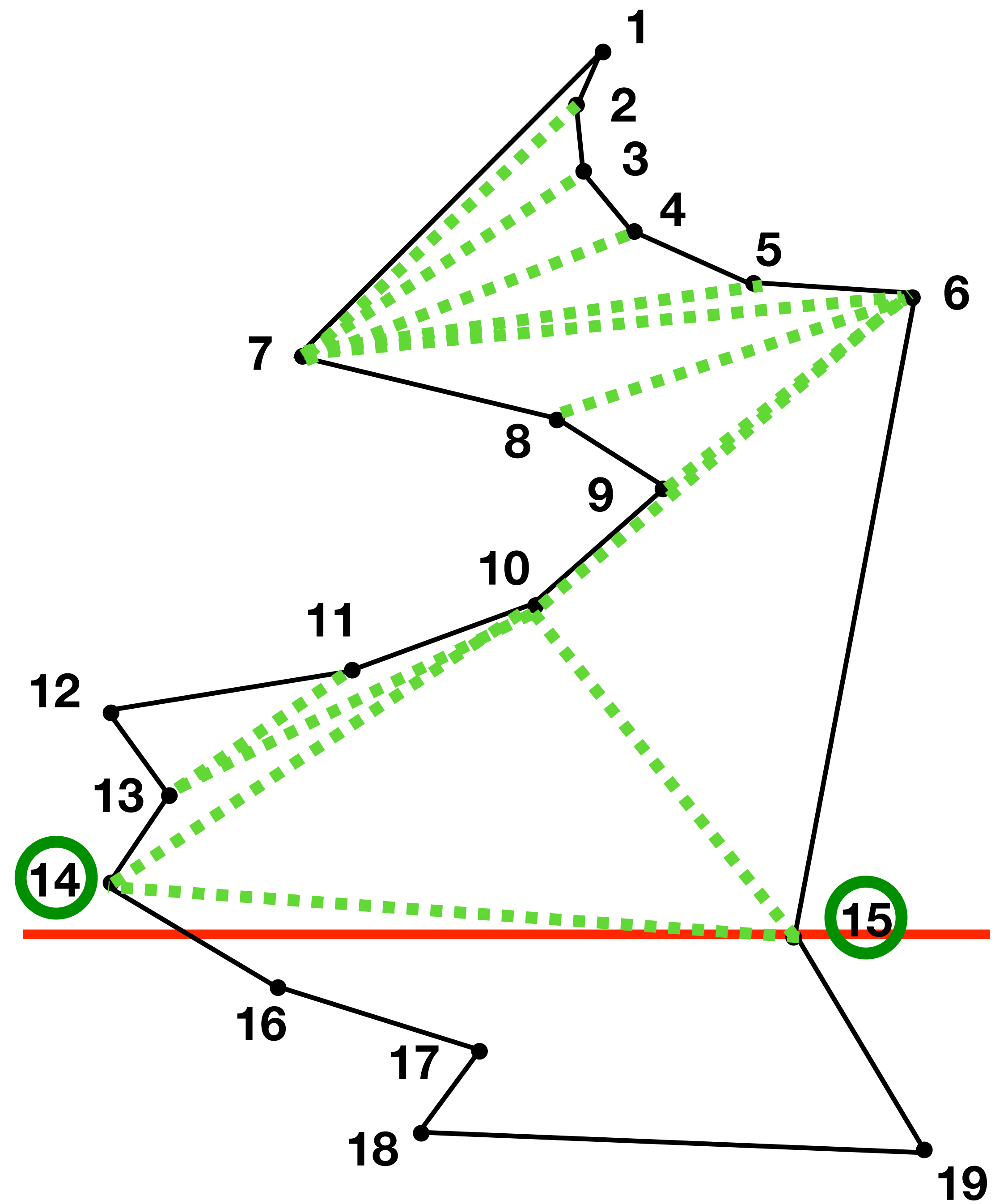
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue — **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue — **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update — **Ear**

Queue: 14



Algorithm 5.16

Input: A y-monotone polygon P

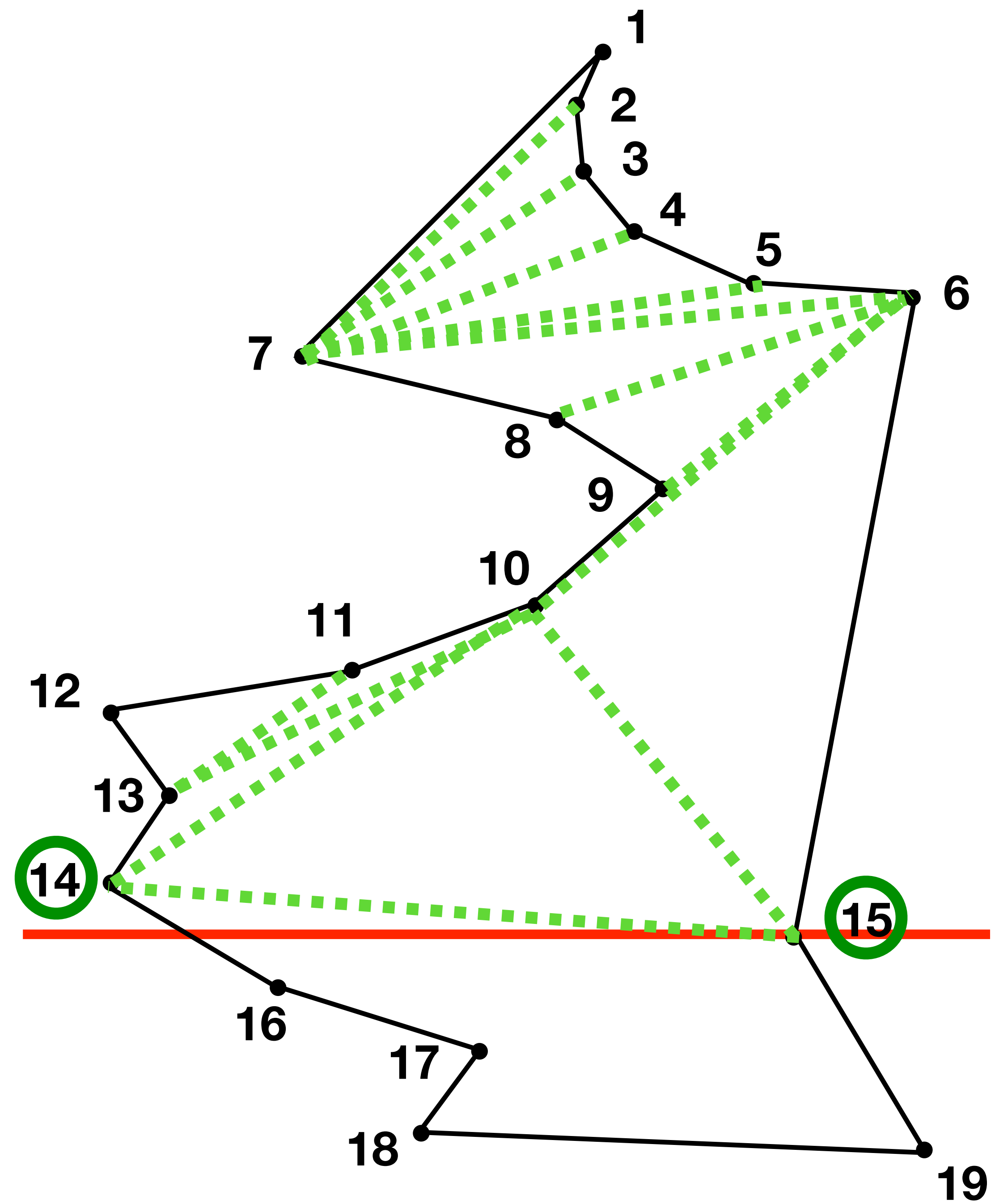
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 14 15



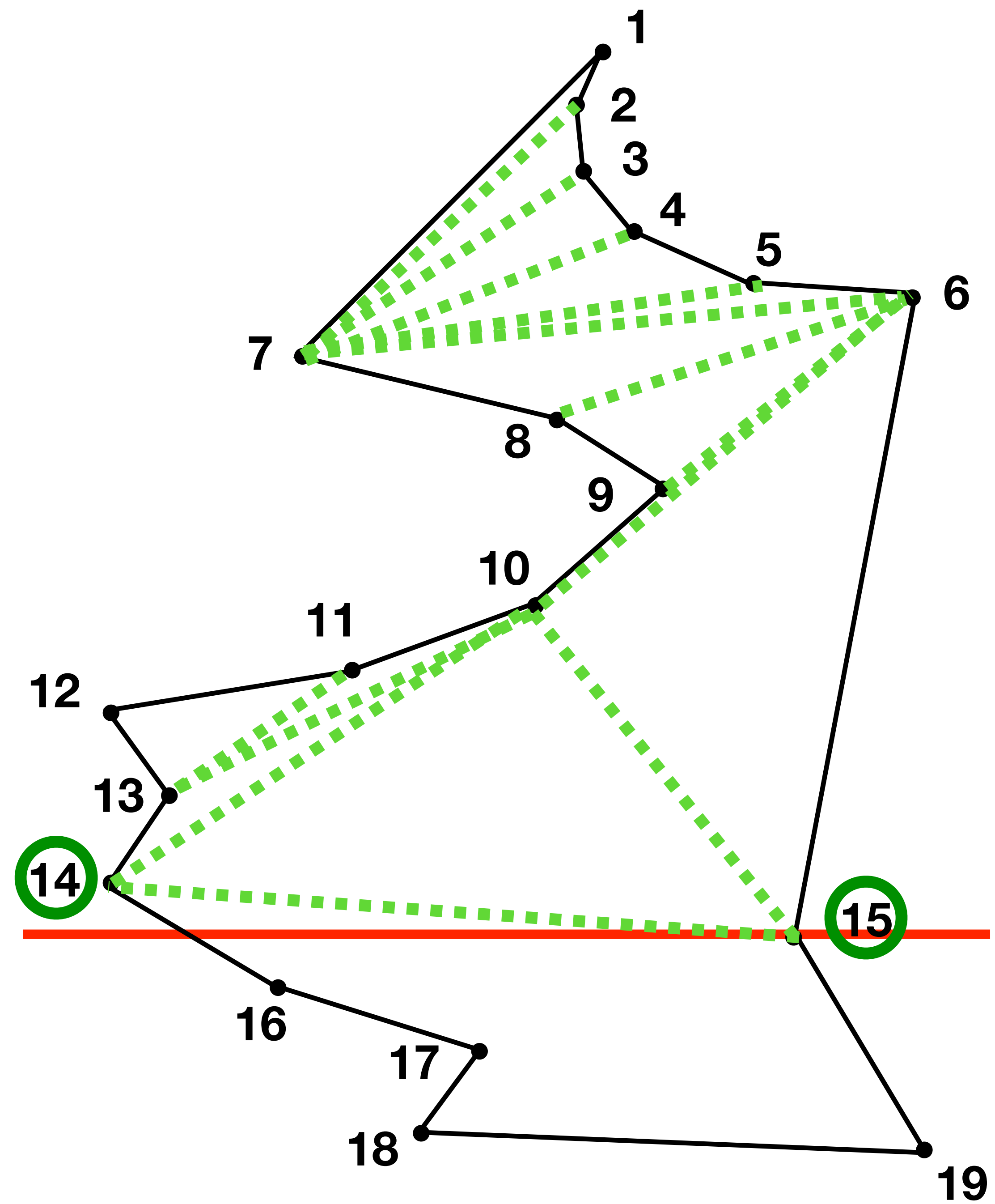
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 14 15



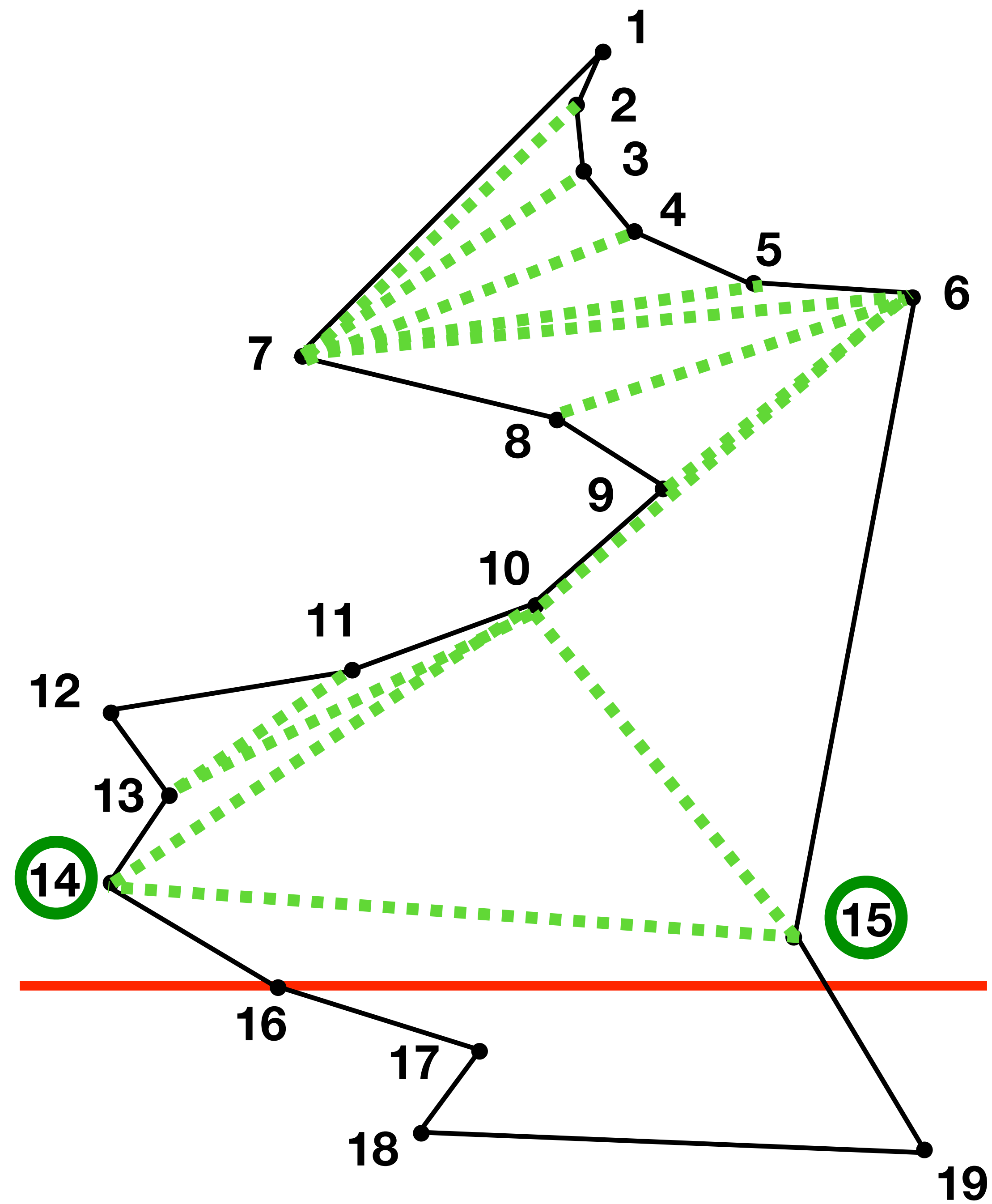
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 14 15



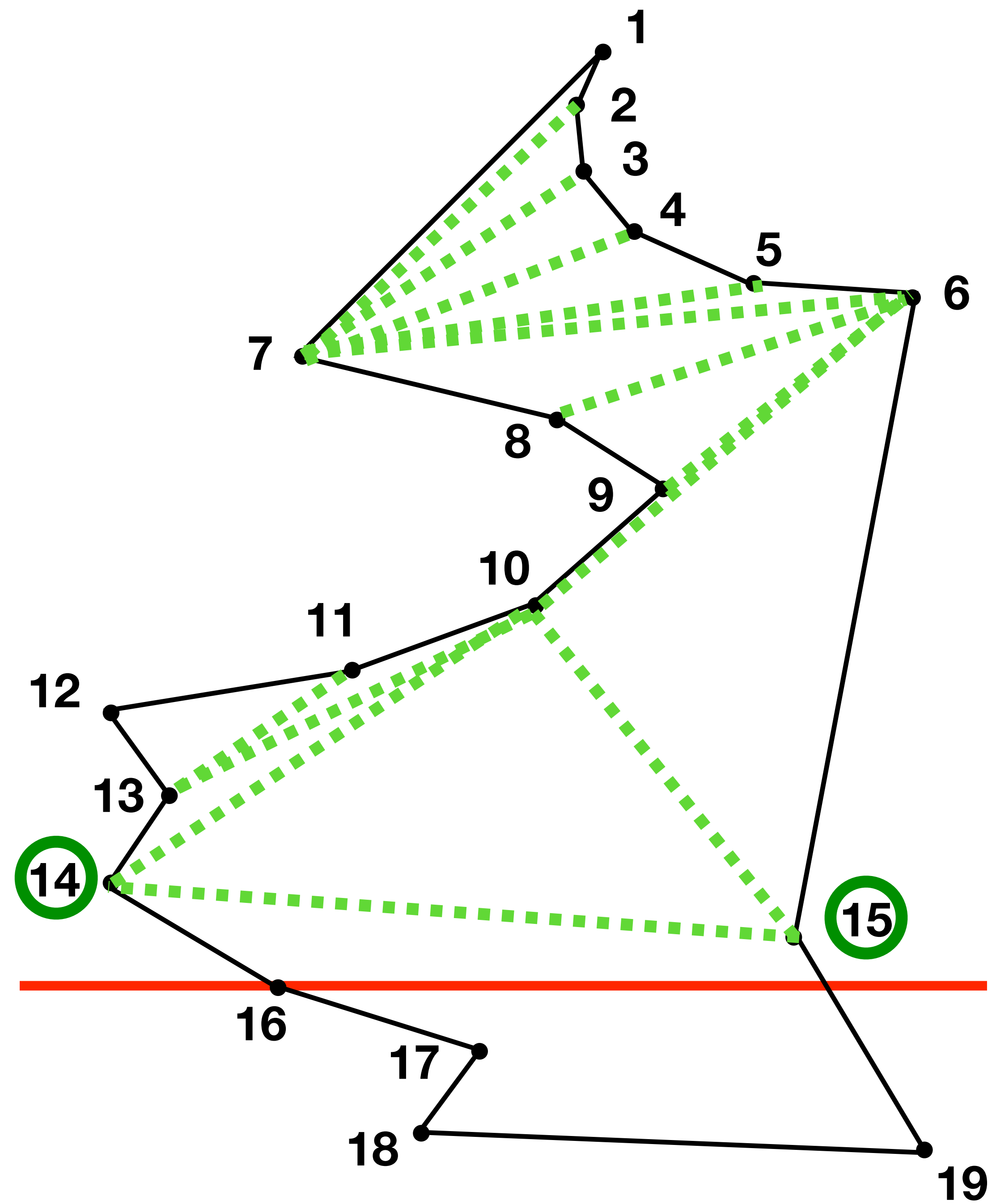
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 14 15



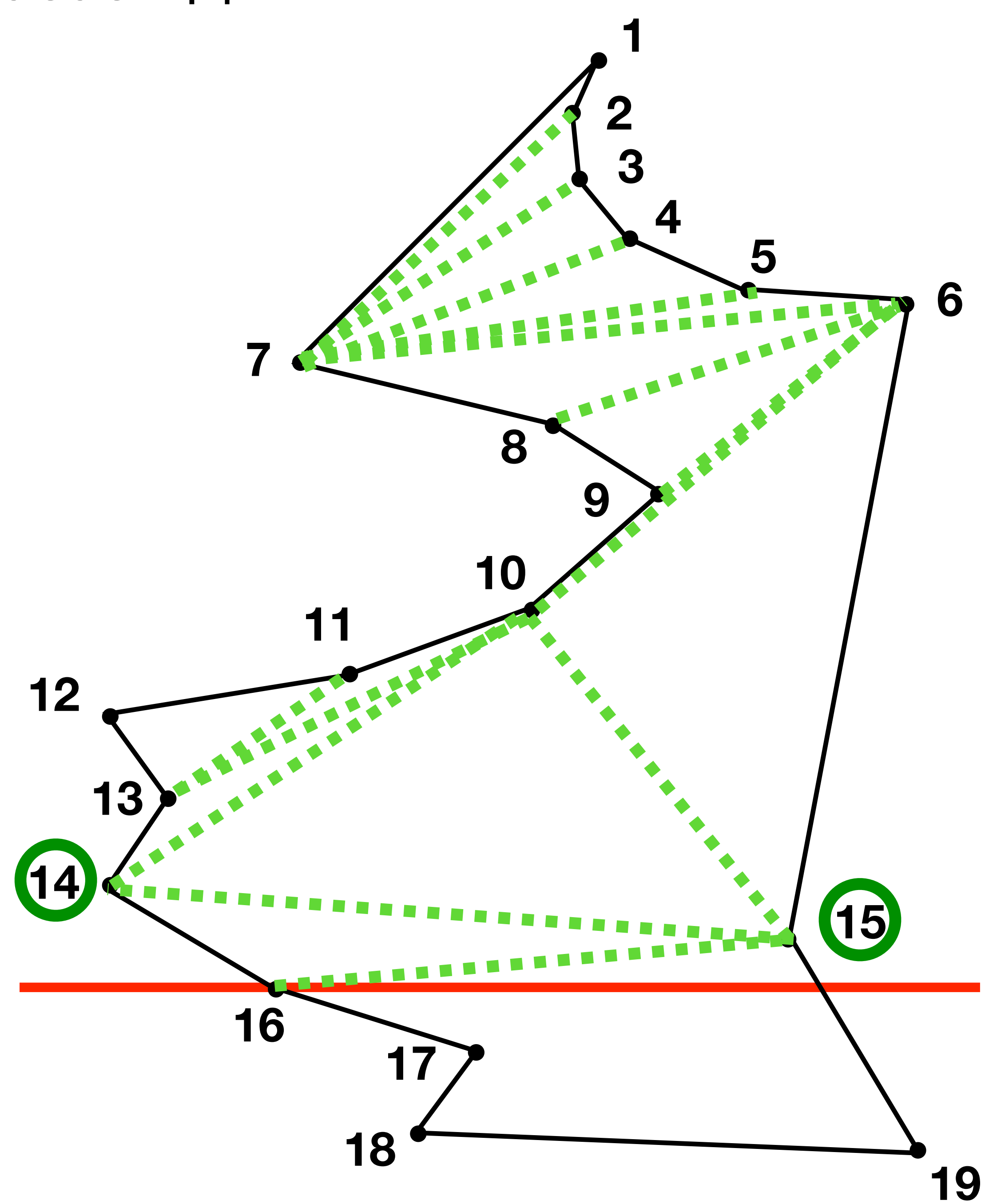
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 14 15



Algorithm 5.16

Input: A y-monotone polygon P

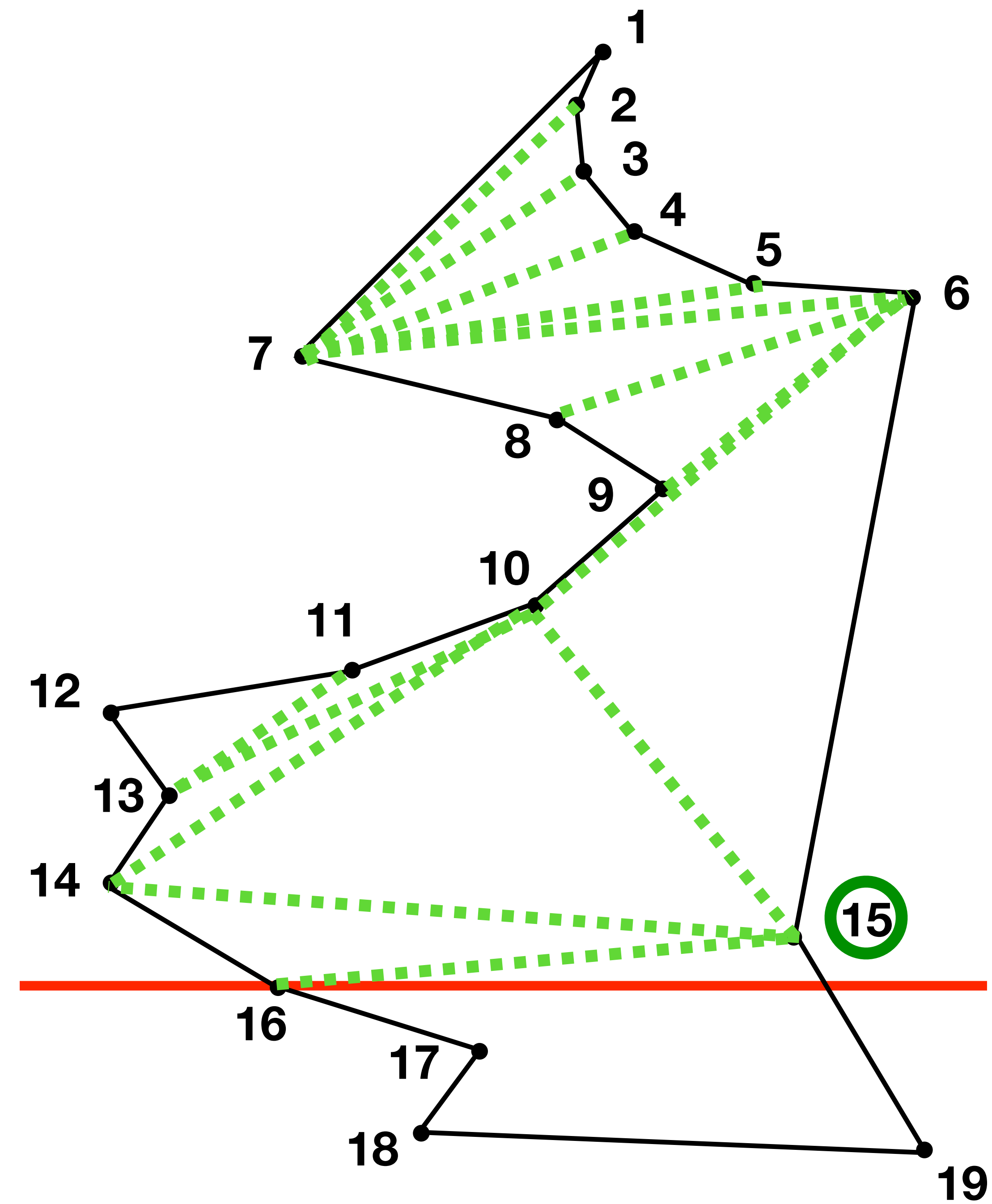
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 14 15



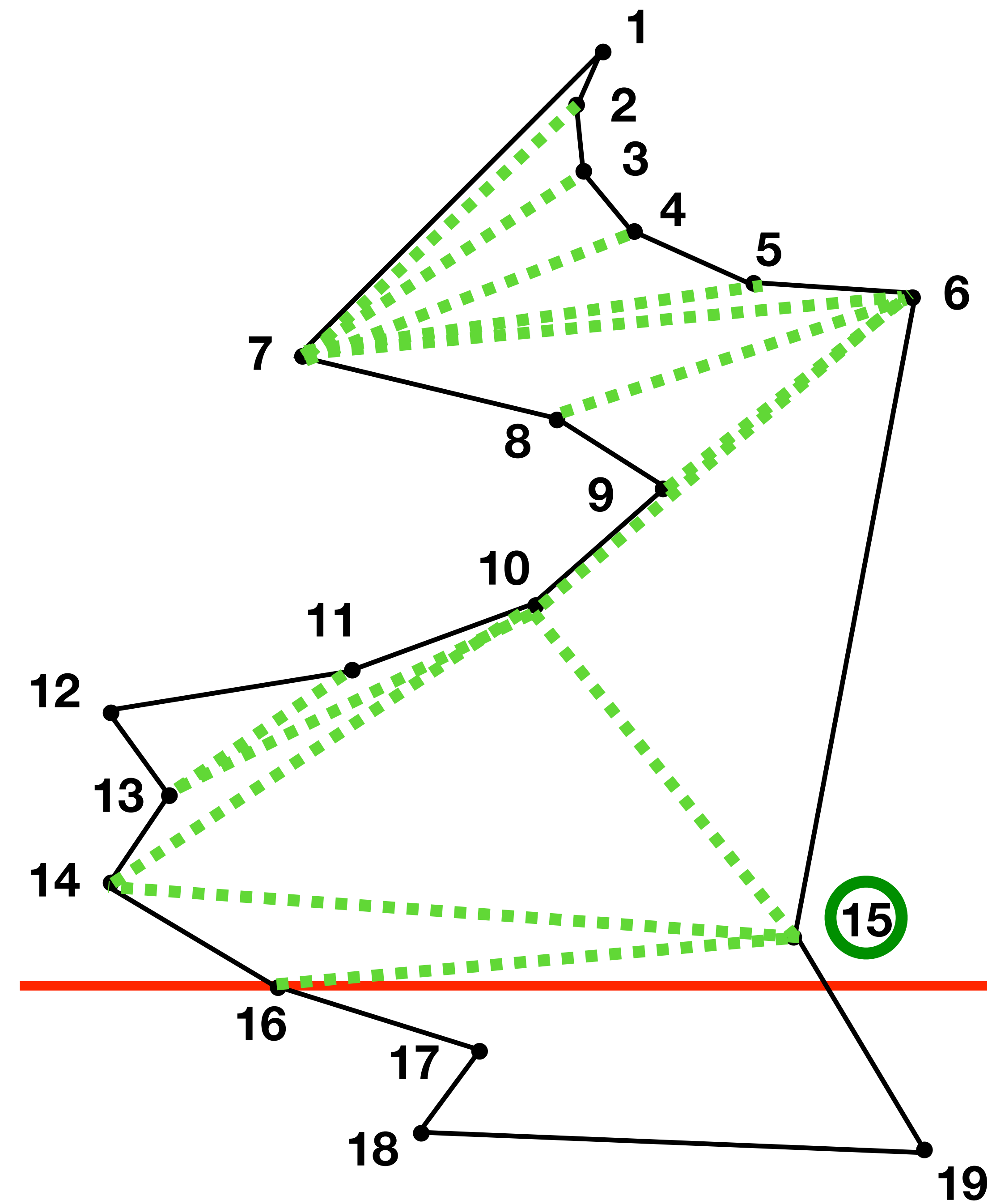
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15



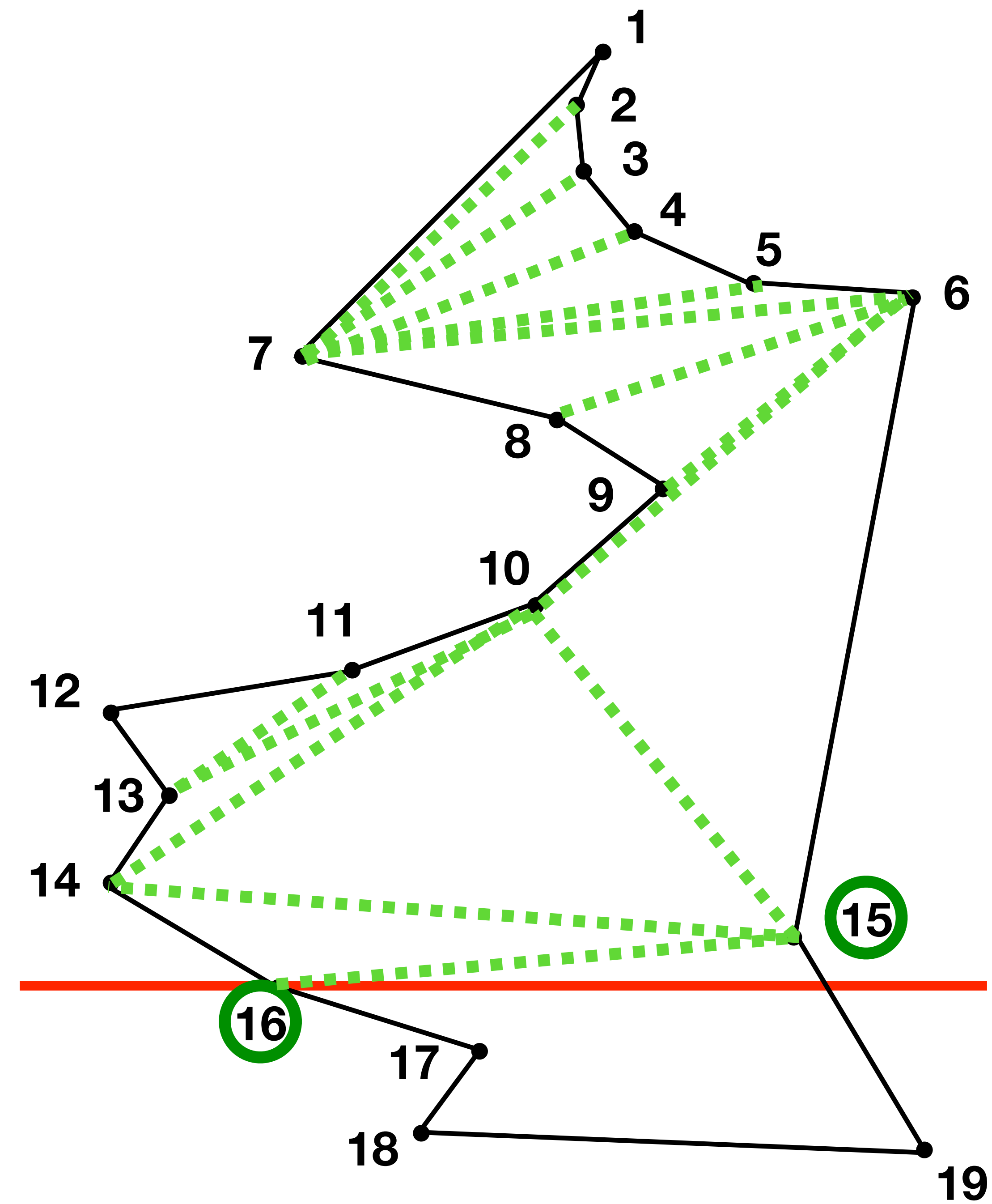
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15



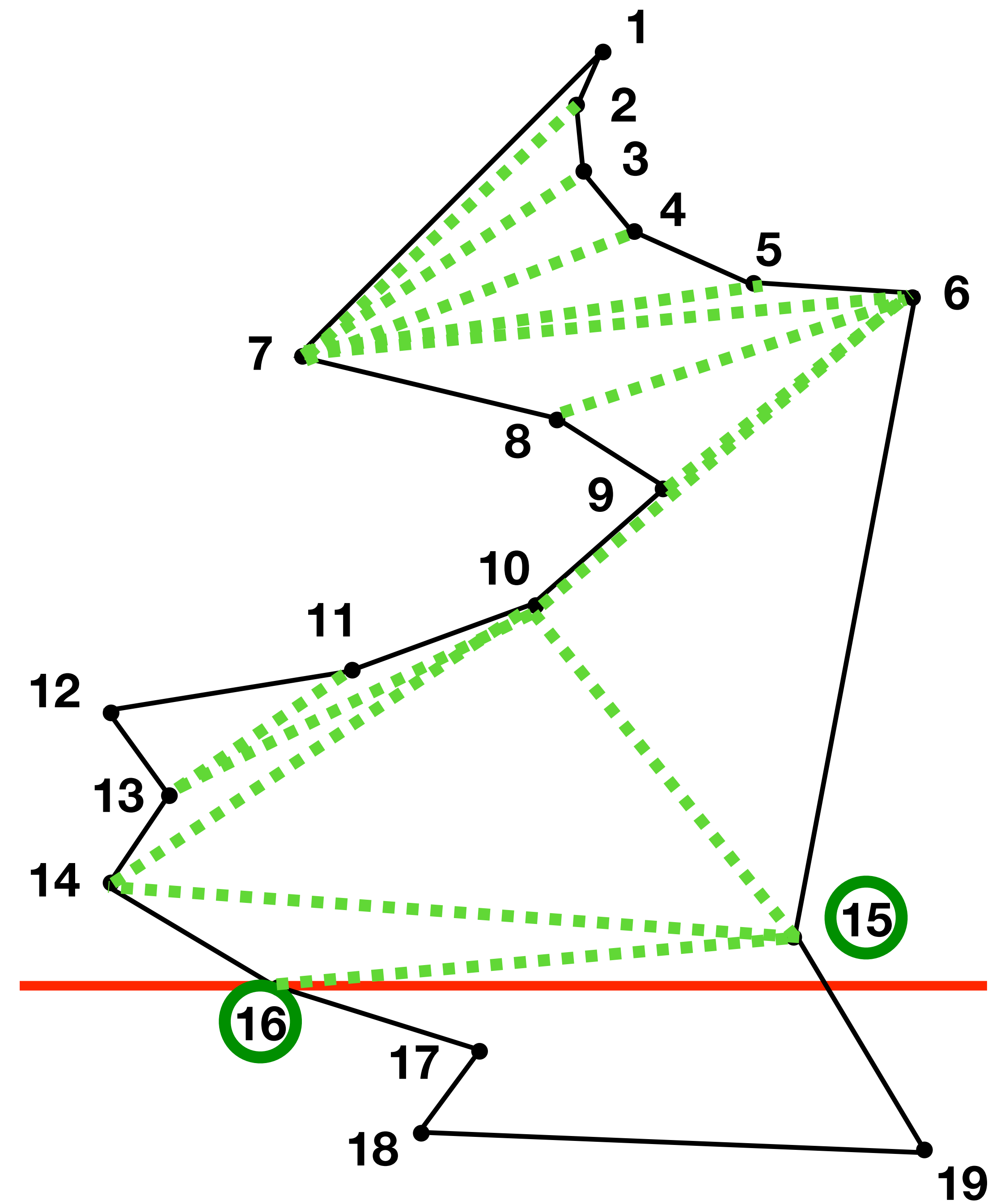
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 16



Technische Universität Braunschweig

Algorithm 5.16

Input: A y-monotone polygon P

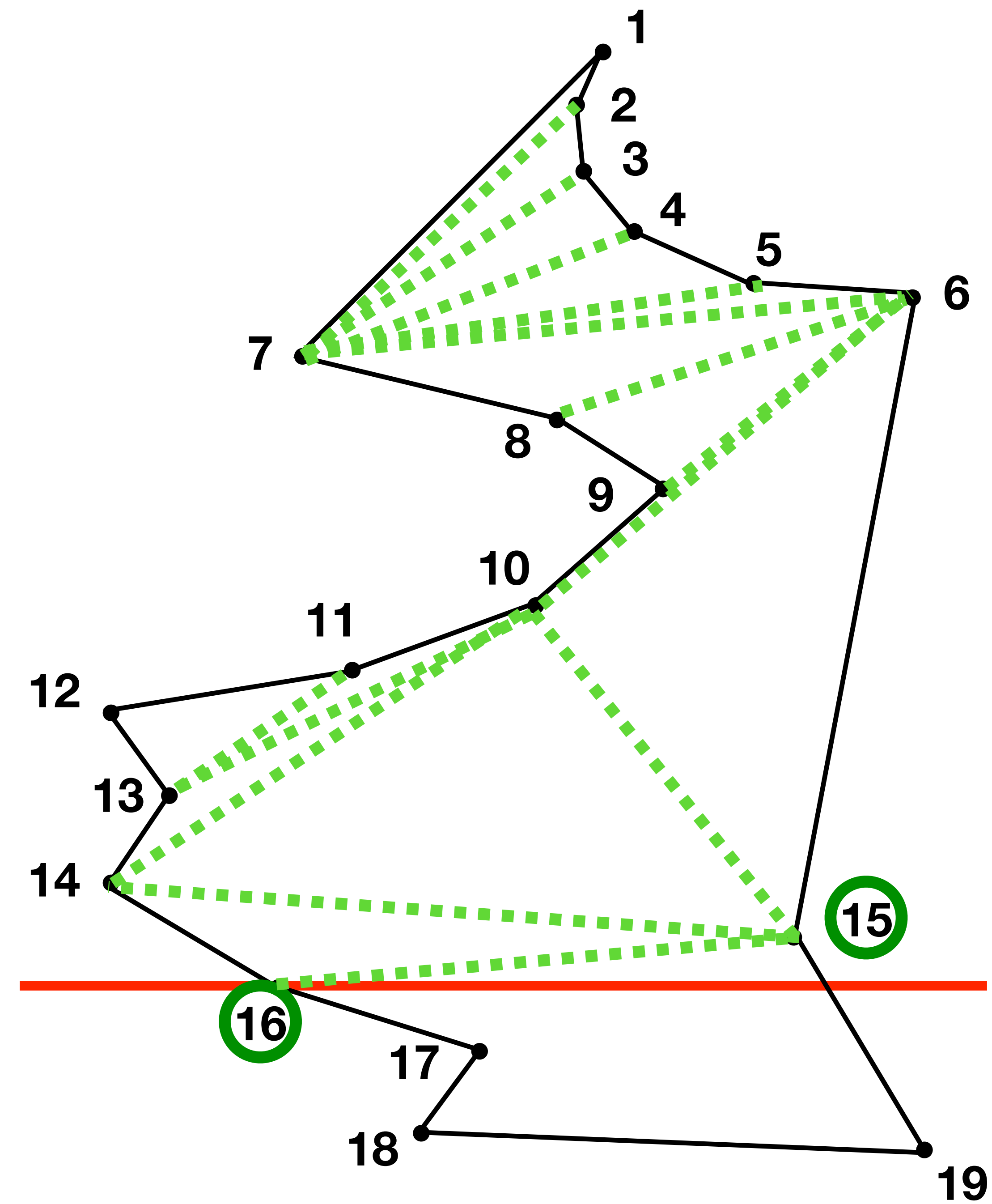
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 15 16



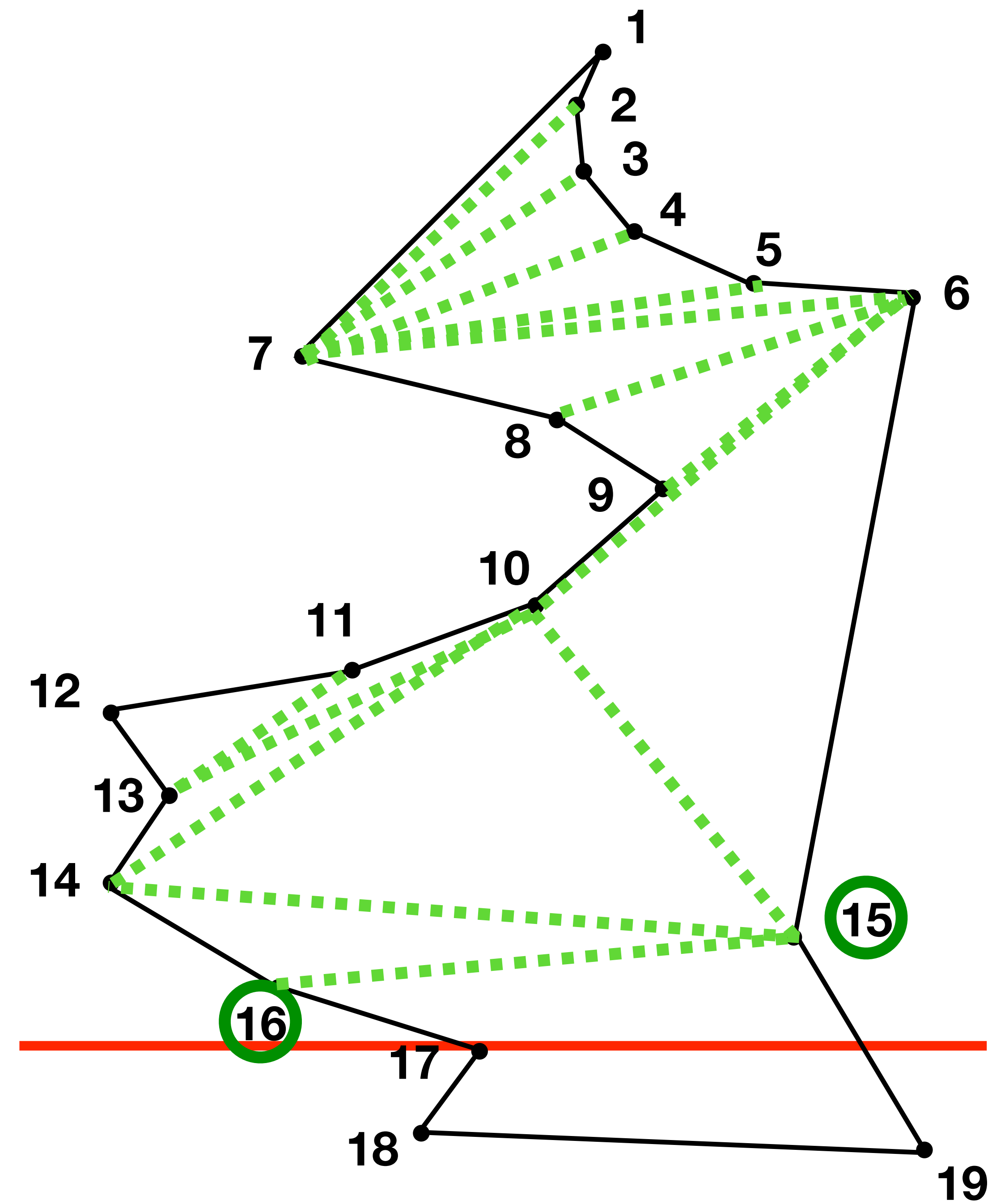
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 16



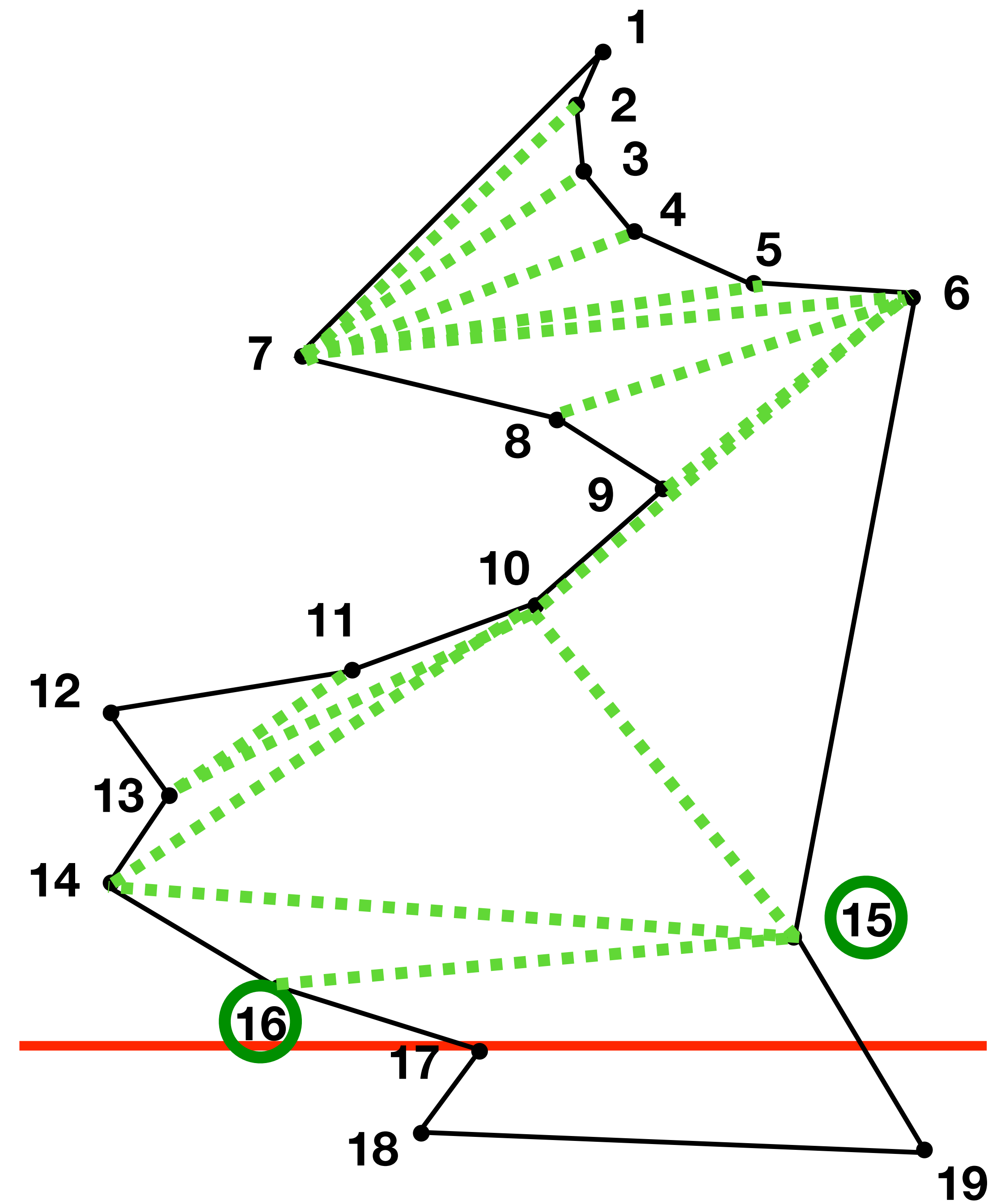
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15 16



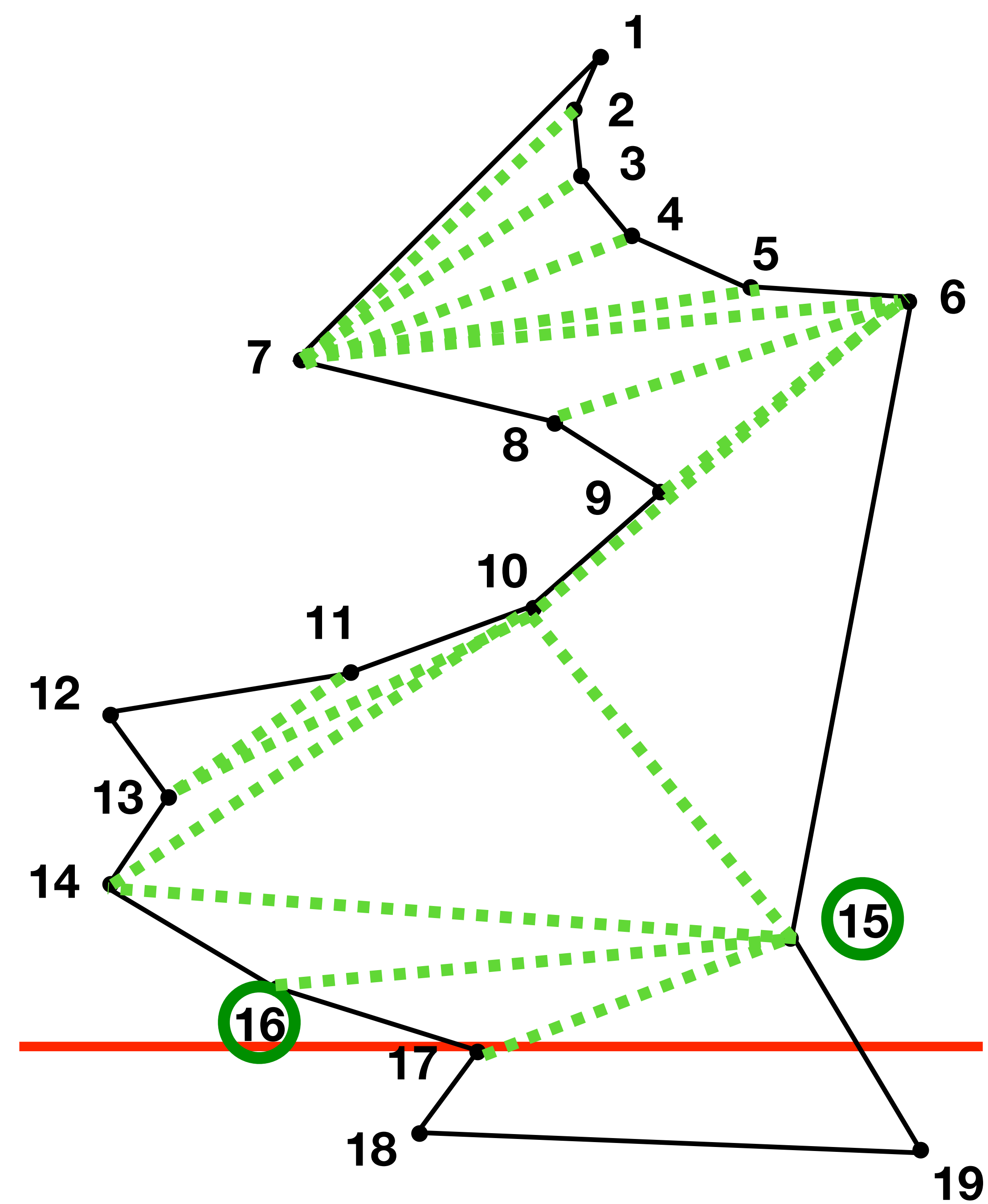
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 16



Algorithm 5.16

Input: A y-monotone polygon P

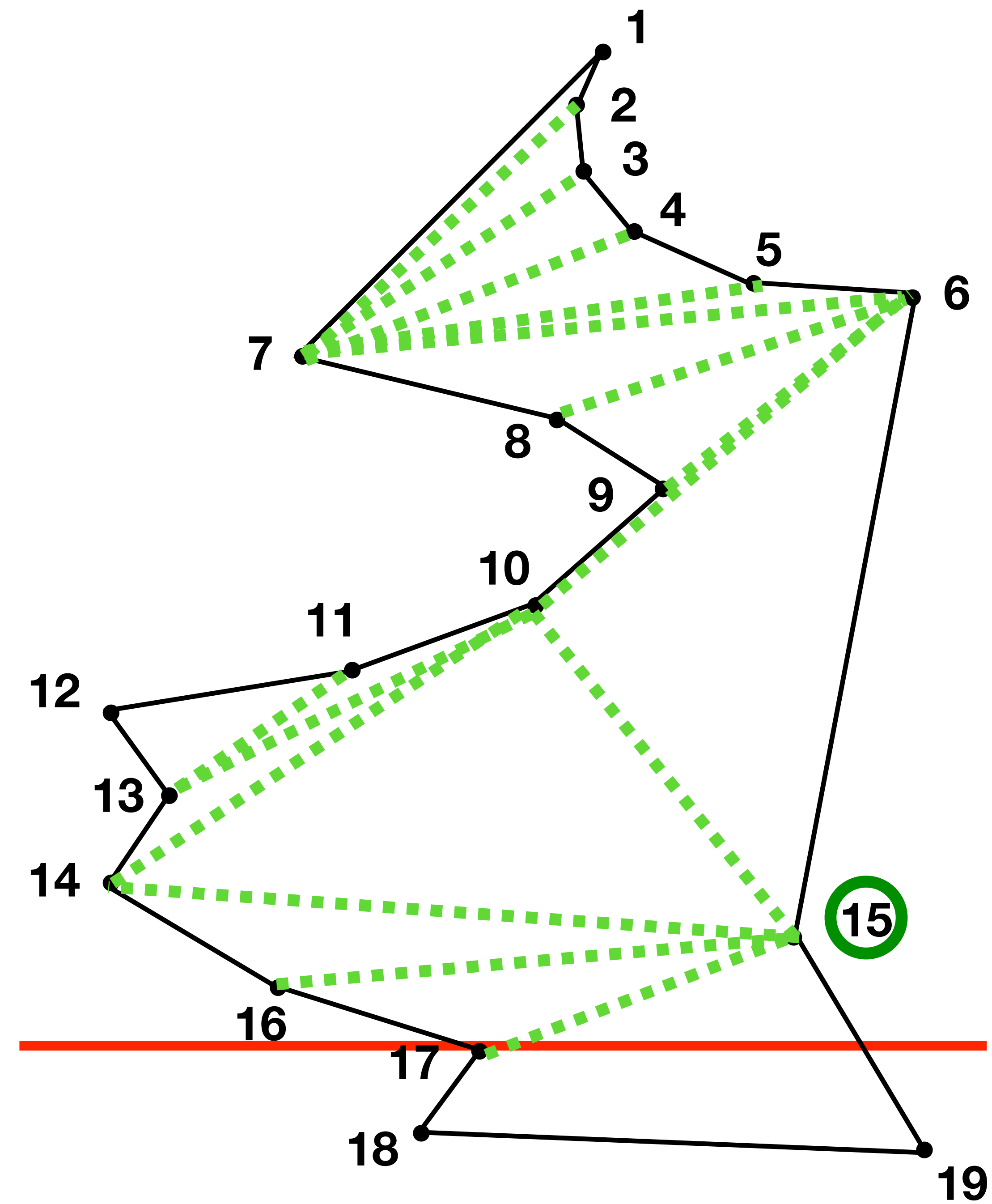
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 15 16



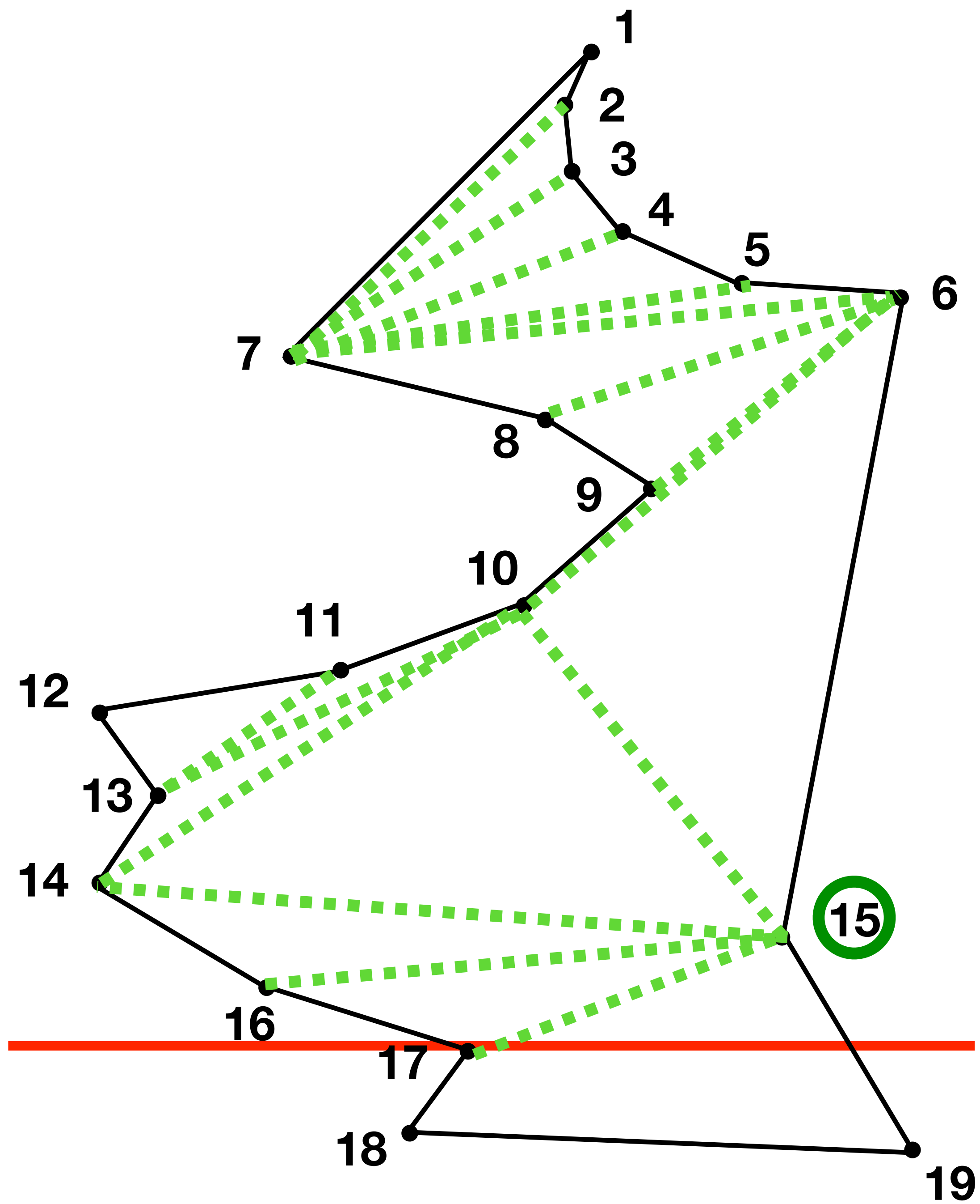
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15



Algorithm 5.16

Input: A y-monotone polygon P

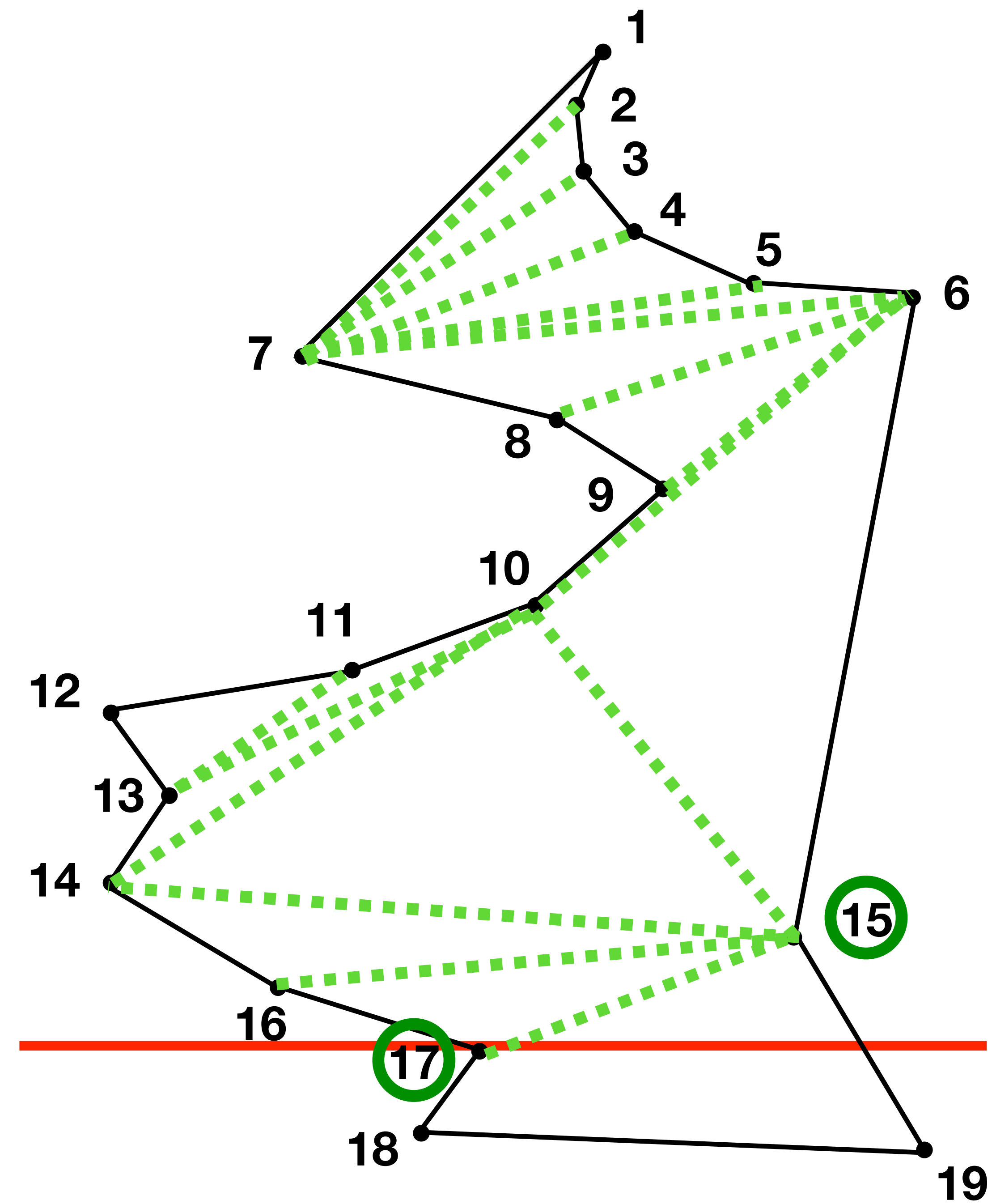
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 15



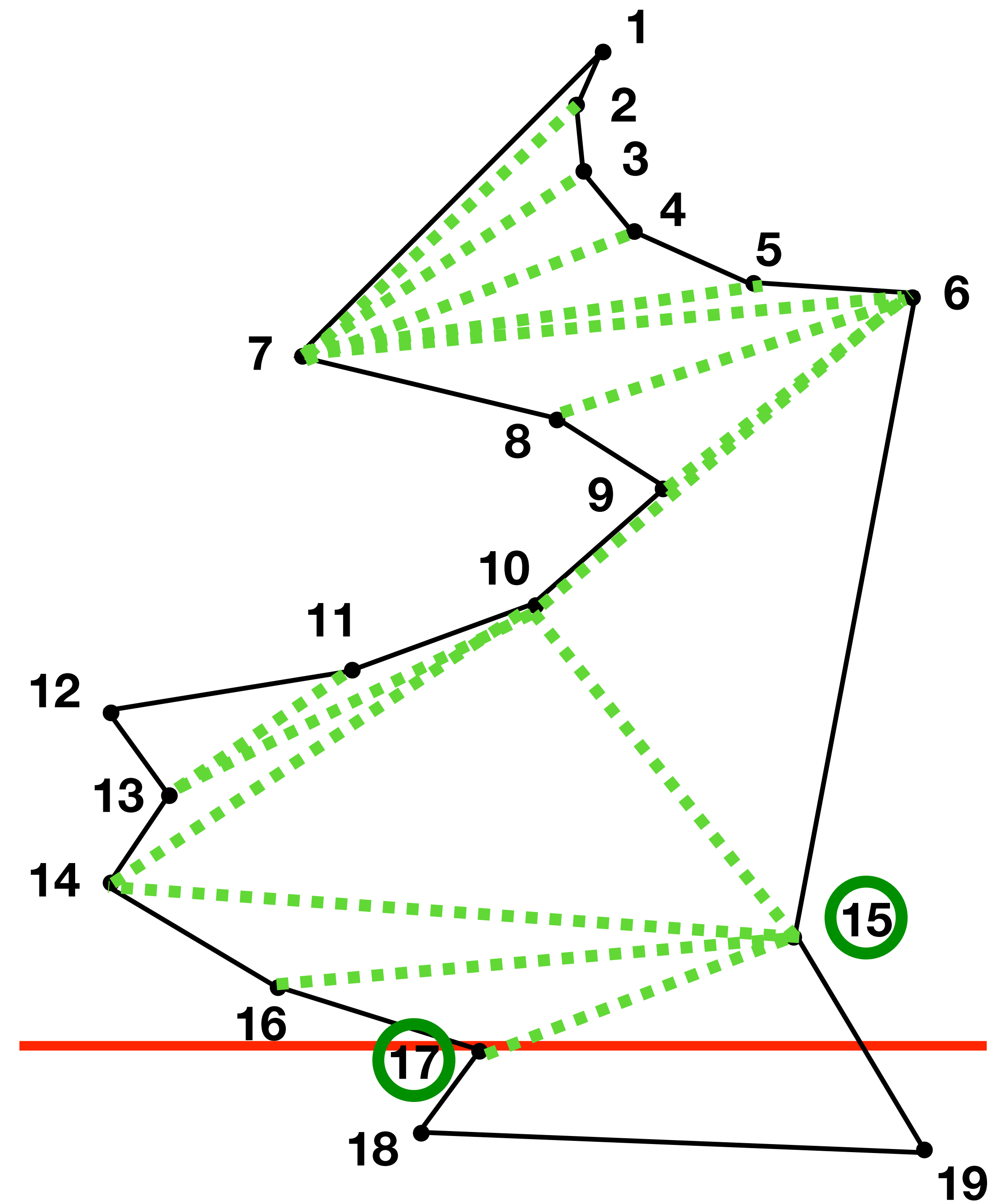
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15 17



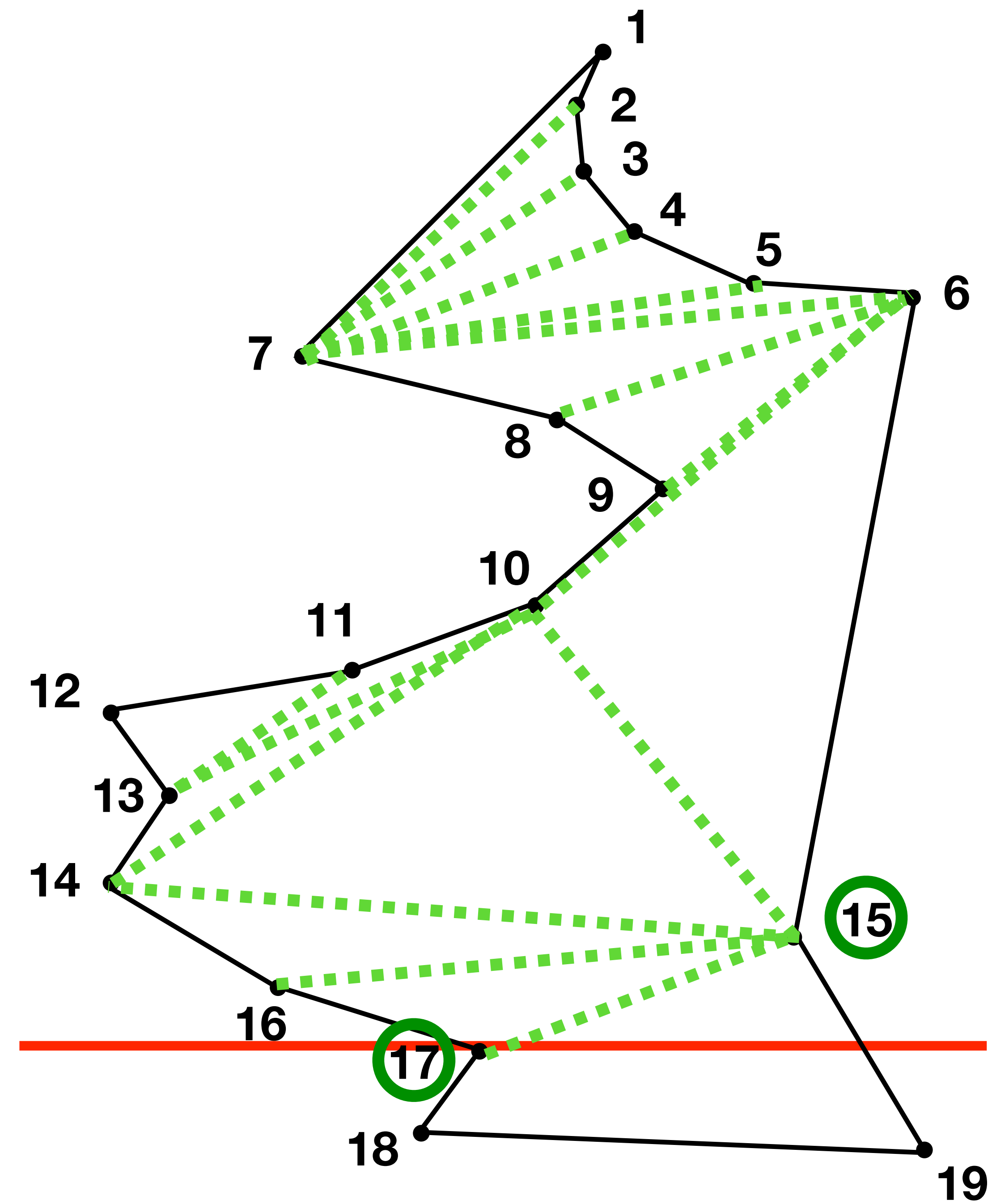
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 17



Algorithm 5.16

Input: A y-monotone polygon P

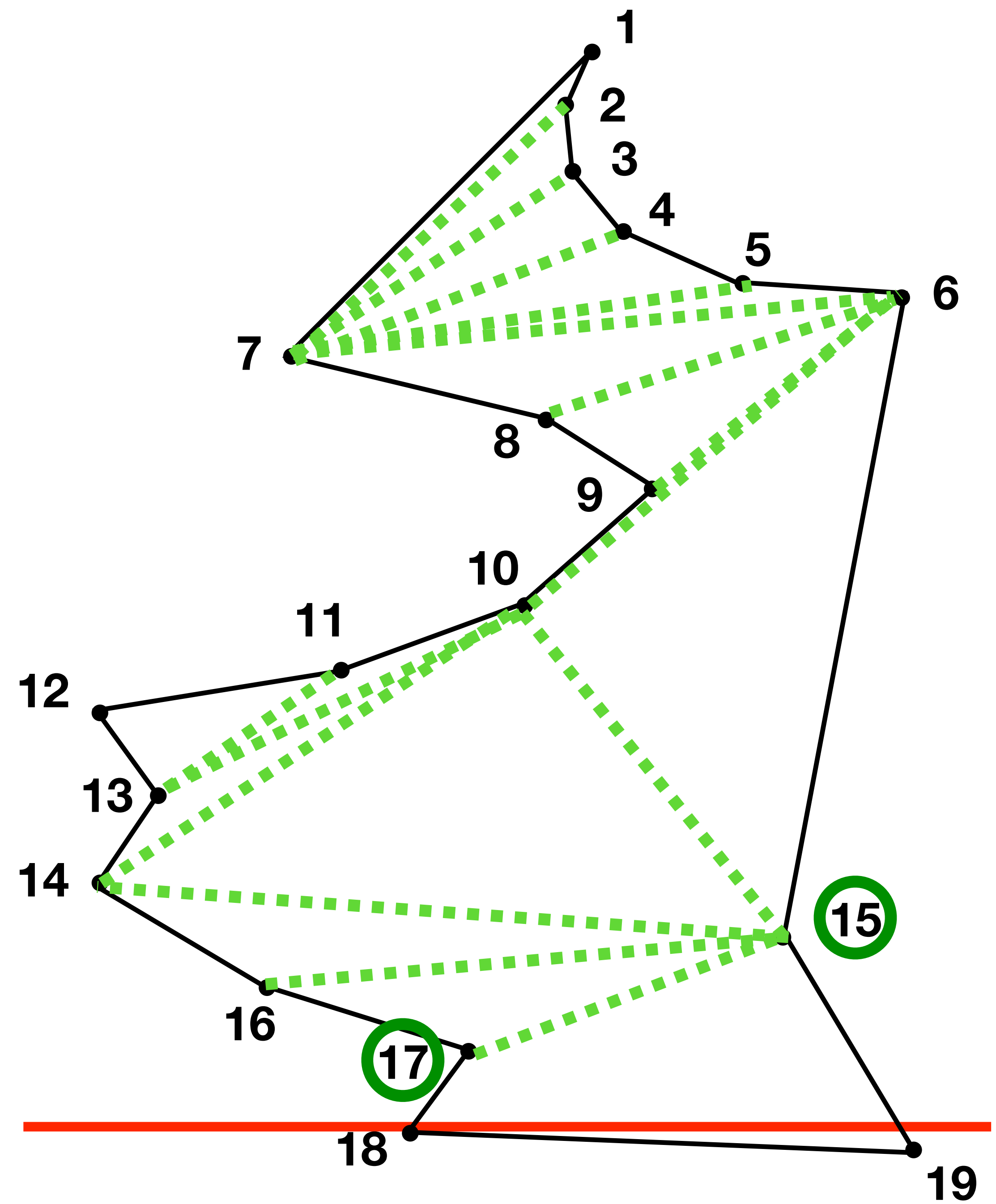
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 15 17



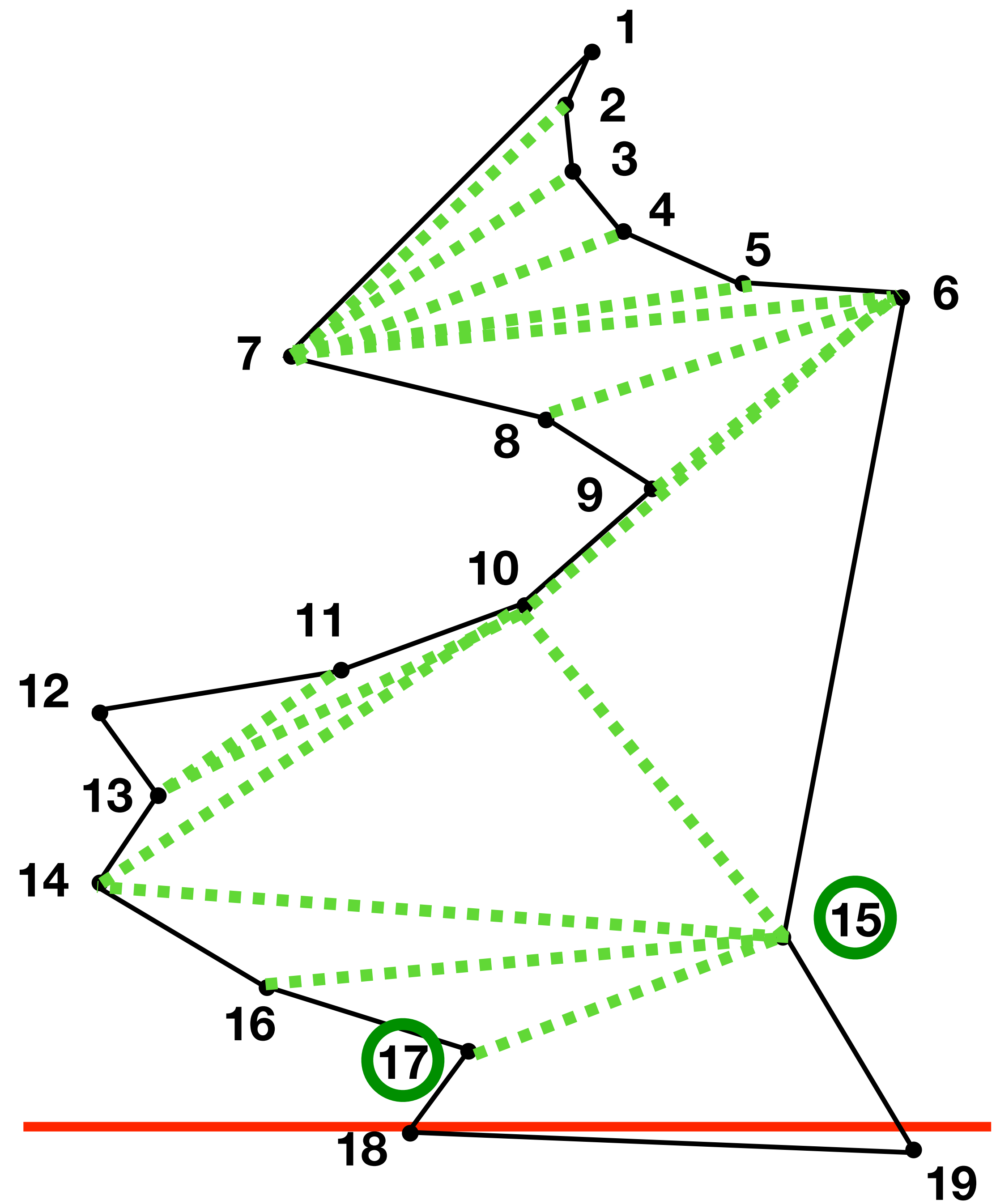
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 17



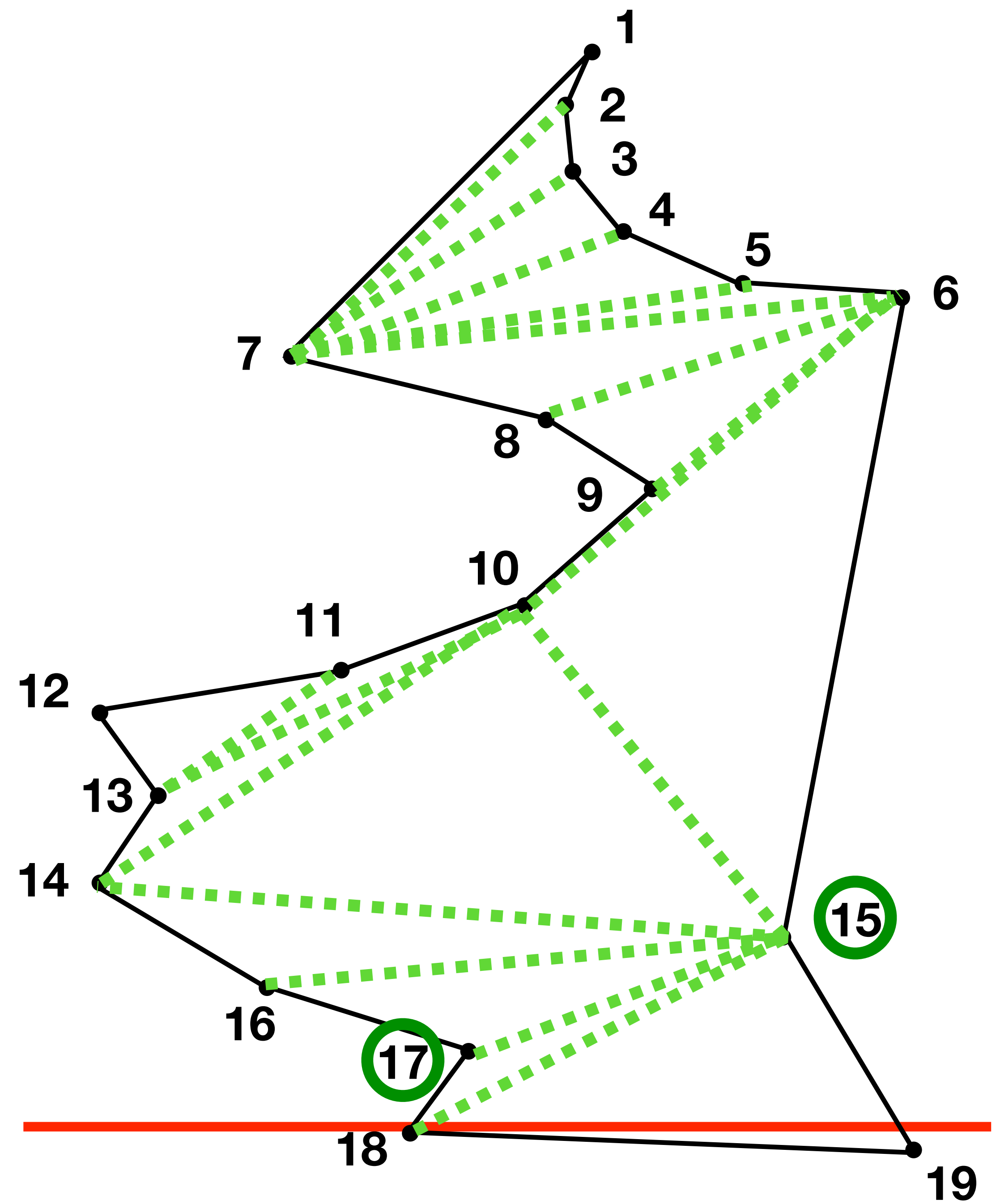
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 17



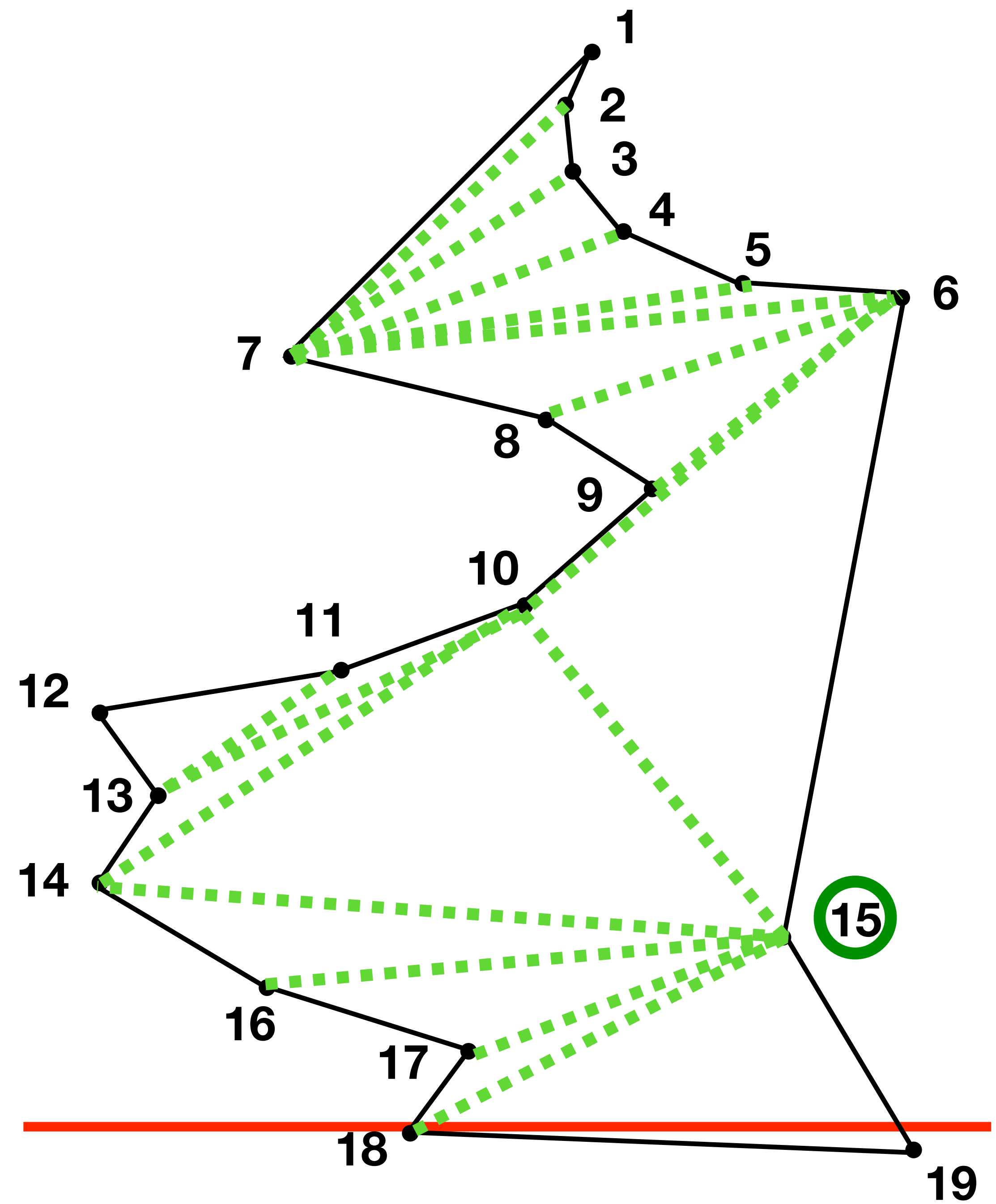
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15 17



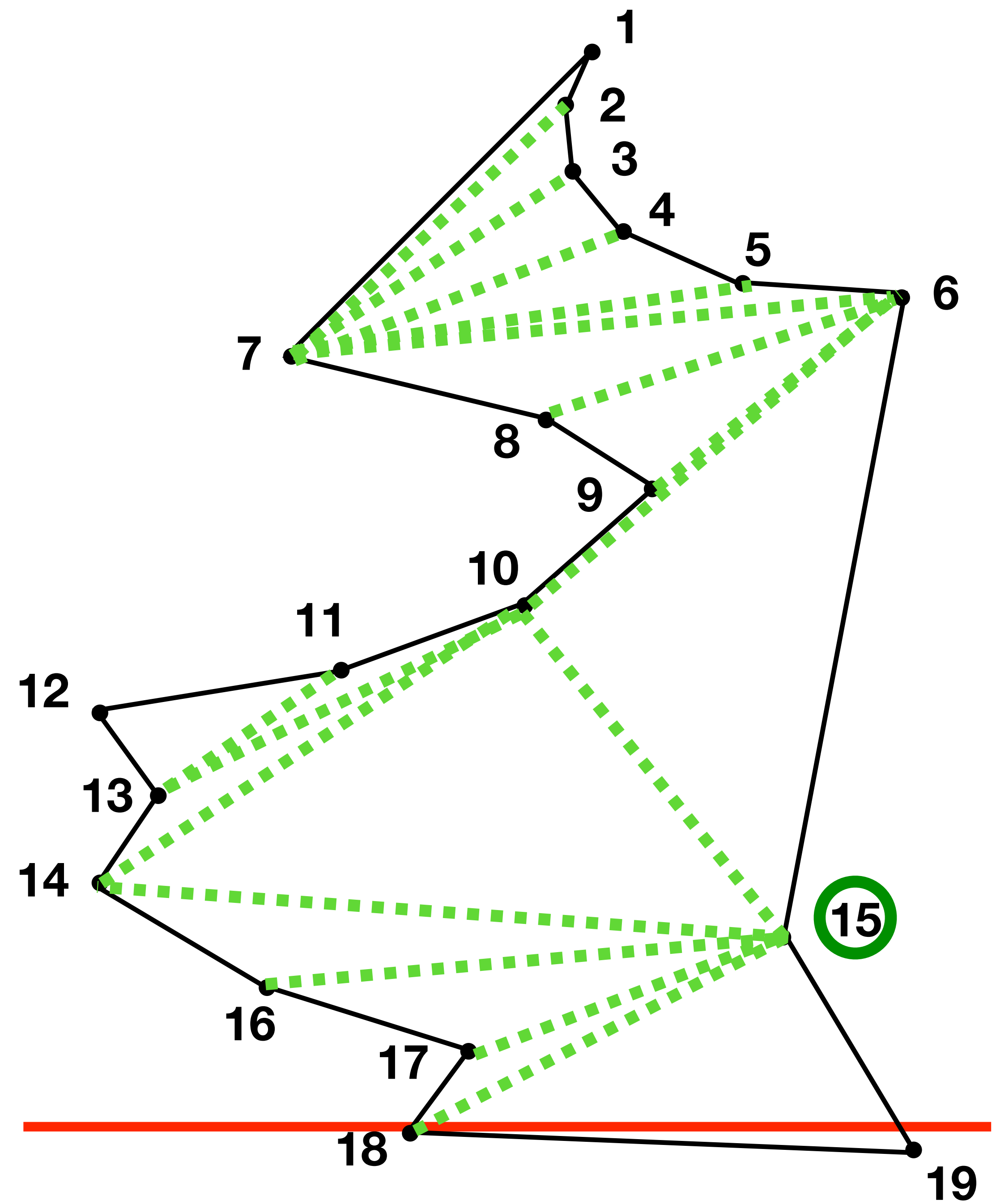
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15



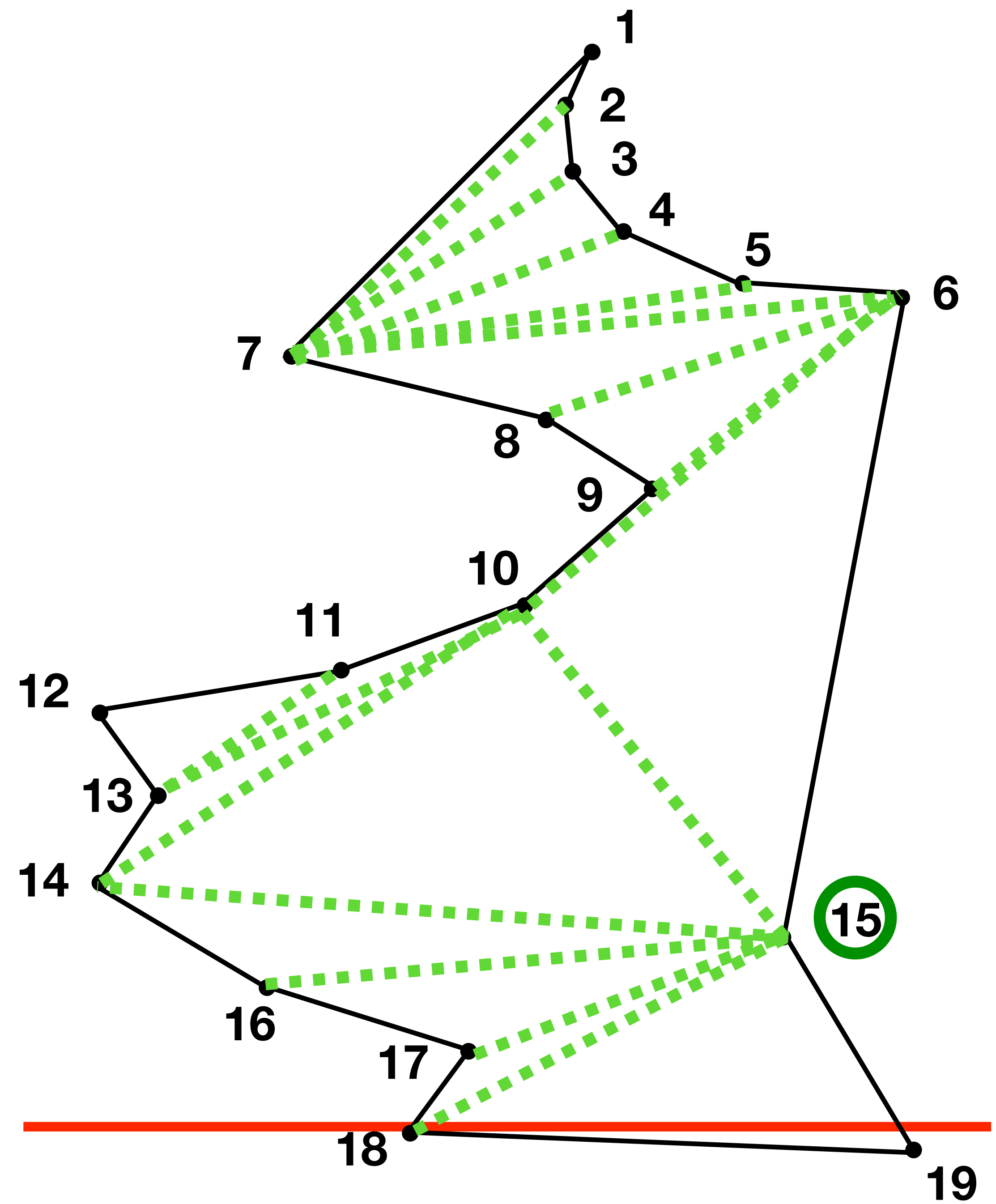
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15



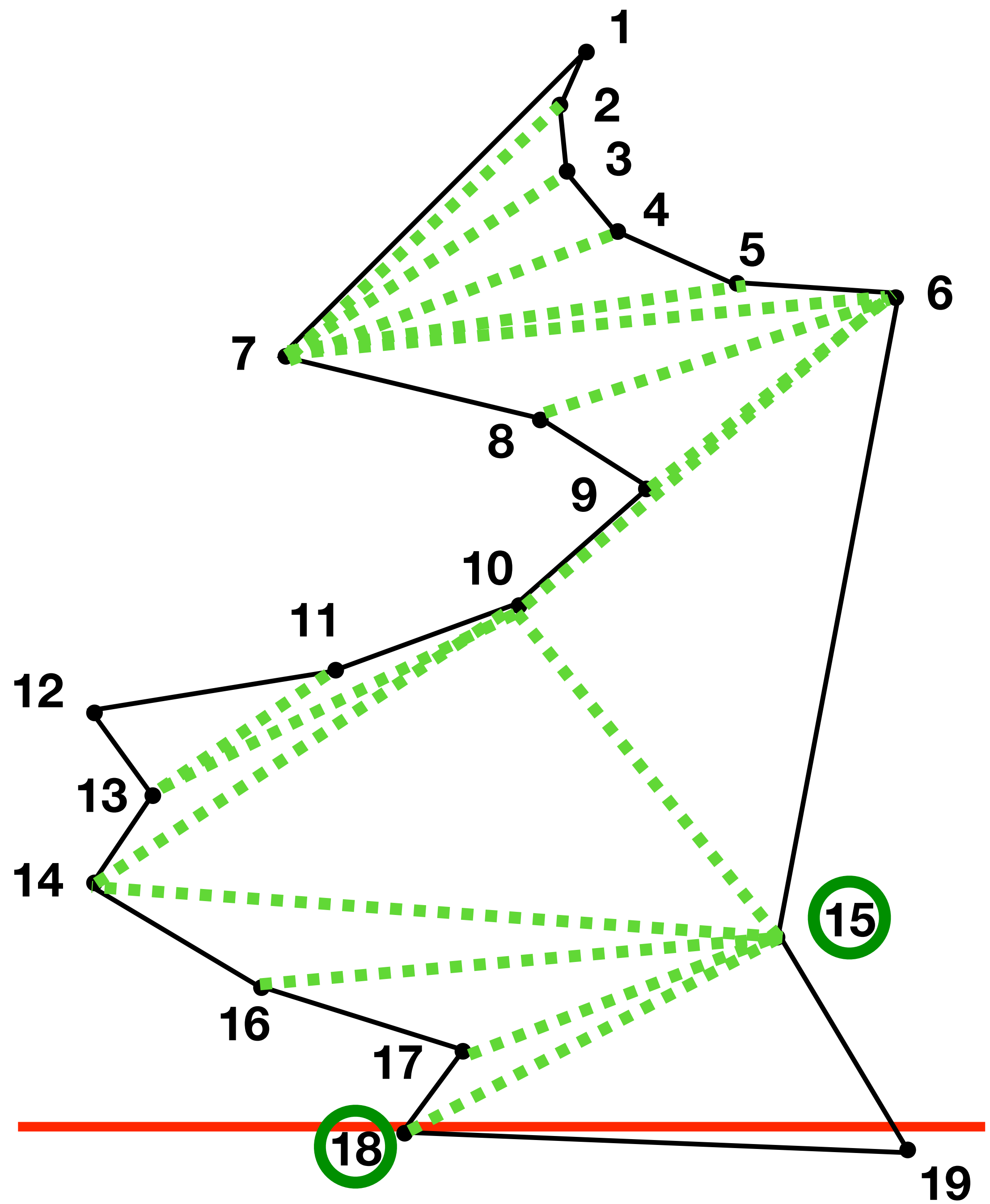
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue — **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue — **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update — **Ear**

Queue: 15



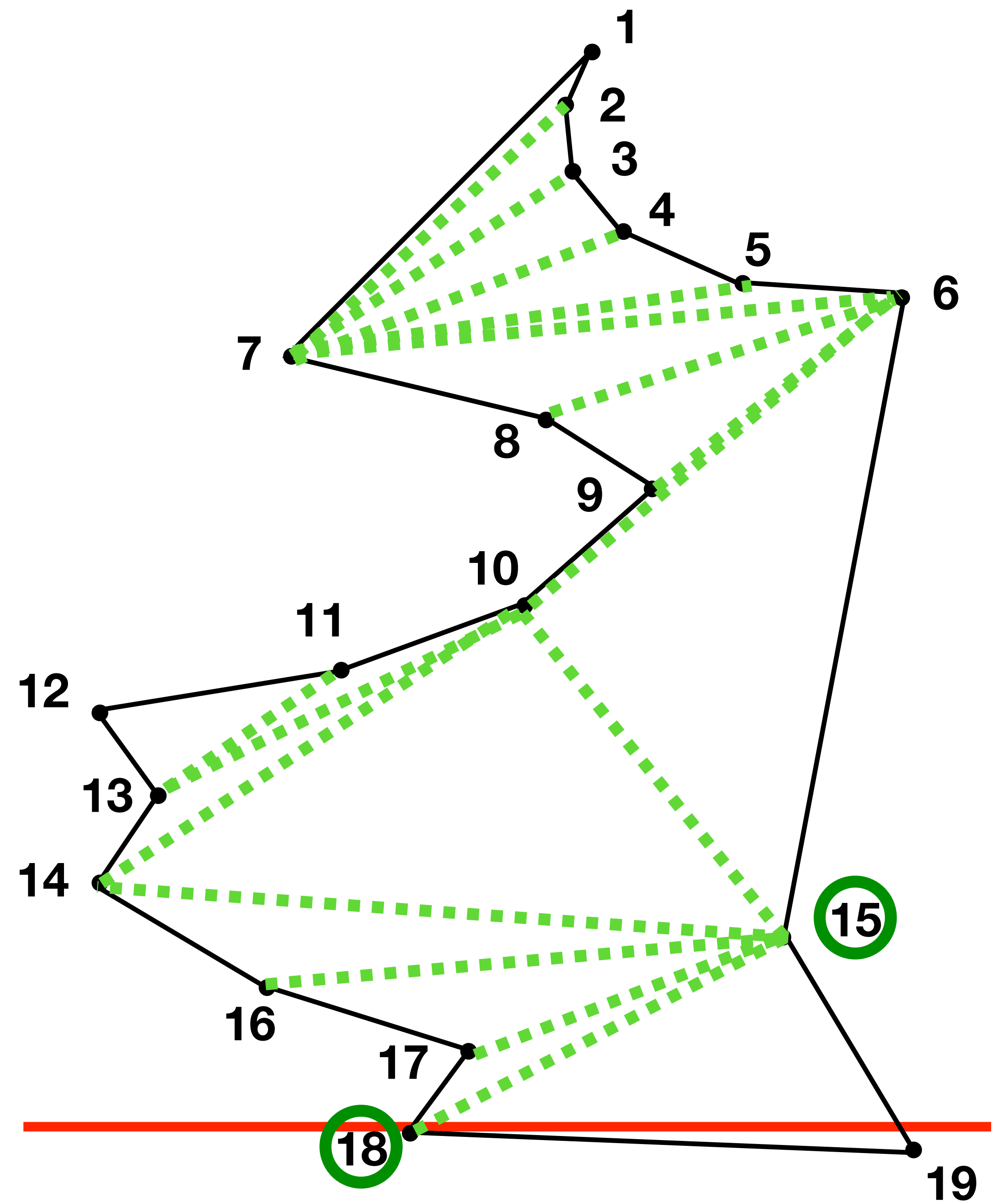
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 18



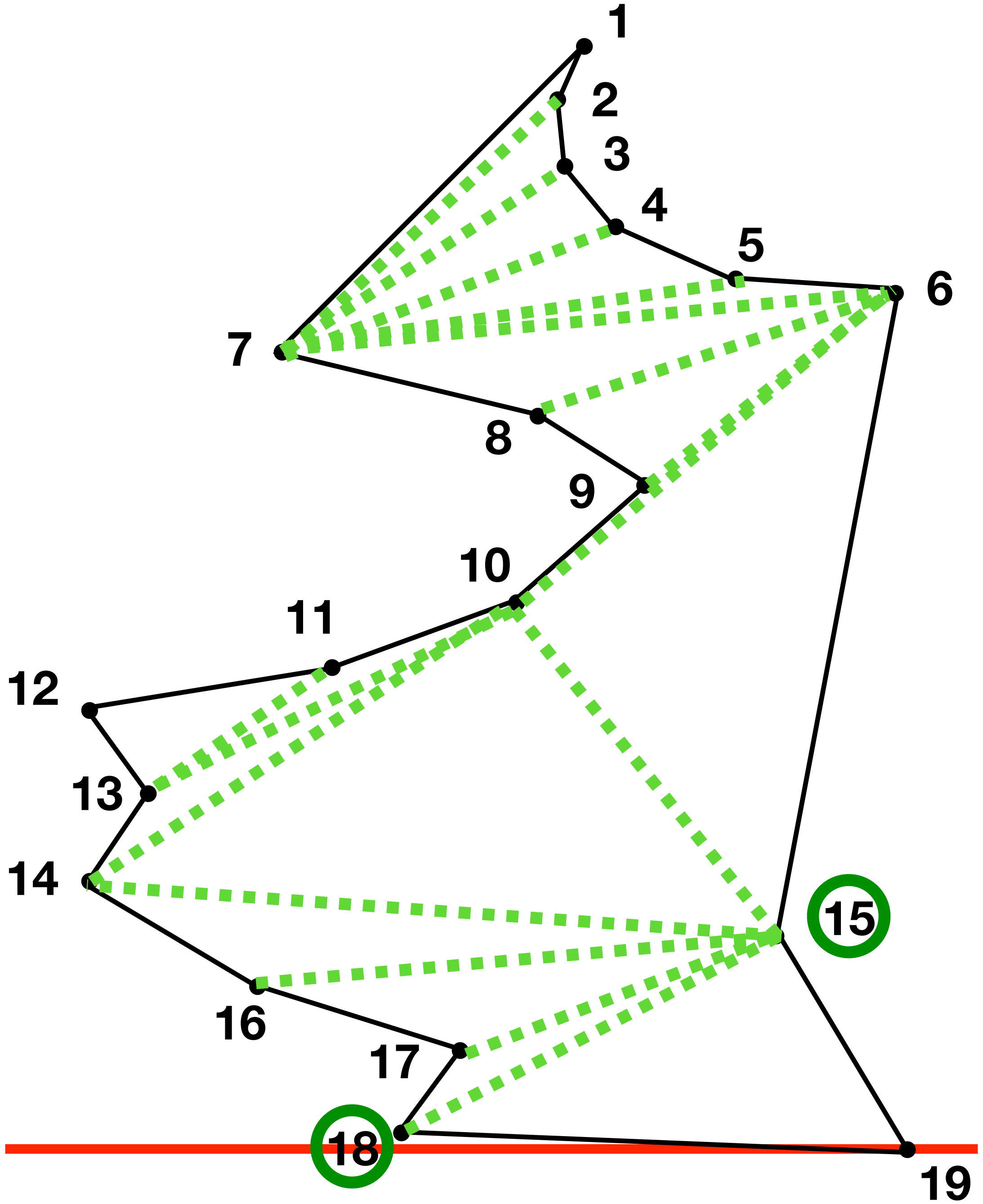
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 18



Algorithm 5.16

Input: A y-monotone polygon P

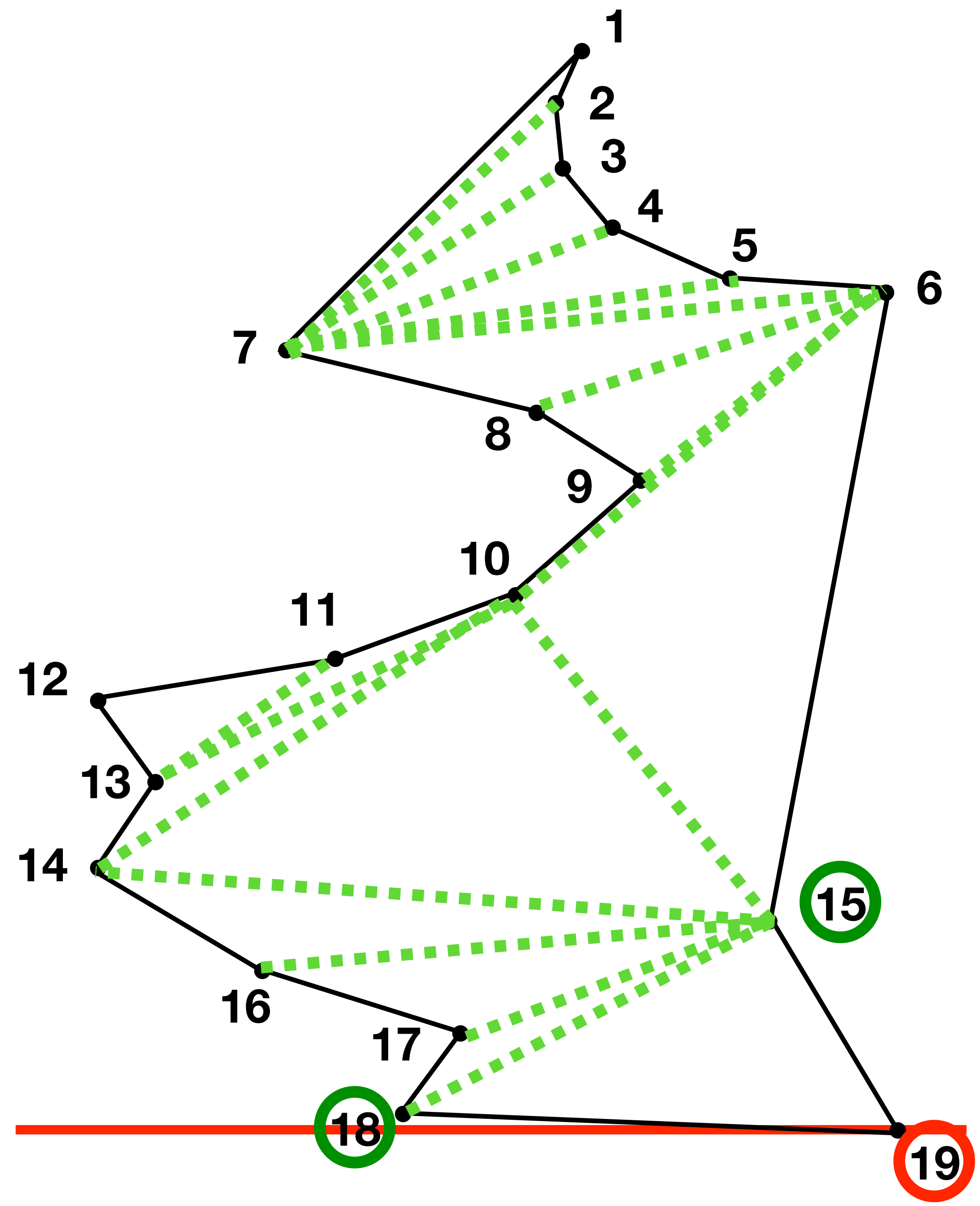
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 15 18



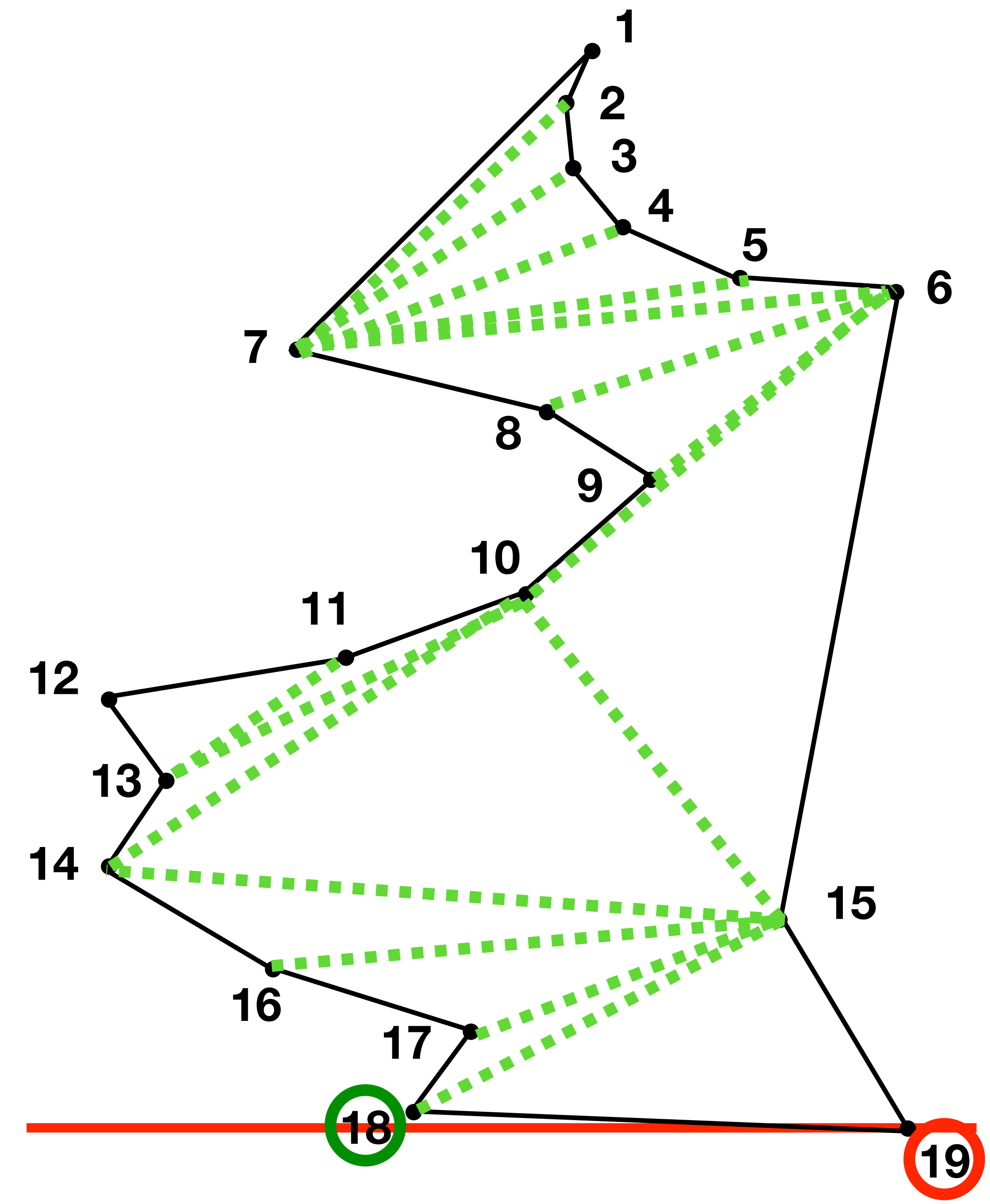
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 15 18



Algorithm 5.16

Input: A y-monotone polygon P

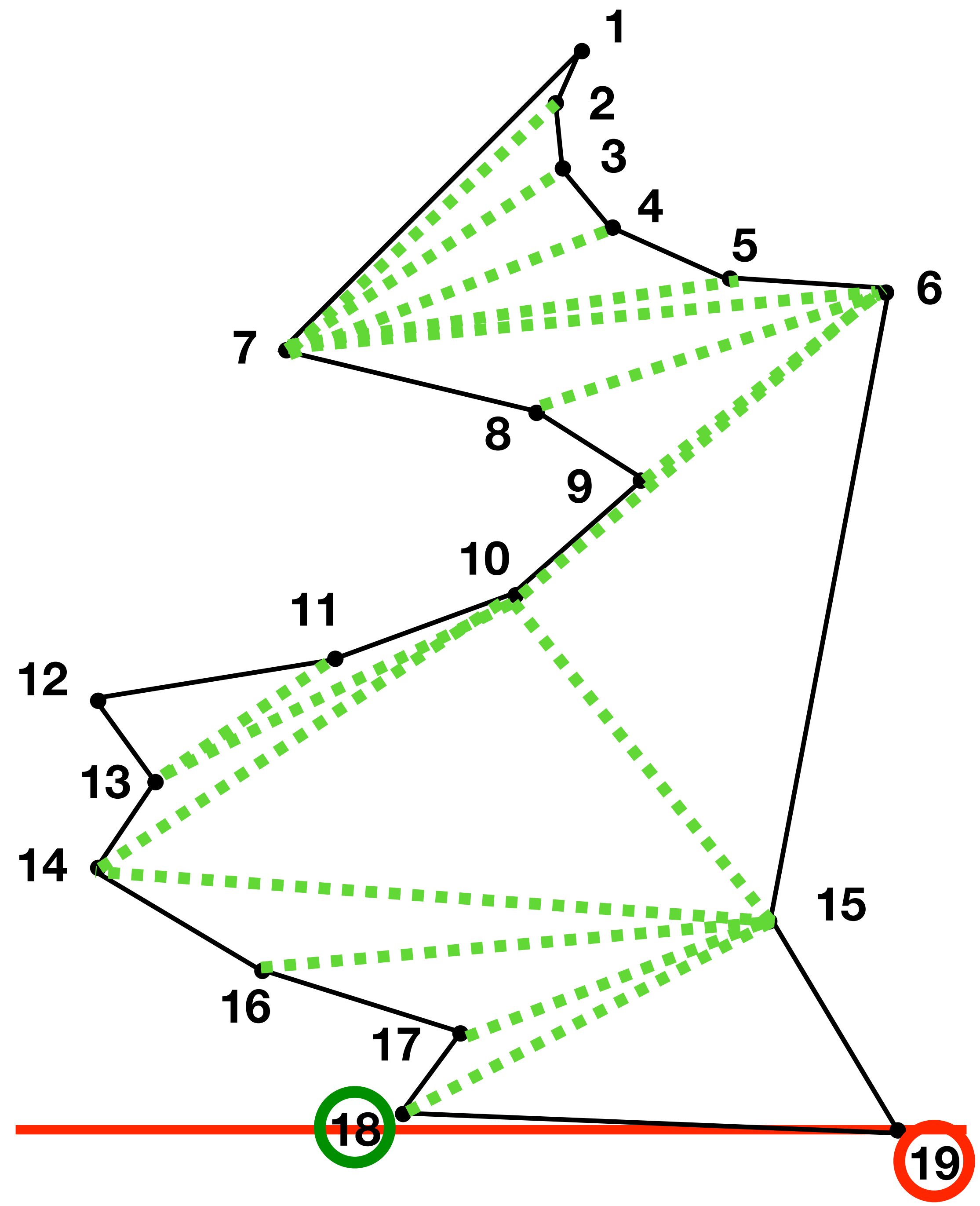
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue: 18



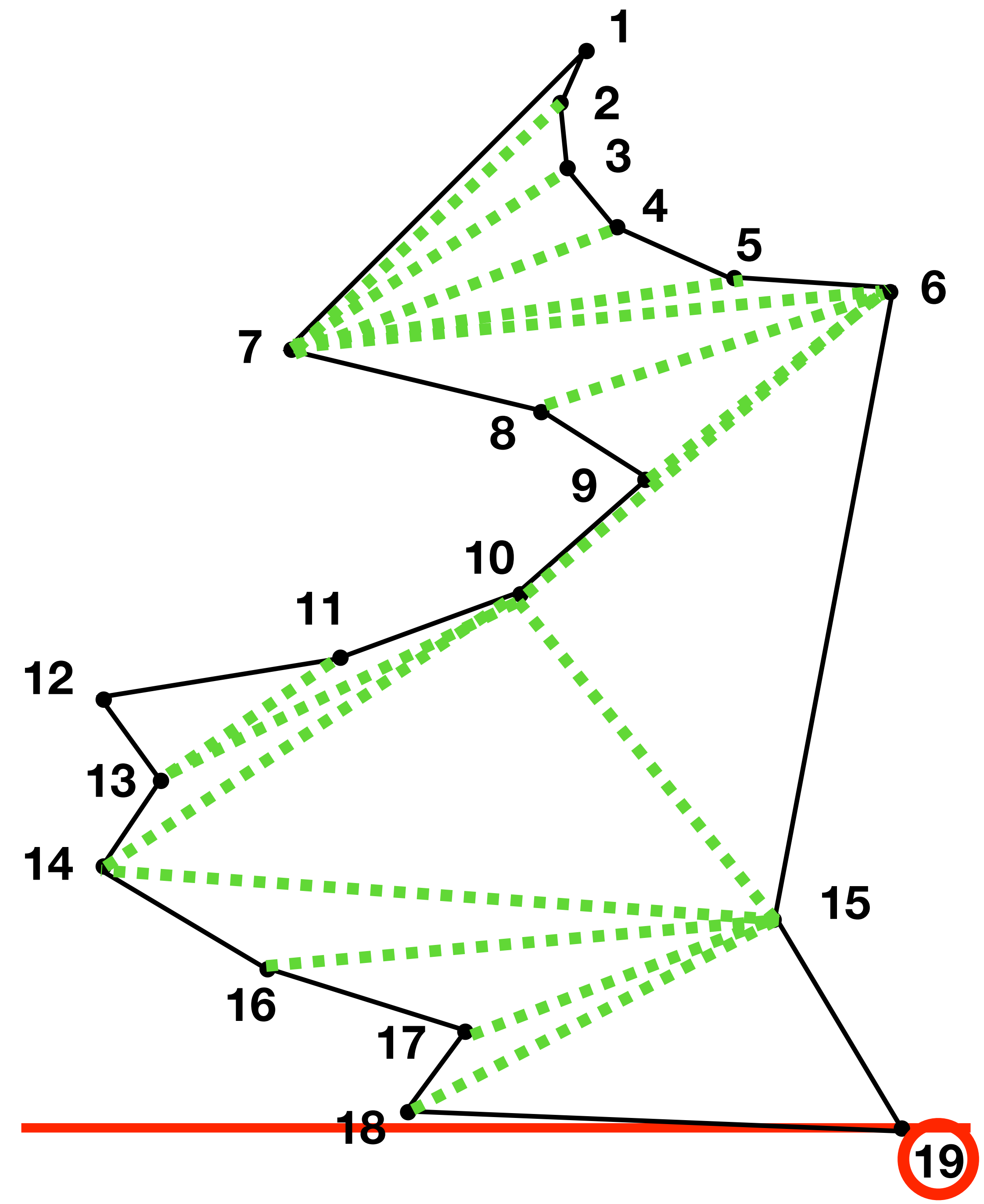
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue: 18



Algorithm 5.16

Input: A y-monotone polygon P

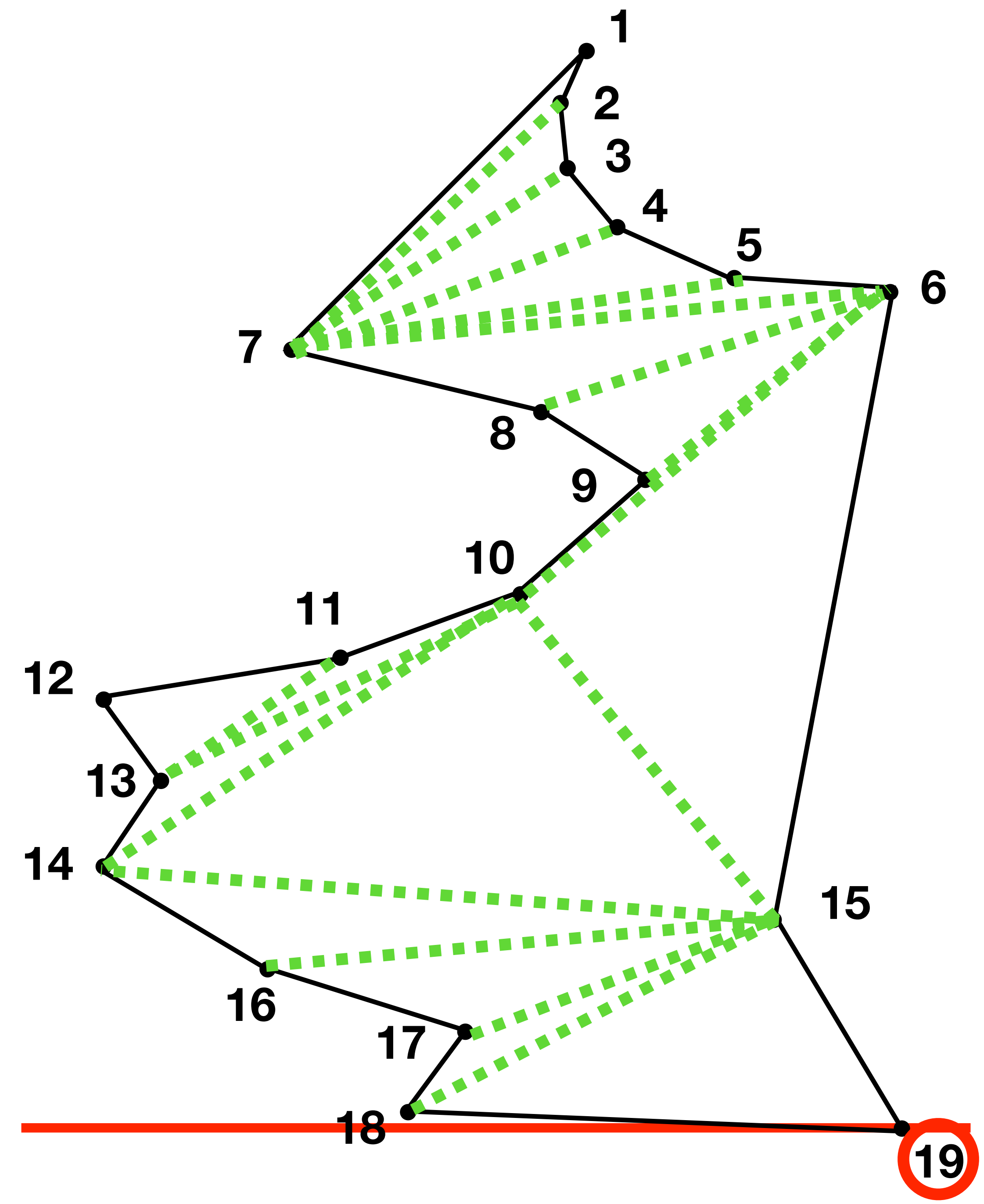
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Queue:



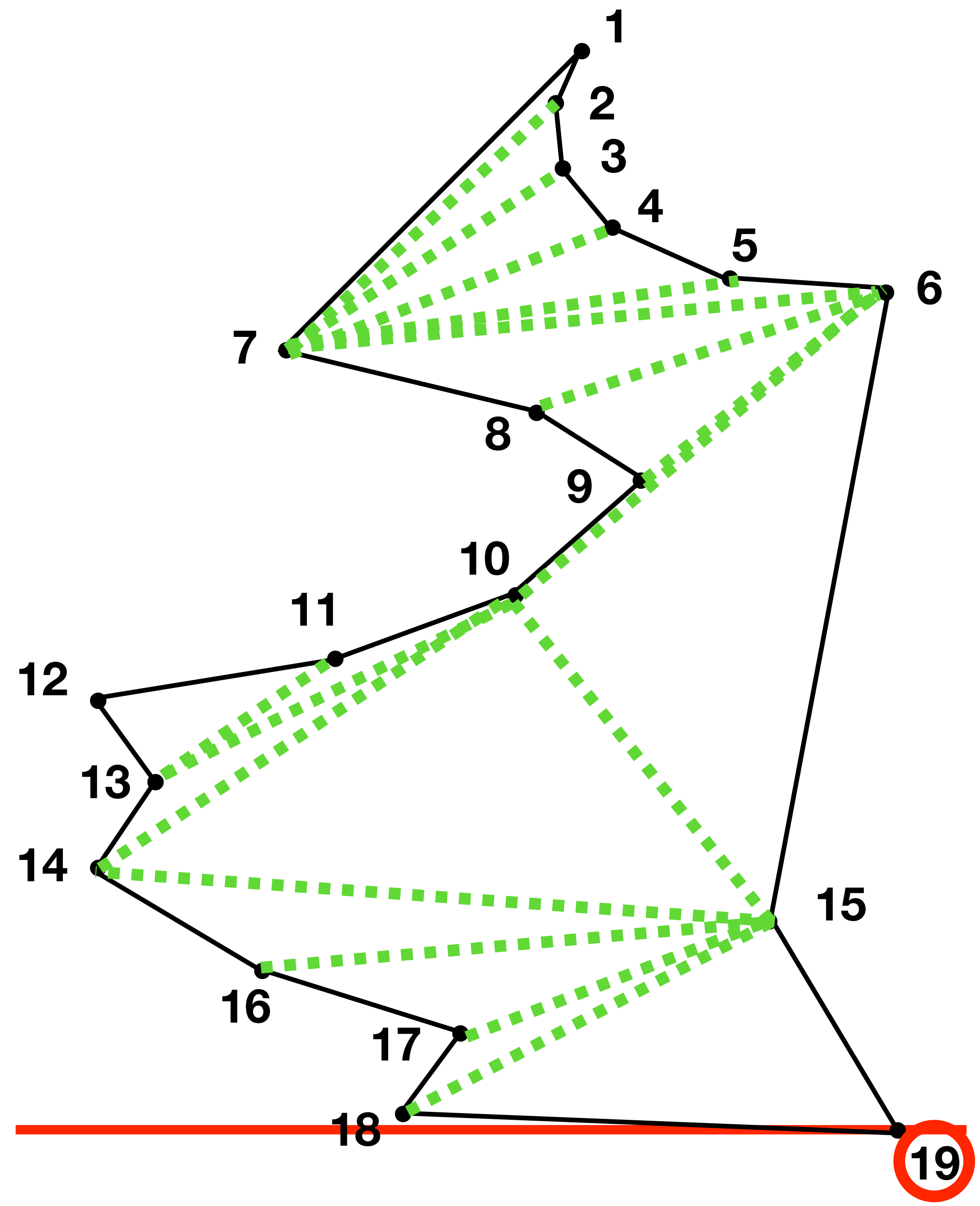
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue:



Theorem 5.17

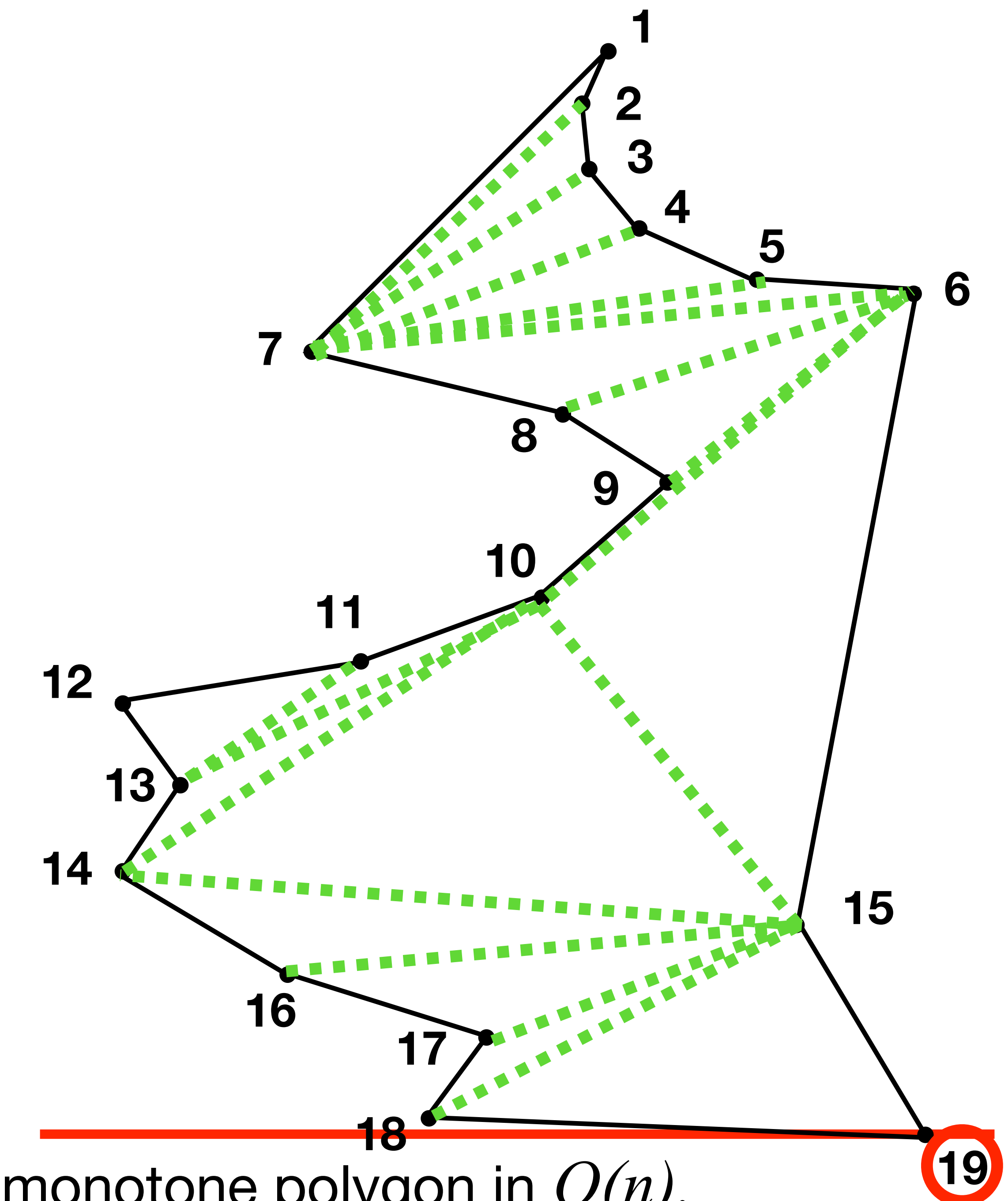
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue:



Theorem 5.17 Algorithm 5.16 can triangulate a y-monotone polygon in $O(n)$.

Algorithm 5.16

Input: A y-monotone polygon P

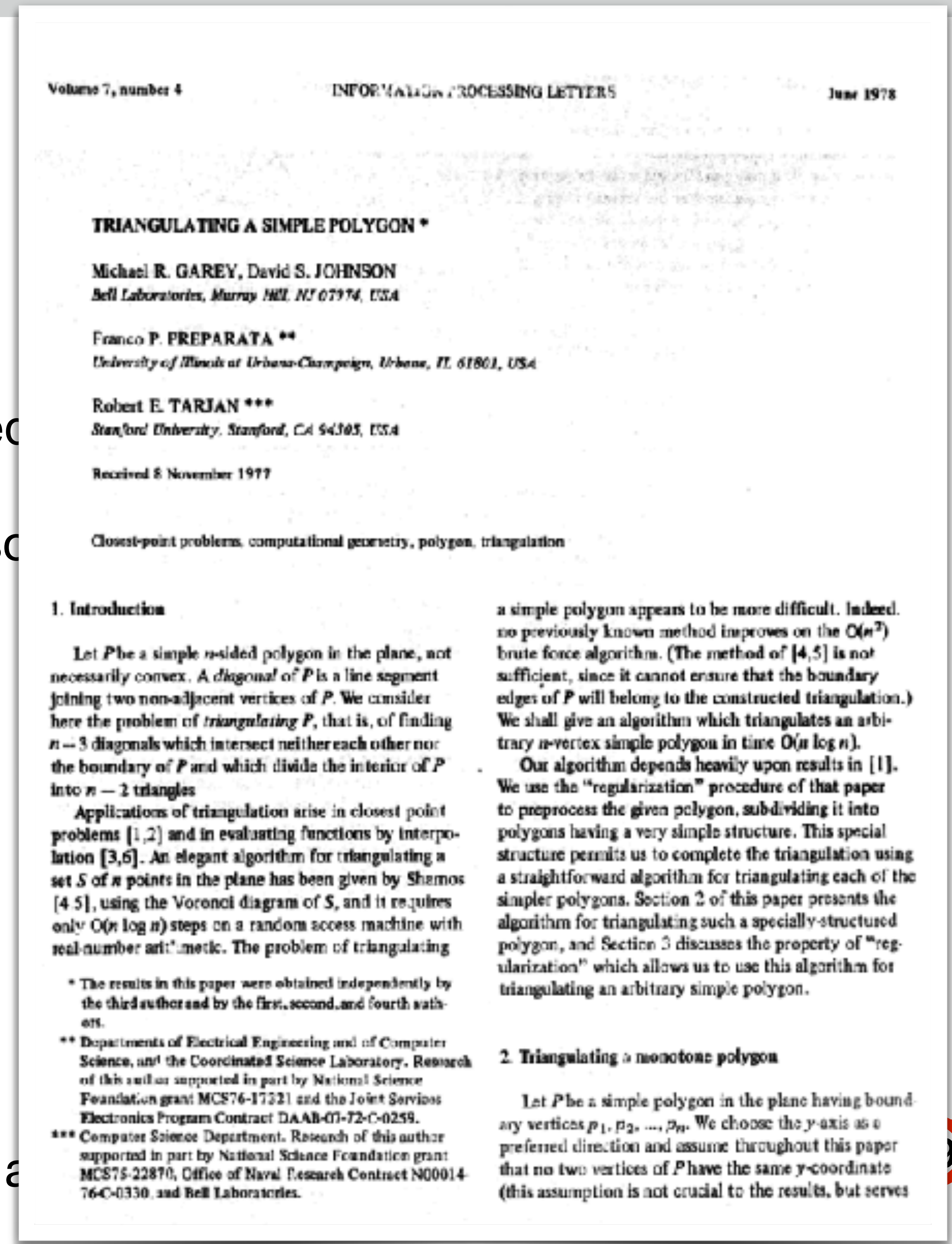
Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   3.1 If opposite chain:
   - Connect to all vertices in chain, leave in queue **Chain**

   3.2 If same chain, predecessor reflex:
   - Add to queue **Add**

   3.3 If same chain, predecessor convex:
   - Ear! Connect and update **Ear**

Theorem 5.17 Algorithm 5.16 can triangulate a



Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles.

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

2. Triangulating a monotone polygon

Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

Technische Universität Braunschweig

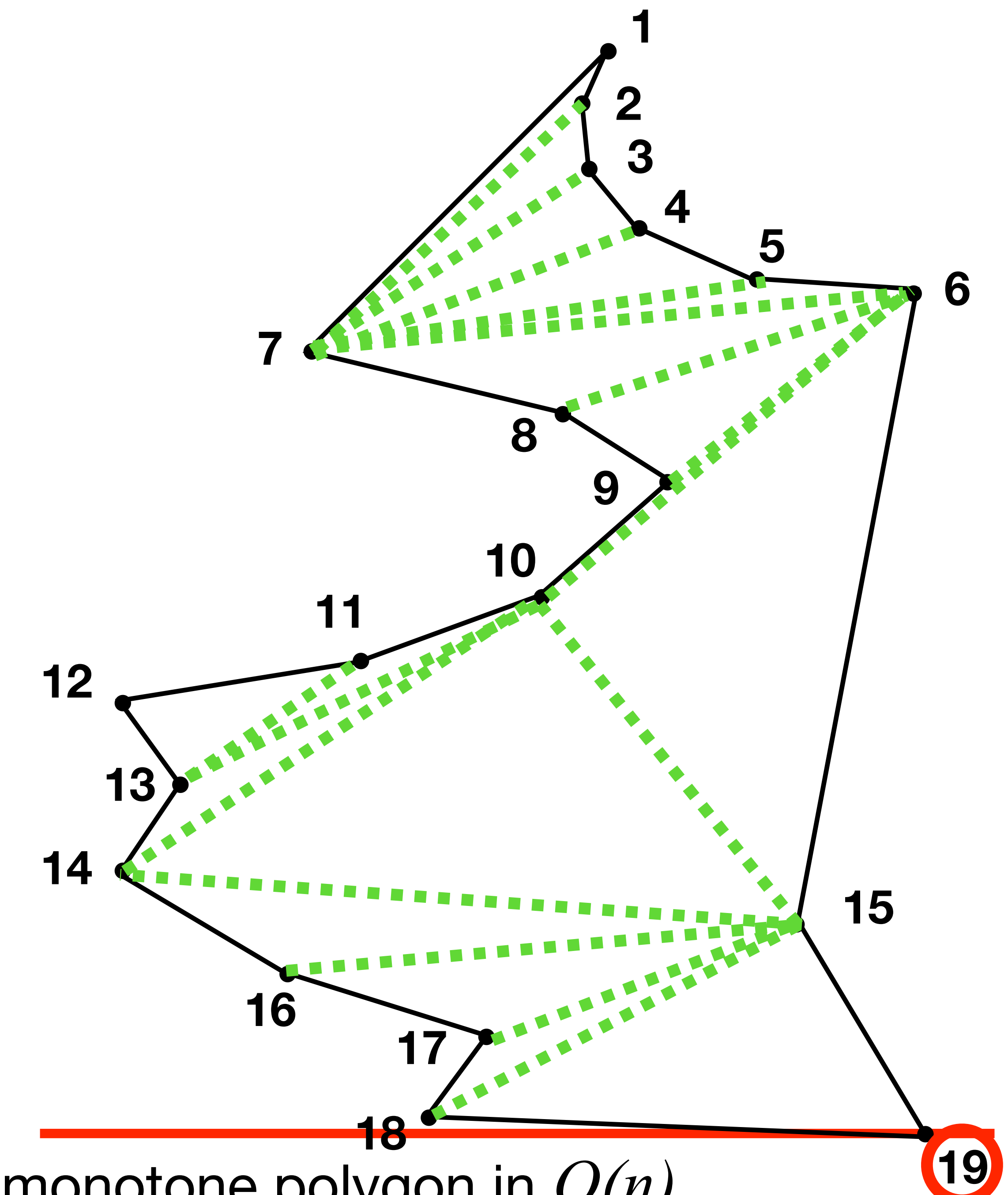
Algorithm 5.16

Input: A y-monotone polygon P

Output: A triangulation of P

1. Sort vertices by y-coordinate
2. Maintain queue of vertices still to be processed
   - First two vertices of unfinished subpolygon
   - Further vertices in same chain if predecessor reflex
3. Processing vertices in sorted order:
   - 3.1 If opposite chain:
     - Connect to all vertices in chain, leave in queue **Chain**
   - 3.2 If same chain, predecessor reflex:
     - Add to queue **Add**
   - 3.3 If same chain, predecessor convex:
     - Ear! Connect and update **Ear**

Queue:



Theorem 5.17 Algorithm 5.16 can triangulate a y-monotone polygon in $O(n)$.

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let *P* be a simple *n*-sided polygon in the plane, not necessarily convex. A *diagonal* of *P* is a line segment joining two non-adjacent vertices of *P*. We consider here the problem of *triangulating P*, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of *P* and which divide the interior of *P* into $n-2$ triangles.

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set *S* of *n* points in the plane has been given by Shamos [4,5], using the Voronoi diagram of *S*, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of *P* will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary *n*-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

2. Triangulating a monotone polygon

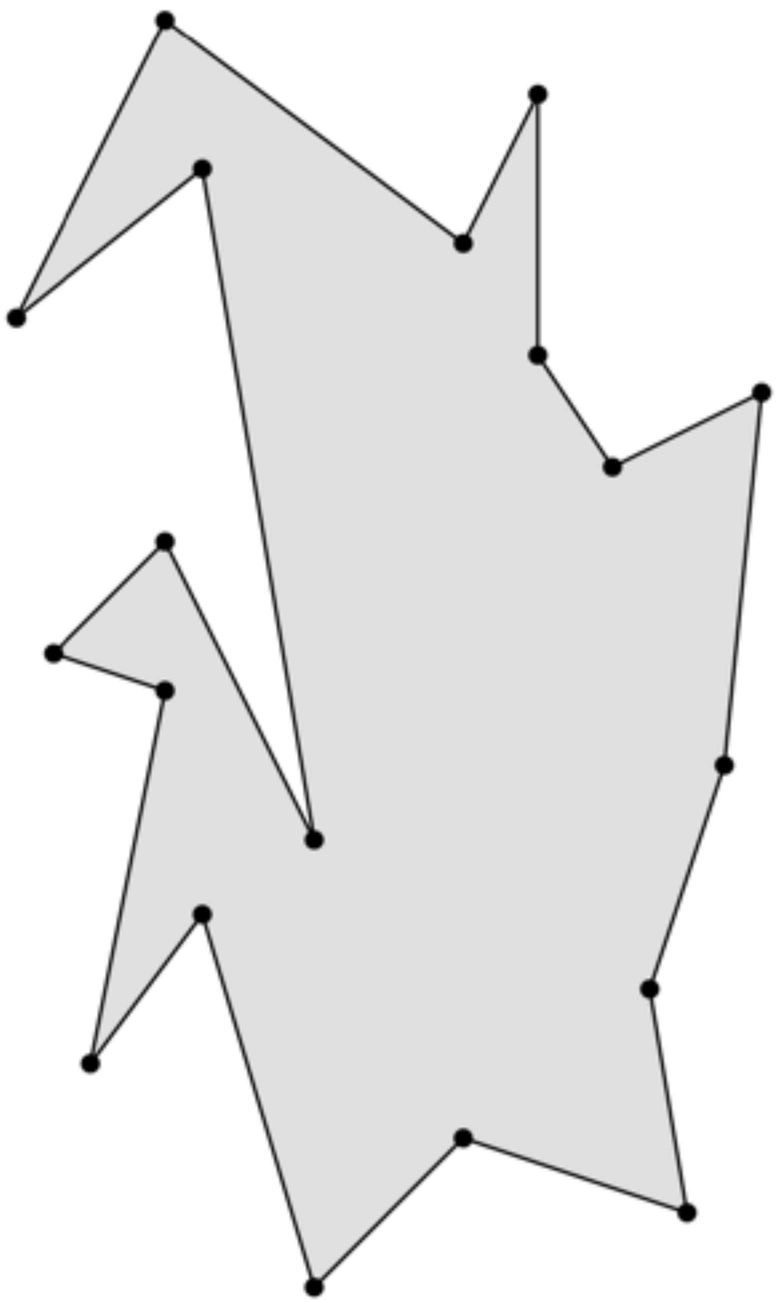
Let *P* be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the *y*-axis as a preferred direction and assume throughout this paper that no two vertices of *P* have the same *y*-coordinate (this assumption is not crucial to the results, but serves

* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]



Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

2. Triangulating a monotone polygon

Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

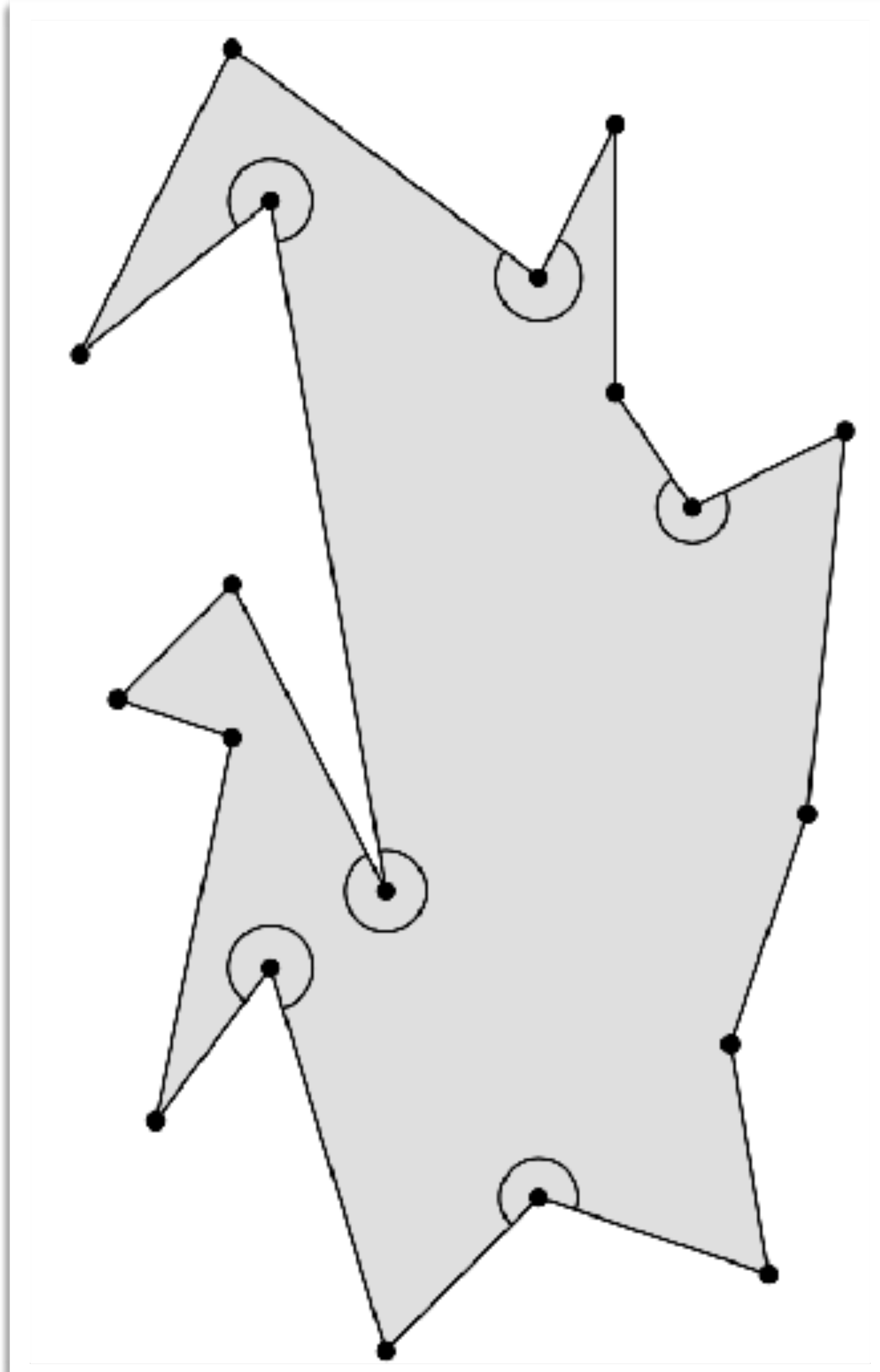
* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

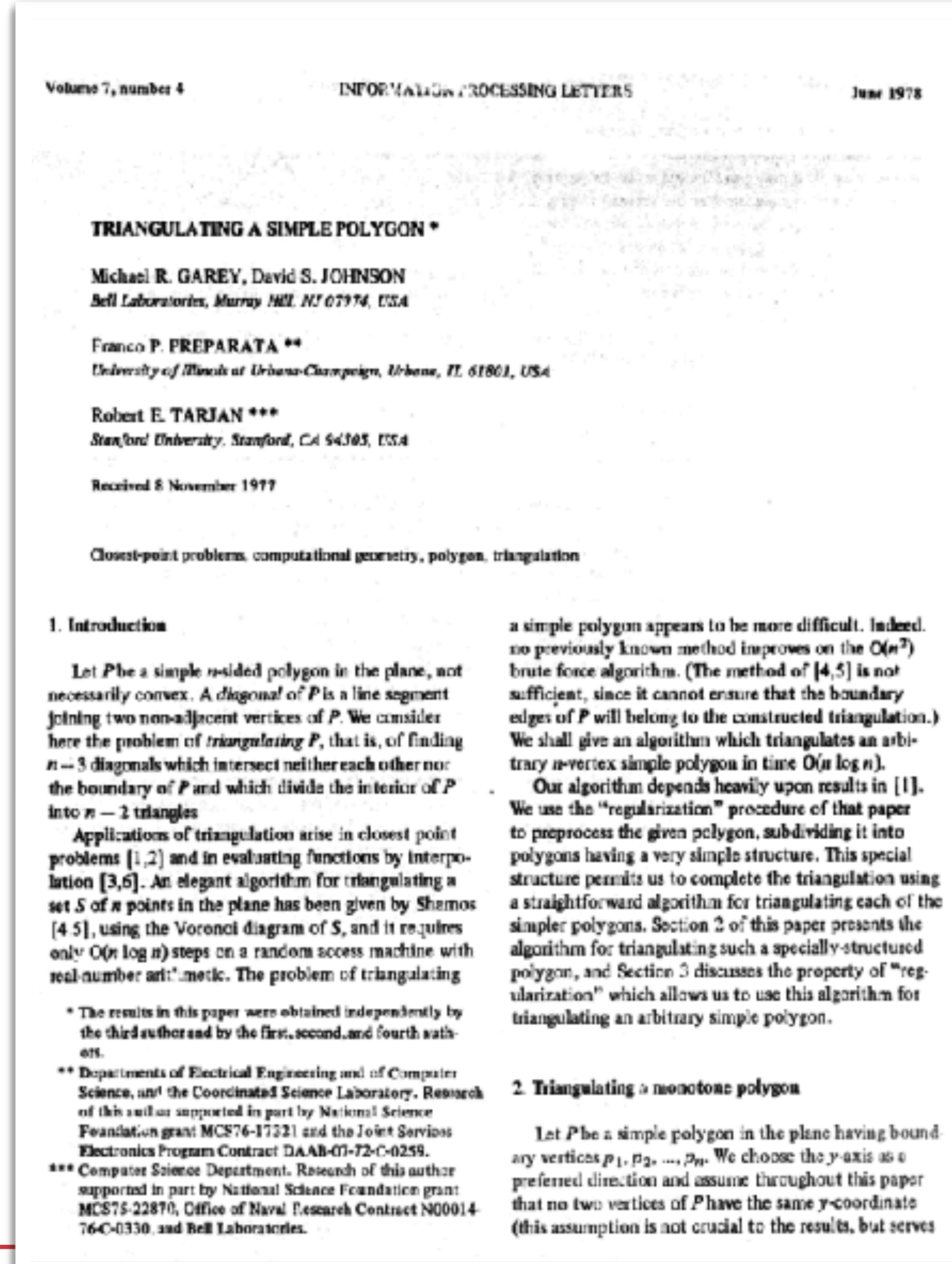
** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

Technische Universität Braunschweig

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]





Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles.

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

2. Triangulating a monotone polygon

Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

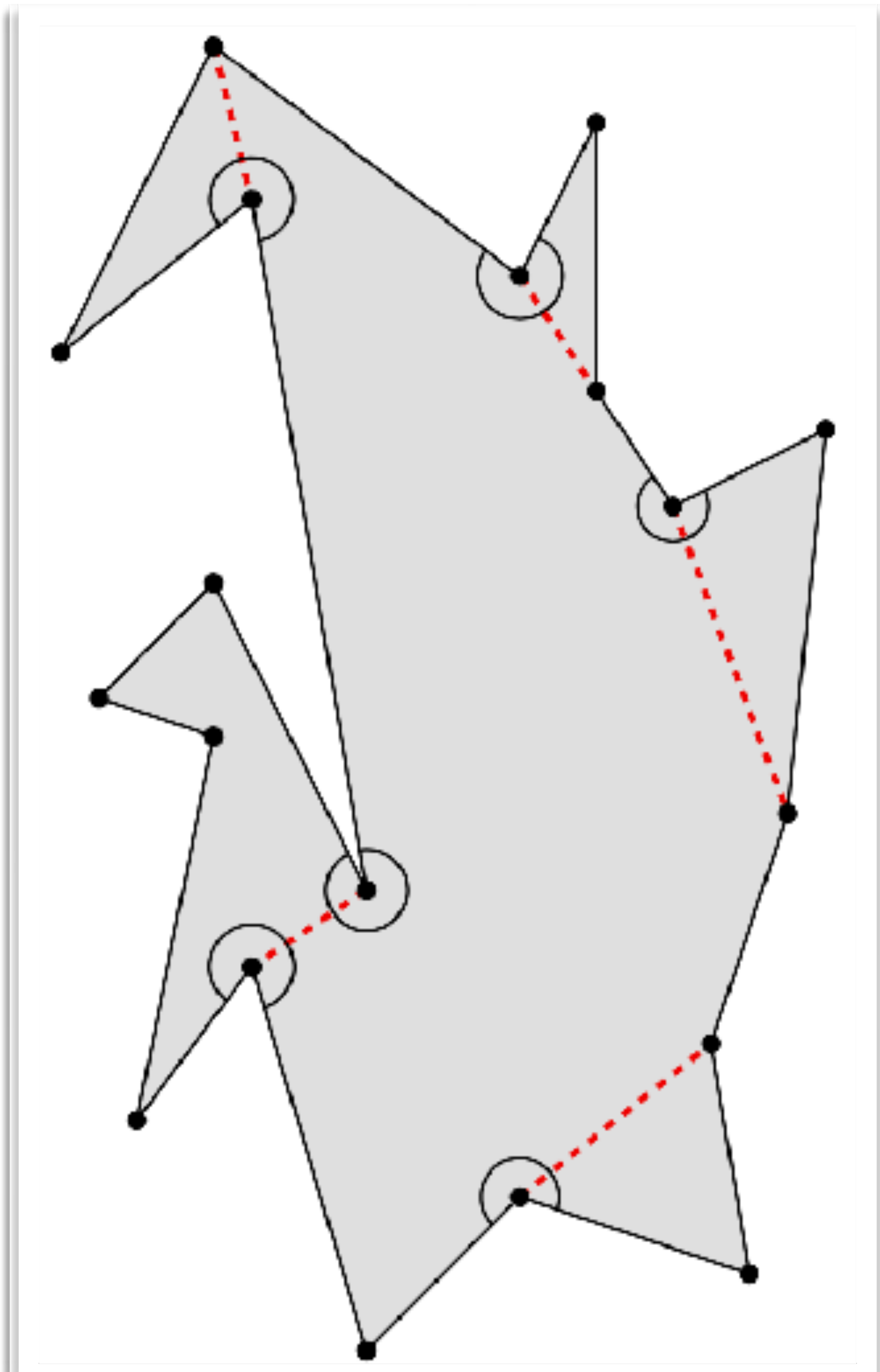
* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

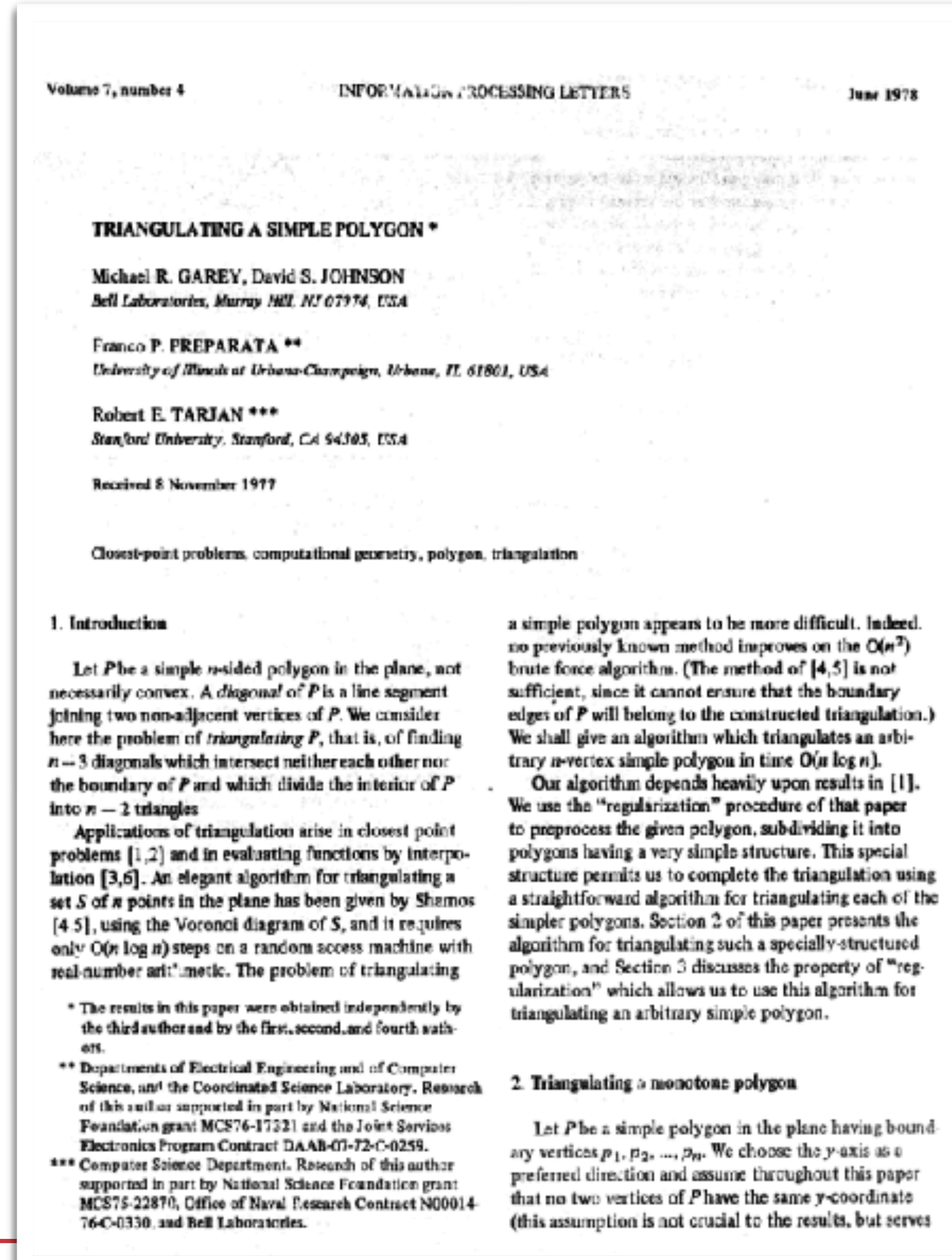
** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

Technische Universität Braunschweig

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]





Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles.

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

2. Triangulating a monotone polygon

Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

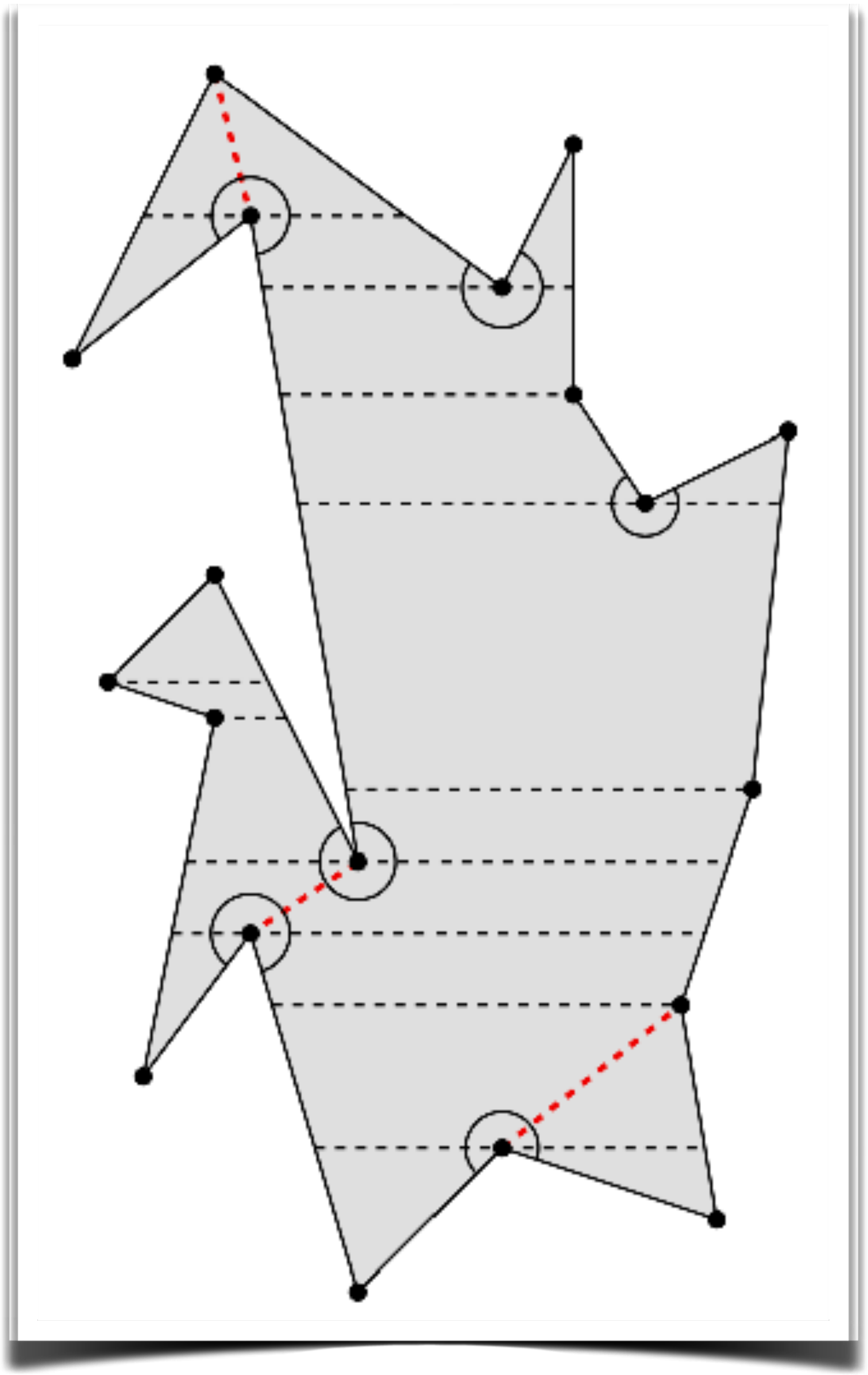
* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

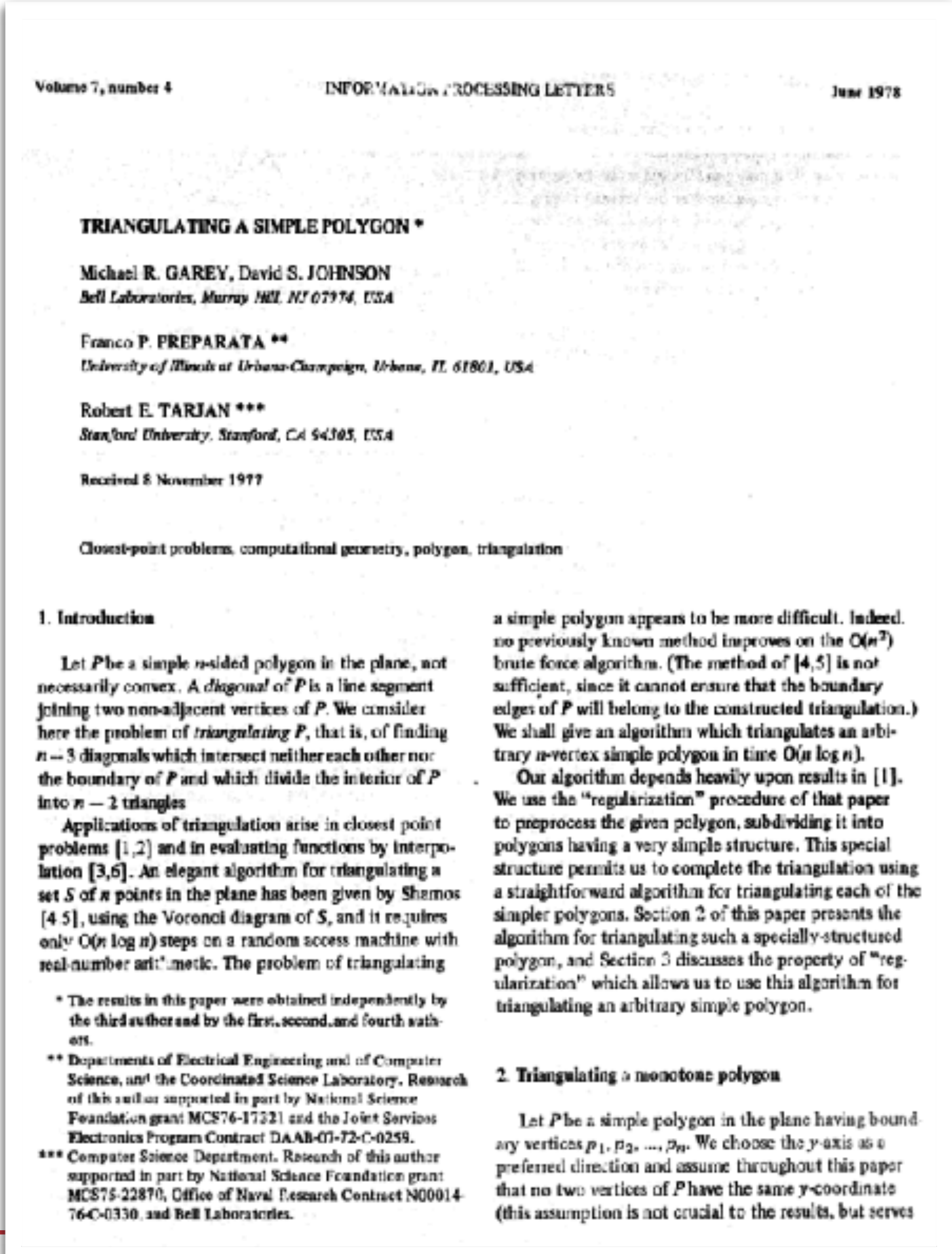
** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

Technische Universität Braunschweig

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]





Volume 7, number 4 INFORMATION PROCESSING LETTERS June 1978

TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles.

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

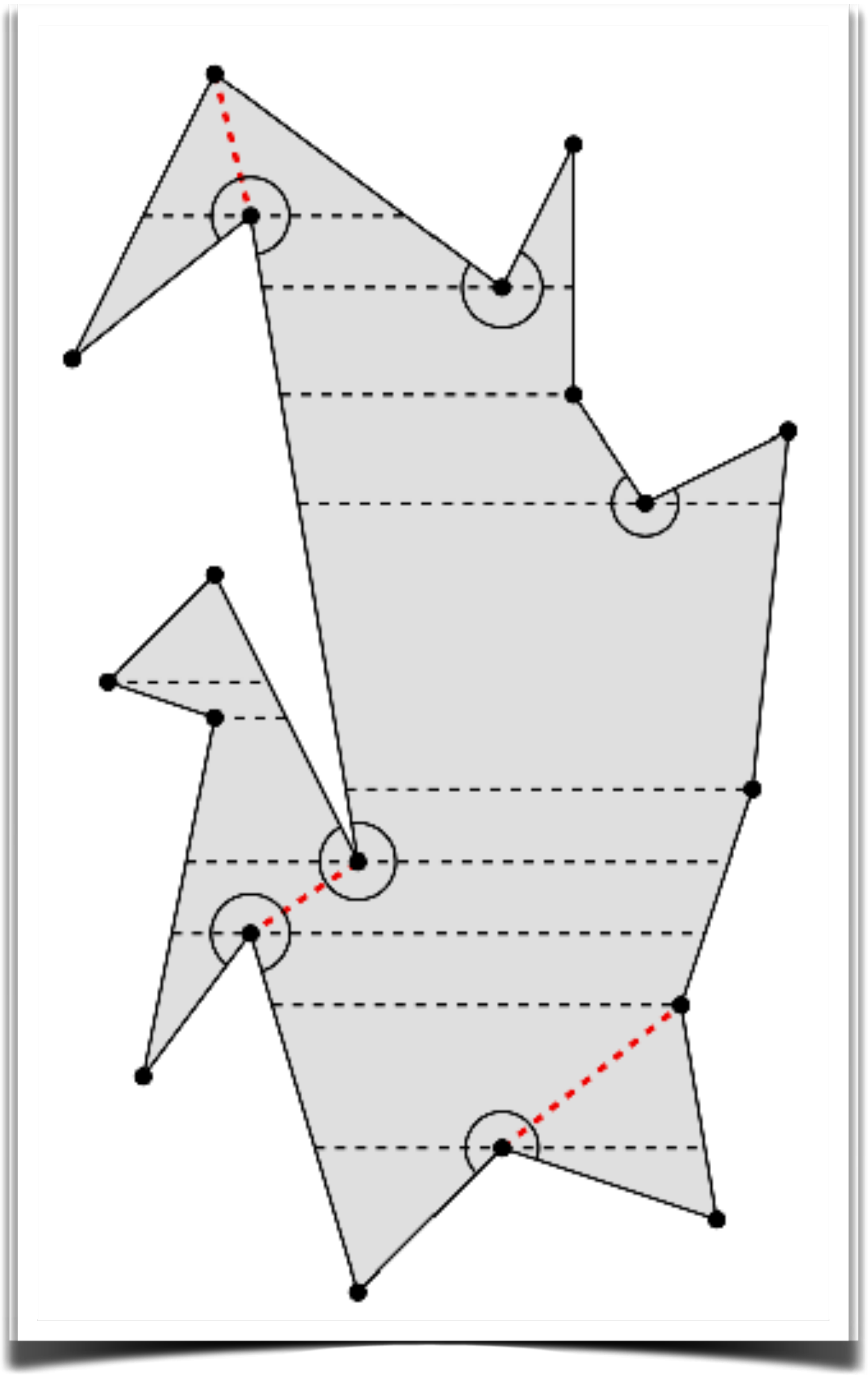
*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

2. Triangulating a monotone polygon

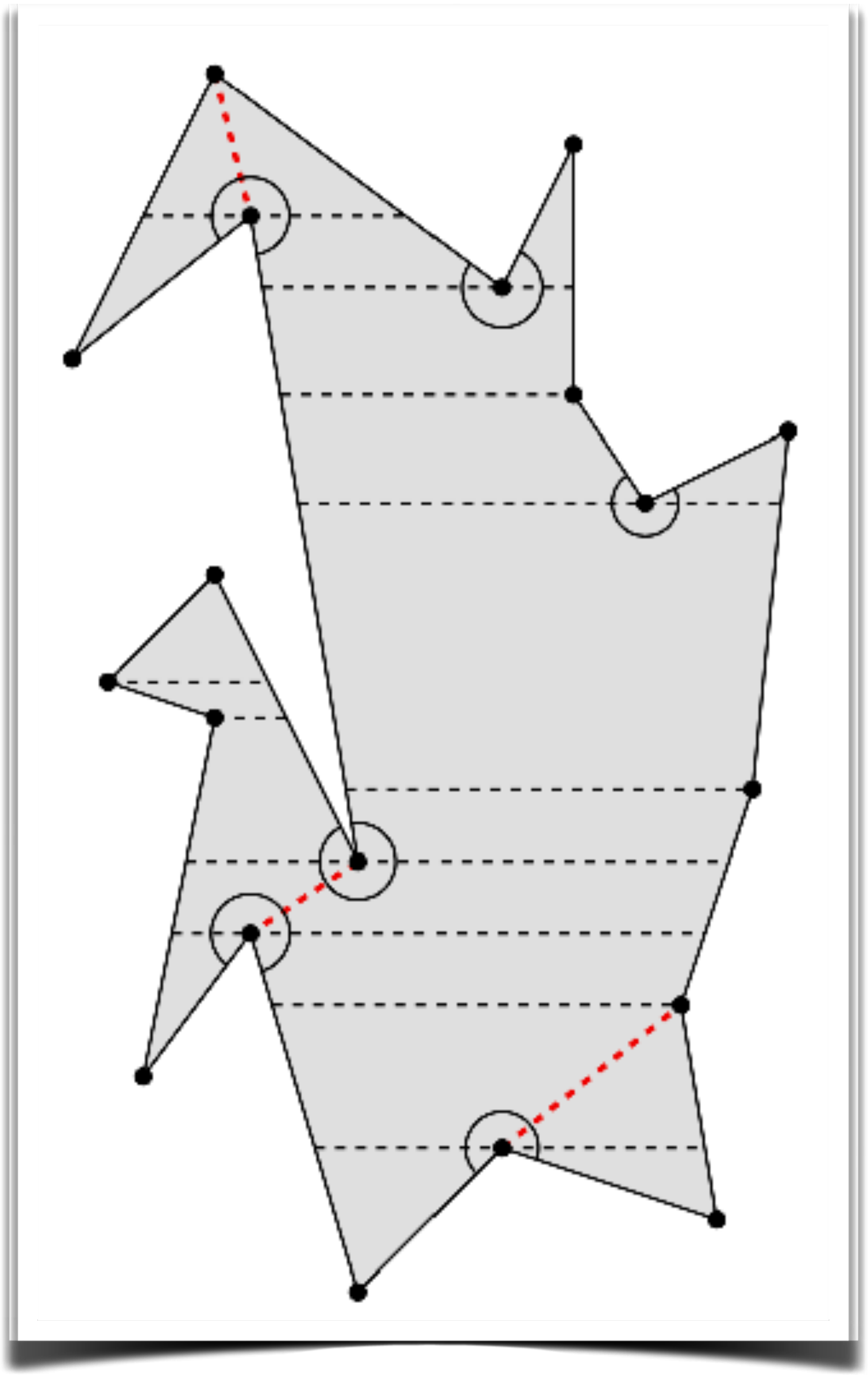
Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, ..., p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

Technische
Universität
Braunschweig

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

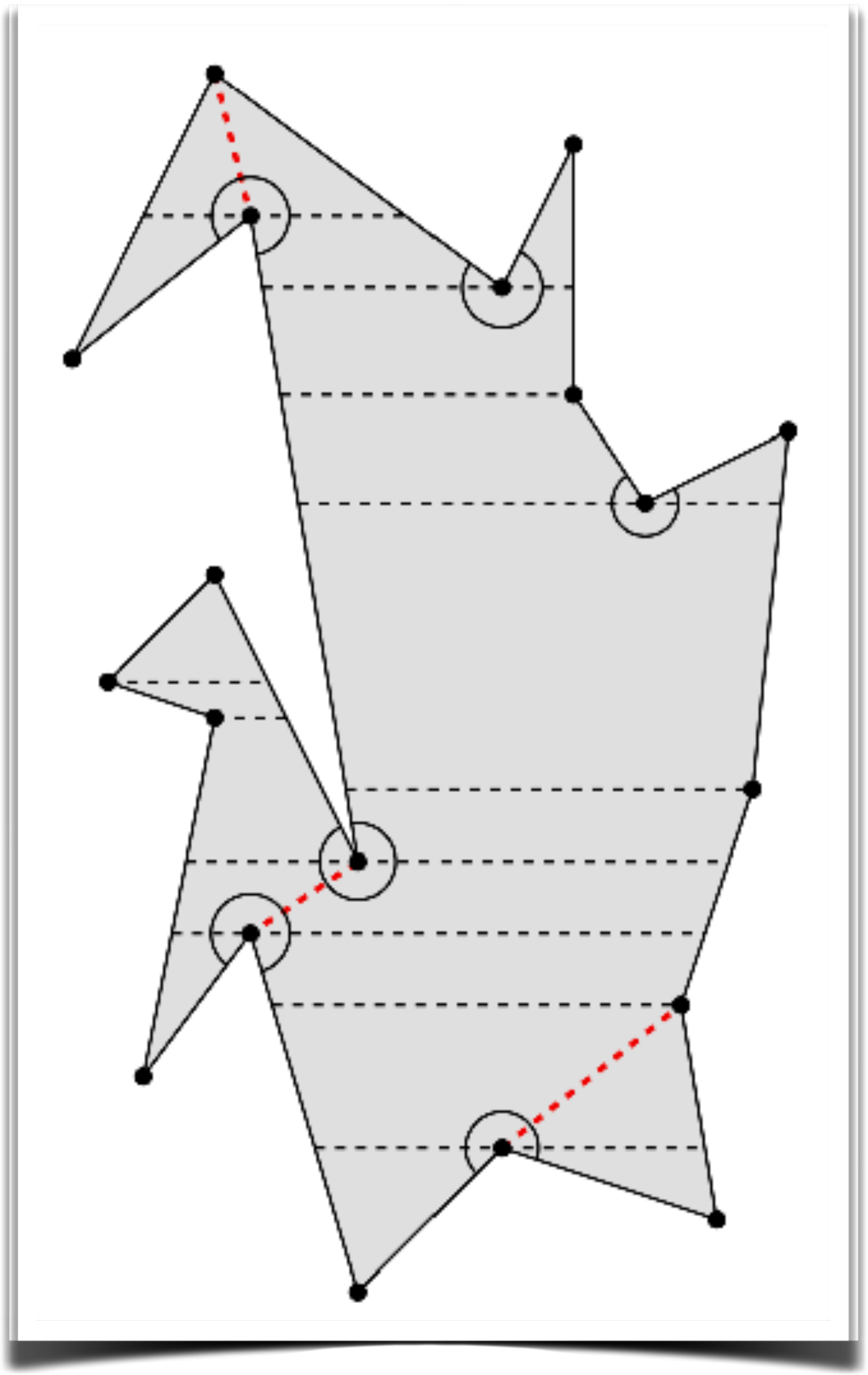


Technische Universität Braunschweig



Algorithm 5.18:

Partitioning into monotone subpolygons

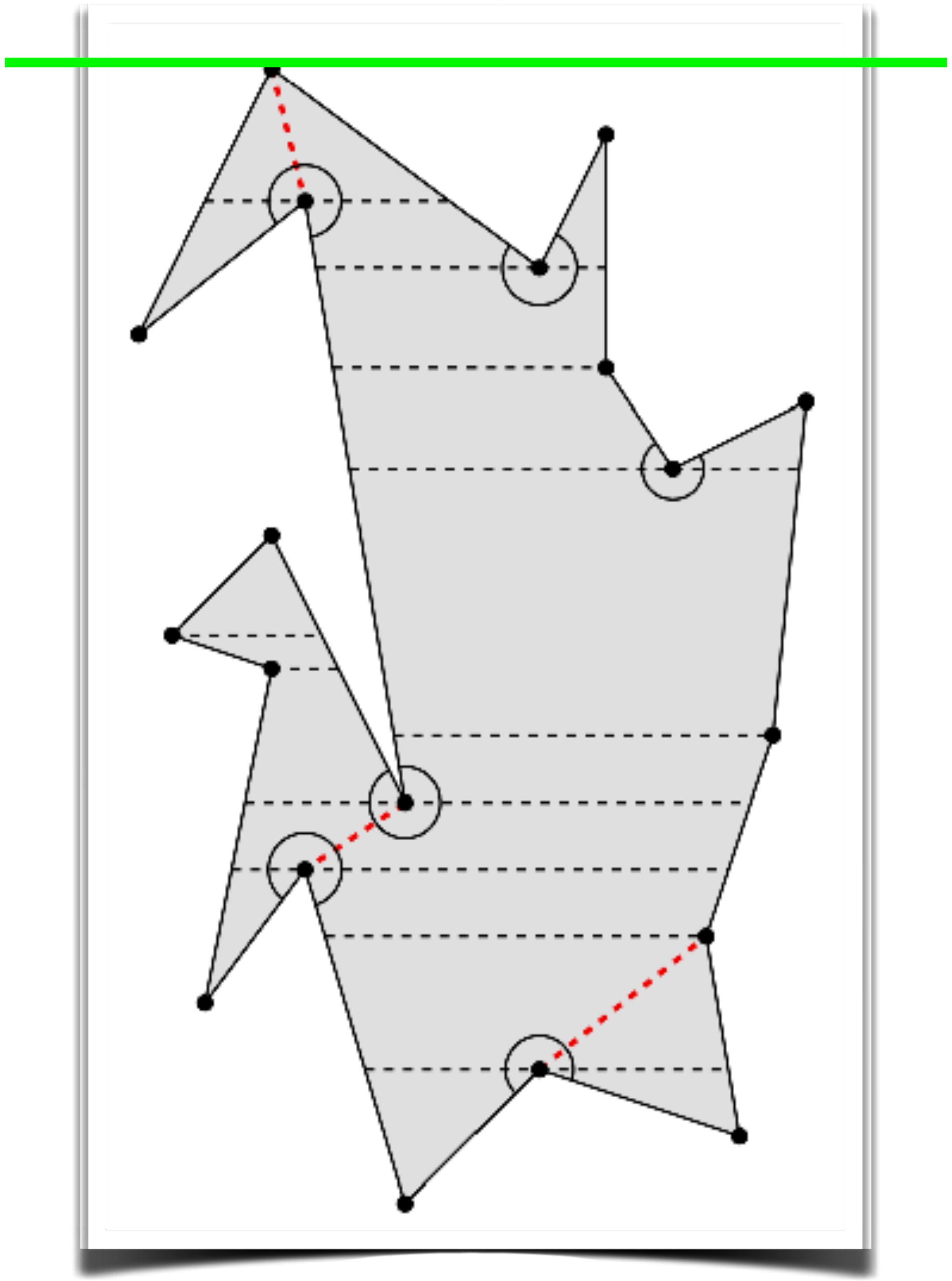


Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

Technische Universität Braunschweig

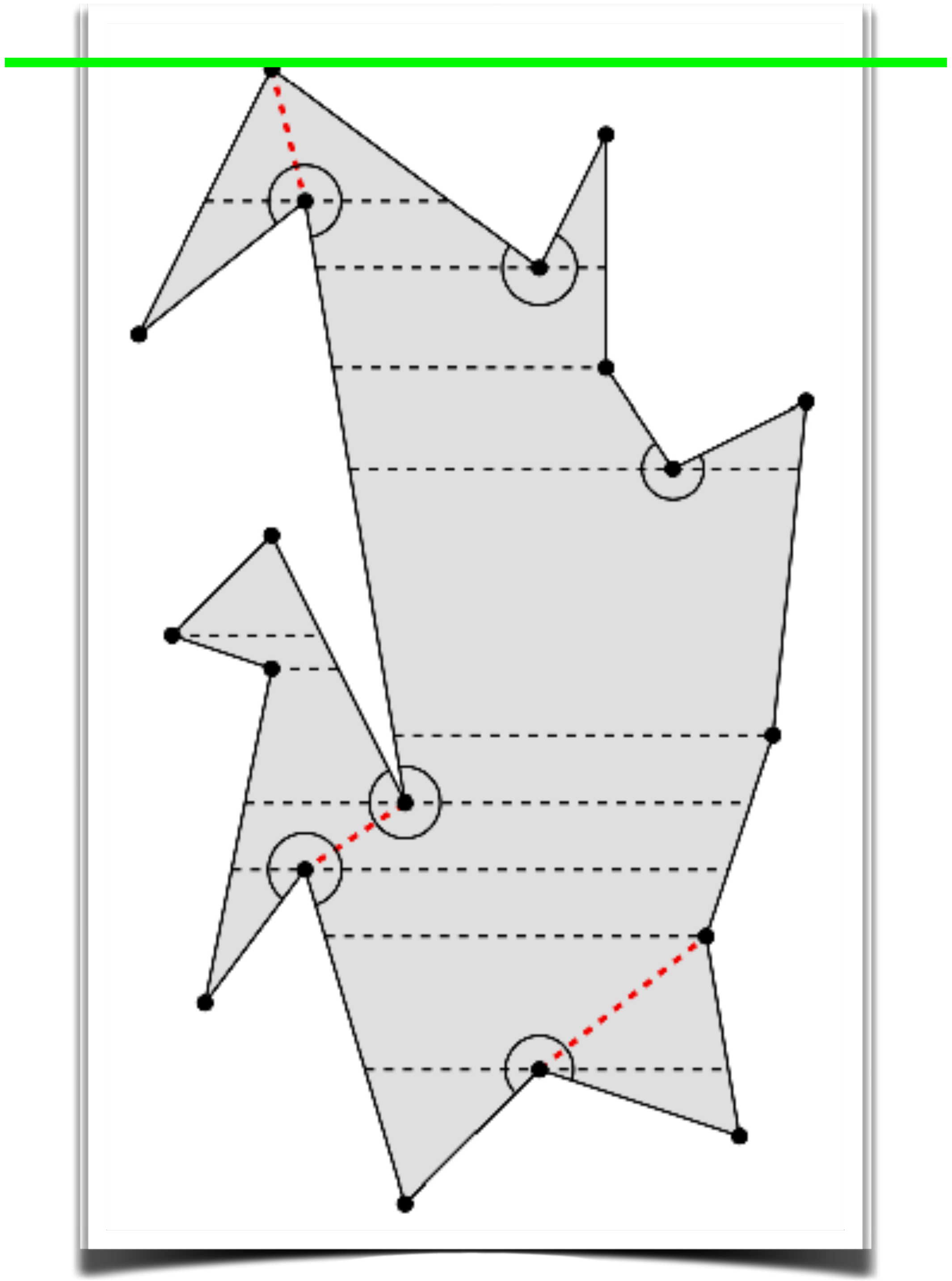


Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]



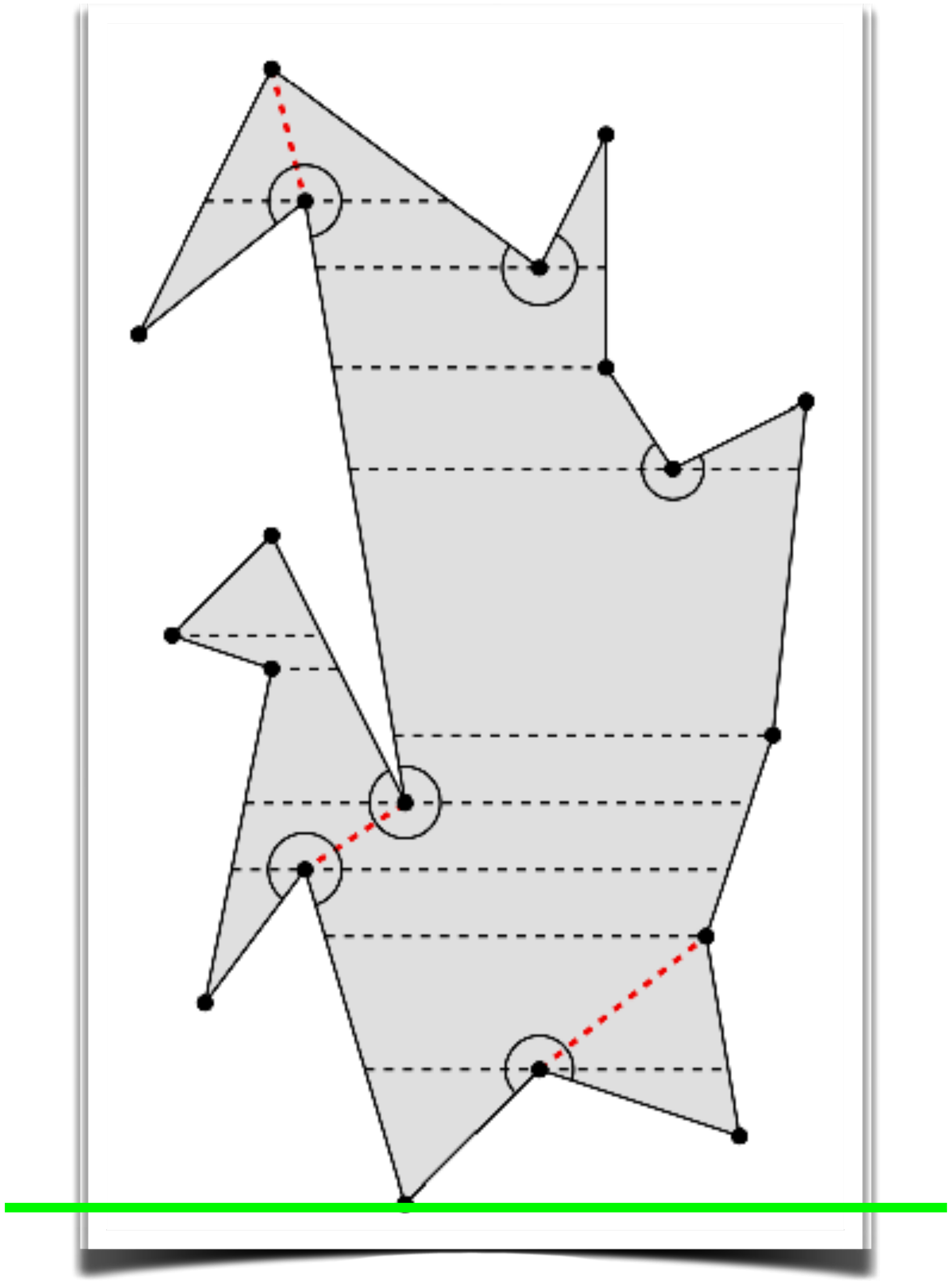
Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

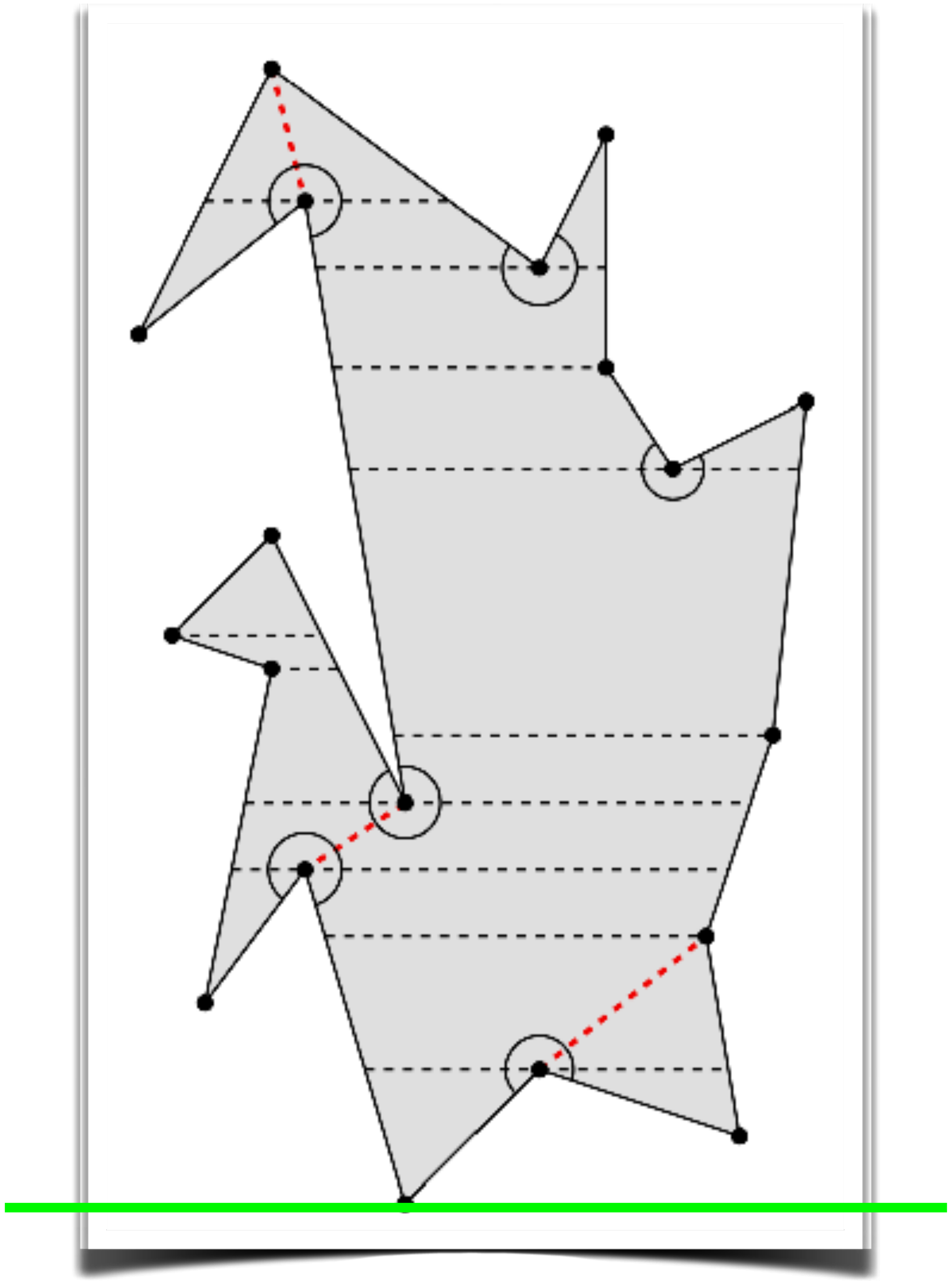


Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices



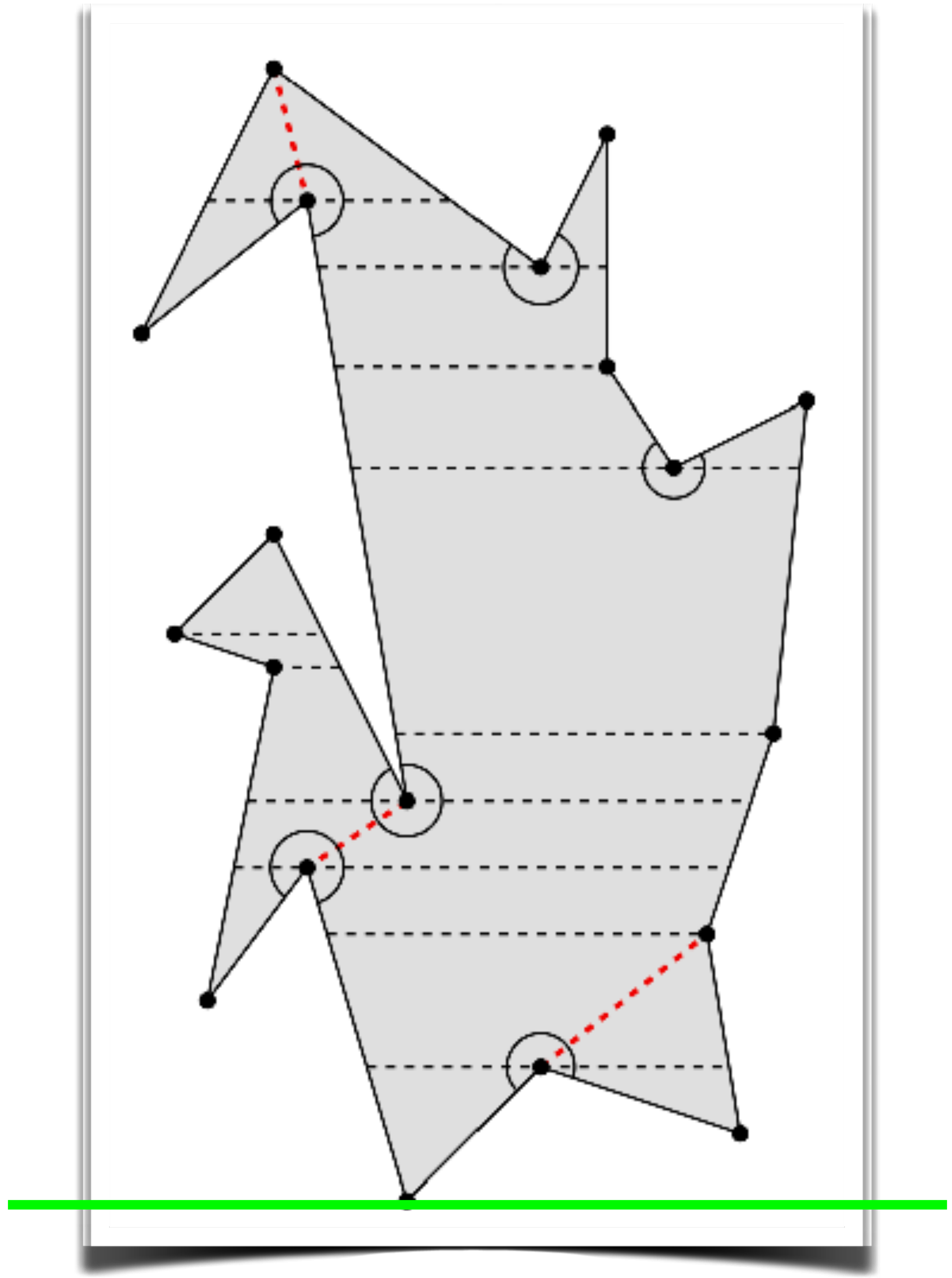
Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges

Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

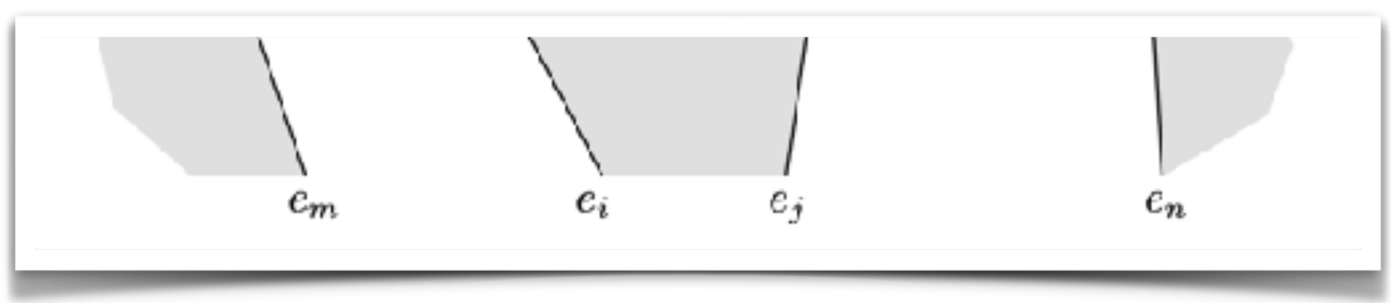


Algorithm 5.18:

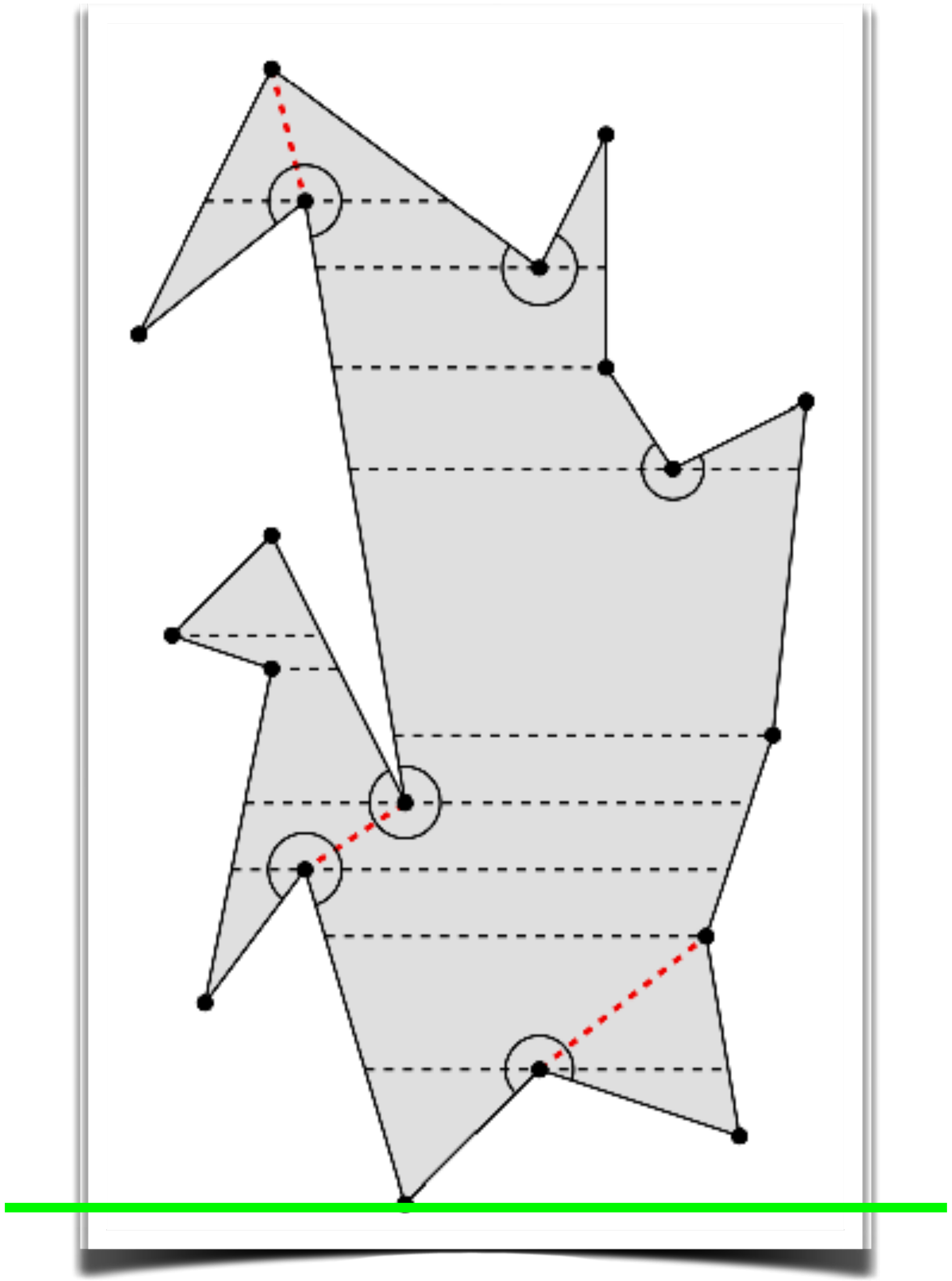
Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges



Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

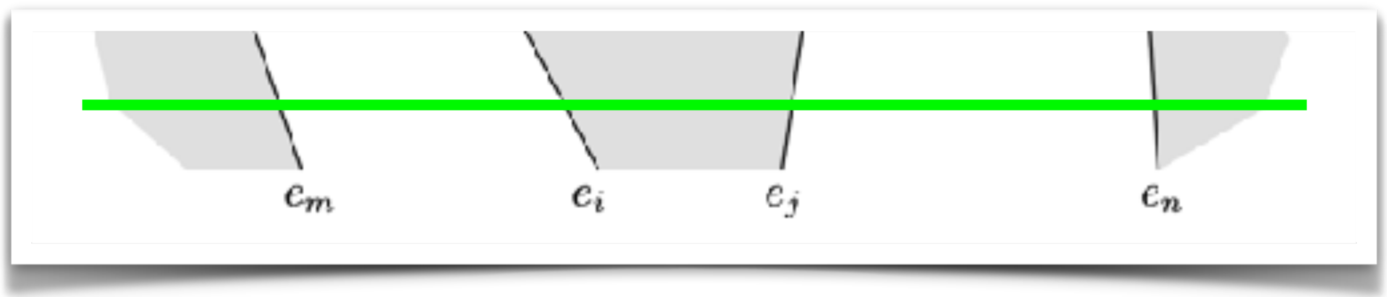


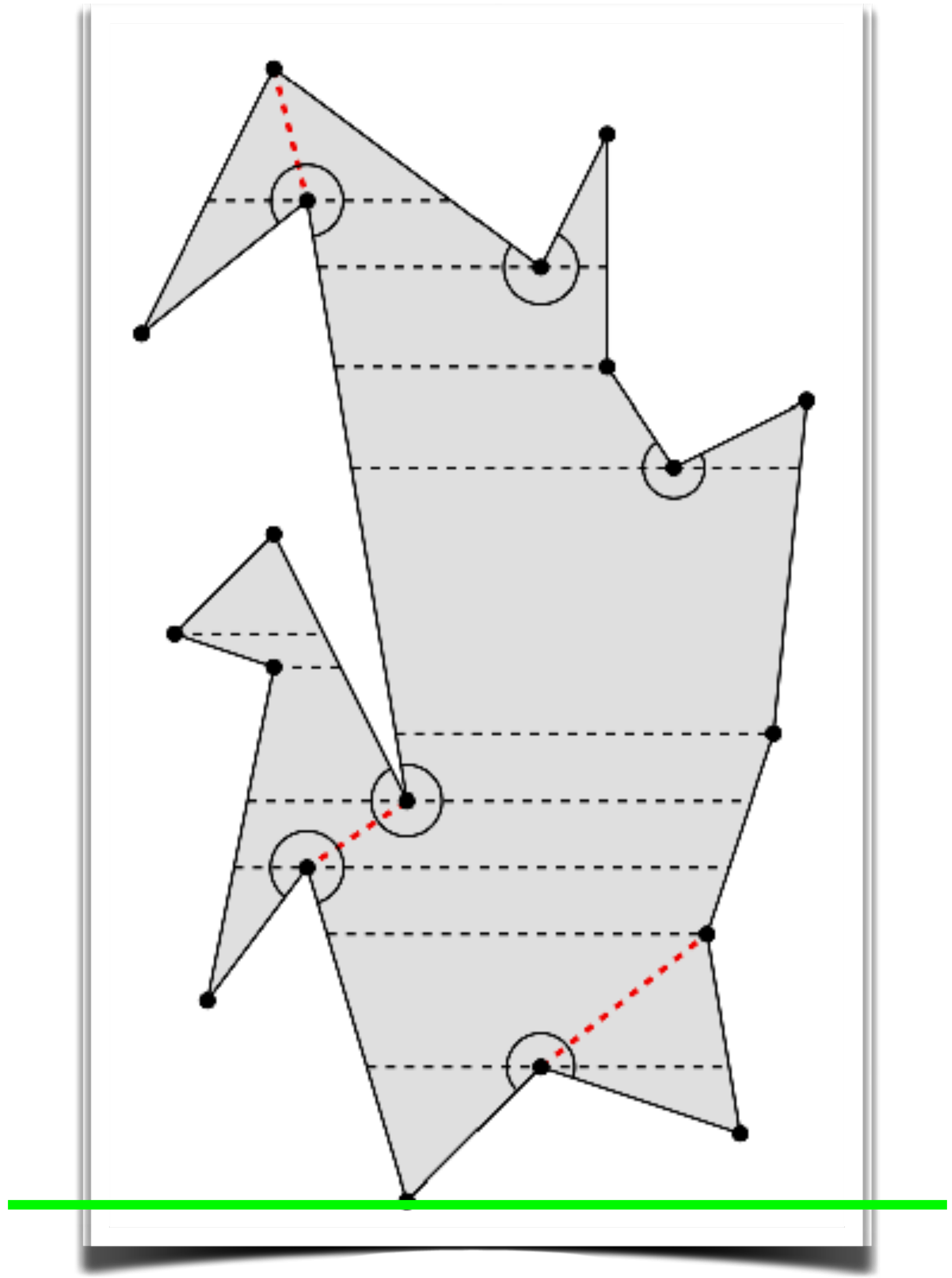
Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges



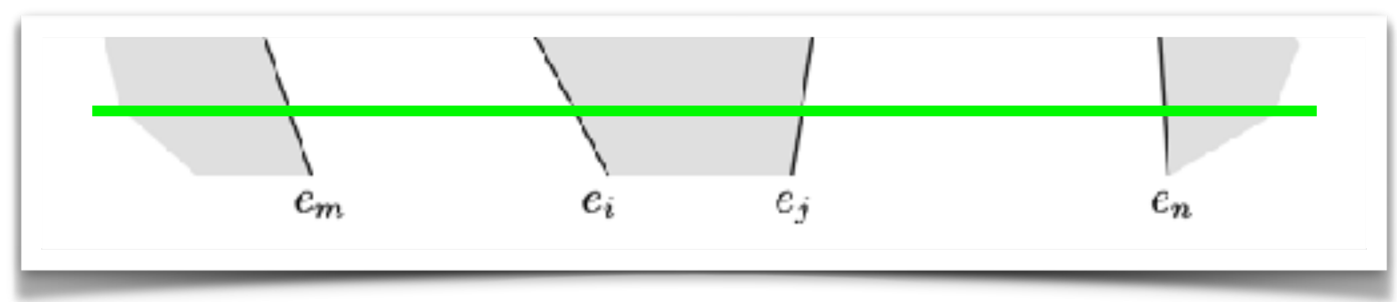


Algorithm 5.18:

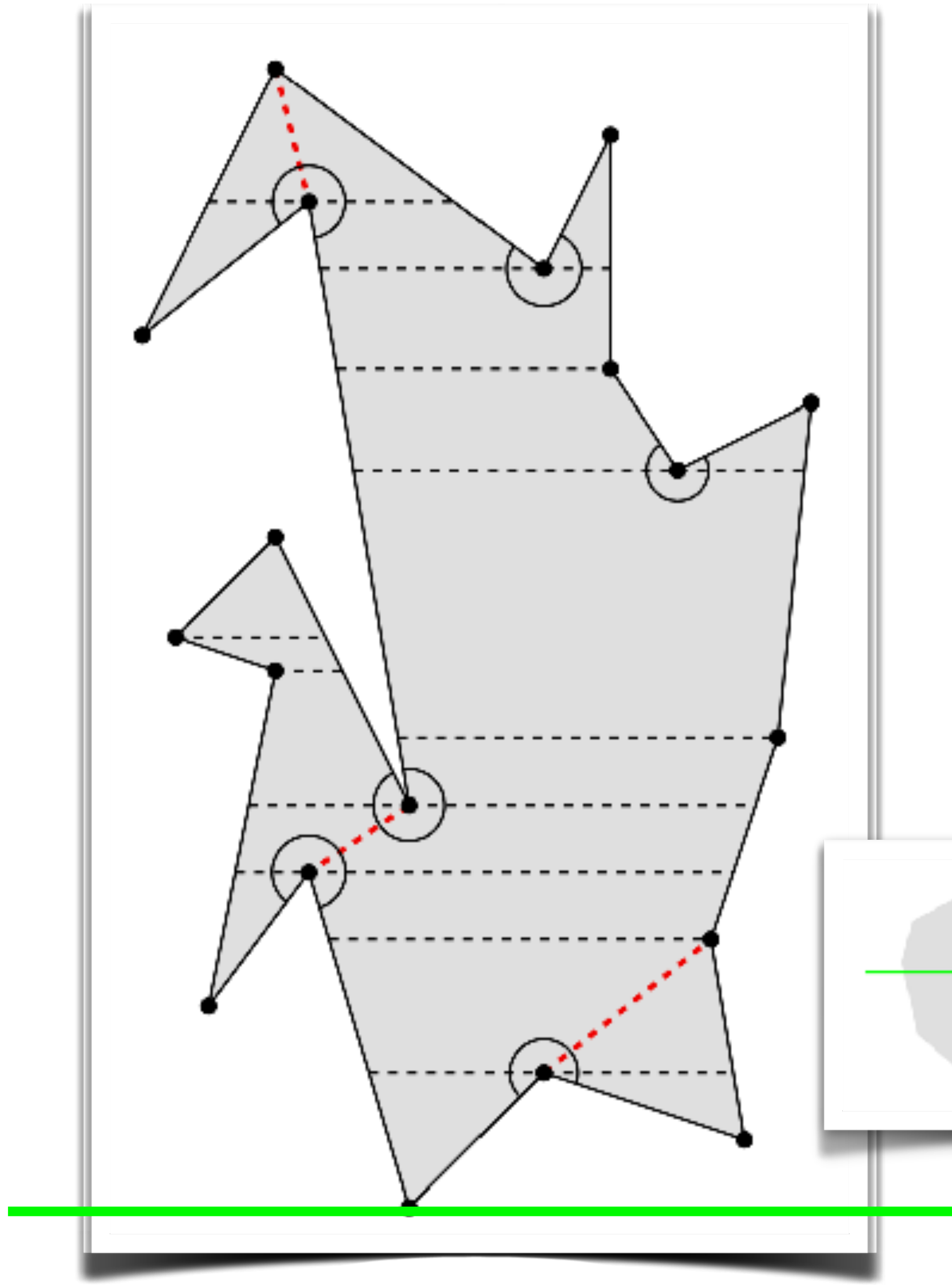
Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events



Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

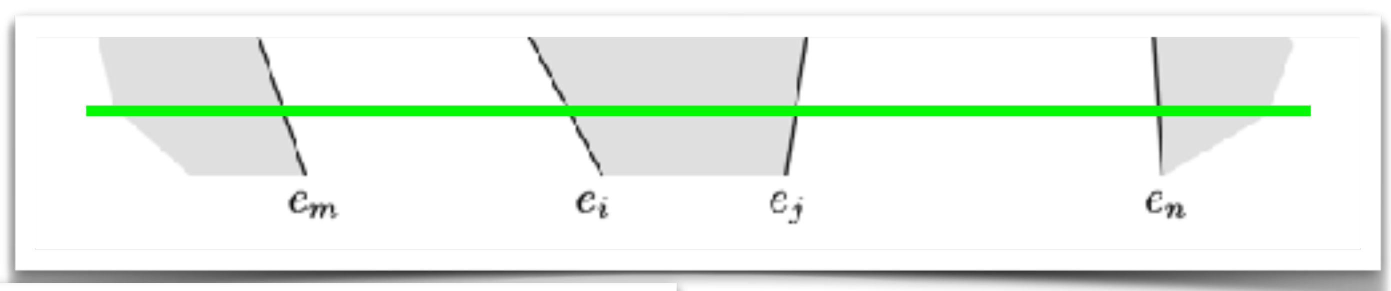


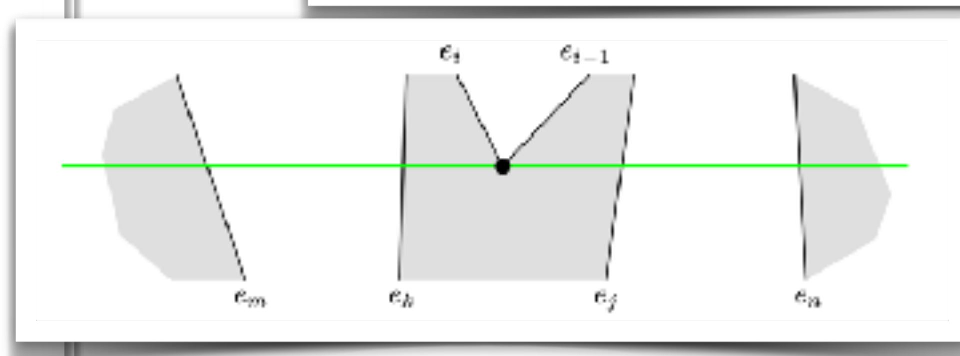
Algorithm 5.18:

Partitioning into monotone subpolygons

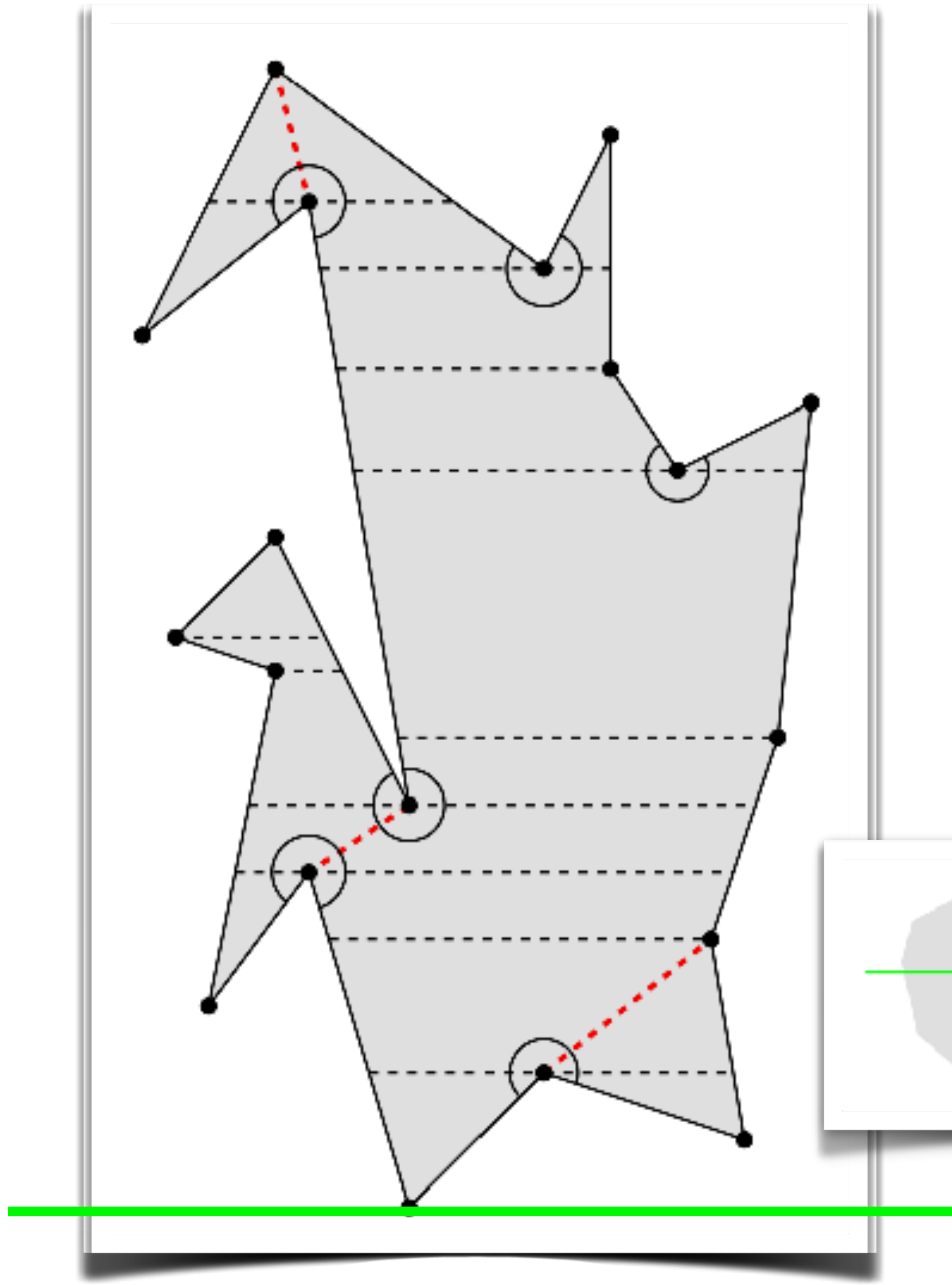
Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events



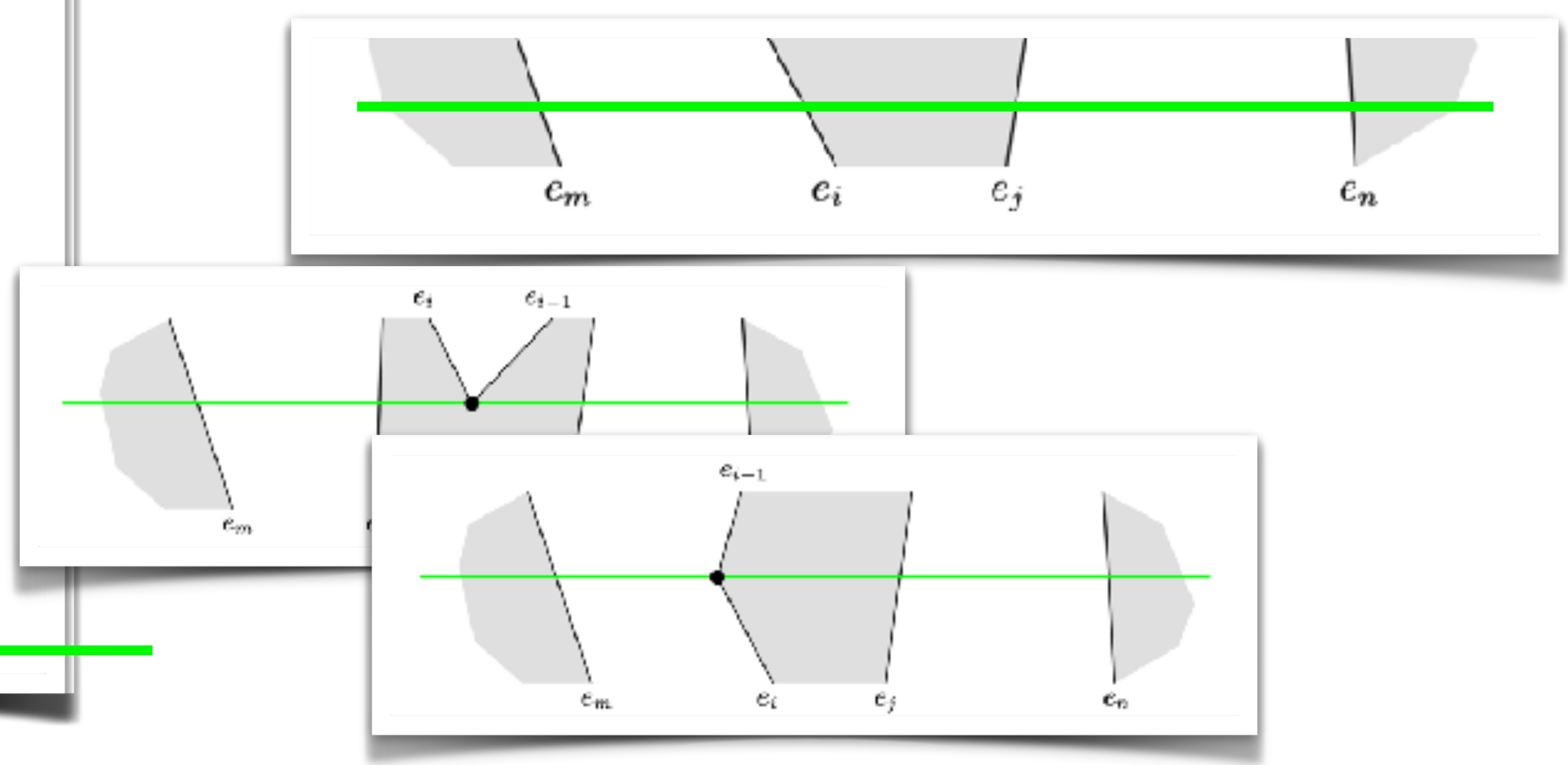


Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

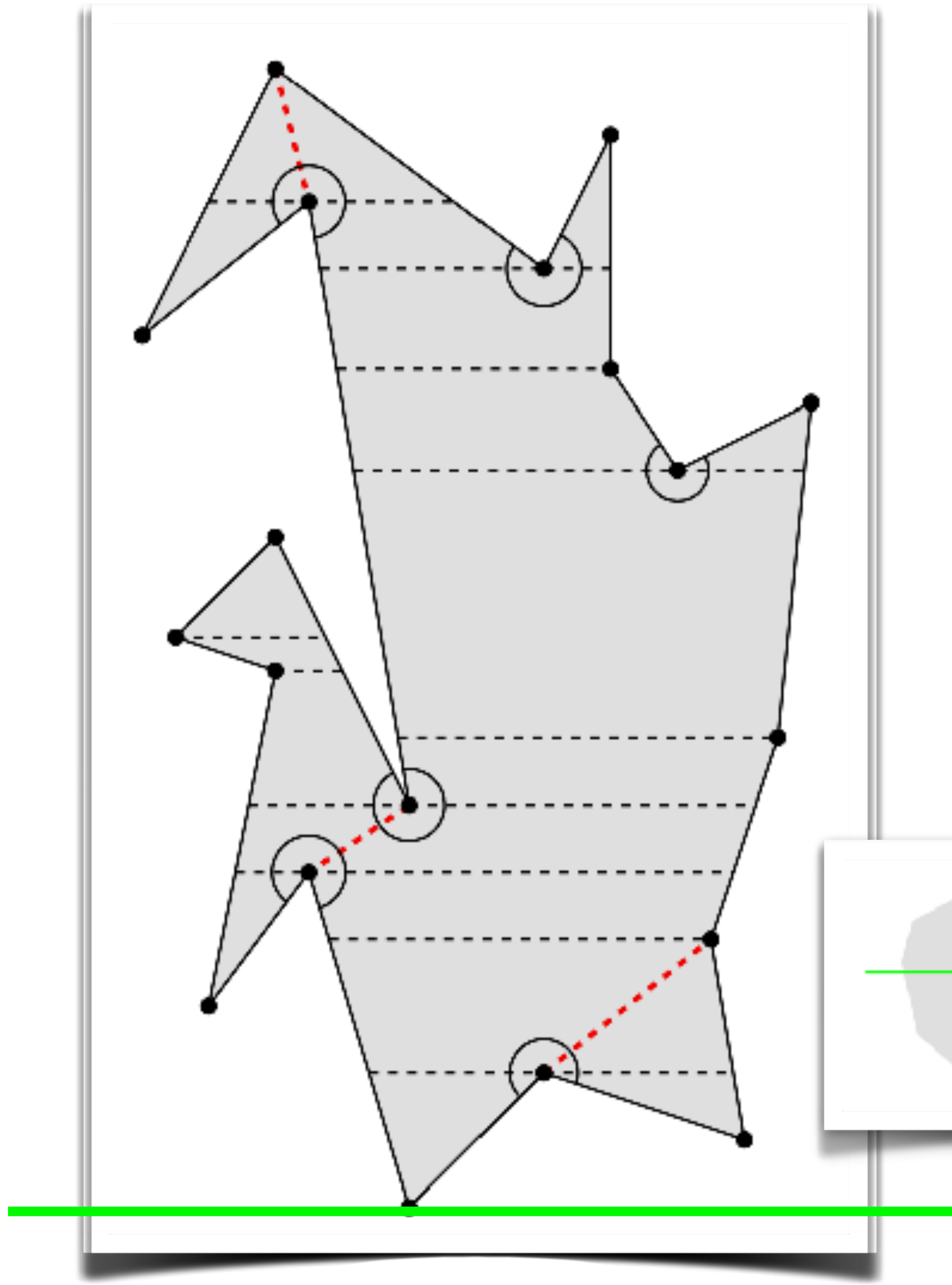


Algorithm 5.18:
Partitioning into monotone subpolygons

Ideas:
- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events



Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

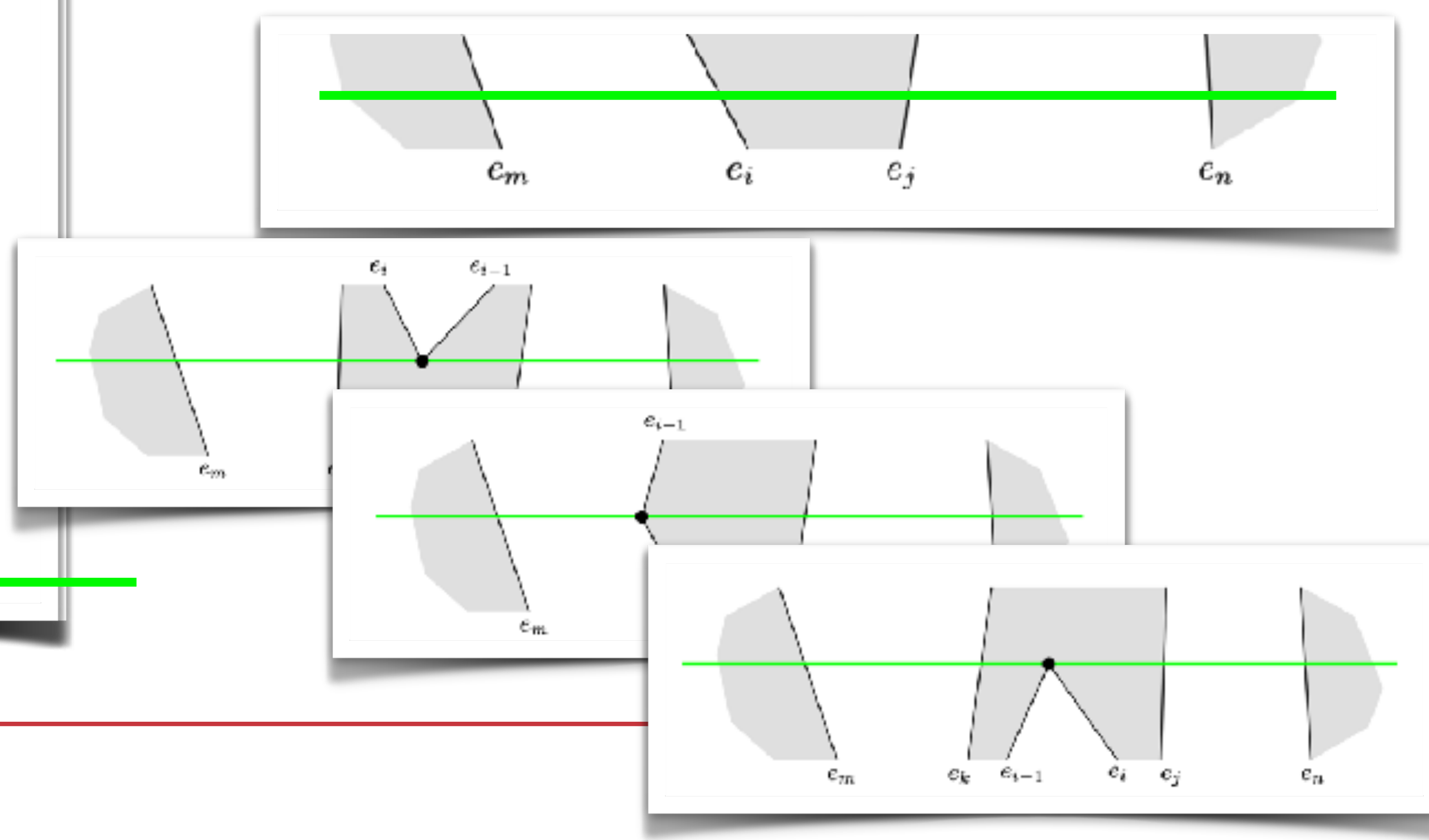


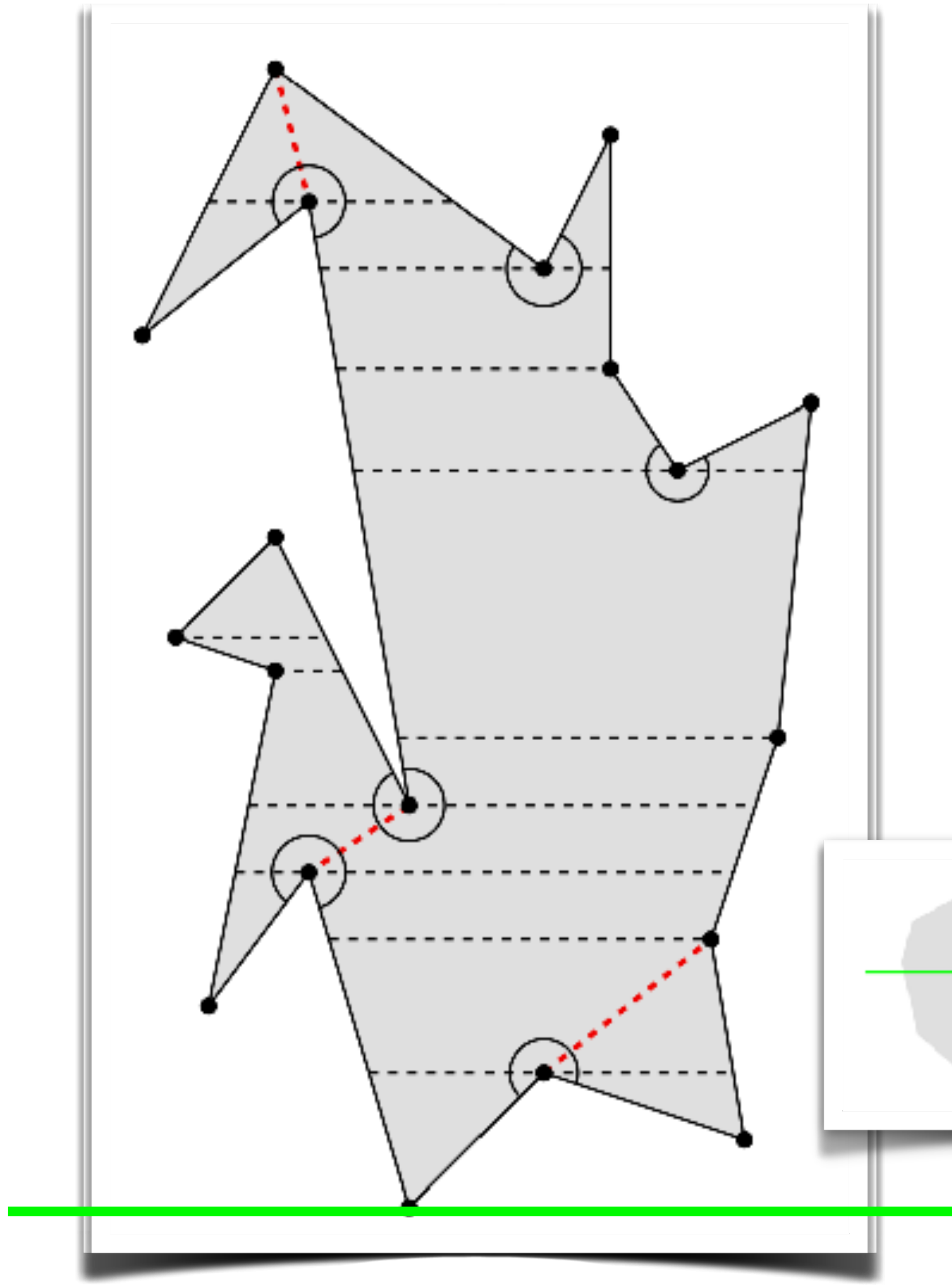
Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events



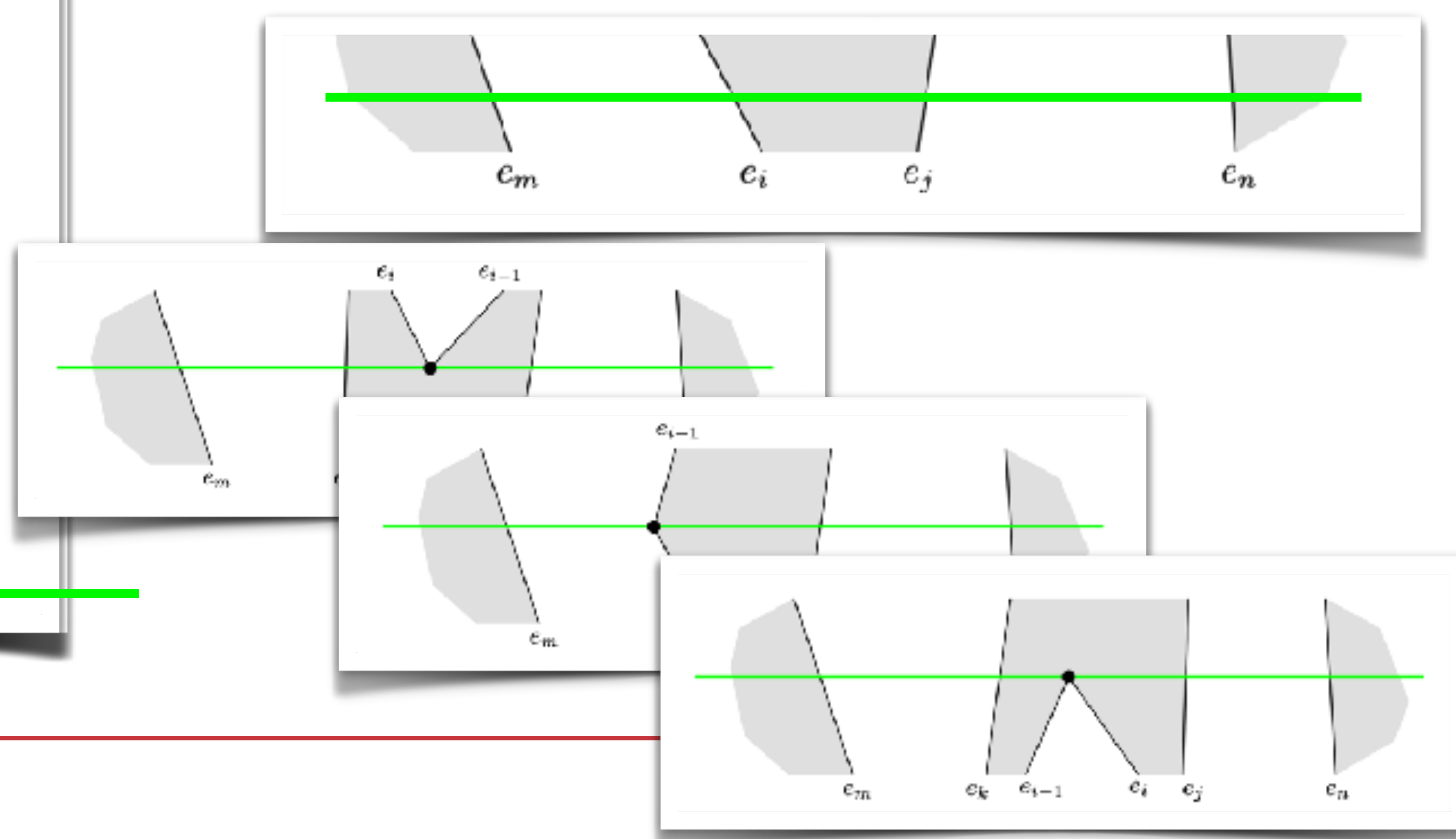


Algorithm 5.18:

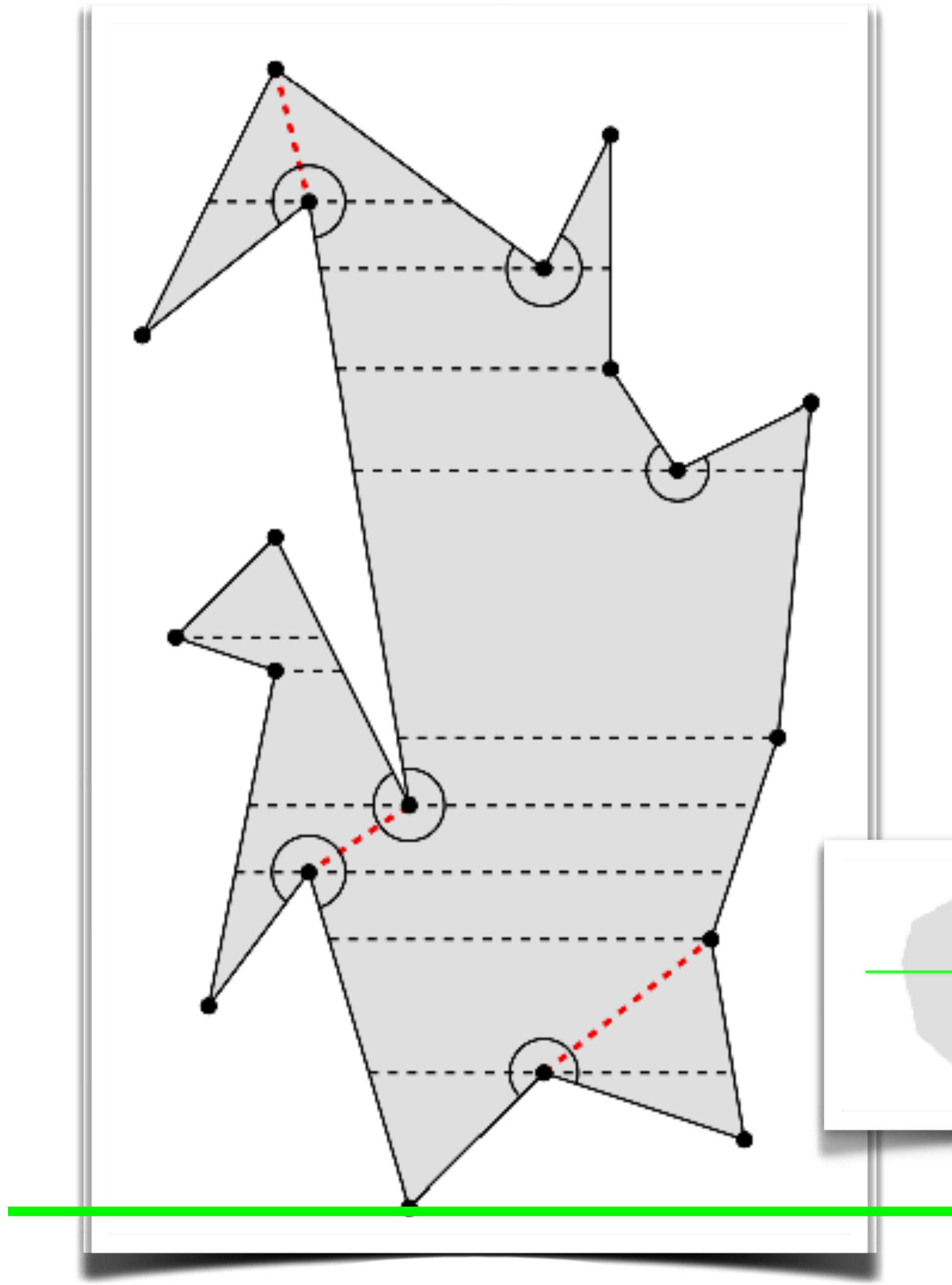
Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events
- Insert or delete in $O(\log n)$



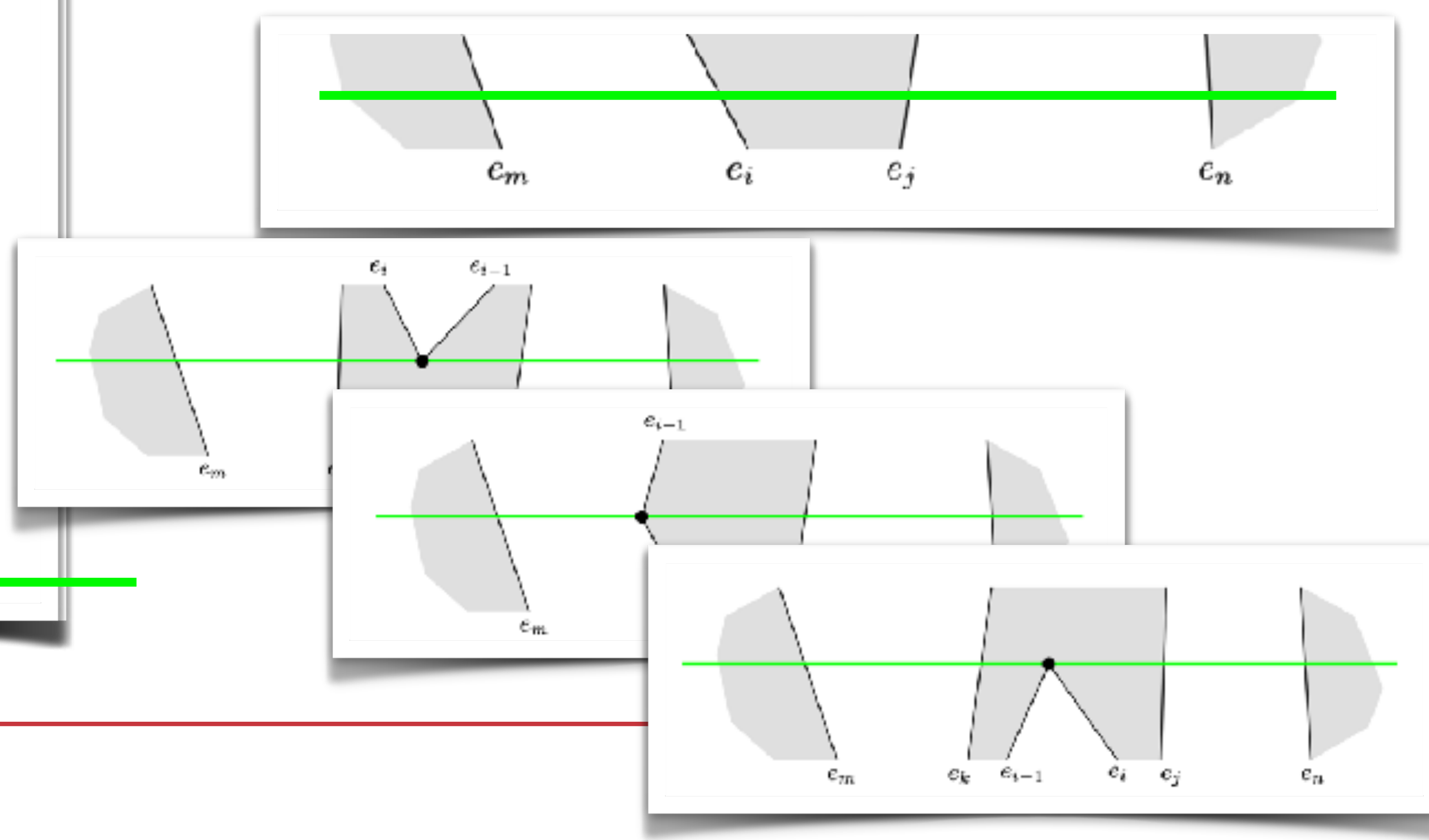
Partitioning into Monotone Polygons [Garey, Johnson, Preparata, Tarjan 1978]

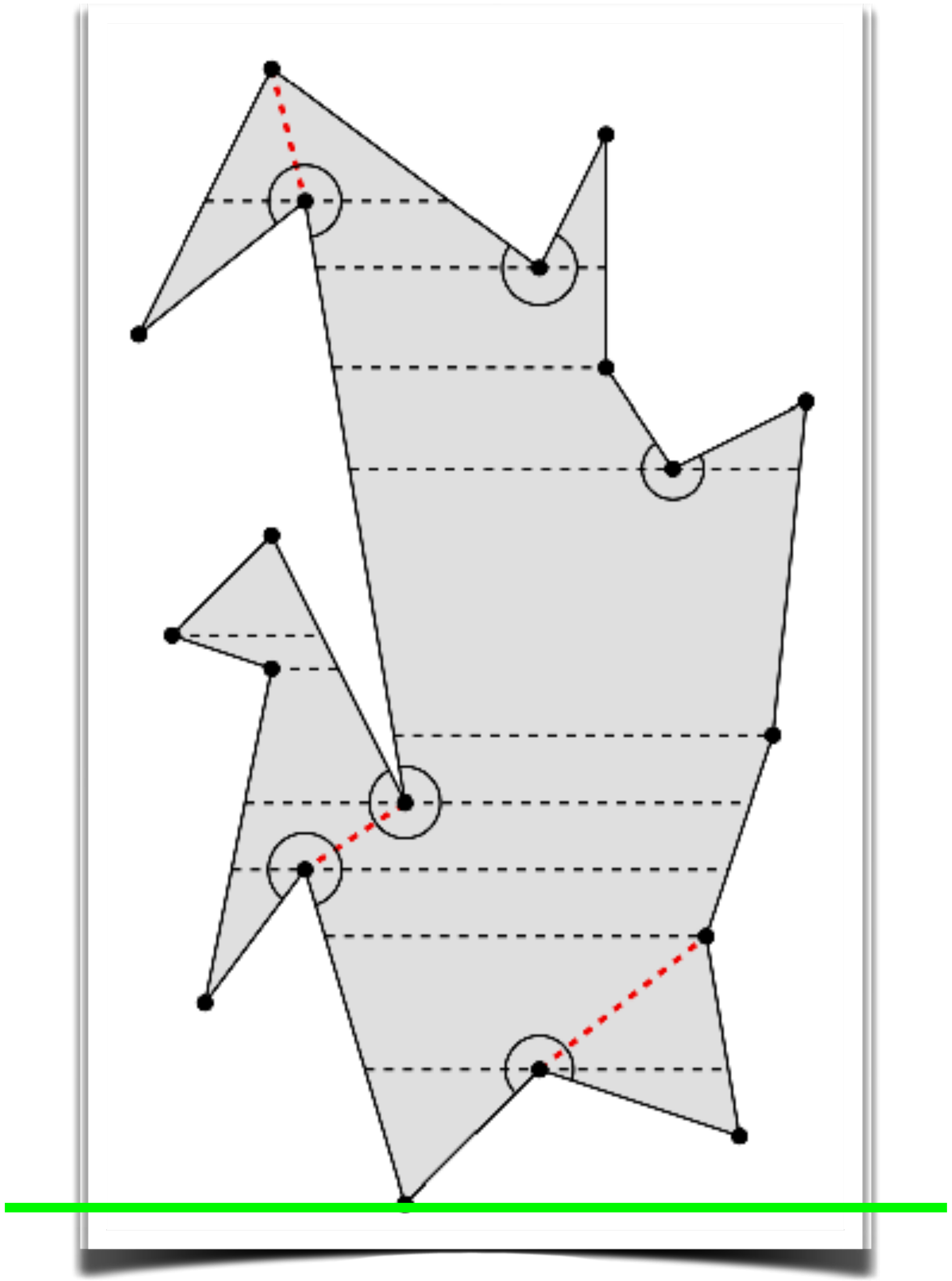


Algorithm 5.18:
Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events
- Insert or delete in $O(\log n)$
- Connect cusps up or down



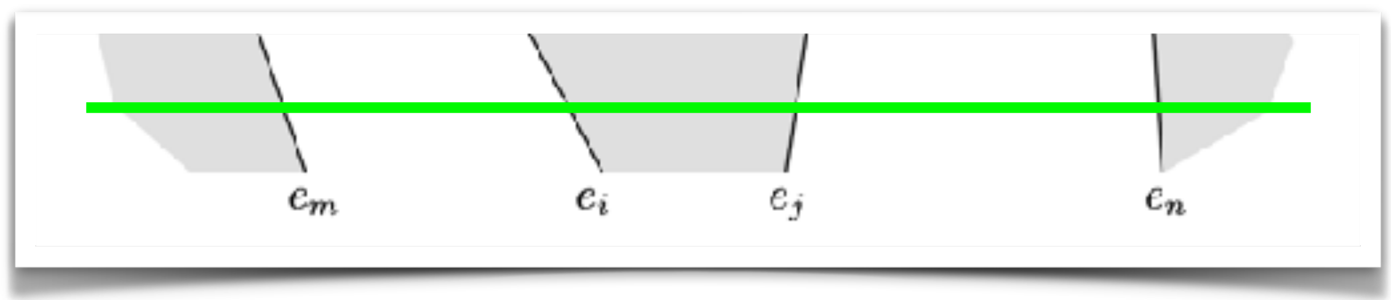


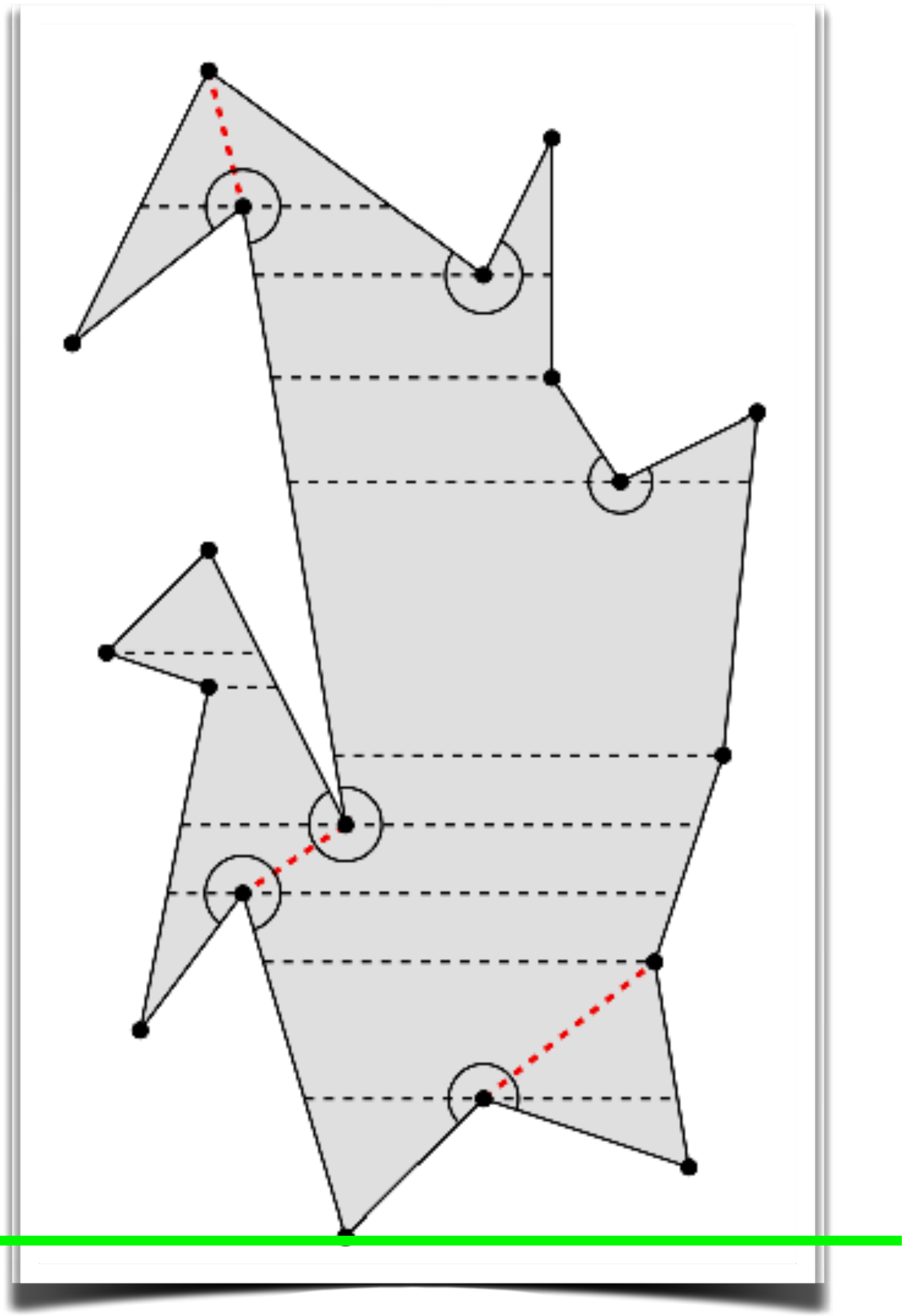
Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events
- Insert or delete in $O(\log n)$
- Connect cusps up or down



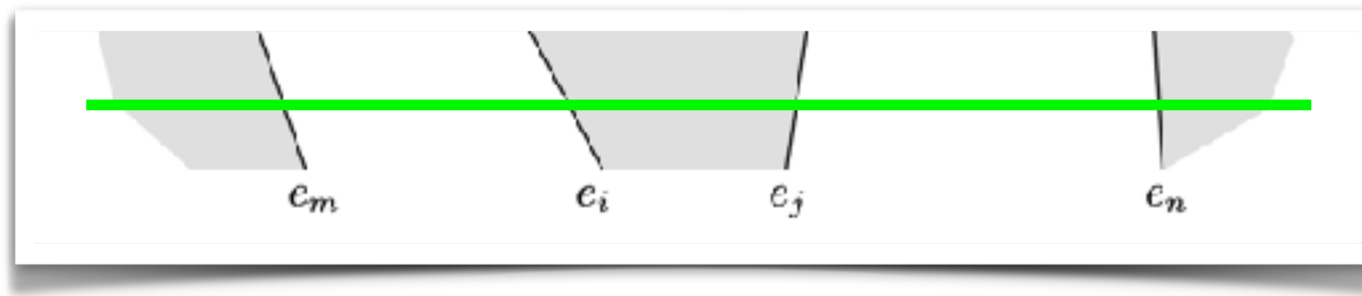


Algorithm 5.18:

Partitioning into monotone subpolygons

Ideas:

- Vertical plane sweep through vertices
- Maintain horizontal sequence of edges
- Perform updates at events
- Insert or delete in $O(\log n)$
- Connect cusps up or down



Theorem 5.19:

Triangulating a simple polygon can be done in $O(n \log n)$.

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation

Note:

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988):

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

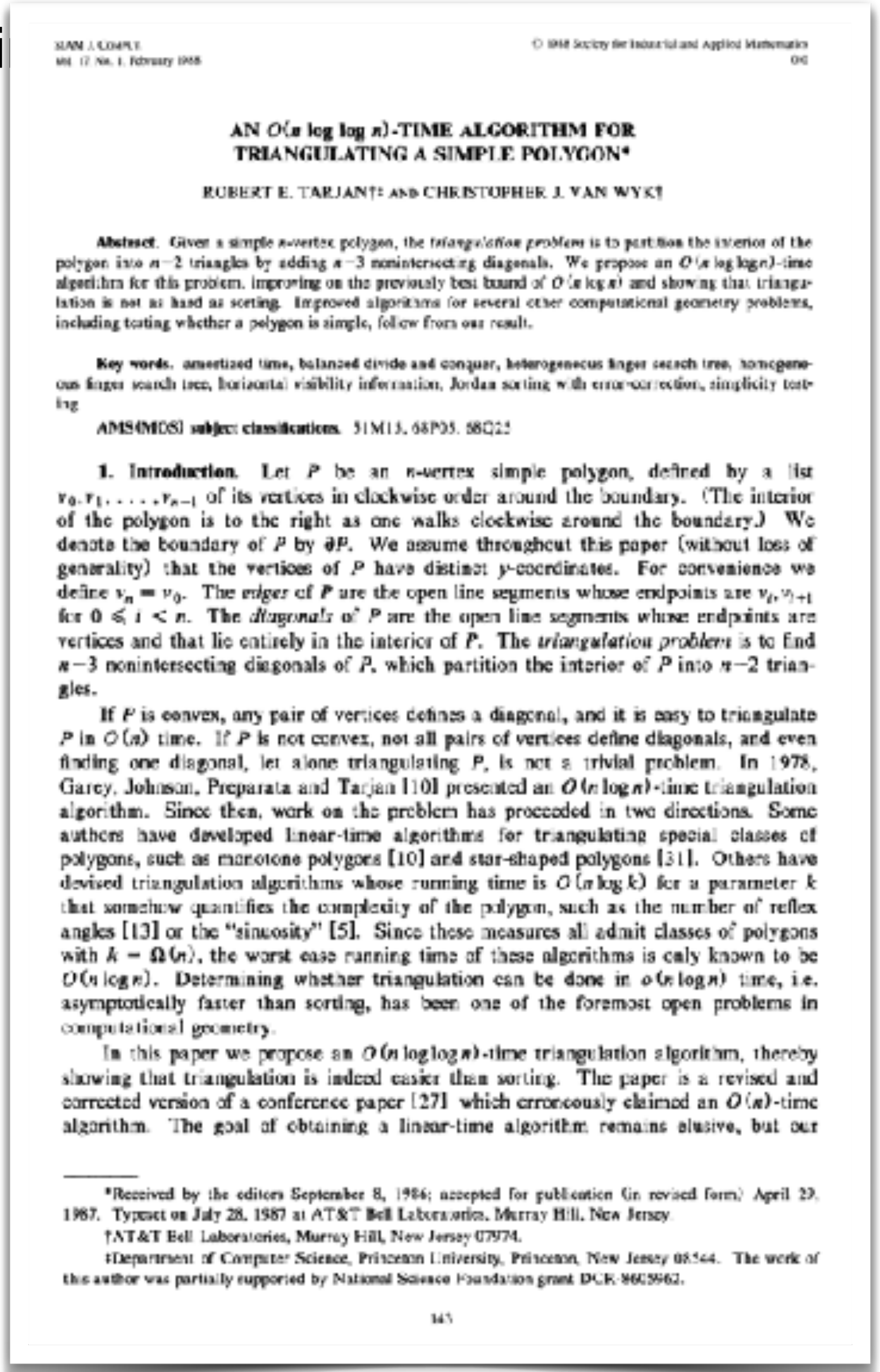
Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Note: Lower bound of $\Omega(n \log n)$ only applies to non-si

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$



SIAM J. COMPUT.
Vol. 17, No. 1, February 1988

AN $O(n \log \log n)$-TIME ALGORITHM FOR TRIANGULATING A SIMPLE POLYGON*

ROBERT E. TARJAN†‡ AND CHRISTOPHER J. VAN WYK†

Abstract. Given a simple n-vertex polygon, the *triangulation problem* is to partition the interior of the polygon into $n-2$ triangles by adding $n-3$ nonintersecting diagonals. We propose an $O(n \log \log n)$-time algorithm for this problem, improving on the previously best bound of $O(n \log n)$ and showing that triangulation is not as hard as sorting. Improved algorithms for several other computational geometry problems, including testing whether a polygon is simple, follow from our result.

Key words. amortized time, balanced divide and conquer, heterogeneous finger search tree, homogeneous finger search tree, horizontal visibility information, Jordan sorting with error-correction, simplicity testing

AMS(MOS) subject classifications. 51M15, 68P05, 68Q25

1. Introduction. Let P be an n-vertex simple polygon, defined by a list $v_0, v_1, \ldots, v_{n-1}$ of its vertices in clockwise order around the boundary. (The interior of the polygon is to the right as one walks clockwise around the boundary.) We denote the boundary of P by ∂P. We assume throughout this paper (without loss of generality) that the vertices of P have distinct y-coordinates. For convenience we define $v_n = v_0$. The *edges* of P are the open line segments whose endpoints are v_i, v_{i+1} for $0 \leq i < n$. The *diagonals* of P are the open line segments whose endpoints are vertices and that lie entirely in the interior of P. The *triangulation problem* is to find $n-3$ nonintersecting diagonals of P, which partition the interior of P into $n-2$ triangles.

If P is convex, any pair of vertices defines a diagonal, and it is easy to triangulate P in $O(n)$ time. If P is not convex, not all pairs of vertices define diagonals, and even finding one diagonal, let alone triangulating P, is not a trivial problem. In 1978, Garey, Johnson, Preparata and Tarjan [10] presented an $O(n \log n)$-time triangulation algorithm. Since then, work on the problem has proceeded in two directions. Some authors have developed linear-time algorithms for triangulating special classes of polygons, such as monotone polygons [10] and star-shaped polygons [31]. Others have devised triangulation algorithms whose running time is $O(n \log k)$ for a parameter k that somehow quantifies the complexity of the polygon, such as the number of reflex angles [13] or the "sinuosity" [5]. Since these measures all admit classes of polygons with $k = \Omega(n)$, the worst case running time of these algorithms is only known to be $O(n \log n)$. Determining whether triangulation can be done in $o(n \log n)$ time, i.e. asymptotically faster than sorting, has been one of the foremost open problems in computational geometry.

In this paper we propose an $O(n \log \log n)$-time triangulation algorithm, thereby showing that triangulation is indeed easier than sorting. The paper is a revised and corrected version of a conference paper [27] which erroneously claimed an $O(n)$-time algorithm. The goal of obtaining a linear-time algorithm remains elusive, but our

*Received by the editors September 8, 1986; accepted for publication (in revised form) April 29, 1987. Typeset on July 28, 1987 at AT&T Bell Laboratories, Murray Hill, New Jersey.

†AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

‡Department of Computer Science, Princeton University, Princeton, New Jersey 08544. The work of this author was partially supported by National Science Foundation grant DCR-8605962.

Technische Universität Braunschweig

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992):

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Note: Lower bound of $\Omega(n \log n)$ only applies to non-si

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Discrete Comput Geom 7: 329–346 (1992)

Discrete & Computational Geometry

Polygon Triangulation in $O(n \log \log n)$ Time with Simple Data Structures*

David G. Kirkpatrick,[1] Maria M. Klawe,[1] and Robert E. Tarjan[2]

[1] Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z2

[2] Department of Computer Science, Princeton University, Princeton, NJ 08540, USA, and NEC Research Institute

Abstract. We give a new $O(n \log \log n)$-time deterministic algorithm for triangulating simple n-vertex polygons, which avoids the use of complicated data structures. In addition, for polygons whose vertices have integer coordinates of polynomially bounded size, the algorithm can be modified to run in $O(n \log^* n)$ time. The major new techniques employed are the efficient location of horizontal visibility edges that partition the interior of the polygon into regions of approximately equal size, and a linear-time algorithm for obtaining the horizontal visibility partition of a subchain of a polygonal chain, from the horizontal visibility partition of the entire chain. The latter technique has other interesting applications, including a linear-time algorithm to convert a Steiner triangulation of a polygon into a true triangulation.

1. Introduction

Let P be a simple polygon with n vertices. The *diagonals* of P are the open line segments whose endpoints are vertices of the polygon and that lie entirely in the interior of P. A *triangulation* of P is a partition of its interior into $n - 2$ triangles by adding $n - 3$ nonintersecting diagonals. The problem of triangulating a simple polygon, that is, determining the set of nonintersecting diagonals, has attracted considerable attention in computational geometry literature and elsewhere

* This research was partially supported by the following grants: NSERC 583584, NSERC 580485, ONR-N00014-87-0467, and by DIMACS, an NSF Science and Technology Center (NSF-STC88-09648).

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989):

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Technische Universität Braunschweig

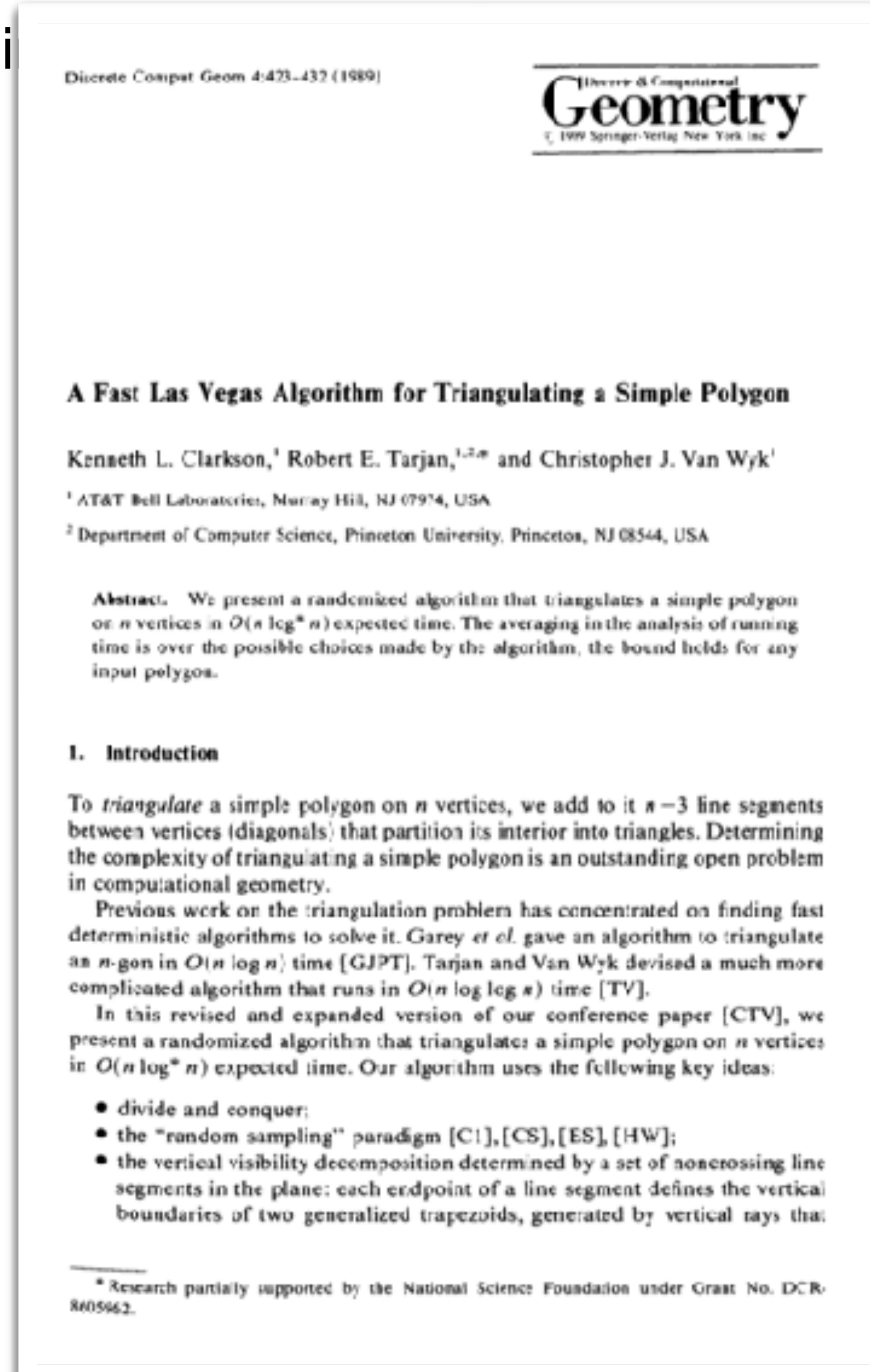
Note: Lower bound of $\Omega(n \log n)$ only applies to non-si

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$


Discrete Comput Geom 4:423–432 (1989)

Discrete & Computational Geometry

A Fast Las Vegas Algorithm for Triangulating a Simple Polygon

Kenneth L. Clarkson,[1] Robert E. Tarjan,[1,2,*] and Christopher J. Van Wyk[1]

[1] AT&T Bell Laboratories, Murray Hill, NJ 07974, USA

[2] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

Abstract. We present a randomized algorithm that triangulates a simple polygon on n vertices in $O(n \log^* n)$ expected time. The averaging in the analysis of running time is over the possible choices made by the algorithm; the bound holds for any input polygon.

1. Introduction

To *triangulate* a simple polygon on n vertices, we add to it $n-3$ line segments between vertices (diagonals) that partition its interior into triangles. Determining the complexity of triangulating a simple polygon is an outstanding open problem in computational geometry.

Previous work on the triangulation problem has concentrated on finding fast deterministic algorithms to solve it. Garey *et al.* gave an algorithm to triangulate an n-gon in $O(n \log n)$ time [GJPT]. Tarjan and Van Wyk devised a much more complicated algorithm that runs in $O(n \log \log n)$ time [TV].

In this revised and expanded version of our conference paper [CTV], we present a randomized algorithm that triangulates a simple polygon on n vertices in $O(n \log^* n)$ expected time. Our algorithm uses the following key ideas:

- divide and conquer;
- the "random sampling" paradigm [C1], [CS], [ES], [HW];
- the vertical visibility decomposition determined by a set of noncrossing line segments in the plane: each endpoint of a line segment defines the vertical boundaries of two generalized trapezoids, generated by vertical rays that

* Research partially supported by the National Science Foundation under Grant No. DCR-8605962.

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Technische Universität Braunschweig

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Chazelle (1991):

Technische Universität Braunschweig

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Chazelle (1991): $O(n)$

Note: Lower bound of $\Omega(n \log n)$ only applies to non-si

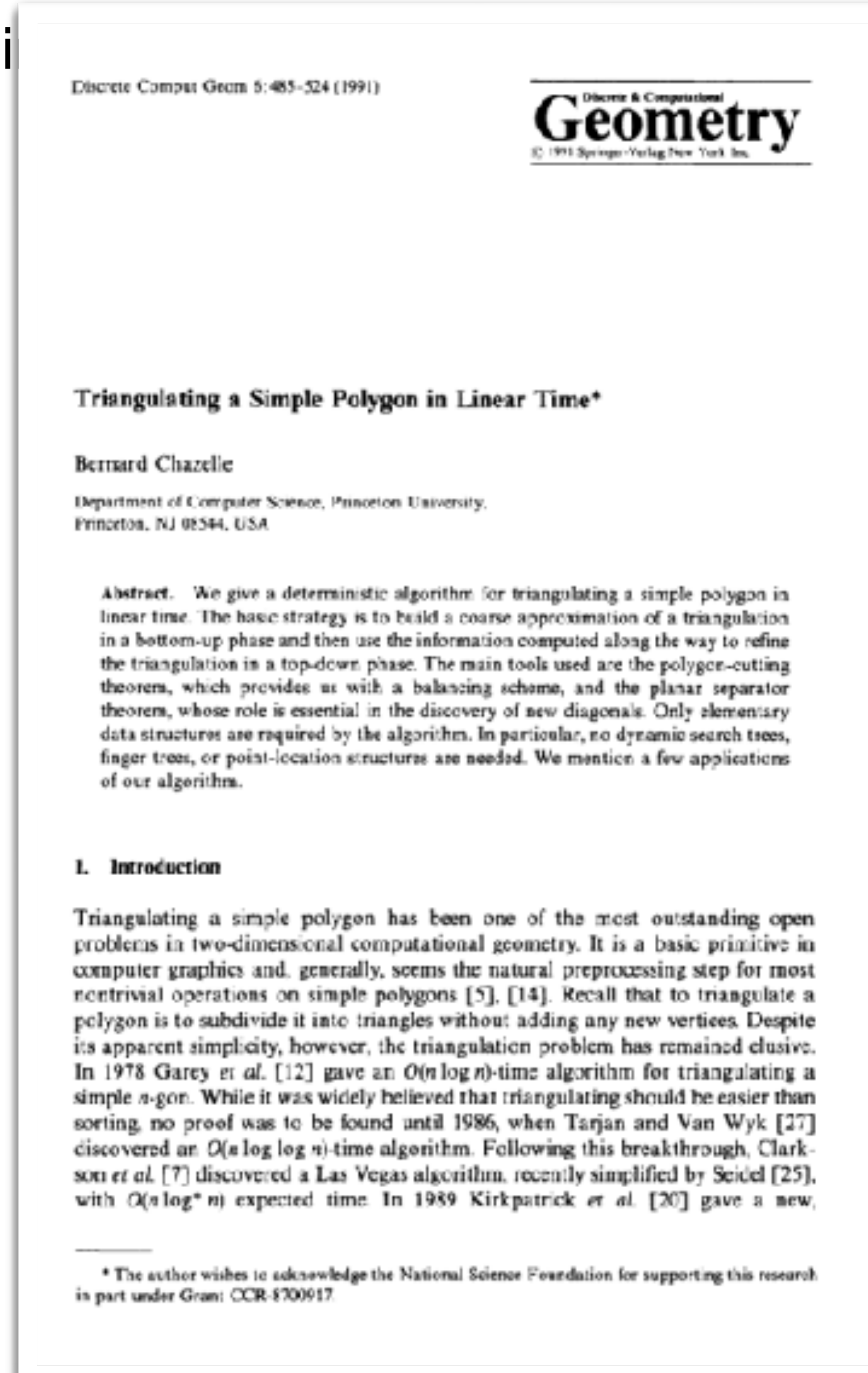
Progress:

Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Chazelle (1991): $O(n)$


Discrete Comput Geom 6:485–524 (1991)

Discrete & Computational Geometry

Triangulating a Simple Polygon in Linear Time*

Bernard Chazelle

Department of Computer Science, Princeton University,
Princeton, NJ 08544, USA

Abstract. We give a deterministic algorithm for triangulating a simple polygon in linear time. The basic strategy is to build a coarse approximation of a triangulation in a bottom-up phase and then use the information computed along the way to refine the triangulation in a top-down phase. The main tools used are the polygon-cutting theorem, which provides us with a balancing scheme, and the planar separator theorem, whose role is essential in the discovery of new diagonals. Only elementary data structures are required by the algorithm. In particular, no dynamic search trees, finger trees, or point-location structures are needed. We mention a few applications of our algorithm.

1. Introduction

Triangulating a simple polygon has been one of the most outstanding open problems in two-dimensional computational geometry. It is a basic primitive in computer graphics and, generally, seems the natural preprocessing step for most nontrivial operations on simple polygons [5], [14]. Recall that to triangulate a polygon is to subdivide it into triangles without adding any new vertices. Despite its apparent simplicity, however, the triangulation problem has remained elusive. In 1978 Garey *et al.* [12] gave an $O(n \log n)$-time algorithm for triangulating a simple n-gon. While it was widely believed that triangulating should be easier than sorting, no proof was to be found until 1986, when Tarjan and Van Wyk [27] discovered an $O(n \log \log n)$-time algorithm. Following this breakthrough, Clarkson *et al.* [7] discovered a Las Vegas algorithm, recently simplified by Seidel [25], with $O(n \log^* n)$ expected time. In 1989 Kirkpatrick *et al.* [20] gave a new,

* The author wishes to acknowledge the National Science Foundation for supporting this research in part under Grant CCR-8700917.

Note: Lower bound of $\Omega(n \log n)$ only applies to non-simple polygons.

Progress:

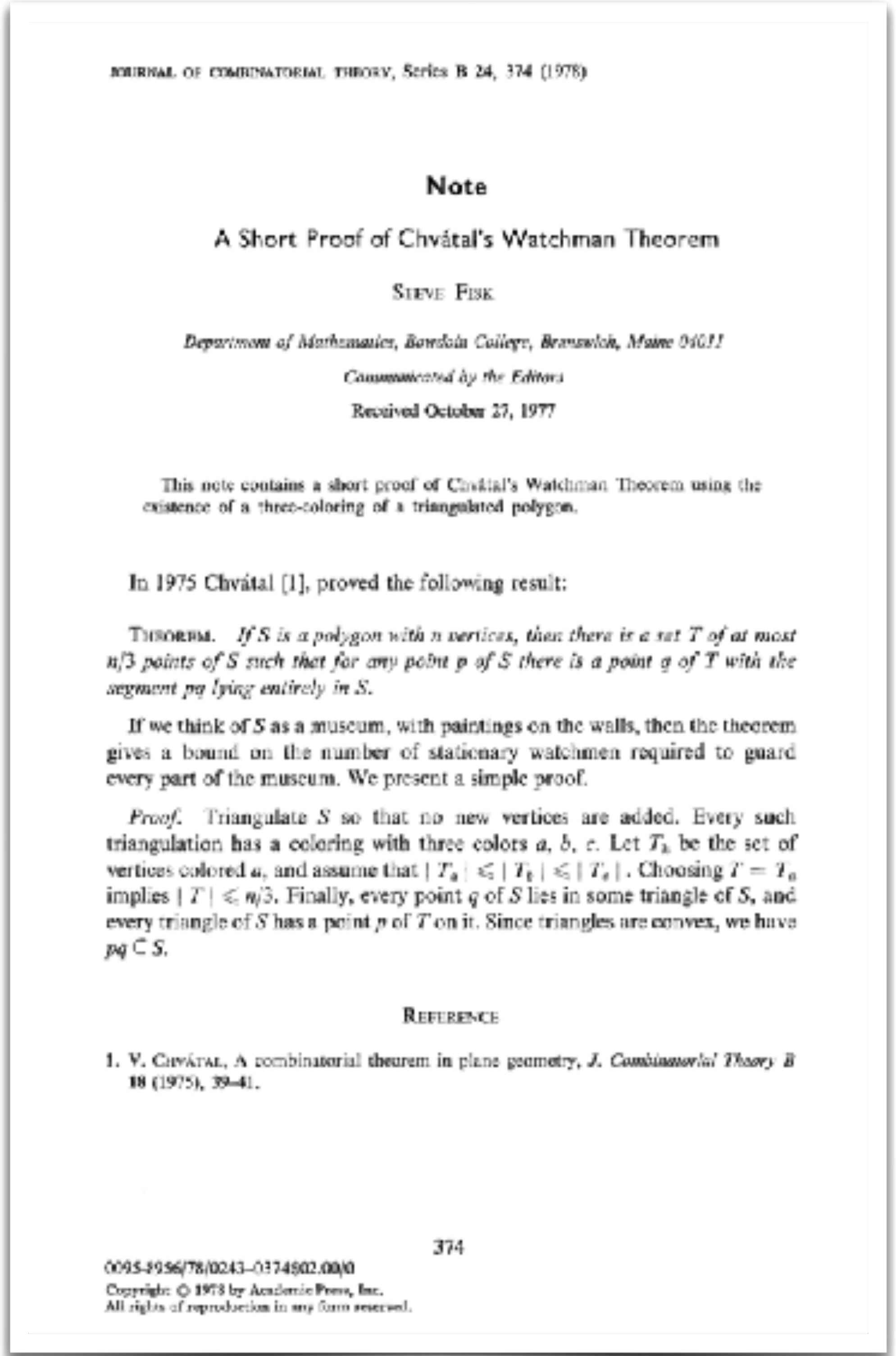
Tarjan and van Wyk (1988): $O(n \log \log n)$

Kirkpatrick, Klawe, Tarjan (1992): $O(n \log \log n)$

Clarkson, Tarjan, van Wyk (1989): $O(n \log^* n)$

Chazelle (1991): $O(n)$

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation



JOURNAL OF COMBINATORIAL THEORY, Series B 24, 374 (1978)

Note

A Short Proof of Chvátal's Watchman Theorem

STEVE FISK

Department of Mathematics, Bowdoin College, Brunswick, Maine 04011

Communicated by the Editors

Received October 27, 1977

This note contains a short proof of Chvátal's Watchman Theorem using the existence of a three-coloring of a triangulated polygon.

In 1975 Chvátal [1], proved the following result:

THEOREM. *If S is a polygon with n vertices, then there is a set T of at most $n/3$ points of S such that for any point p of S there is a point q of T with the segment pq lying entirely in S.*

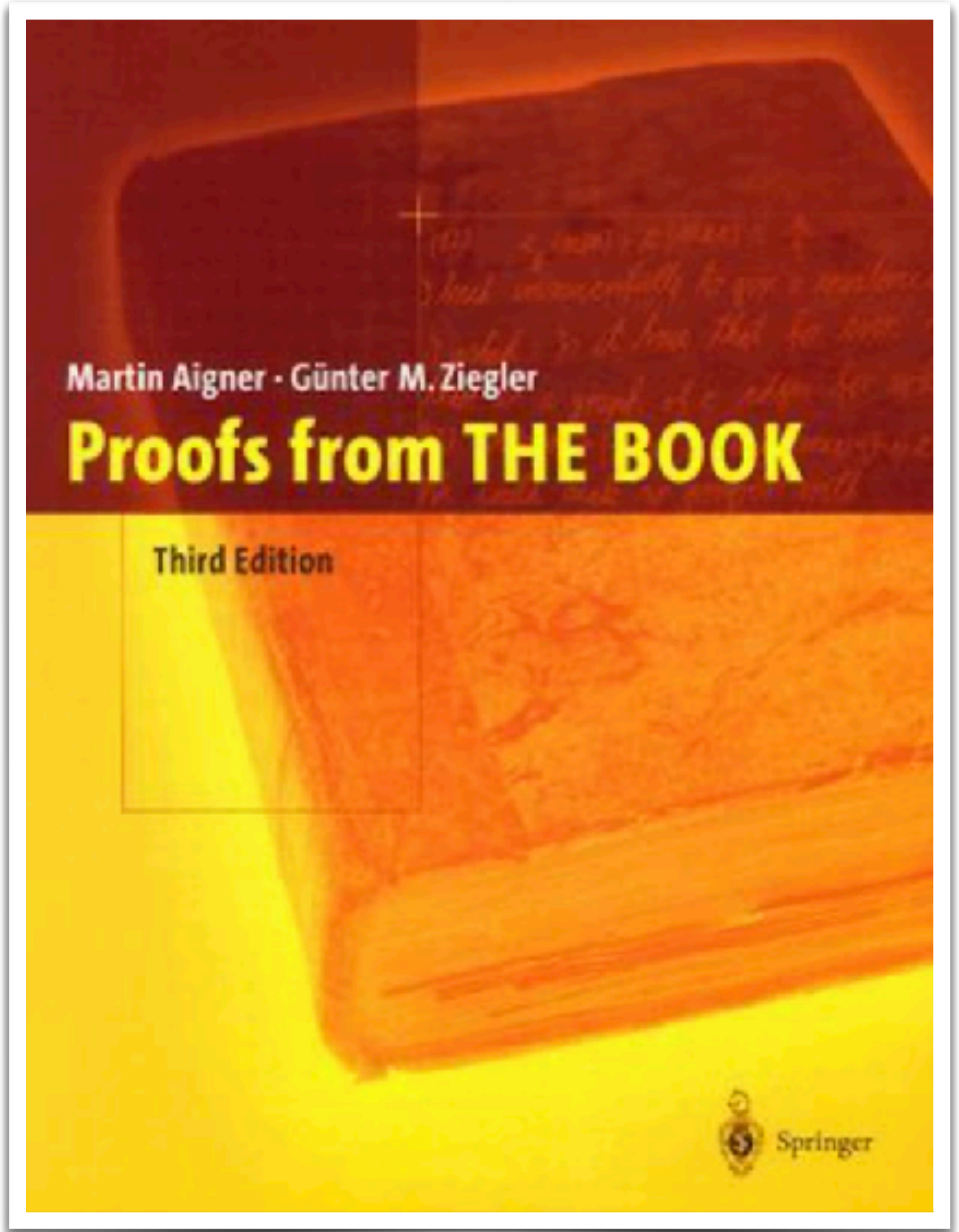
If we think of S as a museum, with paintings on the walls, then the theorem gives a bound on the number of stationary watchmen required to guard every part of the museum. We present a simple proof.

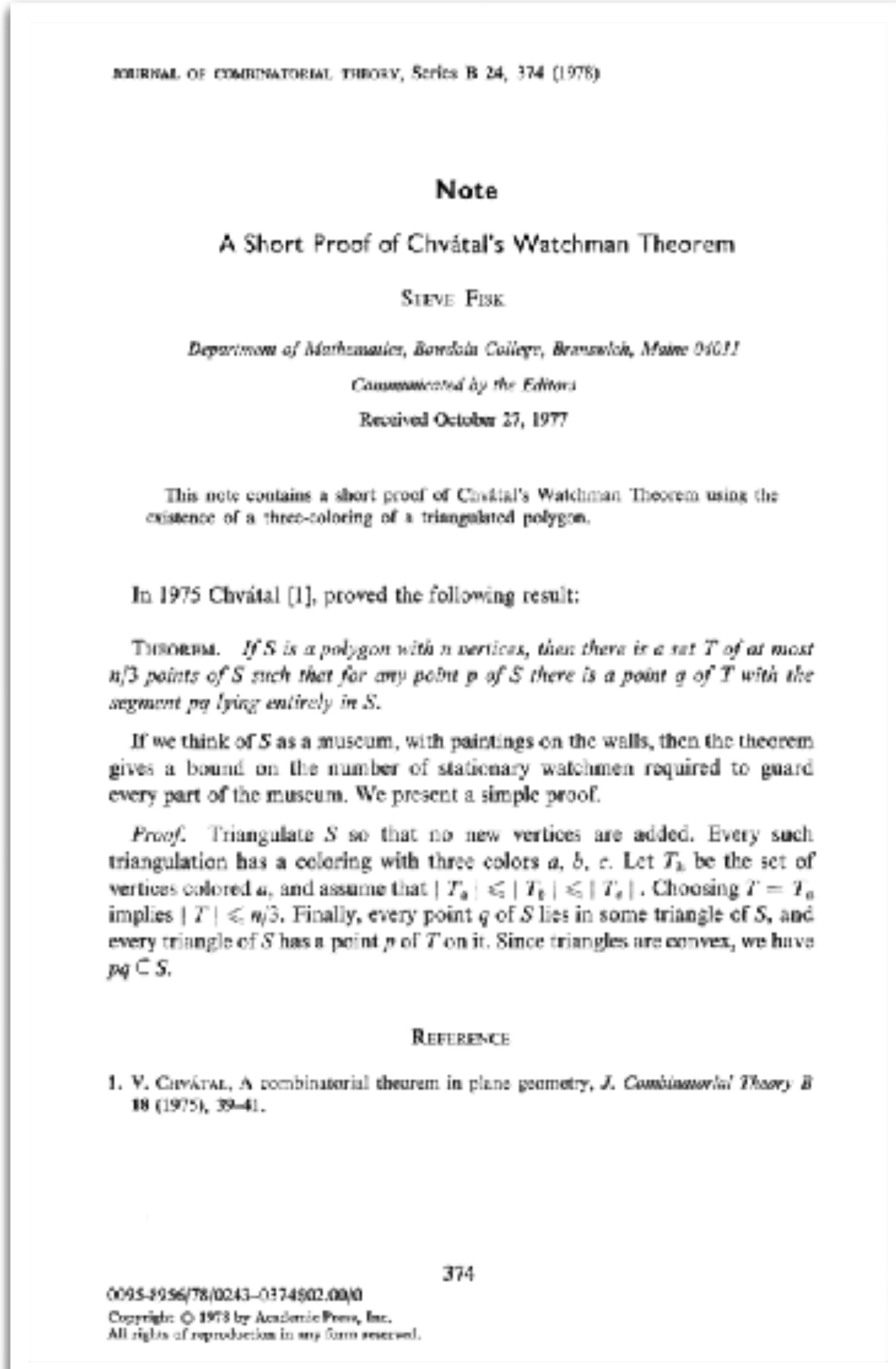
Proof. Triangulate S so that no new vertices are added. Every such triangulation has a coloring with three colors a, b, c. Let T_k be the set of vertices colored k, and assume that $|T_a| \leqslant |T_b| \leqslant |T_c|$. Choosing $T = T_a$ implies $|T| \leqslant n/3$. Finally, every point q of S lies in some triangle of S, and every triangle of S has a point p of T on it. Since triangles are convex, we have $pq \subset S$.

REFERENCE

1. V. CHVÁTAL, A combinatorial theorem in plane geometry, *J. Combinatorial Theory B* **18** (1975), 39–41.

0095-8956/78/0243-0374$02.00/0





JOURNAL OF COMBINATORIAL THEORY, Series B 24, 374 (1978)

Note

A Short Proof of Chvátal's Watchman Theorem

STEVE FISK

Department of Mathematics, Bowdoin College, Brunswick, Maine 04011

Communicated by the Editors

Received October 27, 1977

This note contains a short proof of Chvátal's Watchman Theorem using the existence of a three-coloring of a triangulated polygon.

In 1975 Chvátal [1], proved the following result:

THEOREM. *If S is a polygon with n vertices, then there is a set T of at most $n/3$ points of S such that for any point p of S there is a point q of T with the segment pq lying entirely in S.*

If we think of S as a museum, with paintings on the walls, then the theorem gives a bound on the number of stationary watchmen required to guard every part of the museum. We present a simple proof.

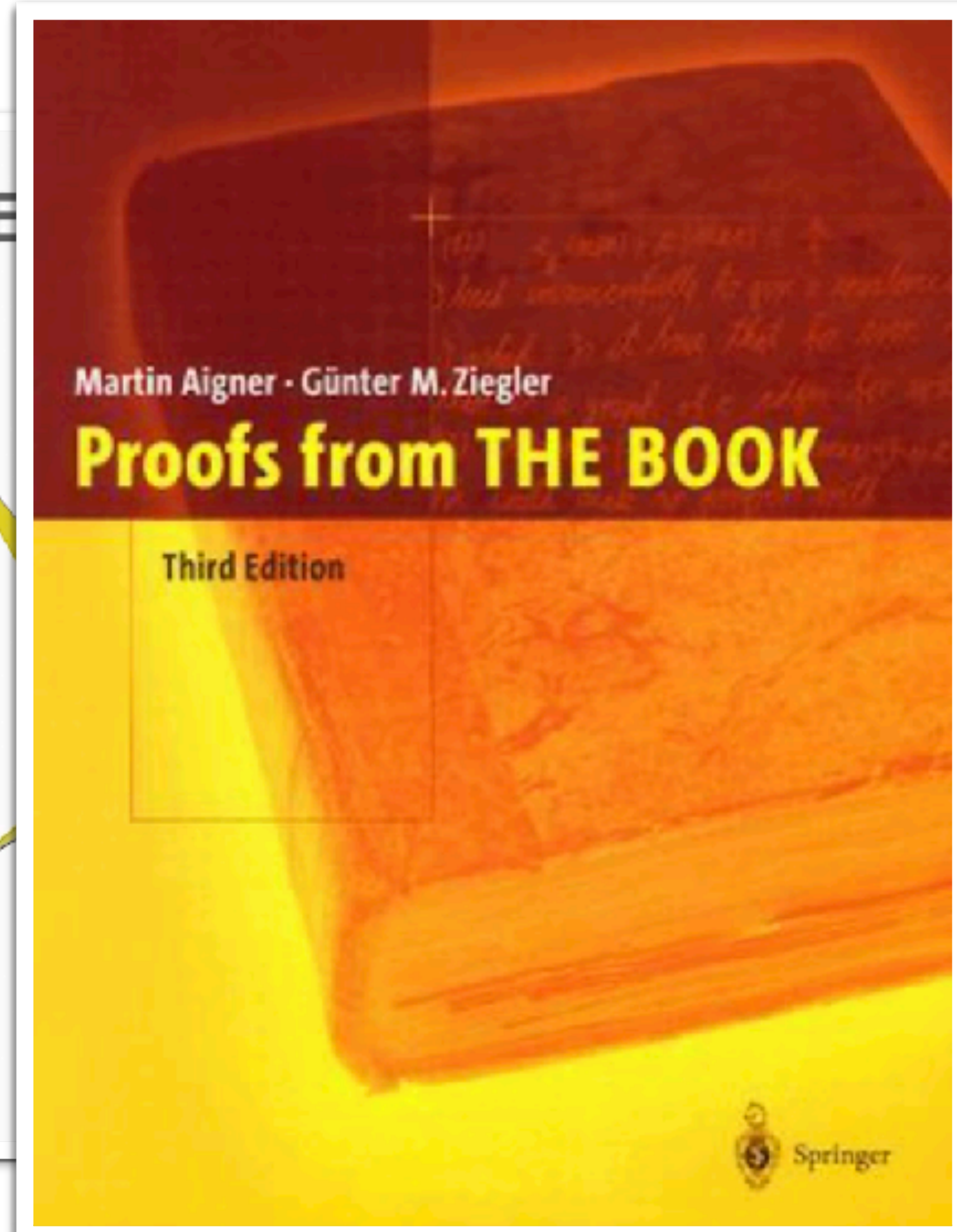
Proof. Triangulate S so that no new vertices are added. Every such triangulation has a coloring with three colors a, b, c. Let T_k be the set of vertices colored a, and assume that $|T_a| \leqslant |T_b| \leqslant |T_c|$. Choosing $T = T_a$ implies $|T| \leqslant n/3$. Finally, every point q of S lies in some triangle of S, and every triangle of S has a point p of T on it. Since triangles are convex, we have $pq \subset S$.

REFERENCE

1. V. CHVÁTAL, A combinatorial theorem in plane geometry, *J. Combinatorial Theory B* 18 (1975), 39–41.

374

0095-8956/78/0243-0374$02.00/0



Technische Universität Braunschweig

1. Introduction
2. Existence
3. Properties
4. Algorithms: Removing ears
5. Algorithms: Finding diagonals
6. Algorithms: Monotone polygons
7. Algorithms: Monotone decompositions
8. Faster algorithms
9. Application: Art Gallery problems
10. Application: Online triangulation


Triangulating Unknown Environments
using Robot Swarms
Aaron Becker
James McLurkin
SeoungKyou Lee
RICE
Sándor P. Fekete
Alexander Kröller
Christiane Schmidt
Technische
Universität
Braunschweig

Thank you for today!

Technische
Universität
Braunschweig