Computational Geometry
Chapter 3: Closest Pair

Prof. Dr. Sándor Fekete

Algorithms Division
Department of Computer Science
TU Braunschweig

Technische Universität Braunschweig

1. Introduction
2. Prelude: Solving recursions
3. Lower bounds
4. Divide-and-conquer
5. Randomized incremental construction

myshapers.com

Closest Pear

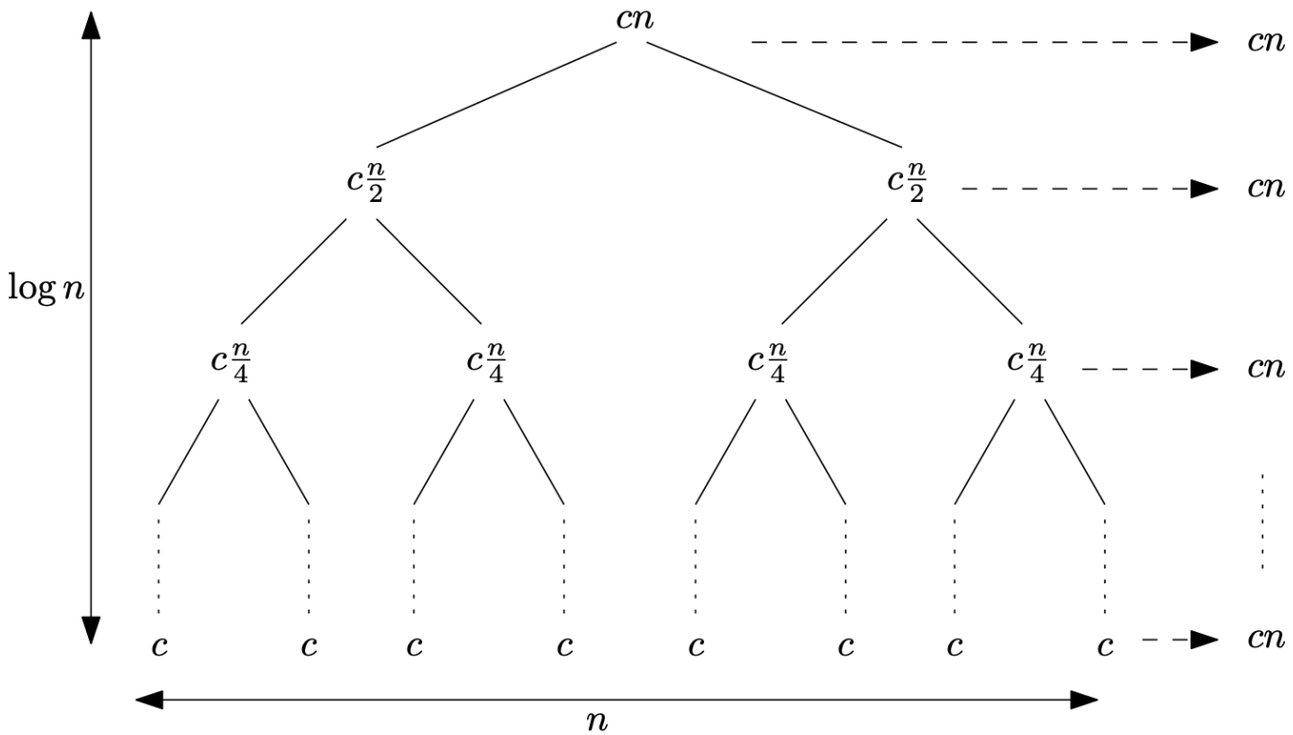
1. Introduction
2. Prelude: Solving recursions
3. Lower bounds
4. Divide-and-conquer
5. Randomized incremental construction

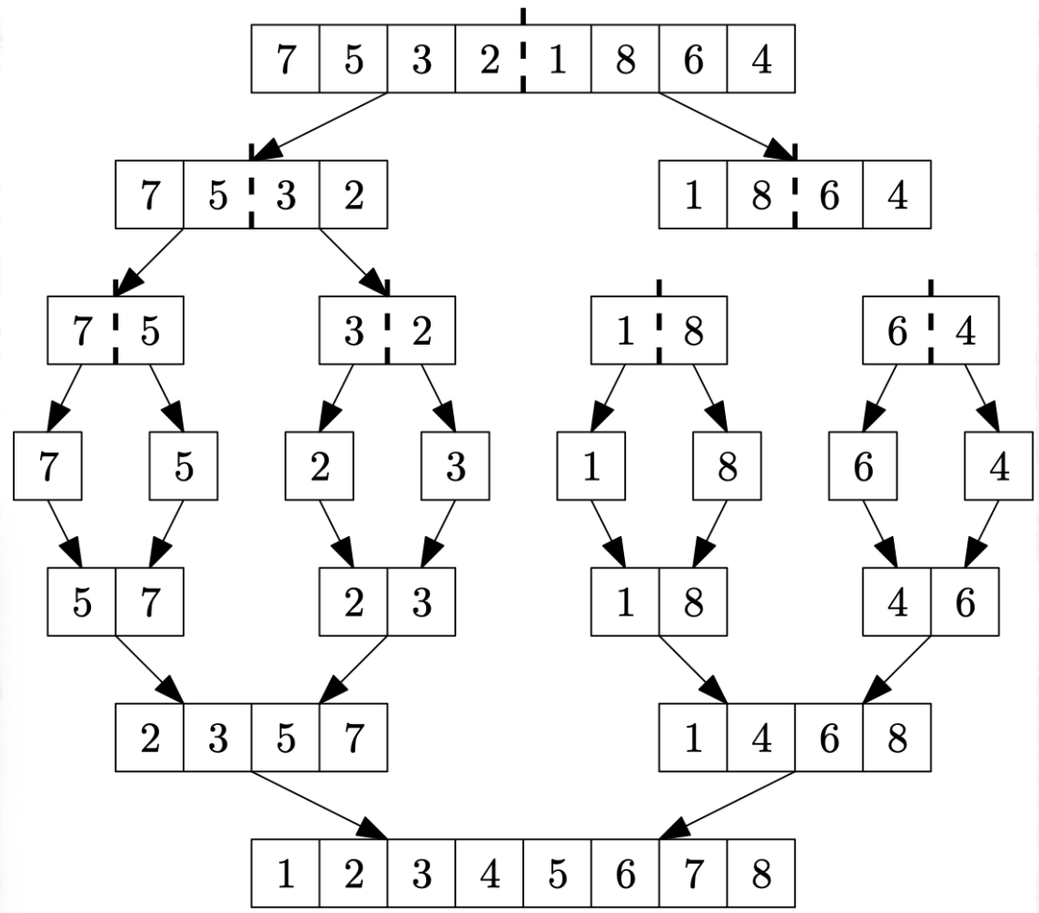
Theorem 3.3 (Master Theorem) *Let* $T : \mathbb{N} \to \mathbb{R}$ *with*

$$T(n) = \sum_{i=1}^{m} T(\alpha_i \cdot n) + \Theta(n^k),$$

where $\alpha_i \in \mathbb{R}$ *with* $0 < \alpha_i < 1$, $m \in \mathbb{N}$ *and* $k \in \mathbb{R}$. *Then*

$$T(n) \in \begin{cases} \Theta(n^k) & \text{for } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta(n^k \log(n)) & \text{for } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c) & \text{with } \sum_{i=1}^{m} \alpha_i^c = 1 \text{ for } \sum_{i=1}^{m} \alpha_i^k > 1 \end{cases}$$

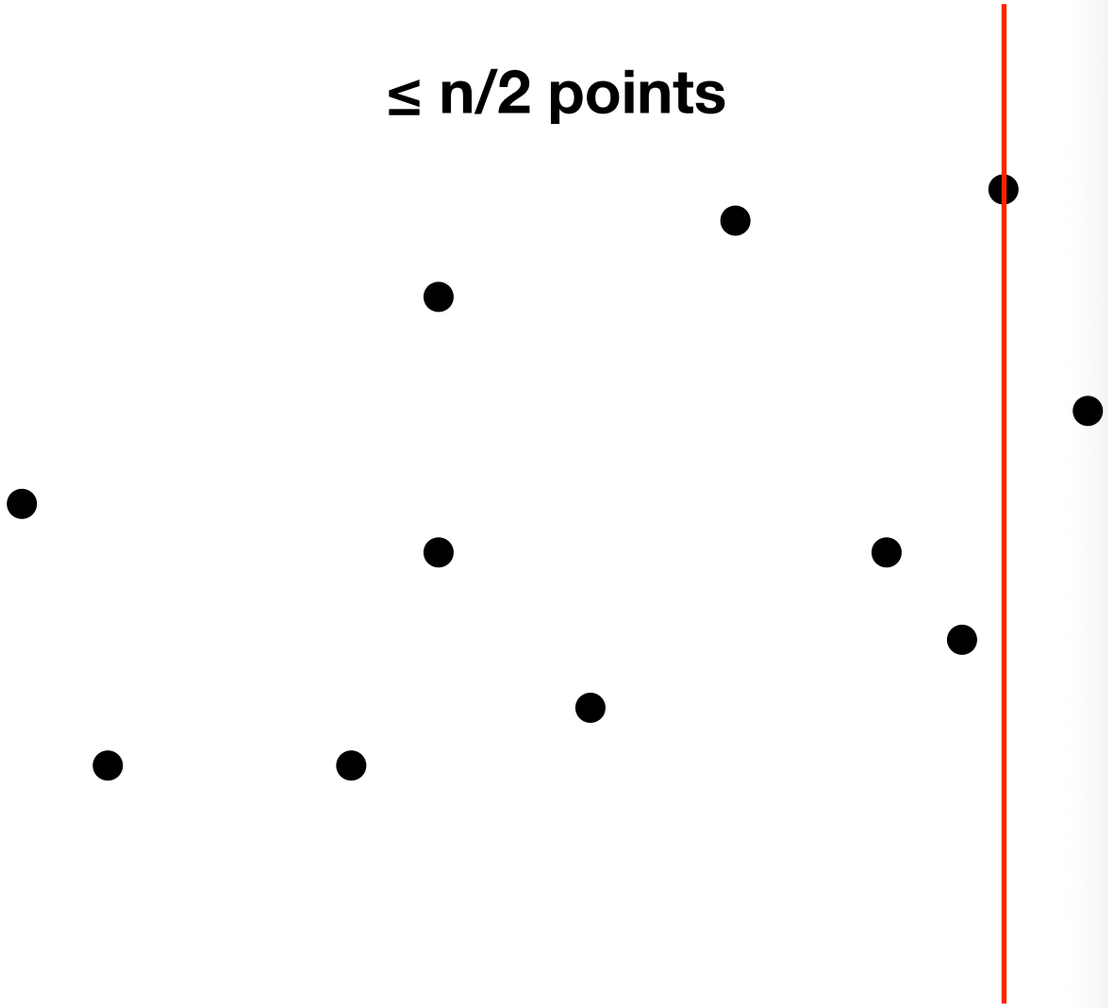




Mergesort

$$T(n) = 2 \cdot T(\frac{n}{2}) + c \cdot n, \quad n \geq 2, c \in \mathbb{R}$$

Medians [Blum, Floyd, Pratt, Rivest, Tarjan 1973]



Theorem 3.4

A median for n numbers can be computed in $\mathcal{O}(n)$.

JOURNAL OF COMPUTER AND SYSTEM SCIENCES 7, 448–461 (1973)

Time Bounds for Selection*

MANUEL BLUM, ROBERT W. FLOYD, VAUGHAN PRATT,
RONALD L. RIVEST, AND ROBERT E. TARJAN

Department of Computer Science, Stanford University, Stanford, California 94305

Received November 14, 1972

The number of comparisons required to select the i-th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm—PICK. Specifically, no more than $5.430\dot{5}\,n$ comparisons are ever required. This bound is improved for extreme values of i, and a new lower bound on the requisite number of comparisons is also proved.

1. INTRODUCTION

In this paper we present a new selection algorithm, PICK, and derive by an analysis of its efficiency the (surprising) result that the cost of selection is at most a linear function of the number of input items. In addition, we prove a new lower bound for the cost of selection.

The selection problem is perhaps best exemplified by the computation of medians. In general, we may wish to select the i-th smallest of a set of n distinct numbers, or the element ranking closest to a given percentile level.

Interest in this problem may be traced to the realm of sports and the design of (traditionally, tennis) tournaments to select the first- and second-best players. In 1883, Lewis Carroll published an article [1] denouncing the unfair method by which the second-best player is usually determined in a "knockout tournament"—the loser of the final match is often not the second-best! (Any of the players who lost only to the best player may be second-best.) Around 1930, Hugo Steinhaus brought the problem into the realm of algorithmic complexity by asking for the minimum number of matches required to (correctly) select both the first- and second-best players from a field of n contestants. In 1932, J. Schreier [8] showed that no more than $n + \lceil \log_2(n) \rceil - 2$ matches are required, and in 1964, S. S. Kislitsin [6] proved this number to be necessary as well. Schreier's method uses a knockout tournament to determine the winner, followed by a second knockout tournament among the

* This work was supported by the National Science Foundation under grant GJ-992.

Theorem 3.4

A median for n numbers can be computed in $\mathcal{O}(n)$.

Proof idea:

$$X = \{1, 22, 10, 13, 24, 6, 18, 21, 4, 25, 11, 16, 2, 20, 8, 17, 5, 12, 19, 14, 3, 9, 15, 7, 23\}$$

- Group the numbers into sets of 5.

1	6	11	17	3
22	18	16	5	9
10	21	2	12	15
13	4	20	19	7
24	25	8	14	23

$\mathcal{O}(n)$

- Sort all quintuples.

1	4	2	5	3
10	6	8	12	7
13	18	11	14	9
22	21	16	17	15
24	25	20	19	23

$\mathcal{O}(n)$

Proof idea (cont):

- Compute the median of medians.

1	4	2	5	3
10	6	8	12	7
13	18	11	14	9
22	21	16	17	15
24	25	20	19	23

$T(\frac{n}{5})$

- Use the median of medians as pivot to reduce the set of numbers.

≥ n/4 numbers

≥ n/4 numbers

3	2	1	5	4
7	8	10	12	6
9	11	13	14	18
15	16	22	17	21
23	20	24	19	25

$T(\frac{3n}{4})$

$$T(n) = T(\frac{n}{5}) + T(\frac{3n}{4}) + \Theta(n)$$

Theorem 3.3 (Master Theorem) *Let* $T : \mathbb{N} \to \mathbb{R}$ ...

$$T(n) \in \Theta(n^k) = \Theta(n)$$

where $\alpha_i \in$ ... $\in \mathbb{N}$ *and* $k \in \mathbb{R}$. *Then*

$$T(n) \in \begin{cases} \Theta(n^k) & \text{for } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta(n^k \log(n)) & \text{for } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c) & \text{with } \sum_{i=1}^{m} \alpha_i^c = 1 \text{ for } \sum_{i=1}^{m} \alpha_i^k > 1 \end{cases}$$

$$T(n) = T(\frac{n}{5}) + T(\frac{3n}{4}) + \Theta(n)$$

$$\sum_{i=1}^{m} \alpha_i^k = \frac{1}{5} + \frac{3}{4} = \frac{19}{20} < 1.$$

1. Introduction
2. Prelude: Solving recursions
3. Lower bounds
4. Divide-and-conquer
5. Randomized incremental construction

Theorem 3.6

Computing a closest pair for n numbers t

Theorem 1. *Any algebraic computation tree that solves the n-element distinctness problem must have complexity of at least* $\Omega(n \log n)$.

Lower Bounds For Algebraic Computation Trees

(Preliminary Report)

Michael Ben-Or †

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract — A topological method is given for obtaining lower bounds for the height of algebraic computation trees, and algebraic decision trees. Using this method we are able to generalize, and present in a uniform and easy way, almost all the known nonlinear lower bounds for algebraic

Despite the extensive research in algebraic complexity theory in recent years, no general lower bound method has been provided for algorithms that involve arithmetical operations and comparisons. Much less is known if we further allow the operation of root extraction or the algebraic operation of finding the root of a polynomial. Consider the following decision problem:

Example 1. Element Distinctness. *Given* $x_1, \ldots, x_n \in \mathbf{R}$, *is there a pair* i, j *with* $i \neq j$ *and* $x_i = x_j$ *?*

One can solve the element distinctness problem with the help of any efficient sorting algorithm using $O(n \log n)$ comparisons, or by computing the product $\prod_{i \neq j}(x_i - x_j)$ and comparing the computed result to zero (using $O(n \log n)$ mult/div). Allowing linear operations for free, we know of no previous result that indicates why more than $O(1)$ operations are required to solve this problem in the model considered here.

In this paper we provide a new topological method for obtaining lower bounds for this general type of algorithms, formally described as *algebraic computation trees.* Before giving the detailed computational model it is worthwhile to mention a concrete application of the method presented here.

Theorem 1. *Any algebraic computation tree that solves the n-element distinctness problem must have complexity of at least* $\Omega(n \log n)$.

This result extends the lower bounds of Dobkin and Lipton [5] for the linear decision tree model, and the lower bounds of Baur and Strassen [1] for the straight line complexity of the above product.

Our new lower bound method rests heavily on a result from real algebraic geometry due to Milnor [11] and Thom [23] that bounds the "topological complexity" of real algebraic varieties. Except for this result the proofs of our main theorems are elementary and require only basic knowledge of algebra and topology. The new method also provides a unified and easy way to prove nonlinear lower bounds for straight line computations, algebraic decision trees, and other previously untouchable problems such as lower bounds for the complexity of constructions with a ruler and compass in plane Euclidean geometry.

In the next section we rigorously specify our basic computational model. The third section is devoted to a technical result needed for our main theorems that are presented in section four. In section four we also show how to extend our computational model to allow more algebraic operations such as taking k-th roots or computing the roots of a polynomial.

In section five we show how to apply our method to the bounded degree algebraic decision tree model, thus solving the open problems in [20]. Section six is devoted to applications and in particular to the proof of the result on the element distinctness problem (Theorem 1) mentioned above.

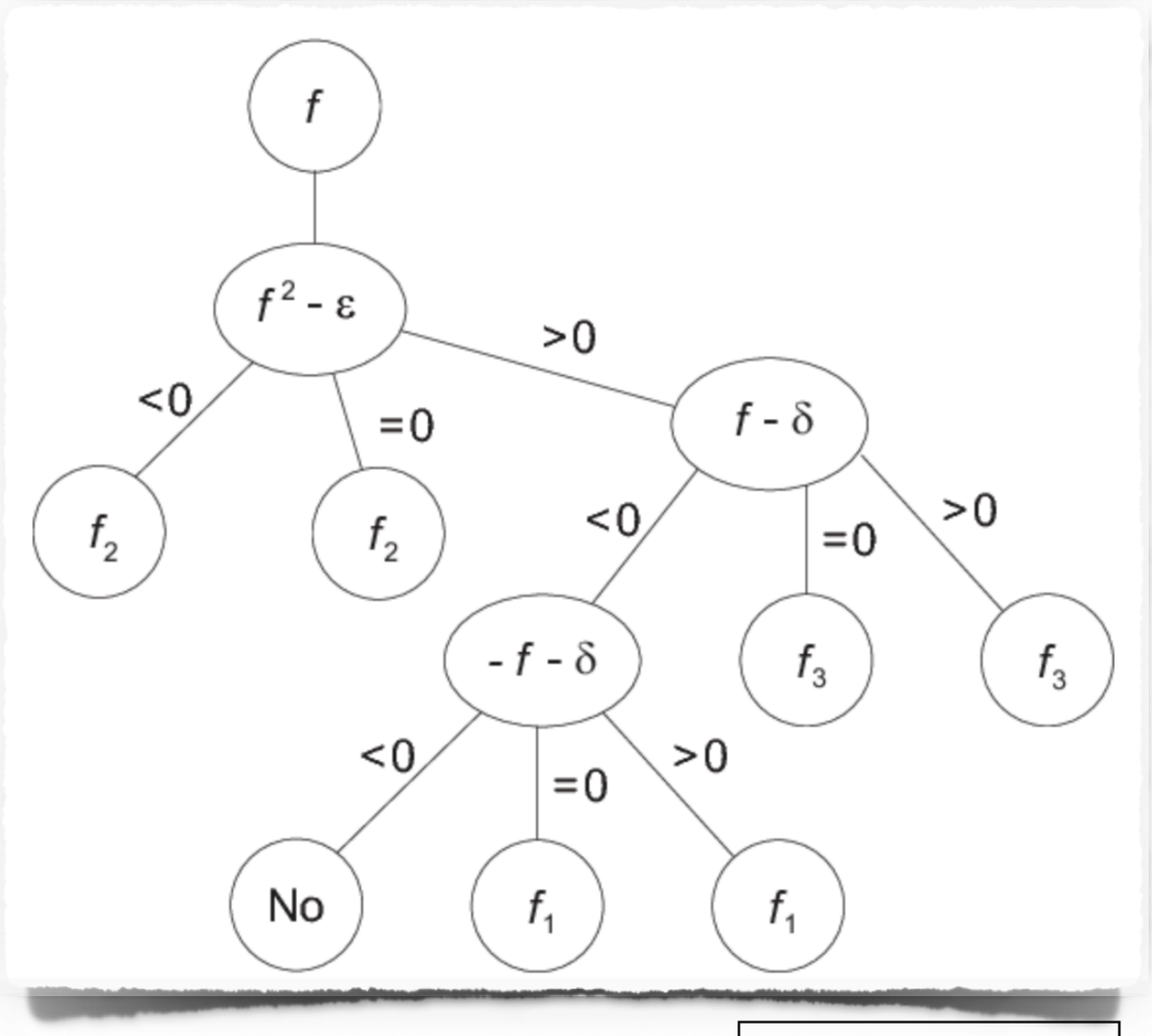
† Research supported by a Weizmann Postdoctoral fellowship and by NSF grant MCS-8006938.

Theorem 3.6

Computing a closest pair for n numbers takes at least $\Omega(n \log n)$.

Ideas:

- Consider *algebraic decision trees.*



A Gabrielov, N Vorobjov

Theorem 3.6

Computing a closest pair for n numbers takes at least $\Omega(n \log n)$.

Ideas:

- Consider *algebraic decision trees.*
- Each set of numbers corresponds to a point in $\mathbb{R}^n$.
- Consider subsets of $\mathbb{R}^n$ that share the same membership properties.
- Consider the number N of connected components of $\mathbb{R}^n$.
- Show that the height of an algebraic decision tree is at least $\Omega(\log N - n)$.
- Show that ELEMENT UNIQUENESS has many connected components.
- Note that determining a small minimum distance solves ELEMENT UNIQUENESS.

1. Introduction
2. Prelude: Solving recursions
3. Lower bounds
4. Divide-and-conquer
5. Randomized incremental construction

Divide-and-Conquer [Bentley and Shamos 1976]

DIVIDE-AND-CONQUER IN MULTIDIMENSIONAL SPACE

Jon Louis Bentley
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

and

Michael Ian Shamos
Departments of Computer Science and Mathematics
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

We investigate a divide-and-conquer technique in multidimensional space which decomposes a geometric problem on N points in k dimensions into two problems on N/2 points in k dimensions plus a single problem on N points in k-1 dimension. Special structure of the subproblems is exploited to obtain an algorithm for finding the two closest of N points in O(N log N) time in any dimension. Related results are discussed, along with some conjectures and unsolved geometric problems.

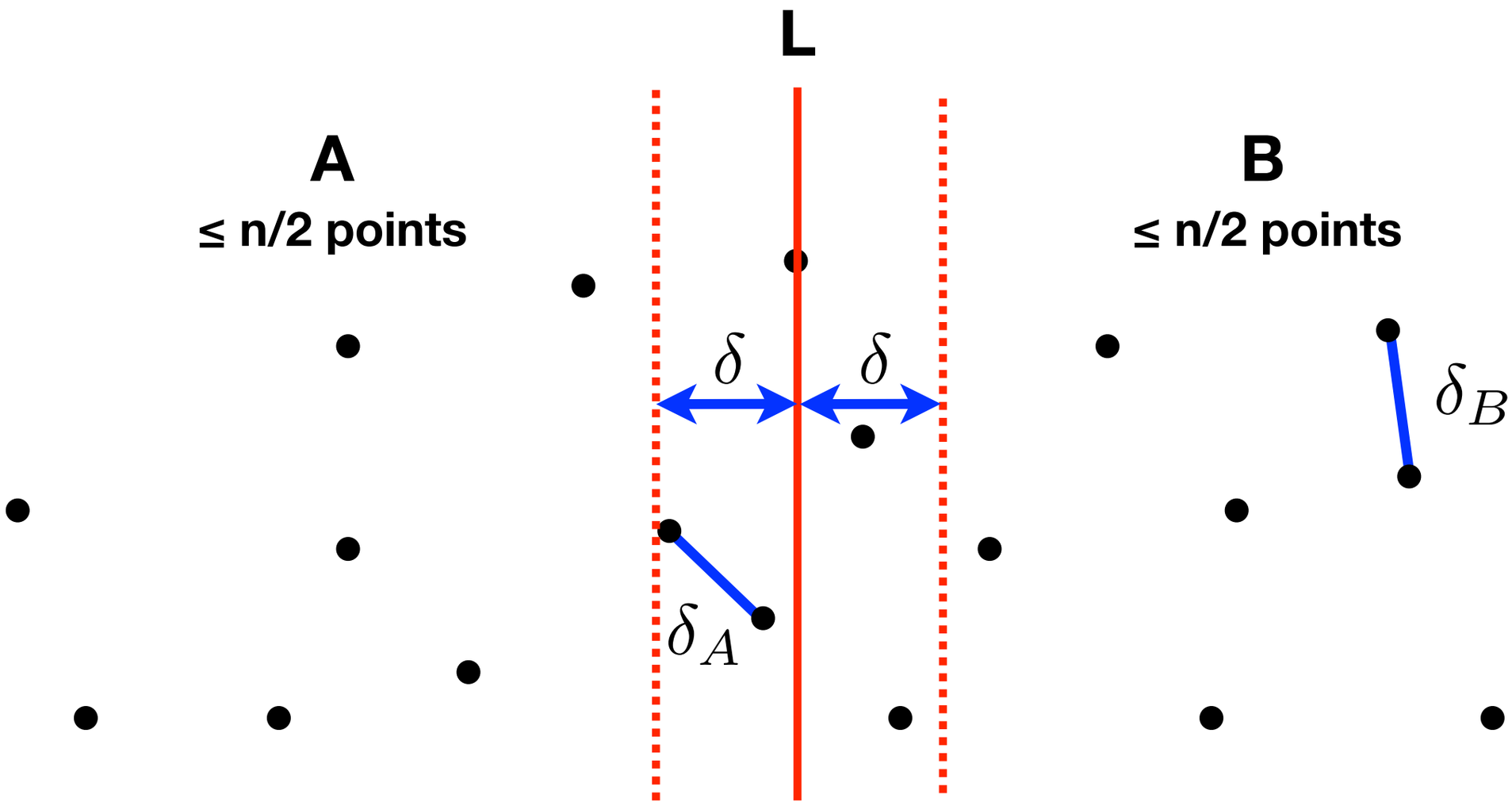
Introduction

A principal failing of computational geometry [Shamos, Shamos and Hoey] is that it has not successfully addressed problems in greater than two dimensions. Such a study would have important practical and theoretical benefits -- it would shed light on linear programming, multidimensional data analysis, geometric optimization, and retrieval on multiple keys, as well as provide a link between complexity and dimensionality. Some preliminary results are known. [Preparata and Hong] show that the convex hull of N points can be found in O(N log N) time in three dimensions but that $O(N^2)$ is a lower bound in any higher dimension. The maxima of a set of vectors can be found in $O(N \log^{k-2} N)$ time in dimension k [Kung et al.] and we conjecture that this time suffices to identify the extreme points of a k-dimensional set.

The most basic questions, however, have not been studied. These include finding nearest and farthest points, determining separability of point sets and other elementary properties. We intend in this paper to begin a systematic investigation of higher-dimensional geometry and its relation to complexity.

Closest-Point Problems

In this section we will investigate a number of problems dealing with the proximity of N points in Euclidean k-space. The most primitive closest-point problem is that of finding the two nearest of the N points; we will let P(N,k) denote the worst-case time of the best possible algorithm (the minimax complexity) for solving the closest-pair problem. The fixed radius-near-neighbor problem asks for all pairs of points within some fixed distance δ of one another. A special case of this



- Subdivide the set by a median line.
- Solve the two subproblems recursively in $2T(\frac{n}{2})$.
- Merge the two subproblems in $\mathcal{O}(n)$.

Divide-and-Conquer 3 [Bentley and Shamos 1976]

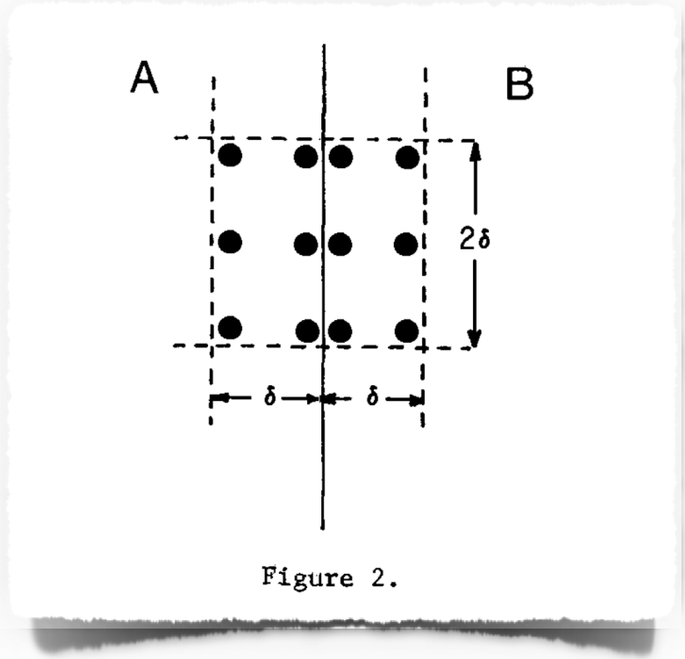
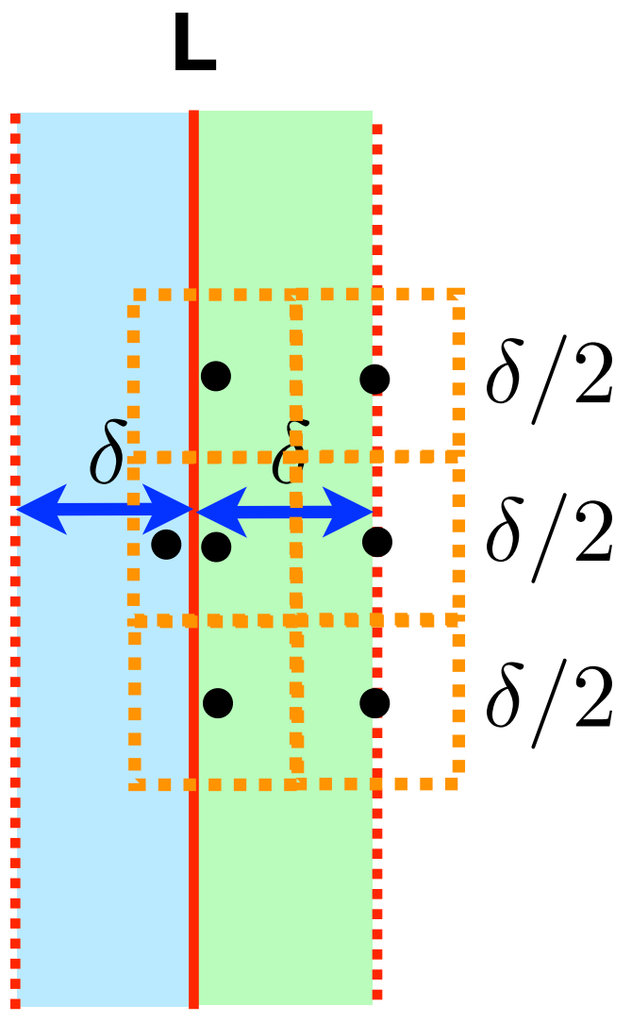
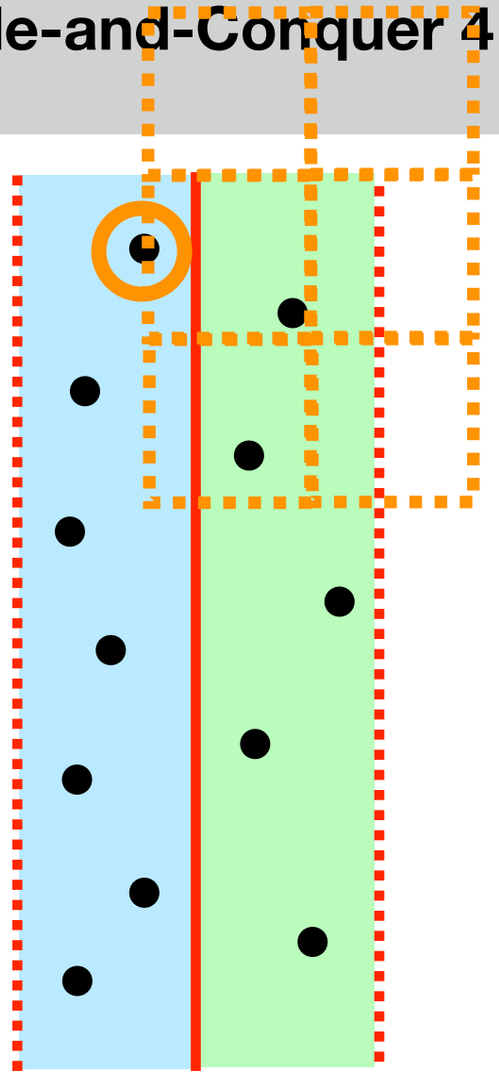


Figure 2.



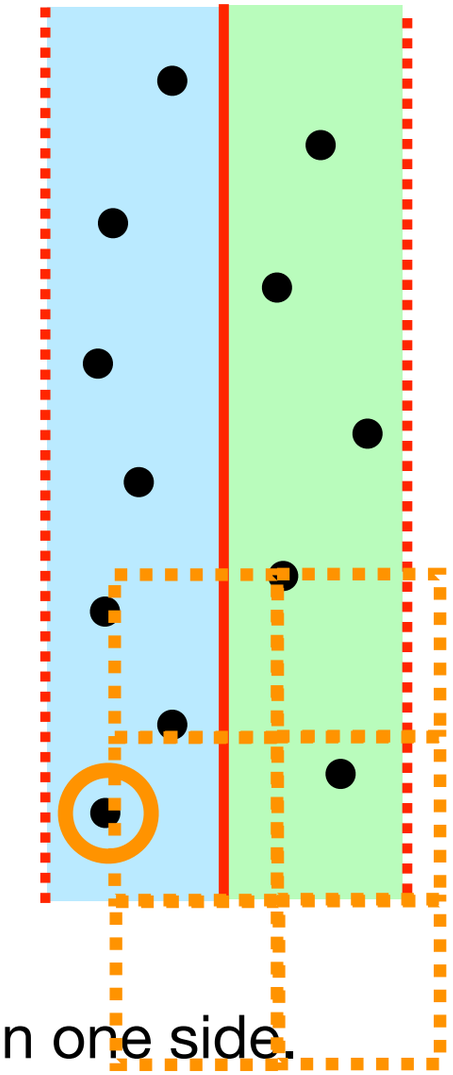
- Observe: Points on one side cannot be closer than δ.
- As consequence, no point can have many close neighbors on the other side.
- Based on a packing arguments for $\delta/2$-balls; works in any fixed dimension.

Divide-and-Conquer 4 [Bentley and Shamos 1976]



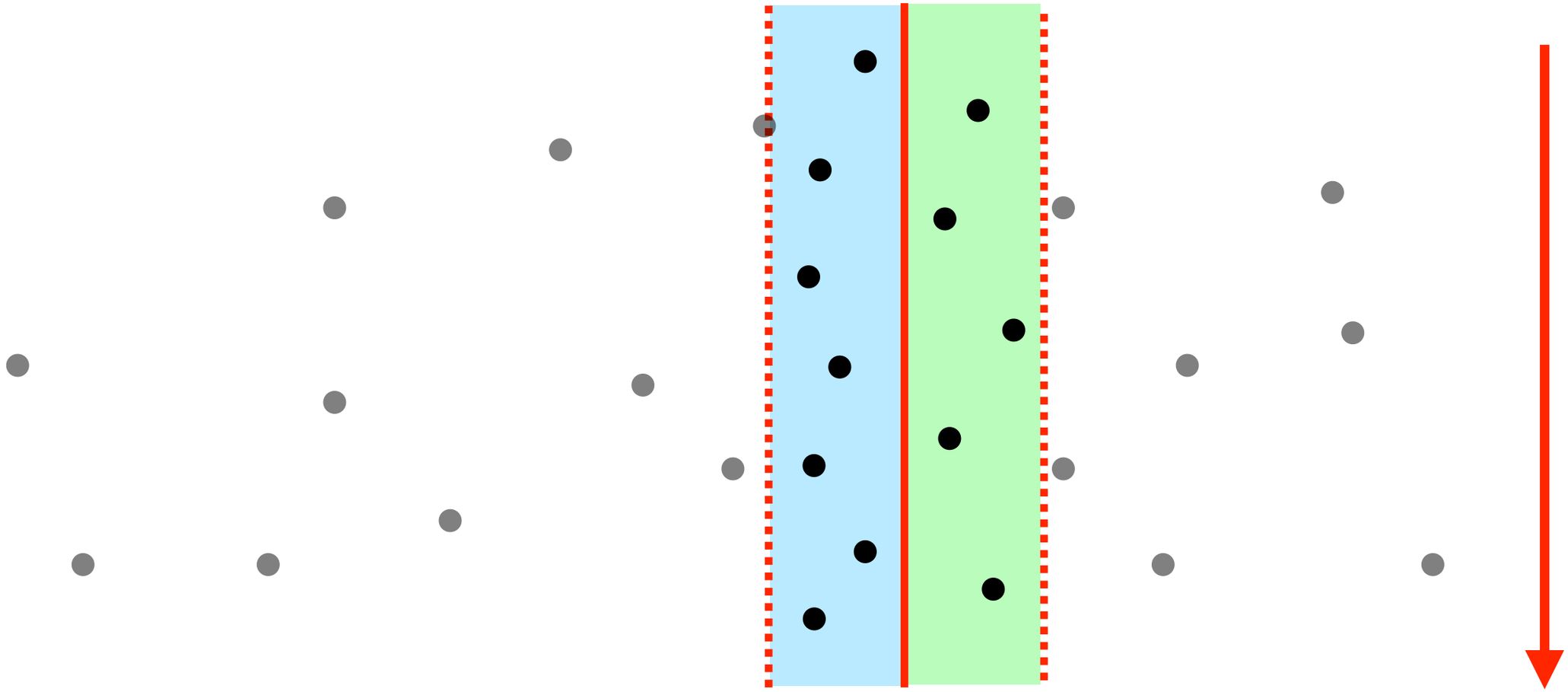
- Thus: Scan linear sequence on one side.
- Keep track of potential neighbors on other side.
- Sorting takes $\mathcal{O}(n \log n)$; total time $\mathcal{O}(n \log^2 n)$.
- After global presorting: $\mathcal{O}(n)$; total time $\mathcal{O}(n \log n)$.

Divide-and-Conquer 4 [Bentley and Shamos 1976]



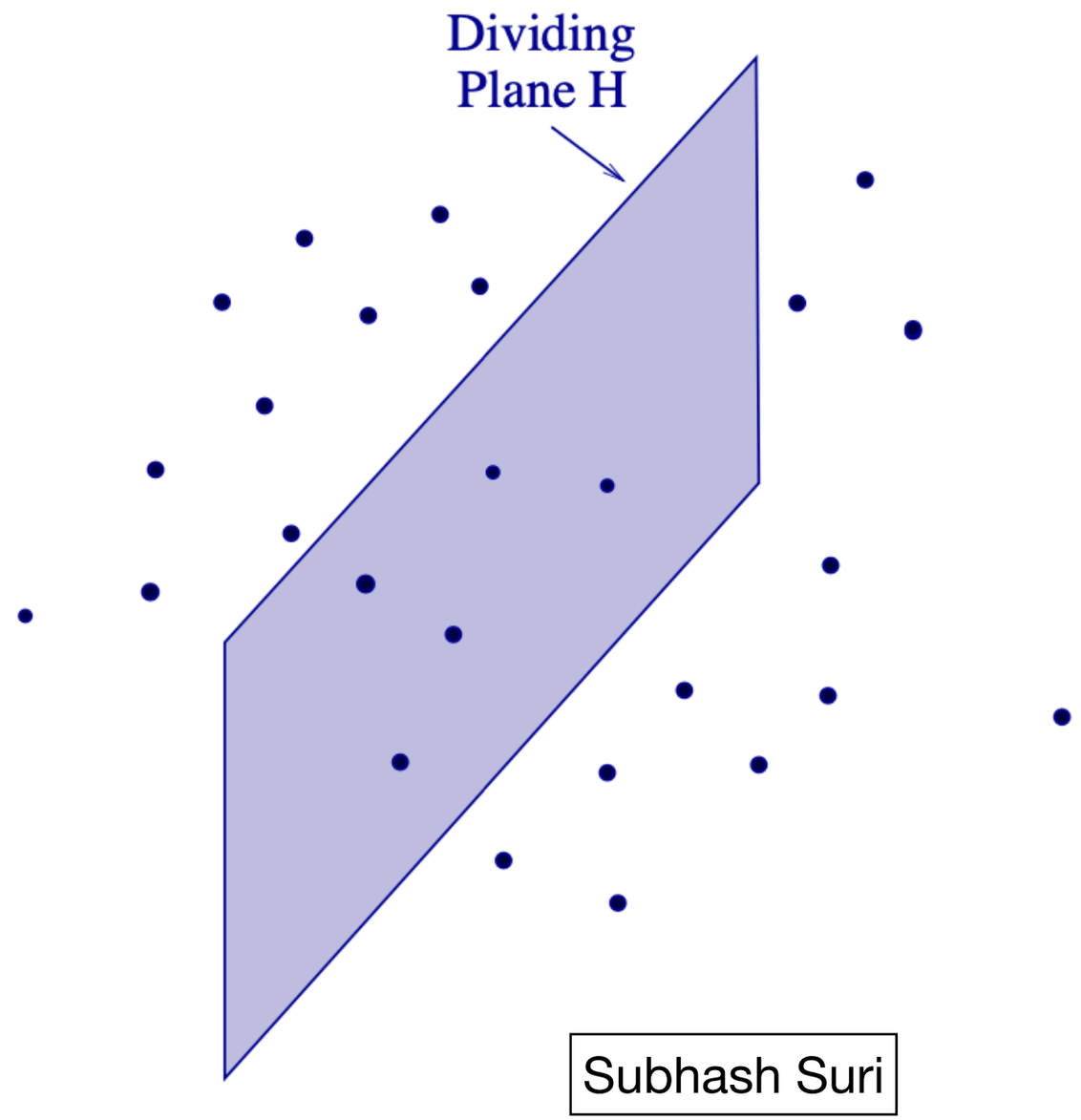
- Thus: Scan linear sequence on one side.
- Keep track of potential neighbors on other side.
- Sorting takes $\mathcal{O}(n \log n)$; total time $\mathcal{O}(n \log^2 n)$.
- After global presorting: $\mathcal{O}(n)$; total time $\mathcal{O}(n \log n)$.

Divide-and-Conquer 4 [Bentley and Shamos 1976]



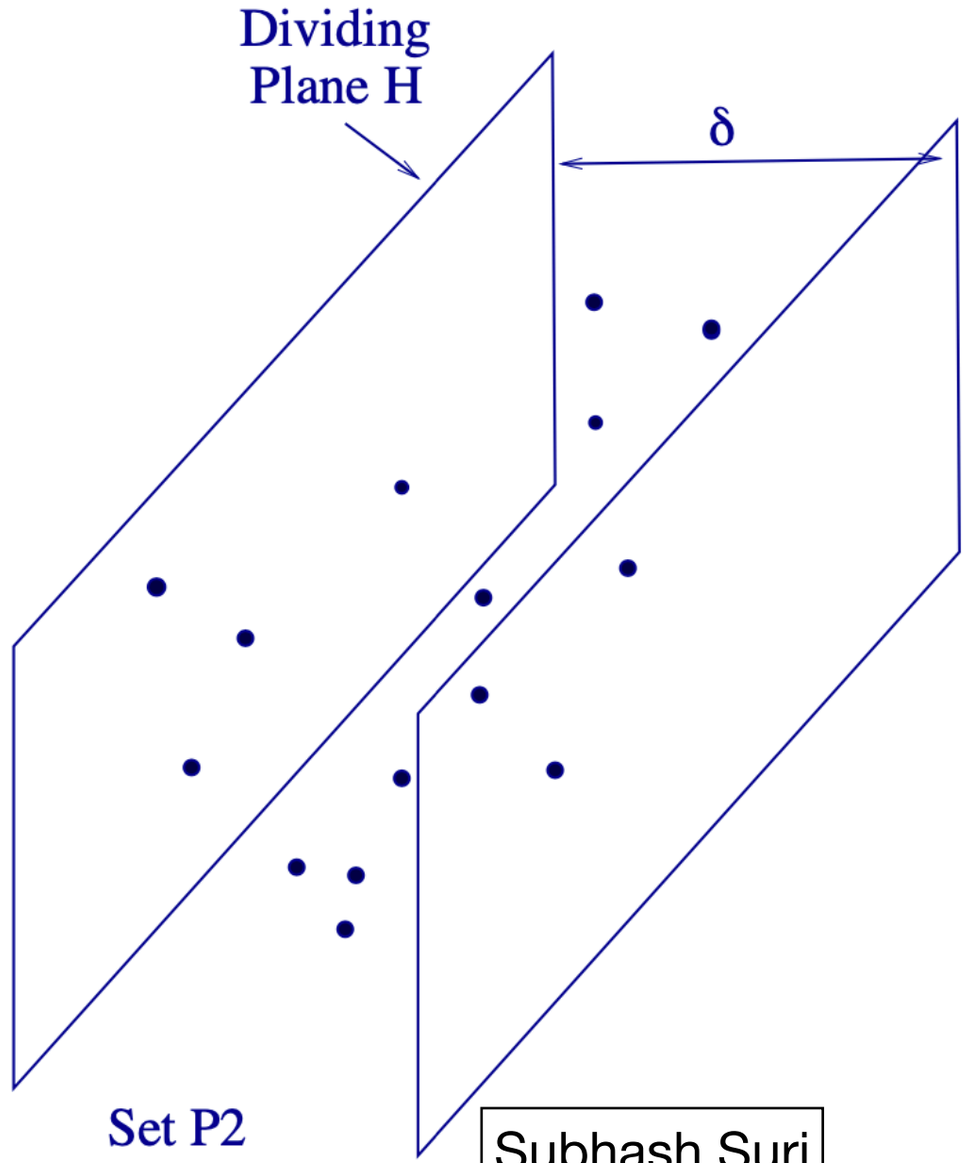
- Thus: Scan linear sequence on one side.
- Keep track of potential neighbors on other side.
- Sorting takes $\mathcal{O}(n \log n)$; total time $\mathcal{O}(n \log^2 n)$.
- After global presorting: $\mathcal{O}(n)$; total time $\mathcal{O}(n \log n)$.

- Same idea: Divide-and-conquer.



Subhash Suri

- Same idea: Divide-and-conquer.
- Merge step: one dimension lower:



Subhash Suri

- **If we could show that the problem size in the conquer step is $m \leq n/(\log n)^{d-2}$, then $U(m, d-1) = O(m(\log m)^{d-2}) = O(n)$.**

- **Theorem: Given a set S with δ-sparsity, there exists a hyperplane H normal to some axis such that**

1. $|S_1|, |S_2| \geq n/4d$.
2. **Number of points within δ of H is $O(\frac{n}{(\log n)^{d-2}})$.**
3. **H can be found in $O(n)$ time.**

- **If we could show that the**
  the conquer step is $m \leq n$
  $U(m, d-1) = O(m(\log m)^{d-2})$

- **Theorem: Given a set** S **with**
  there exists a hyperplane H **n**
  some axis such that

1. $|S_1|, |S_2| \geq n/4d$.
2. **Number of points within** δ **of** H **is** $O(\frac{n}{(\log n)^{d-2}})$.
3. H **can be found in** $O(n)$ **time.**

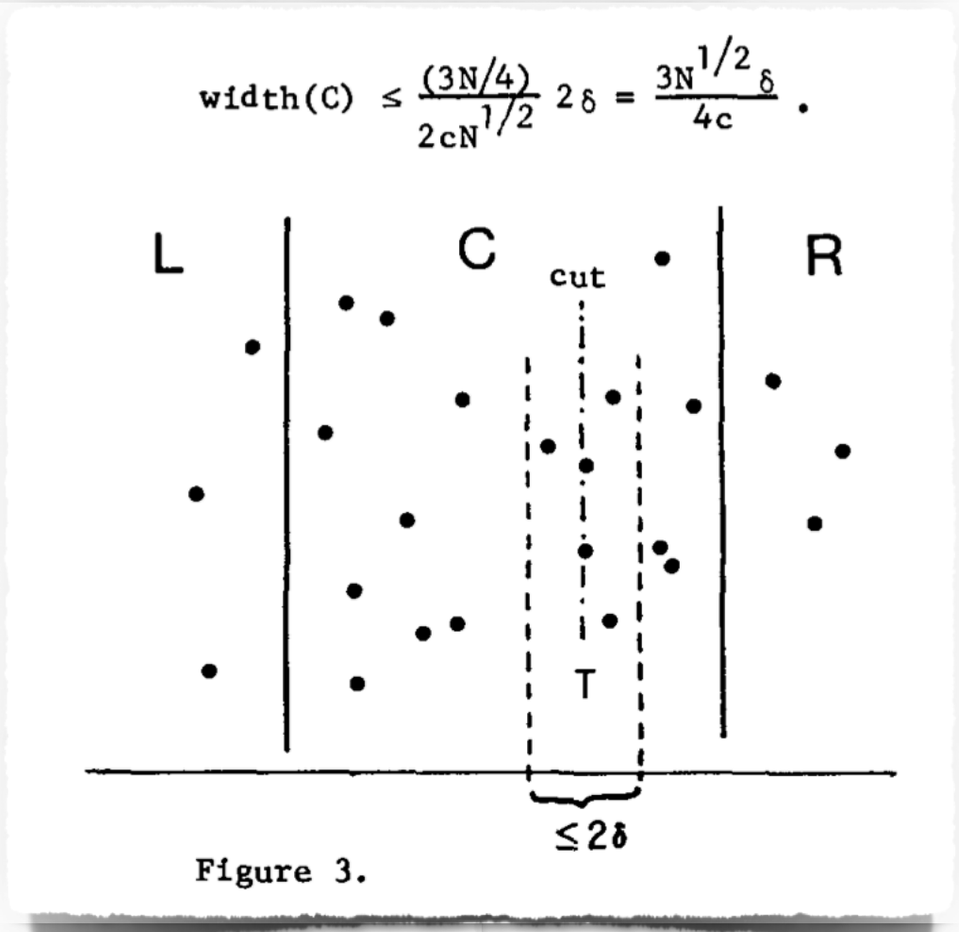


Figure 3.

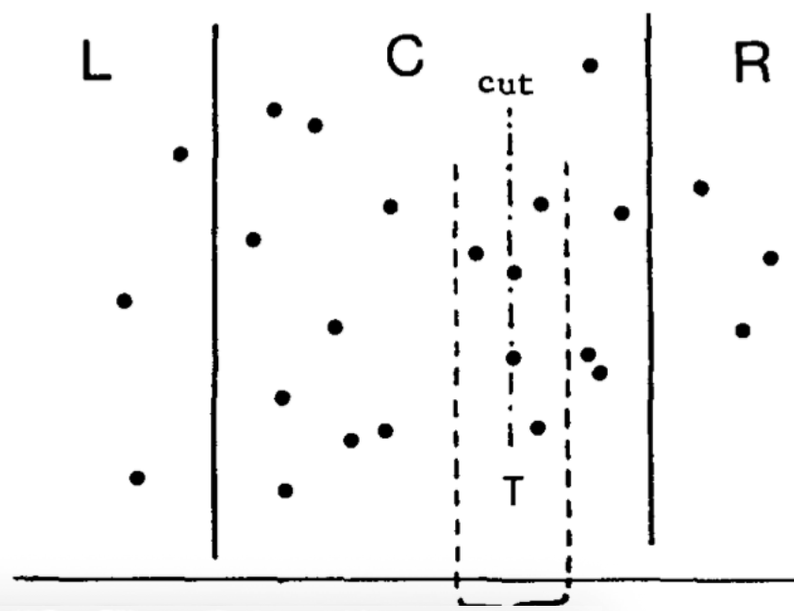
- **If we could show that the**
  the conquer step is $m \leq n$
  $U(m, d-1) = O(m(\log m)^{d-2})$

- **Theorem: Given a set** S **with**
  there exists a hyperplane H **n**
  some axis such that

$(\log n)^{d-2})$

3. H **can be found in** $O(n)$ **time.**

$$\text{width}(C) \leq \frac{(3N/4)}{2cN^{1/2}}\, 2\delta = \frac{3N^{1/2}\delta}{4c}.$$



Theorem 8. P(N,k) ≤ O(N log N).

1. Introduction
2. Prelude: Solving recursions
3. Lower bounds
4. Divide-and-conquer
5. Randomized incremental construction

Chapter 33

Randomized Data Structures for the Dynamic Closest-Pair Problem

Mordecai Golin* Rajeev Raman† Christian Schwarz† Michiel Smid†

Abstract

We describe a new randomized data structure, the *sparse partition*, for solving the dynamic closest-pair problem. Using this data structure the closest pair of a set of n points in k-dimensional space, for any fixed k, can be found in constant time. If the points are chosen from a finite universe, and if the floor function is available at unit-cost, then the data structure supports insertions into and deletions from the set in expected $O(\log n)$ time and requires expected $O(n)$ space. Here, it is assumed that the updates are chosen by an adversary who does not know the random choices made by the data structure. The data structure can be modified to run in $O(\log^2 n)$ expected time per update in the algebraic decision tree model of computation. Even this version is more efficient than the currently best known deterministic algorithms for solving the problem for $k > 1$.

1 Introduction

We consider the *dynamic closest-pair problem*: We are given a set S of points in k-dimensional space (we assume k is an arbitrary constant) and want to keep track of the closest pair of points in S as S is being modified by insertions and deletions. Distances are measured in the L_t-metric, where t is fixed, $1 \leq t \leq \infty$.

The precursor to this problem is the classical *closest-pair problem* which is to compute the closest pair of points in a static set S, $|S| = n$. Shamos and Hoey [12] and Bentley and Shamos [2] gave $O(n \log n)$ time algorithms for solving the closest-pair problem in the plane and in arbitrary but fixed dimension, respectively. This running time is optimal in the algebraic decision tree model [1]. If we allow randomization as well as the use of the (non-algebraic) floor function, we find algorithms with better (expected) running times for the closest-pair problem. Rabin, in his seminal paper [9] on randomized algorithms, gave an $O(n)$ expected time algorithm for this problem. A different approach, leading to a simpler algorithm also with $O(n)$ expected running time, was recently described by Khuller and Matias [7]. A randomized "sieving" procedure described in this paper is at the heart of our dynamic algorithm.

There has been a lot of work on maintaining the closest pair of a dynamically changing set of points. When restricted to the case where only insertions of points are allowed (sometimes known as the *on-line closest-pair problem*) a series of papers culminated in an optimal data structure due to Schwarz, Smid and Snoeyink [11]. Their data structure required $O(n)$ space and supported insertions in $O(\log n)$ time.

The existing results are not as satisfactory when deletions must be performed. If only deletions are to be performed, Supowit [15] gave a data structure with $O(\log^k n)$ amortized update time that uses $O(n \log^{k-1} n)$ space. When both insertions and deletions are allowed, Smid [14] described a data structure that uses $O(n \log^k n)$ space and runs in $O(\log^k n \log\log n)$ amortized time per update. Another data structure due to Smid [13], with improvements stemming from results of Salowe [10] and Dickerson and Drysdale [5], uses $O(n)$ space and requires $O(\sqrt{n} \log n)$ time for updates; this is the best linear-space data structure currently known for insertions and deletions.

In this paper we discuss a randomized data structure, the *sparse partition*, which solves the dynamic closest pair problem in arbitrary fixed dimension using $O(\log n)$ expected time per update. The data structure needs $O(n)$ expected space. We assume that the updates are generated by an adversary who can insert or delete arbitrary points but has no knowledge of the random choices that the algorithm makes. The above bound is obtained assuming the use of the floor function and assuming that there is some prior bound on the size of the points (in order to make possible the use of hashing). If we want to dispense with hashing, then the algorithm can be modified to run in $O(\log n \log\log n)$ expected time per update. If we remove both assumptions, we obtain an algorithm with $O(\log^2 n)$ expected

*INRIA-Rocquencourt, 78153 Le Chesnay Cedex, France, and Hongkong UST. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II) and by NSF grant CCR-8918152. This work was done while this author was visiting the Max-Planck-Institut für Informatik.

†Max-Planck-Institut für Informatik, W-6600 Saarbrücken, Germany. These authors were supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Simple Randomized Algorithms for
Closest Pair Problems

M. Golin R. Raman C. Schwarz M. Smid

MPI–I–92–155 December 1992

INFORMATIK

Im Stadtwald
W 6600 Saarbrücken
Germany

Approach:

- Incremental computation
- Current minimal distance δ
- Consider next $p \in \mathcal{P} \rightarrow$ update δ if necessary
- Sparsity $\Rightarrow \mathcal{O}(1)$ many candidates $q \in \mathcal{P}$

Issue:

- Efficiently finding $q \in \mathcal{P}$.

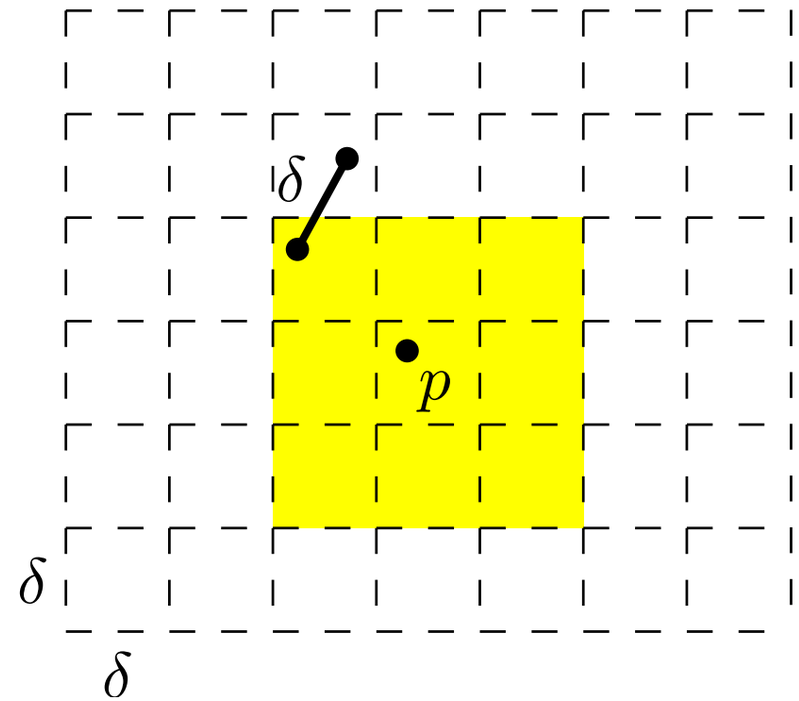
Problem:

- Processing unsorted sequence $\Rightarrow$ Points cannot be deleted from sequence.

$\Rightarrow$ Runtime could be bad…?!

Idea [Golin et al., 1993]:

- Cover $\mathbb{R}^2$ with an infinite grid G_δ of width δ.
- Each (half-) open grid cell stores all previously considered points in it.
- Denote cells with column/row (x, y) as $G_\delta[x, y]$.



- W.l.o.g.: $G_\delta[0, 0]$ anchored at origin
- For $(x, y) \in \mathbb{Z}^2$ let $N_\delta(x, y) := \bigcup_{\zeta=x-1,x,x+1;\nu=y-1,y,y+1} G_\delta[\zeta, \nu]$.

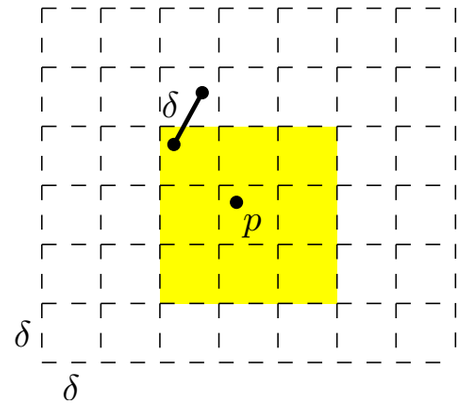
Lemma

For $p, q \in \mathcal{P}$ with $p \in G_\delta[x, y]$: $[d(p, q) \leq \delta \Rightarrow q \in N_\delta(x, y)]$

point p_{i+1} is located. Then every point in S_i that is within distance $\delta(S_i)$ of p_{i+1} must be located in one of the 9 grid boxes that are adjacent to b. (We consider the box b as being adjacent to itself.) We call these 9 boxes the *neighbors* of b.

Using the grid:

- For $(x, y) \in \mathbb{Z}^2$ let $N_\delta(x, y) := \bigcup_{\zeta=x-1,x,x+1;\nu=y-1,y,y+1} G_\delta[\zeta, \nu]$.



- Sparsity $\Rightarrow N_\delta(x, y)$ contains $\mathcal{O}(1)$ many points with pairwise distance $\geq \delta$.
- For current $p \in \mathcal{P}$: $\mathcal{O}(1)$ many distance computations suffice.

Updating the grid:

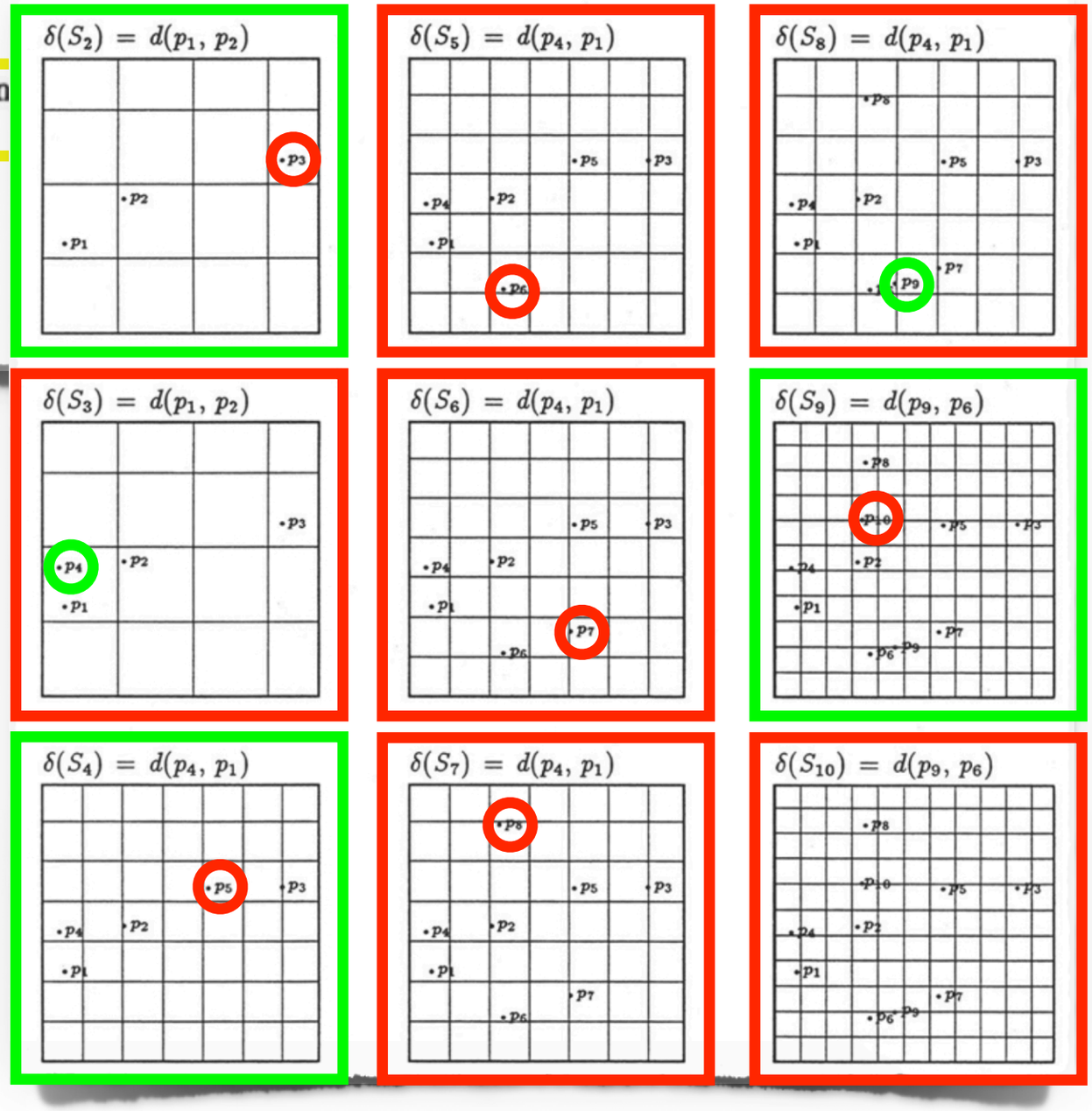
- p belongs to a pair with minimal distance $\Rightarrow$ rebuild grid.

„Black Box“

- Representing the grid / assignment: „Point $\rightarrow$ cell“

Algorithm $CP(p_1, p_2, \ldots, p_n)$

(1) $\delta := d(p_1, p_2)$; $\mathcal{G} := Build(S_2, \delta)$;
(2) **for** $i := 2$ **to** $n - 1$ **do**
(3) **begin**
(4) $V := \{Report(\mathcal{G}, b) : b$ is a neighbor of the box contain
(5) $d := \min_{q \in V} d(p_{i+1}, q)$;
(6) **if** $d > \delta$ **then** $Insert(\mathcal{G}, p_{i+1})$
(7) **else** $\delta := d$; $\mathcal{G} := Build(S_{i+1}, \delta)$;
(8) **end**;
(9) **return**(δ).



- Cell representation $(x, y) \in \mathbb{N}^2$
- G as OrderedDictionary
- Key of $p \in \mathcal{P}$: $(p.x, p.y)$.

Computing the key value:

- Assignment:

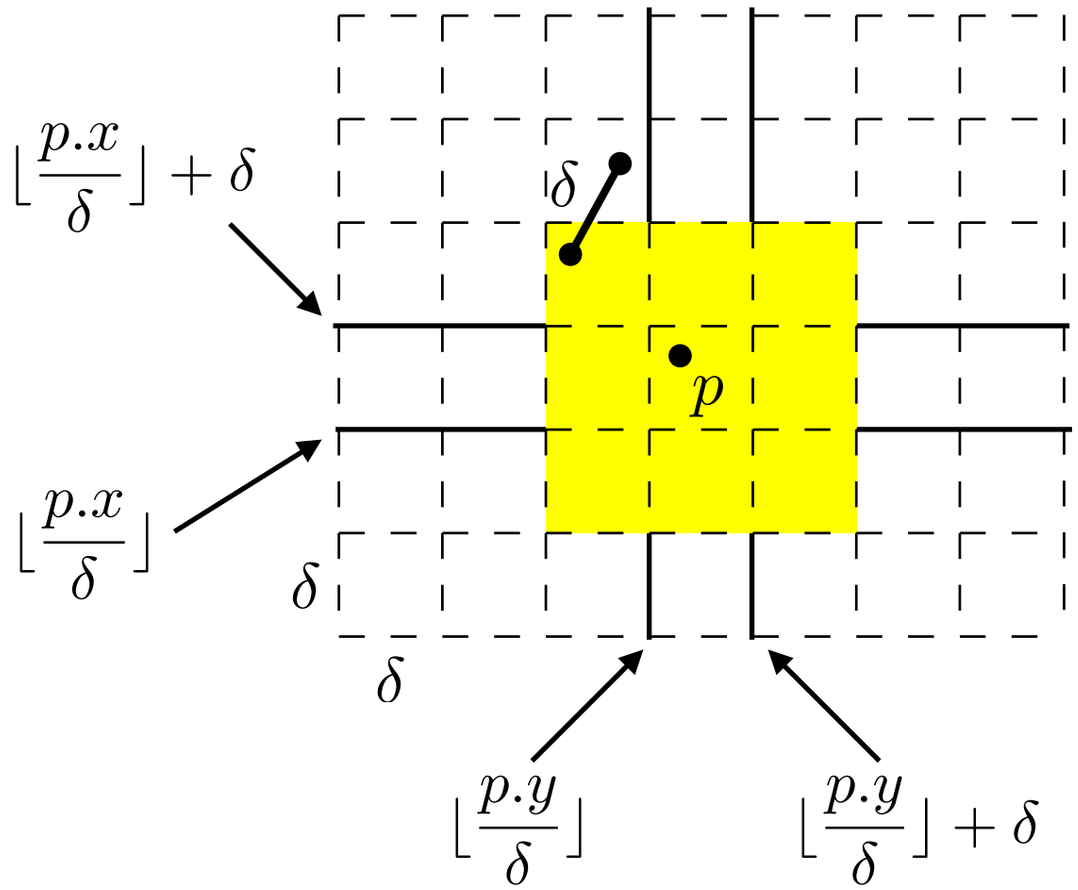
$$p \in G_\delta[x, y] \Leftrightarrow x = \lfloor \frac{p.x}{\delta} \rfloor, y = \lfloor \frac{p.y}{\delta} \rfloor$$

Note:

- Operation $\lfloor \cdot \rfloor$

Implementing the grid:

- Data structure:
  - $T_{build}(n)$: Construction cost
  - $T_{insert}(n)$: Insertion cost
  - $T_{query}(n)$: Query cost



- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence
- $\mathcal{P}_i := \{p_0, \ldots, p_{i-1}\}$

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence
- $\mathcal{P}_i := \{p_0, \ldots, p_{i-1}\}$
- $\delta_i := \min\{d(p_j, p_k) \mid 0 \leq j < k \leq i-1\}$

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence
- $\mathcal{P}_i := \{p_0, \ldots, p_{i-1}\}$
- $\delta_i := \min\{d(p_j, p_k) \mid 0 \leq j < k \leq i-1\}$

Randomization:

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence
- $\mathcal{P}_i := \{p_0, \ldots, p_{i-1}\}$
- $\delta_i := \min\{d(p_j, p_k) \mid 0 \leq j < k \leq i-1\}$

Randomization:

- Expected runtime?

- Construction cost: $\in \Omega(n), \mathcal{O}(n^2)$.

Terminology:

- $(p_0, \ldots, p_{n-1})$:= permuted sequence
- $\mathcal{P}_i := \{p_0, \ldots, p_{i-1}\}$
- $\delta_i := \min\{d(p_j, p_k) \mid 0 \leq j < k \leq i-1\}$

Randomization:

- Expected runtime?
- Random variable:

$$X(p_{i-1}, \mathcal{P}_{i-1}) := \begin{cases} 1, & \text{if} \quad \delta_i < \delta_{i-1} \\ 0, & \text{else} \end{cases}$$

Runtime:

- $T(i)$: cost for $\mathcal{P}_{i-1} \rightarrow \mathcal{P}_i$

$$T(i) \in \mathcal{O}\left(T_{insert}(i-1) + T_{query}(i-1) + X(p_{i-1}, \mathcal{P}_{i-1}) \cdot T_{build}(i)\right)$$

- Random permutation:

$$\Rightarrow E[T(i)] \in \mathcal{O}\left(E[T_{insert}(i-1)] + E[T_{query}(i-1)] + P\left(\{\delta_1 < \delta_{i-1}\}\right) \cdot E[T_{build}(i)]\right)$$

Lemma 1 *Let $p_1, p_2, \ldots, p_n$ be a random permutation of the points of S. Let $S_i := \{p_1, p_2, \ldots, p_i\}$. Then $\Pr[\delta(S_{i+1}) < \delta(S_i)] \leq 2/(i+1)$.*

Lemma: $P(\{\delta_i < \delta_{i-1}\}) \leq \frac{2}{i}$

Proof:

- Backward analysis: Let $p = \mathcal{P}_i \setminus \mathcal{P}_{i-1}$
- $S_i := \{r \in \mathcal{P}_i \mid \exists q \in \mathcal{P}_i : r \neq q, d(q,r) = \delta_i\}$
- Grid rebuilt $\Leftrightarrow \delta_i < \delta_{i-1}$
- If $|S_i| = 2$: $P(\{\delta_i < \delta_{i-1}\}) = P(\{p \in S_i\}) = \frac{2}{i}$
- If $|S_i| > 2$:
  - Case 1: All pairs $q, r \in \mathcal{P}_i$ with $d(q,r) = \delta_i$ share a unique point:

$$P(\{\delta_i < \delta_{i-1}\}) = P(p = s) = \frac{1}{i}$$

  - Case 2: Such a unique point does not exist:

$$P(\{\delta_i < \delta_{i-1}\}) = 0$$

□

- Expected cost:

$$\Rightarrow E[T(i)] \in \mathcal{O}\left(E[T_{insert}(i-1)] + E[T_{query}(i-1)] + \frac{2}{i} \cdot E[T_{build}(i)]\right)$$

- Implementing the grid:

Data structure	$T_{query}(n)$	$T_{insert}(n)$	$T_{build}(n)$
AVL tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n \log n)$
Dynamic perfect hashing	$\mathcal{O}(1)$ (exp.)	$\mathcal{O}(1)$ (exp.)	$\mathcal{O}(n)$ (exp.)

- Implementation as AVL tree:

$$\sum_{i=3}^{n} E[T(i)] \in \mathcal{O}\left(\sum_{i=3}^{n}\left(\log i + \log i + \frac{i \log i}{i}\right)\right) = \mathcal{O}(n \log n)$$

- Implementation as dynamic perfect hashing:

$$\sum_{i=3}^{n} E[T(i)] \in \mathcal{O}\left(\sum_{i=3}^{n}\left(1 + 1 + \frac{i}{i}\right)\right) = \mathcal{O}(n)$$

Theorem 1 *Let S be a set of n points in D-space, and let $1 \leq t \leq \infty$.*

1. *The implementation of the algorithm that uses a binary tree finds a closest pair in S, in $O(n \log n)$ expected time. This implementation uses the floor function and randomization, and it works for any set S.*

2. *The implementation of the algorithm that uses dynamic perfect hashing finds a closest pair in S, in $O(n)$ expected time. This implementation uses the floor function and randomization, and it works for any set S for which the values $\lfloor p^{(j)}/\delta(S) \rfloor$, $p = (p^{(1)}, p^{(2)}, \ldots, p^{(n)}) \in S$ and $1 \leq j \leq D$, come from a finite universe.*

Thank you!