




<p>BRUTE-FORCE SOLUTION: <math>O(n!)</math></p> 	<p>DYNAMIC PROGRAMMING ALGORITHMS: <math>O(n^2 2^n)</math></p> 	<p>Algorithmus 4.3 Greedy          Eingabe: <math>z_1, \dots, z_n, Z, p_1, \dots, p_n</math>          Ausgabe: <math>S \subseteq \{1, \dots, n\}</math>  <math>G_0 = \max_{S \subseteq \{1, \dots, n\}} \sum_{i \in S} p_i</math>          return <math>S</math></p>	
---	---	--	---

# 4 Approximation

*Algorithmen und Datenstrukturen 2  
Sommer 2022*

**Prof. Dr. Sándor Fekete**

# 4.1 Motivation

# Optimierungsprobleme

# Optimierungsprobleme

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
	8	32	11	12	13
	5	5	28	20	16
7	40	9	3	9	24
32	9	8	14	15	4
2	2	20	3	40	
		3	15	10	

Kann Knut die Klausur bestehen?

# Optimierungsprobleme

## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	9	8
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

# Optimierungsprobleme

## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	9	8
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

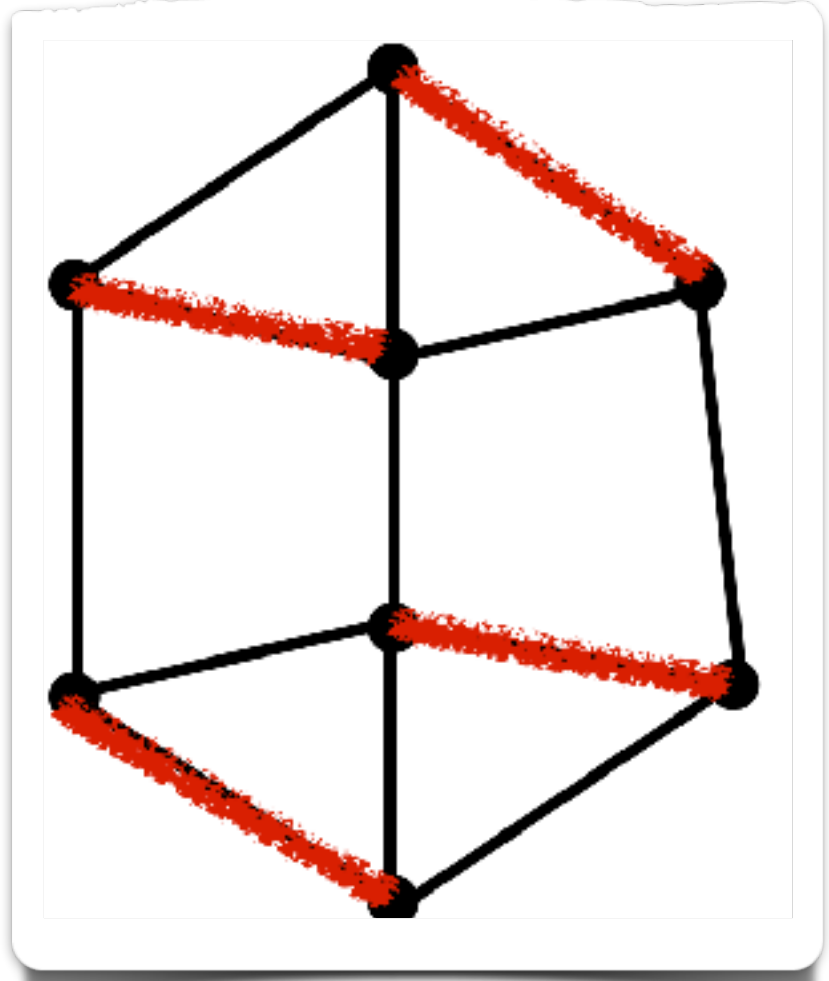
**Max**

# Optimierungsprobleme

## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
3	8	32	11	12	13
40	5	28	20	16	16
9	9	3	9	24	10
32	8	14	15	4	
2	2	20	40		
		3	10		

Kann Knut die Klausur bestehen?



**Max**

# Optimierungsprobleme

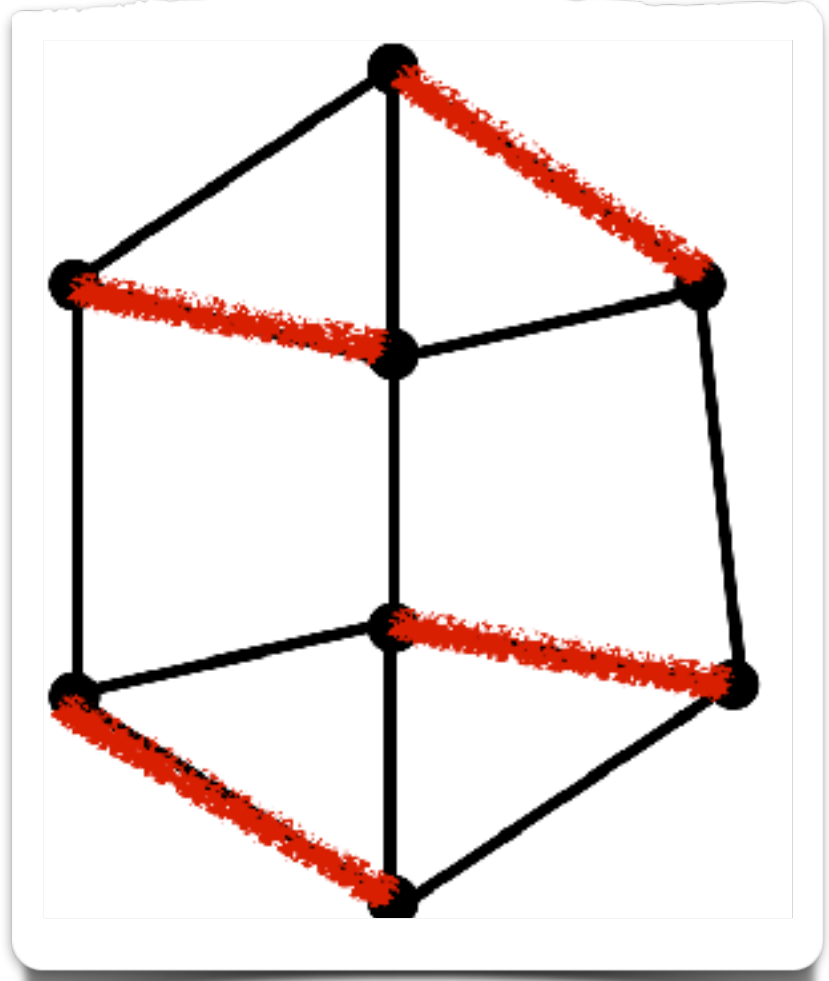
## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	11	12	13	16
40	32	28	20	16	16
9	5	3	9	24	10
32	9	14	15	4	
2	2	20	40		
		3	10		

Kann Knut die Klausur bestehen?

**Max**

## Matching





# Optimierungsprobleme

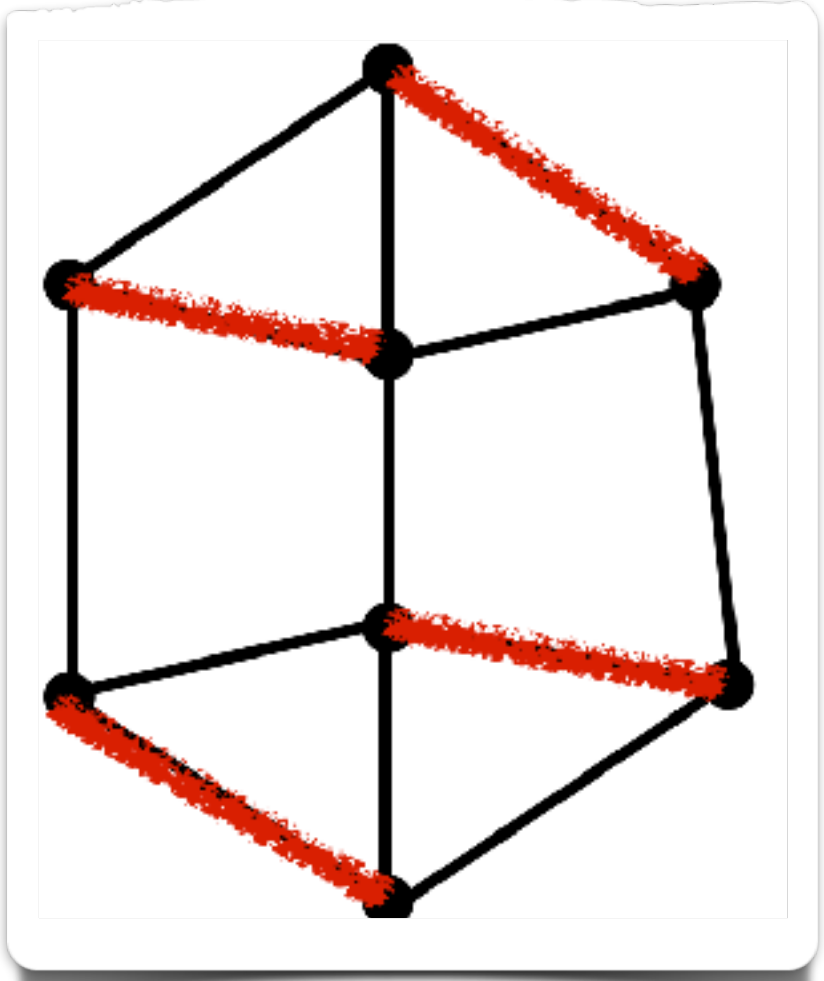
## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

**Max**

## Matching



**Max**

# Optimierungsprobleme

## 0-1 Knapsack

1	Nummer	2	3	4	5	6
20	Minuten	32	40	8	16	4
3	Punkte	3	10	5	2	4

8	10	11	12	13
40	32	28	20	16
9	5	3	9	24

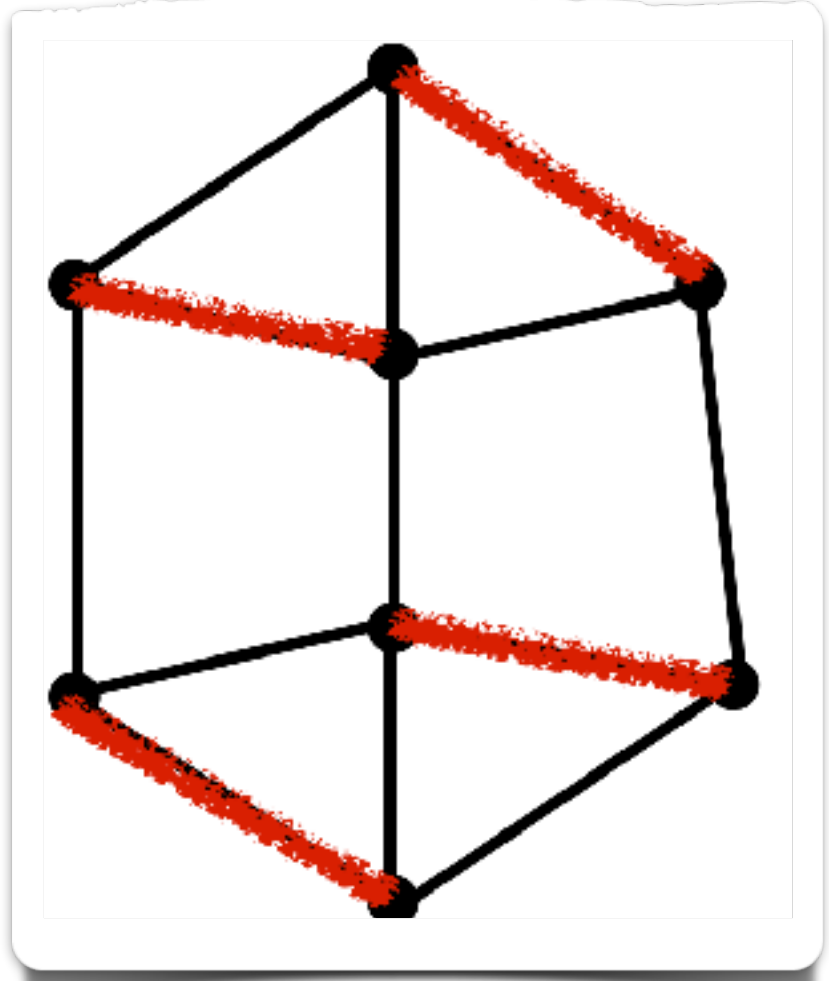
  

7	8	14	15	16
32	40	20	40	4
2	9	3	10	

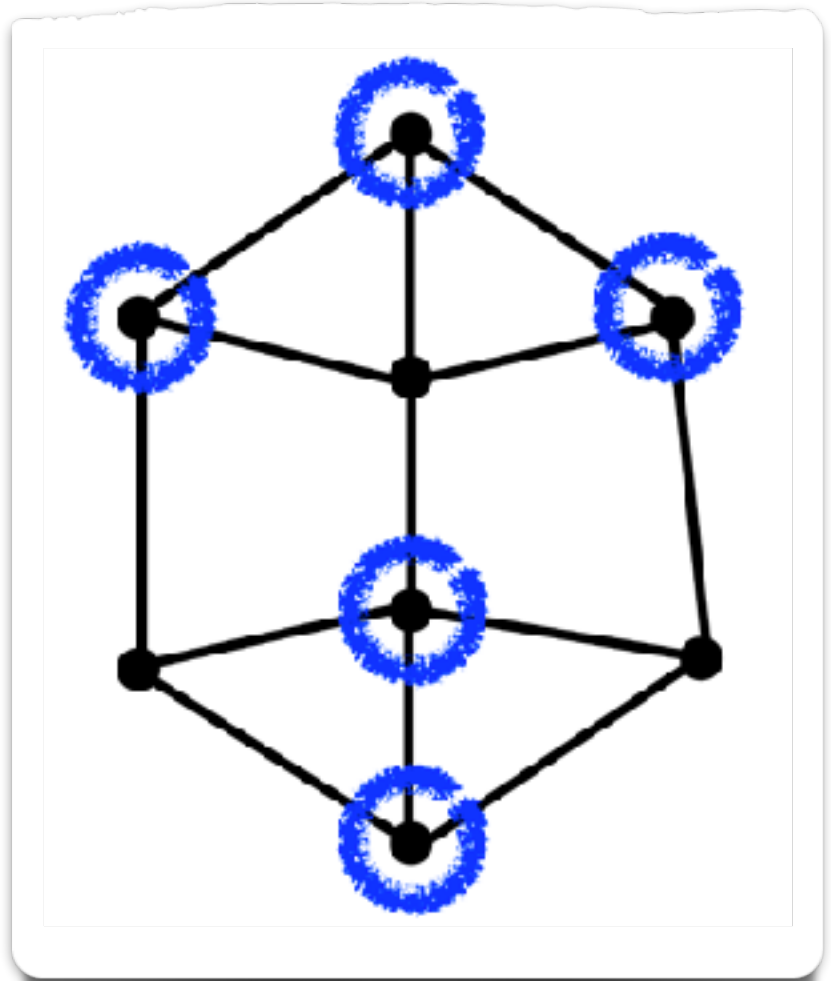
Kann Knut die Klausur bestehen?

**Max**

## Matching



**Max**



# Optimierungsprobleme

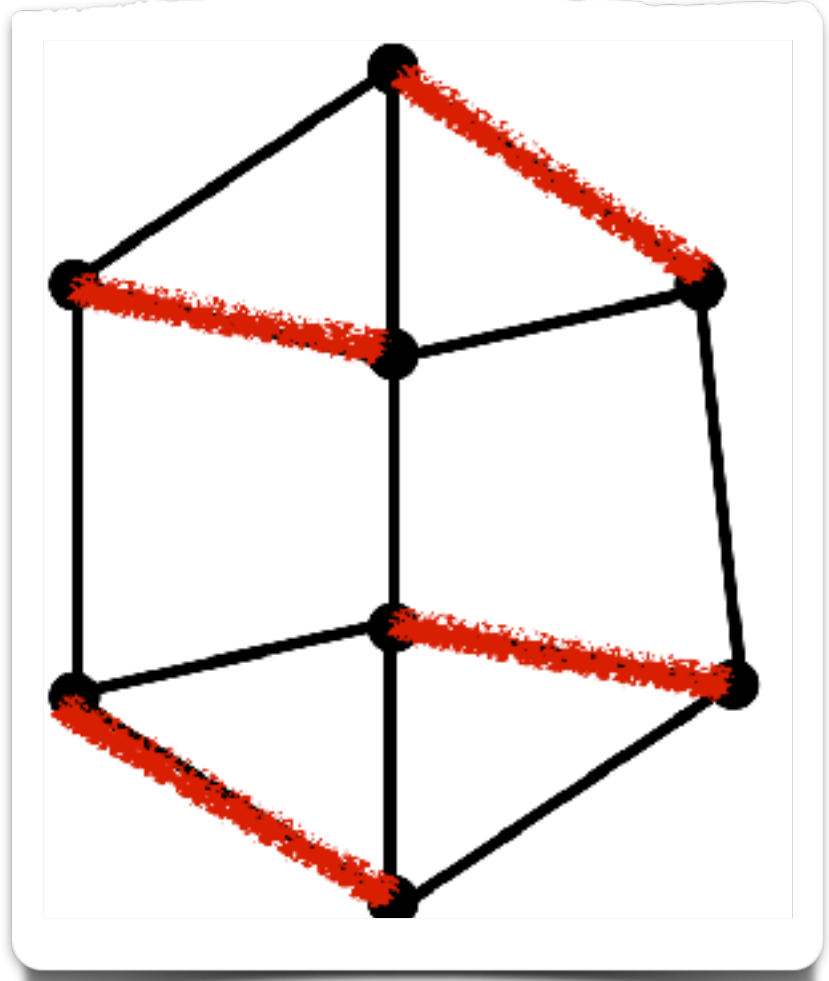
**0-1 Knapsack**

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

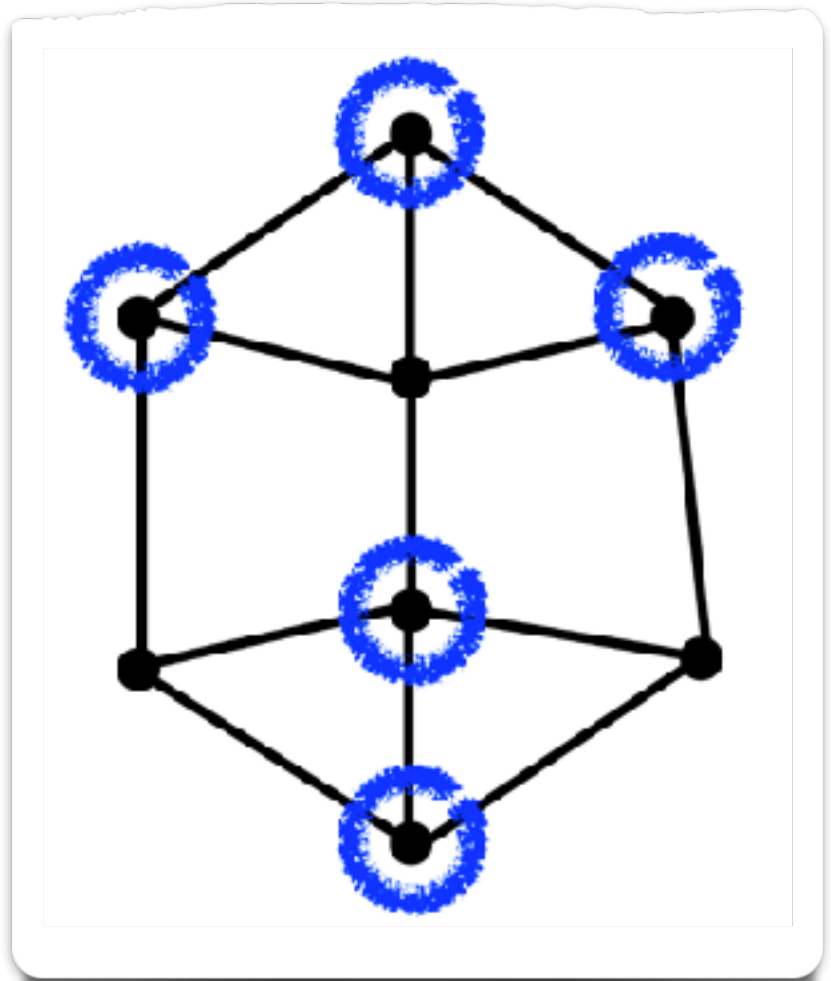
**Max**

**Matching**



**Max**

**Vertex Cover**



# Optimierungsprobleme

**0-1 Knapsack**

1	Nummer	2	3	4	5	6
20	Minuten	32	40	8	16	4
3	Punkte	3	10	5	2	4

7	8	9	10	11	12	13
32	40	8	32	28	20	16
2	9	5	5	3	9	24

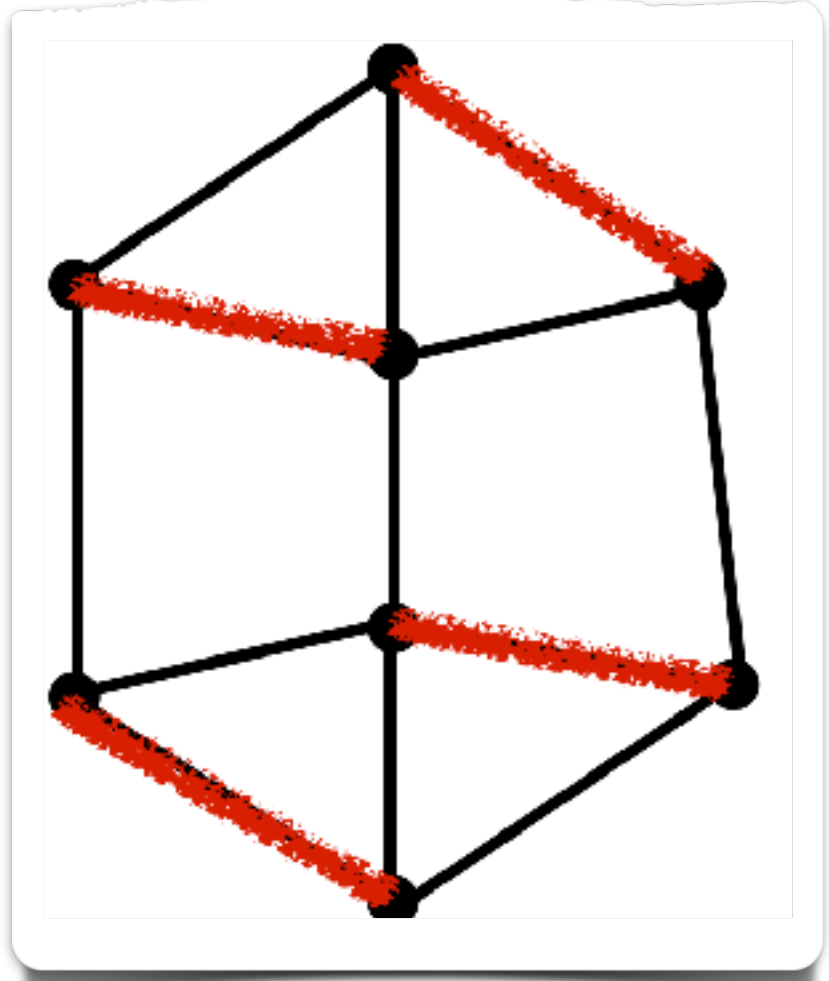
  

14	15	16	17
20	40	4	10
3	10		

Kann Knut die Klausur bestehen?

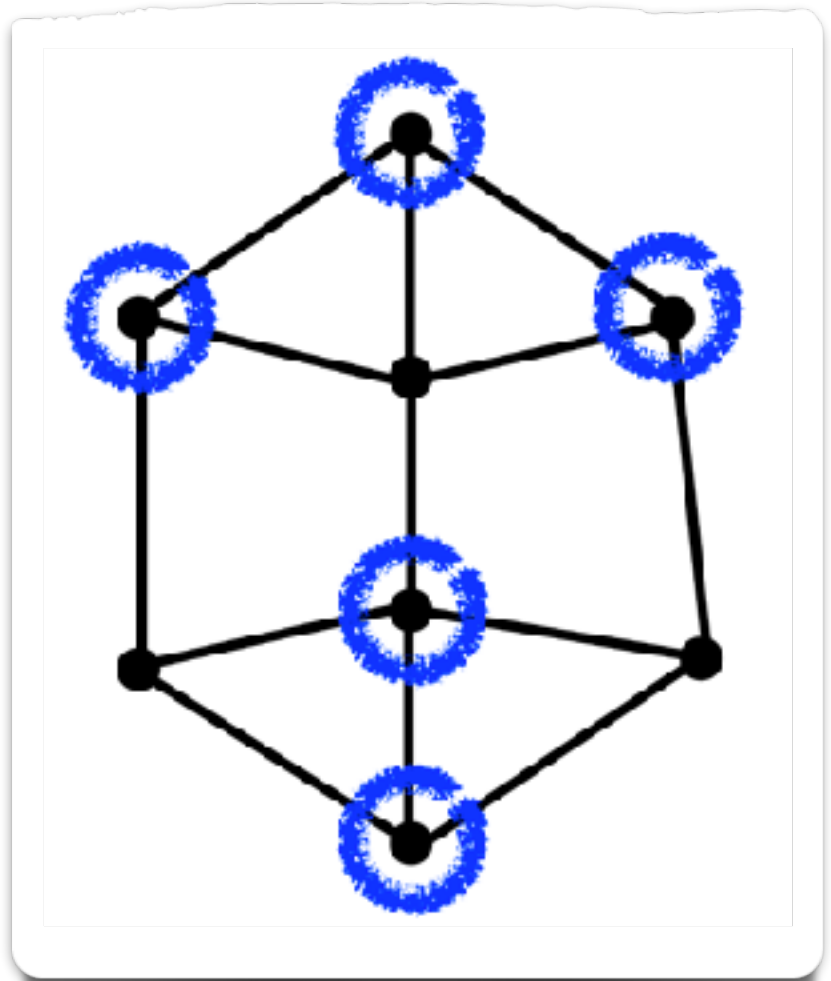
**Max**

**Matching**



**Max**

**Vertex Cover**



**Min**

# Optimierungsprobleme

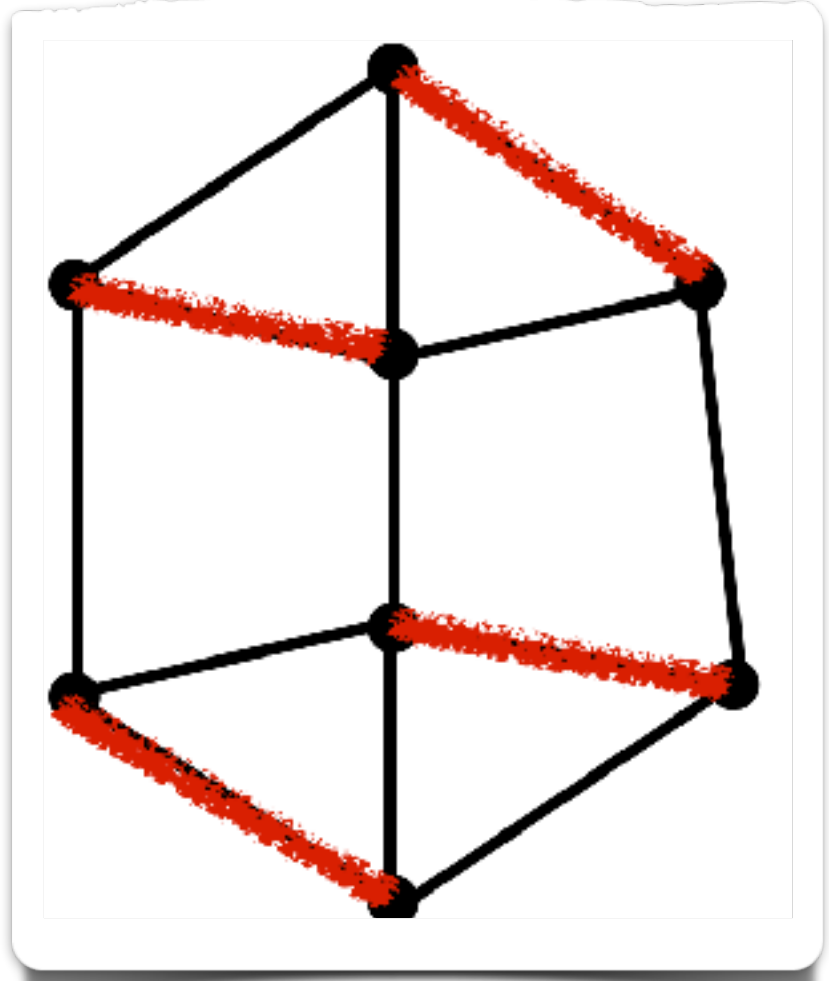
**0-1 Knapsack**

1	2	3	4	5	6
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
		10			
8	11	12	13		
40	28	20	16		
9	3	9	24		
32	14	8	4		
2	20	2	15		
	3		40		
			10		

Kann Knut die Klausur bestehen?

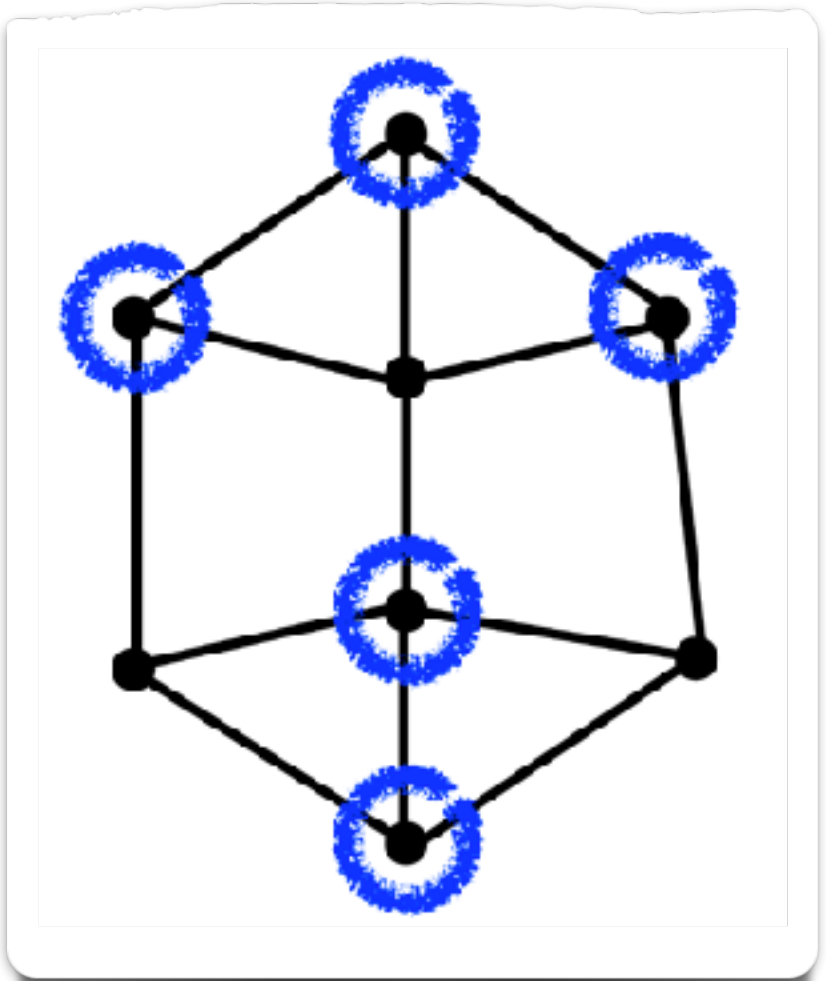
**Max**

**Matching**

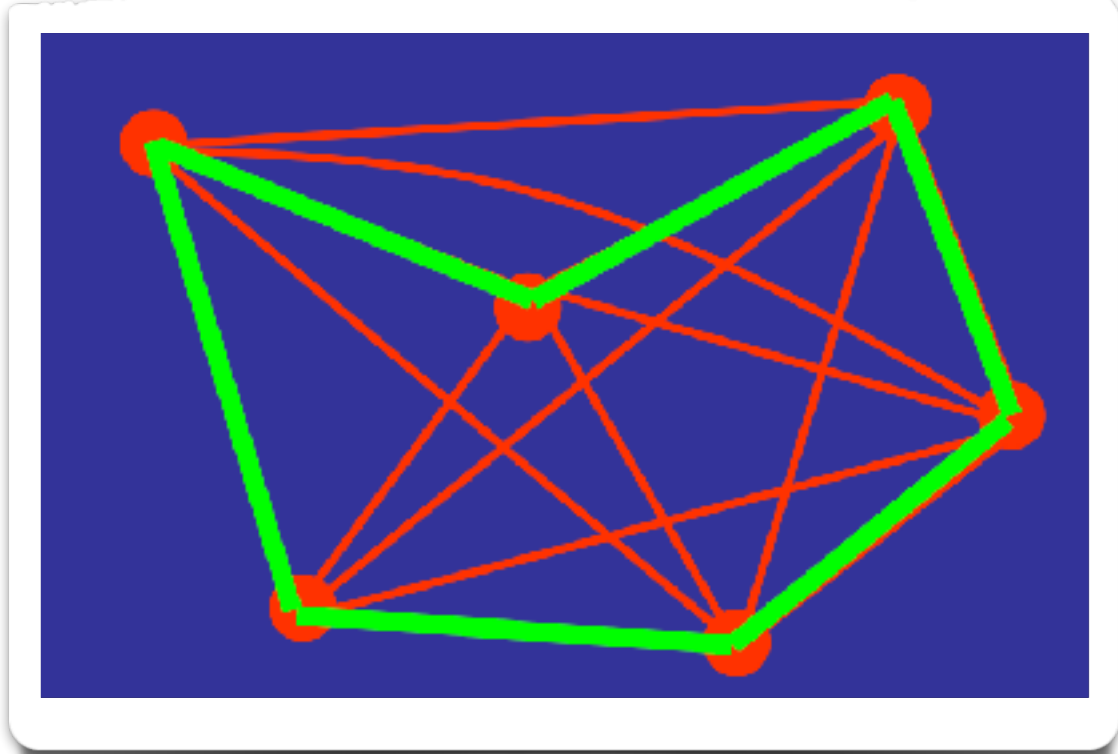


**Max**

**Vertex Cover**



**Min**



# Optimierungsprobleme

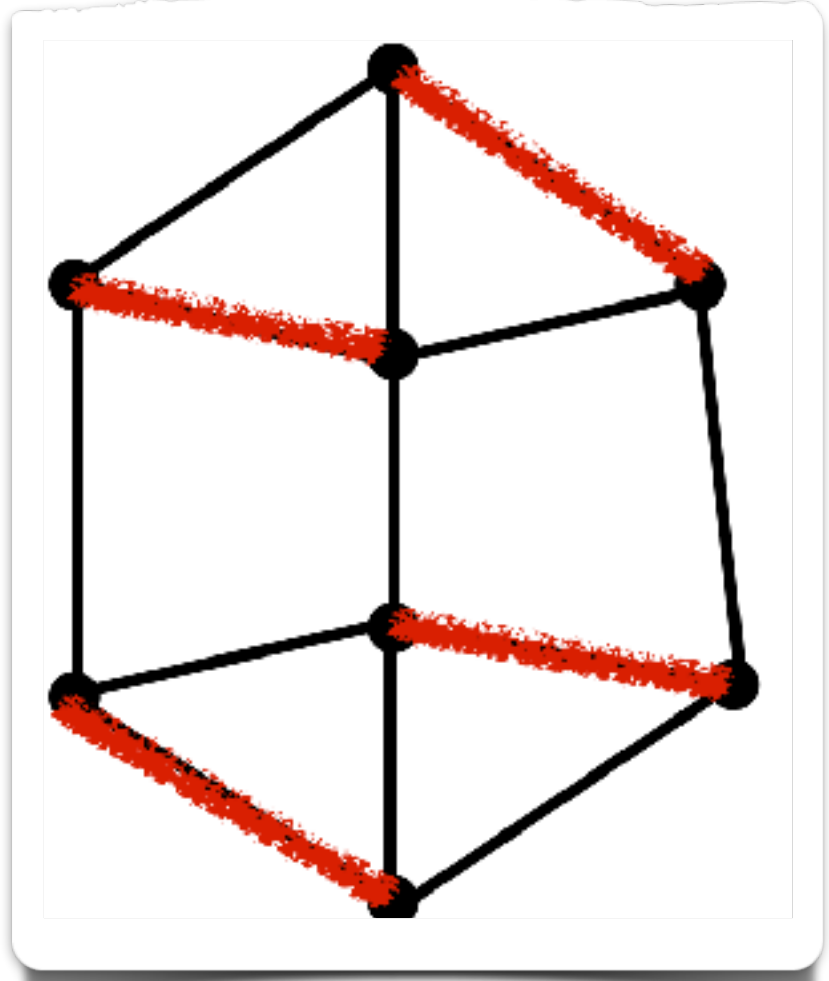
**0-1 Knapsack**

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	11	12	16	13
40	32	28	20	24	16
9	5	3	9	4	10
7	8	14	15	16	13
32	9	20	40	24	16
2	2	3	10	4	10

Kann Knut die Klausur bestehen?

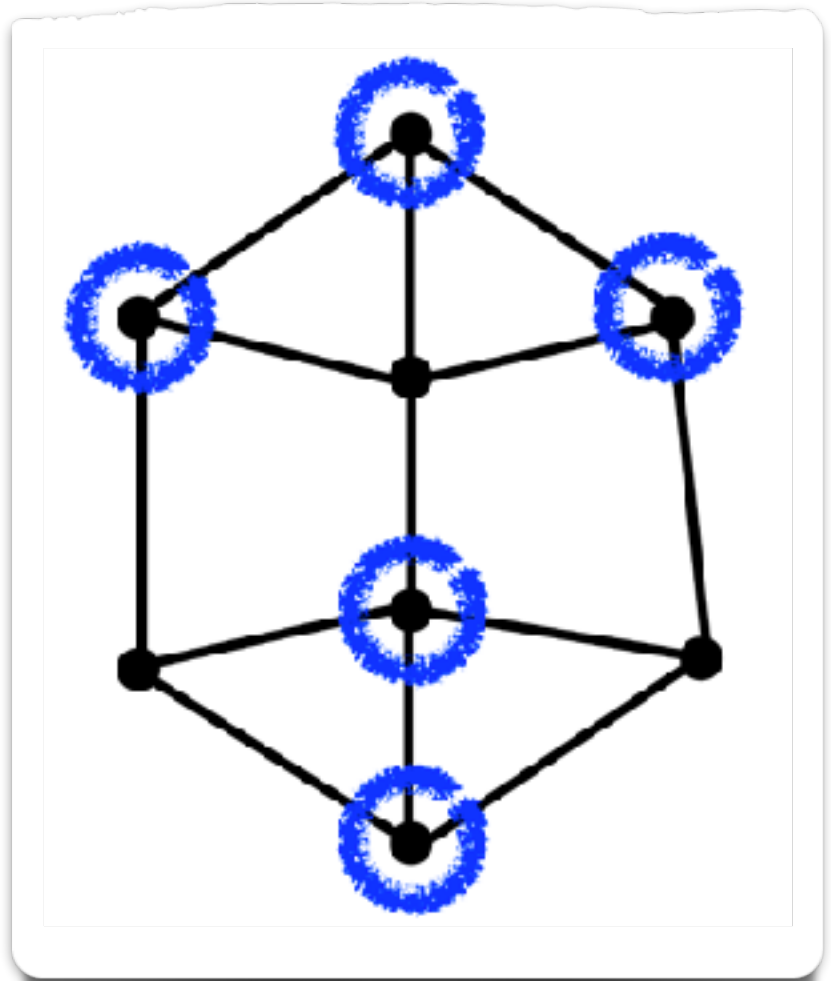
**Max**

**Matching**



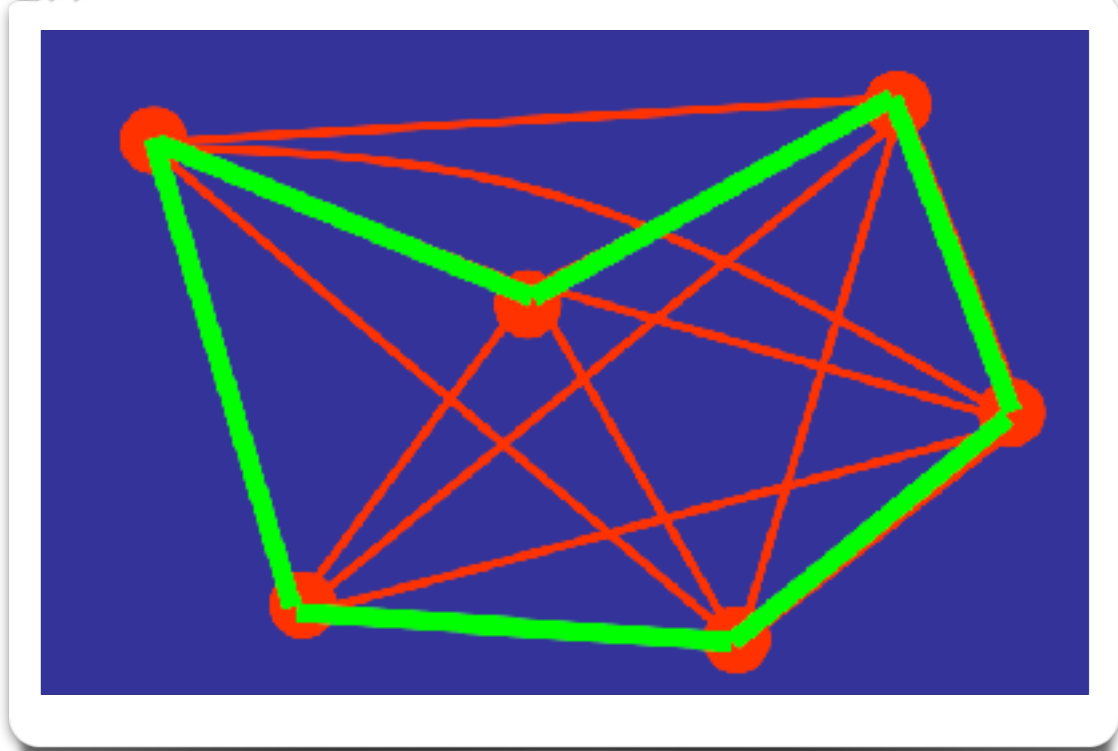
**Max**

**Vertex Cover**



**Min**

**Traveling Salesman**



# Optimierungsprobleme

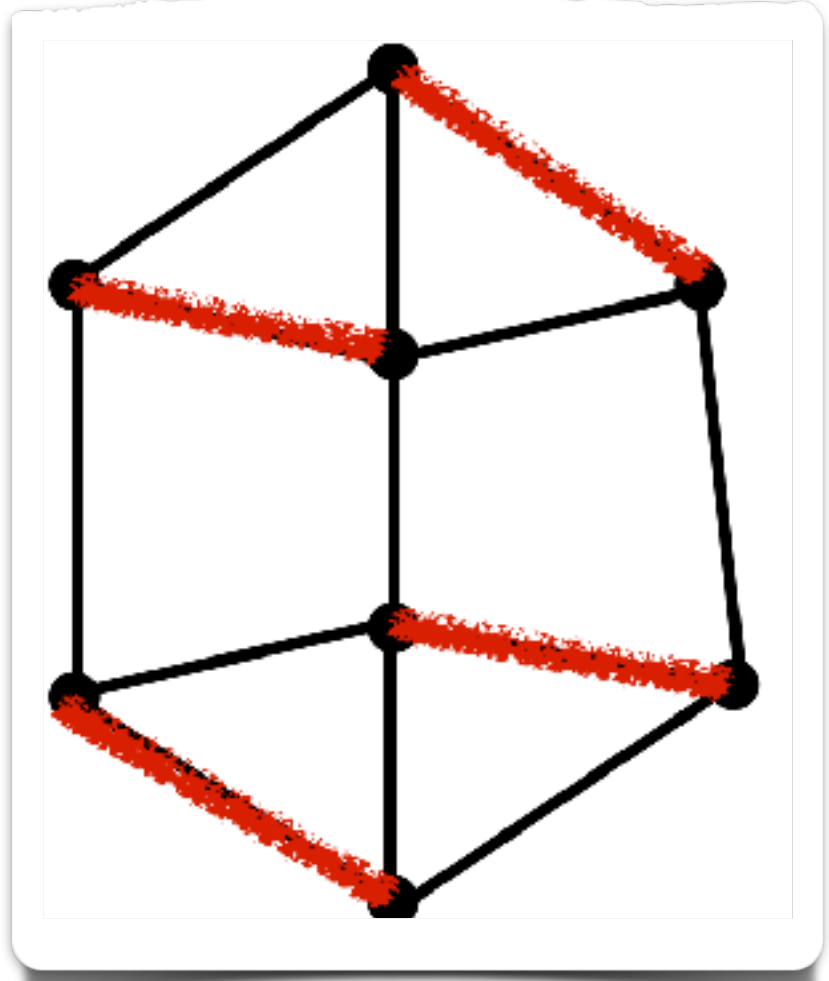
**0-1 Knapsack**

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	14	3	9	24	10
32	20	15	16	4	
2	3	40	10		

Kann Knut die Klausur bestehen?

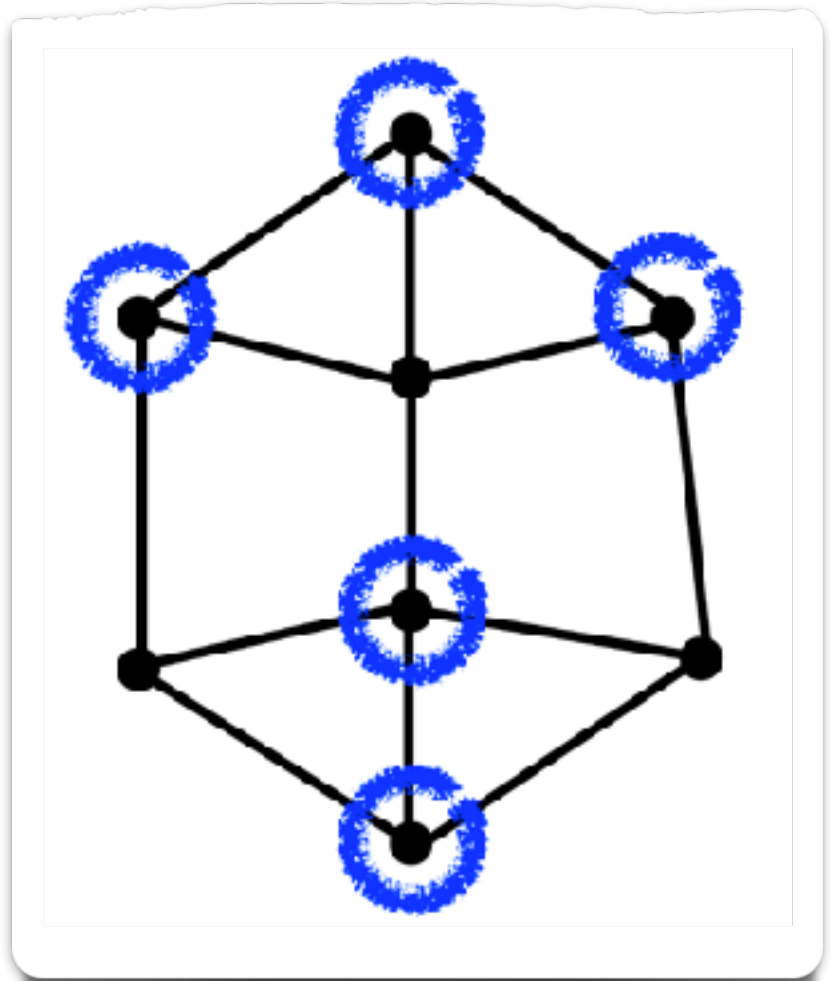
**Max**

**Matching**



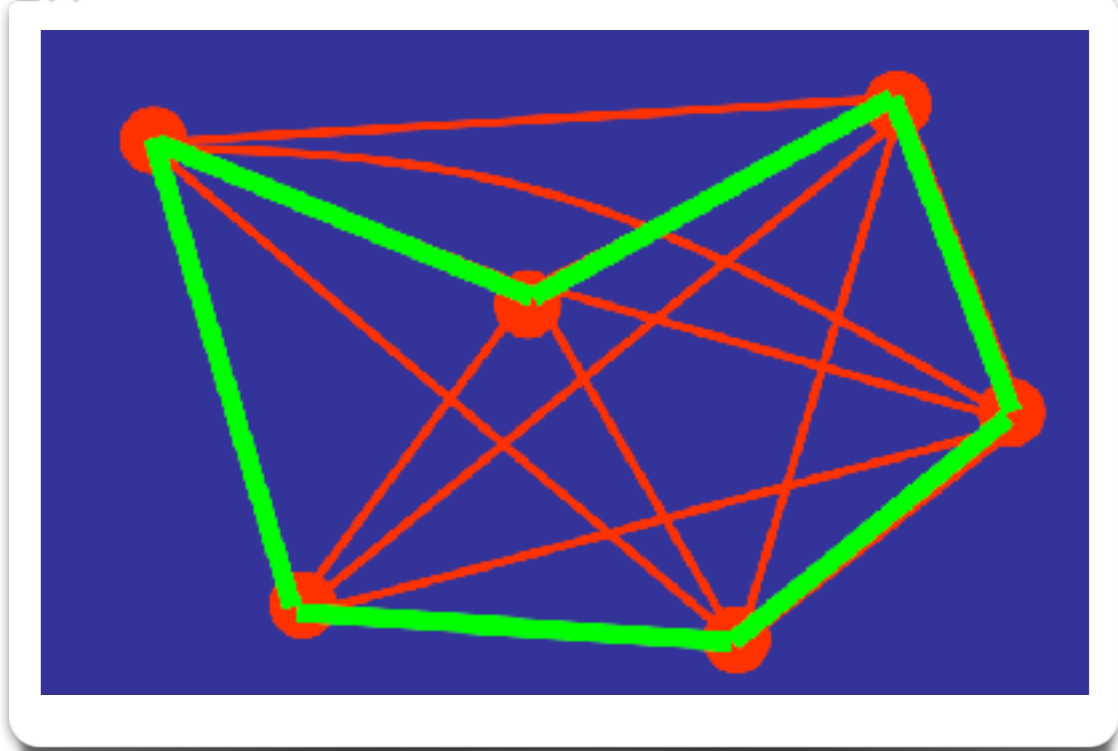
**Max**

**Vertex Cover**



**Min**

**Traveling Salesman**



**Min**

# Optimierungsprobleme

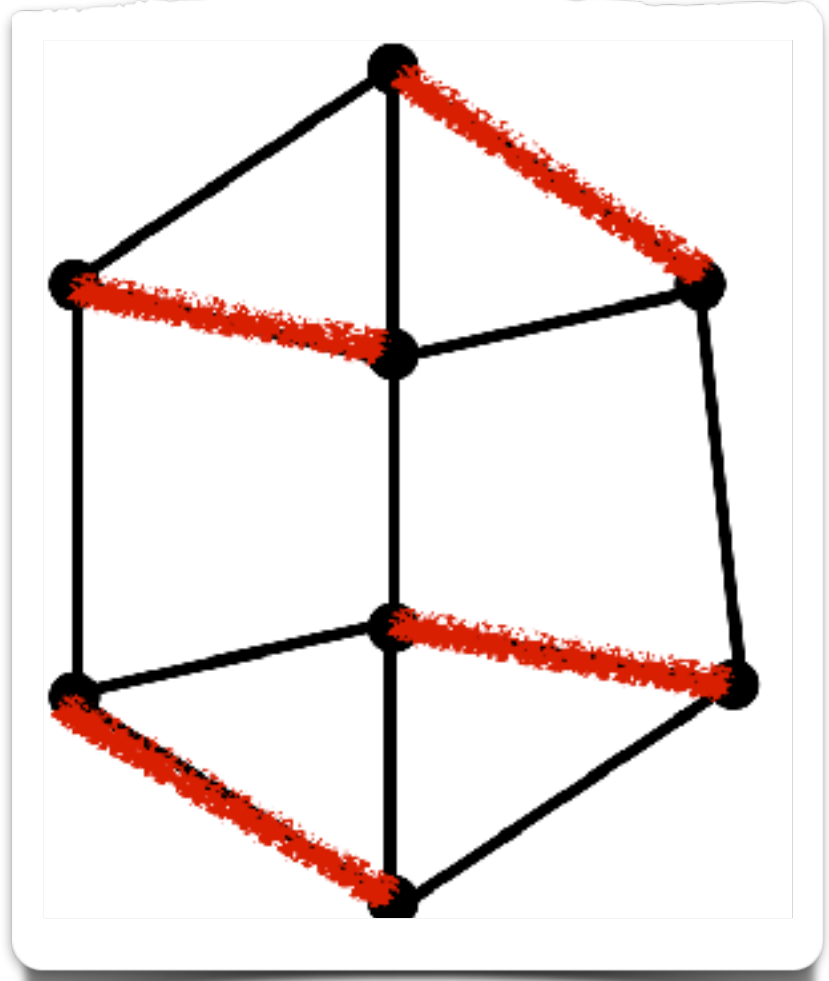
0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	9	3	9	24	10
32	8	14	15	4	
2	2	20	40		
		3	10		

Kann Knut die Klausur bestehen?

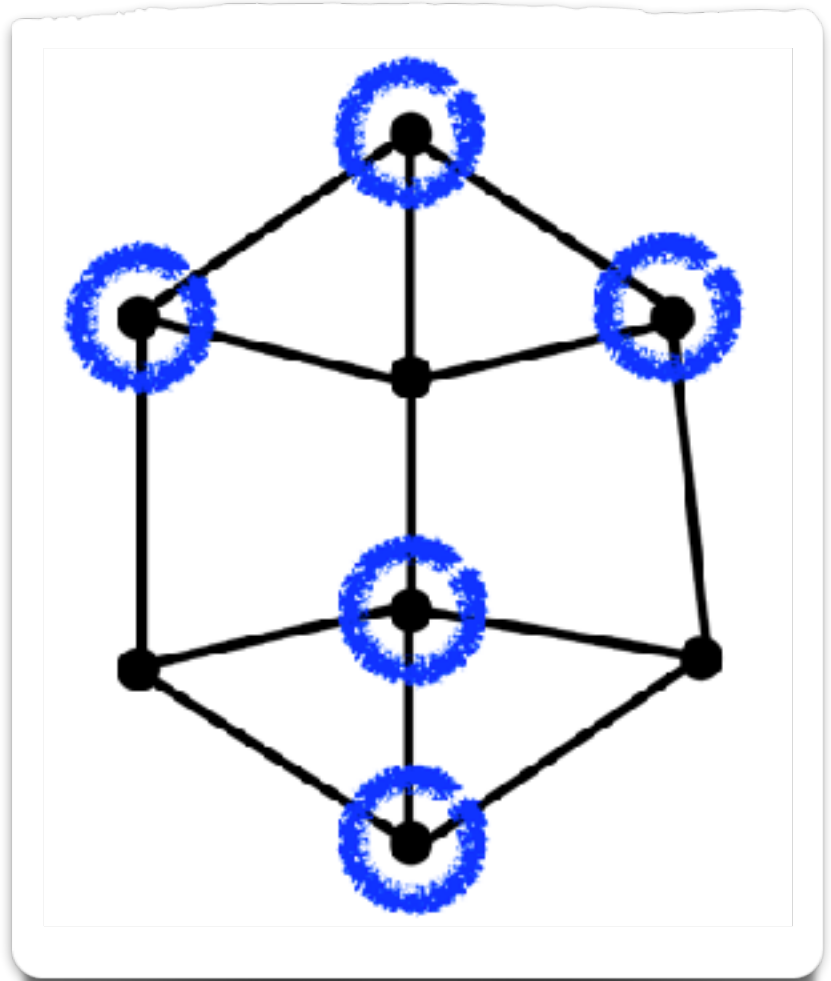
Max

Matching



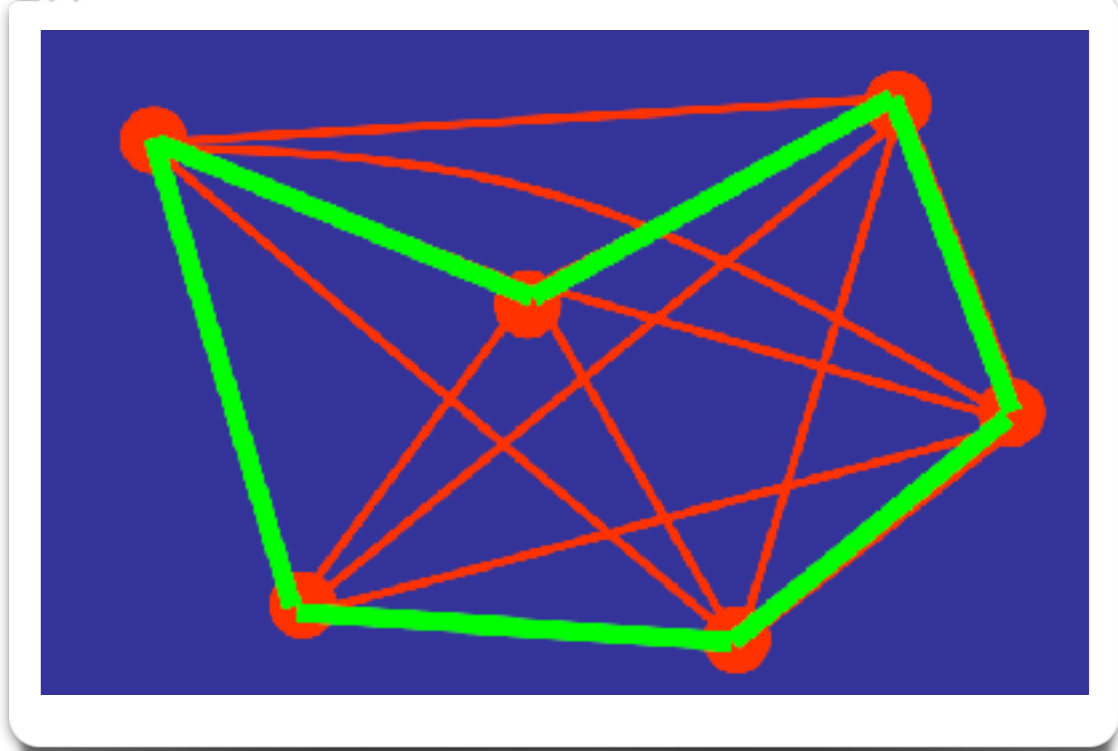
Max

Vertex Cover



Min

Traveling Salesman



Min

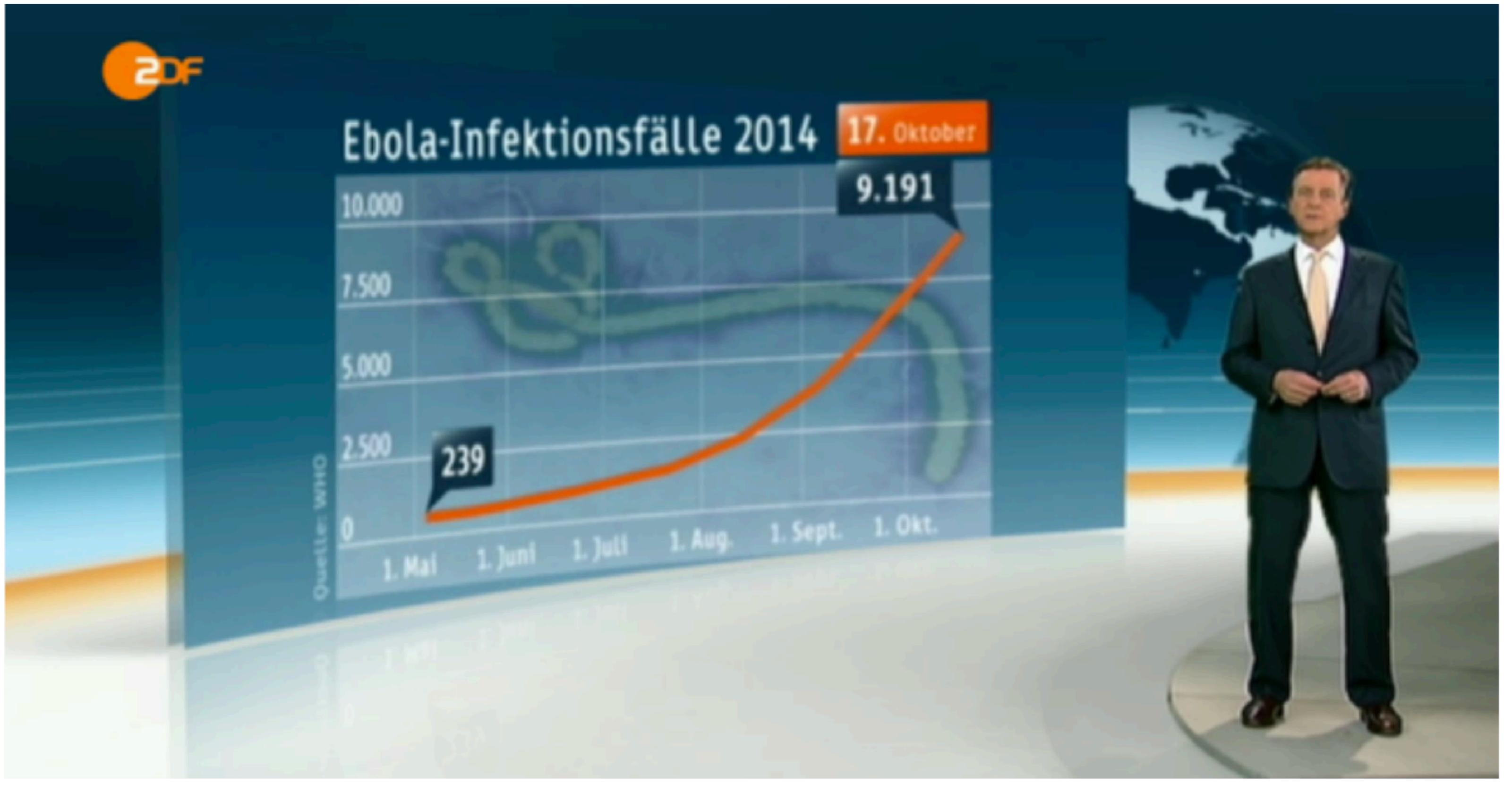
Ziel: Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!



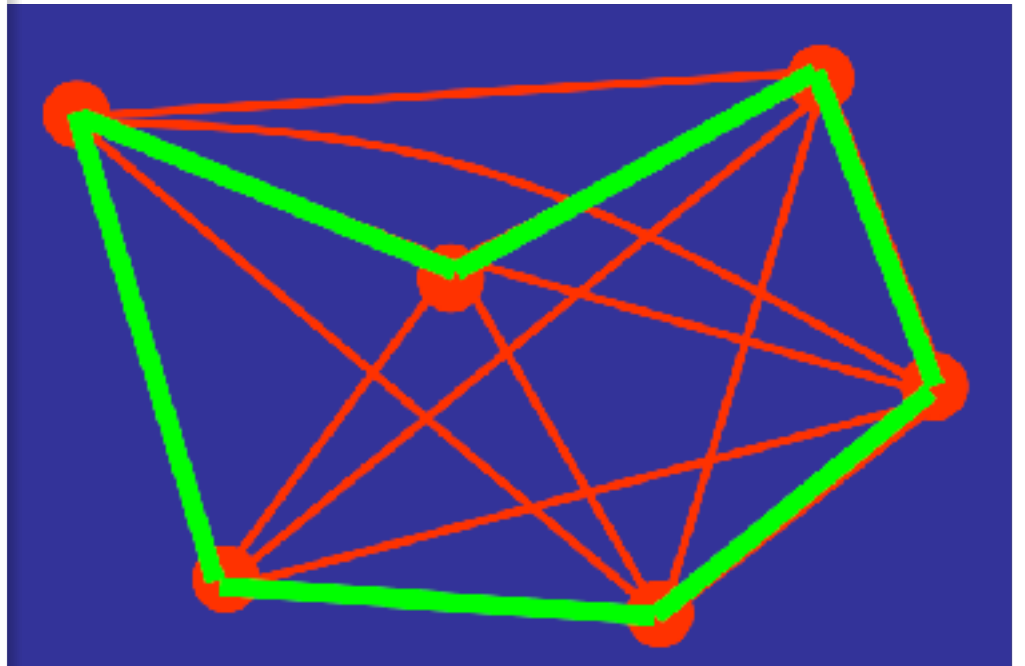
# Optimierungsprobleme

0-1

1	Nummer	2	
20	Minuten	32	
3	Punkte	3	10
			32
	8		5
7	40	9	
32	9	8	
2		2	



Traveling Salesman



Min

**Ziel:** Finde eine bestmögliche Lösung unter **exponentiell** vielen möglichen!

# Laufzeit?

## 0-1 Knapsack

1	Nummer	2	3	4	5	6
20	Minuten	32	40	8	16	4
3	Punkte	3	10	5	2	4

7	8	10	11	12	13
32	40	32	28	20	16
2	9	5	3	9	10

9	14	15	16
8	20	40	24
2	3	10	4

Kann Knut die Klausur bestehen?

# Laufzeit?

## 0-1 Knapsack

$$O(nZ)$$

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4

7	8	9	10	11	12	13
32	40	8	32	28	20	16
2	9	5	5	3	9	10

14	15	16
20	40	24
3	10	4

Kann Knut die Klausur bestehen?

# Laufzeit?

## 0-1 Knapsack

$$O(nZ)$$

$$O(2^n f(n))$$

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4

7	8	9	10	11	12	13
32	40	8	32	28	20	16
2	9	5	5	3	9	10

14	15	16
20	40	24
3	10	4

Kann Knut die Klausur bestehen?

# Laufzeit?

## 0-1 Knapsack

## Traveling Salesman

$$O(nZ)$$

$$O(2^n f(n))$$

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4

7	8	9	10	11	12	13
32	40	8	32	28	20	16
2	9	5	5	3	9	10

14	15	16
20	40	24
3	10	4

Kann Knut die Klausur bestehen?

# Laufzeit?

## 0-1 Knapsack

$O(nZ)$

$O(2^n f(n))$

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4

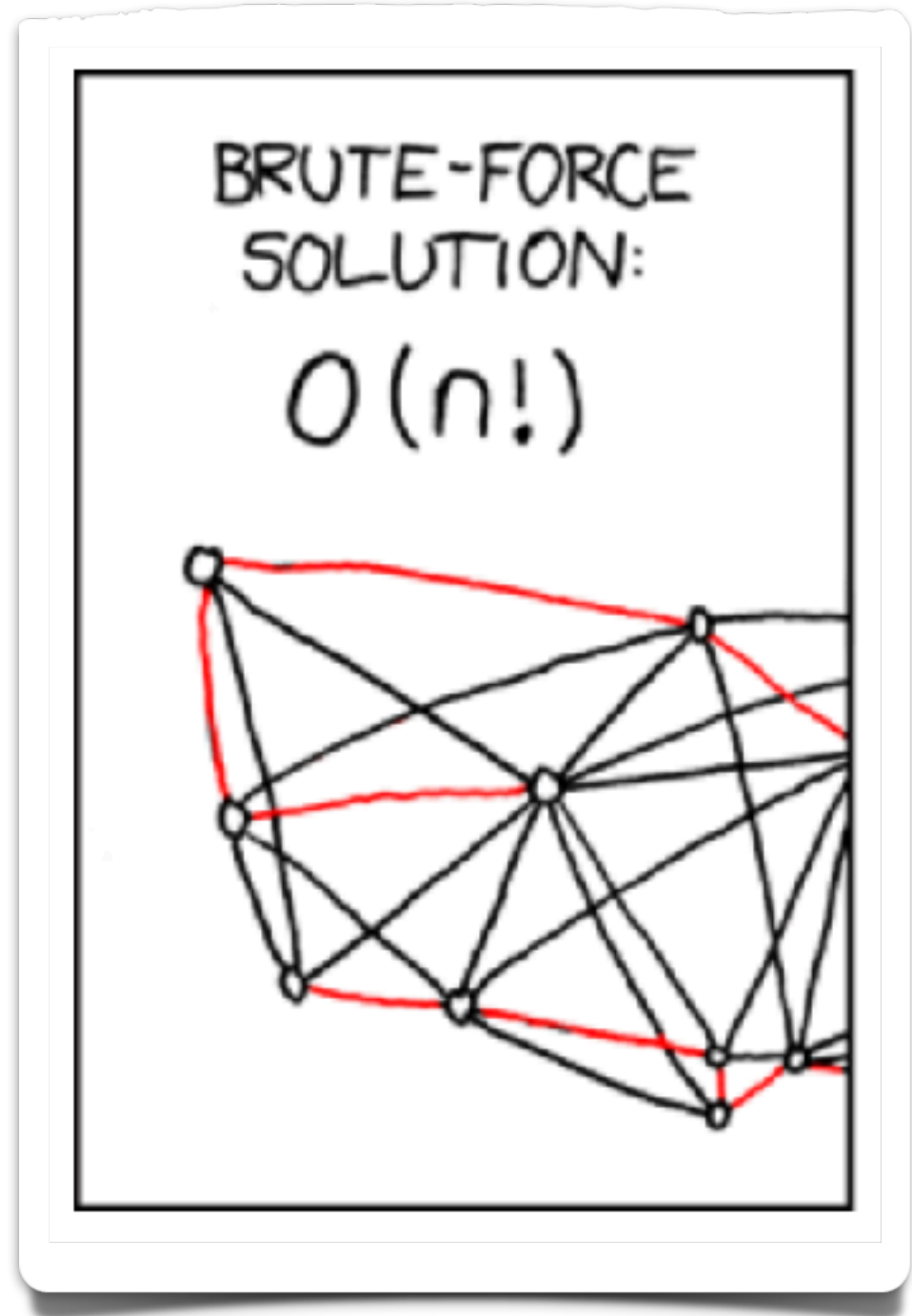
7	8	9	10	11	12	13
32	40	8	32	28	20	16
2	9	5	5	3	9	10

14	15	16
20	40	24
3	10	4

**Kann Knut die Klausur bestehen?**

## Traveling Salesman



# Laufzeit?

## 0-1 Knapsack

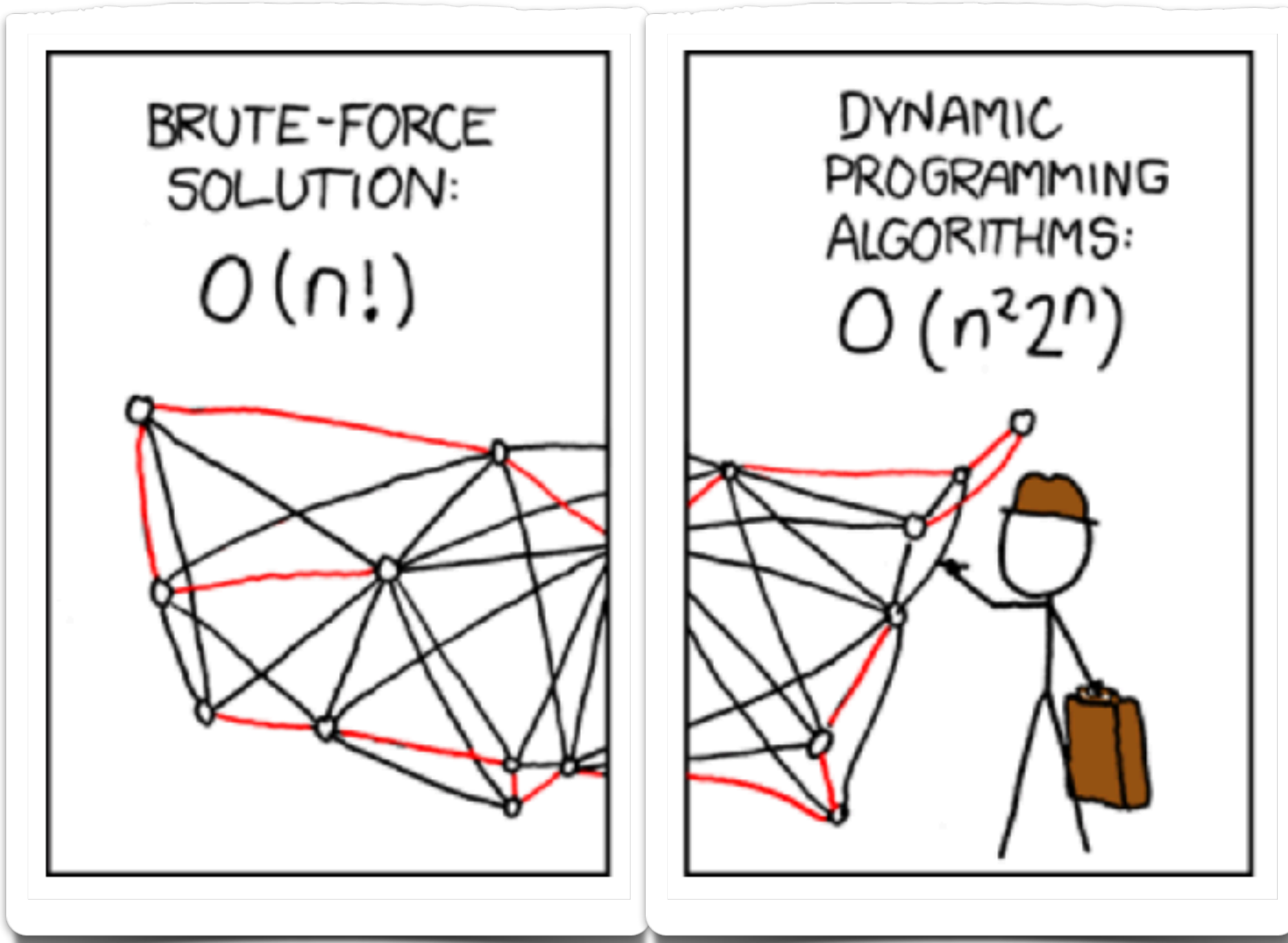
$O(nZ)$

$O(2^n f(n))$

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	24
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

## Traveling Salesman



# Laufzeit?

## 0-1 Knapsack

$O(nZ)$

$O(2^n f(n))$

1	Nummer	2	3	4	5	6
					16	4

**Kann Knut die Klausur bestehen?**

32	9	8	14		9	4
2		2	20	15		
			3	40		
				10		

**Kann Knut die Klausur bestehen?**

## Traveling Salesman

BRUTE-FORCE SOLUTION:  
 $O(n!)$

DYNAMIC PROGRAMMING ALGORITHMS:  
 $O(n^2 2^n)$



# Laufzeit?

## 0-1 Knapsack

$O(nZ)$

$O(2^n f(n))$

1	Nummer	2	3	4	5	6
					16	4

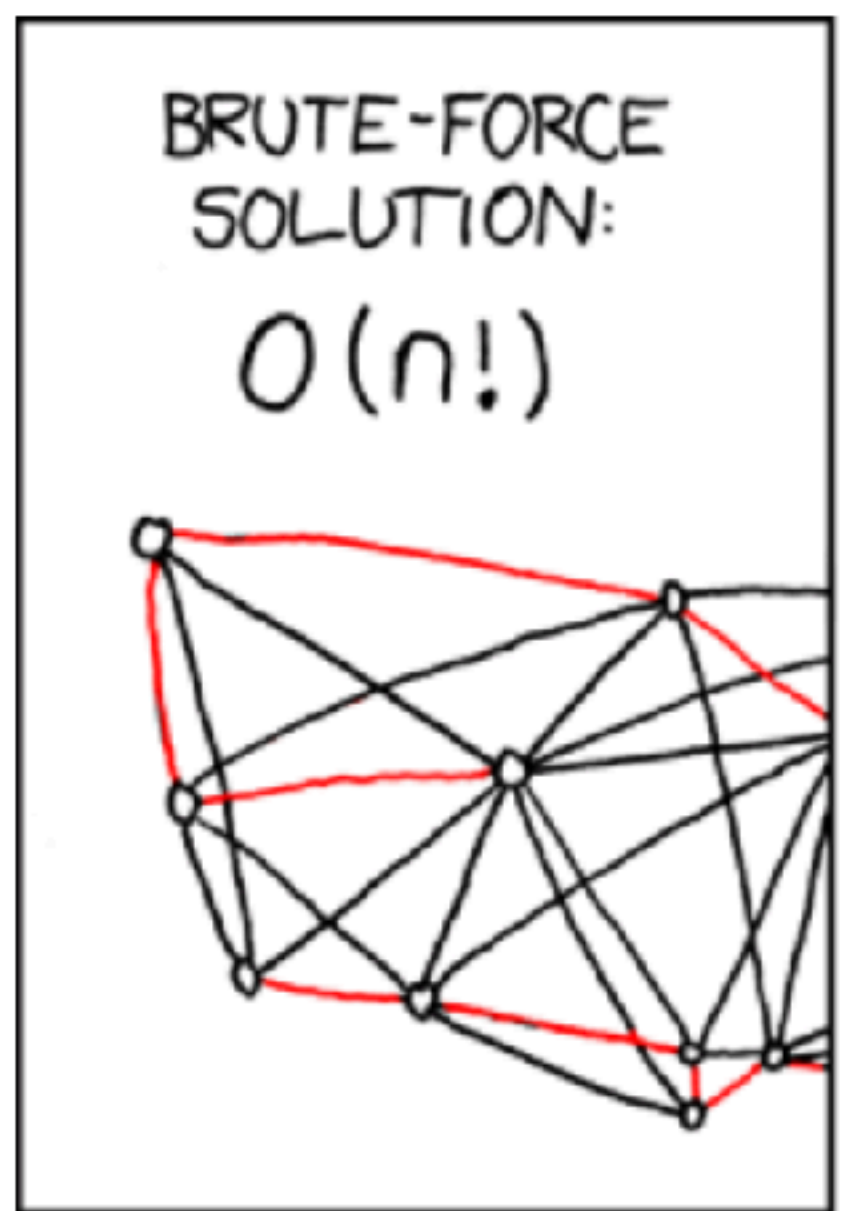
**Kann Knut die Klausur bestehen?**

32	9	8	14		9	4
2		2	20	15		
			3	40		
				10		

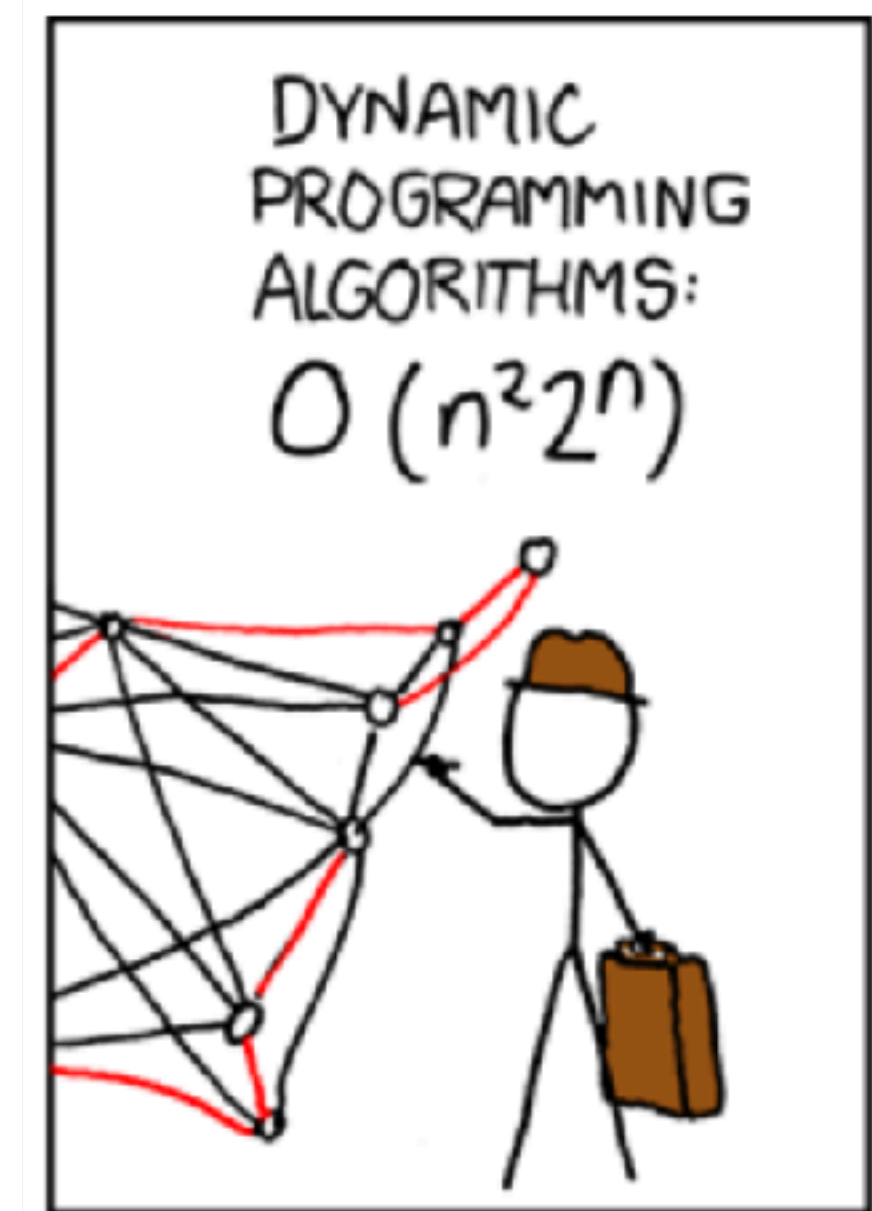
**Kann Knut die Klausur bestehen?**

## Traveling Salesman

BRUTE-FORCE SOLUTION:  
 $O(n!)$



DYNAMIC PROGRAMMING ALGORITHMS:  
 $O(n^2 2^n)$



**Ziel:** Vielleicht nicht die *bestmögliche* Lösung, sondern eine *gute*, aber dafür schneller...?!

# 4.2 Greedy und Approximation

# Beispiel 4.1: Knapsack

# Beispiel 4.1: Knapsack



# Beispiel 4.1: Knapsack



$$Z=N$$

# Beispiel 4.1: Knapsack



$Z=N$   
 $n=2$

# Beispiel 4.1: Knapsack



# Beispiel 4.1: Knapsack



$Z=N$   
 $n=2$



$z_1 = 1$



# Beispiel 4.1: Knapsack

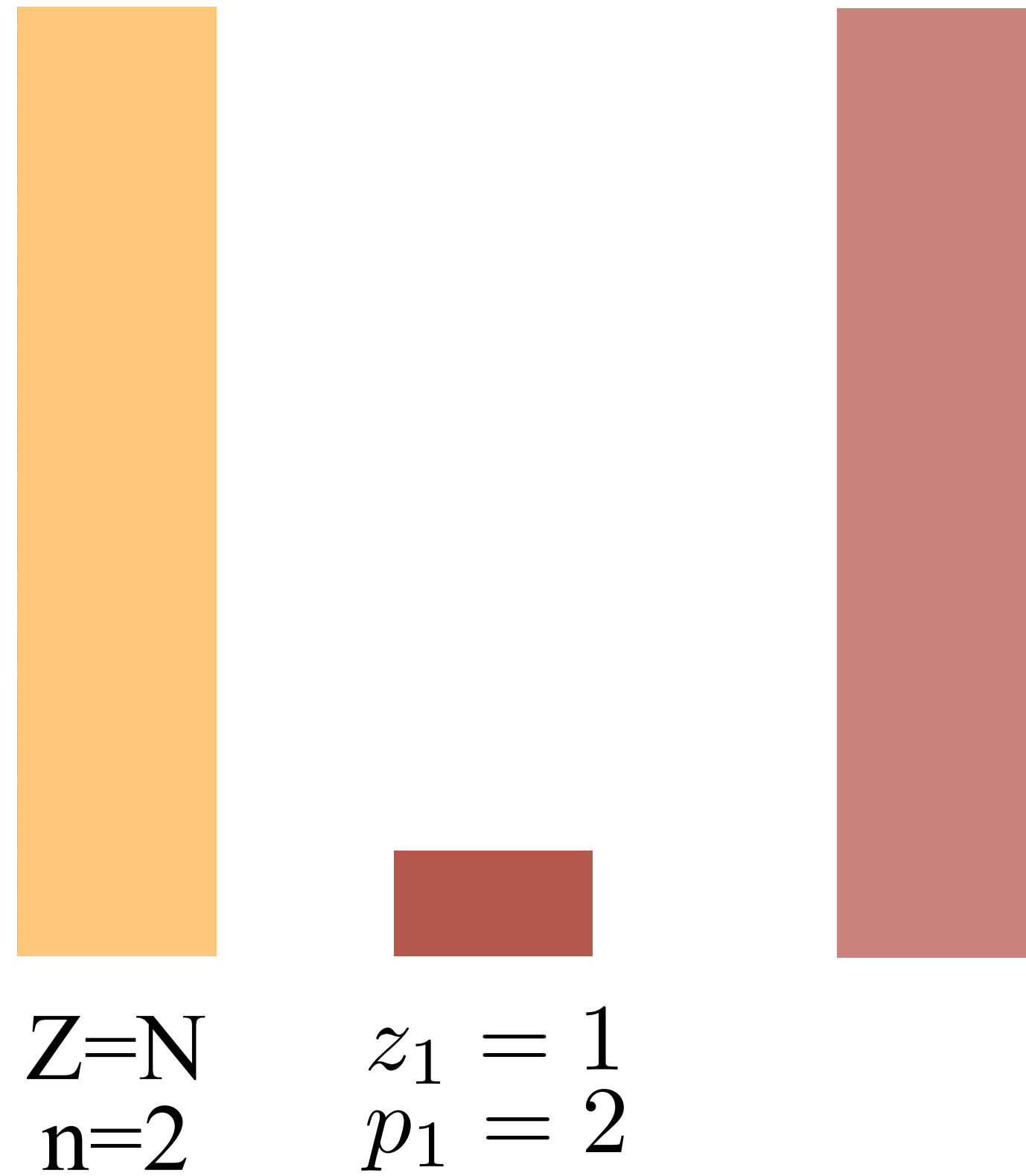


$Z=N$   
 $n=2$

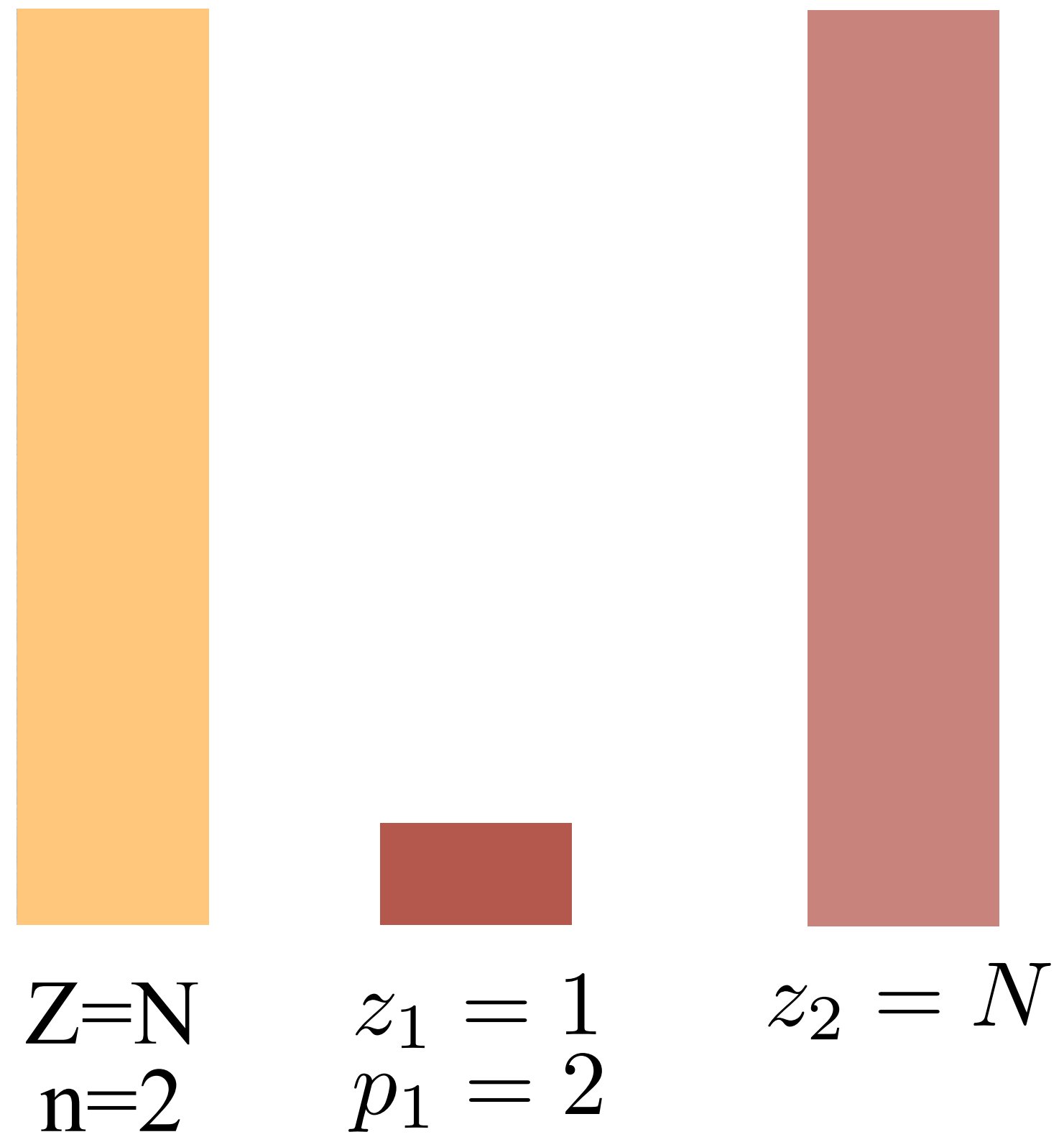


$z_1 = 1$   
 $p_1 = 2$

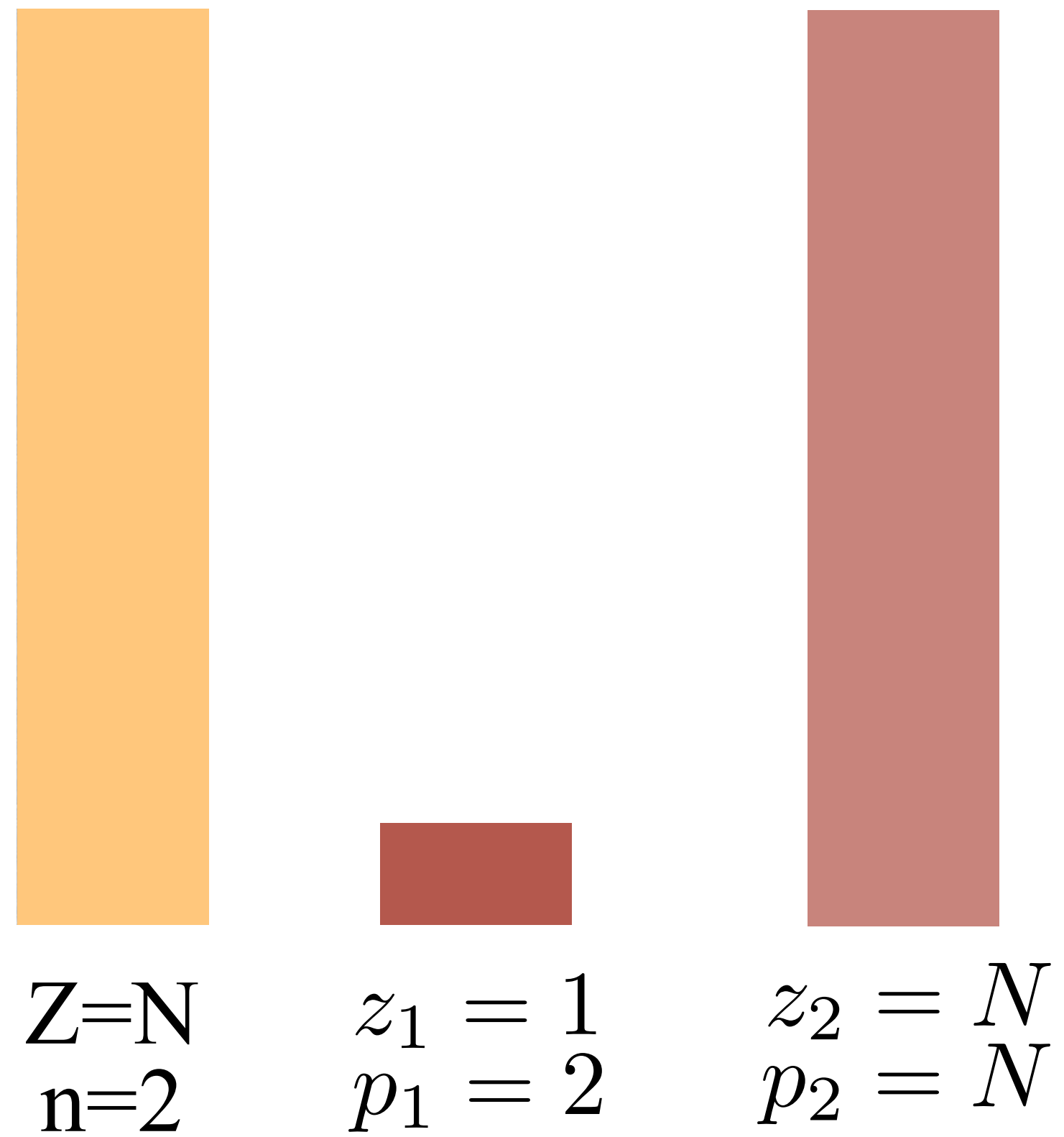
# Beispiel 4.1: Knapsack



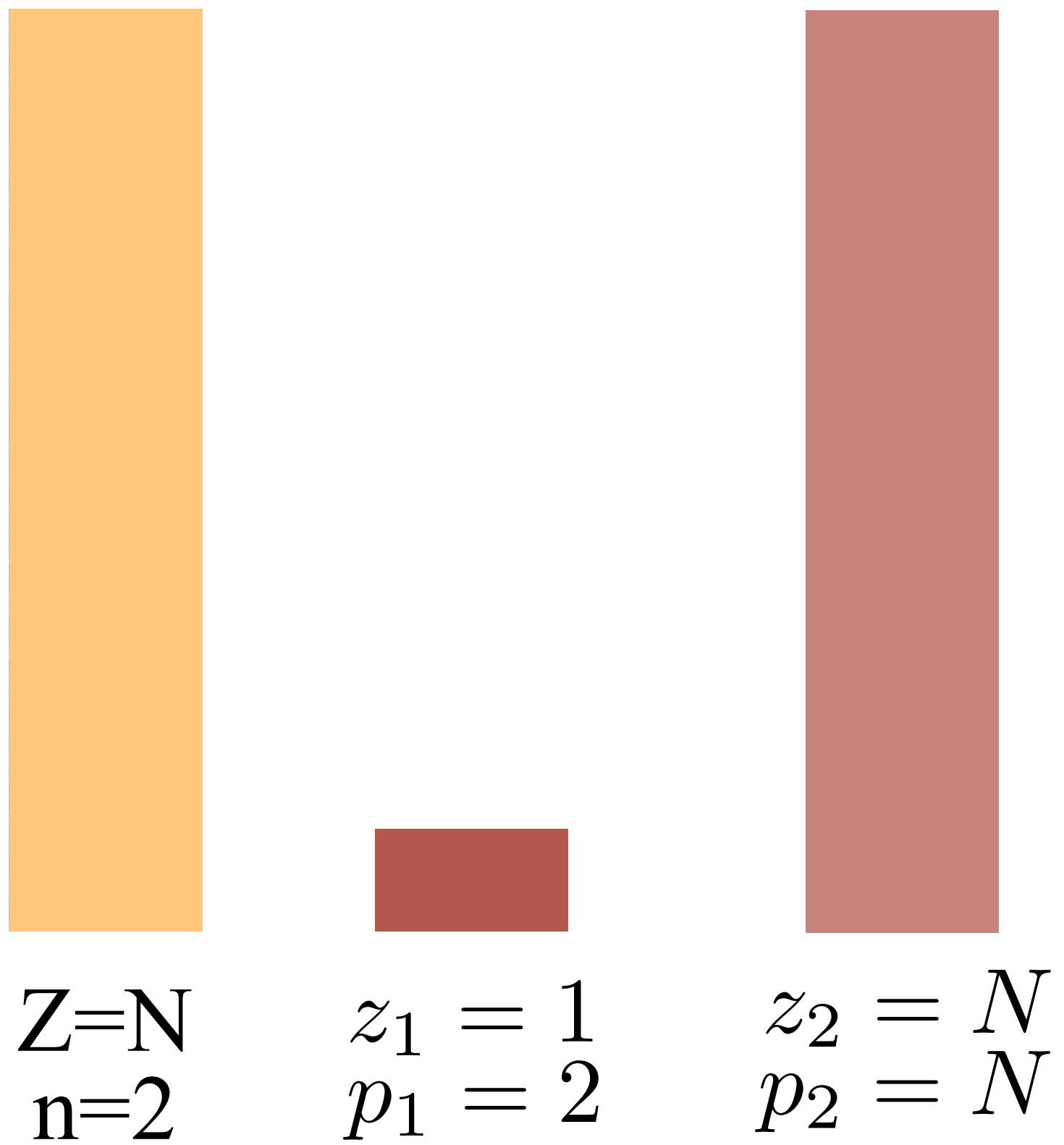
# Beispiel 4.1: Knapsack



# Beispiel 4.1: Knapsack

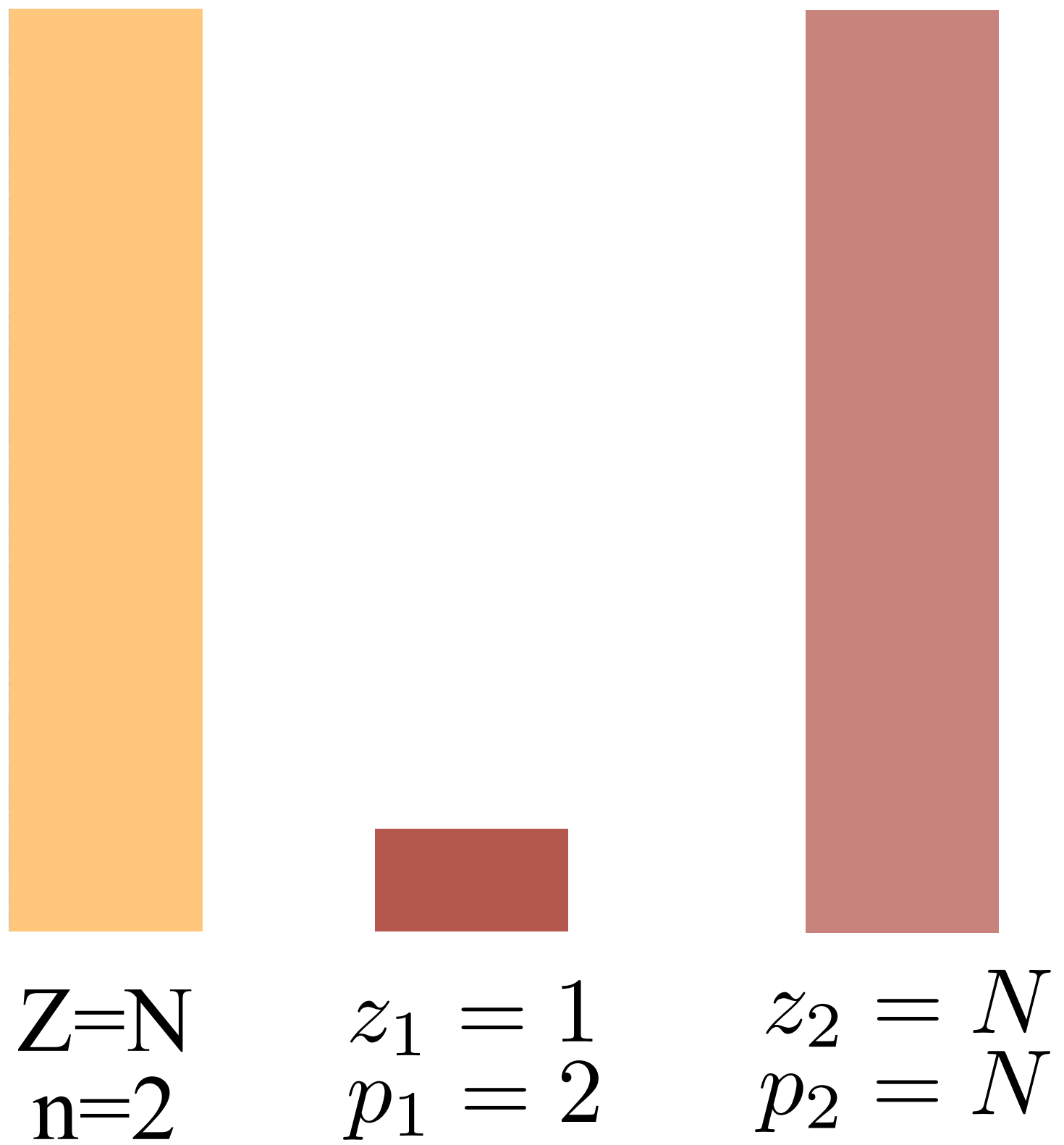


# Beispiel 4.1: Knapsack



Algorithmus 4.2 GREEDY<sub>0</sub>

# Beispiel 4.1: Knapsack



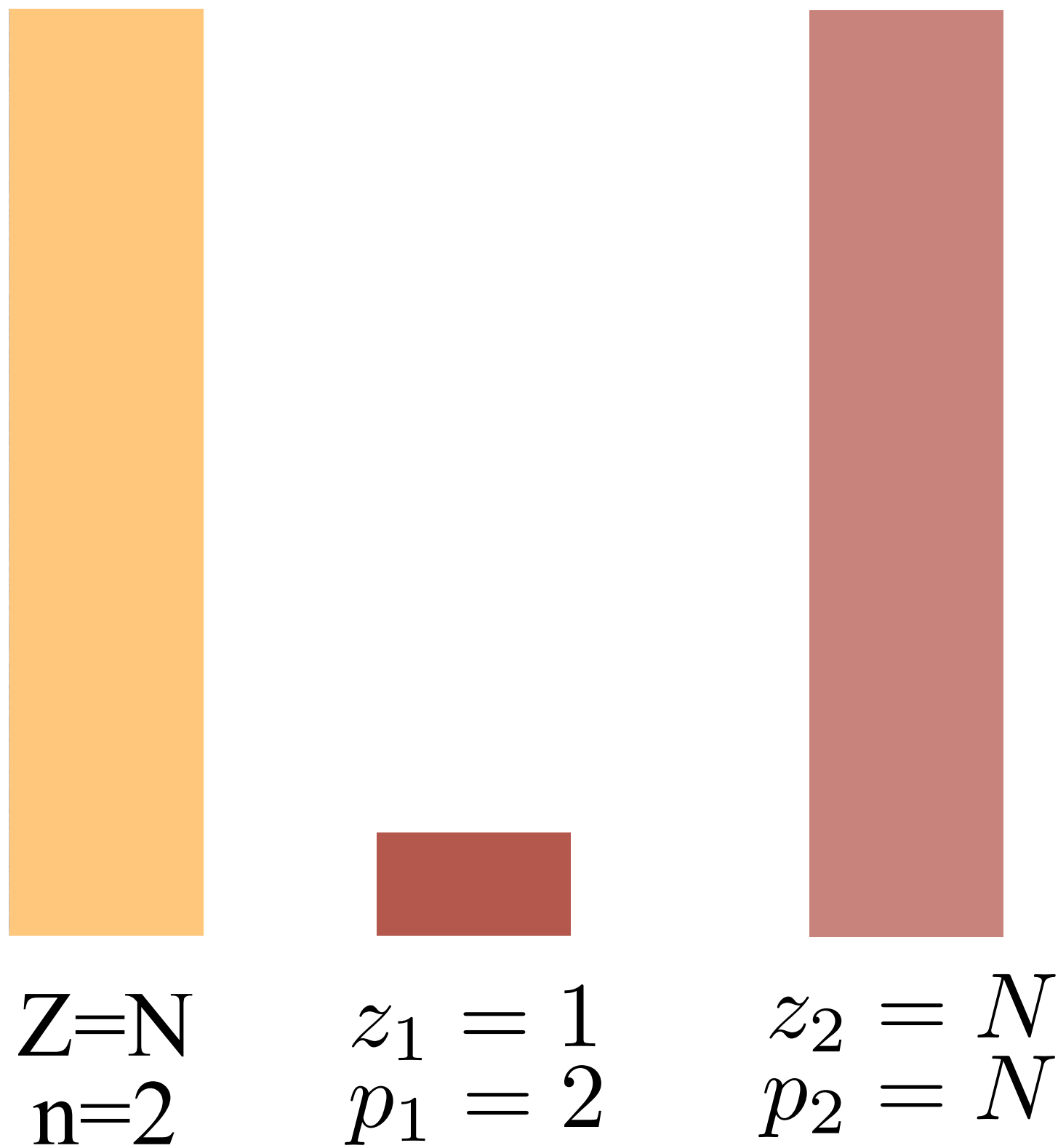
---

**Algorithmus 4.2** GREEDY<sub>0</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

# Beispiel 4.1: Knapsack



---

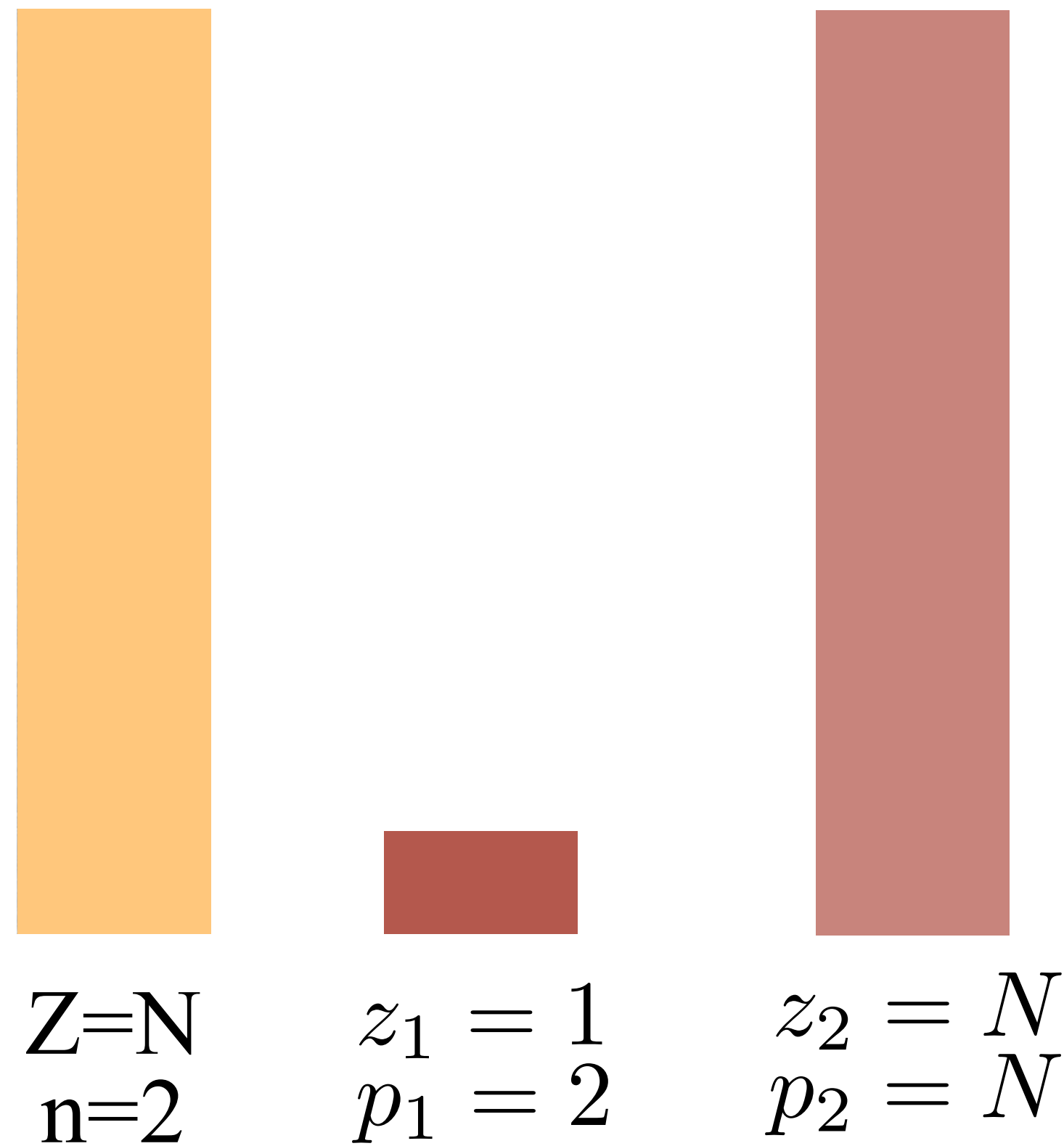
## Algorithmus 4.2 GREEDY<sub>0</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

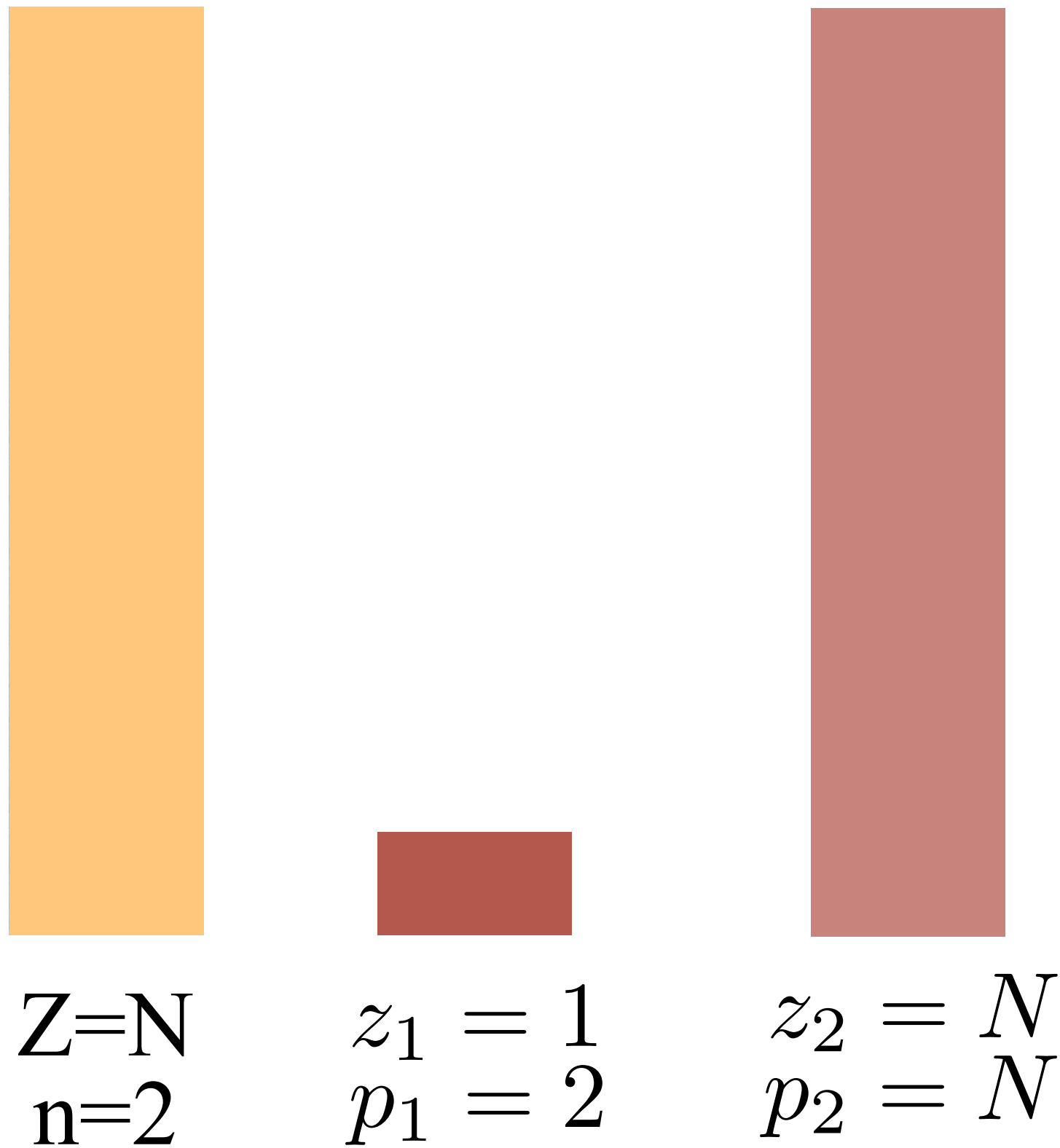
**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$



# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

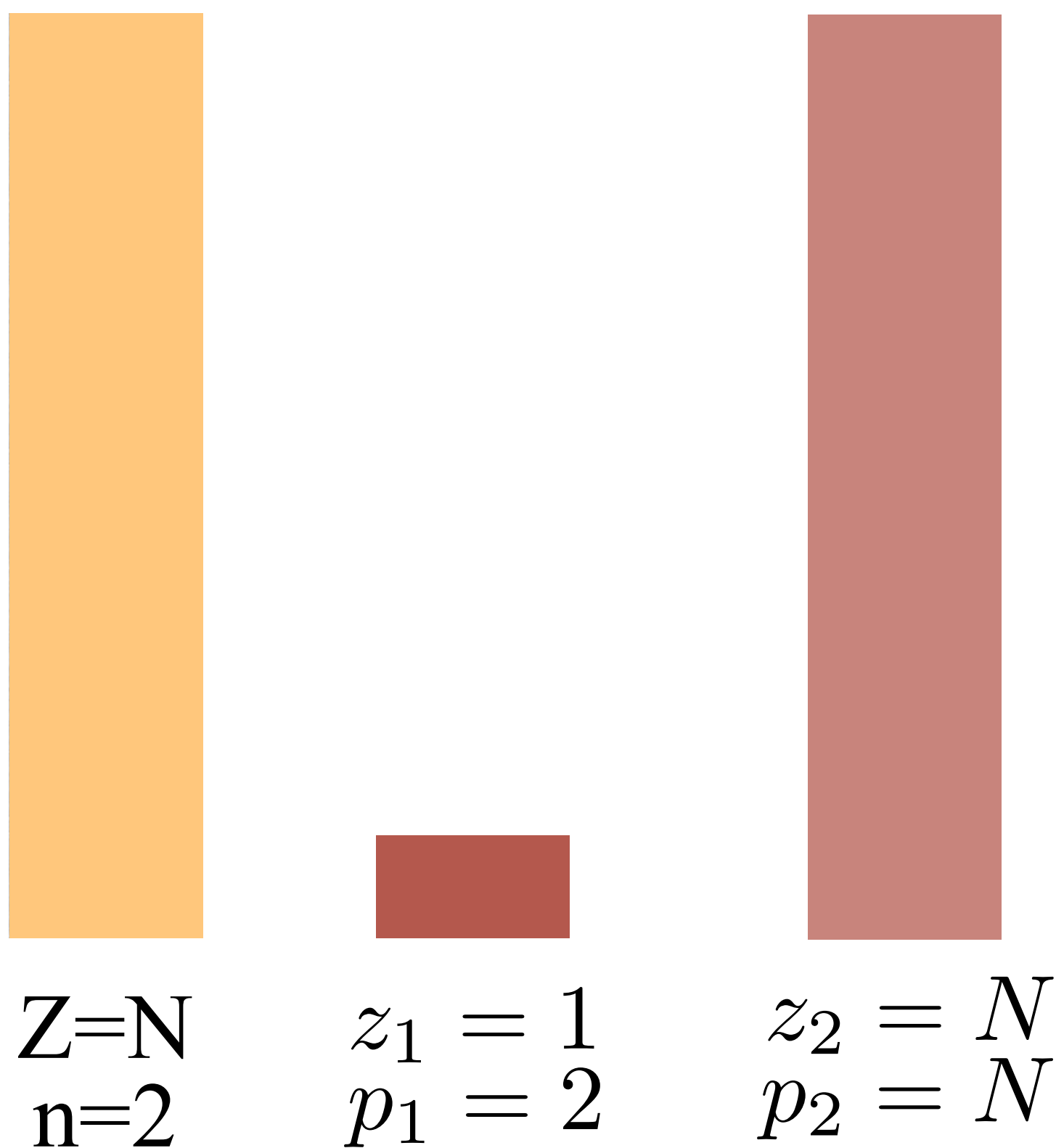
mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

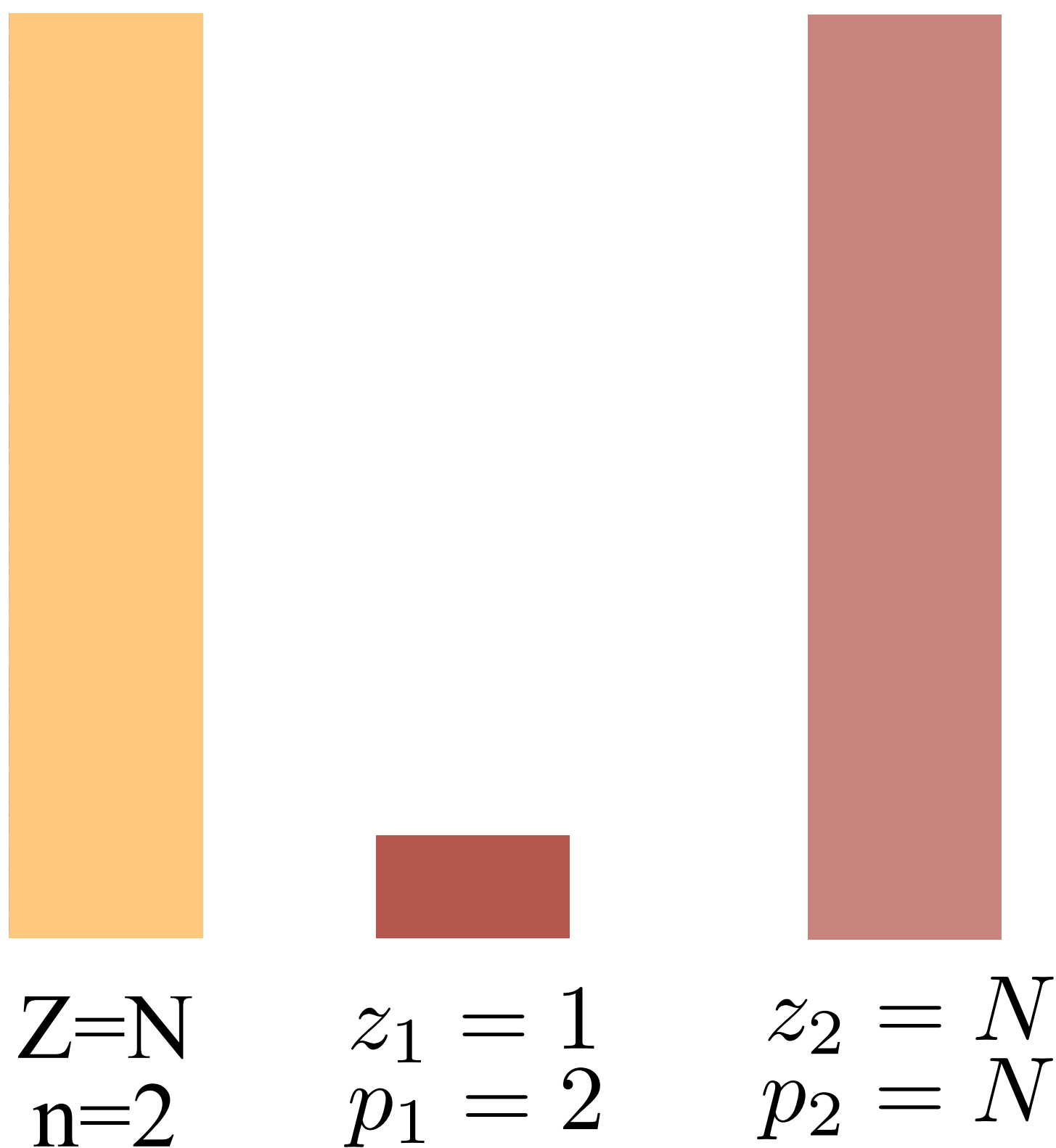
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

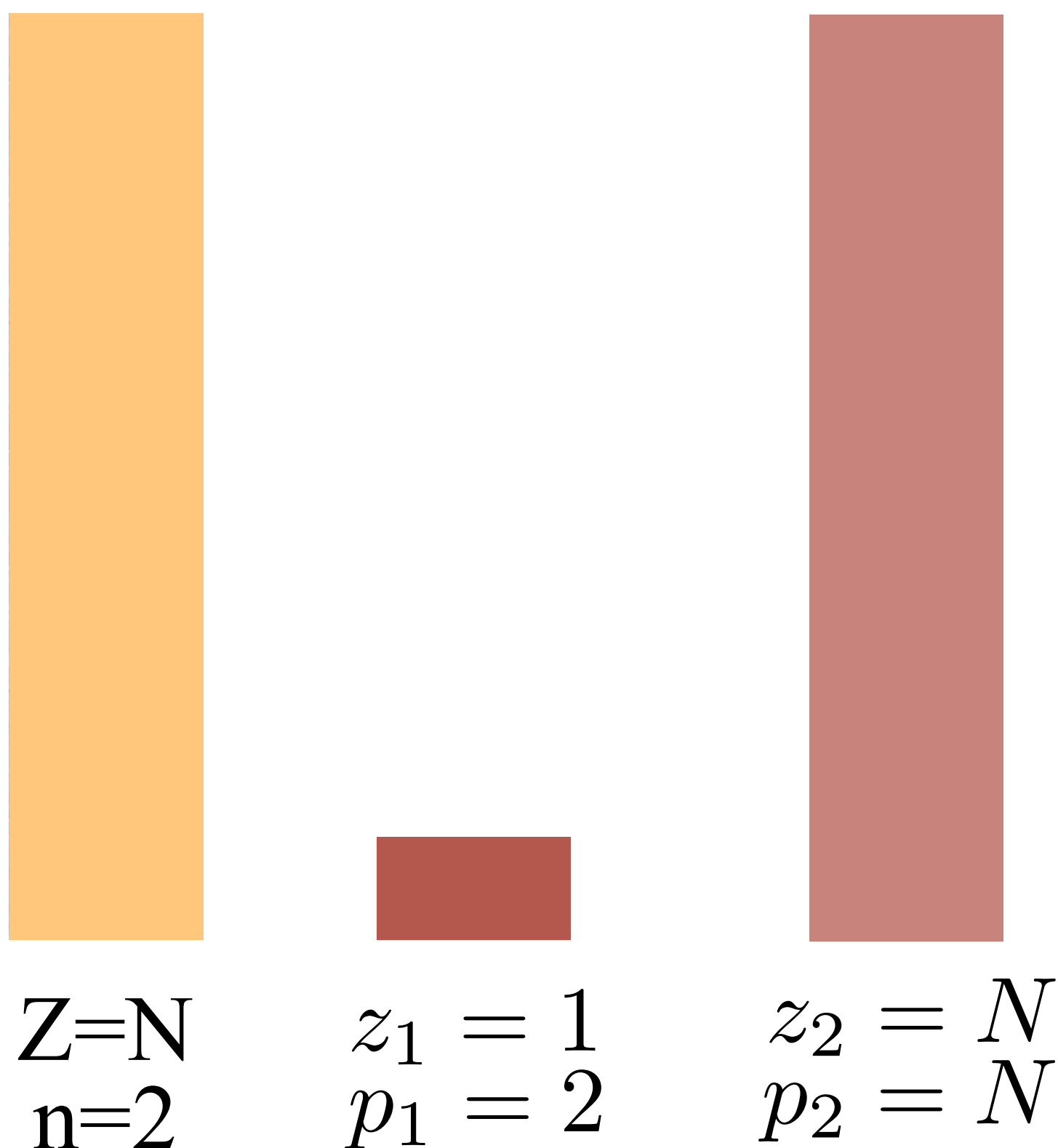
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

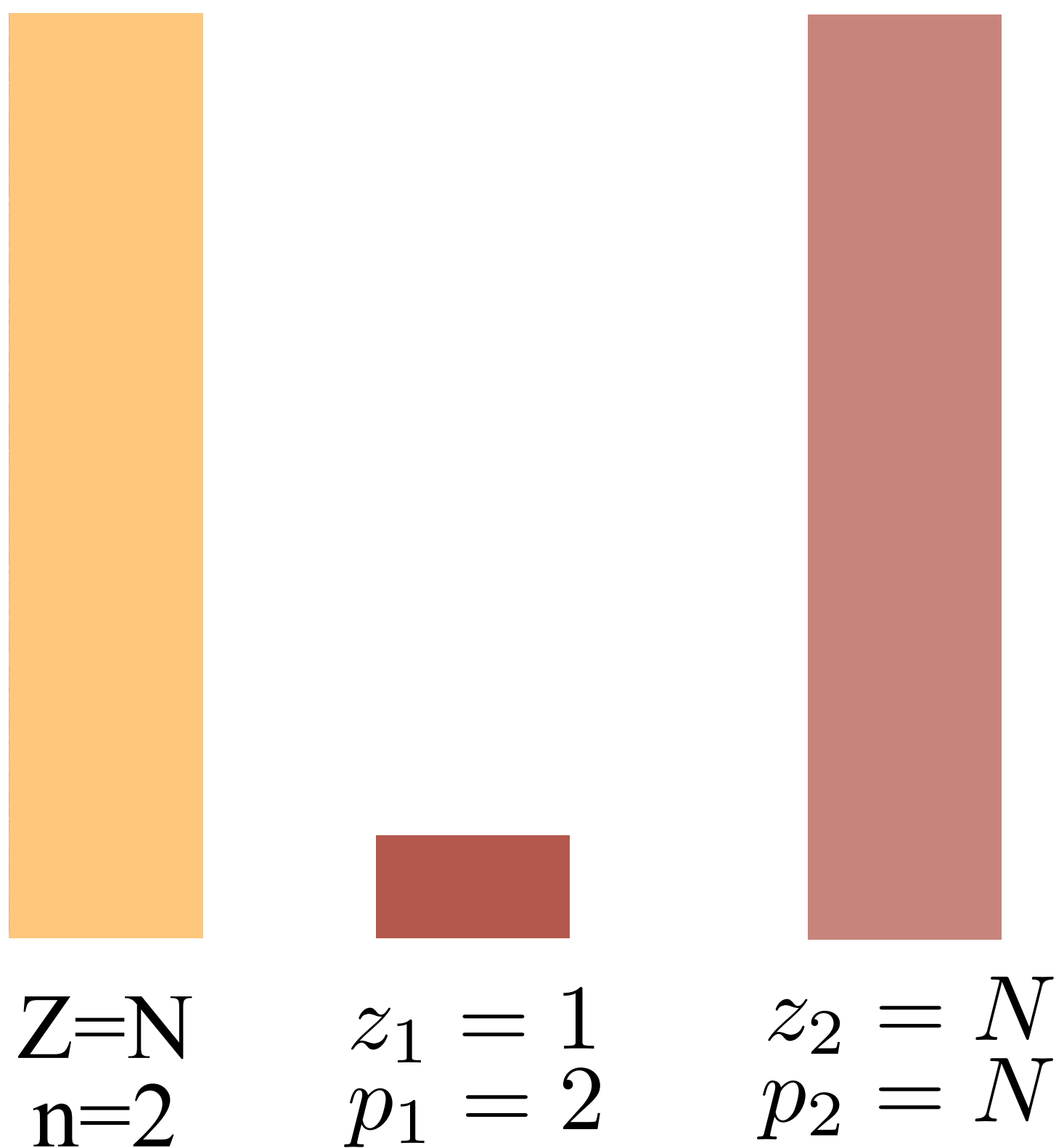
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

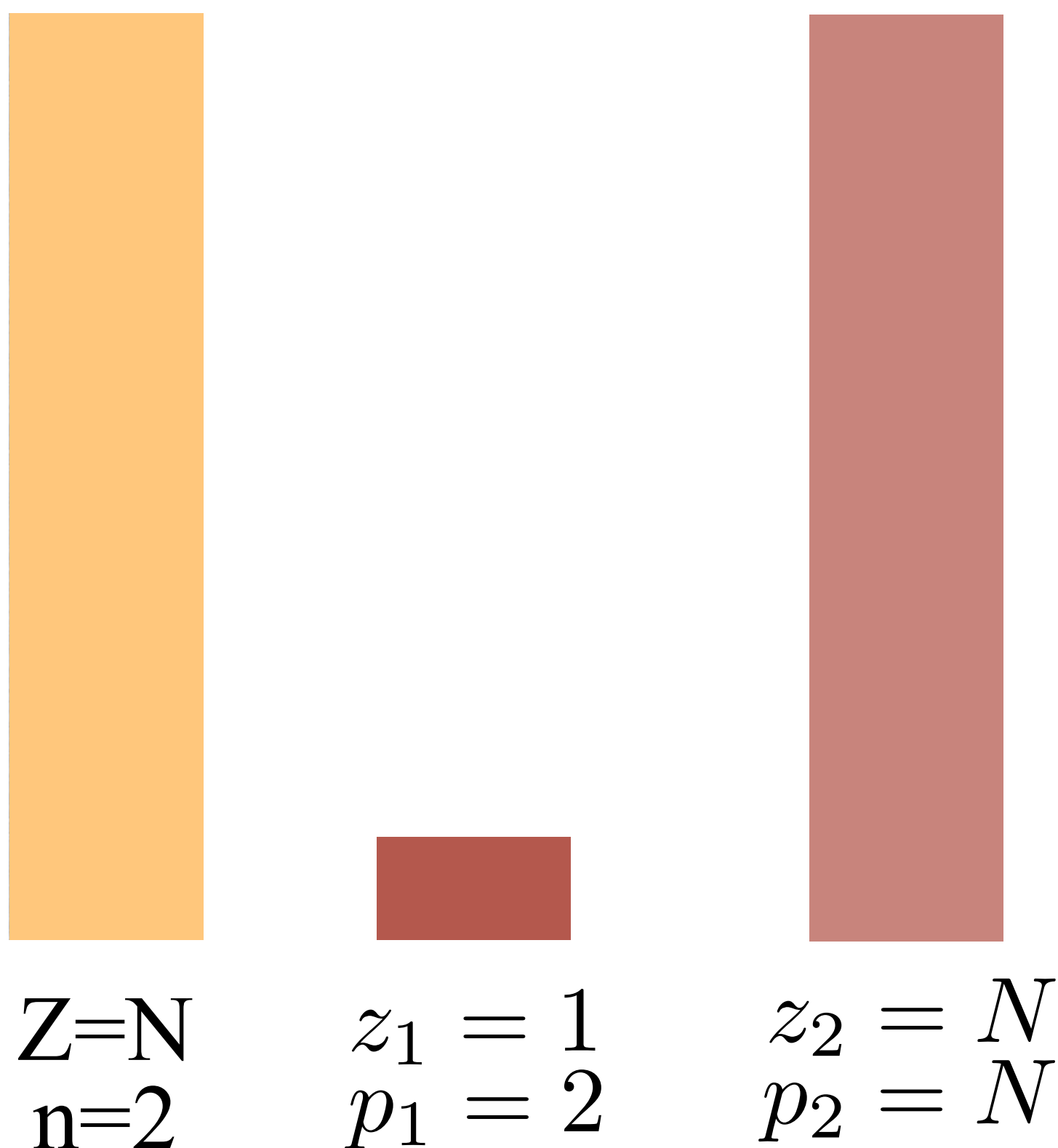
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

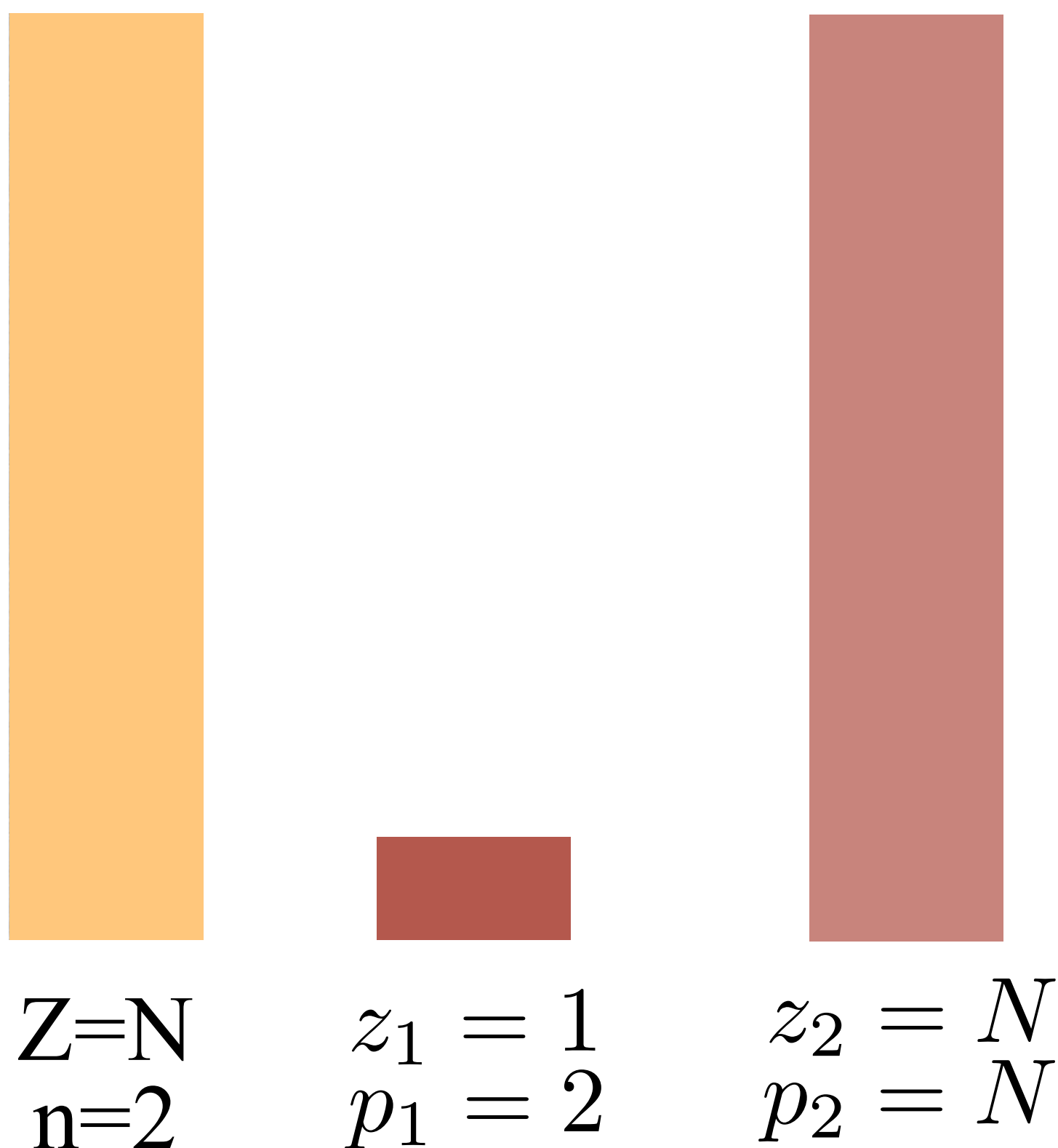
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

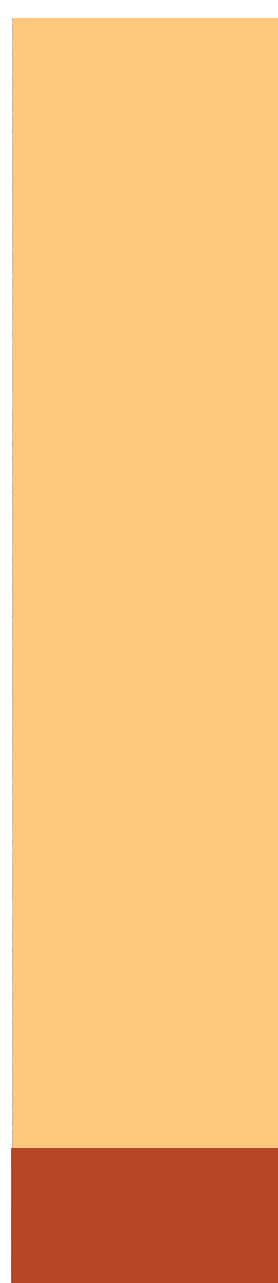
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



$Z=N$   
 $n=2$

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**



# Beispiel 4.1: Knapsack



$Z=N$   
 $n=2$

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



Greedy: 2



$Z=N$   
 $n=2$

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

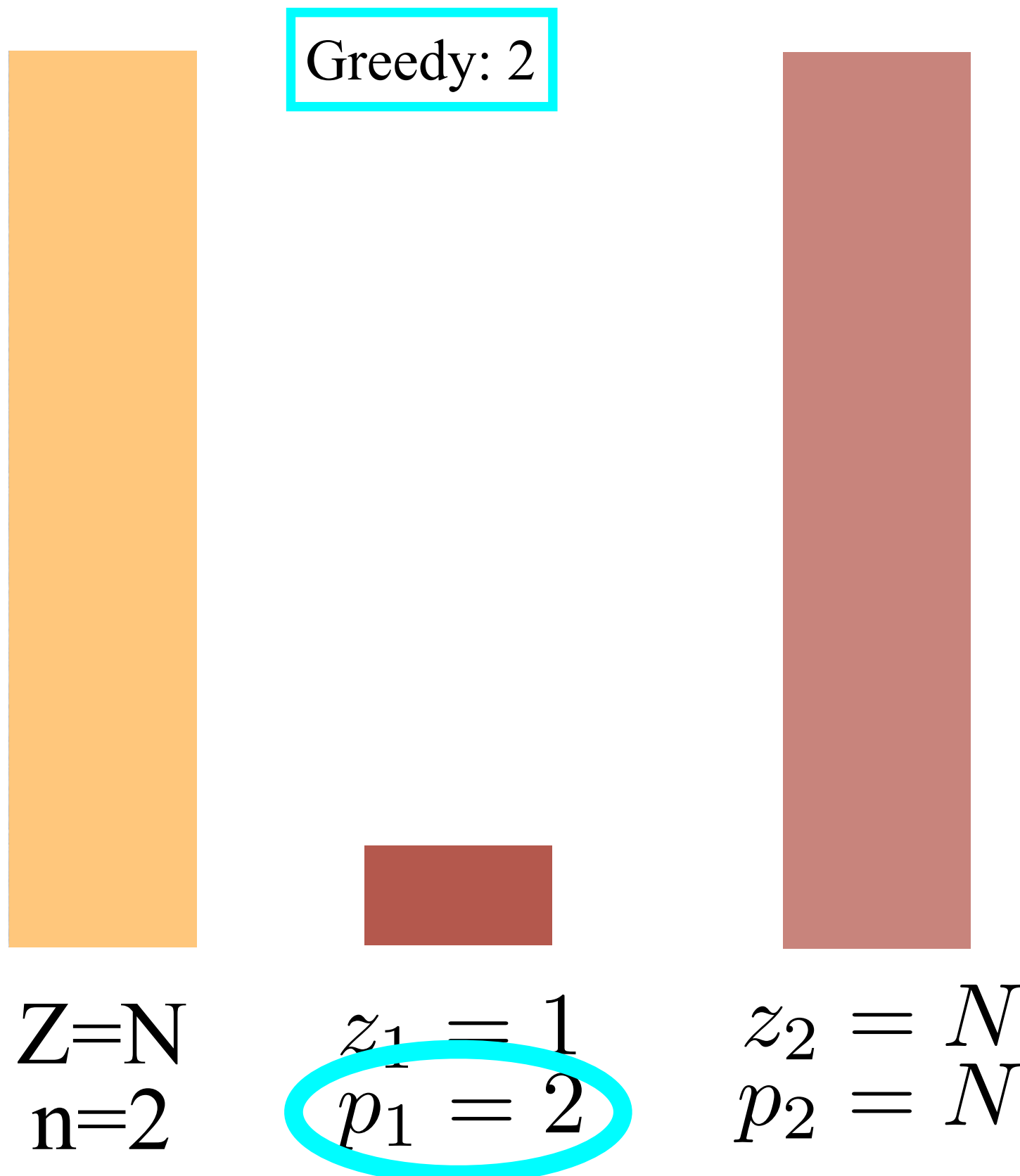
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

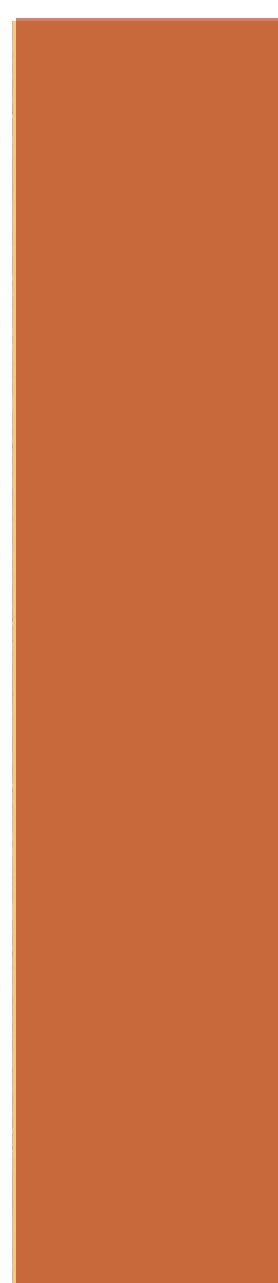
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



Greedy: 2



$Z=N$   
 $n=2$

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

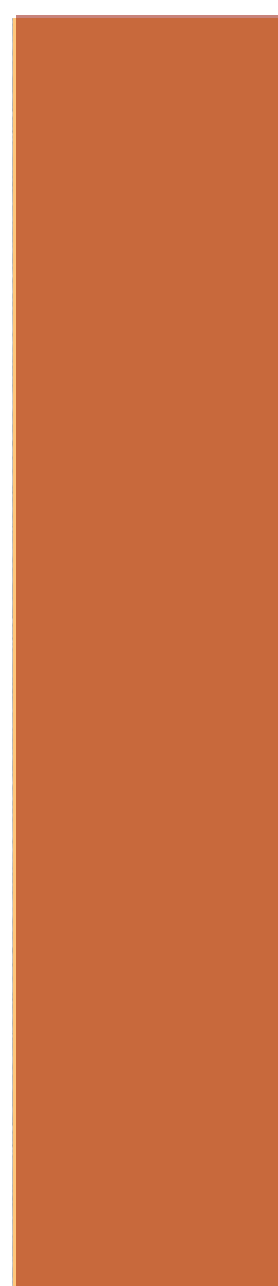
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



Greedy: 2



Z=N  
n=2

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

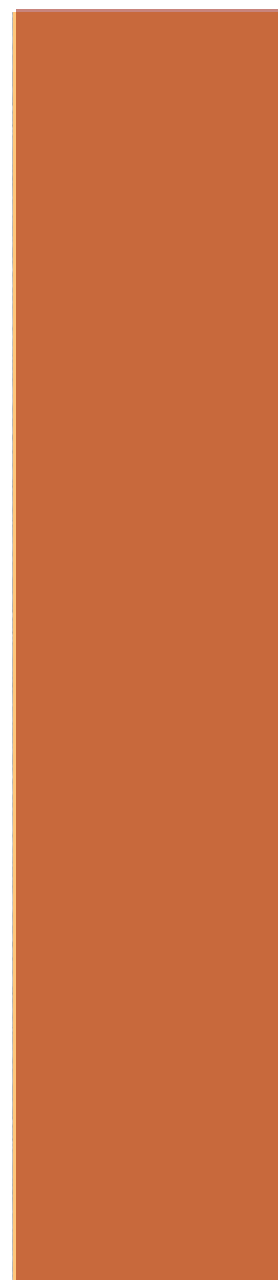
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



Greedy: 2

OPT: N



Z=N  
n=2

$z_1 = 1$   
 $p_1 = 2$

$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

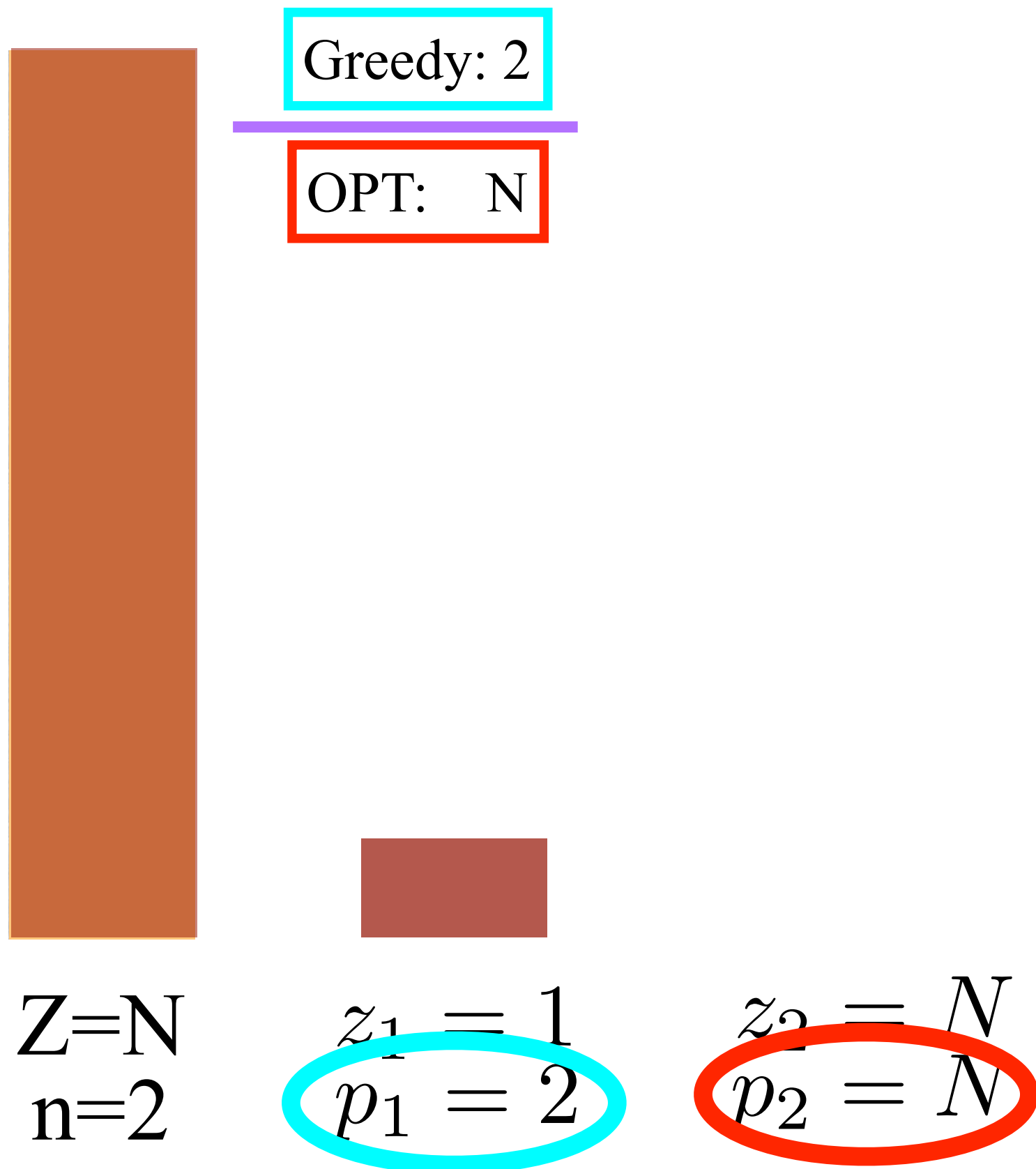
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

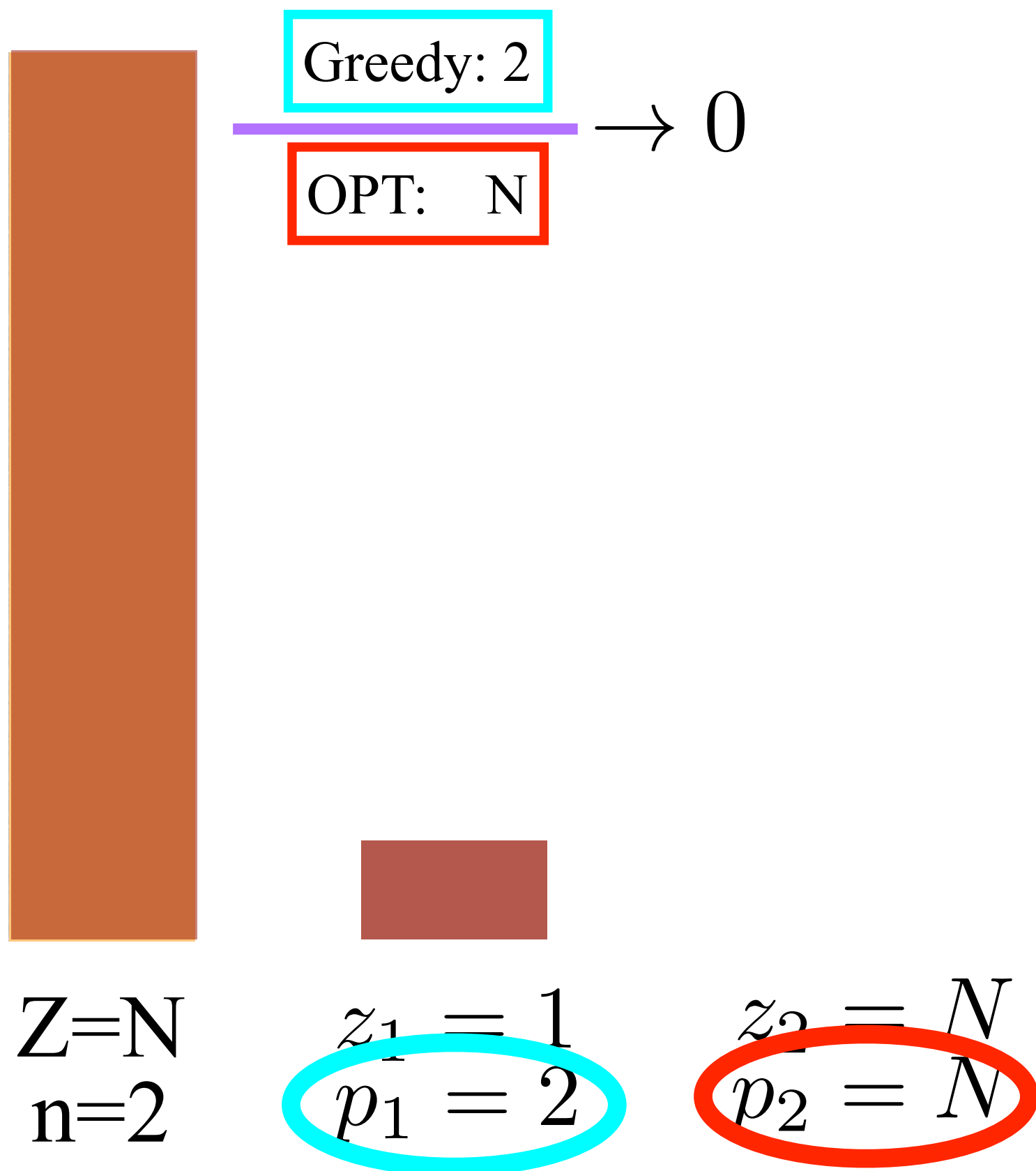
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**

# Beispiel 4.1: Knapsack



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

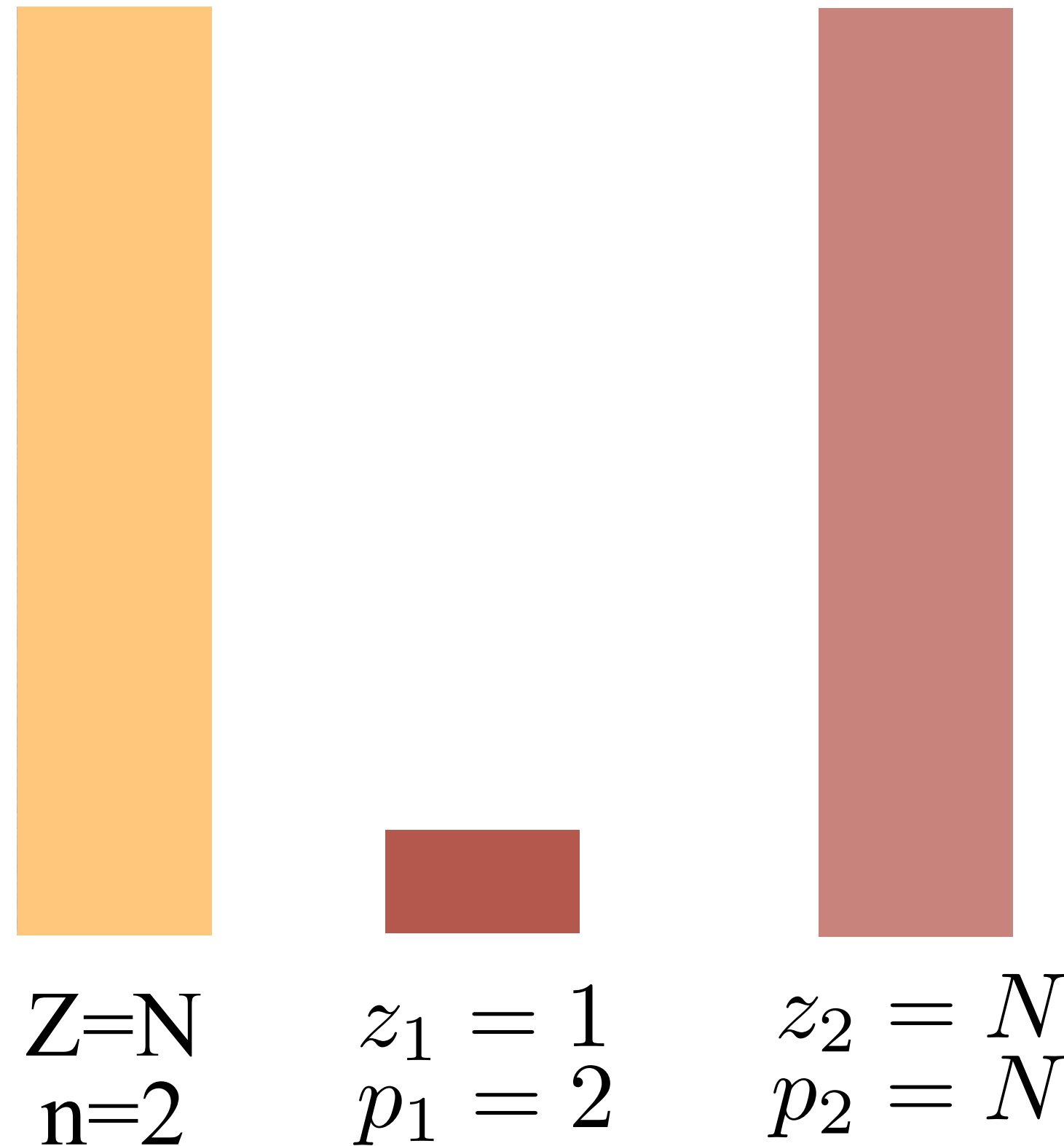
und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6: **return**



# Verbesserung?!



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Verbesserung?!



Greedy: 2



$Z=N$   
 $n=2$



$z_1 = 1$   
 $p_1 = 2$



$z_2 = N$   
 $p_2 = N$

## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \textit{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if**  $\left( \sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z \right)$  **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Verbesserung?!



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

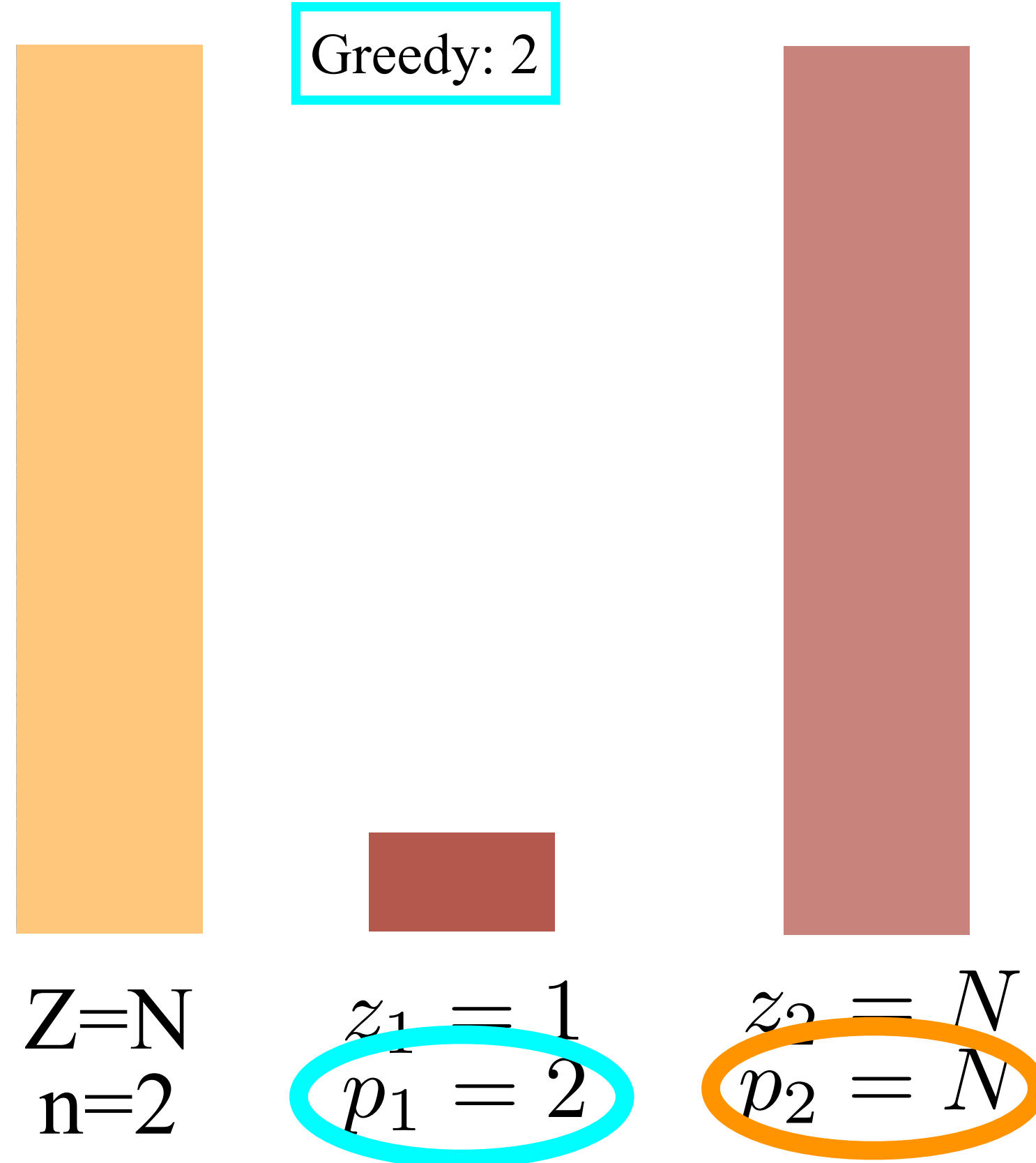
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Verbesserung?!



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

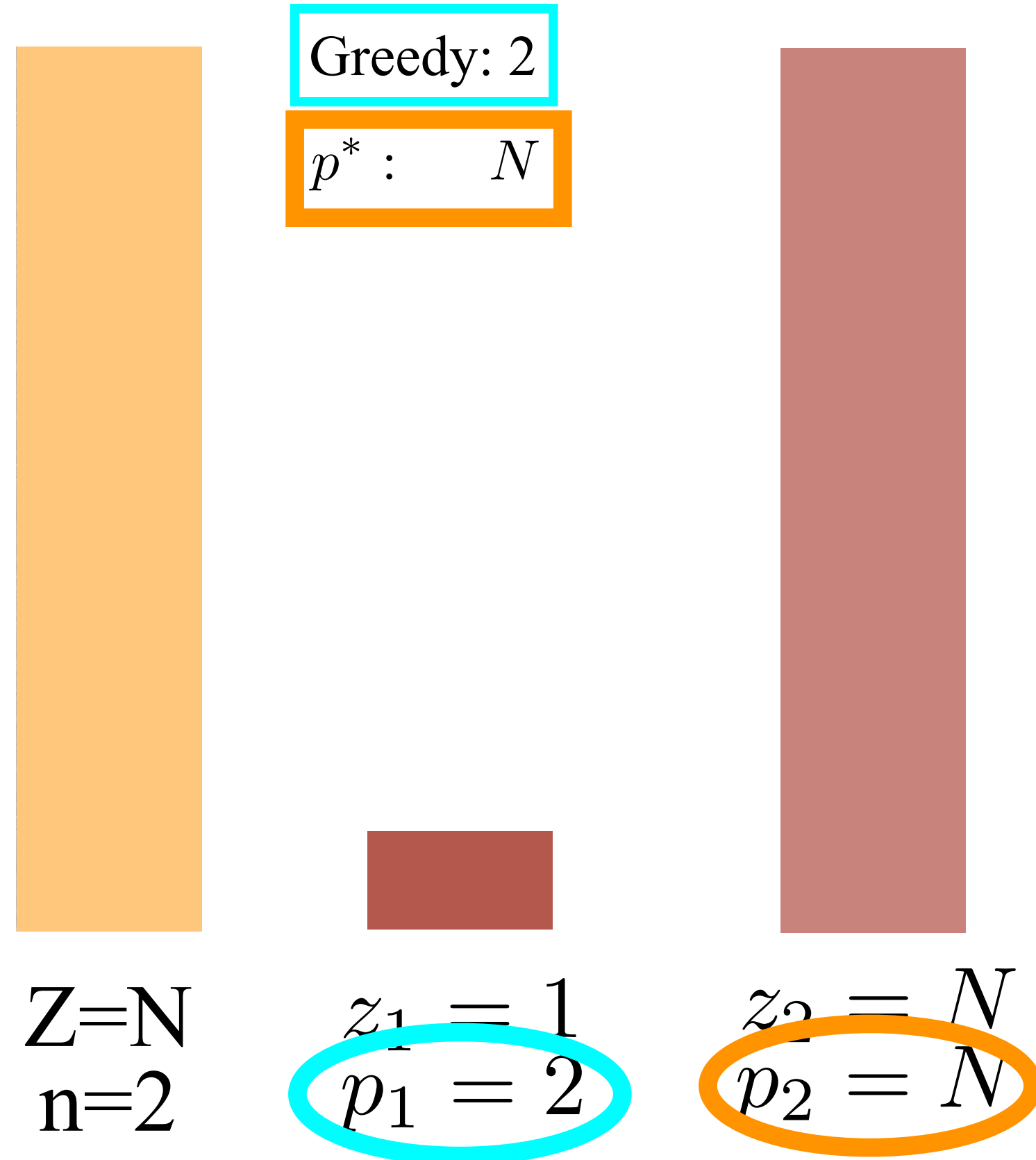
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Verbesserung?!



## Algorithmus 4.2 GREEDY<sub>0</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

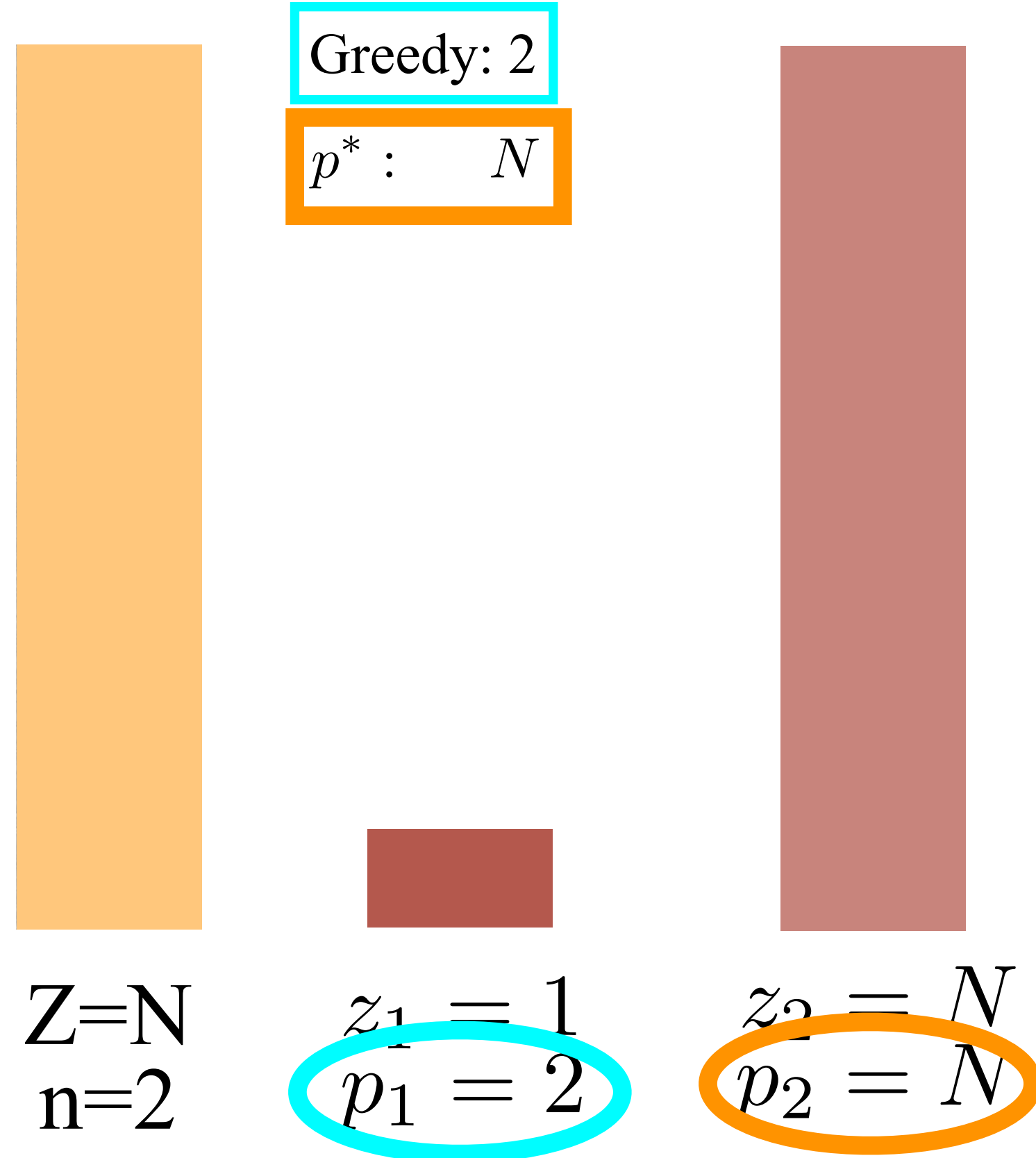
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$

# Verbesserung?!



## Algorithmus 4.3 GREEDY<sub>0</sub>'

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

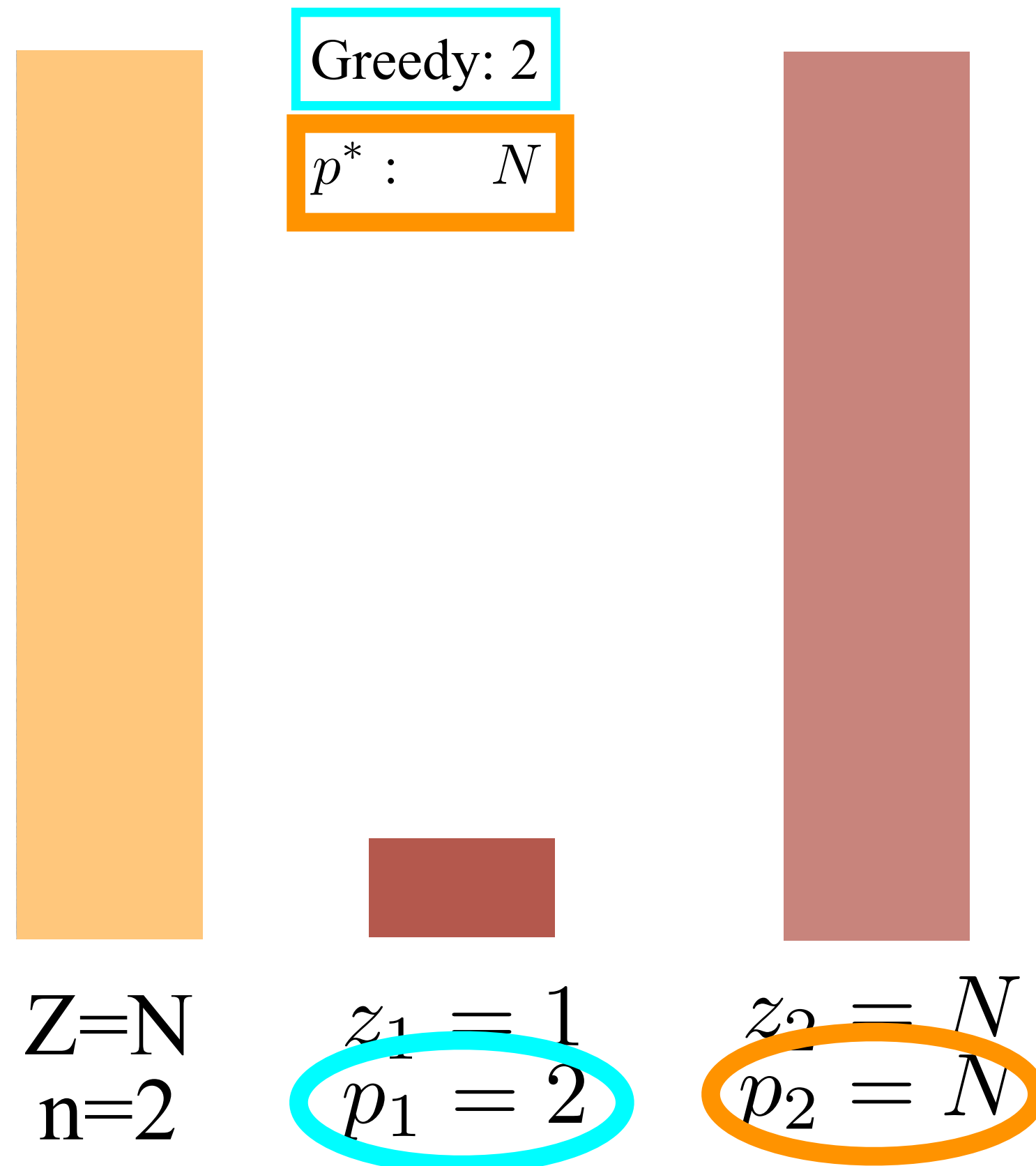
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6:  $G'_0 := \max \{G_0, \max \{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}\}$
- 7: **return**

# Verbesserung?!



## Algorithmus 4.3 GREEDY<sub>0</sub>'

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

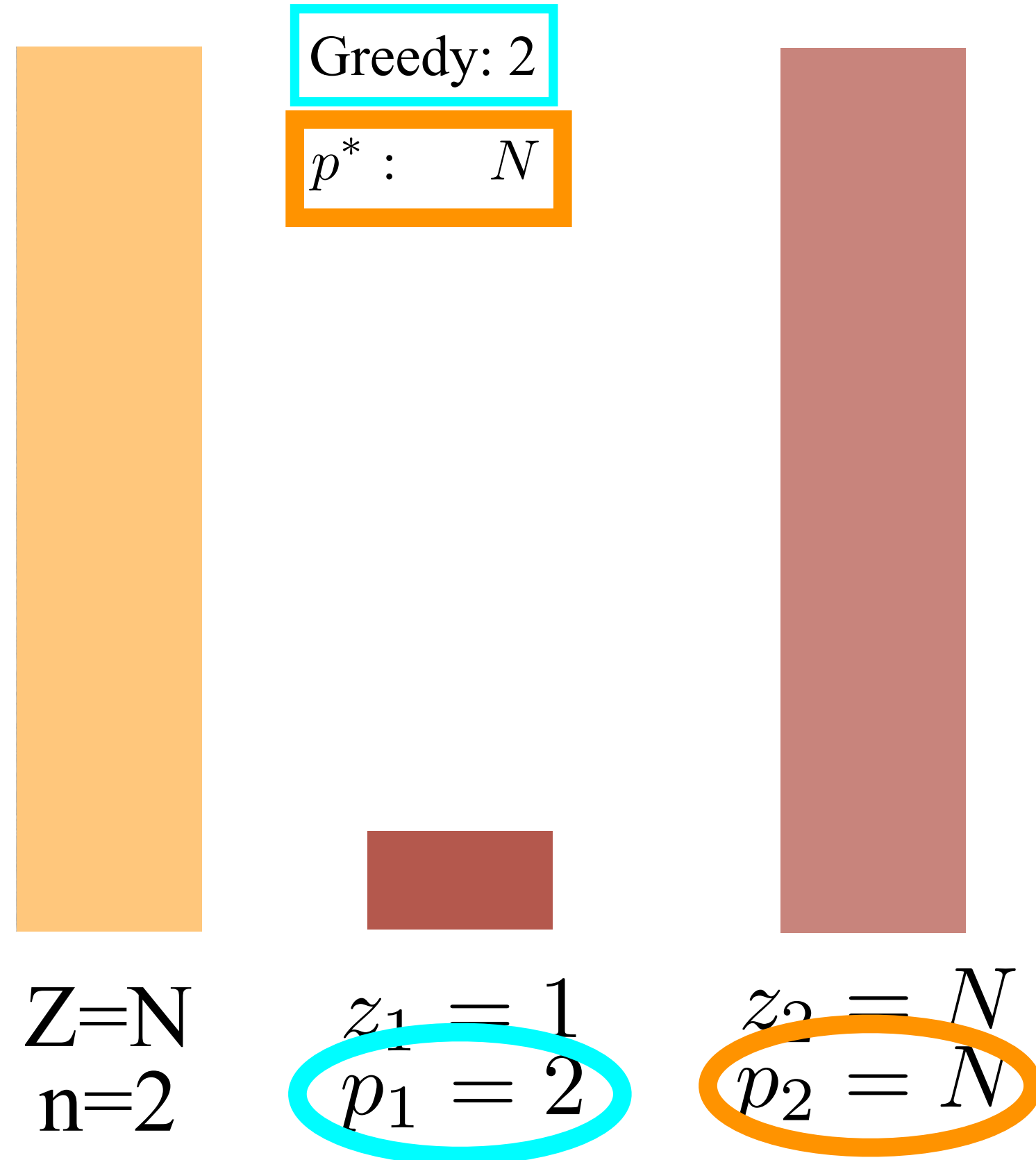
$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6:  $G'_0 := \max \{ G_0, \max \{ p_i \mid z_i < Z, i \in \{1, \dots, n\} \} \}$
- 7: **return**

# Verbesserung?!



## Algorithmus 4.3 GREEDY<sub>0</sub>'

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  und  $G_0$

mit

$$\sum_{i \in S} z_i \leq Z$$

und

$$G_0 := \sum_{i \in S} p_i = \text{Maximal}$$

- 1: Sortiere  $\{1, \dots, n\}$  nach  $\frac{z_i}{p_i}$  aufsteigend;  
Dies ergibt die Permutation  $\pi(1), \dots, \pi(n)$ .  
Setze  $j = 1$ .
- 2: **while** ( $j \leq n$ ) **do**
- 3:     **if** ( $\sum_{i=1}^{j-1} z_{\pi(i)} x_{\pi(i)} + z_{\pi(j)} \leq Z$ ) **then**
- 4:          $x_{\pi(j)} := 1$
- 5:      $j := j + 1$
- 6:  $G'_0 := \max \{ G_0, \max \{ p_i \mid z_i < Z, i \in \{1, \dots, n\} \} \}$
- 7: **return**



# Wie gut ist das allgemein?

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 := \max \{G_0, \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}\}$

2: **return**

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max \{G_0, \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}\}$

2: **return**

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

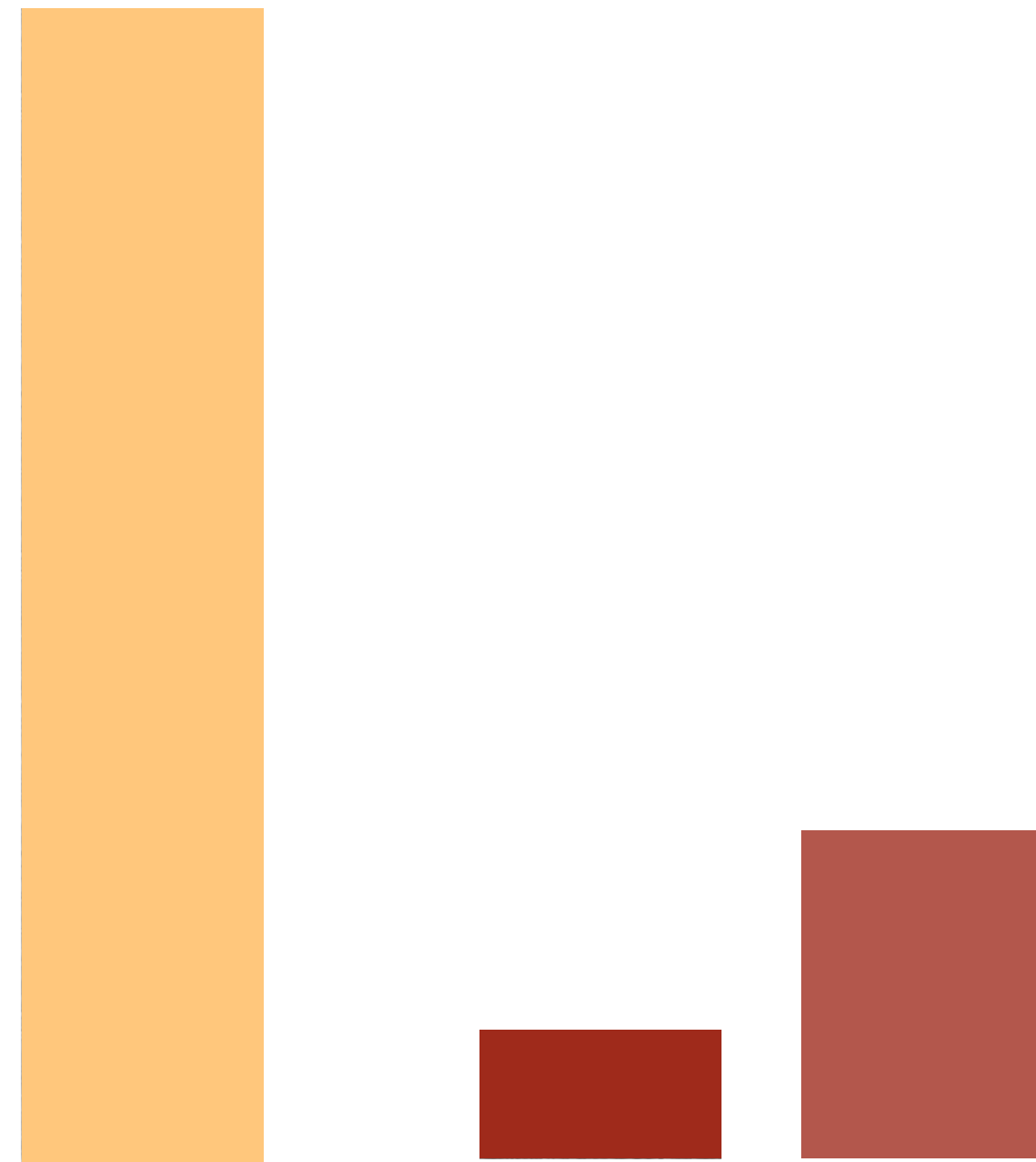
## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return





# Wie gut ist das allgemein?

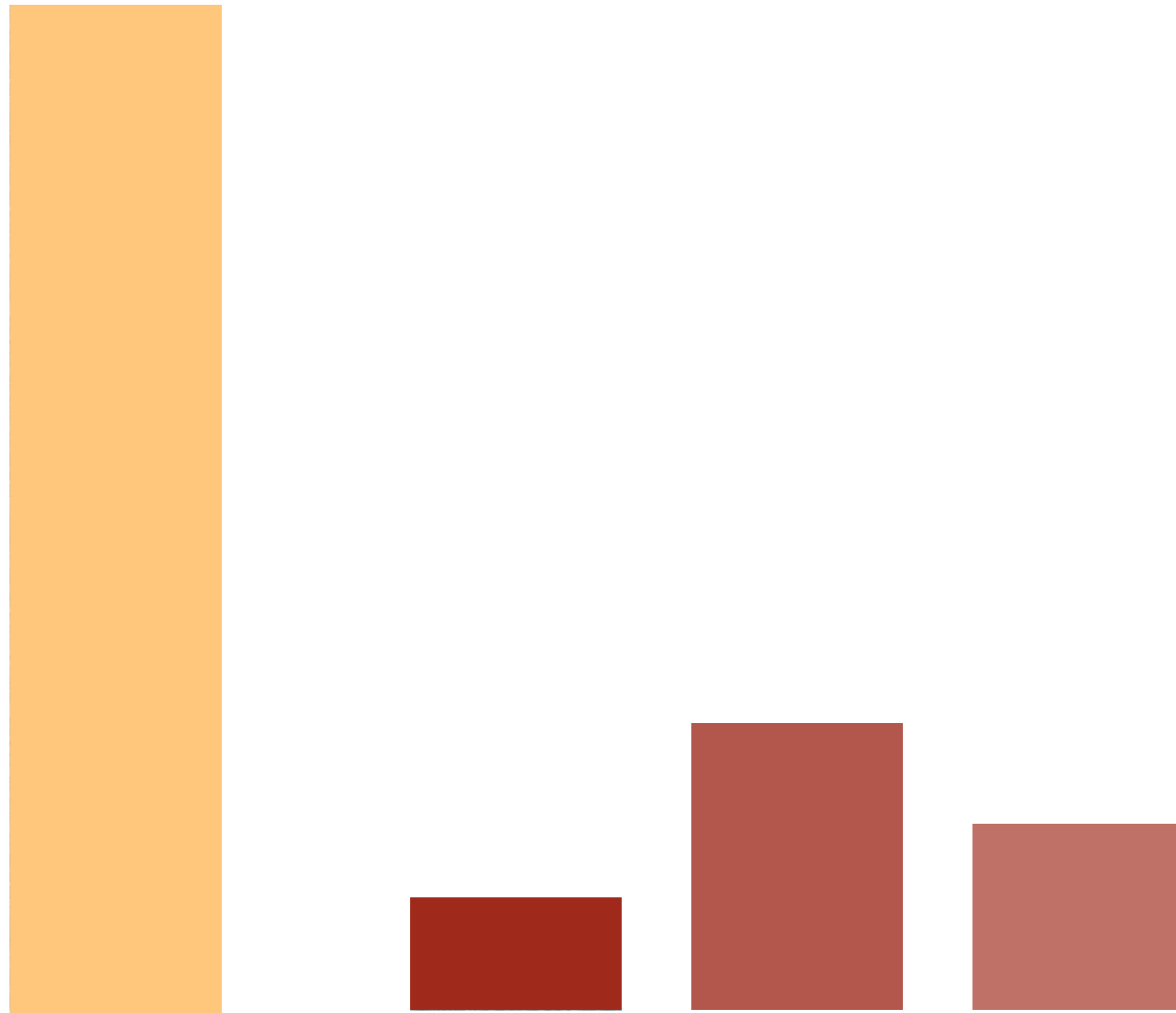
## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return



# Wie gut ist das allgemein?

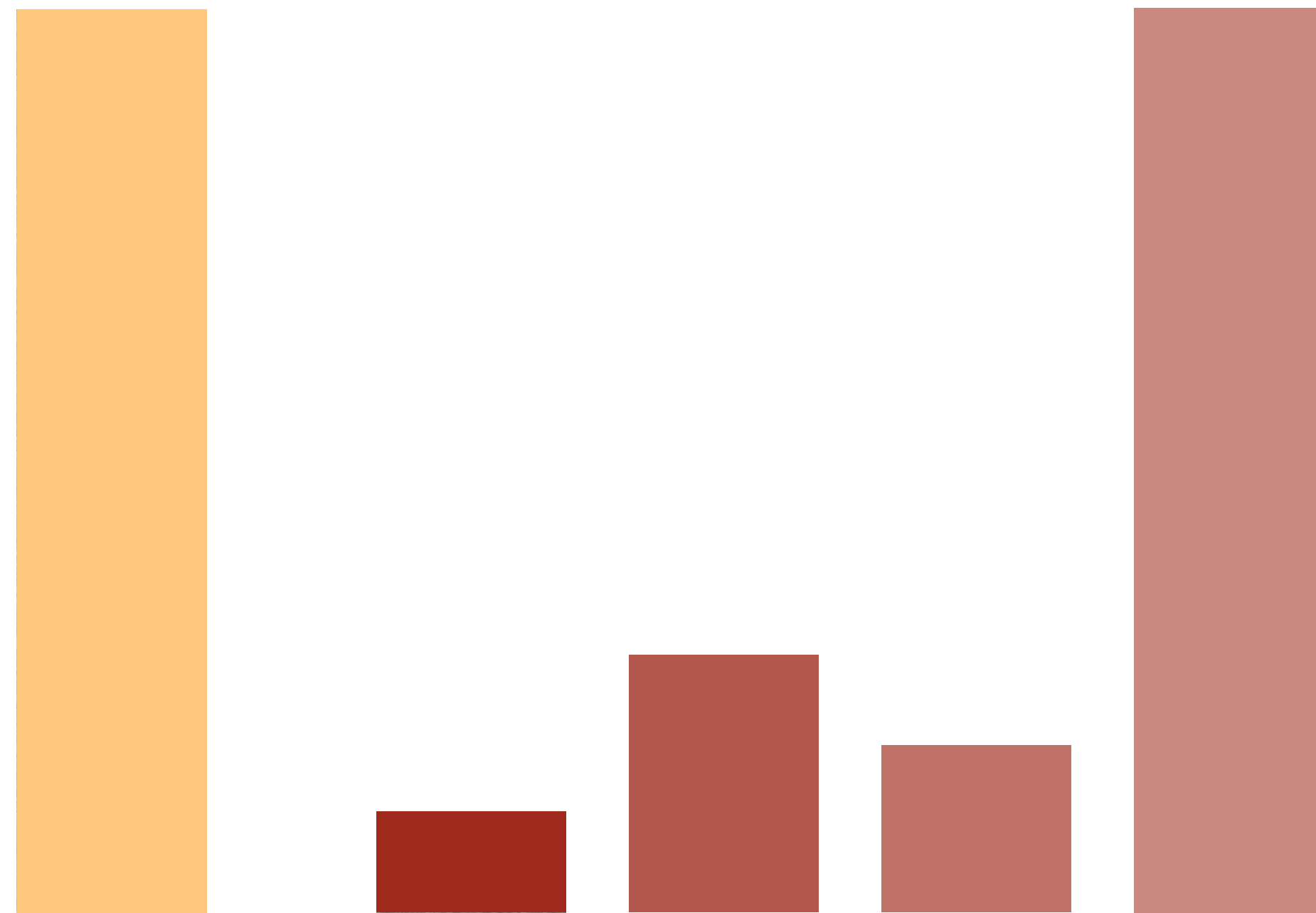
## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return



# Wie gut ist das allgemein?

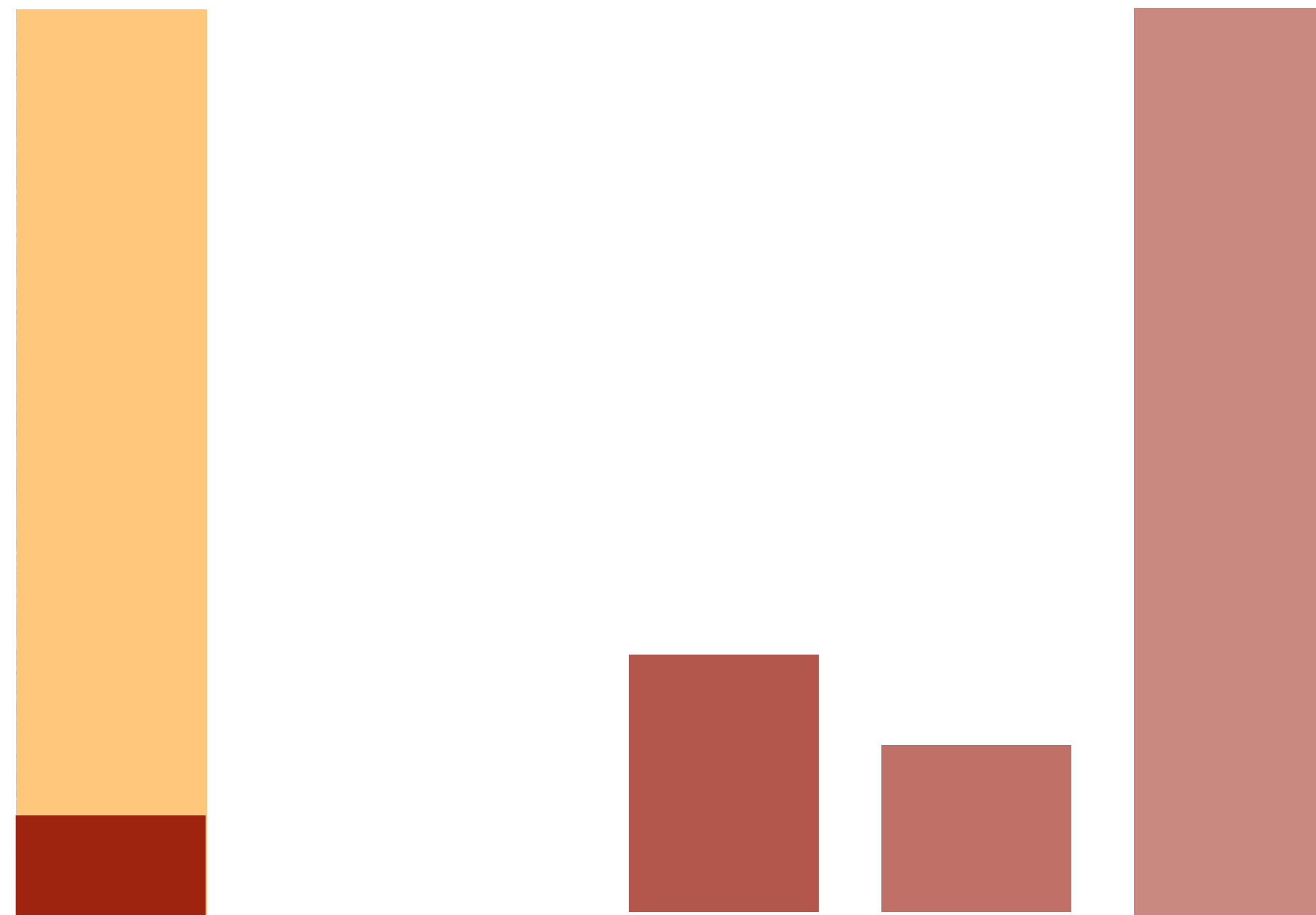
## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return



# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return



# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return



# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$G'_0$

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$G'_0$

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

# Wie gut ist das allgemein?

## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

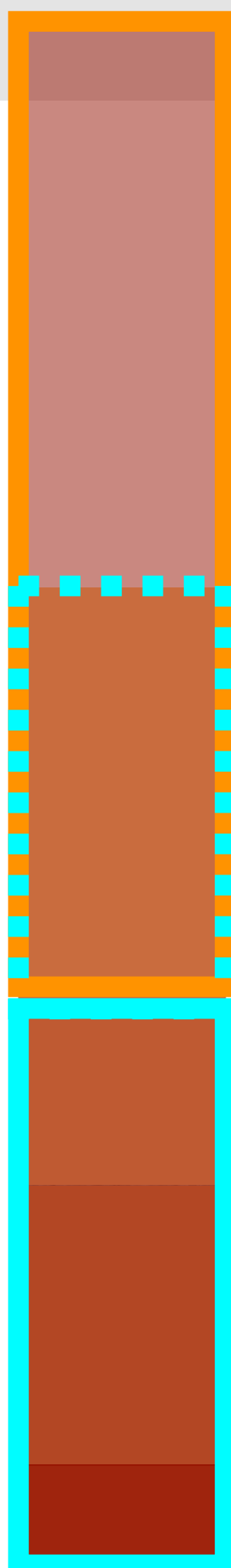
1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

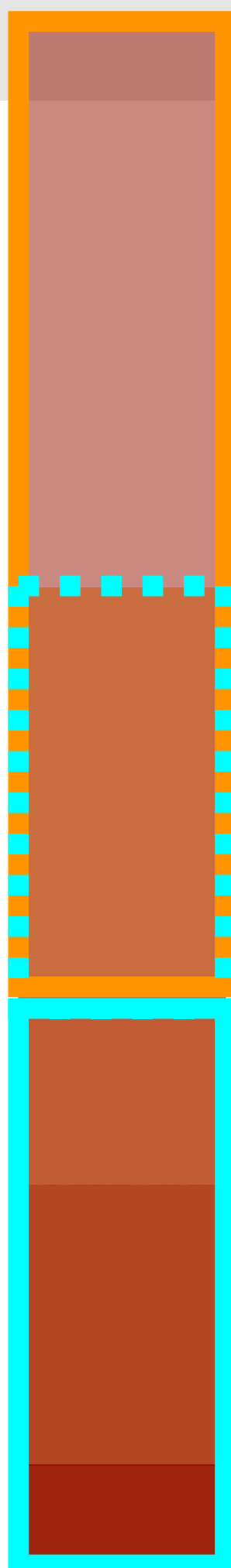
1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0 \quad G'_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

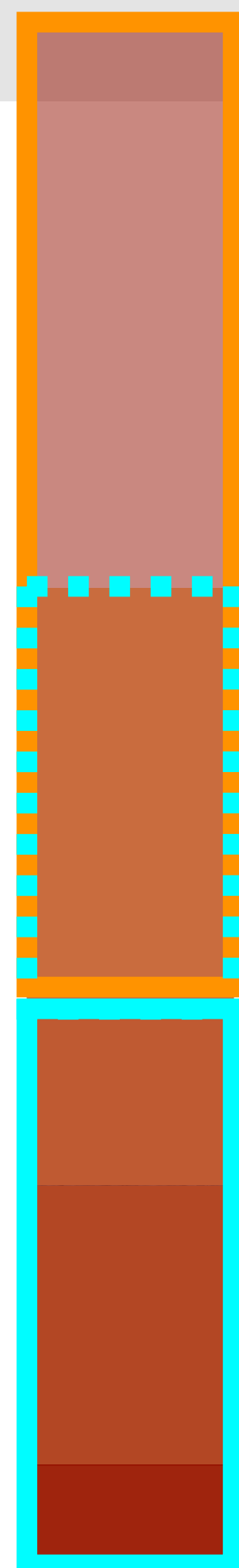
**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0$$



# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

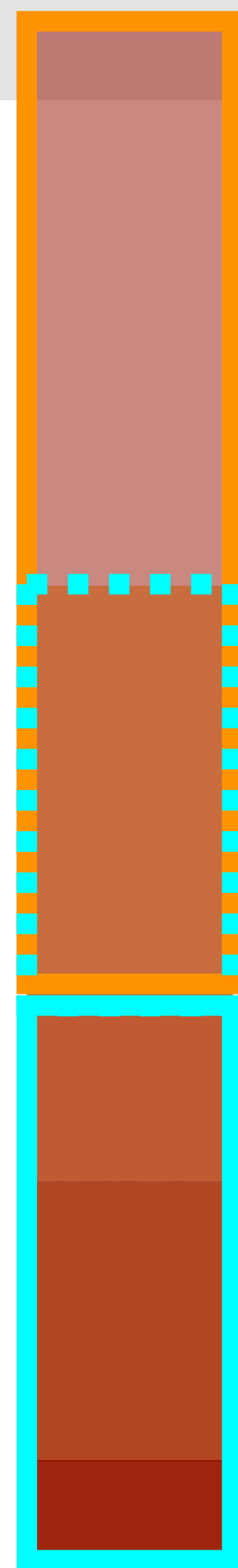
2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0 \geq p^*$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

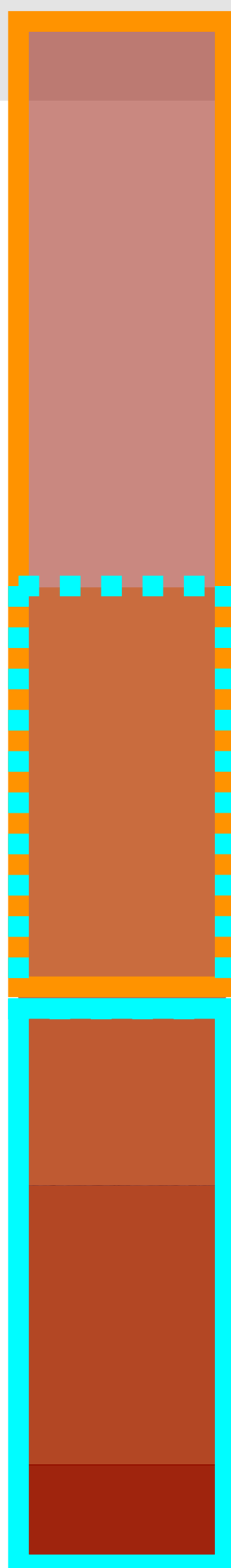
2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0 \geq p^*$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

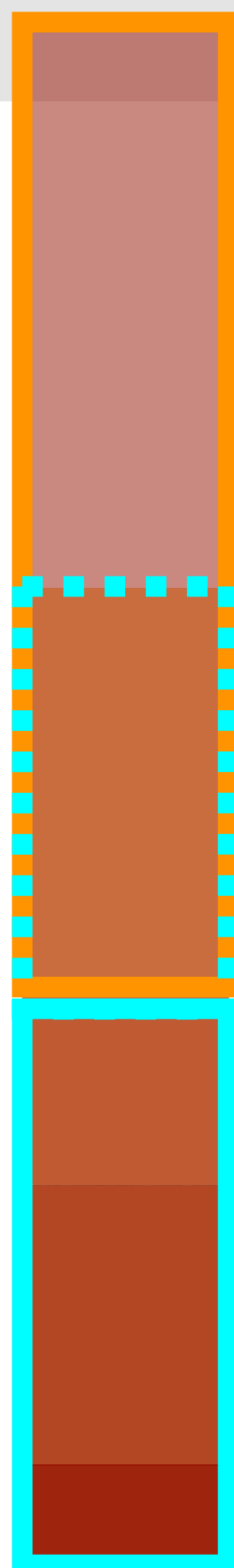
**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0 \geq p^*$$

$$OPT \leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

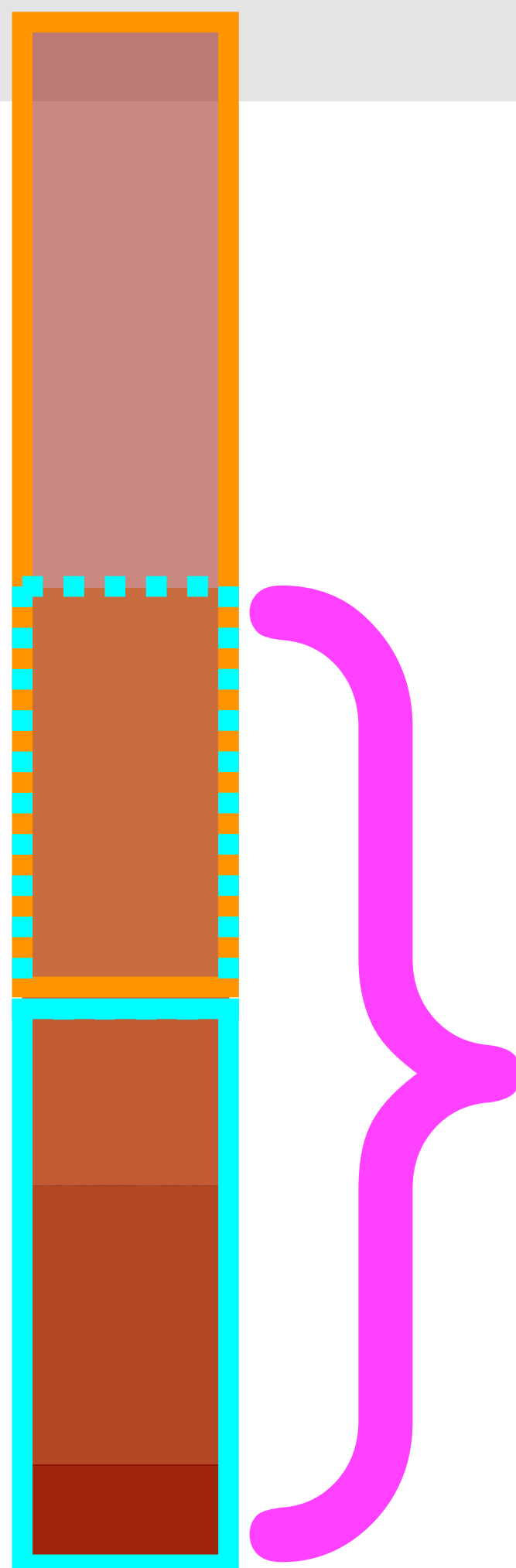
**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0 \geq p^*$$

$$OPT \leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$

2: return

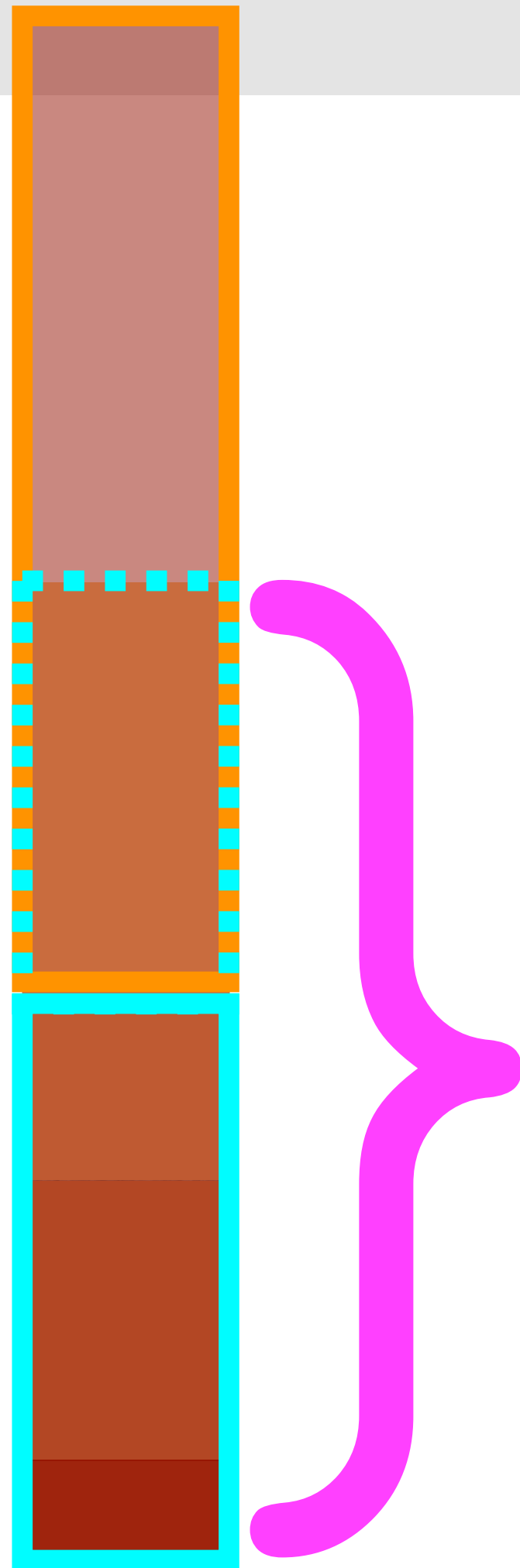
**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0$$

$$G'_0 \geq p^*$$

$$OPT \leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

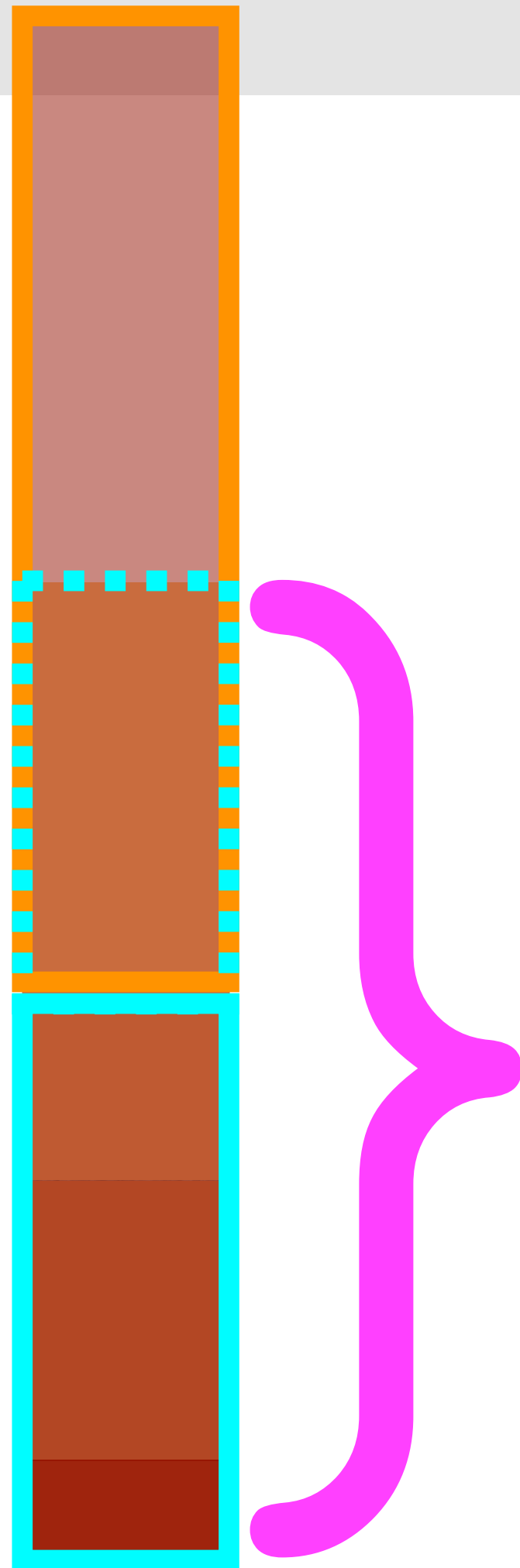
- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$G'_0 \geq G_0 \quad G'_0 \geq p^*$$

$$OPT \leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

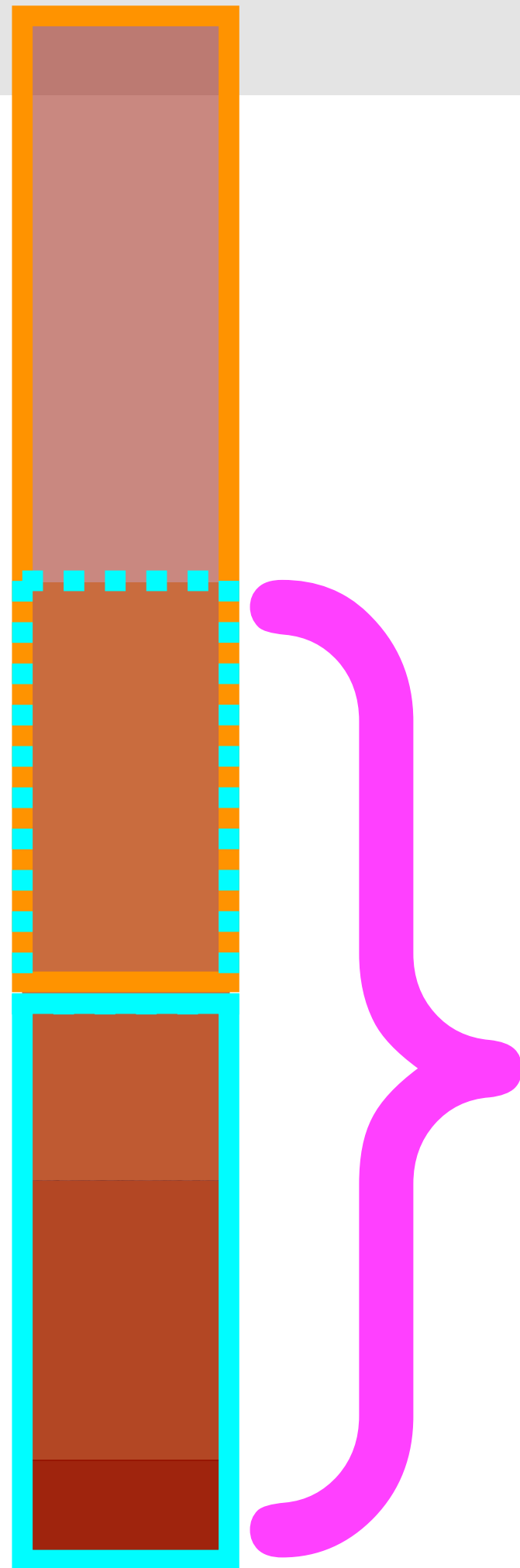
Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$\begin{aligned}
 G'_0 &\geq G_0 & G'_0 &\geq p^* \\
 OPT &\leq \text{FractionalKP} = G_0 + \sum x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{aligned}$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

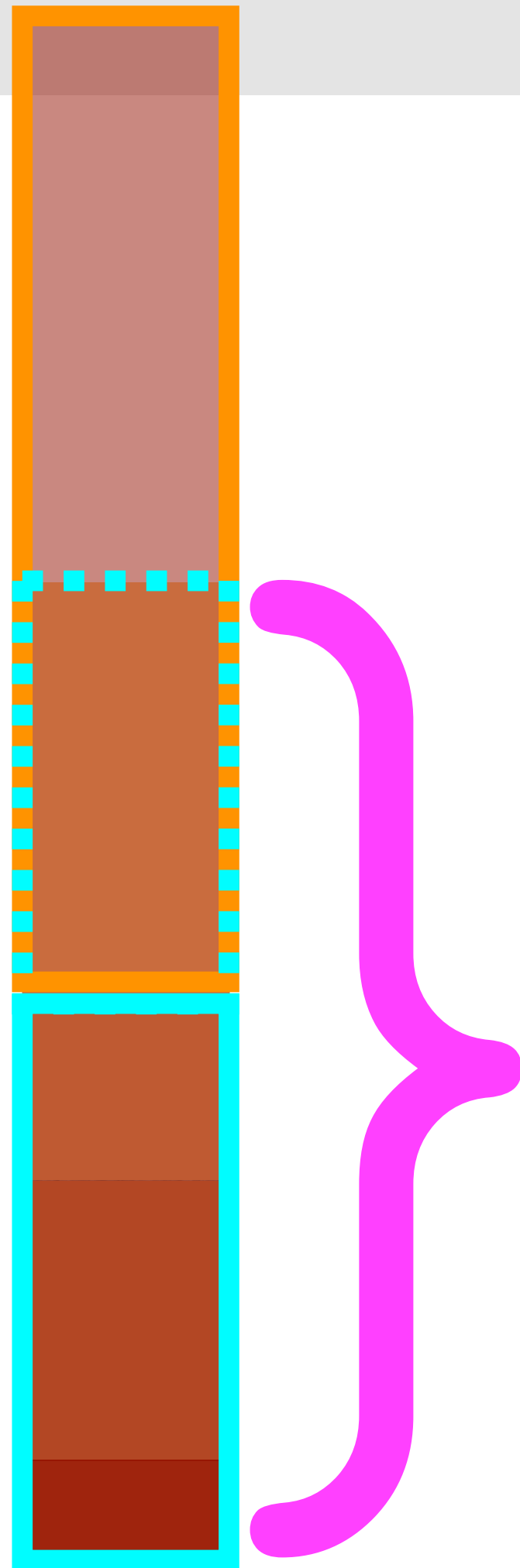
- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$\begin{aligned}
 G'_0 &\geq G_0 & G'_0 &\geq p^* \\
 OPT &\leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{aligned}$$



# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

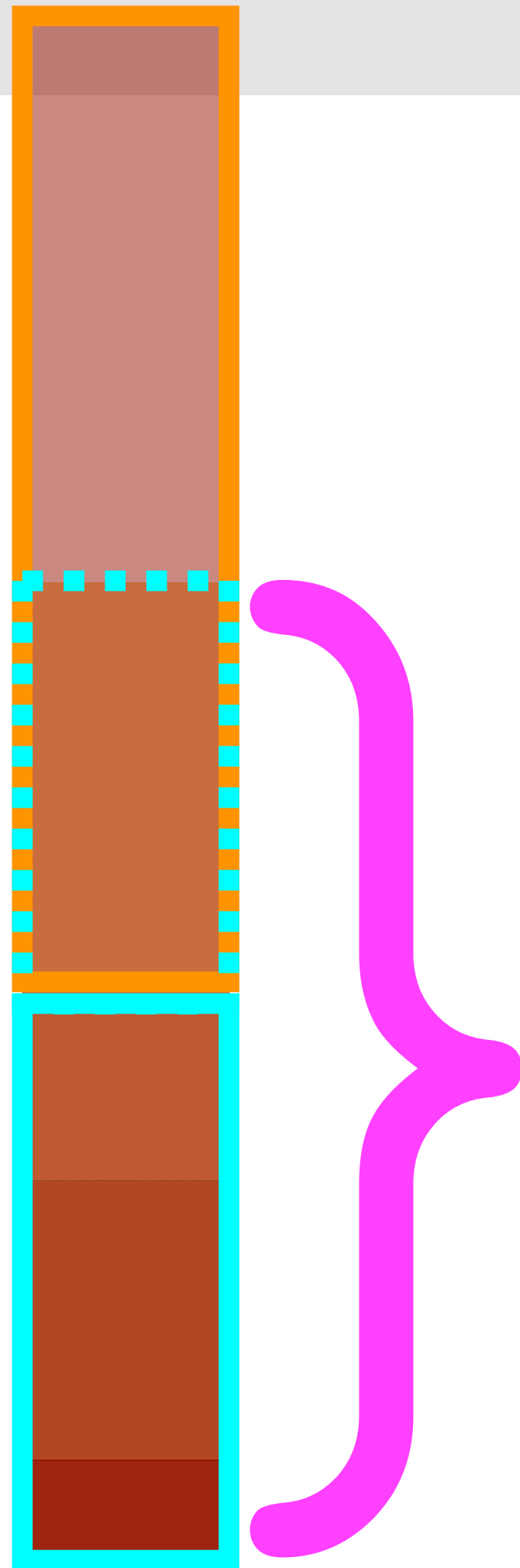
Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$\begin{aligned}
 G'_0 &\geq G_0 & G'_0 &\geq p^* \\
 OPT &\leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{aligned}$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

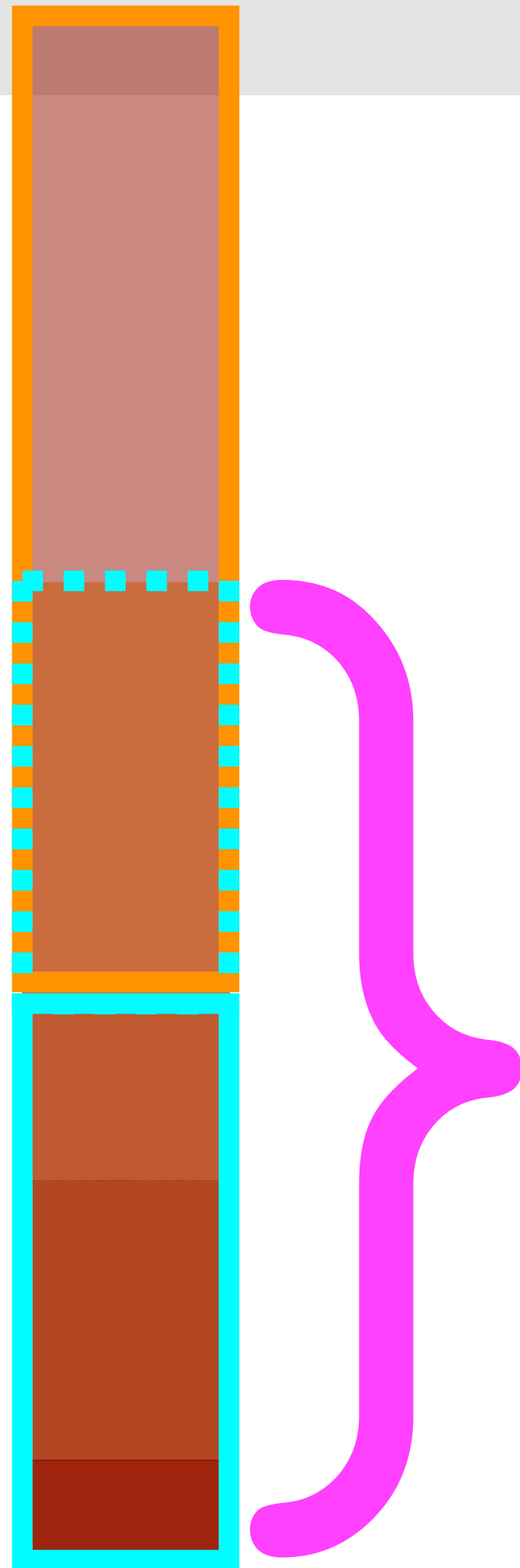
Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$\begin{aligned}
 G'_0 &\geq G_0 & G'_0 &\geq p^* \\
 OPT &\leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{aligned}$$

# Wie gut ist das allgemein?



## Algorithmus 4.3 Greedy-Approximation

Eingabe:  $z_1, \dots, z_n, Z, p_1, \dots, p_n$

Ausgabe:  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i < Z$  und Wert  $G'_0 := \sum_{i \in S} p_i$

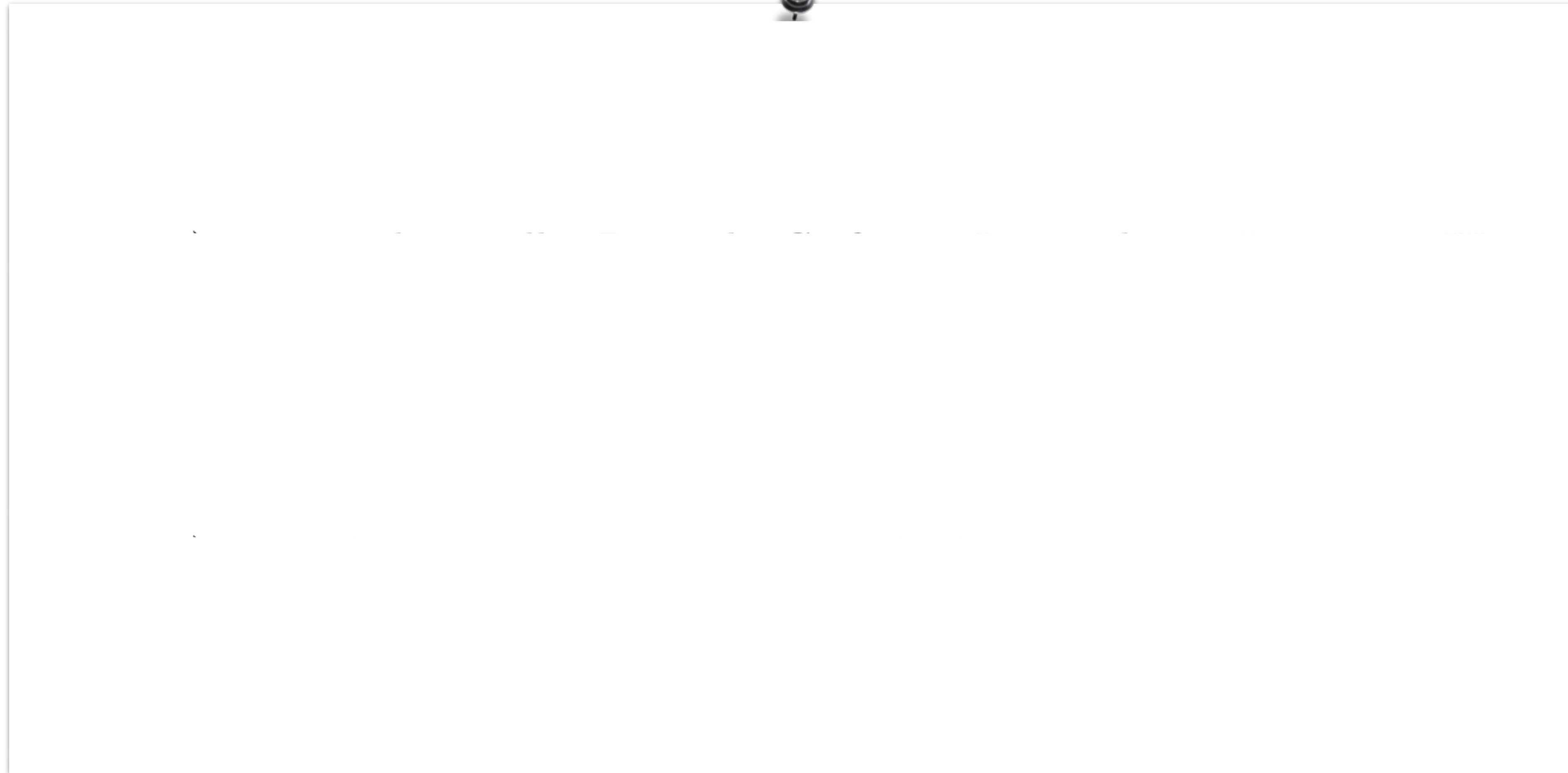
- 1:  $G'_0 = \max_{G_0} \max\{p_i \mid z_i < Z, i \in \{1, \dots, n\}\}$
- 2: return

**Satz 4.4.** Algorithmus 4.3 berechnet eine Lösung mit  $G'_0$ ;  
im Vergleich mit dem Optimalwert  $OPT$  gilt:  $G'_0 \geq \frac{1}{2}OPT$ .

$$\begin{aligned}
 & G'_0 \geq G_0 & G'_0 & \geq p^* \\
 OPT & \leq \text{FractionalKP} = G_0 + x_i p_i \leq G_0 + p^* \leq 2G'_0
 \end{aligned}$$

# 4.3 Approximationsalgorithmen

# Approximation



# Approximation

**Definition 4.5** (Approximationsalgorithmus).

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX



# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  $c \leq 1$ )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  $c \leq 1$ )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für Minimierungsproblem wiederum:



# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$
  - b) es gilt für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$ :  $\text{ALG}(I) \leq c \cdot \text{OPT}(I)$  (dabei ist  $c \geq 1$ )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$
  - b) es gilt für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$ :  **$\text{ALG}(I) \leq c \cdot \text{OPT}(I)$**  (dabei ist  $c \geq 1$ )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$
  - b) es gilt für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$ :  **$\text{ALG}(I) \leq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \geq 1$** )

# Approximation

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein **Maximierungsproblem MAX** ist ein Algorithmus ALG ein *c-Approximationsalgorithmus* für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt  **$\text{ALG}(I) \geq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \leq 1$** )
2. Entsprechend für **Minimierungsproblem** wiederum:
  - a) ALG liefert in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert  $\text{ALG}(I)$
  - b) es gilt für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$ :  **$\text{ALG}(I) \leq c \cdot \text{OPT}(I)$**  (dabei ist  **$c \geq 1$** )

# 4.4 Bessere Approximation



# Ideen:

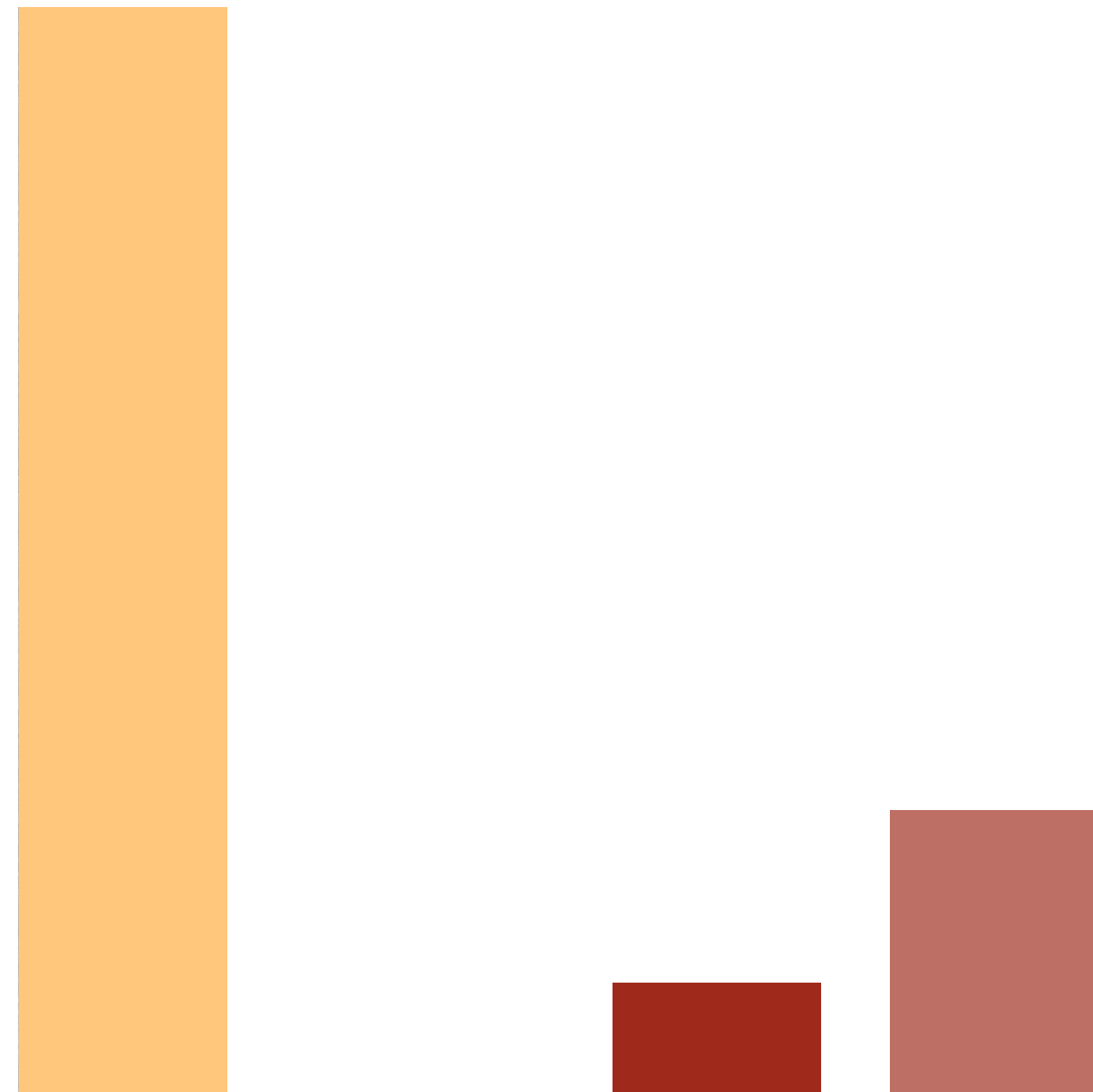
# Ideen:



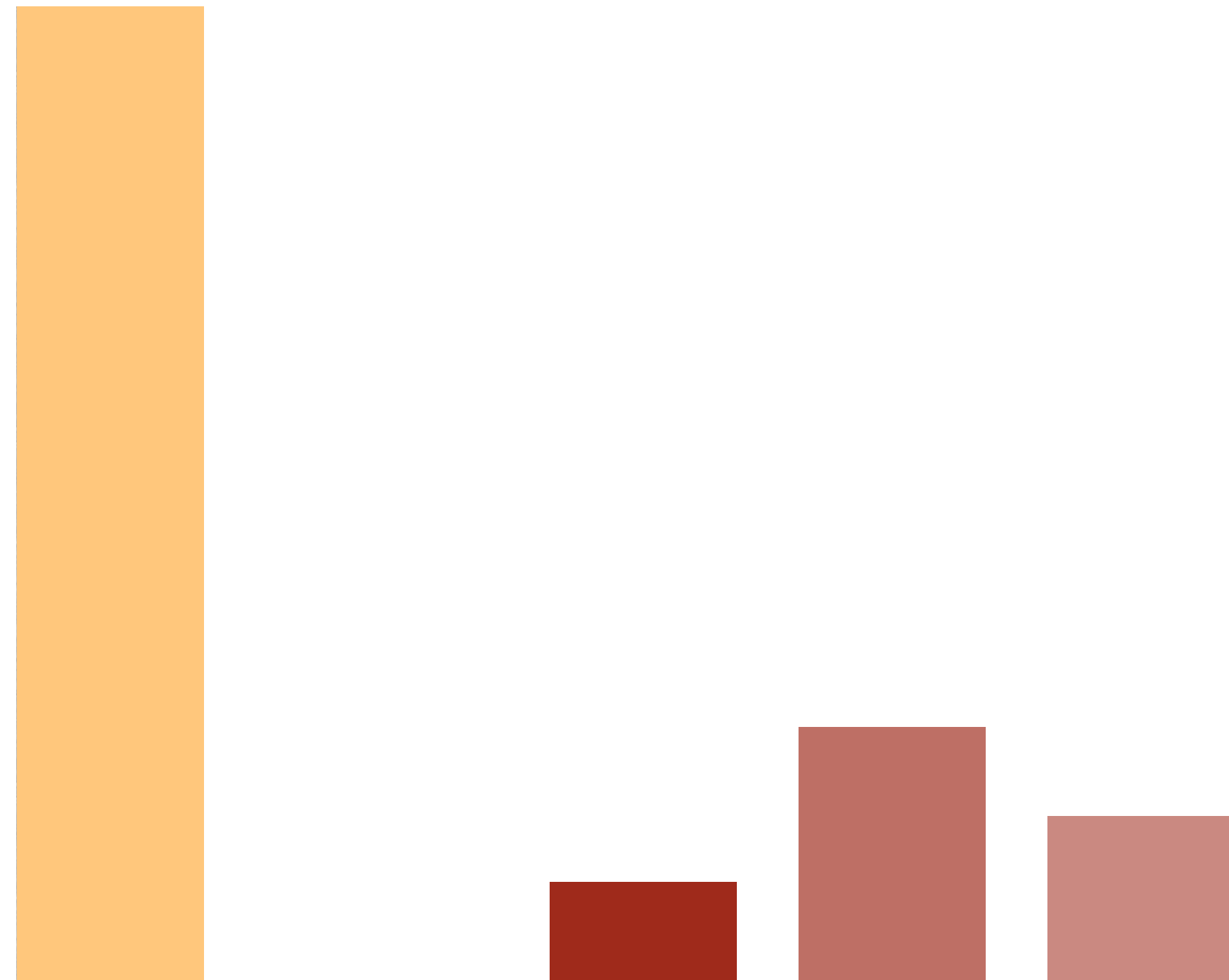
# Ideen:



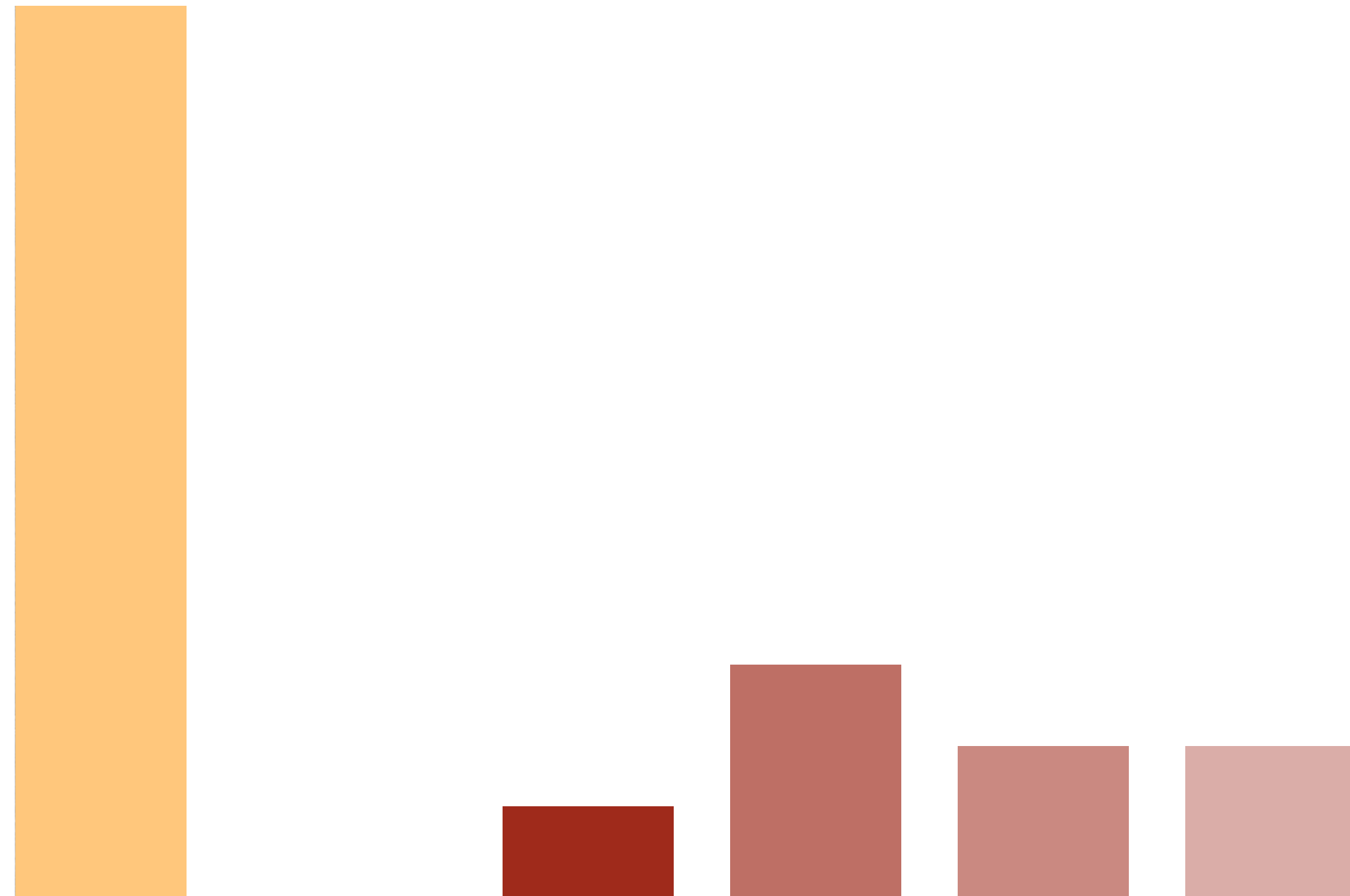
# Ideen:



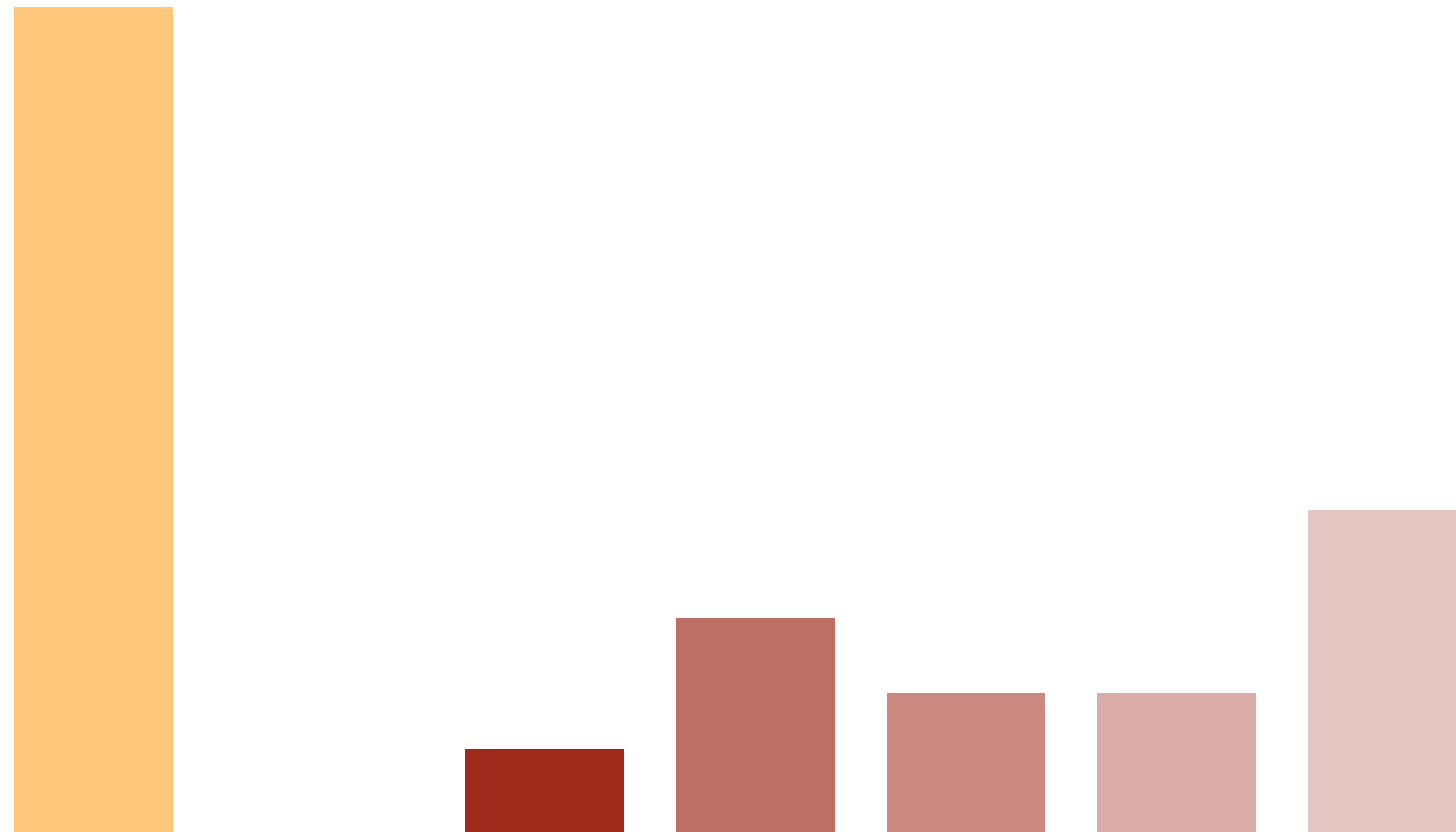
# Ideen:



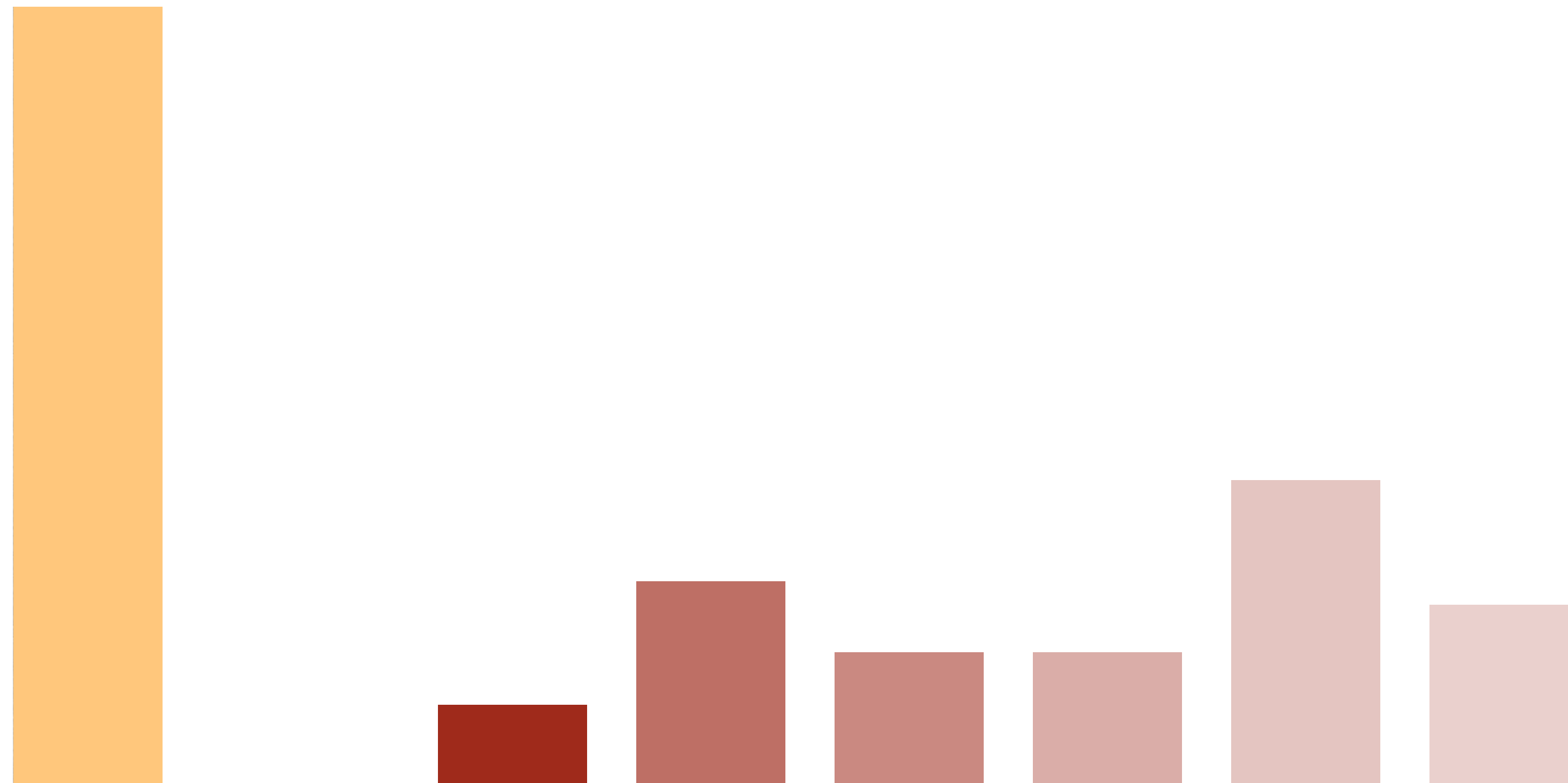
# Ideen:



# Ideen:

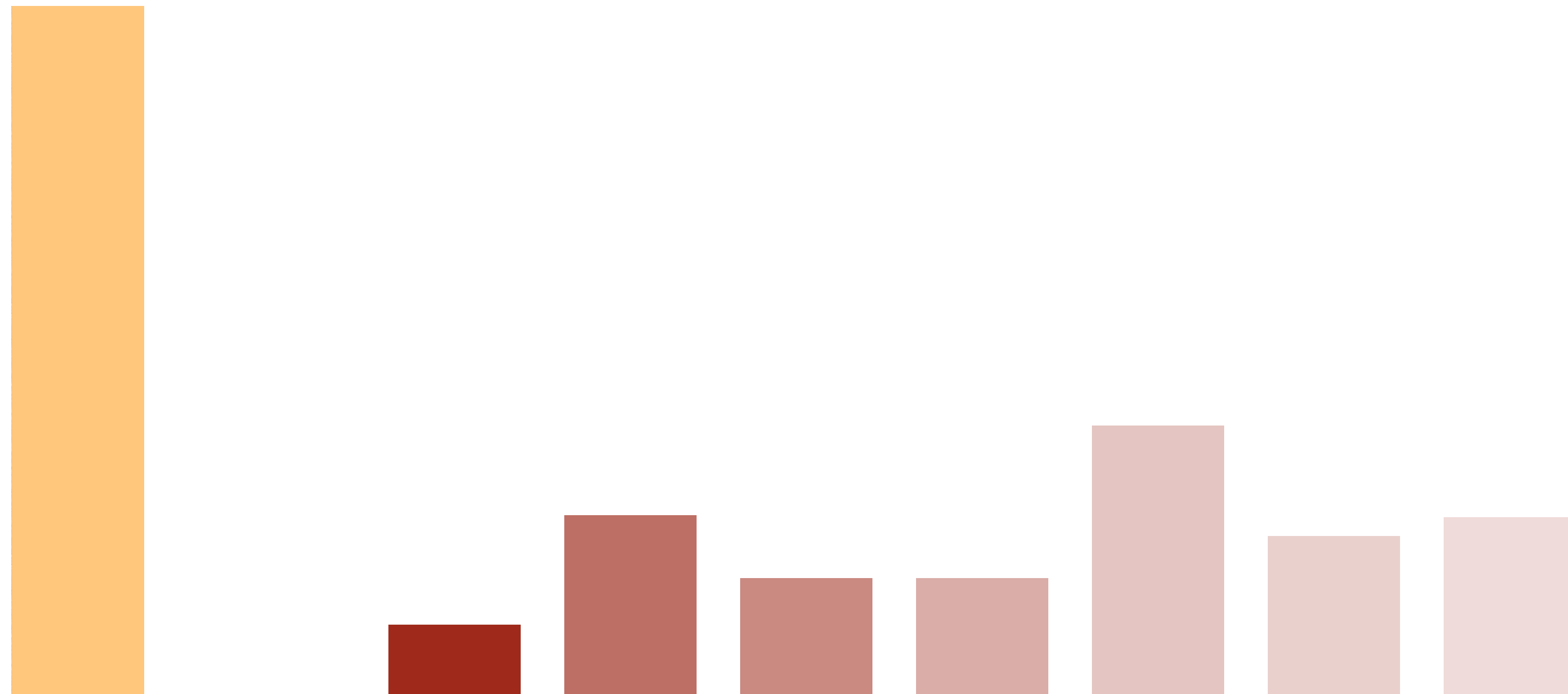


# Ideen:



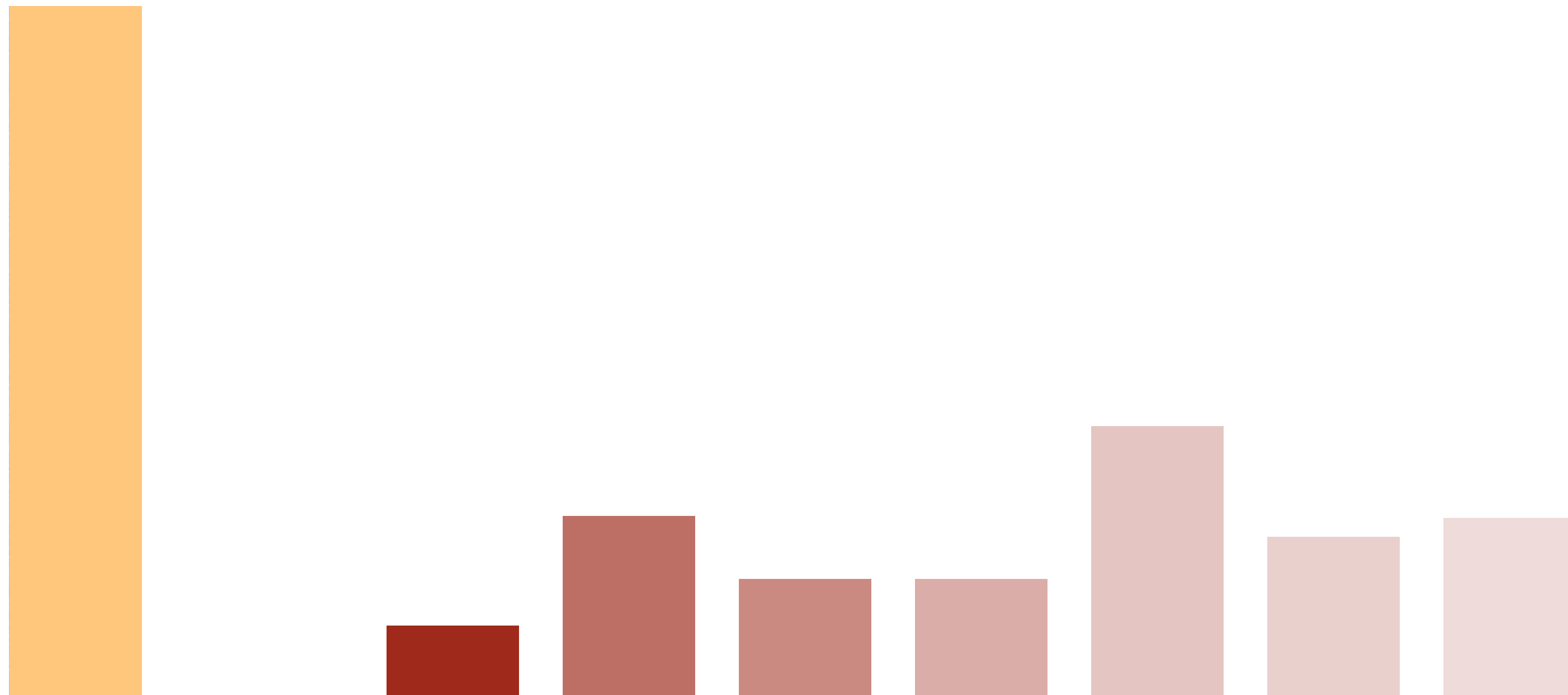


# Ideen:



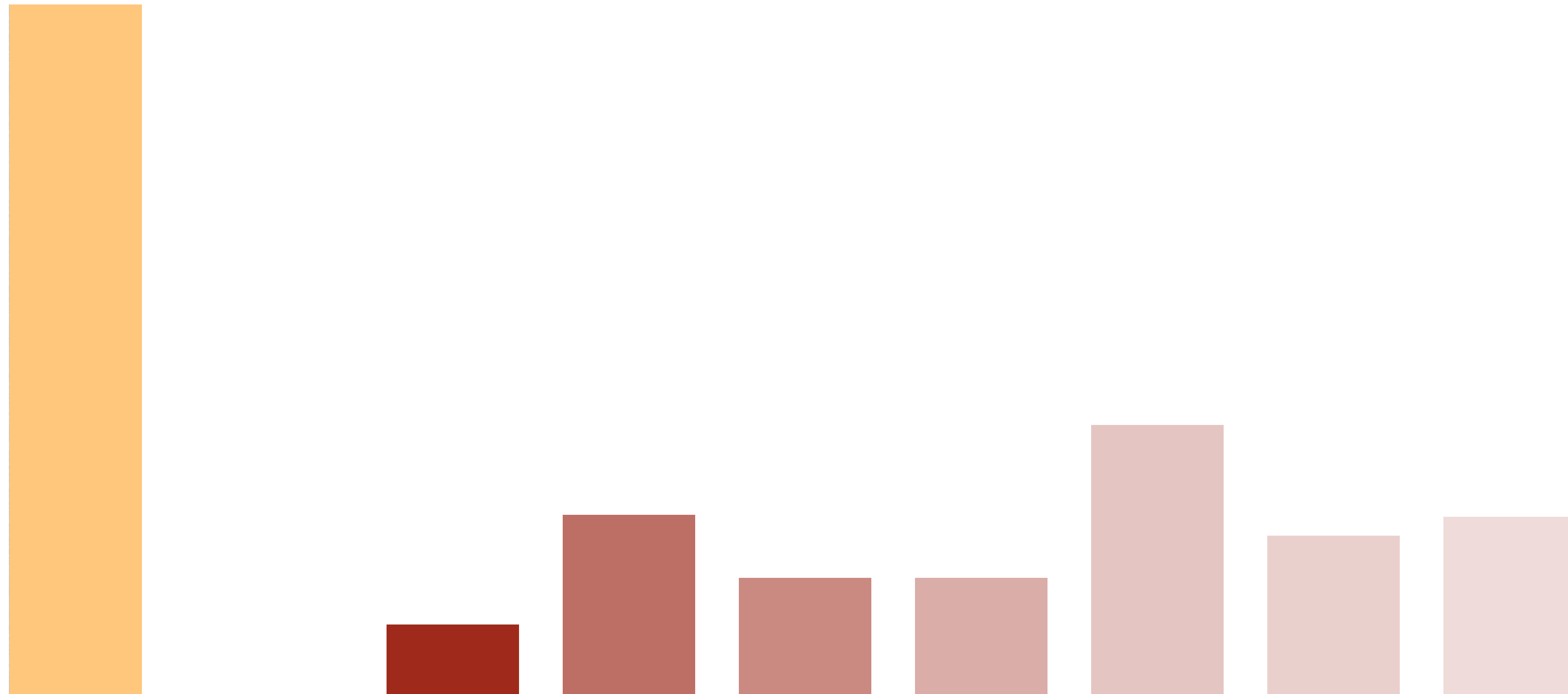
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



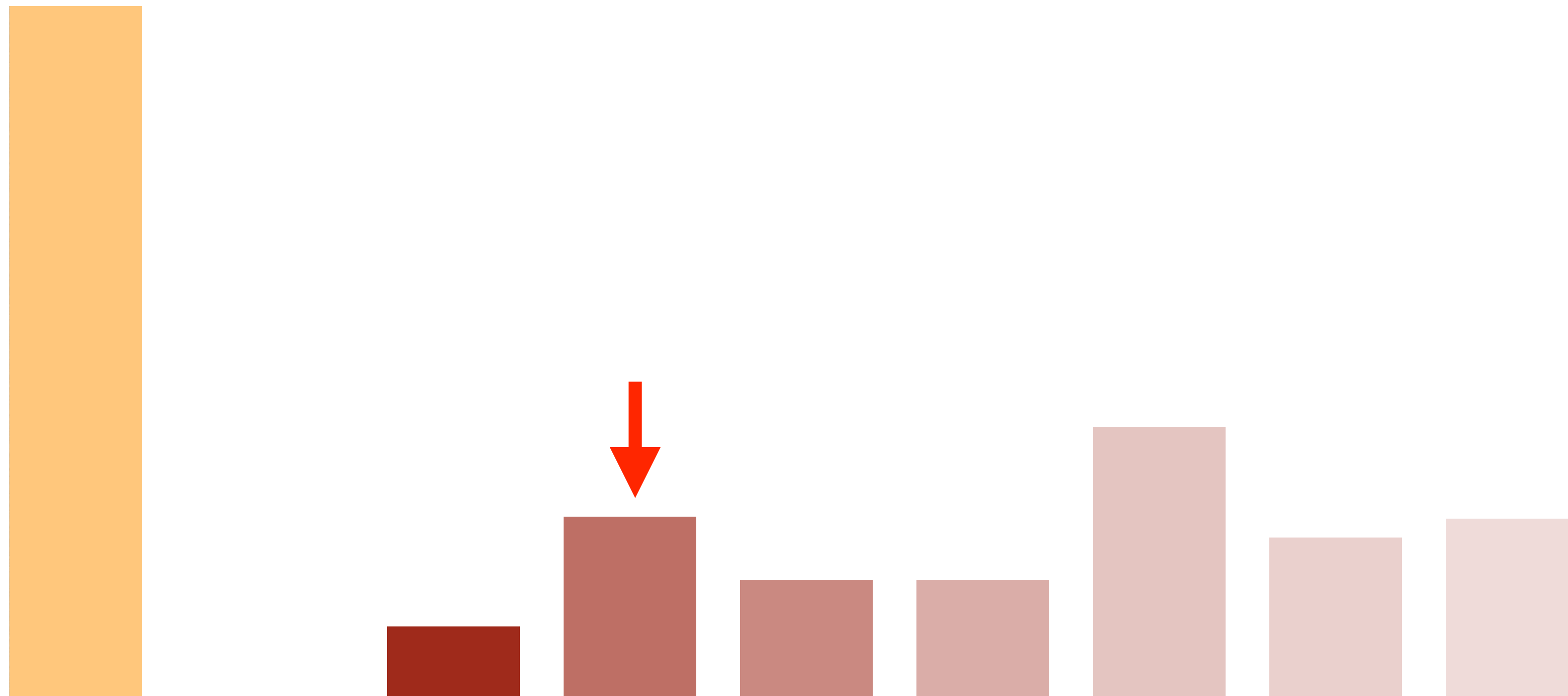
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



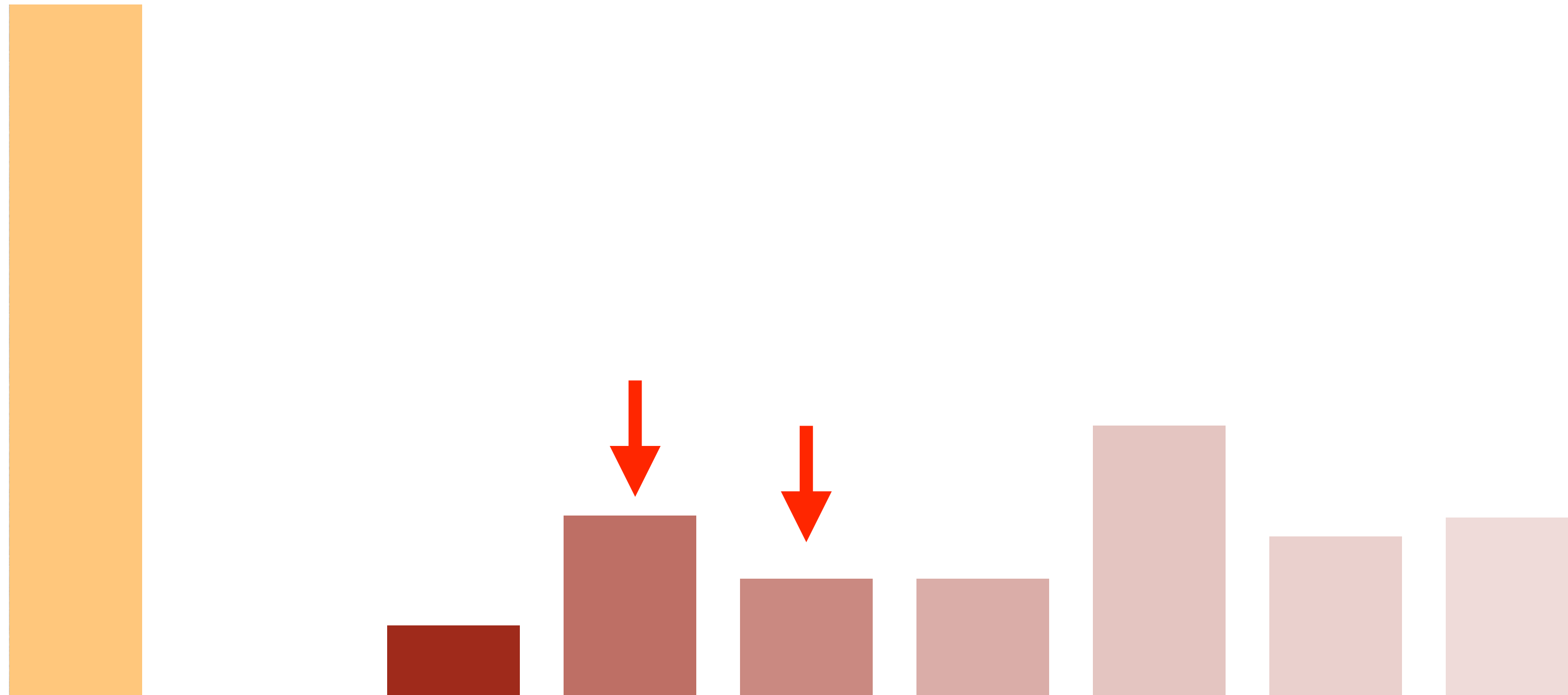
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



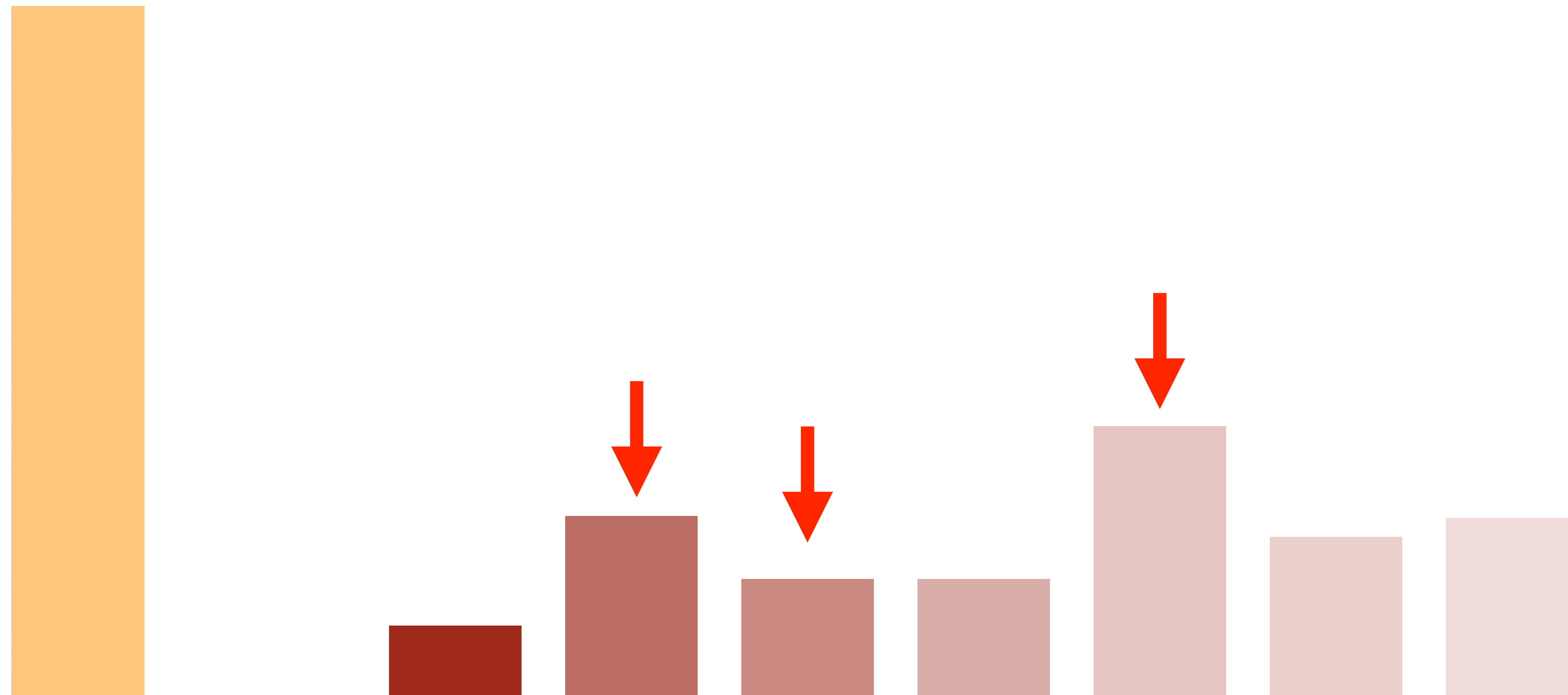
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



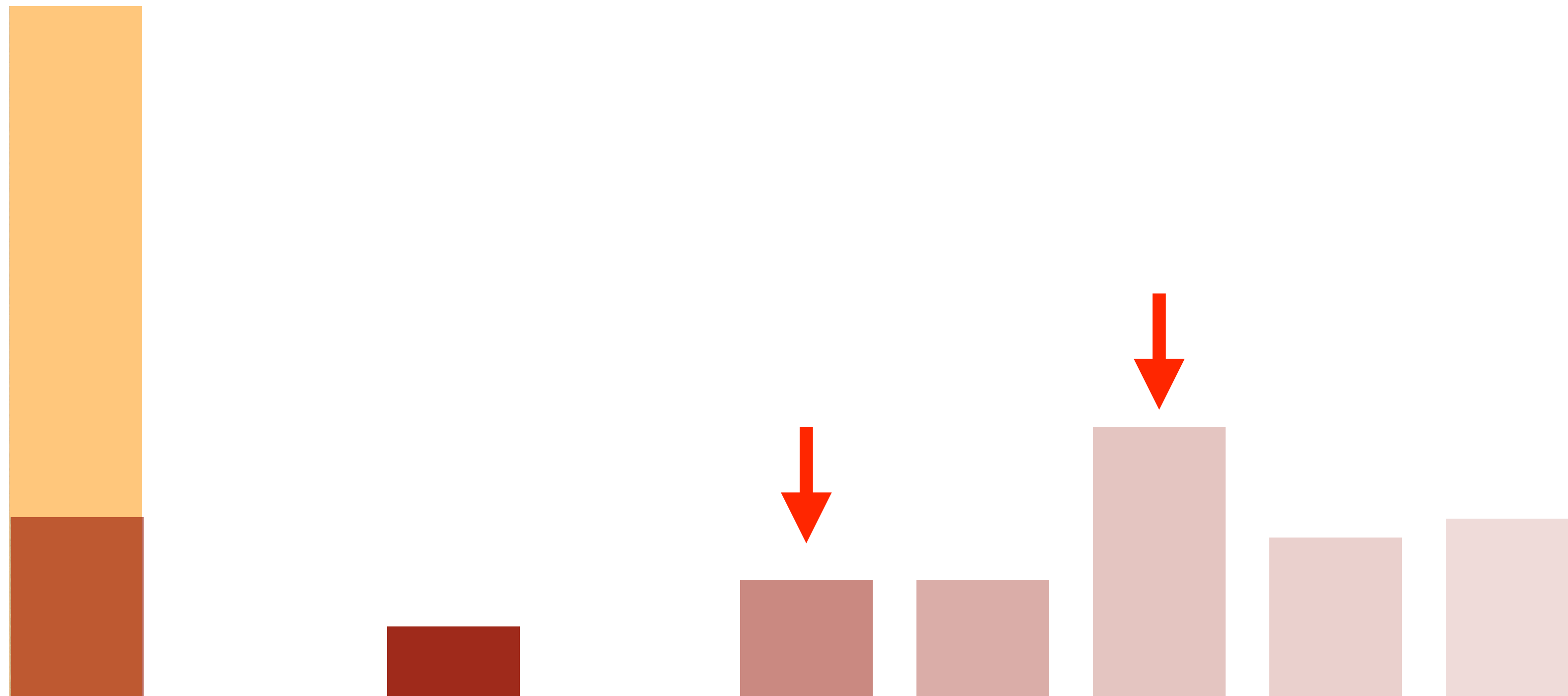
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



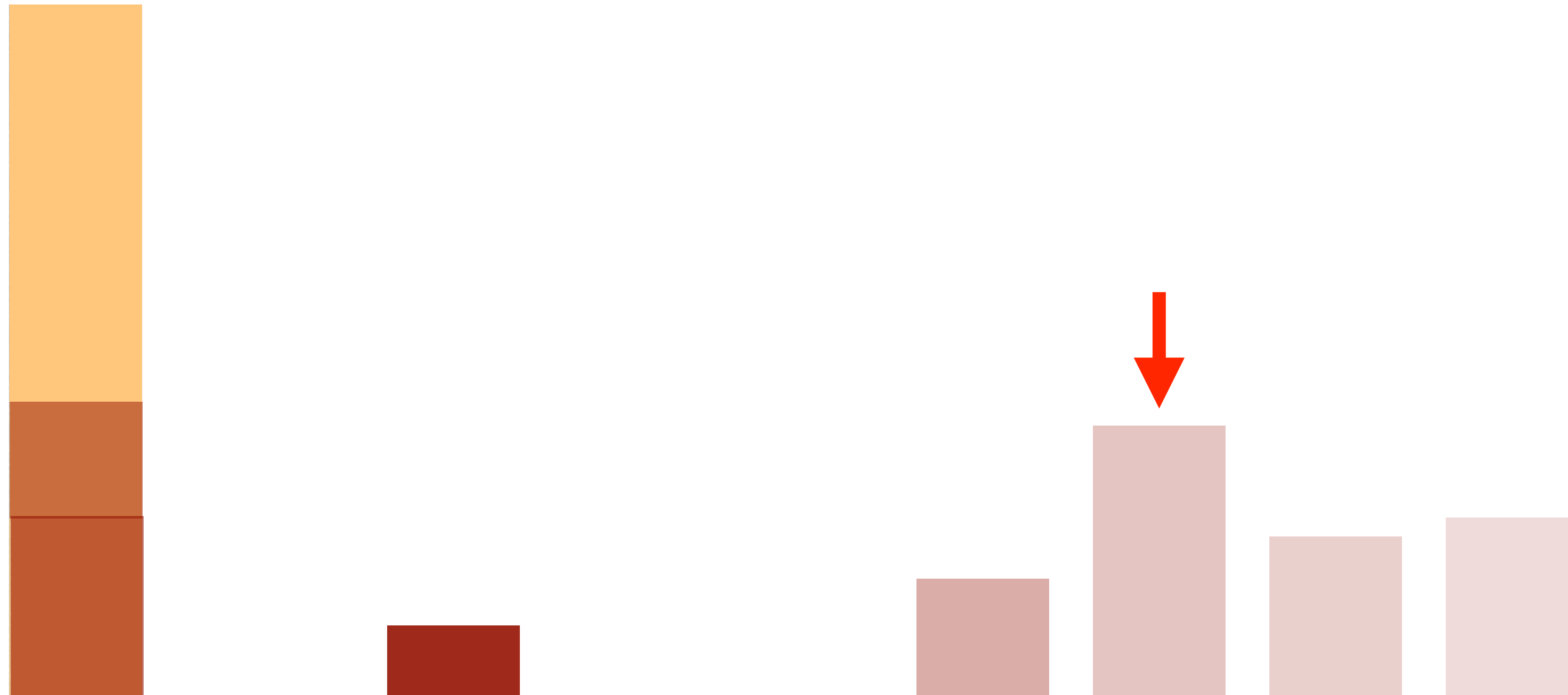
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



# Ideen:

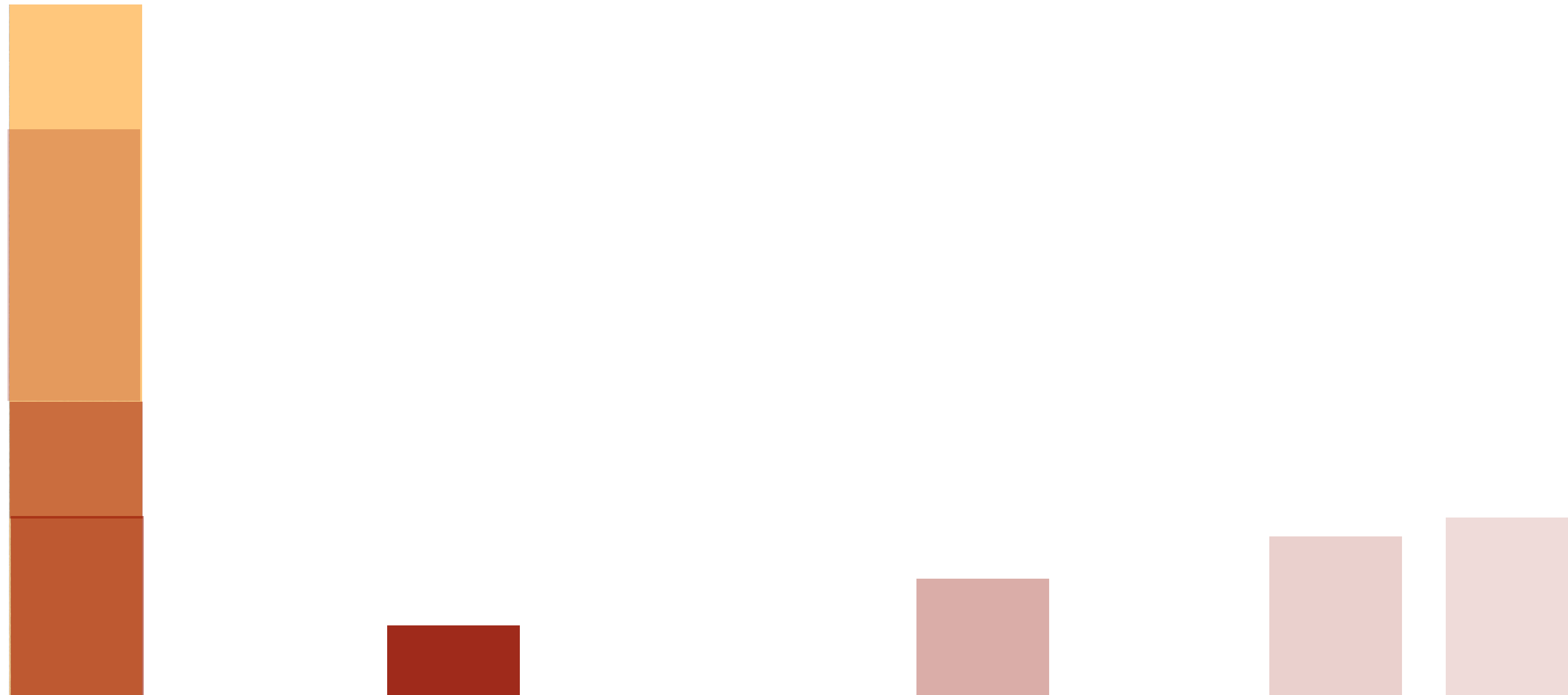
- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.





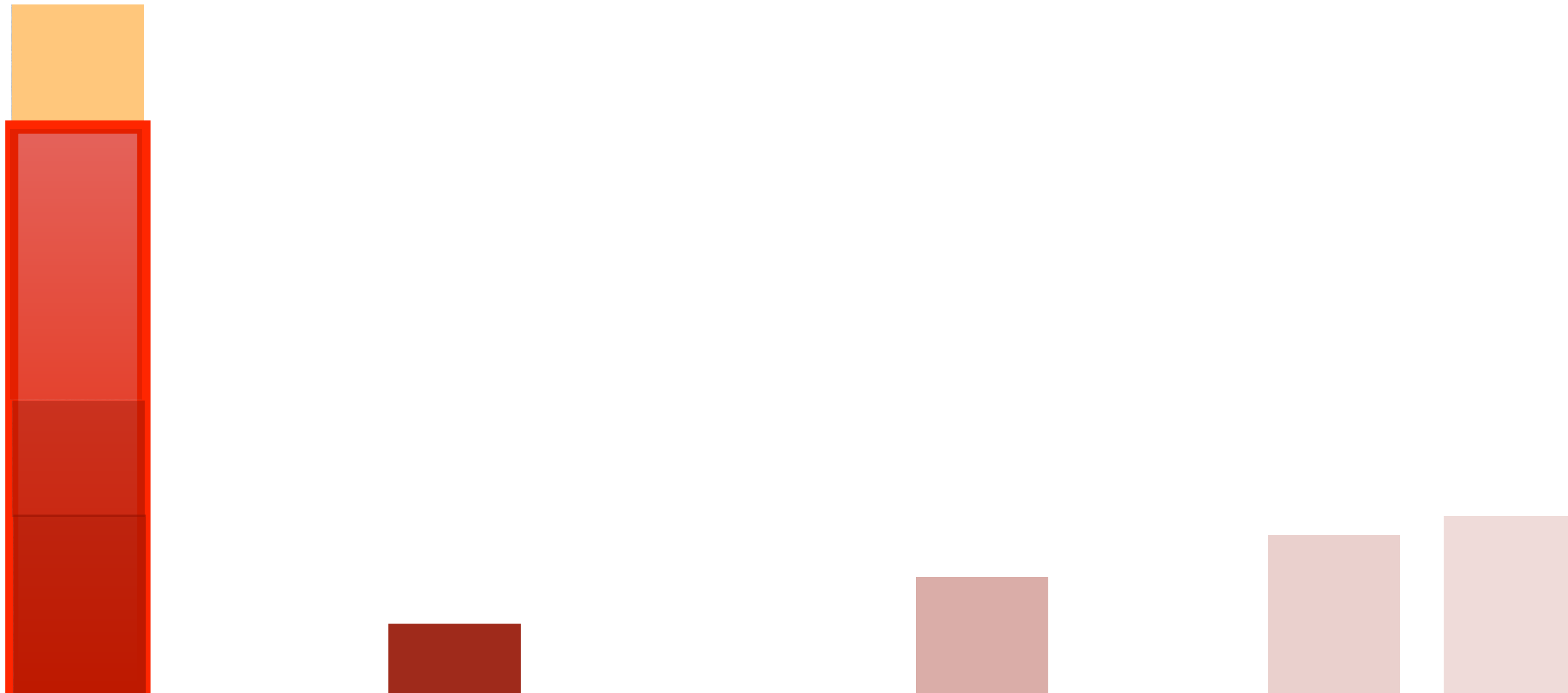
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



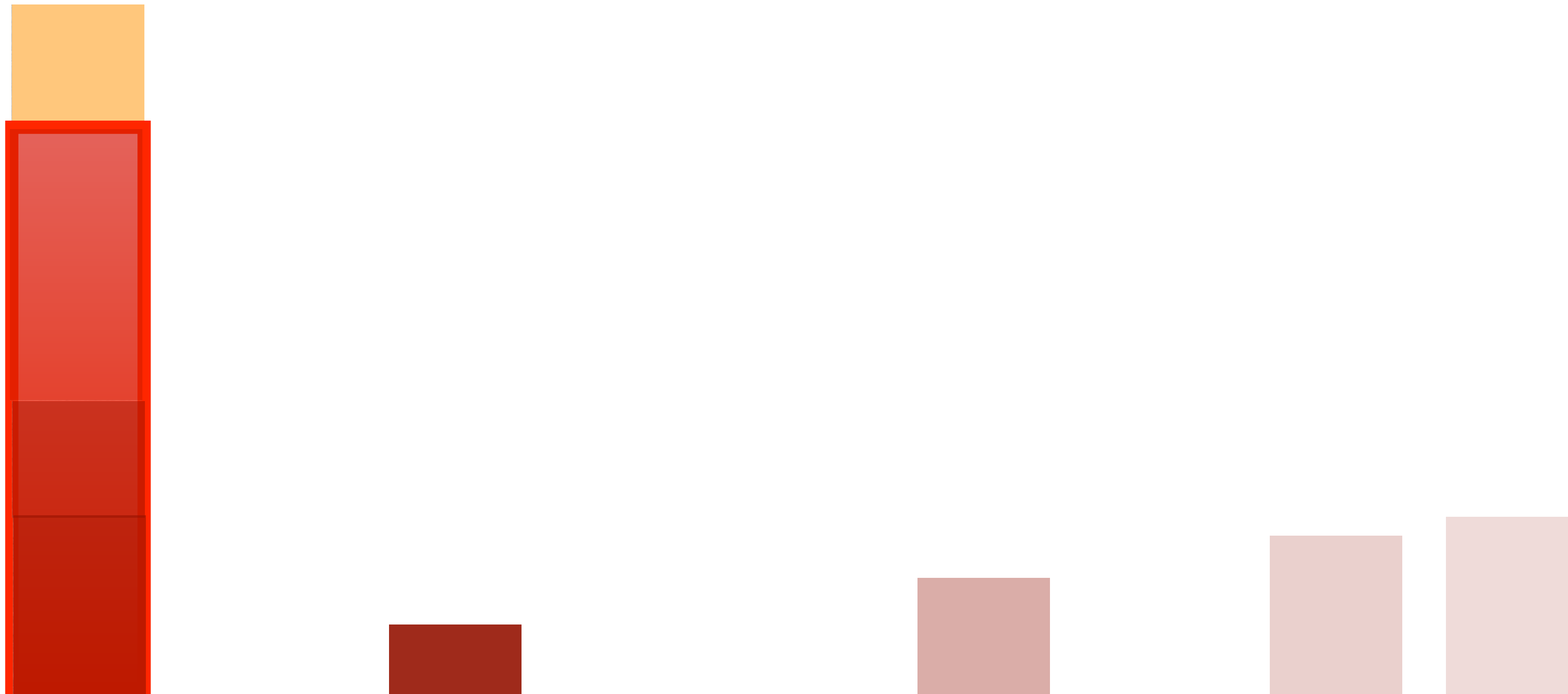
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.



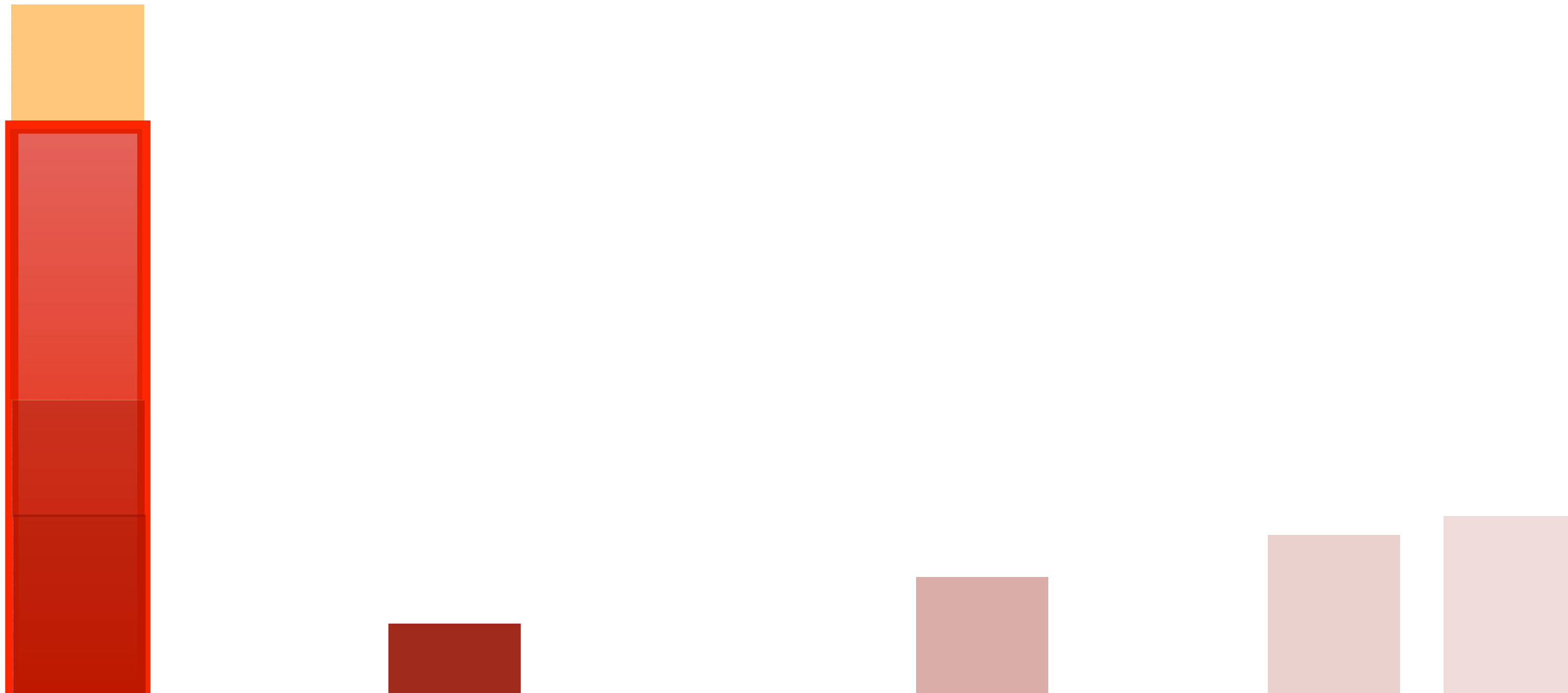
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



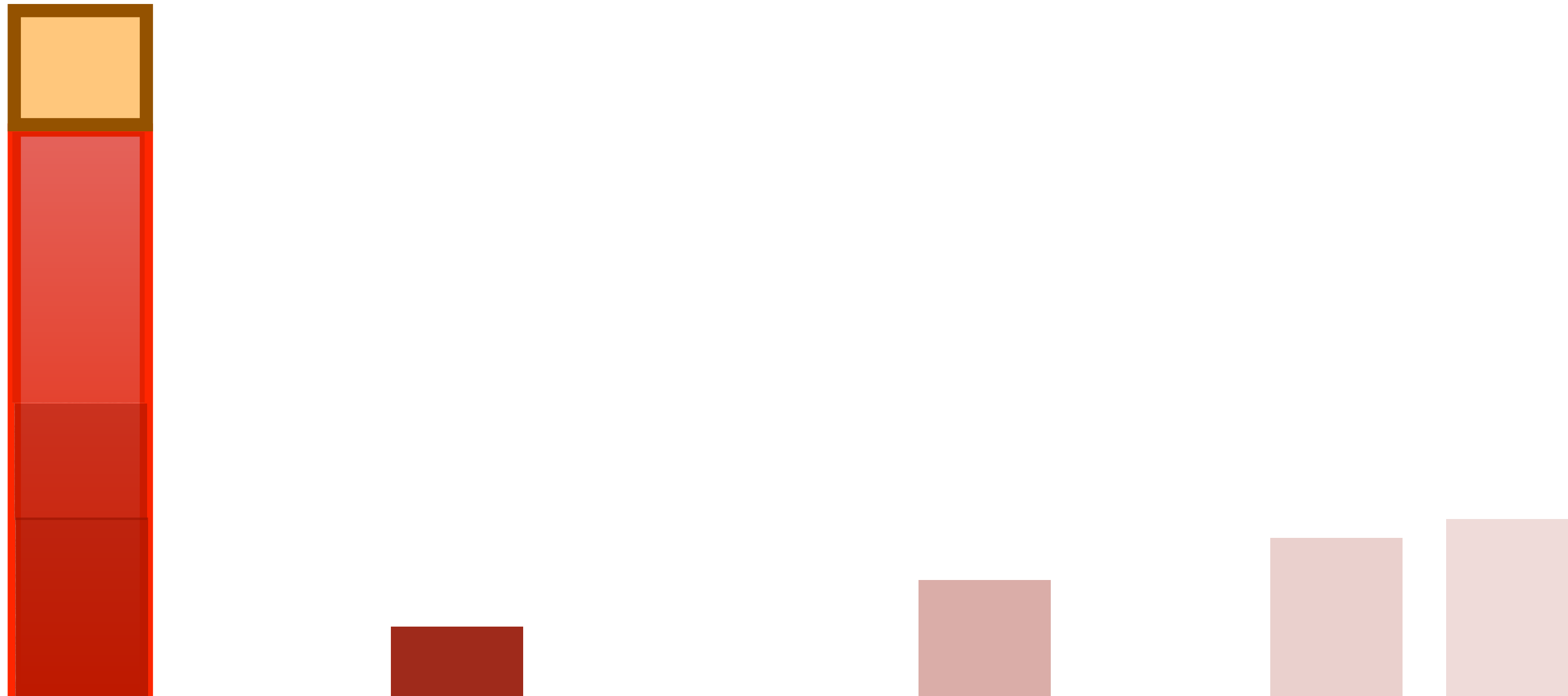
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



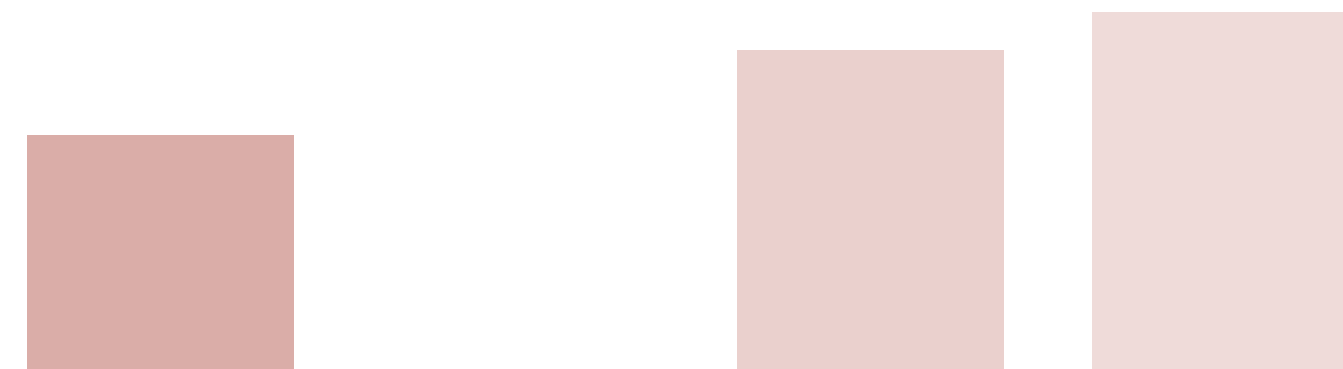
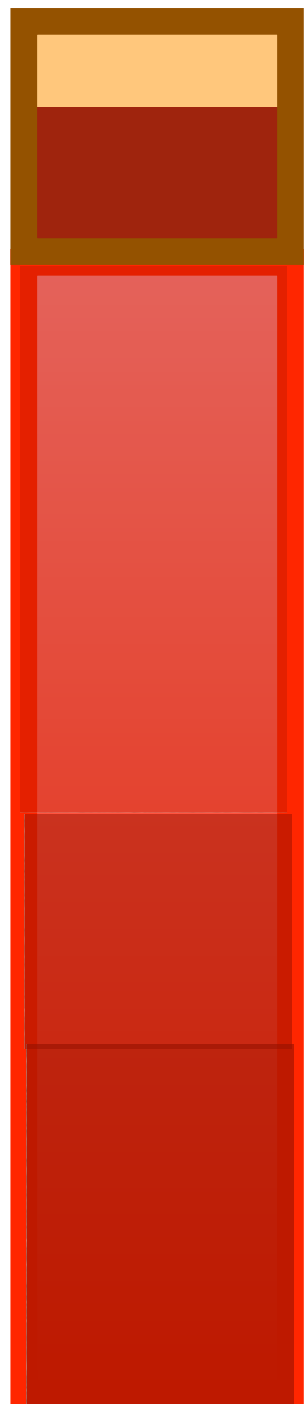
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



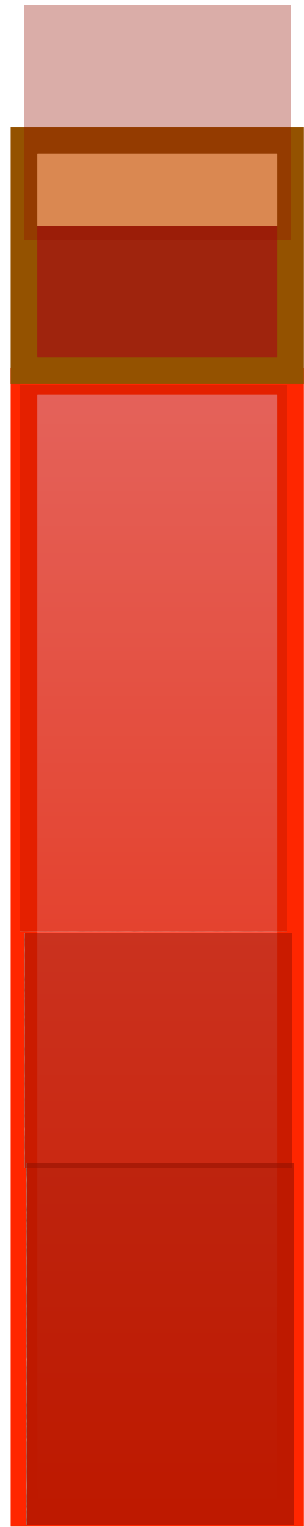
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



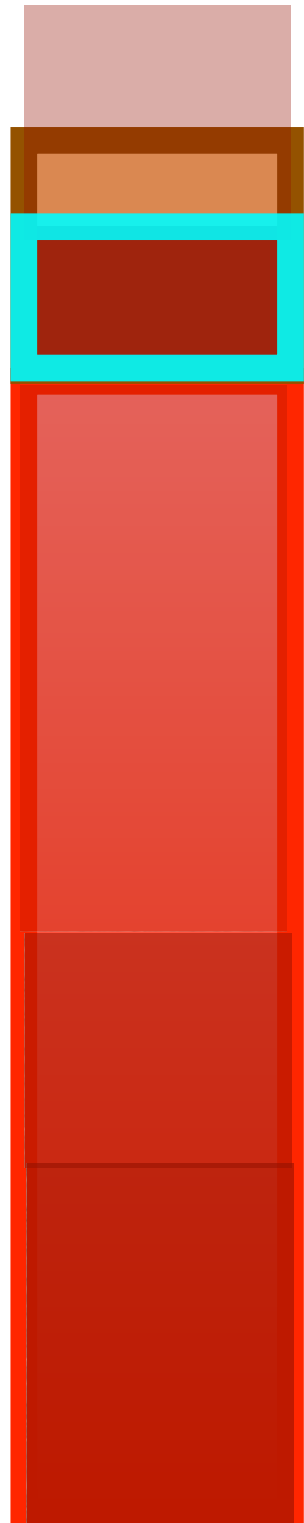
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.



# Ideen:

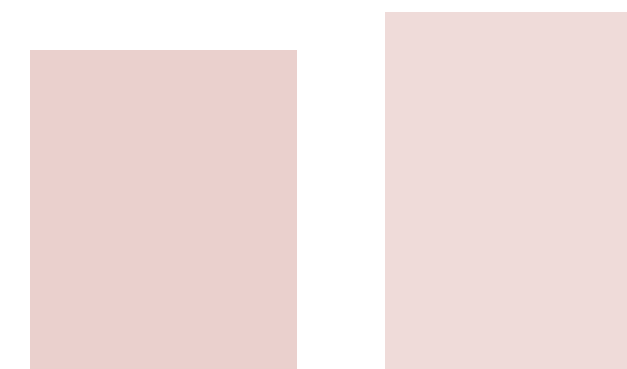
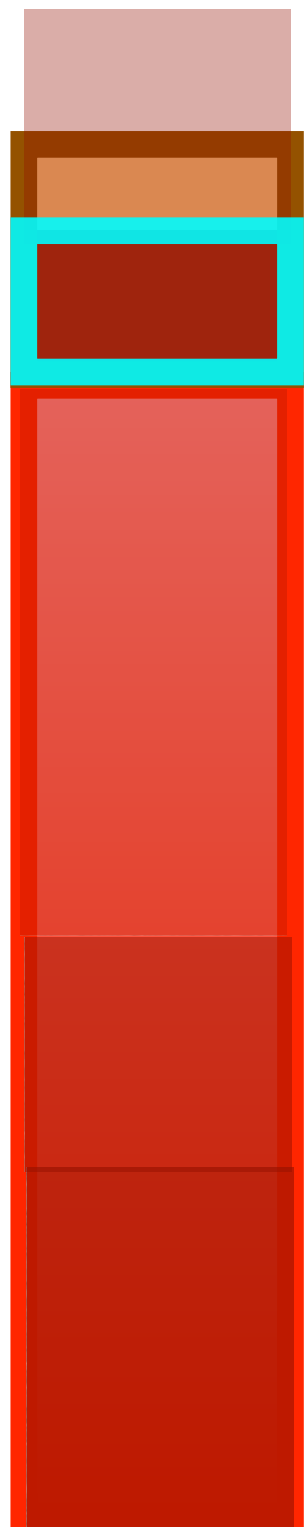
- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.





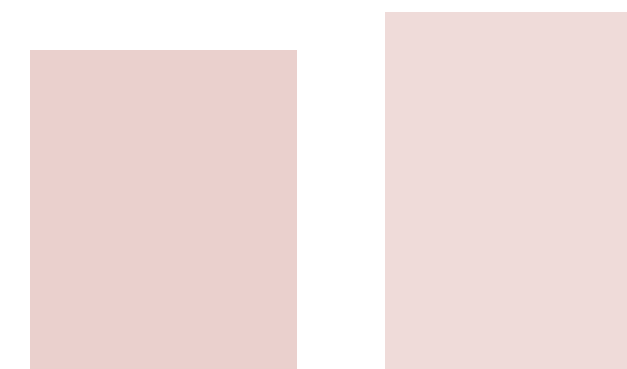
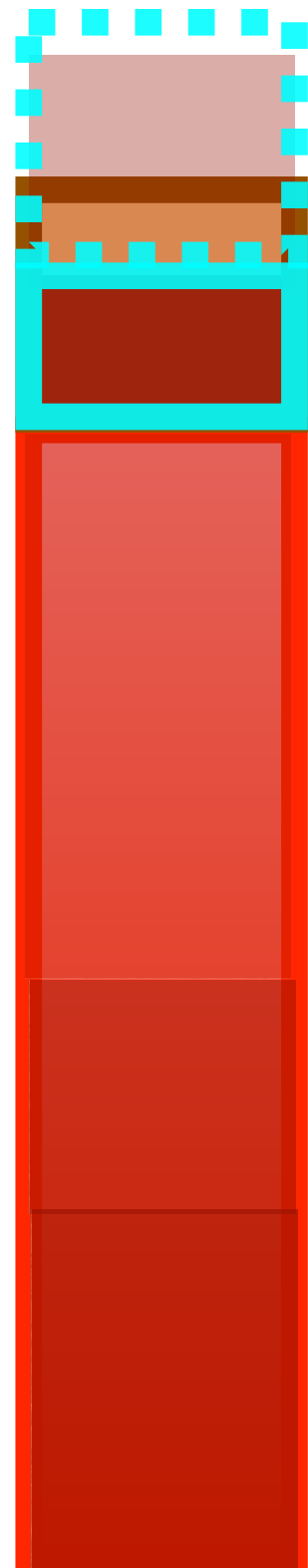
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.



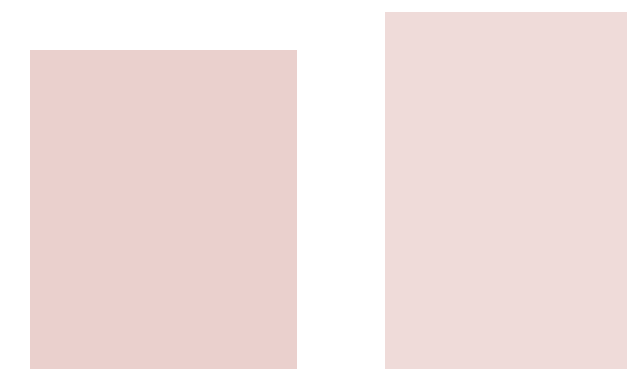
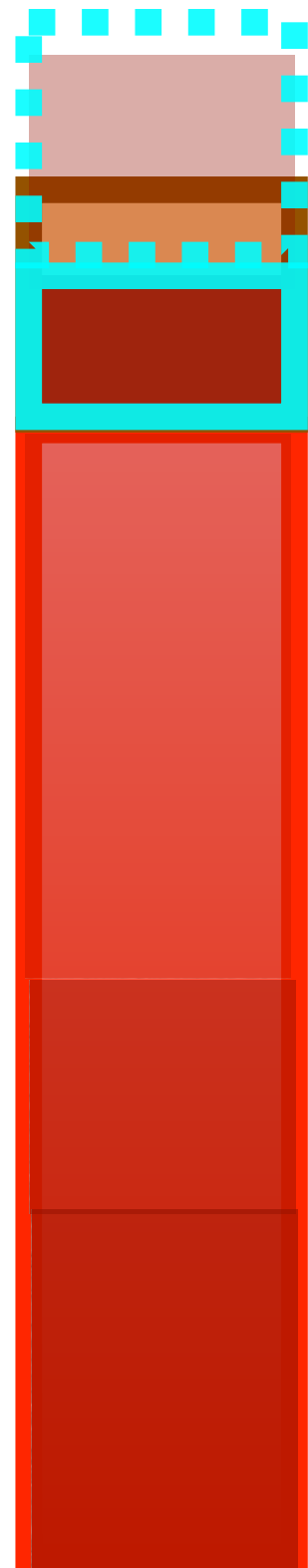
# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.



# Ideen:

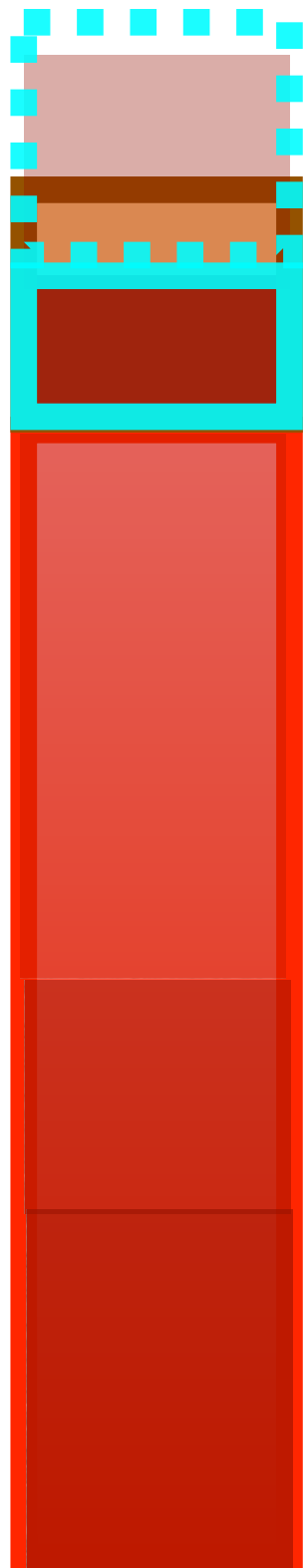
- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.



# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die  $k$  wertvollsten Objekte?

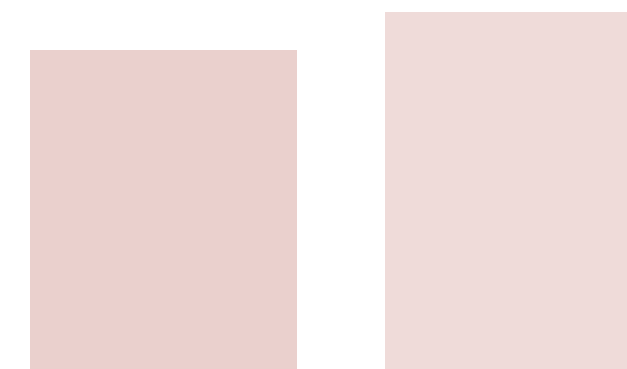
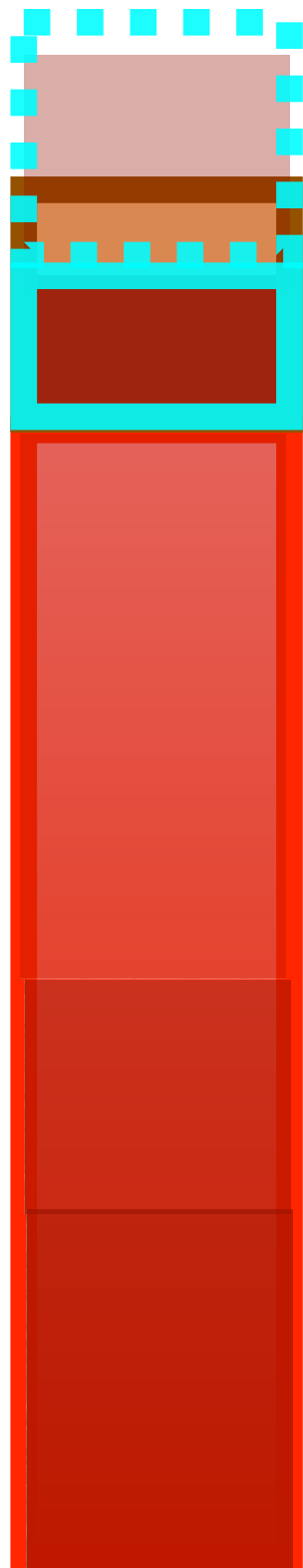


# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die  $k$  wertvollsten Objekte?

- Enumeriere alle Teilmengen der Größe  $k$ !

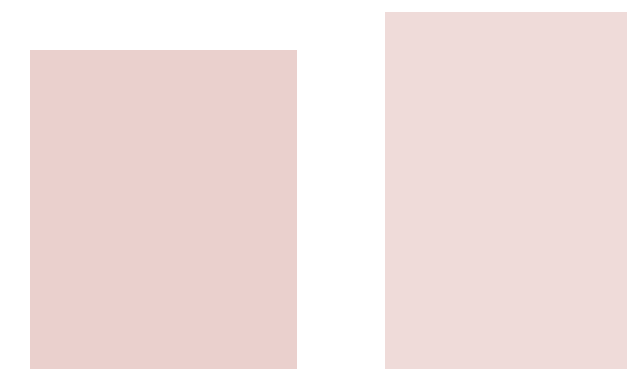
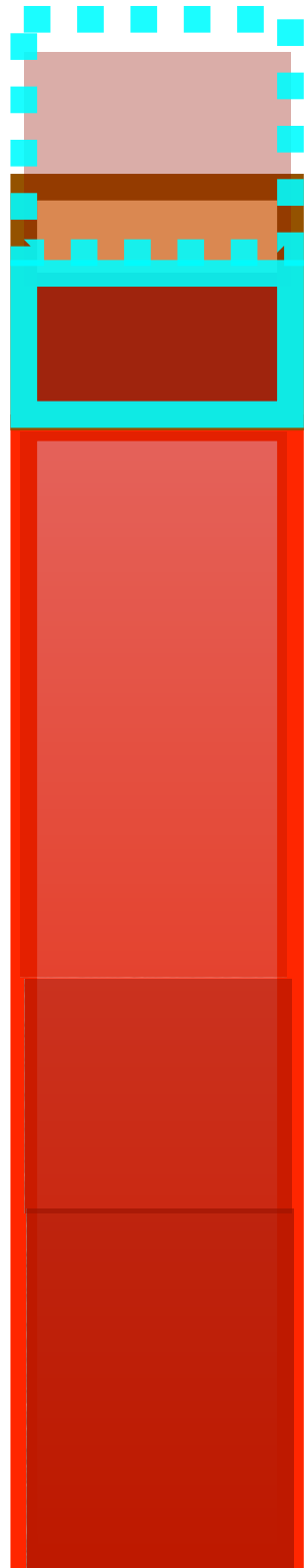


# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die  $k$  wertvollsten Objekte?

- Enumeriere alle Teilmengen der Größe  $k$ !

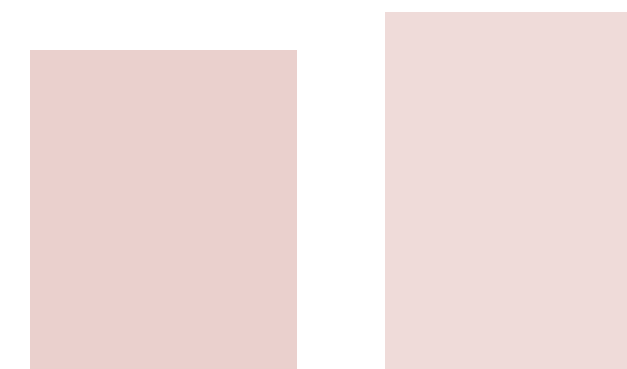
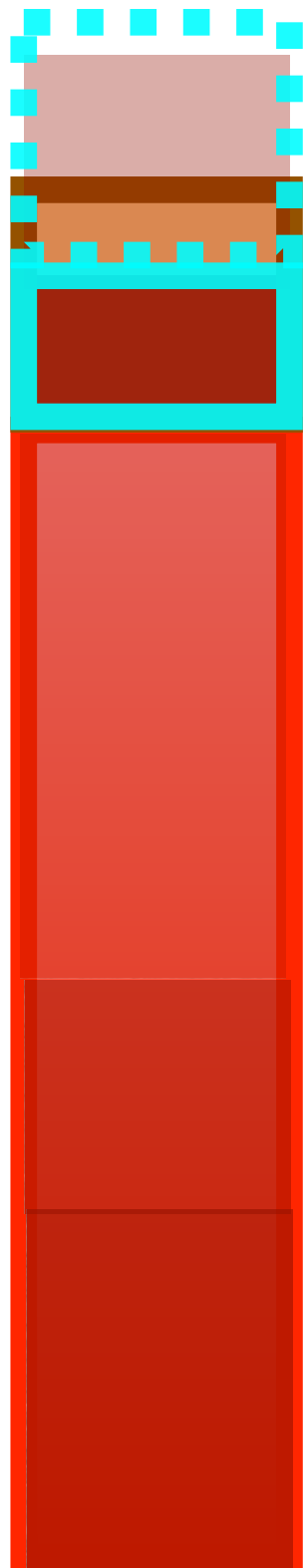


# Ideen:

- Verwende die  $k$  wertvollsten Objekte aus einer Optimallösung.
- Wende Greedy auf den Rest an.
- Der Fehler ist durch *ein* Objekt beschränkt.

Wie findet man die  $k$  wertvollsten Objekte?

- Enumeriere alle Teilmengen der Größe  $k$ !  $\rightarrow O(n^k)$  Teilmengen



# Bessere Approximation von Knapsack



# Bessere Approximation von Knapsack



# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

# Bessere Approximation von Knapsack

---

## Algorithmus 4.6 GREEDY<sub>k</sub>

---

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**



# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:      $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

5:         Update  $S$ ;



# Bessere Approximation von Knapsack

## Algorithmus 4.6 GREEDY<sub>k</sub>

**Eingabe:**  $z_1, \dots, z_n, Z, p_1, \dots, p_n$  [Parameter  $k$  fixiert]

**Ausgabe:**  $S \subseteq \{1, \dots, n\}$  mit  $\sum_{i \in S} z_i \leq Z$  und Wert  $G_k := \sum_{i \in S} p_i$

1:  $G_k := 0, S := \emptyset$

2: **for all**  $\bar{S} \subseteq \{1, \dots, n\}$  mit  $|\bar{S}| \leq k$  **do**

3:     **if**  $(\sum_{i \in \bar{S}} z_i \leq Z)$  **then**

4:          $G_k := \max\{G_k, \sum_{i \in \bar{S}} p_i + \text{GREEDY}_0(\{z_i | i \notin \bar{S}\}, Z - \sum_{i \in \bar{S}} z_i, \{p_i | i \notin \bar{S}\})\};$

5:         Update  $S$ ;

6: **return**  $G_k, S$

# Gütegarantie

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in polynomieller Zeit in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in polynomieller Zeit in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Nicht  $2^n$ , sondern  $n^2$

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in polynomieller Zeit in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

(a) Die Laufzeit ist nicht mehr als  $O(n^{k+2})$  - also polynomiell.

Nicht  $2^n$ , sondern  $n^2$

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in polynomieller Zeit in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

(a) Die Laufzeit ist nicht mehr als  $O(n^{k+2})$  - also polynomiell.

Nicht  $2^n$ , sondern  $n^2$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

**Definition 4.5** (Approximationsalgorithmus).

1. Für ein Maximierungsproblem MAX ist ein Algorithmus ALG ein  $c$ -Approximationsalgorithmus für MAX, wenn für jede Instanz  $I$  von MAX
  - a) ALG in *polynomieller Zeit* in der Größe von  $I$  eine zulässige Lösung mit Wert ALG liefert
  - b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

⋮

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

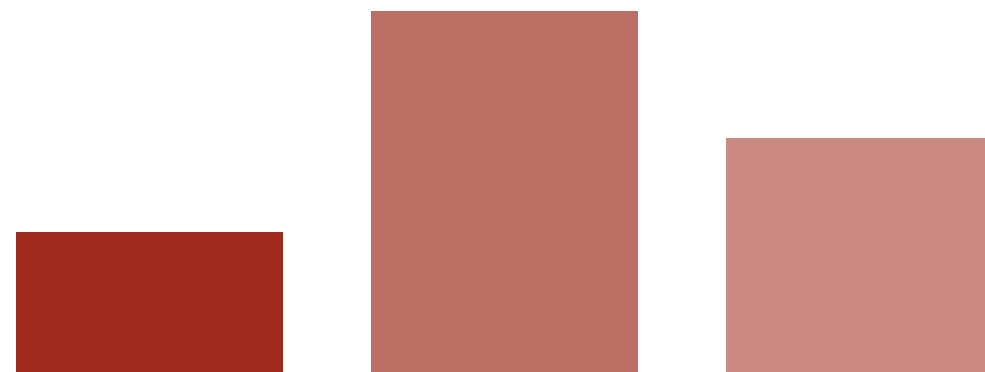


⋮

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )



⋮

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

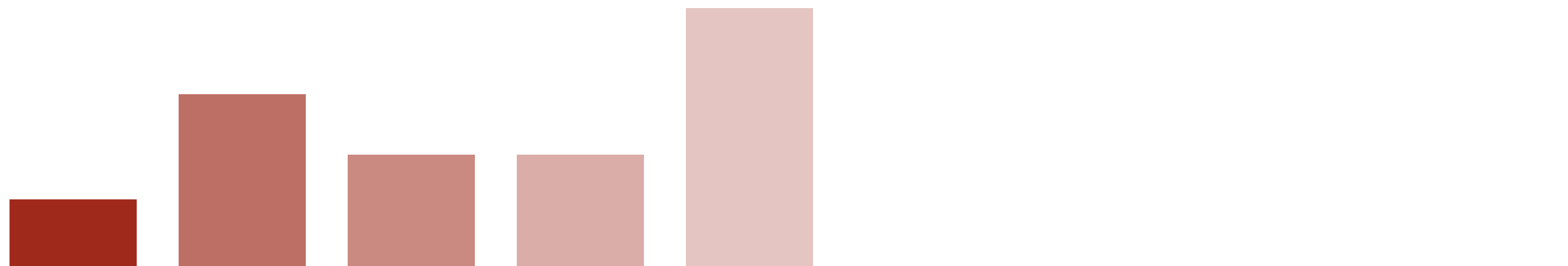
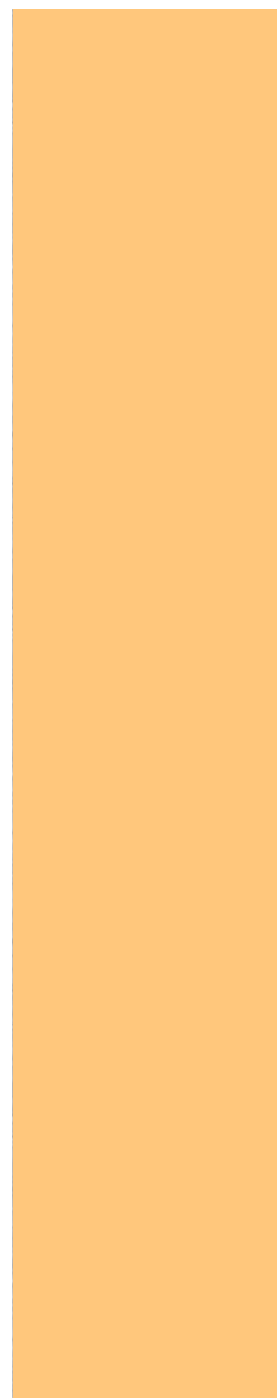


⋮

# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )





# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



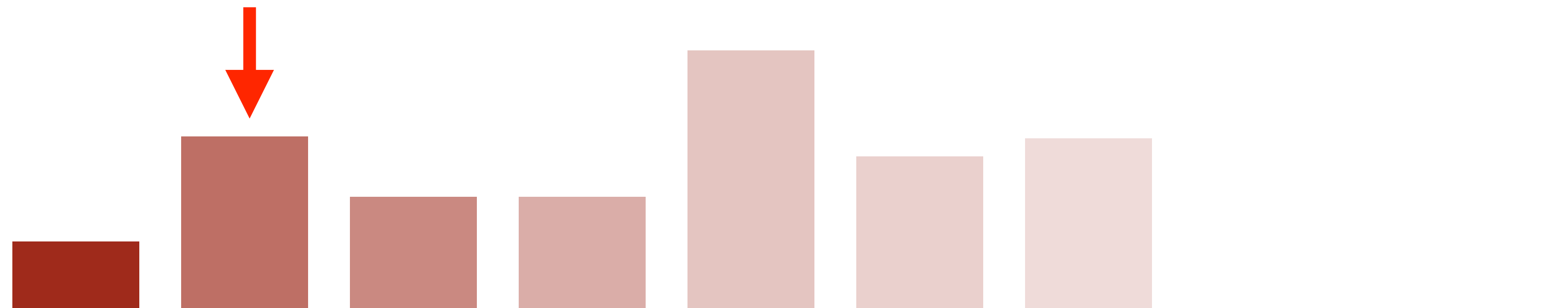
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



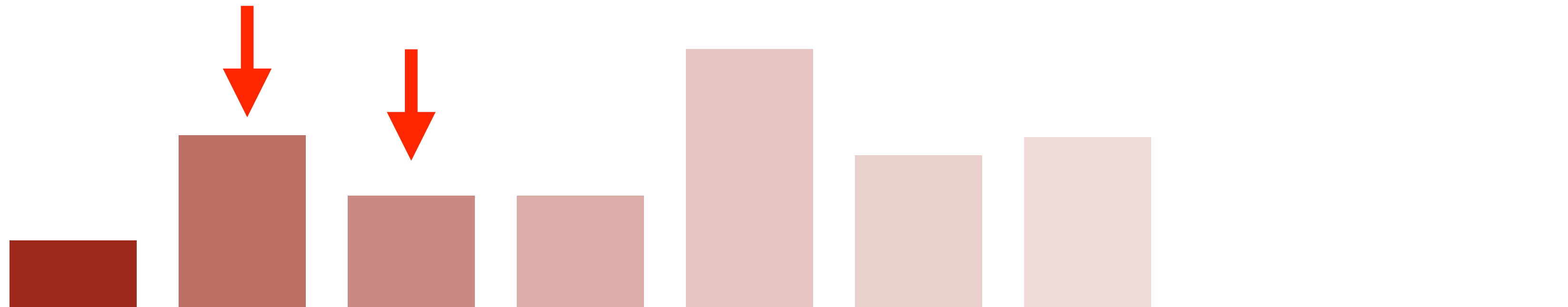
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



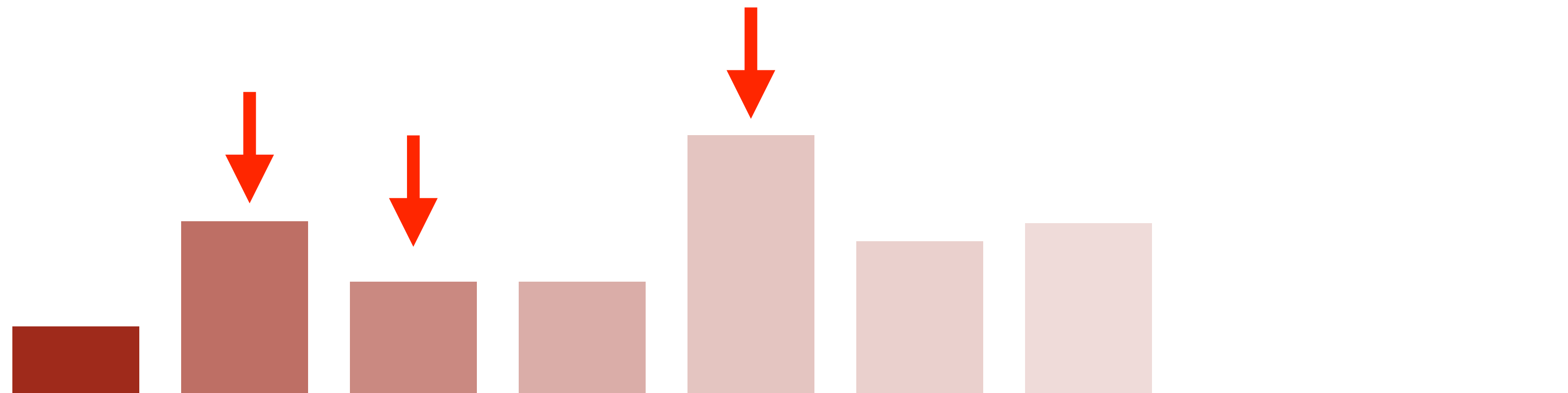
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



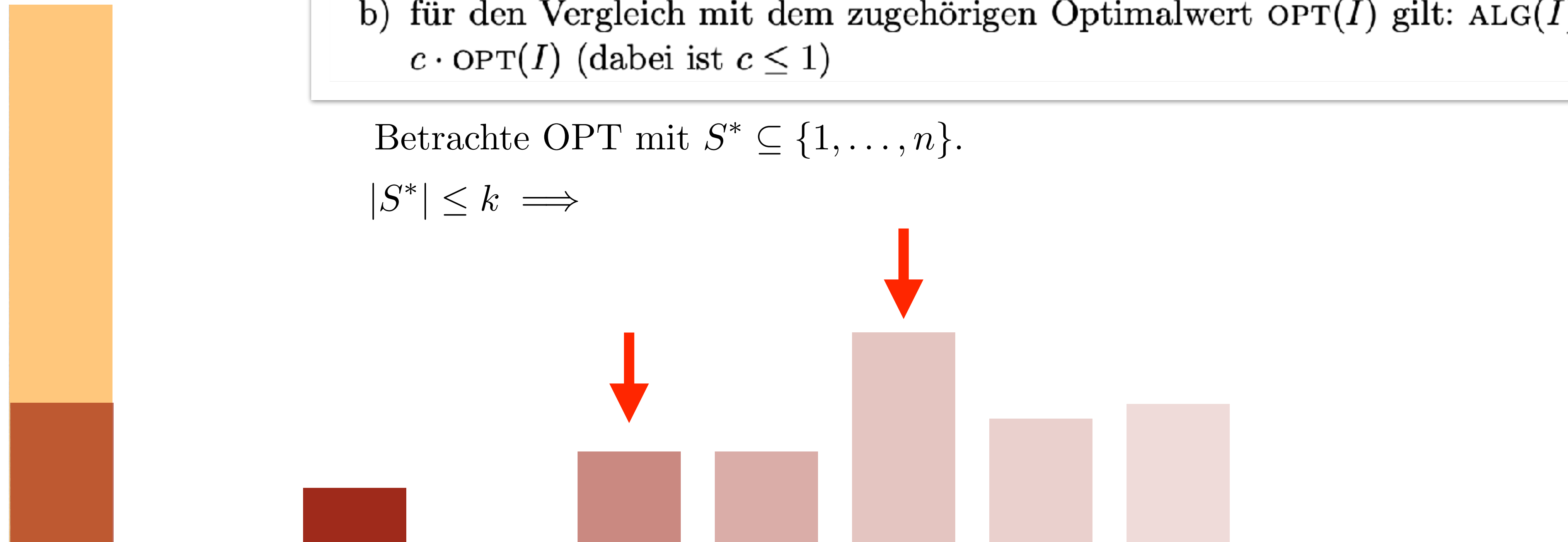
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$





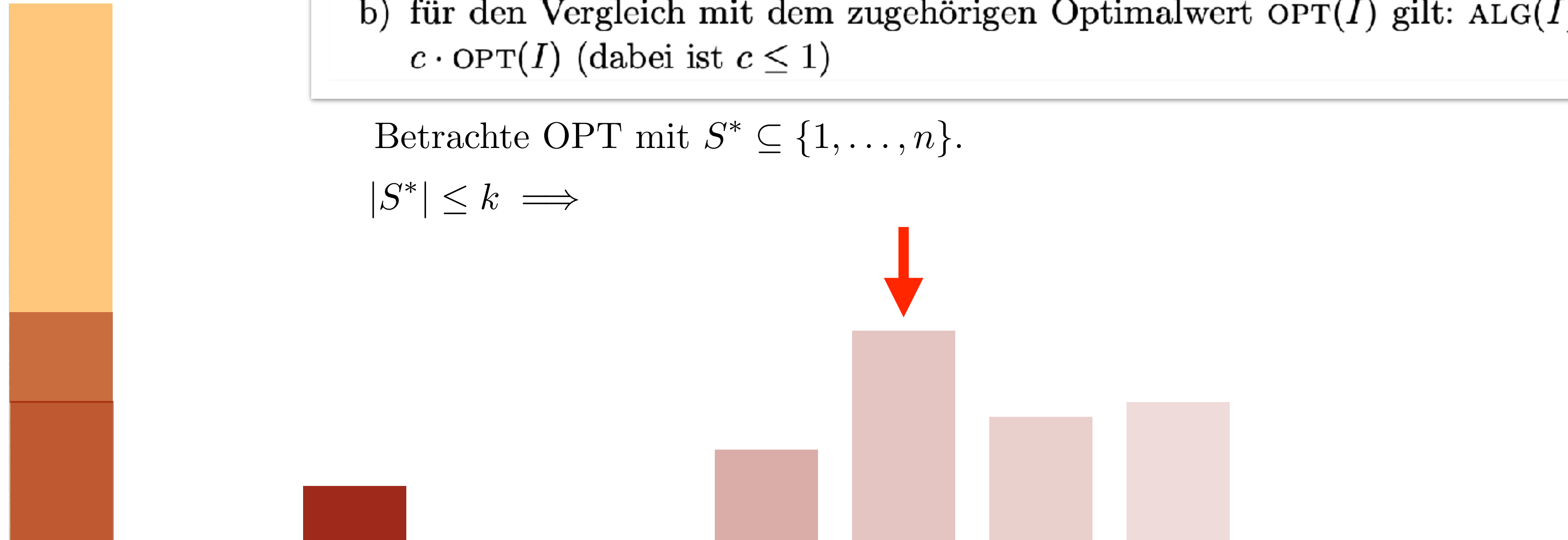
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



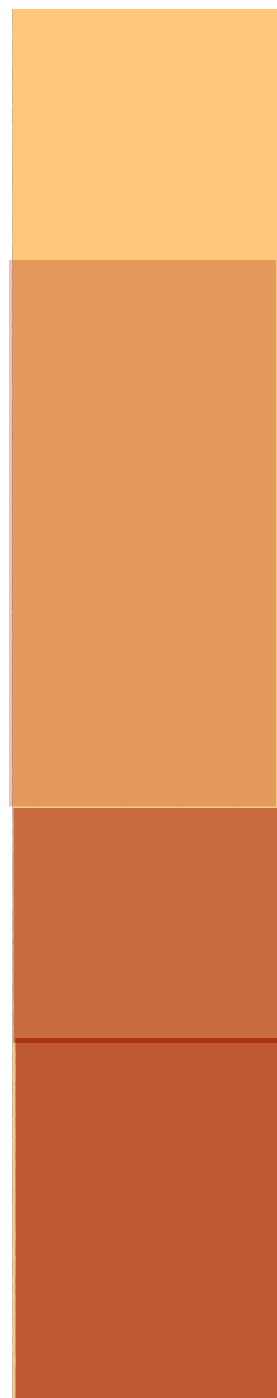
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



⋮

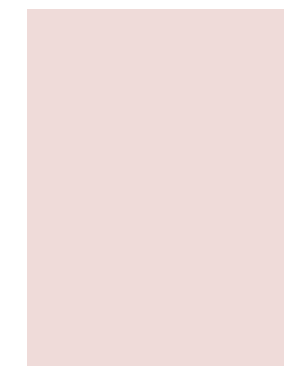
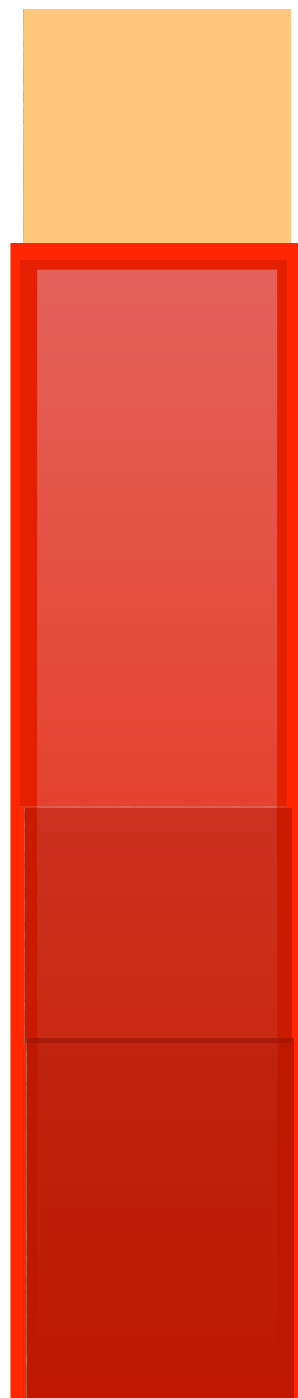
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



⋮

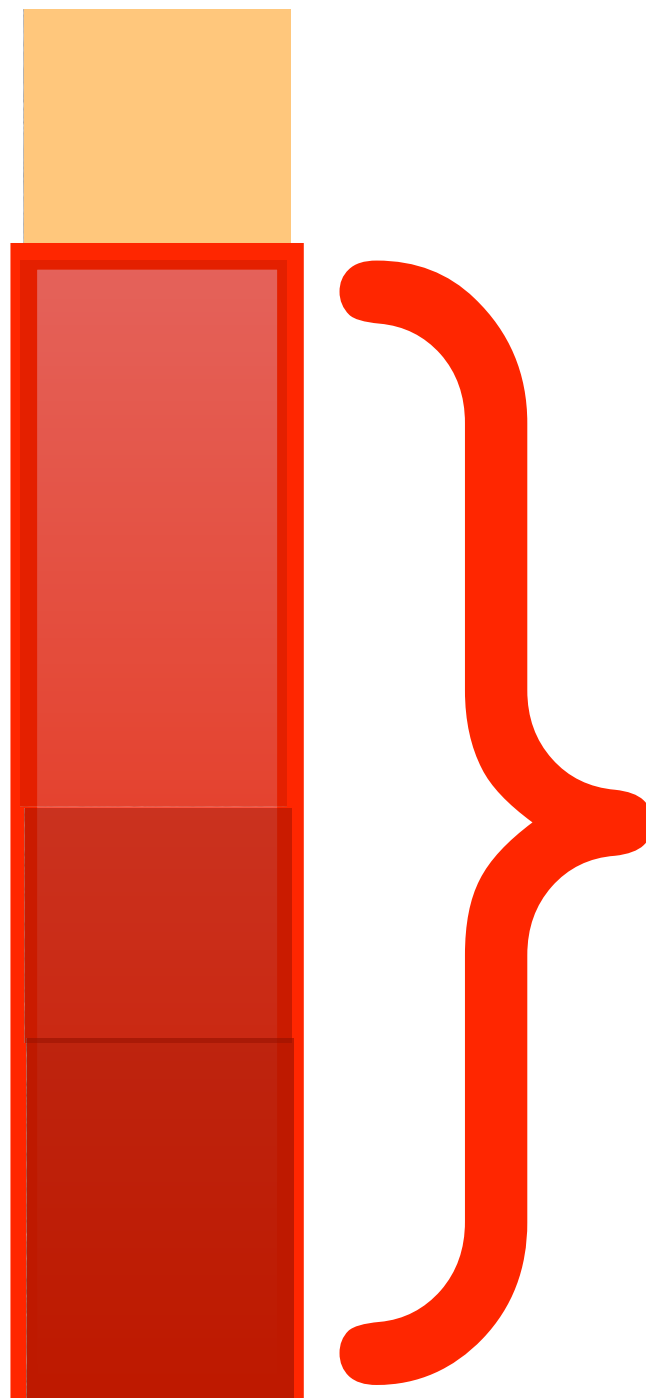
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



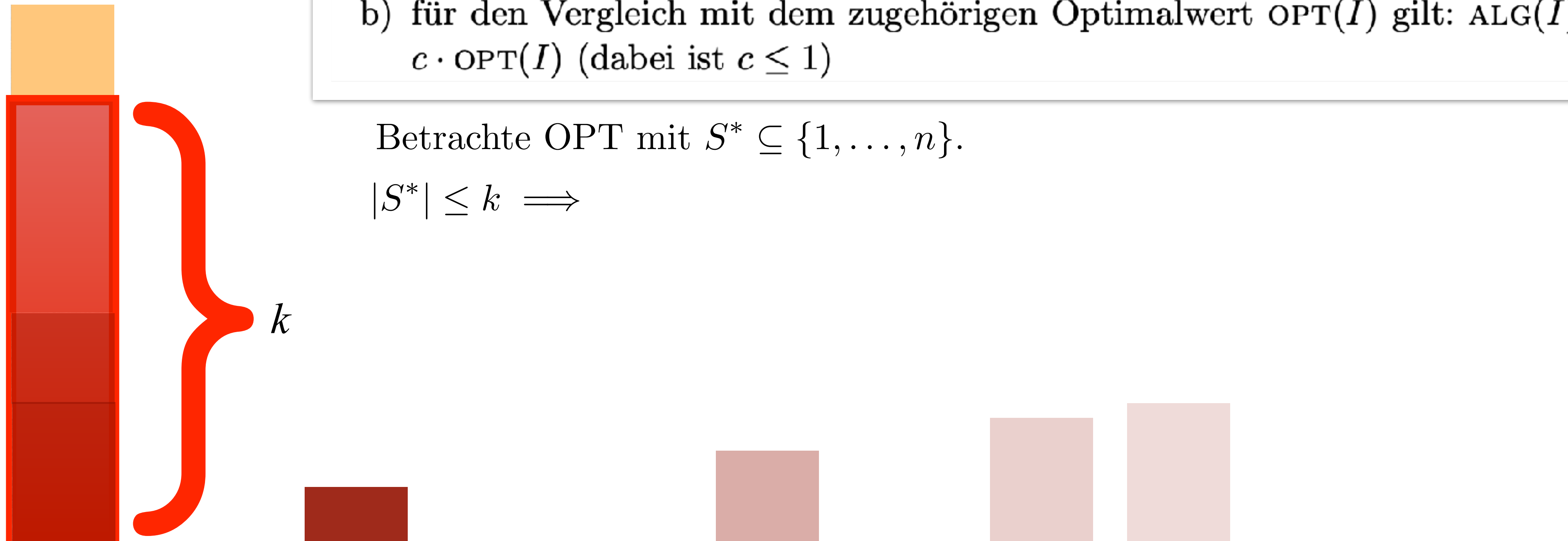
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies$



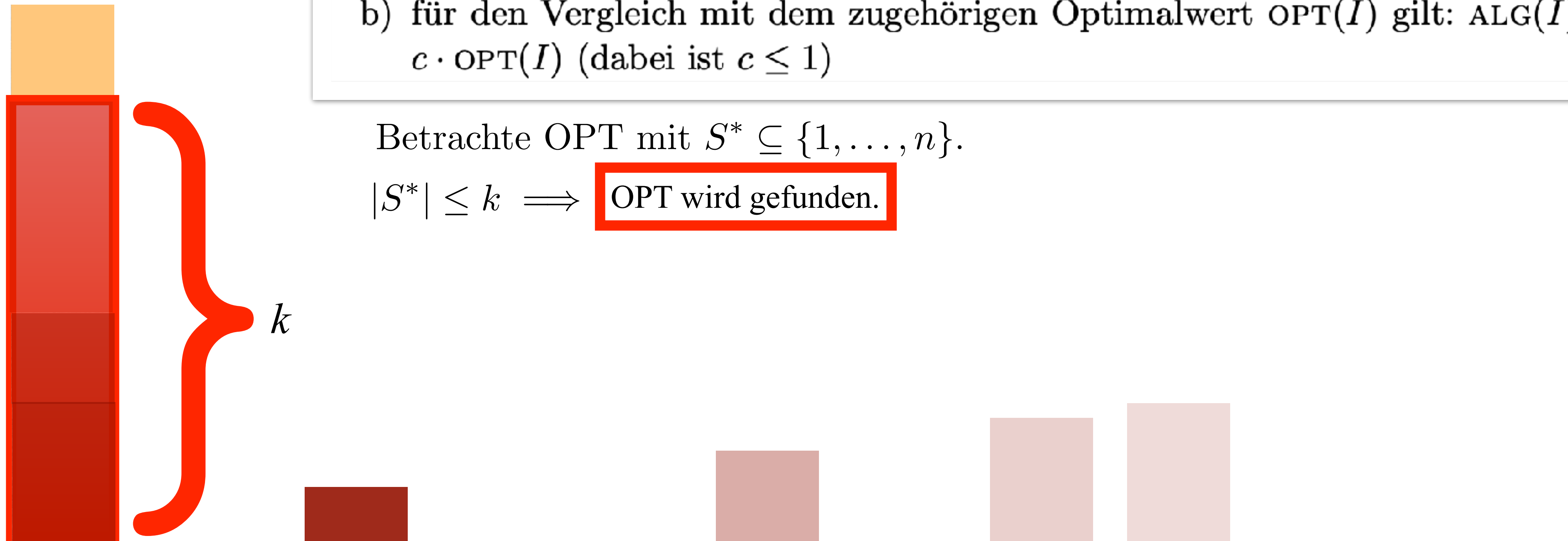
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| \leq k \implies \text{OPT wird gefunden.}$

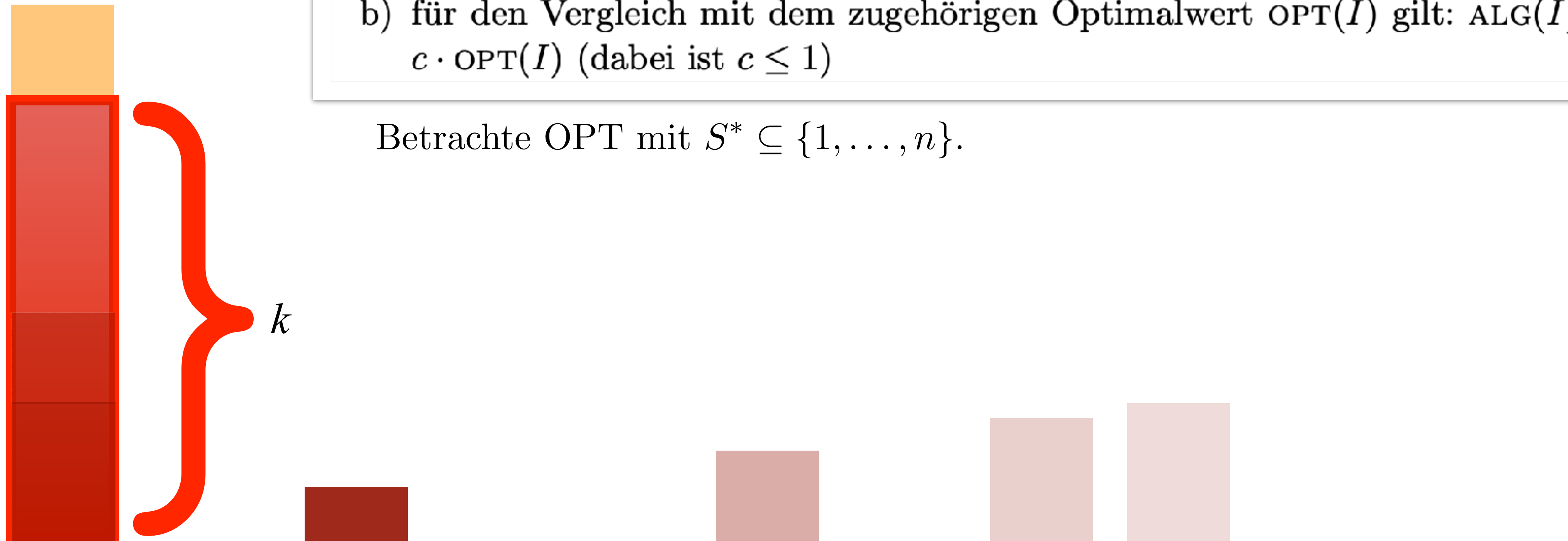


# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .



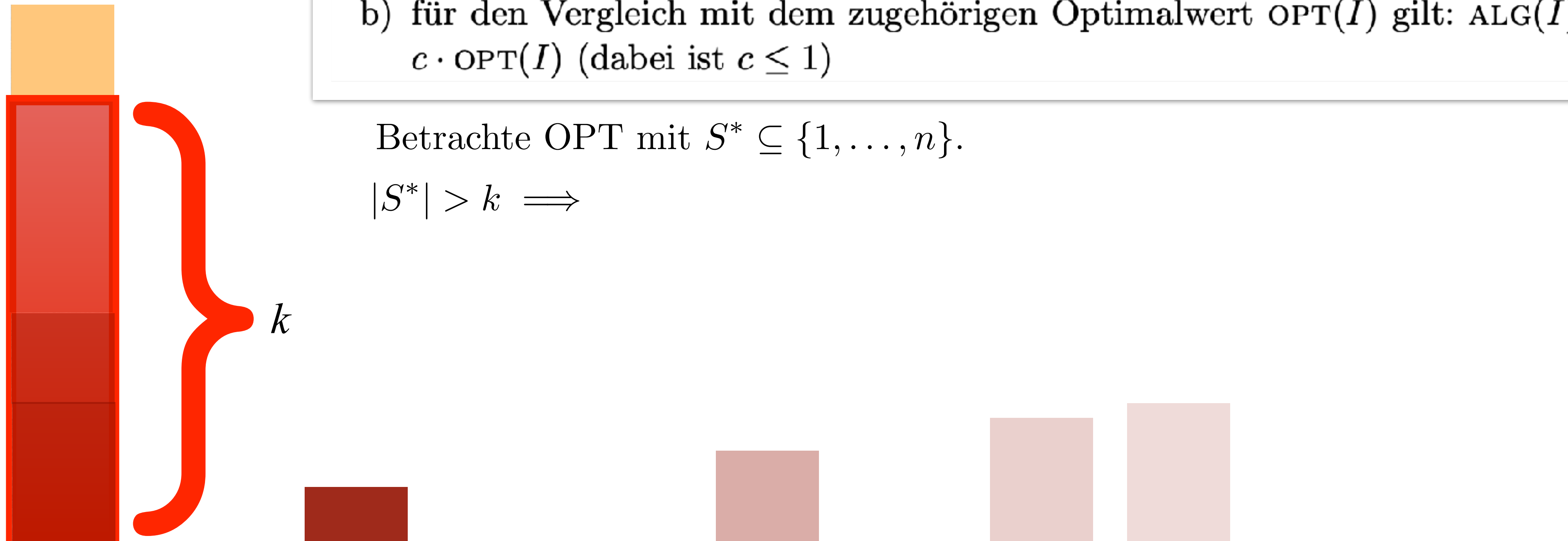
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$





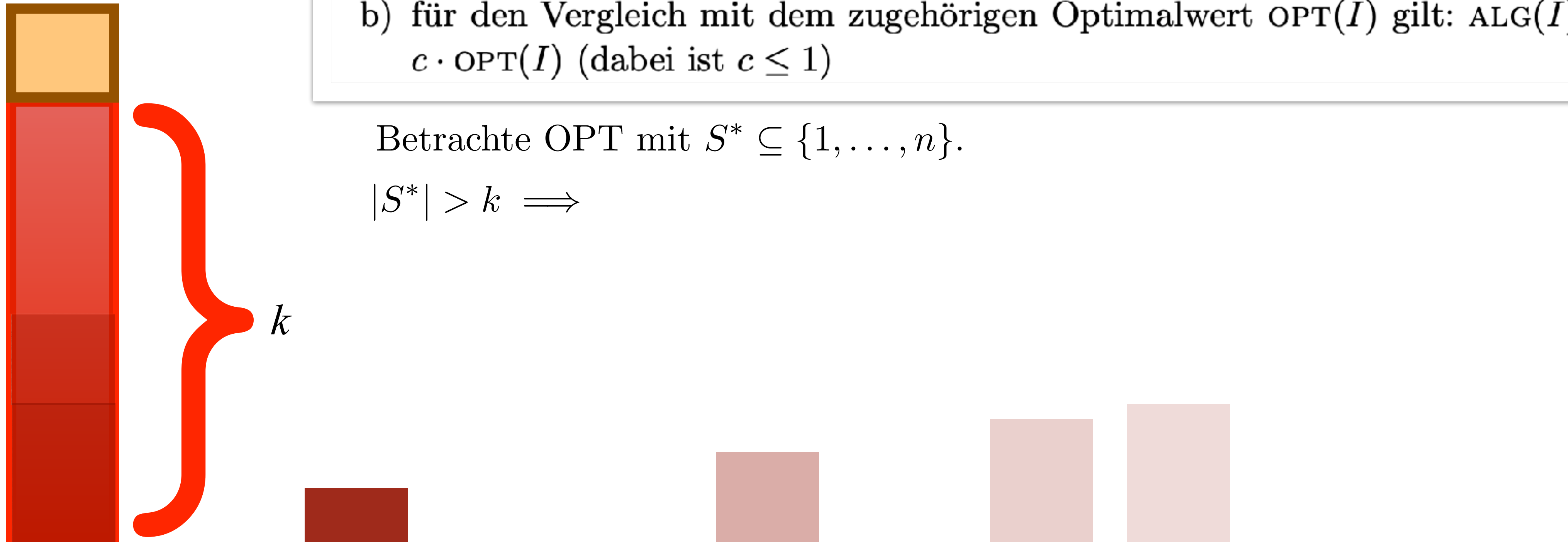
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



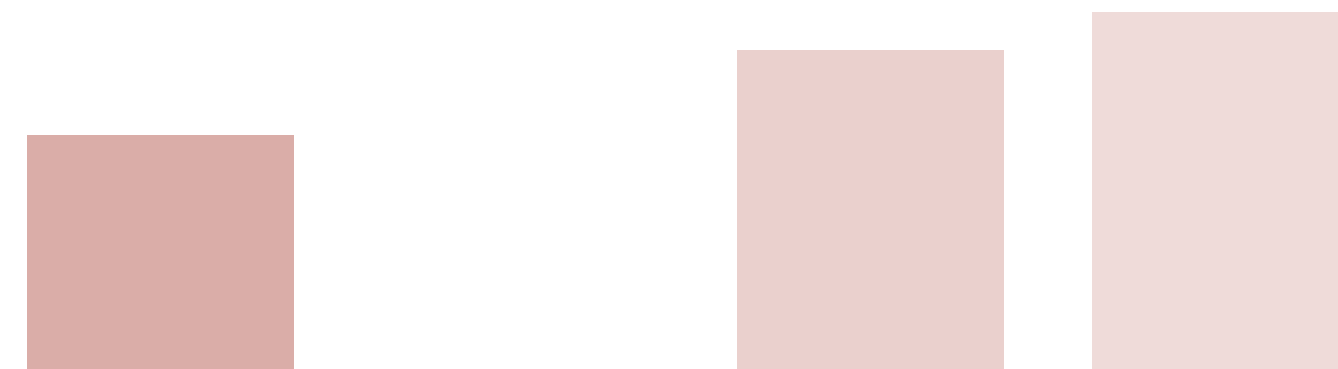
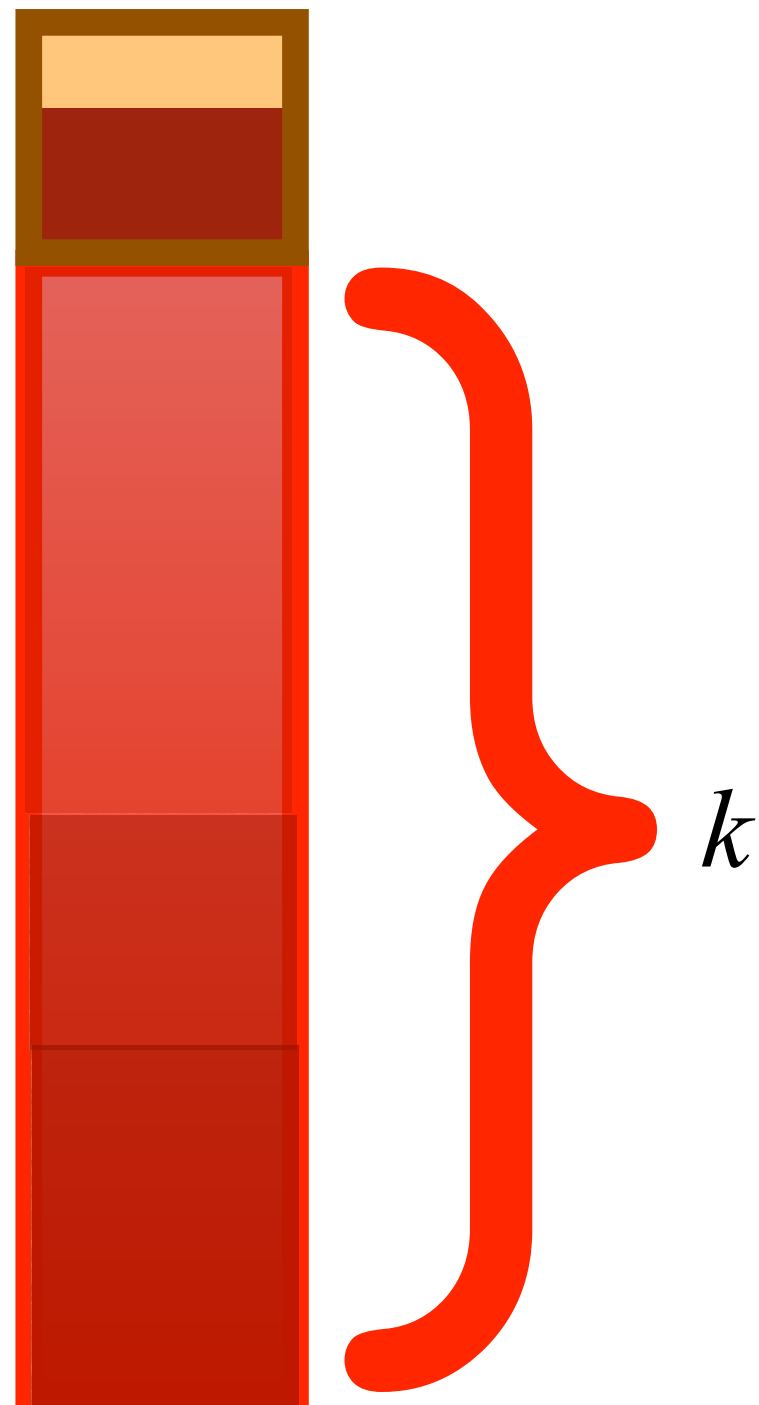
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



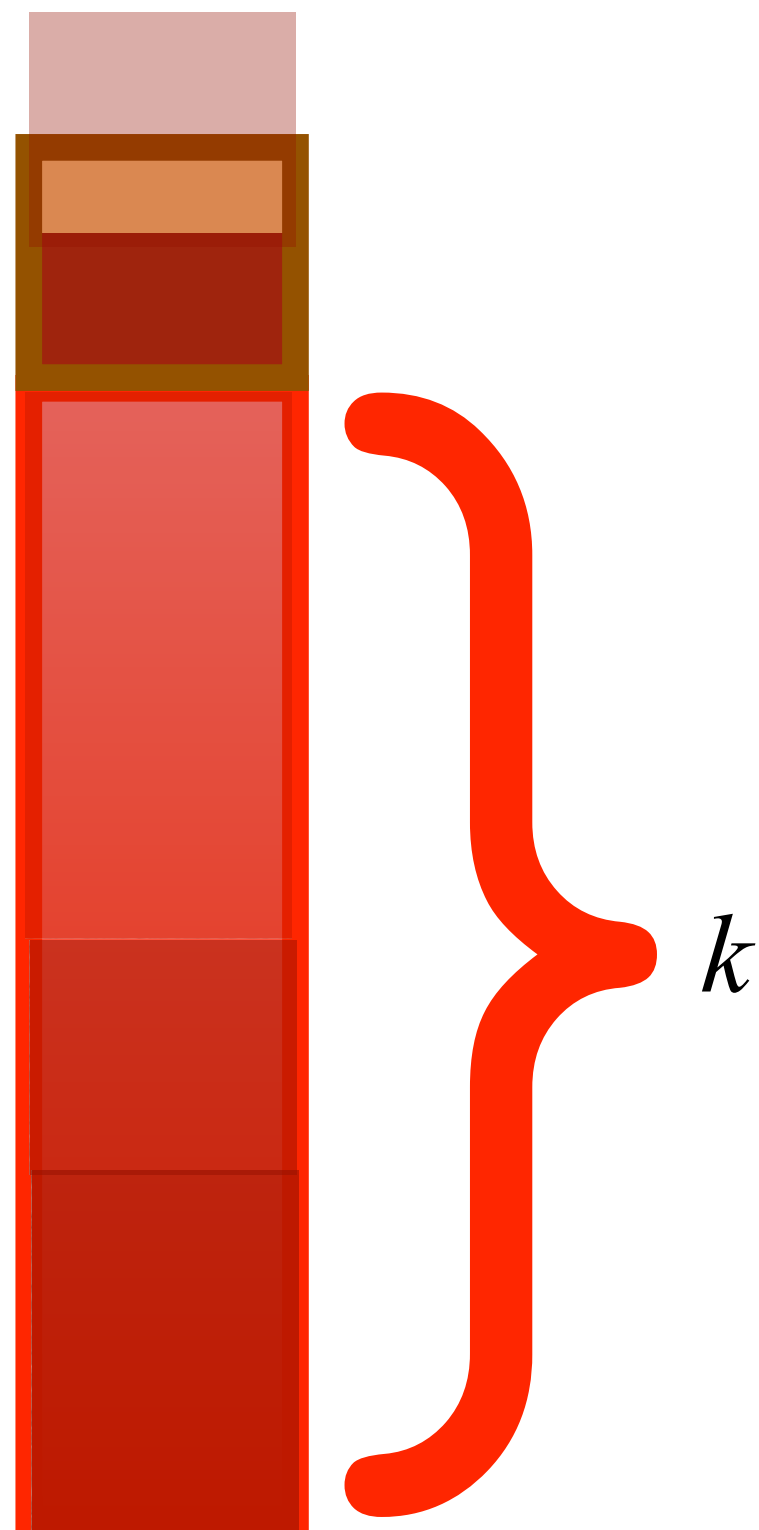
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



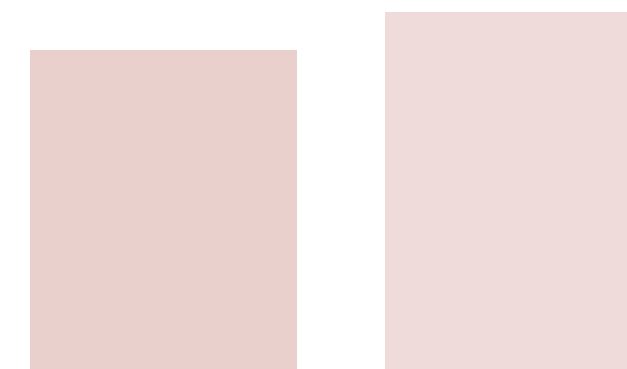
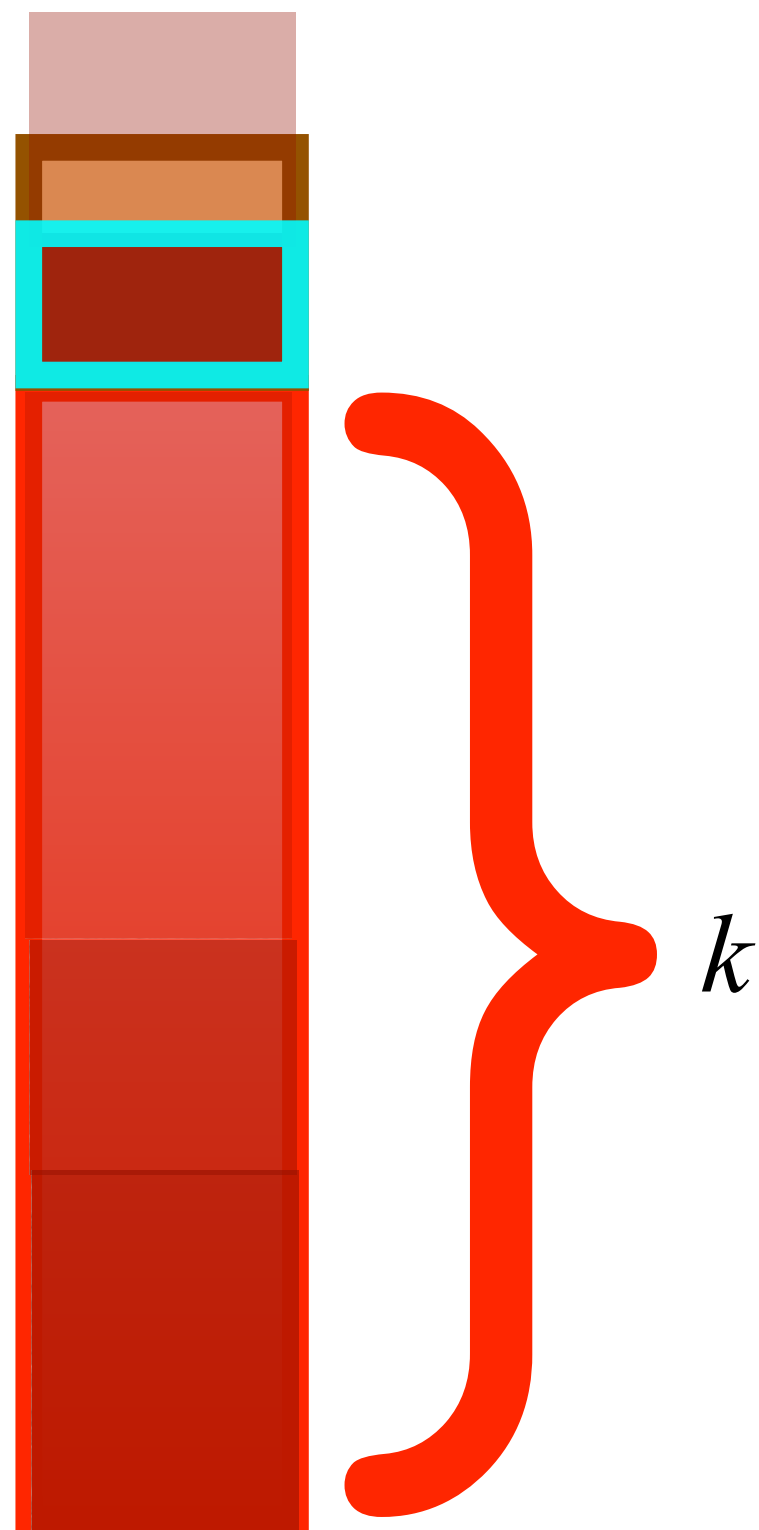
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



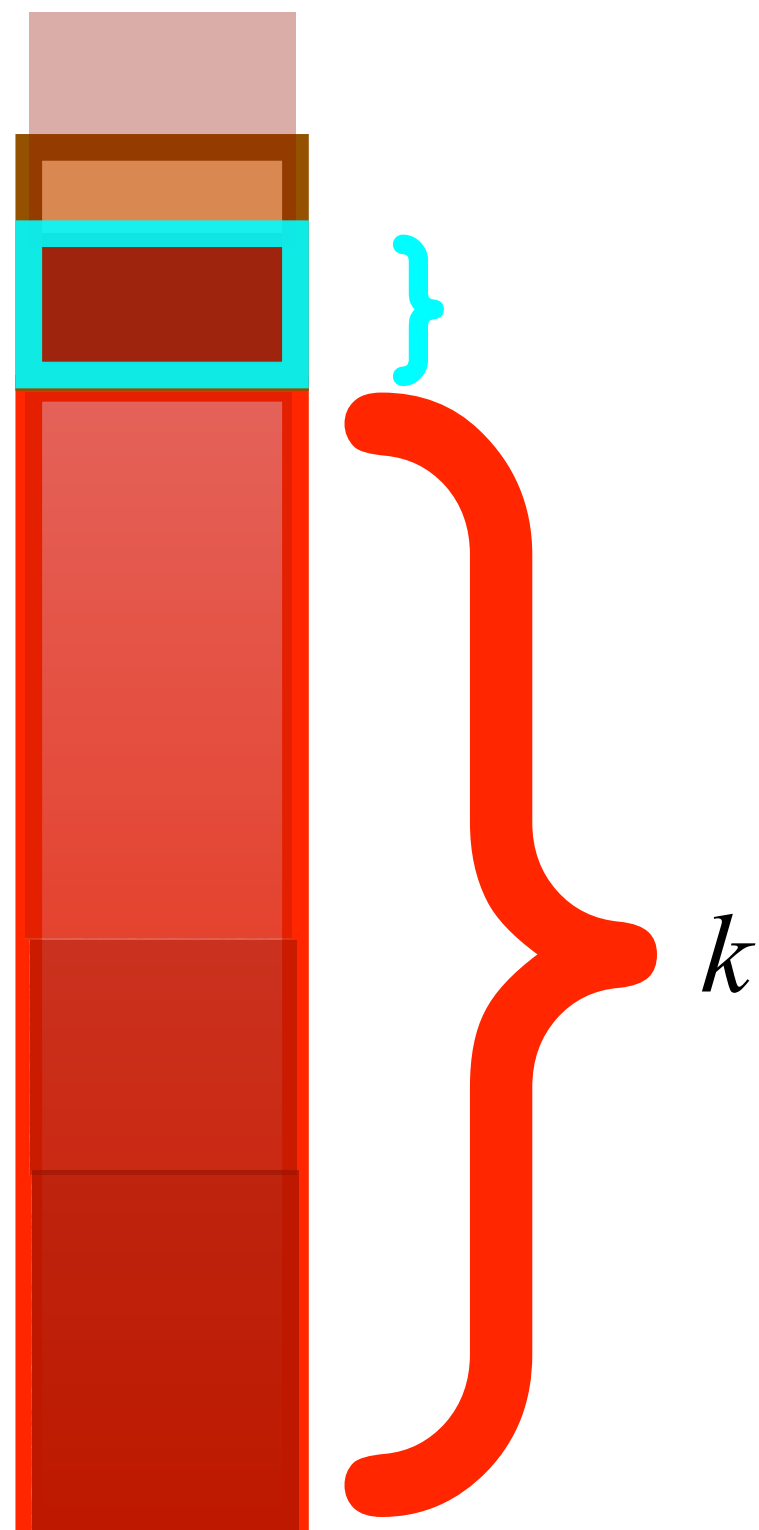
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



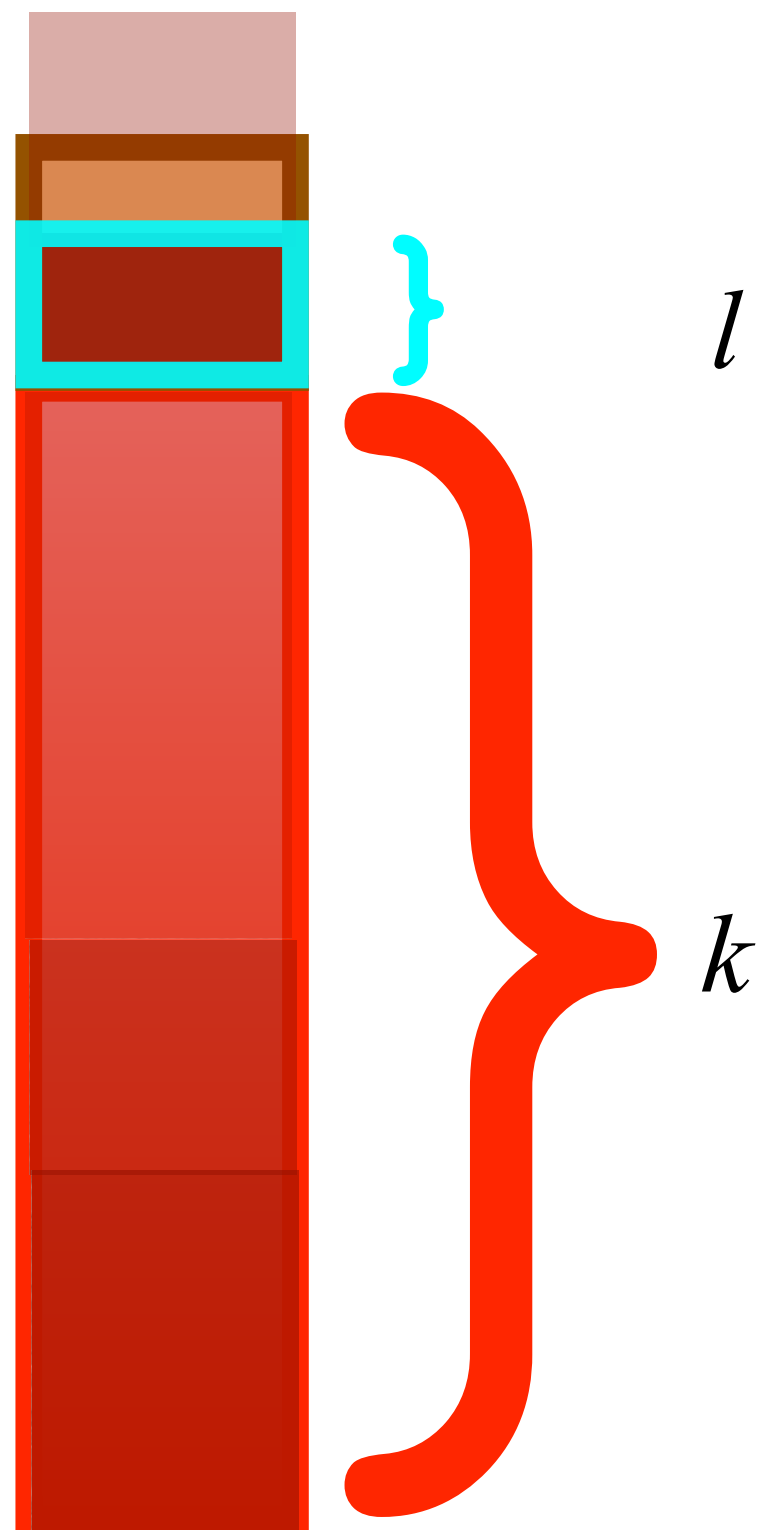
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



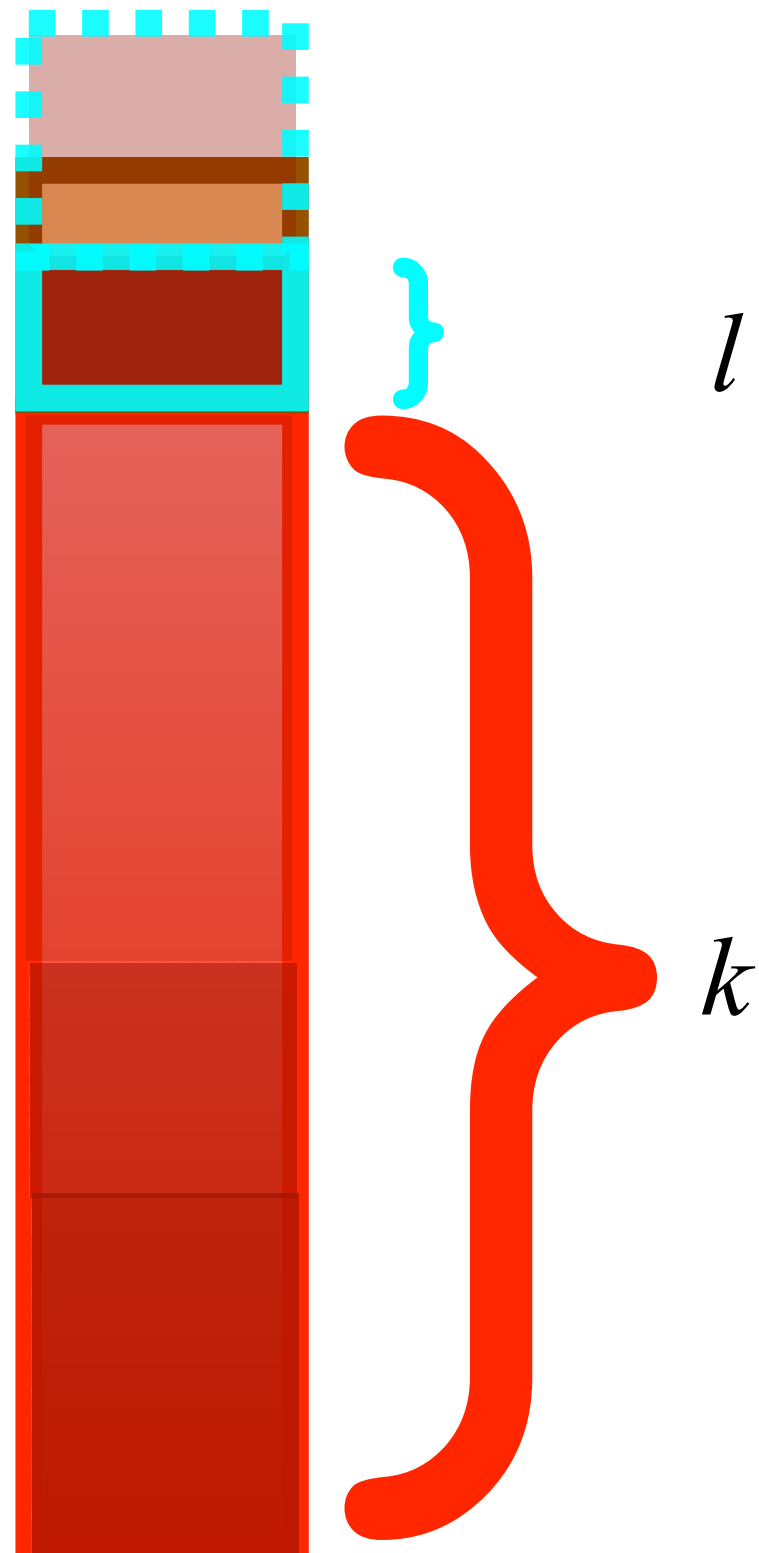
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



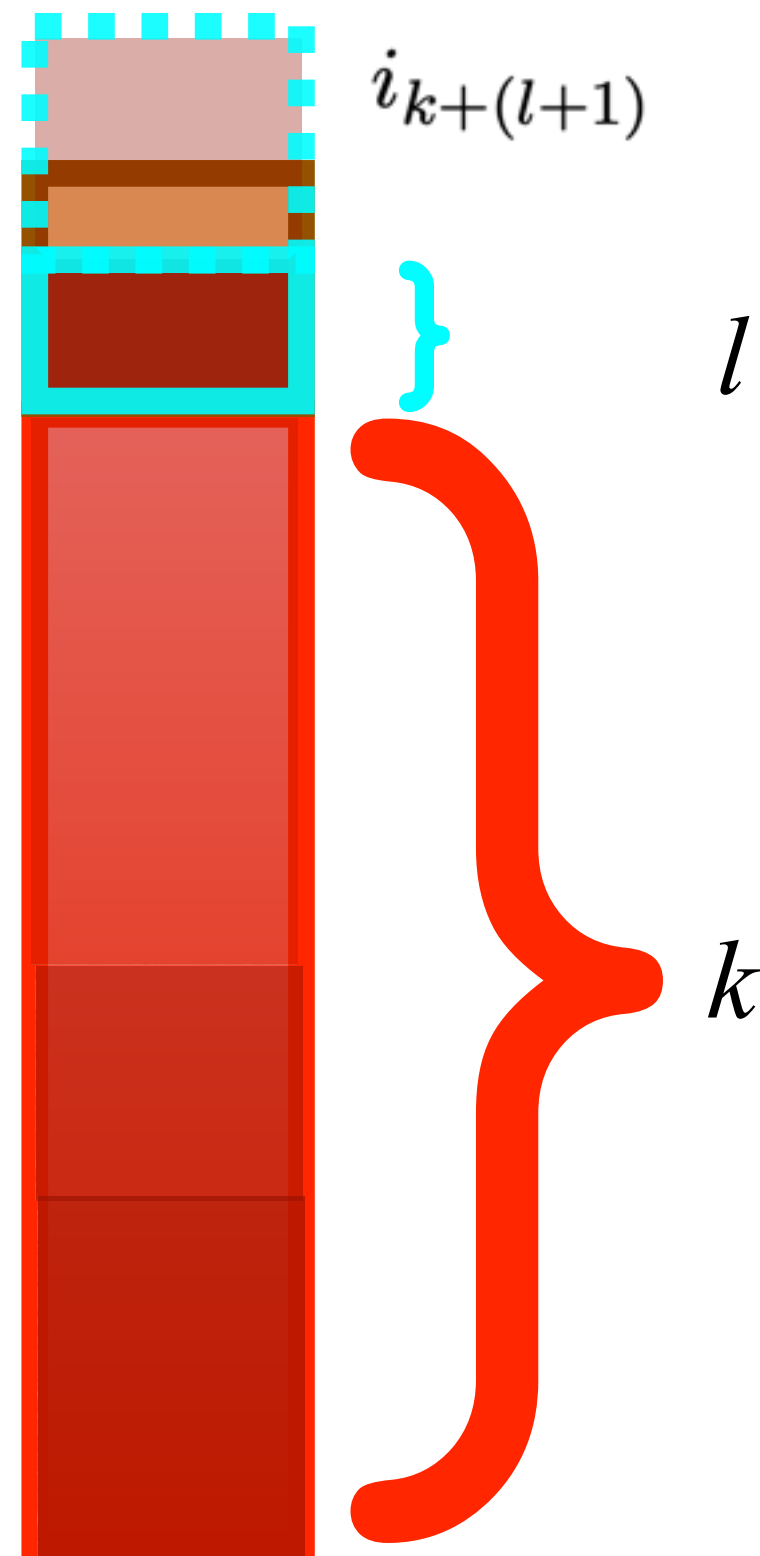
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$





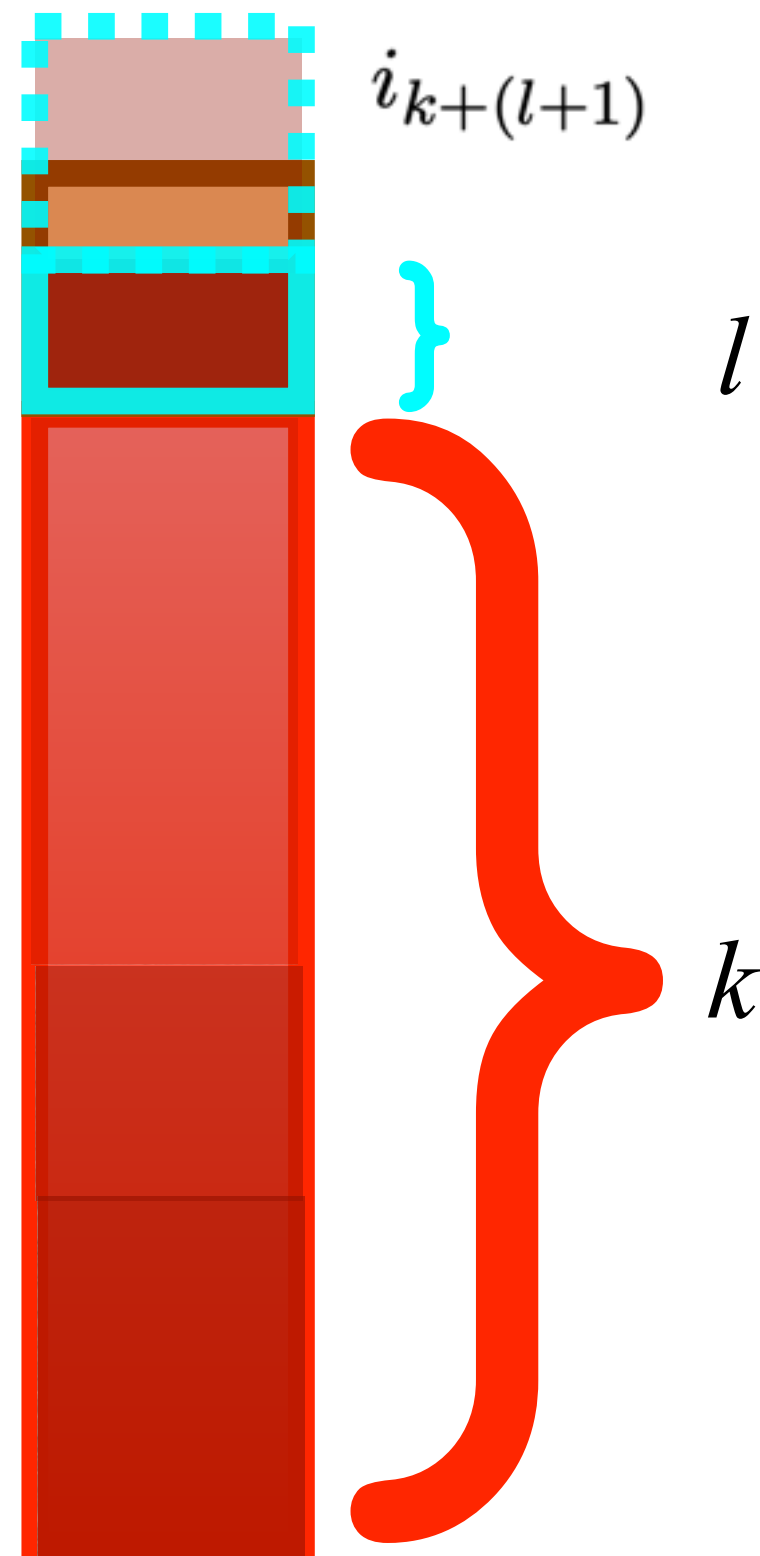
# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$



# Gütegarantie

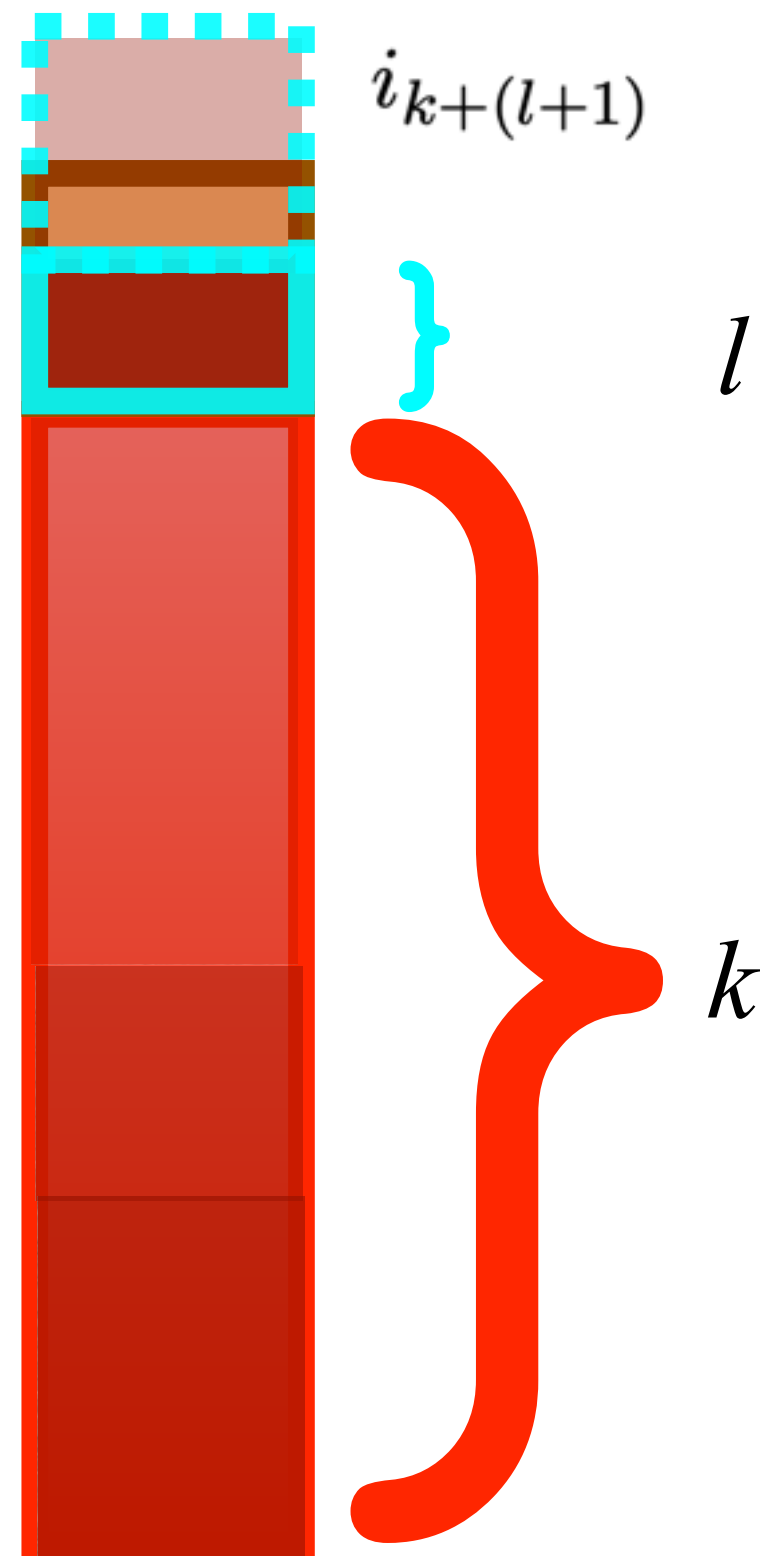
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$$



# Gütegarantie

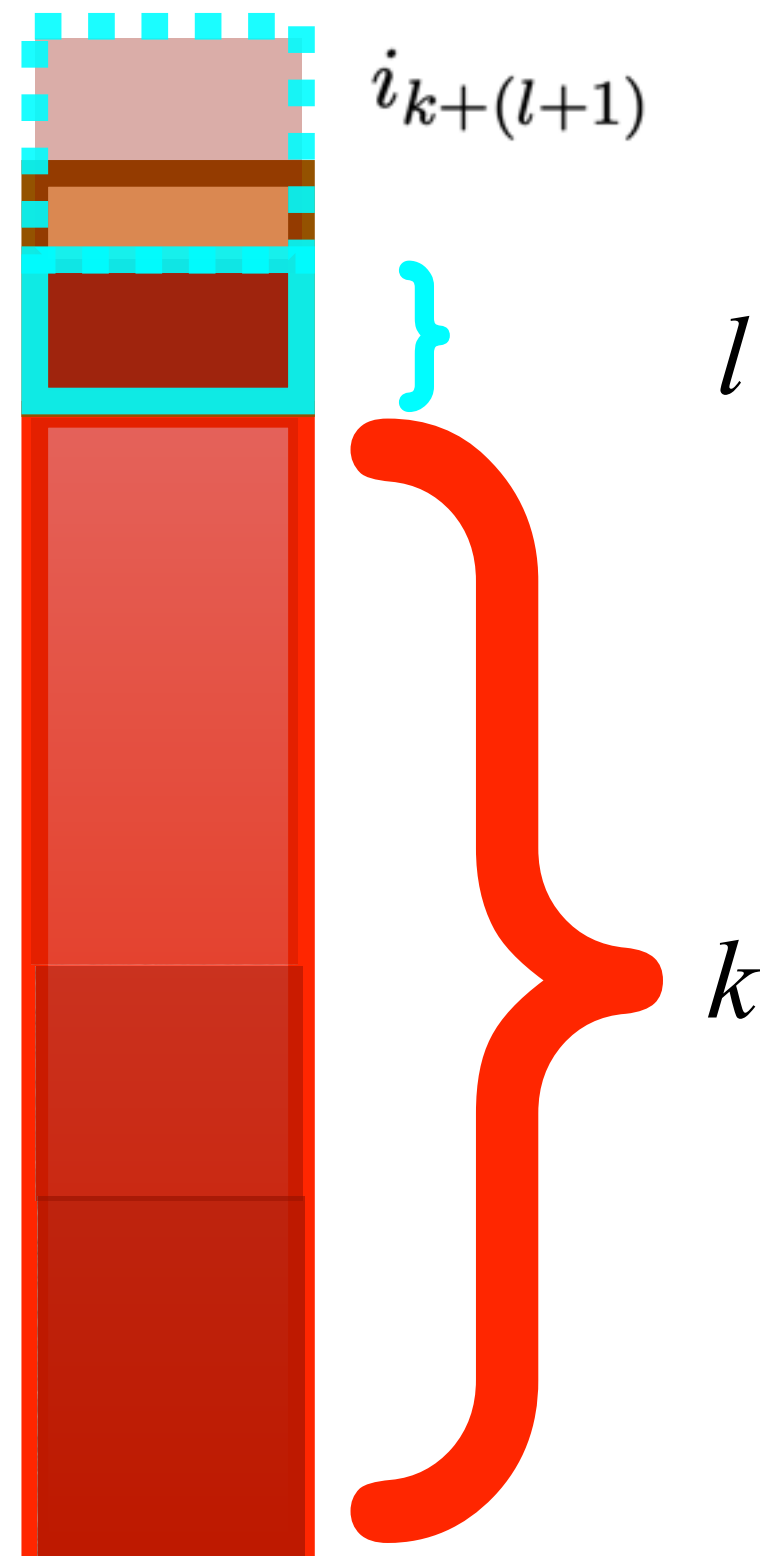
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$



# Gütegarantie

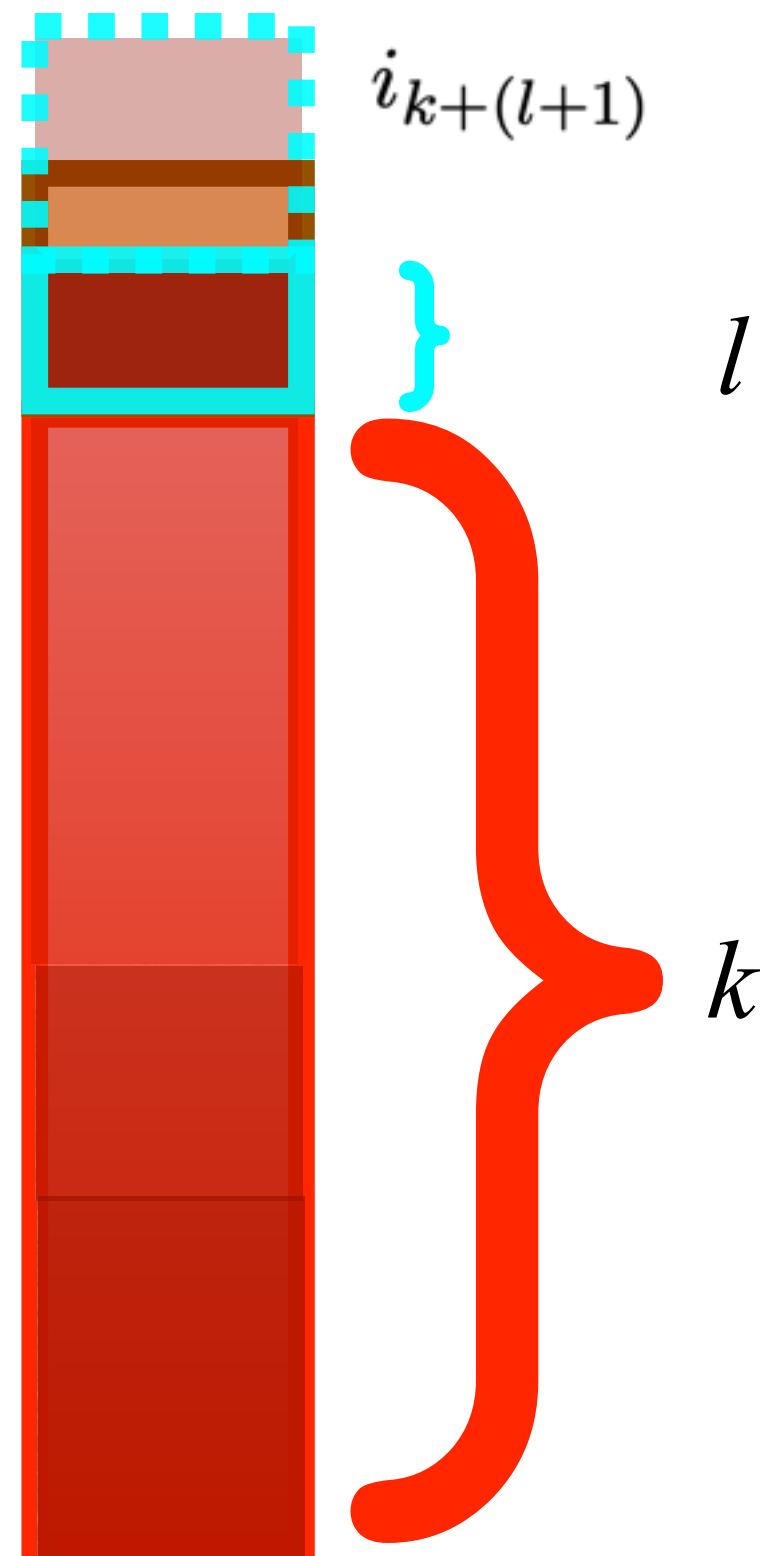
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$$



# Gütegarantie

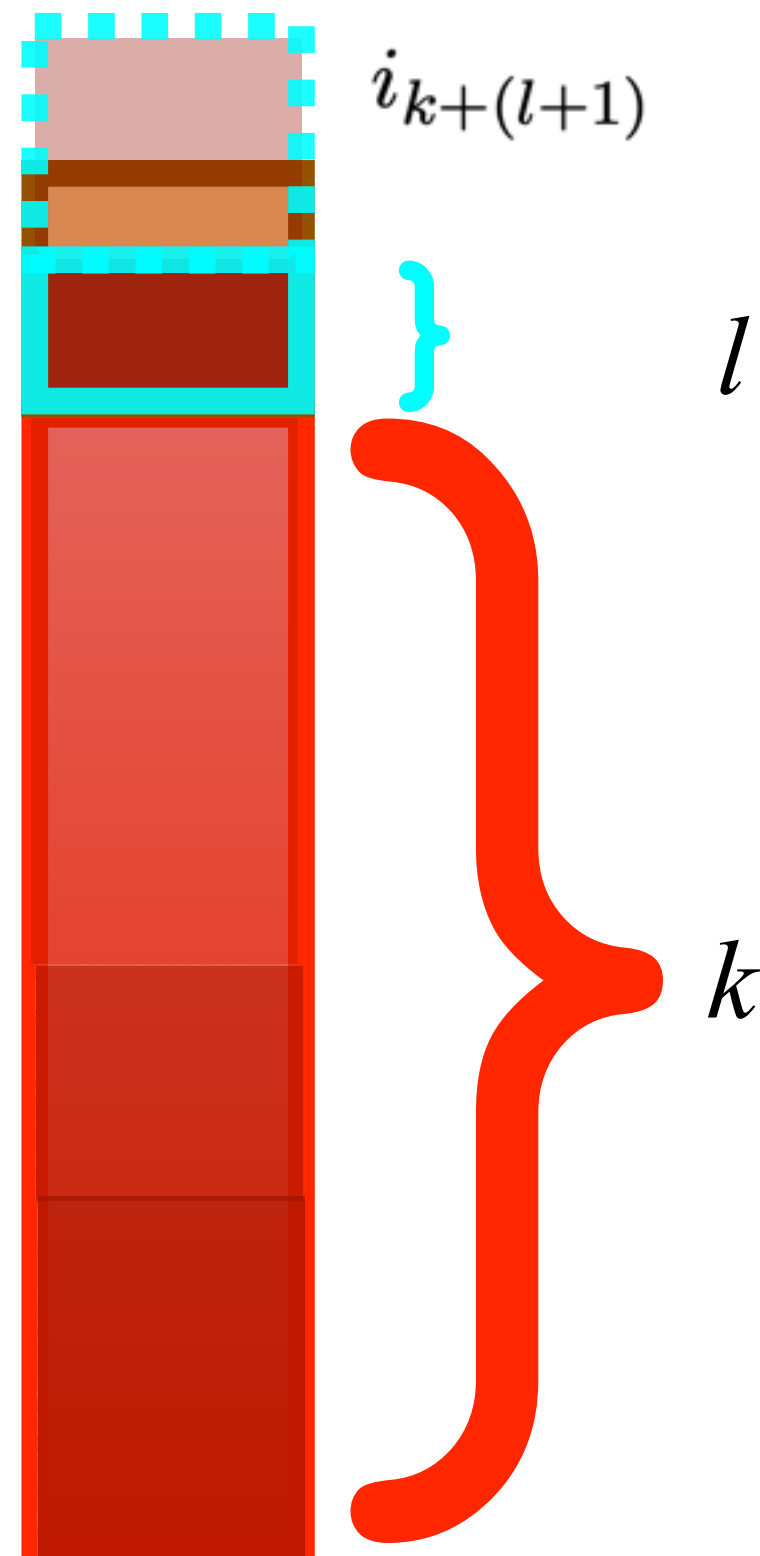
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

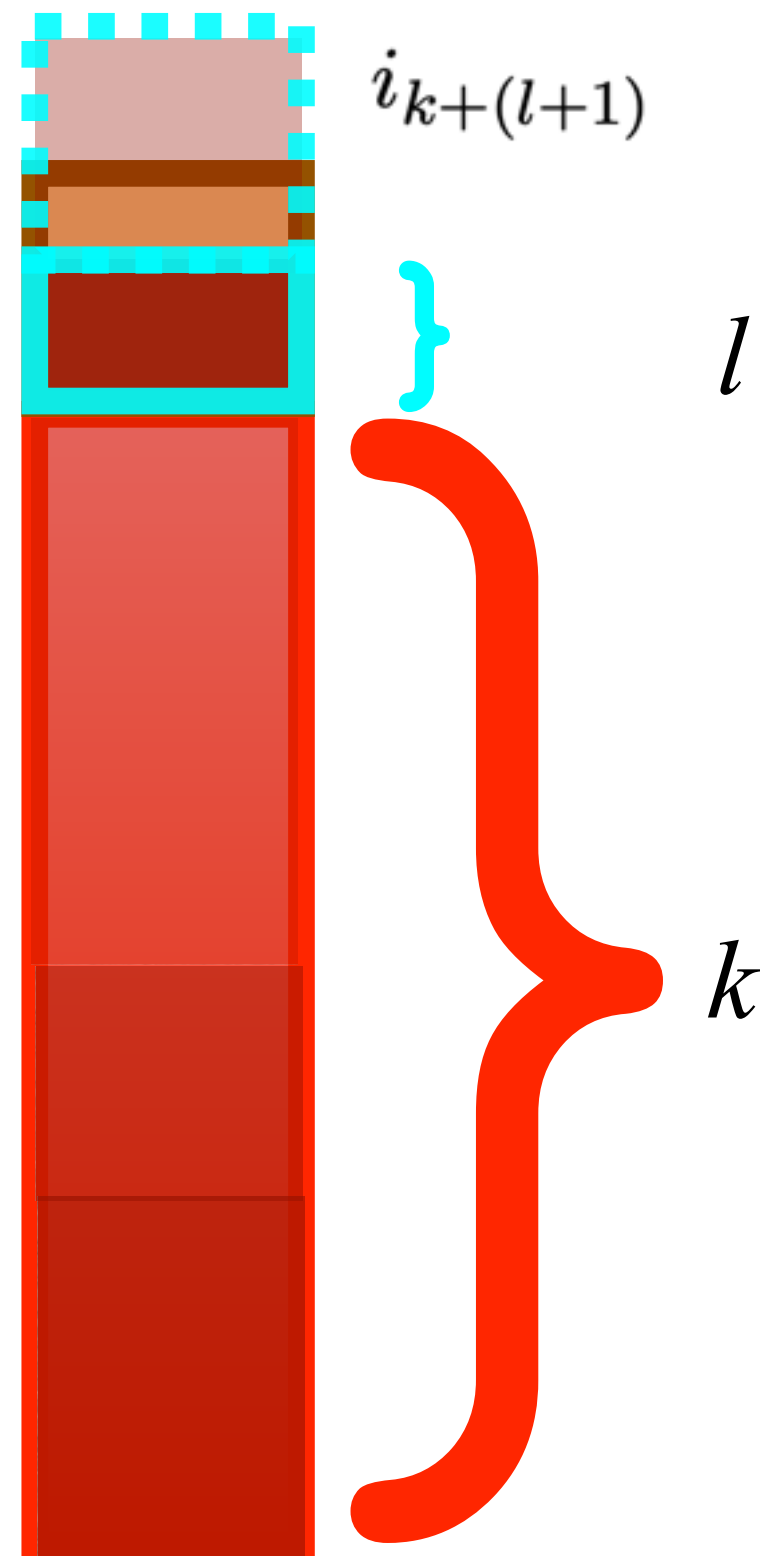
b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$



# Gütegarantie

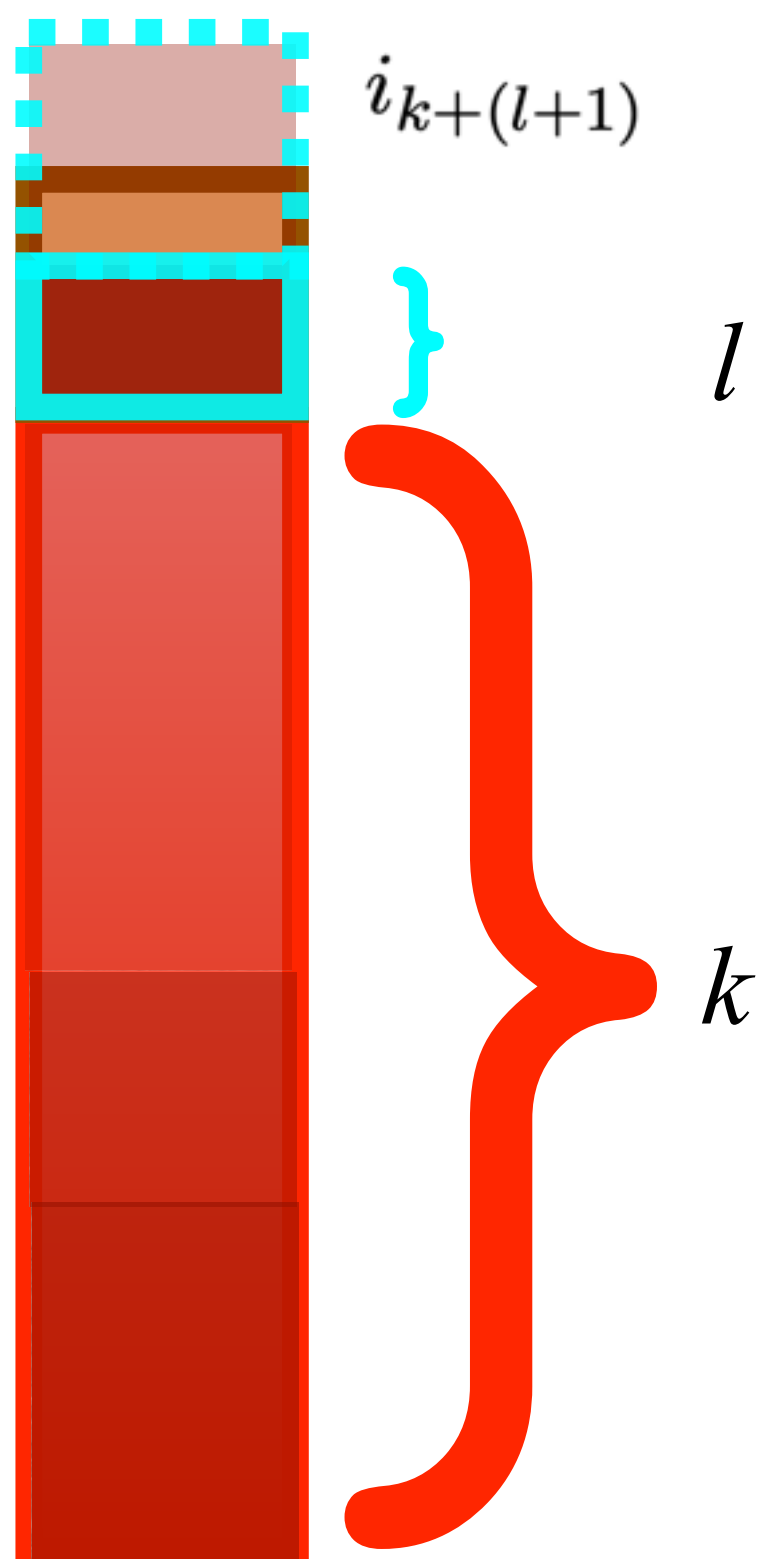
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{aligned} \text{a) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\ \text{b) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \end{aligned}$$



# Gütegarantie

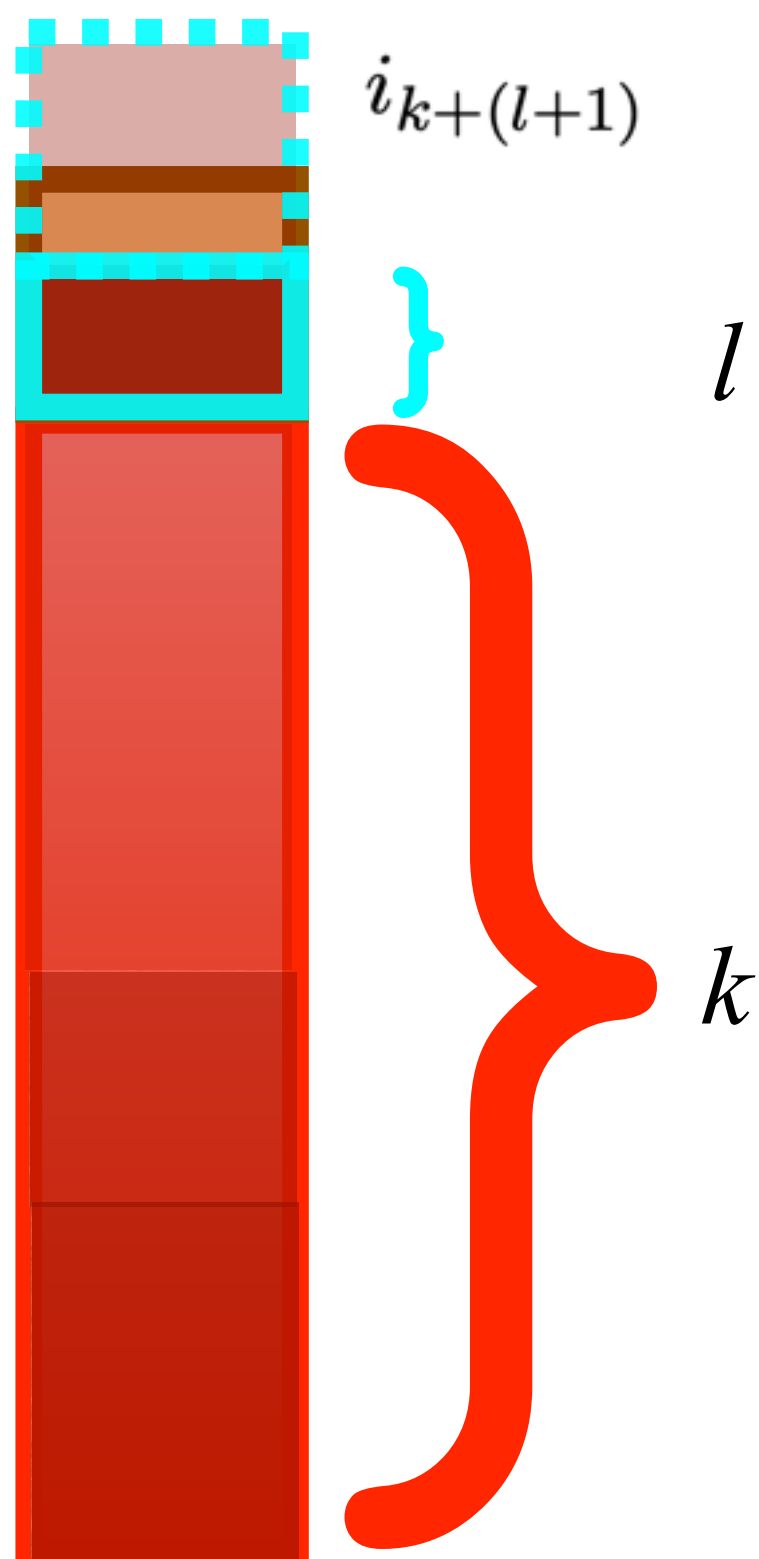
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{aligned} \text{a) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\ \text{b) } & \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \end{aligned}$$





# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

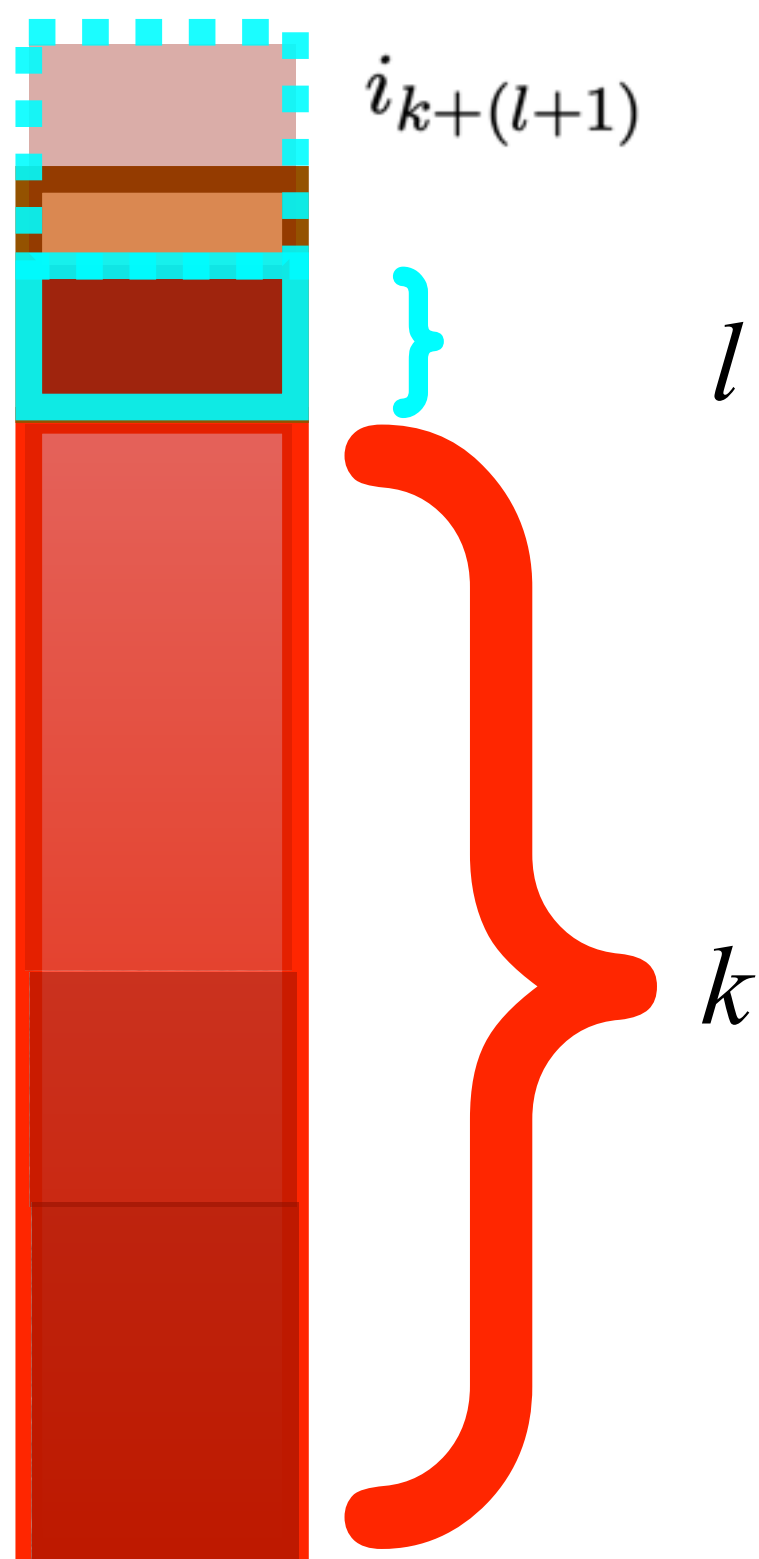
Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

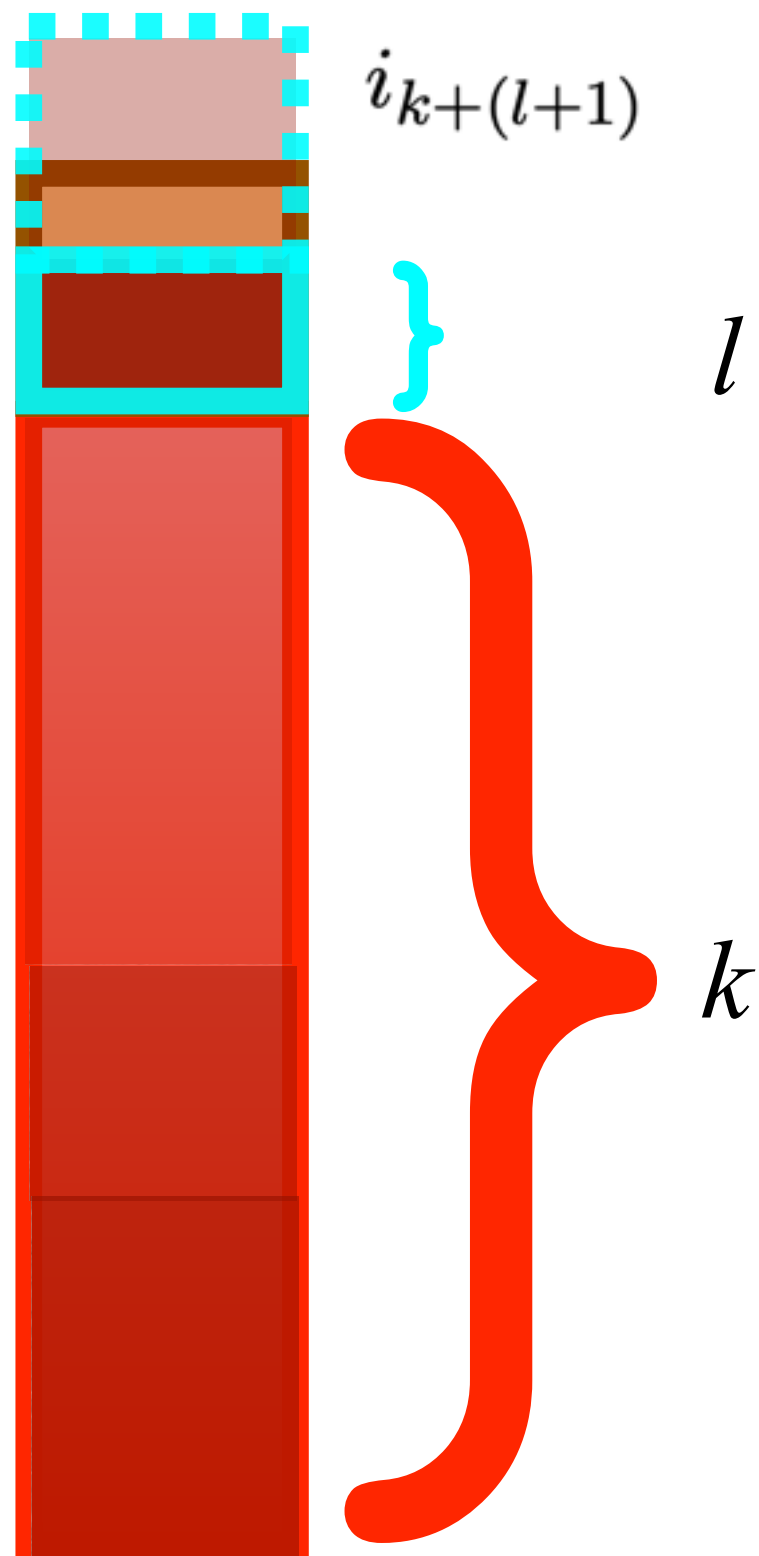
Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

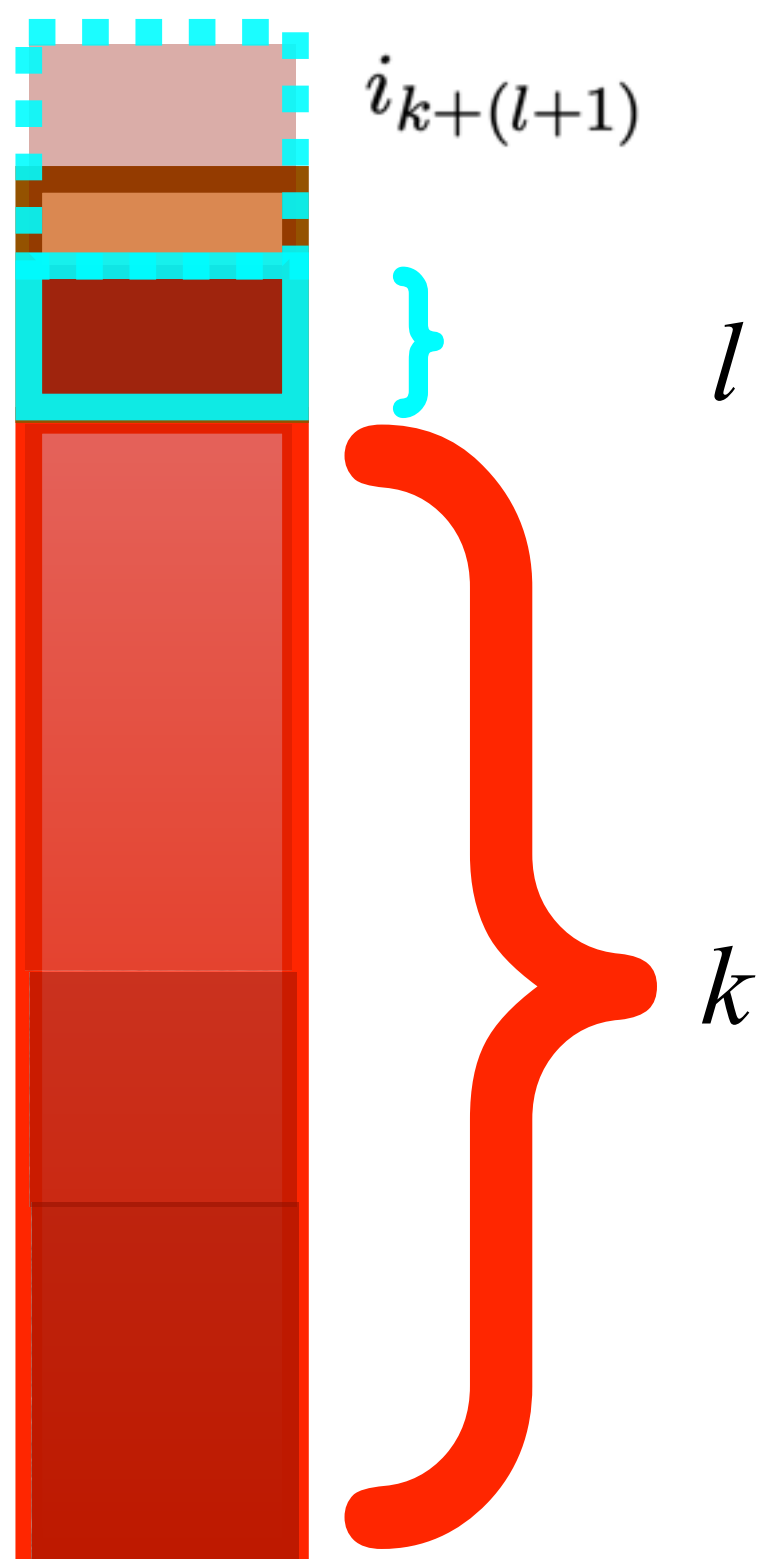
Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

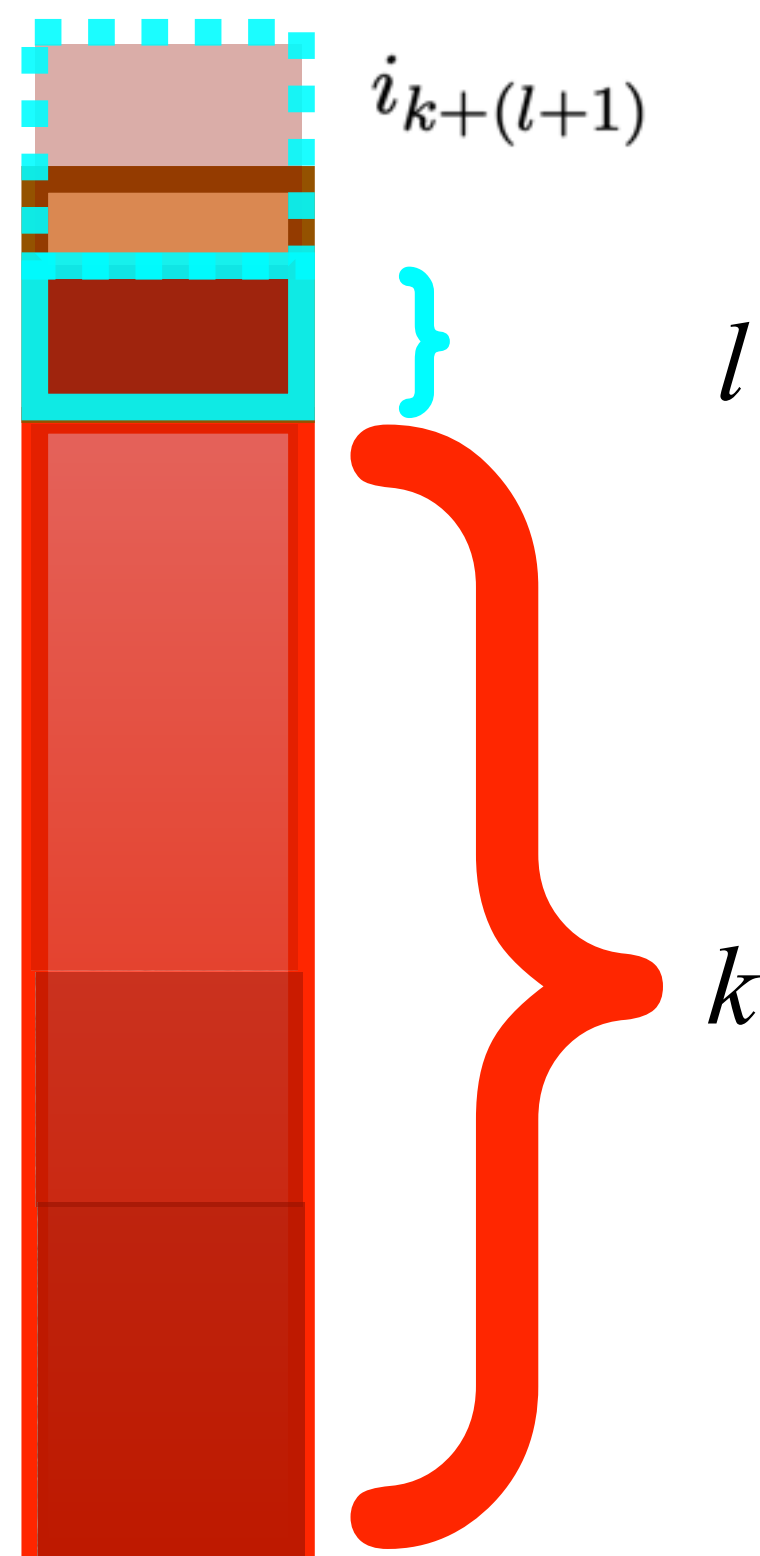
Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte  $\text{OPT}$  mit  $S^* \subseteq \{1, \dots, n\}$ .

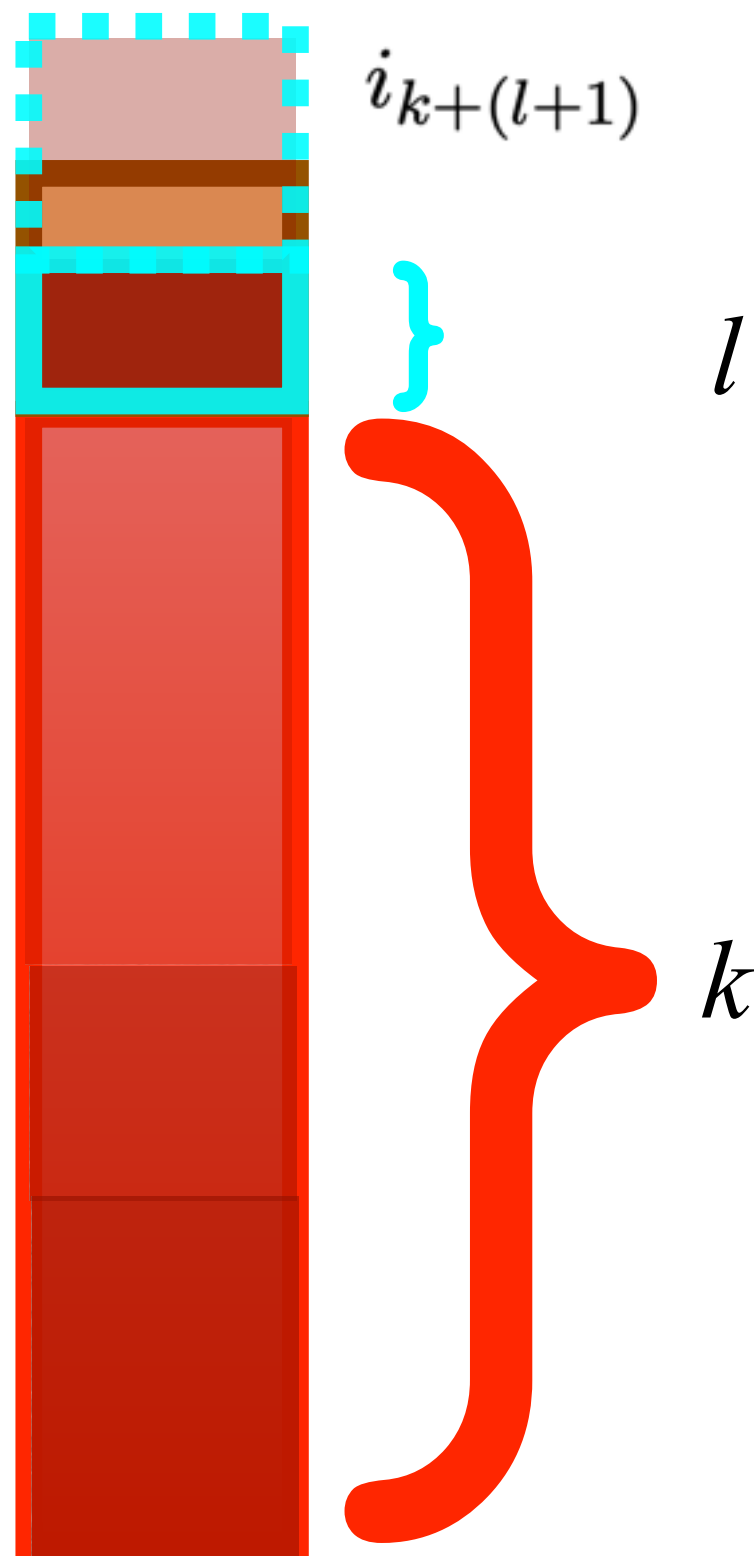
$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$G_k + \frac{1}{k} G_k :$



# Gütegarantie

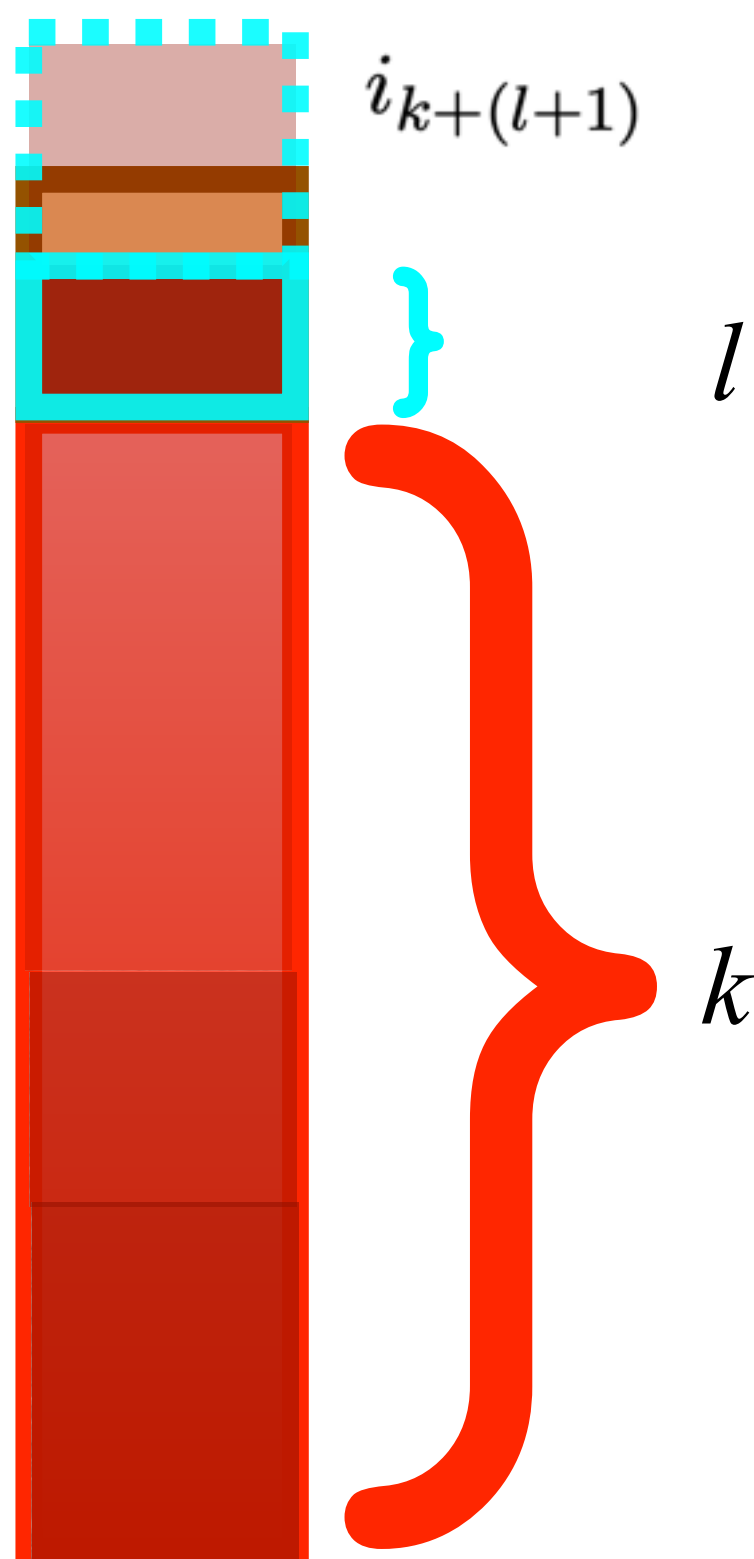
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{array}{l}
 \text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\
 \text{b) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \\
 \text{c) } p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k
 \end{array}
 \left. \vphantom{\begin{array}{l} \text{a) } \\ \text{b) } \\ \text{c) } \end{array}} \right\} G_k + \frac{1}{k} G_k :$$



# Gütegarantie

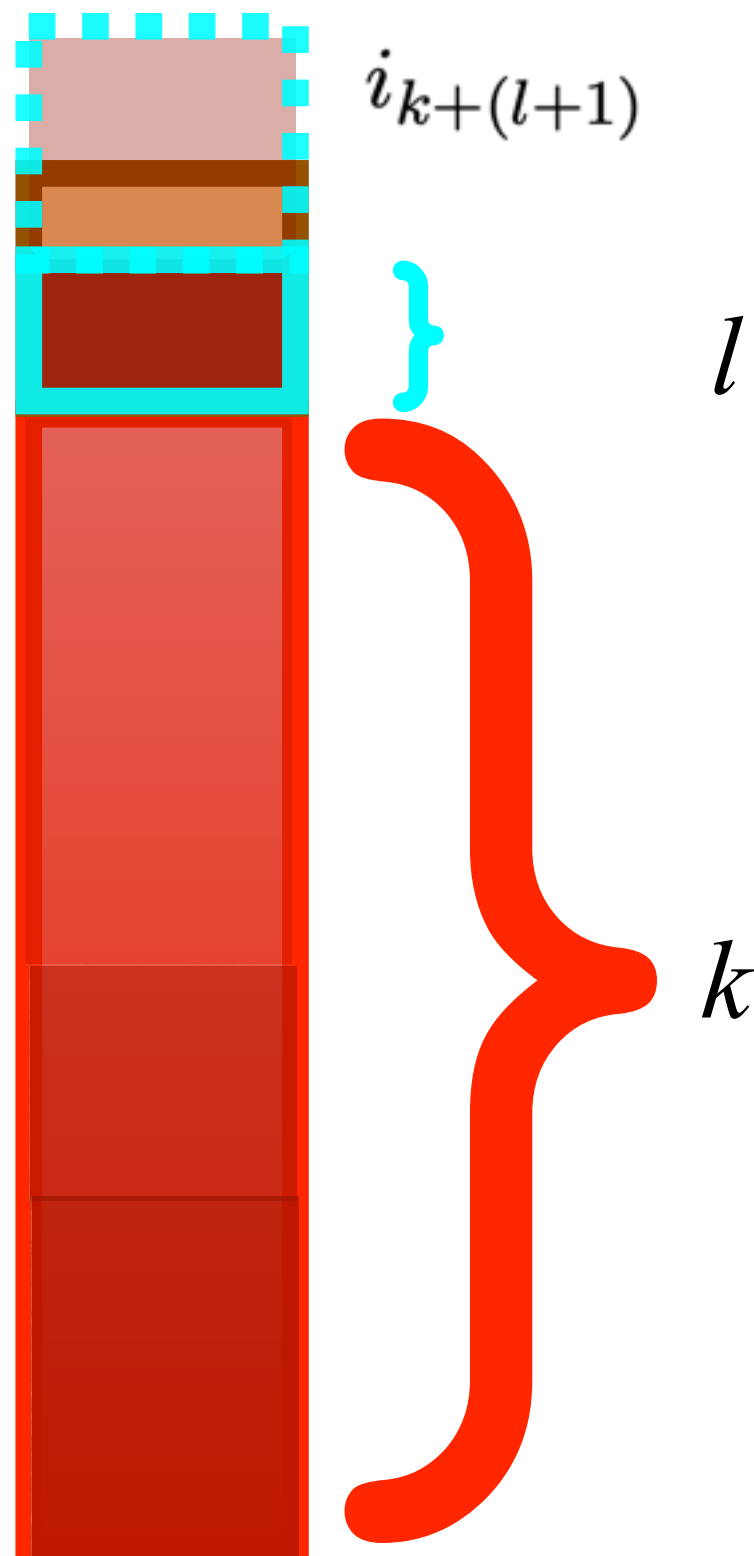
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{array}{l}
 \text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\
 \text{b) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \\
 \text{c) } p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k
 \end{array}
 \left. \vphantom{\begin{array}{l} \text{a) } \\ \text{b) } \\ \text{c) } \end{array}} \right\} G_k + \frac{1}{k} G_k :$$



# Gütegarantie

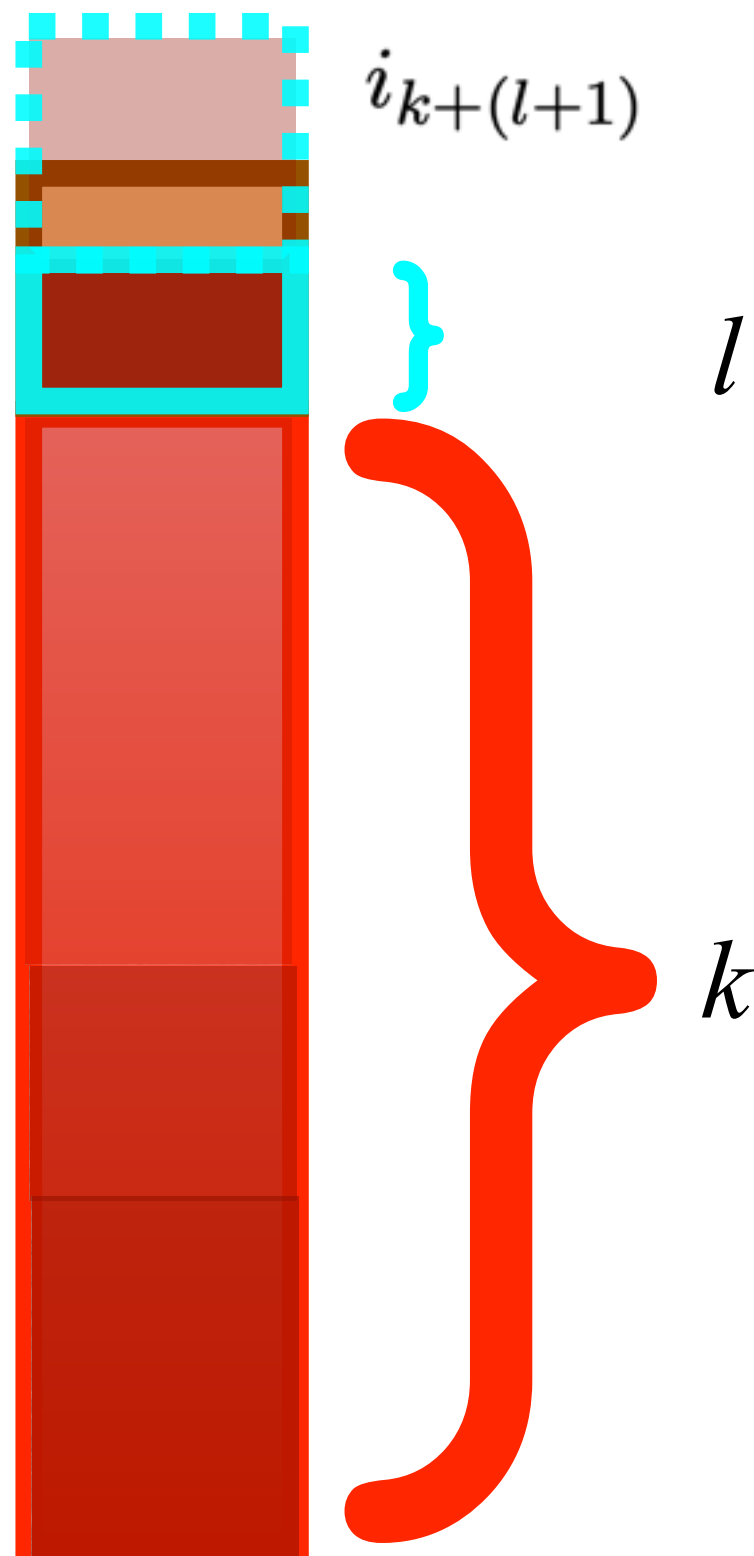
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{array}{l}
 \text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\
 \text{b) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \\
 \text{c) } p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k
 \end{array}
 \left. \vphantom{\begin{array}{l} \text{a) } \\ \text{b) } \\ \text{c) } \end{array}} \right\} G_k + \frac{1}{k} G_k :$$





# Gütegarantie

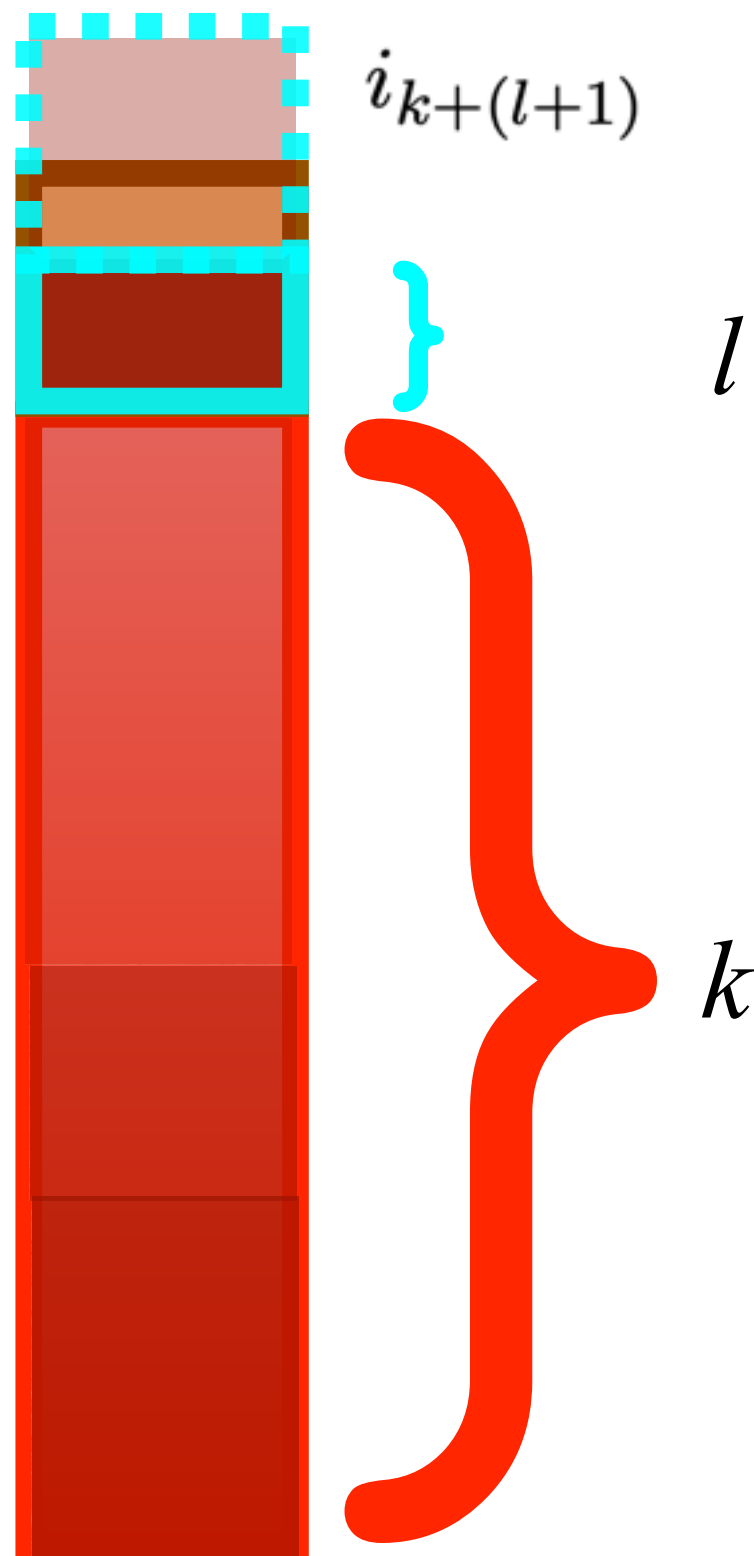
**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

$$\begin{array}{l}
 \text{a) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT} \\
 \text{b) } \sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k \\
 \text{c) } p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k
 \end{array}
 \left. \vphantom{\begin{array}{l} \text{a) } \\ \text{b) } \\ \text{c) } \end{array}} \right\} G_k + \frac{1}{k} G_k$$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

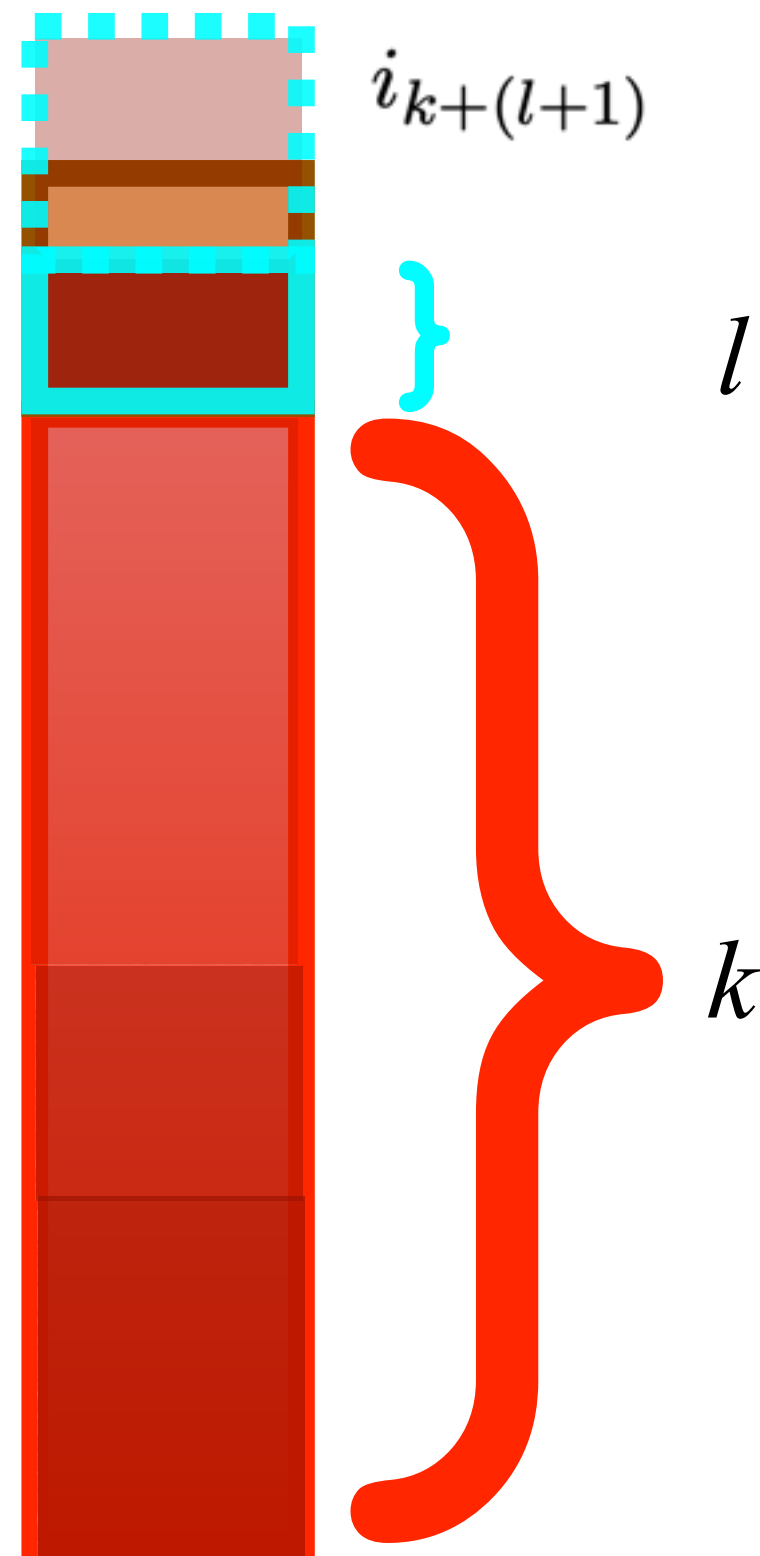
$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$G_k + \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

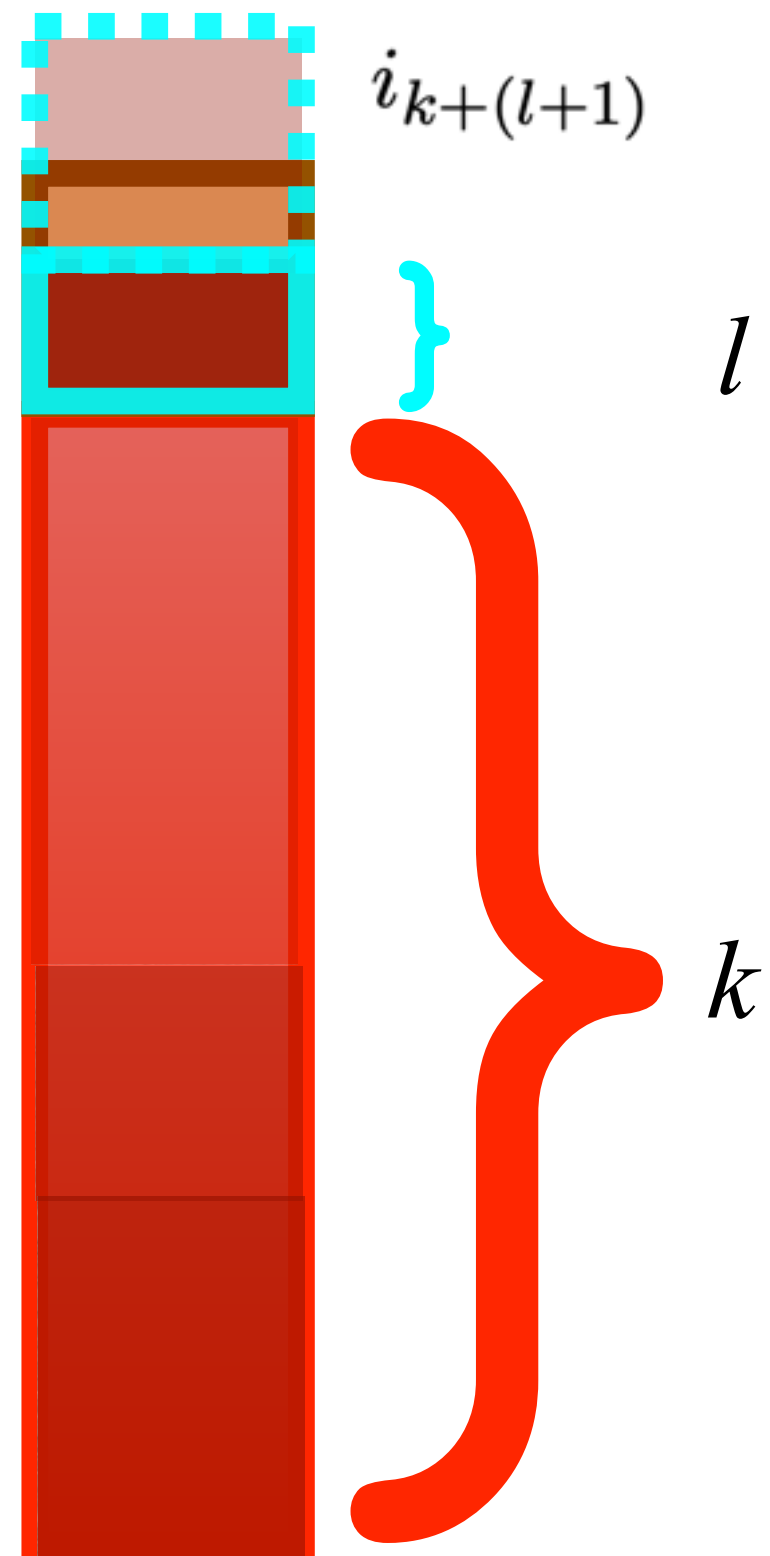
$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$G_k + \frac{1}{k} G_k$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

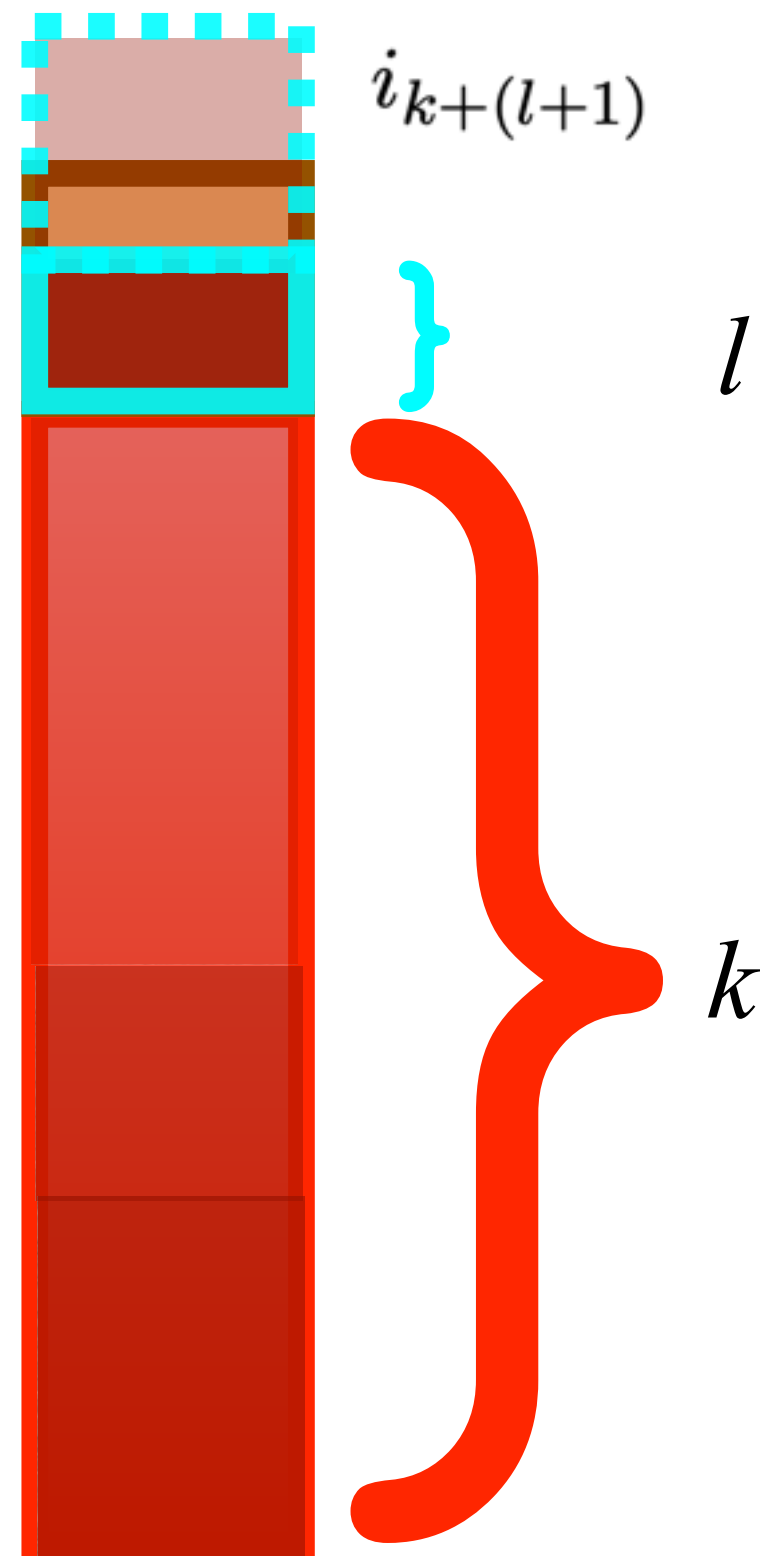
$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$G_k + \frac{1}{k} G_k \geq \text{OPT}$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

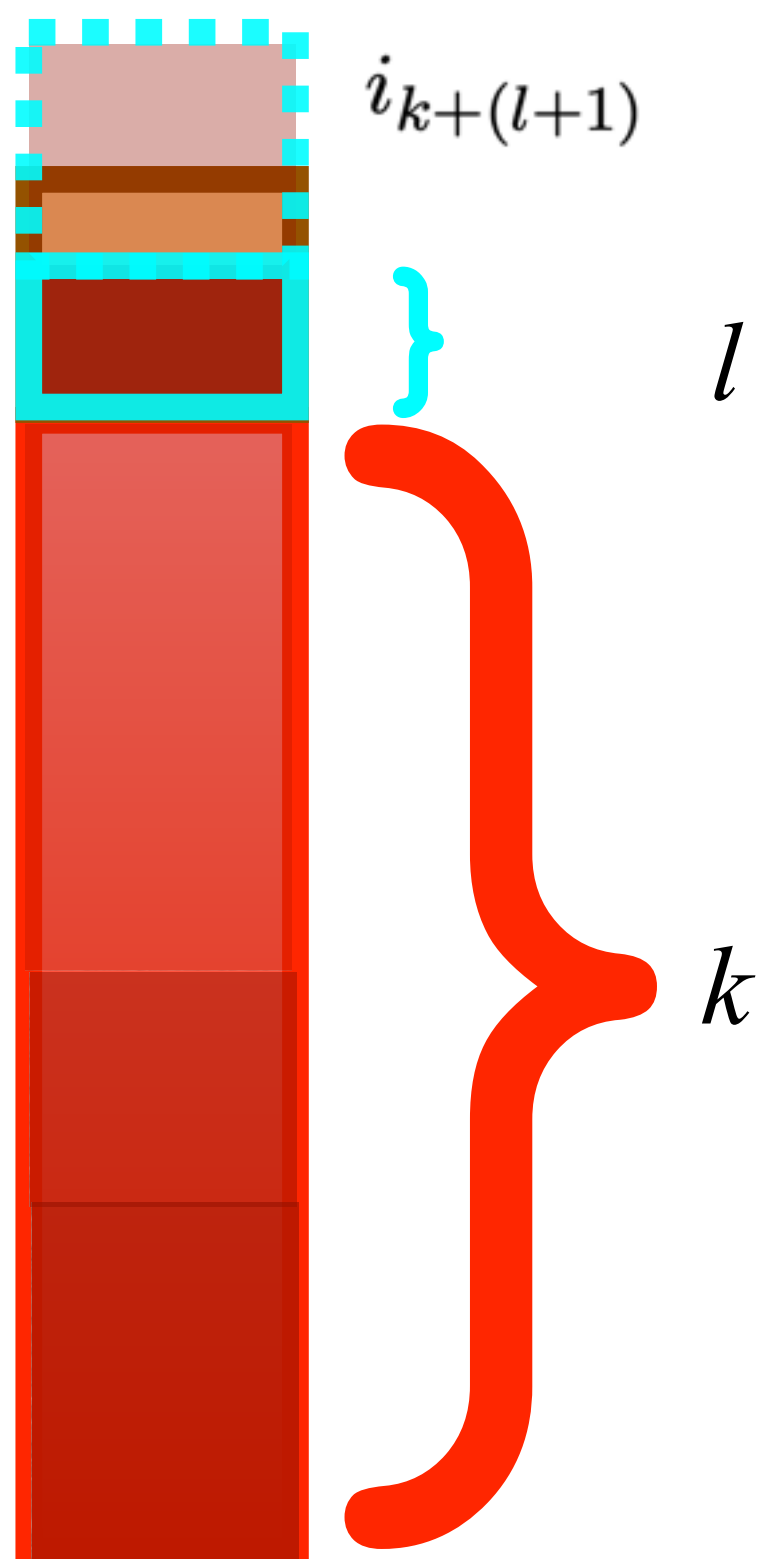
a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$G_k + \frac{1}{k} G_k \geq \text{OPT}$

$\Leftrightarrow G_k \geq \frac{k}{k+1} \text{OPT}$



# Gütegarantie

**Satz 4.7.**  $\text{GREEDY}_k$  ist ein  $(1 - \frac{1}{k+1})$ -Approximationsalgorithmus.

b) für den Vergleich mit dem zugehörigen Optimalwert  $\text{OPT}(I)$  gilt:  $\text{ALG}(I) \geq c \cdot \text{OPT}(I)$  (dabei ist  $c \leq 1$ )

Betrachte OPT mit  $S^* \subseteq \{1, \dots, n\}$ .

$|S^*| > k \implies$

a)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} + p_{i_{k+(l+1)}} \geq \text{OPT}$

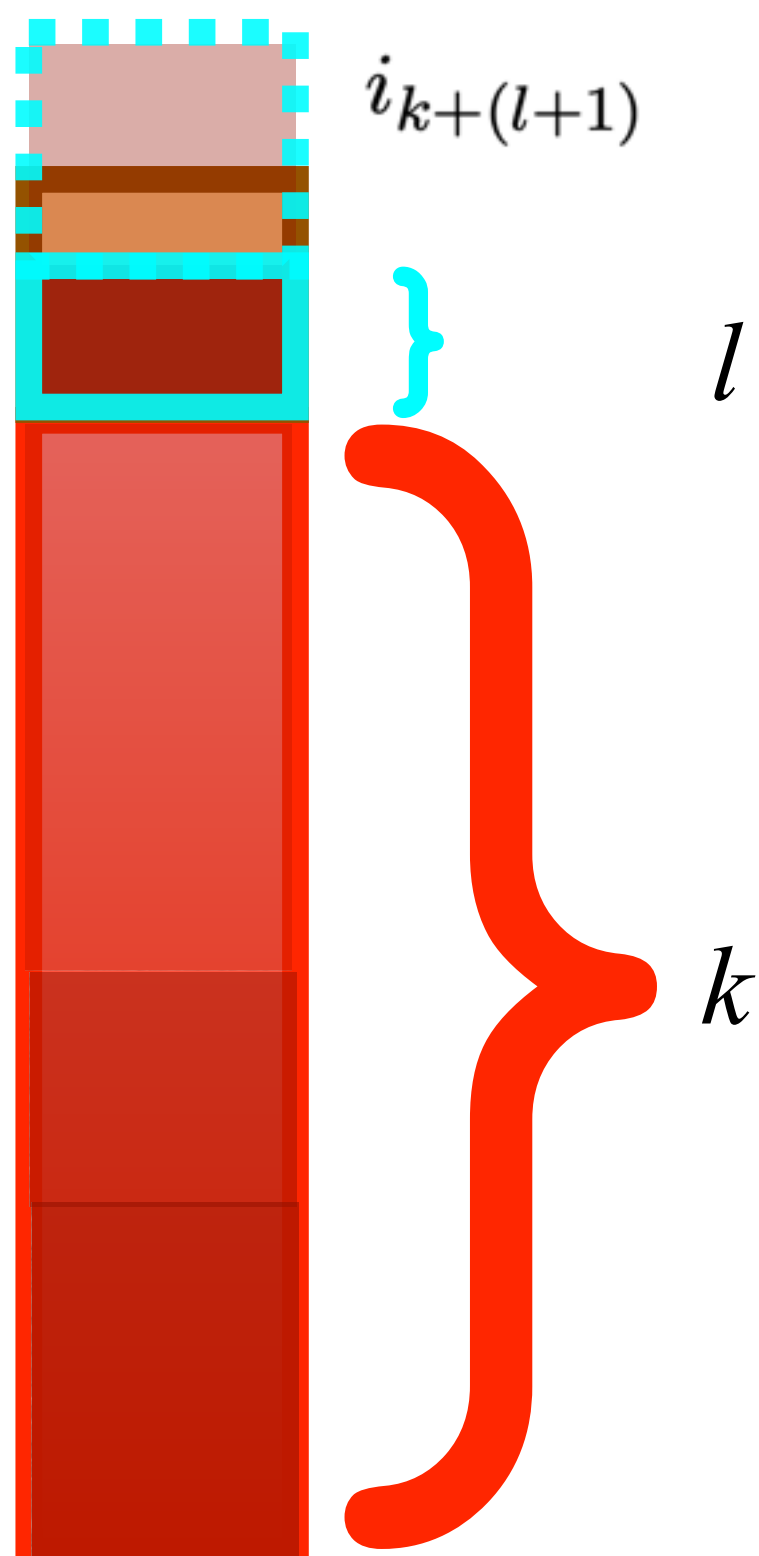
b)  $\sum_{j=1}^k p_{i_j} + \sum_{j=1}^l p_{i_{k+j}} \leq G_k$

c)  $p_{i_{k+(l+1)}} \leq \frac{1}{k} G_k$

$$G_k + \frac{1}{k} G_k \geq \text{OPT}$$

$$\Leftrightarrow G_k \geq \frac{k}{k+1} \text{OPT}$$

$$= \left(1 - \frac{1}{k+1}\right) \text{OPT}$$



# Zusatzbemerkungen

# Zusatzbemerkungen





# Zusatzbemerkungen

Man kann zeigen:

# Zusatzbemerkungen

Man kann zeigen:

- Für  $\text{GREEDY}_k$  ist der Faktor  $(1 - \frac{1}{k+1})$  bestmöglich.

# Zusatzbemerkungen

Man kann zeigen:

- Für  $\text{GREEDY}_k$  ist der Faktor  $(1 - \frac{1}{k+1})$  bestmöglich.
- Wenn man  $G_0$  in Zeile 4 von Algorithmus 4.6 durch  $G'_0$  ersetzt, bekommt man einen Approximationsalgorithmus mit Gütegarantie  $(1 - \frac{1}{k+2})$ .

# Approximationschemata

# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

**Definition 4.8 (PTAS).** Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste  $\epsilon > 0$  einen  $(1 - \epsilon)$ -Approximationsalgorithmus (bzw.  $(1 + \epsilon)$ ) liefert.

# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

**Definition 4.8 (PTAS).** Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste  $\epsilon > 0$  einen  $(1 - \epsilon)$ -Approximationsalgorithmus (bzw.  $(1 + \epsilon)$ ) liefert.

# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

**Definition 4.8 (PTAS).** Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste  $\epsilon > 0$  einen  $(1 - \epsilon)$ -Approximationsalgorithmus (bzw.  $(1 + \epsilon)$ ) liefert.



# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

**Definition 4.8 (PTAS).** Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste  $\epsilon > 0$  einen  $(1 - \epsilon)$ -Approximationsalgorithmus (bzw.  $(1 + \epsilon)$ ) liefert.

# Approximationschemata

Man sieht: Für jedes feste  $\epsilon > 0$  gibt es einen polynomiellen Algorithmus für Knapsack, der eine  $(1 - \epsilon)$ -Approximation liefert. Das motiviert:

**Definition 4.8** (PTAS). Ein polynomielles Approximationsschema (Engl.: Polynomialtime approximation scheme. Kurz: PTAS) für ein Optimierungsproblem ist eine Familie von Algorithmen, die für jedes beliebige, aber feste  $\epsilon > 0$  einen  $(1 - \epsilon)$ -Approximationsalgorithmus (bzw.  $(1 + \epsilon)$ ) liefert.

**Korollar 4.9.**  $\{\text{GREEDY}_k \mid k \in \mathbb{N}\}$  ist ein PTAS für Knapsack.

# 4.4 Ausblicke

# Approximationsfaktoren

# Approximationsfaktoren

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
	8	32	11	12	13
7	40	5	28	20	16
32	9	9	3	9	24
2	8	14	15	16	10
	2	20	40	24	
		3	10	4	

Kann Knut die Klausur bestehen?

# Approximationsfaktoren

## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	10
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

# Approximationsfaktoren

## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	24
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

**Max**

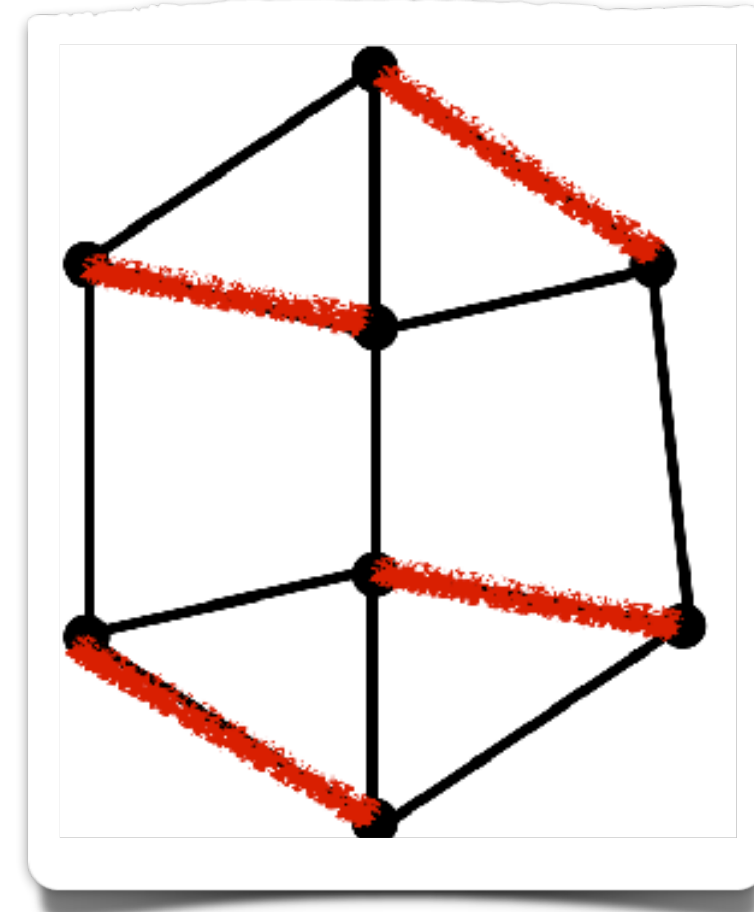
# Approximationsfaktoren

## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

**Max**





# Approximationsfaktoren

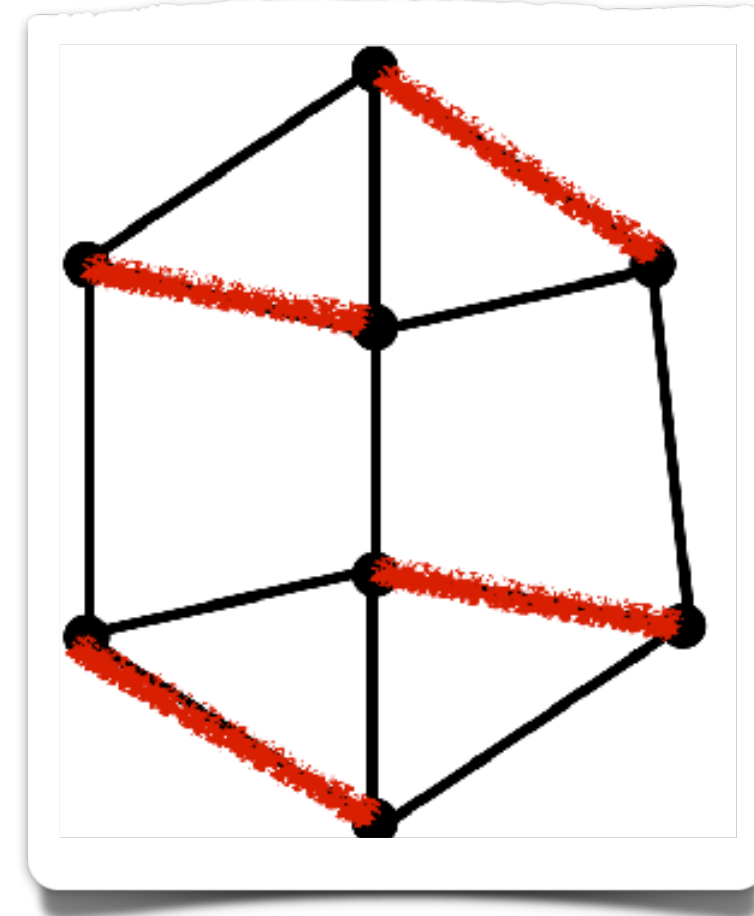
## 0-1 Knapsack

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
20	32	40	8	16	4	32	40	9	32	28	20	16	20	40	24
3	3	10	5	2	4	2	5	8	5	3	9	10	3	10	4
Punkte															

Kann Knut die Klausur bestehen?

**Max**

## Matching



# Approximationsfaktoren

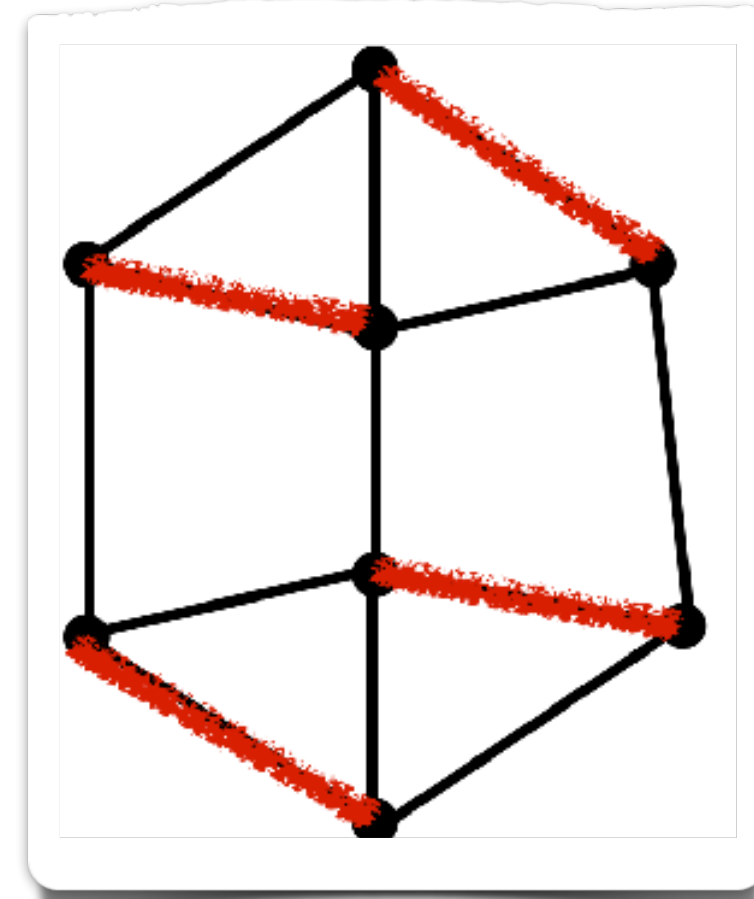
## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	4
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

**Max**

## Matching



**Max**

# Approximationsfaktoren

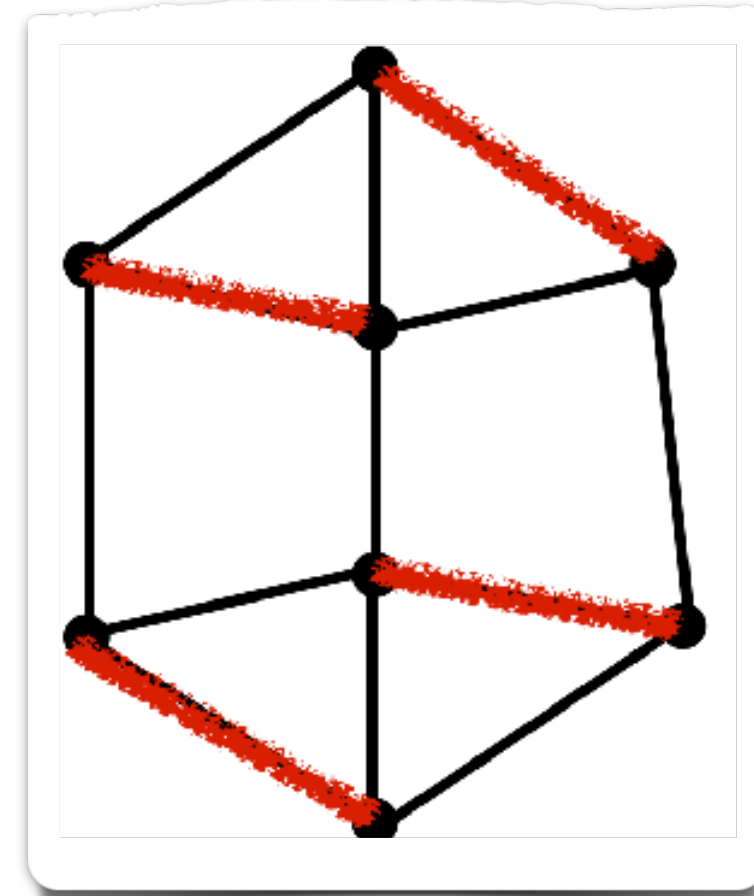
## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	11	3	9	24	10
32	14	15	16	4	
2	20	40			
	3	10			

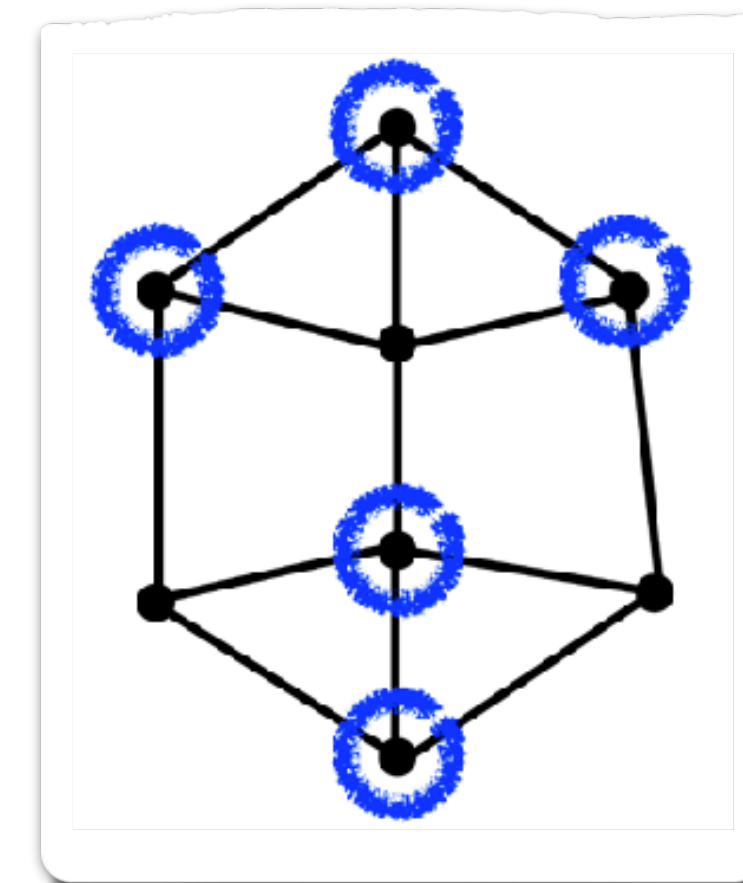
Kann Knut die Klausur bestehen?

**Max**

## Matching



**Max**



# Approximationsfaktoren

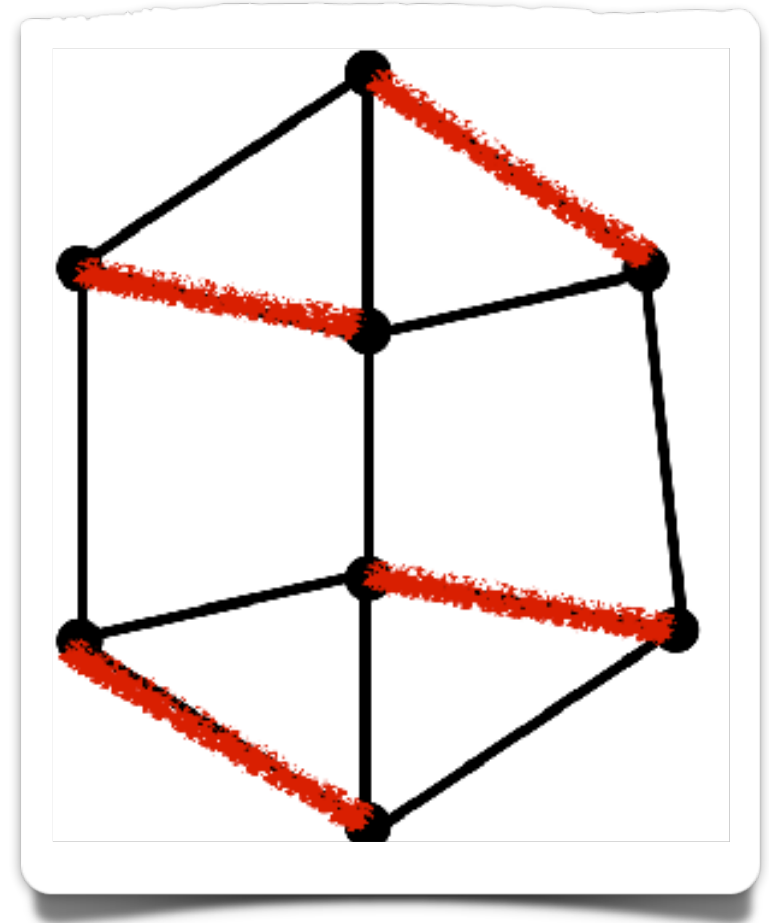
## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	11	3	9	24	10
32	14	15	16	4	
2	20	40			
	3	10			

Kann Knut die Klausur bestehen?

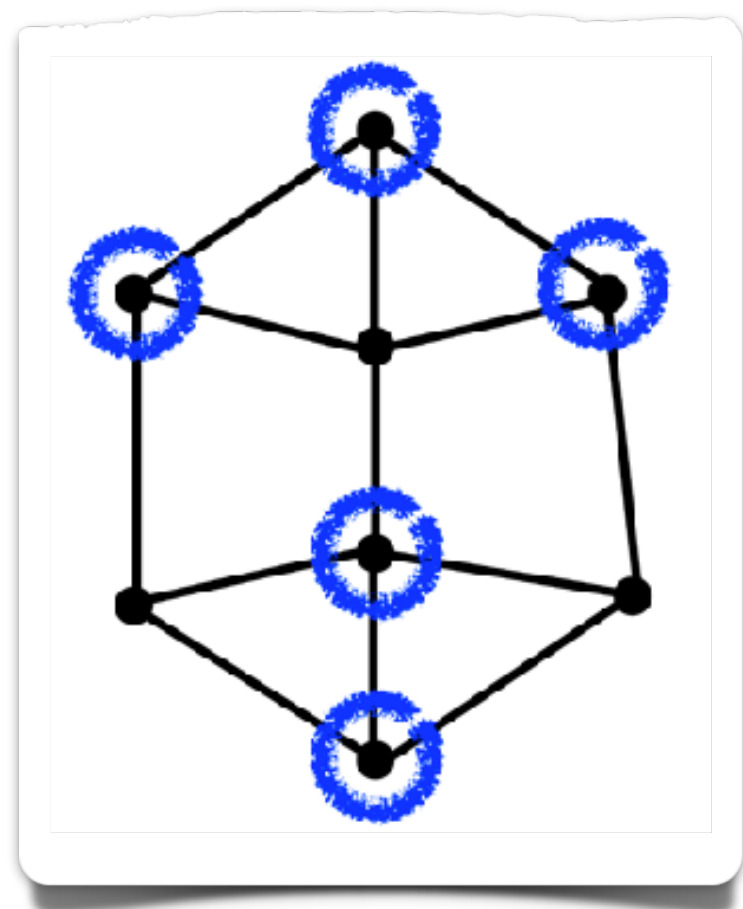
**Max**

## Matching



**Max**

## Vertex Cover



# Approximationsfaktoren

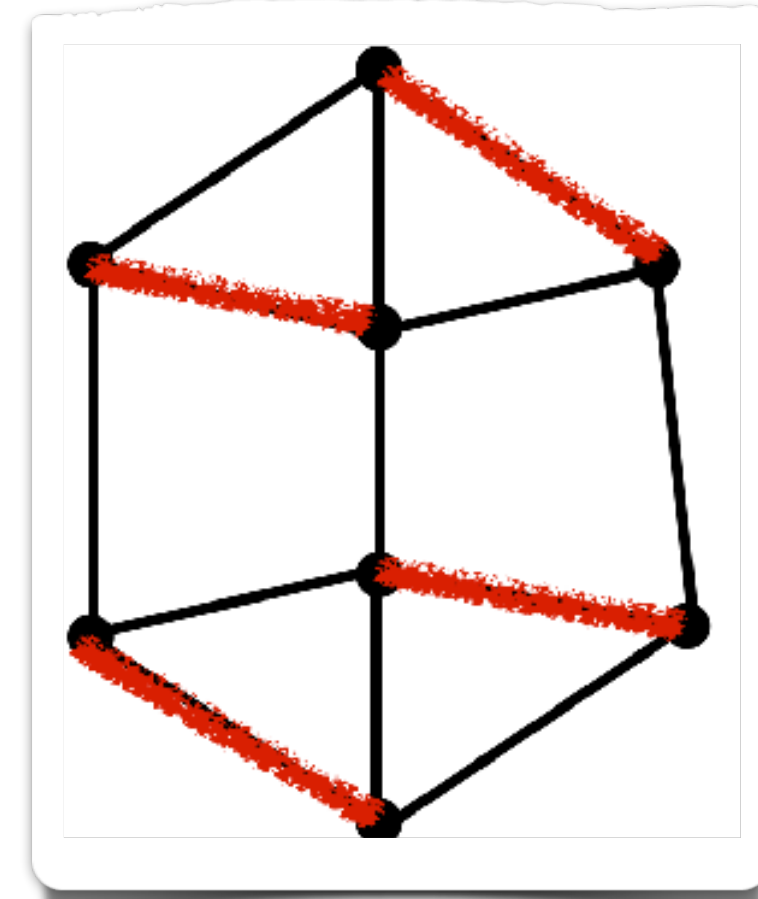
## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	28	20	16	16
9	9	3	9	24	10
32	8	14	15	4	
2	2	20	40		
		3	10		

Kann Knut die Klausur bestehen?

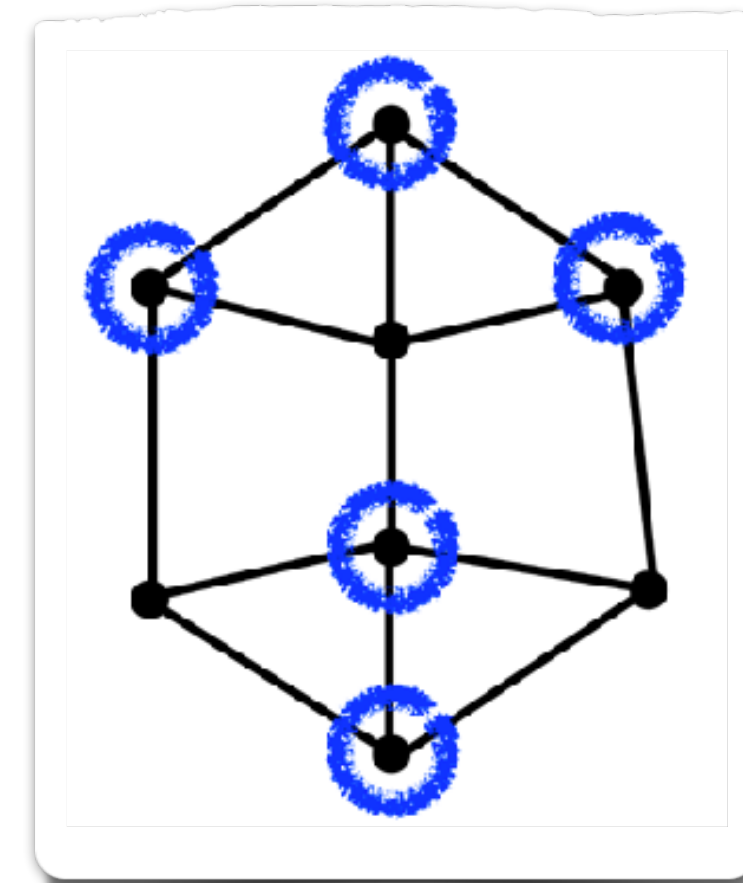
**Max**

## Matching



**Max**

## Vertex Cover



**Min**

# Approximationsfaktoren

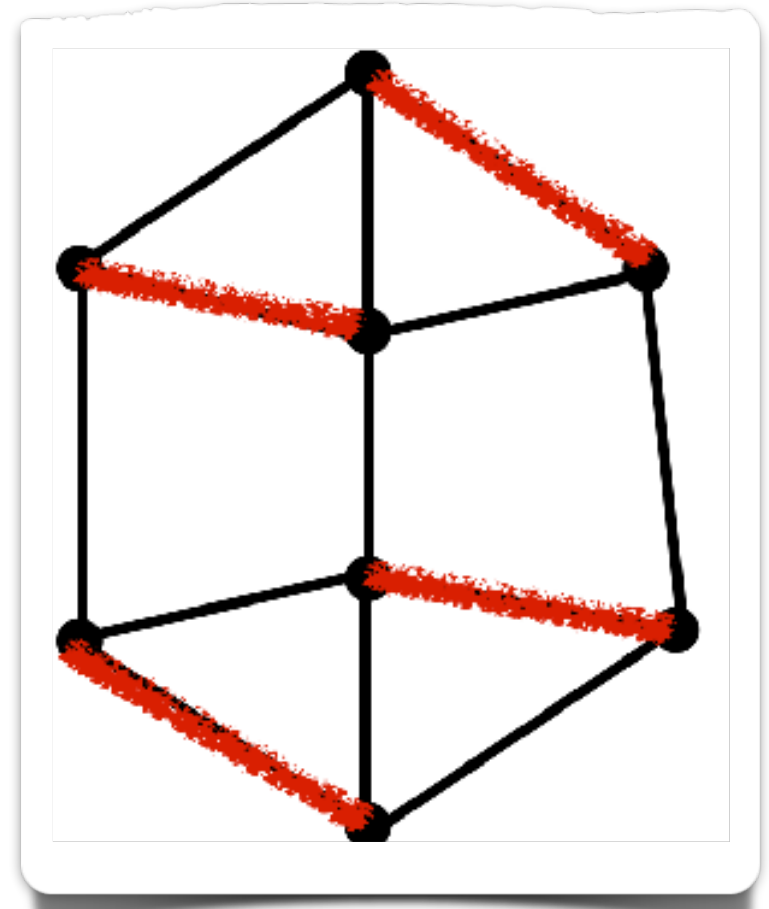
## 0-1 Knapsack

Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	9	8
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10
16	24	4

Kann Knut die Klausur bestehen?

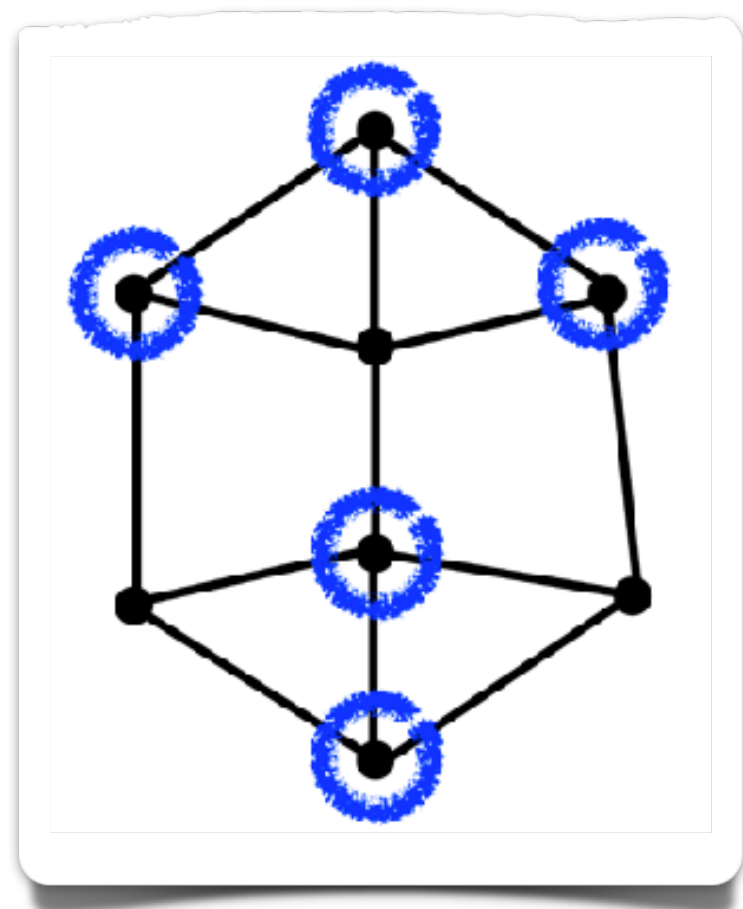
**Max**

## Matching



**Max**

## Vertex Cover



**Min**

## Traveling Salesman

# Approximationsfaktoren

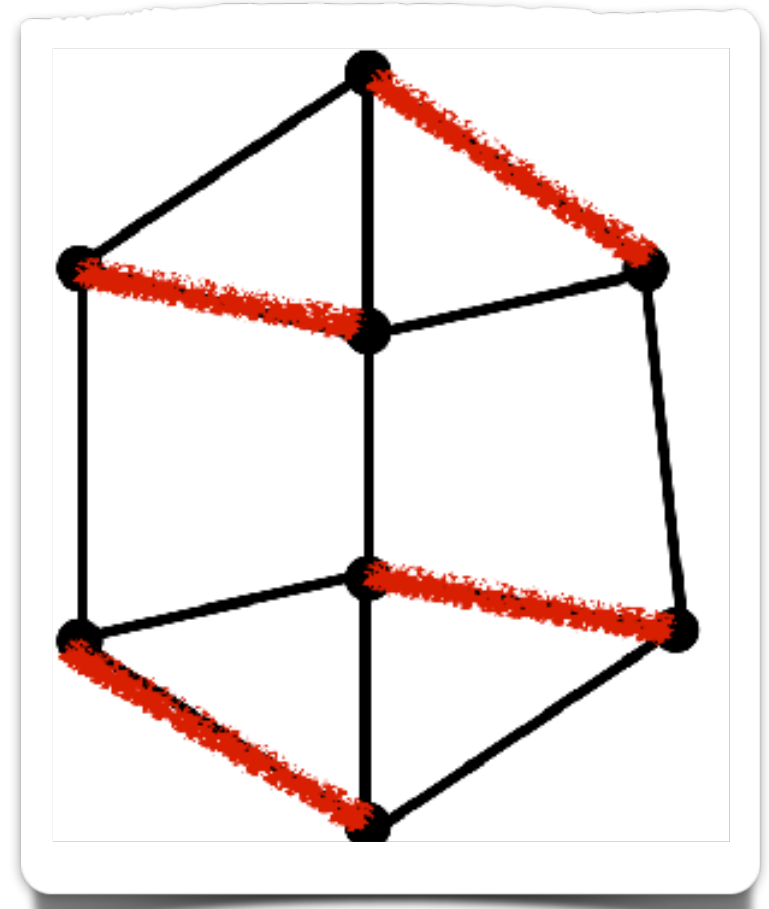
## 0-1 Knapsack

1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	5	28	20	16
9	9	11	3	9	24
32	8	14	15	16	10
2	2	20	40	4	
		3	10		

Kann Knut die Klausur bestehen?

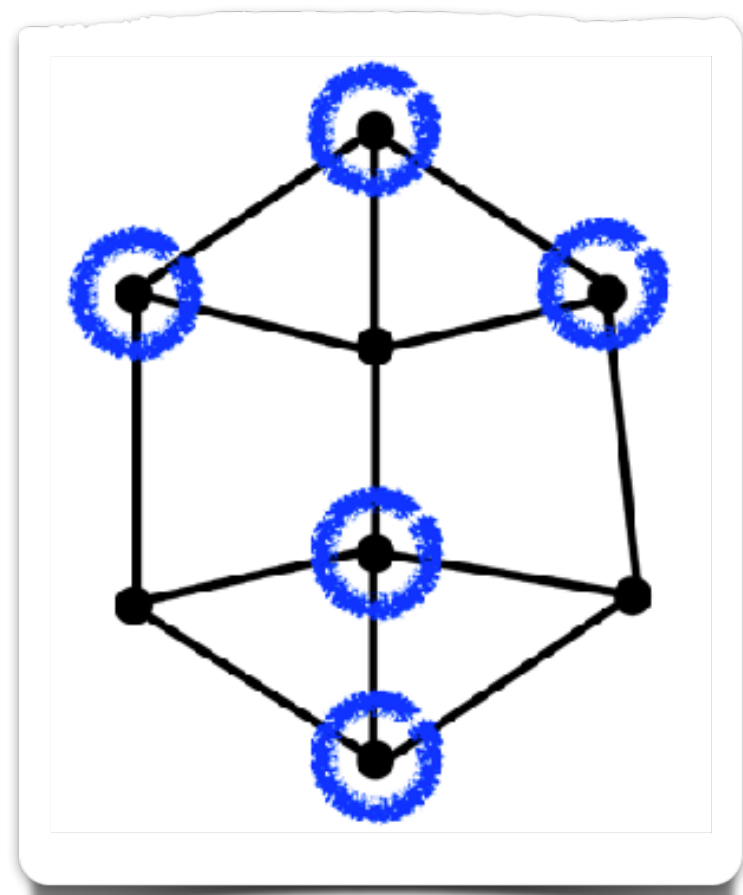
**Max**

## Matching



**Max**

## Vertex Cover



**Min**

## Traveling Salesman

**Min**

# Approximationsfaktoren

## 0-1 Knapsack

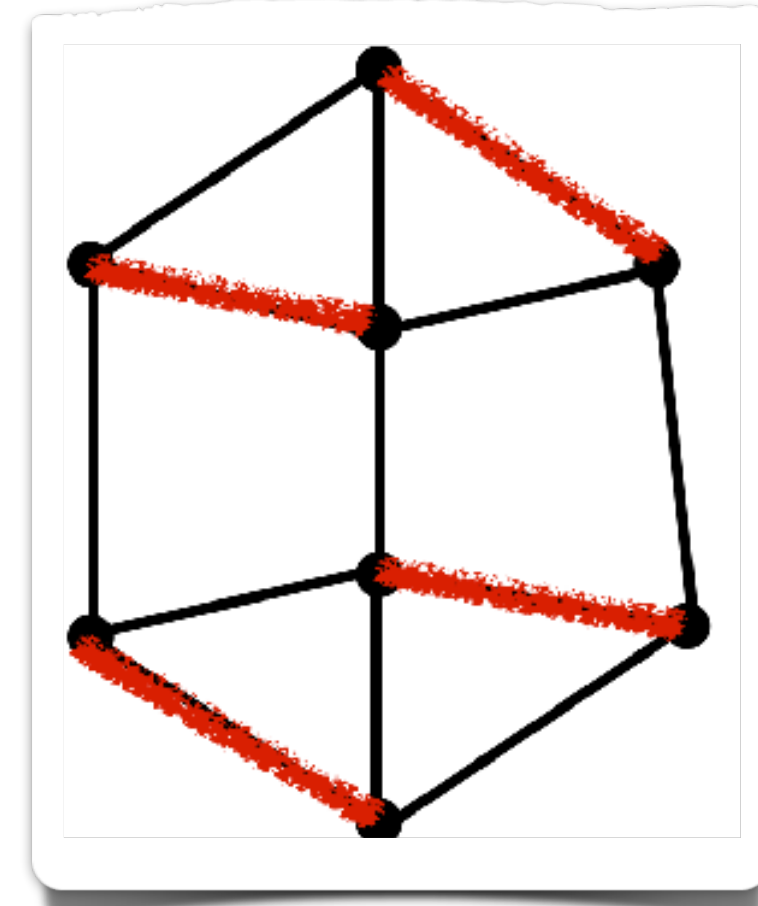
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	11	12	16	13
40	32	28	20	24	16
9	5	3	9	4	10
7	9	14	15	16	13
32	8	20	40	24	16
2	2	3	10	4	10

Kann Knut die Klausur bestehen?

**Max**

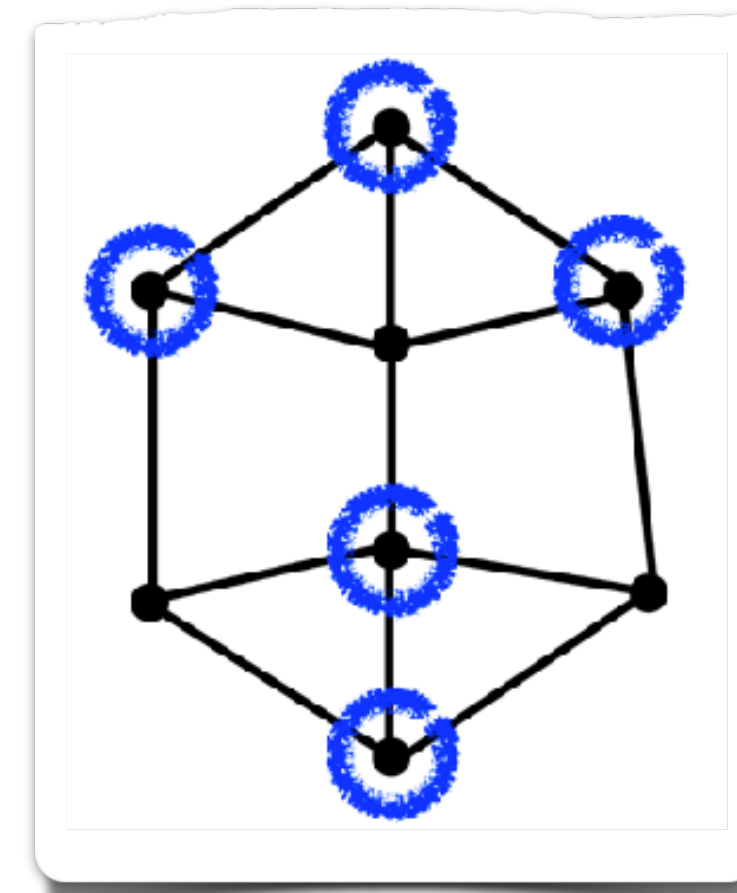
$$1 - \frac{1}{k + 2}$$

## Matching



**Max**

## Vertex Cover



**Min**

## Traveling Salesman

**Min**



# Approximationsfaktoren

## 0-1 Knapsack

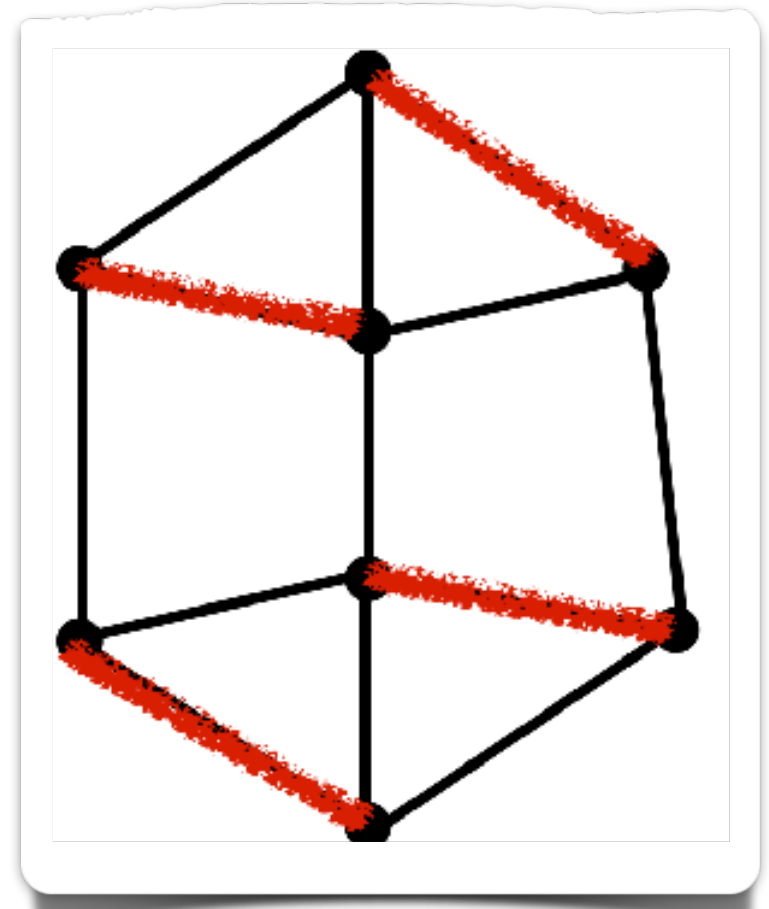
Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	16
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

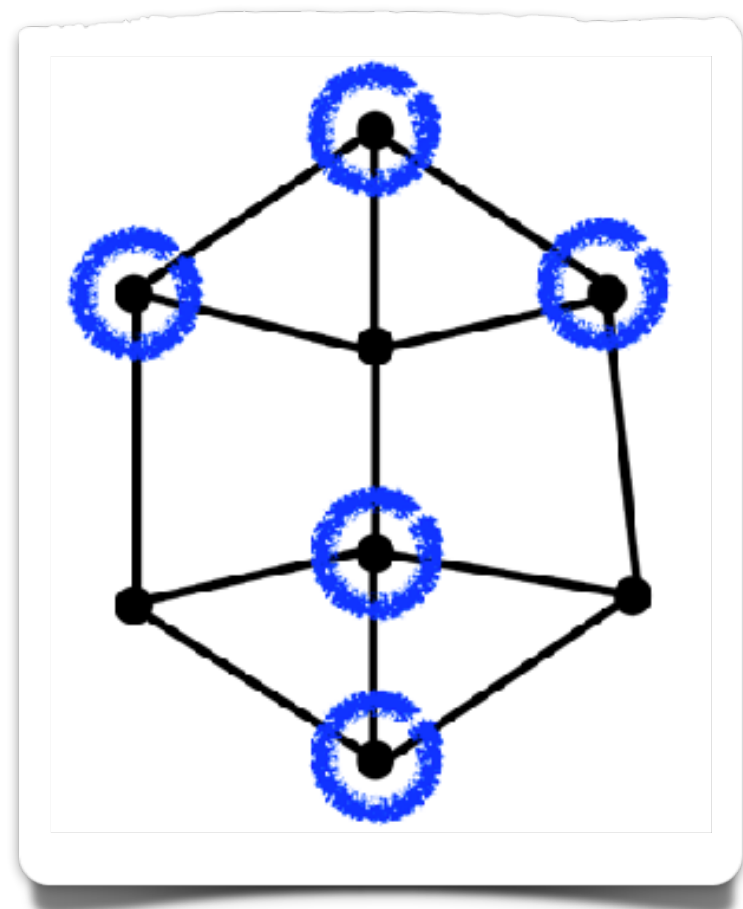
## Matching



**Max**

$$1$$

## Vertex Cover



**Min**

## Traveling Salesman

**Min**

# Approximationsfaktoren

## 0-1 Knapsack

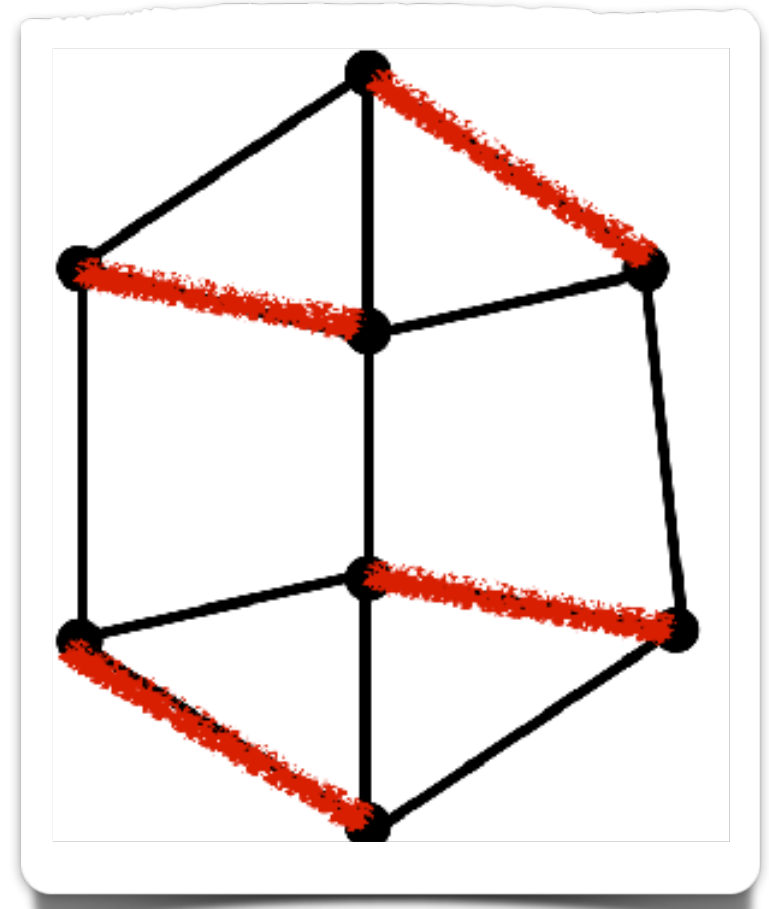
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	11	12	16	13
40	32	28	20	24	16
9	5	3	9	4	10
7	9	14	15	16	13
32	8	20	40	24	16
2	2	3	10	4	10

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

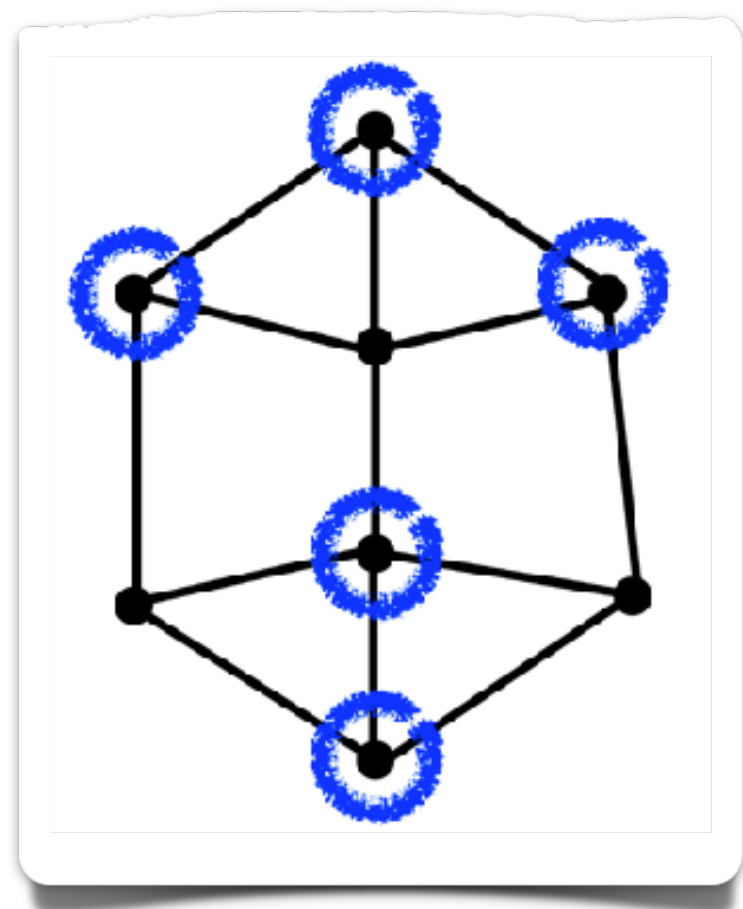
## Matching



**Max**

1

## Vertex Cover



**Min**

2

## Traveling Salesman

**Min**

# Approximationsfaktoren

## 0-1 Knapsack

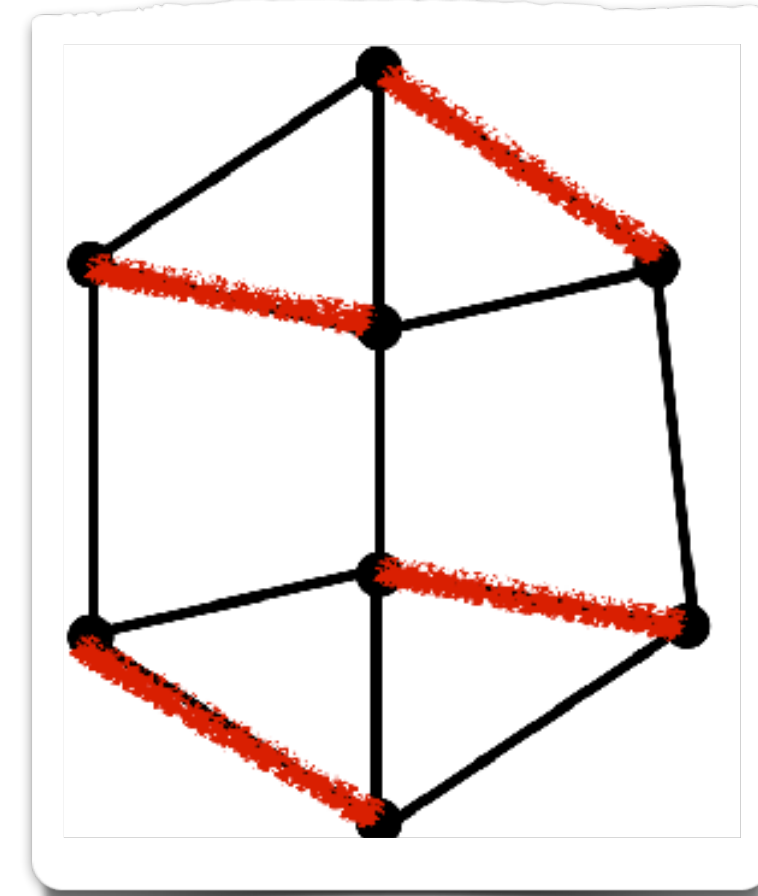
Nummer	Minuten	Punkte
1	20	3
2	32	3
3	40	8
4	8	5
5	16	2
6	4	4
7	32	2
8	9	8
9	8	2
10	10	5
11	28	3
12	20	9
13	16	4
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

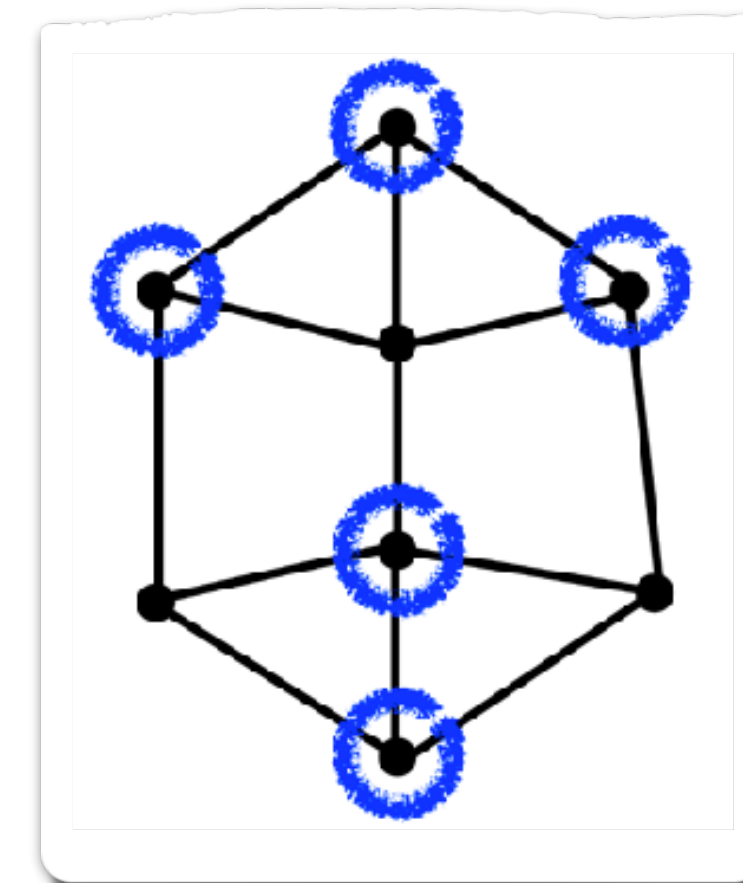
## Matching



**Max**

1

## Vertex Cover

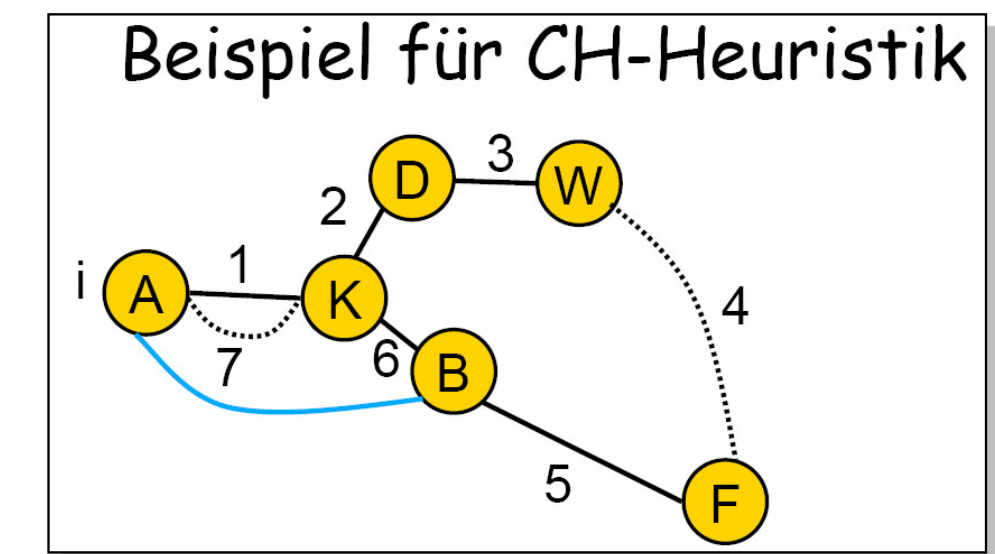


**Min**

2

## Traveling Salesman

**Min**



# Approximationsfaktoren

## 0-1 Knapsack

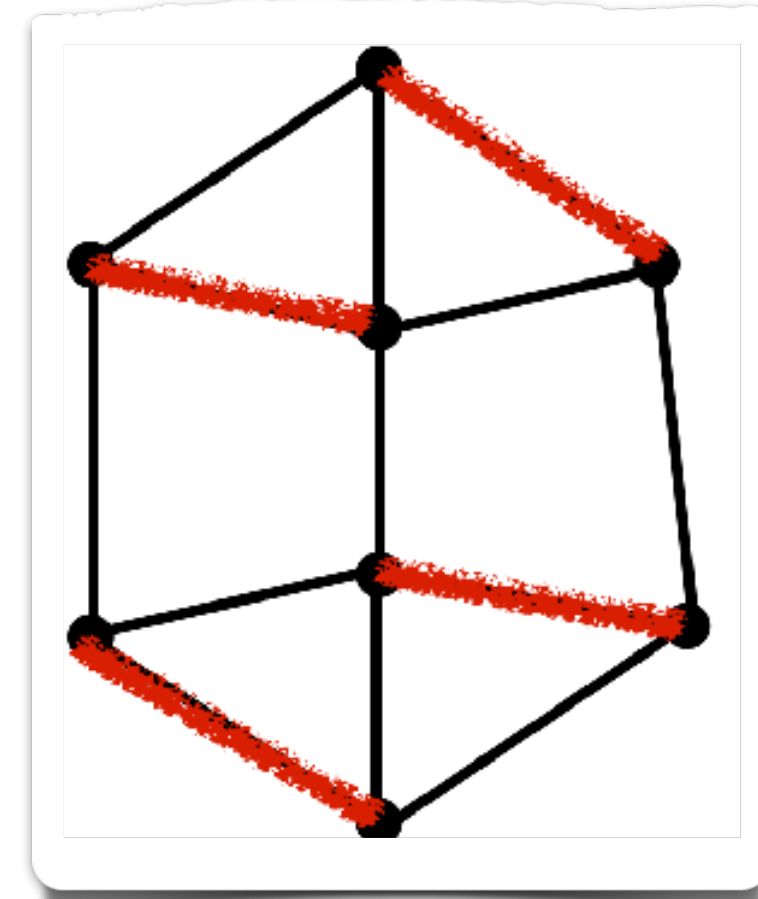
Nummer	Minuten	Punkte
1	20	3
2	32	3
3	10	10
4	8	5
5	16	2
6	4	4
7	32	2
8	40	9
9	8	2
10	32	5
11	28	3
12	20	9
13	16	4
14	20	3
15	40	10

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

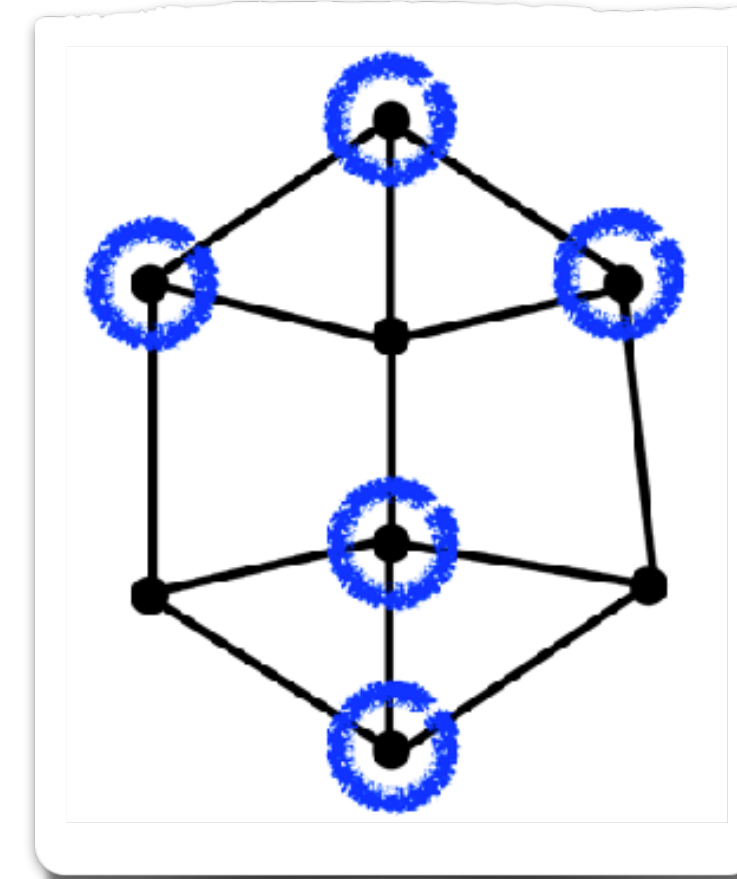
## Matching



**Max**

$$1$$

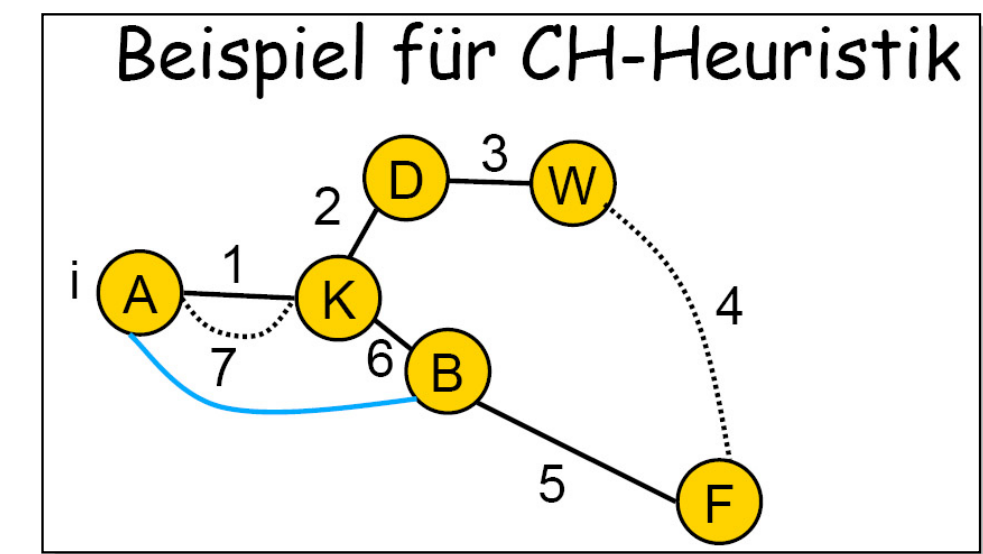
## Vertex Cover



**Min**

$$2$$

## Traveling Salesman



**Min**

$$\frac{3}{2}$$

# Approximationsfaktoren

## 0-1 Knapsack

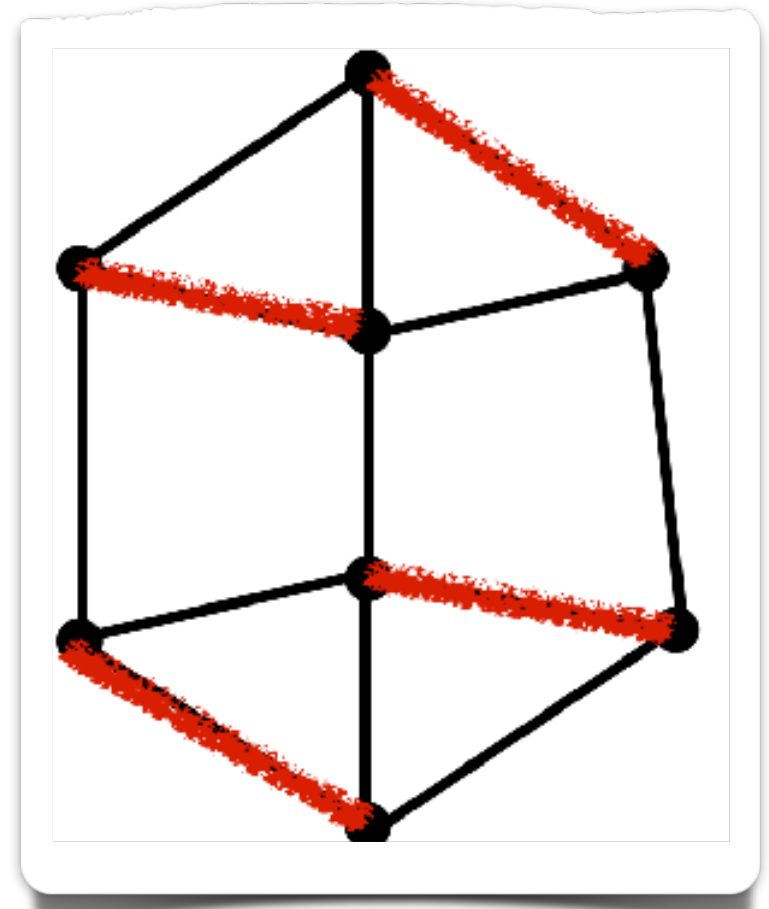
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	32	11	12	13
40	5	5	28	20	16
9	14	3	9	16	16
9	20	15	40	24	10
2	3	10	4	4	

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

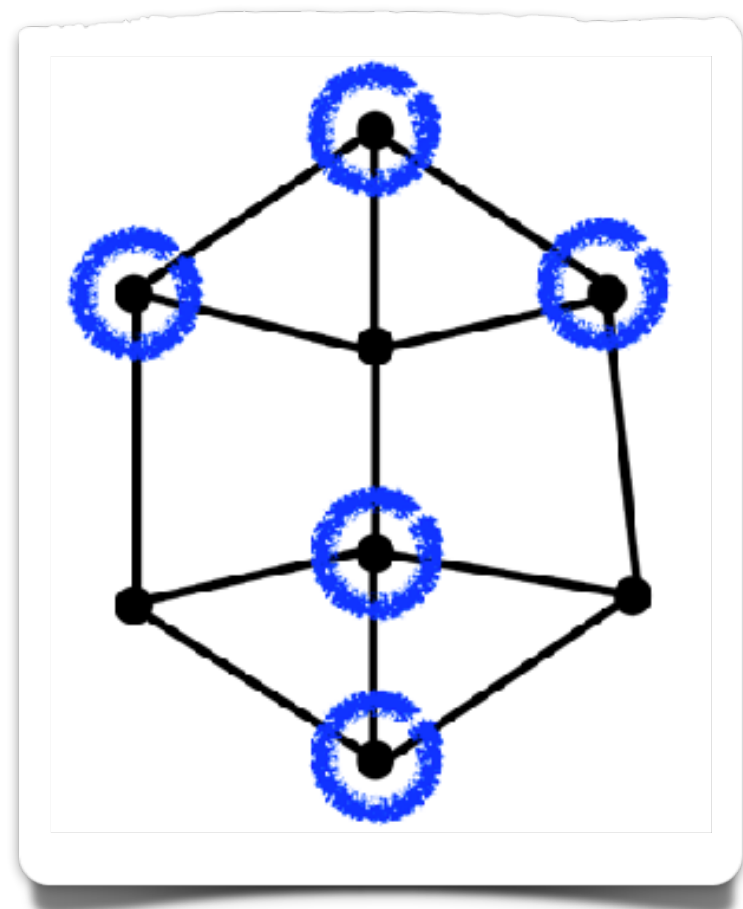
## Matching



**Max**

1

## Vertex Cover

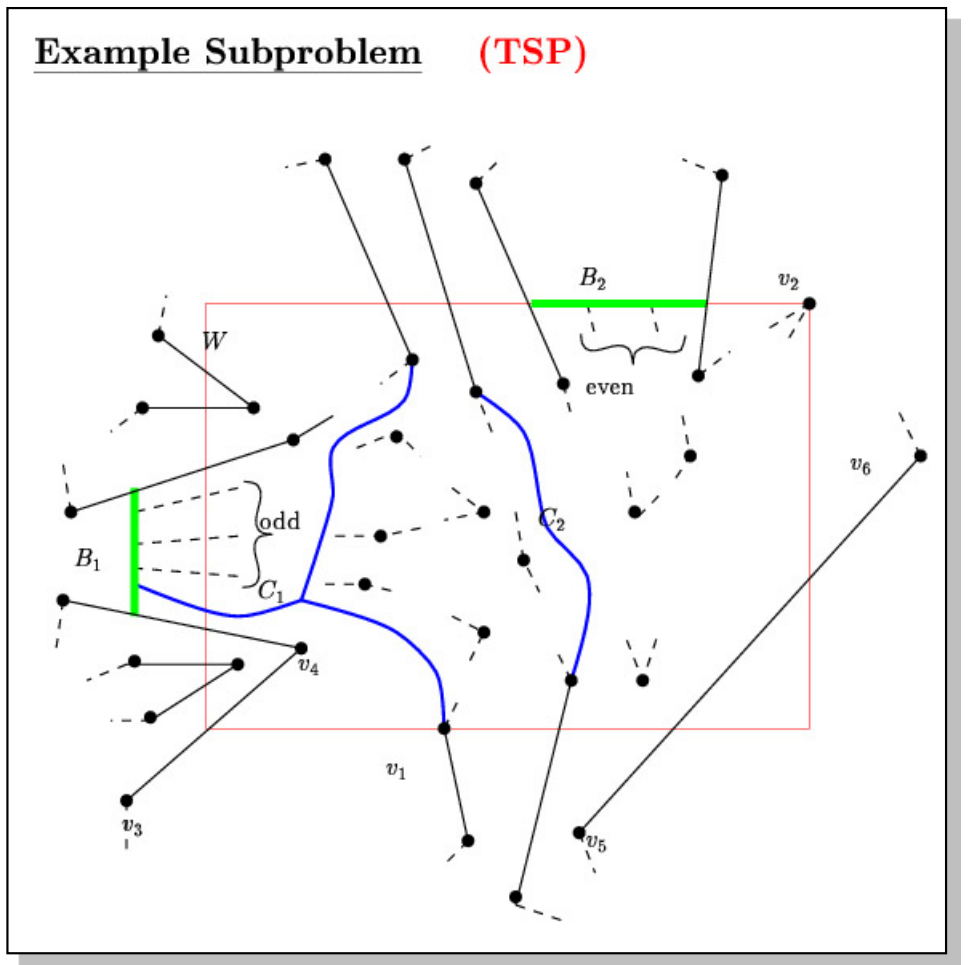
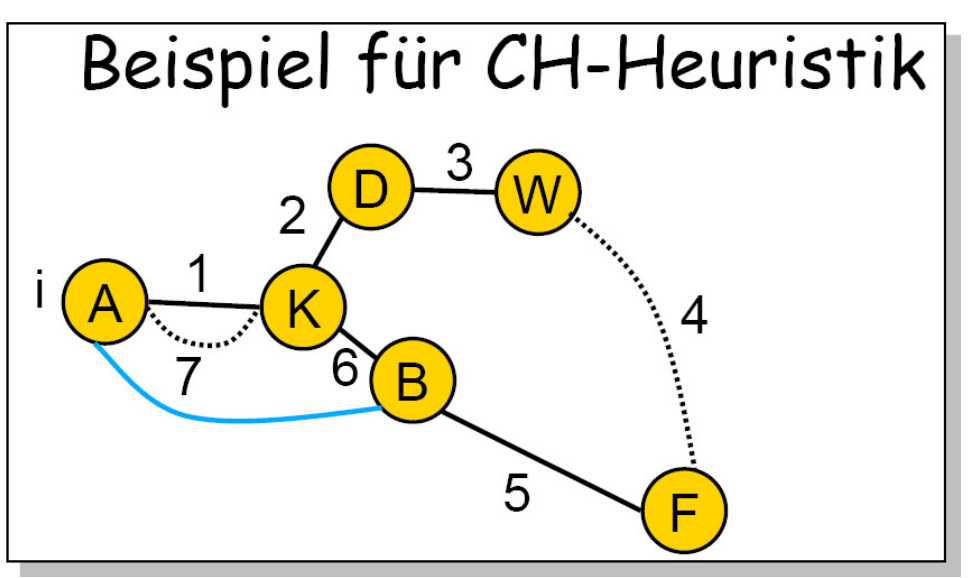


**Min**

2

## Traveling Salesman

**Min**



3  
—  
2

# Approximationsfaktoren

## 0-1 Knapsack

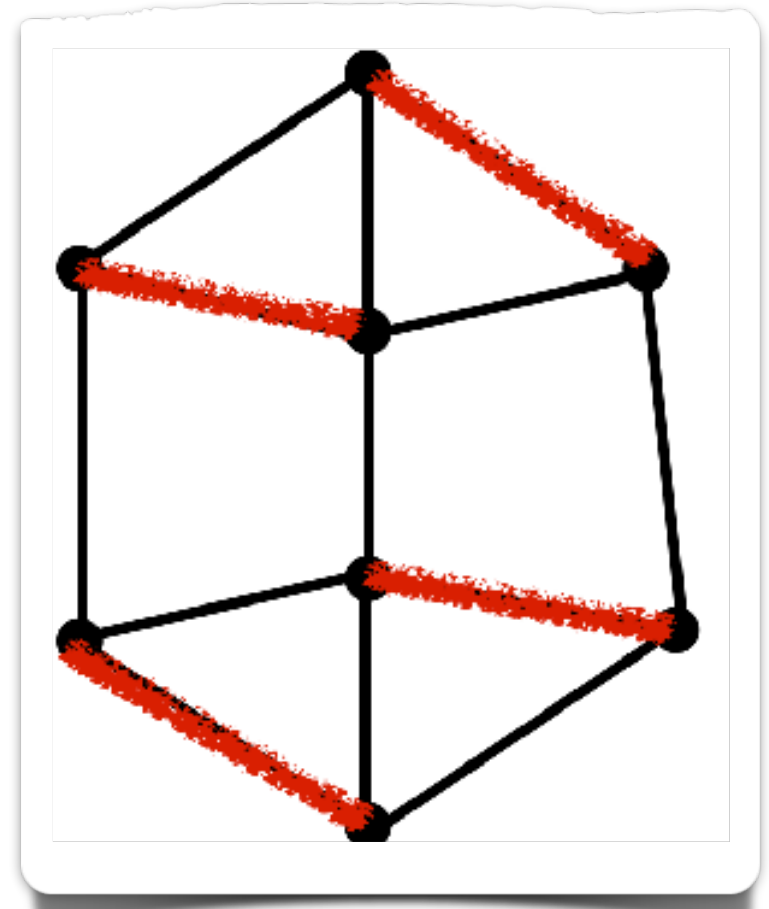
1	2	3	4	5	6
20	32	40	8	16	4
3	3	10	5	2	4
8	10	11	12	16	13
40	32	28	20	24	16
9	5	3	9	4	10
7	9	14	15	16	13
32	8	20	40	24	16
2	2	3	10	4	10

Kann Knut die Klausur bestehen?

**Max**

$$1 - \frac{1}{k + 2}$$

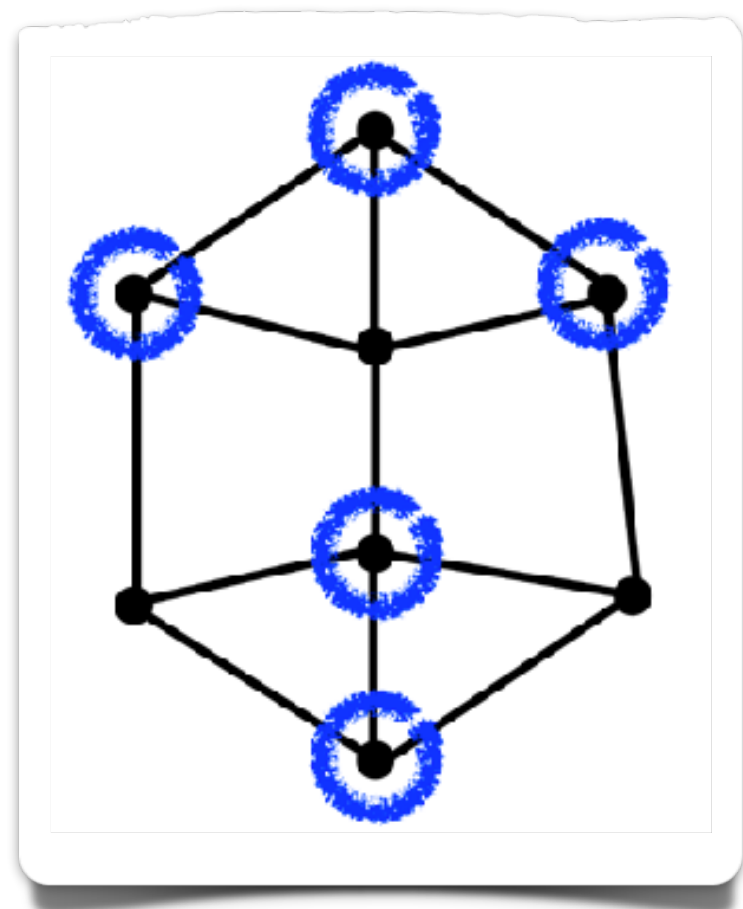
## Matching



**Max**

$$1$$

## Vertex Cover

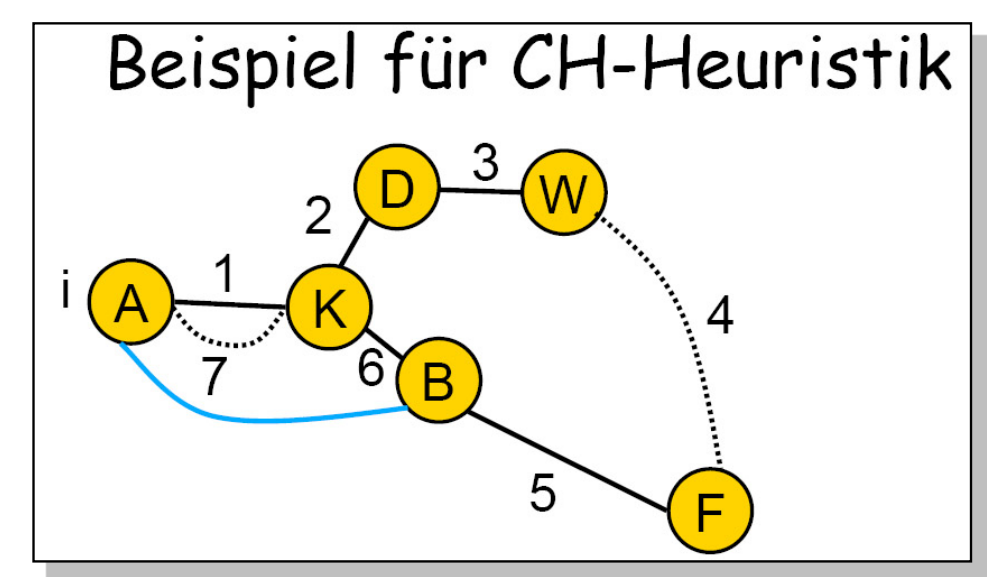


**Min**

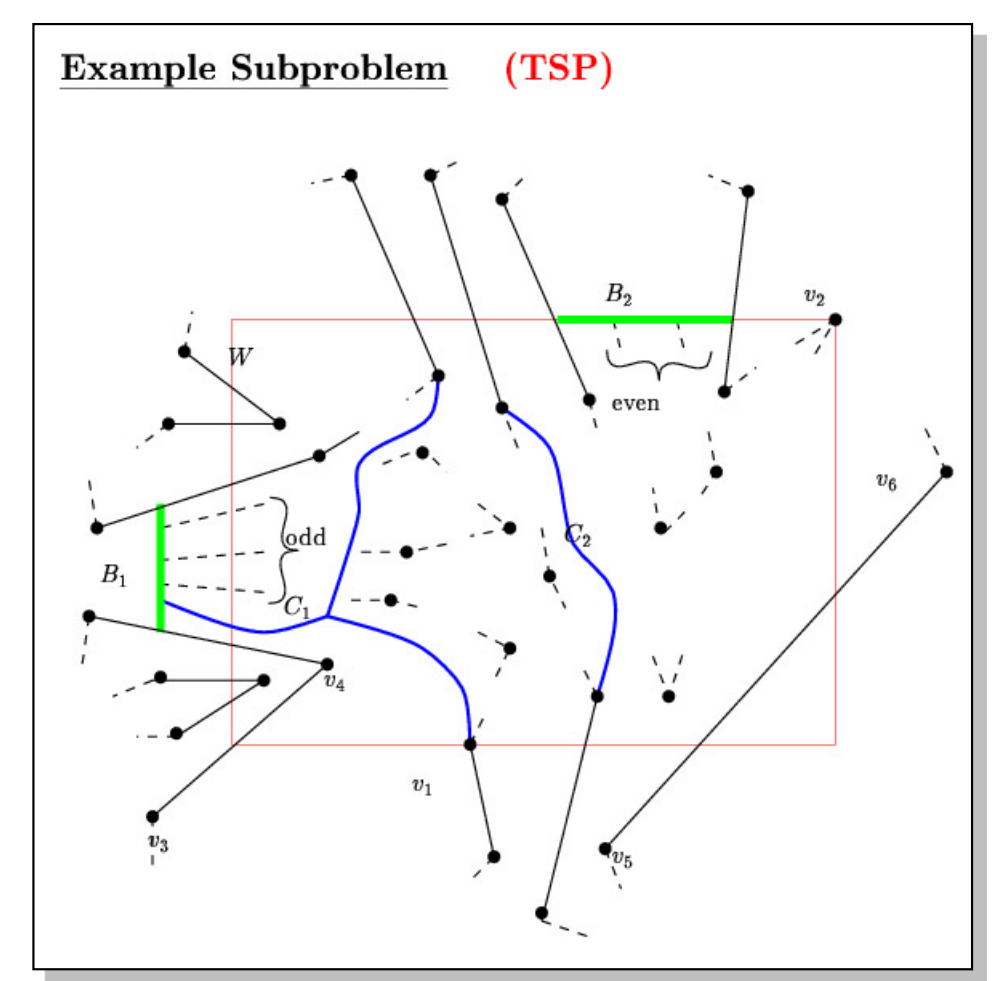
$$2$$

## Traveling Salesman

**Min**



$$\frac{3}{2}$$



$$1 + \epsilon$$

*Vielen Dank!*

*[s.fekete@tu-bs.de](mailto:s.fekete@tu-bs.de)*