Technische Universität Braunschweig

Algorithmen und Datenstrukturen – Übung #6

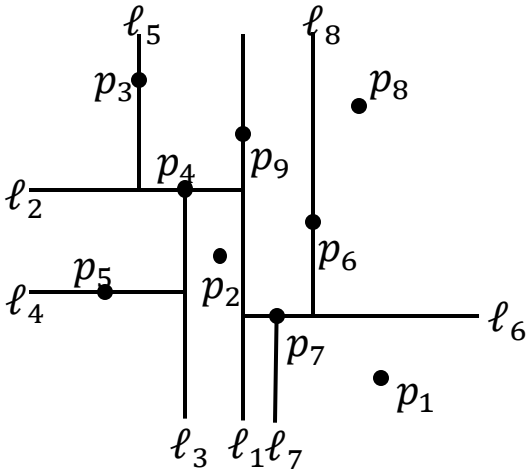
Quicksort, Mediane, kd-Bäume

Ramin Kosfeld & Chek-Manh Loi

23.01.2025

Heute

- Quicksort
- Mediane
- kd-Trees



3	5	4	8	7	13	22	1	9	15	10	6	21	14	0

3	13	10									3	1	0	
5	22	6									4	9	6	
4	1	21			Sortieren						5	13	10	
8	9	14									7	15	14	
7	15	0									8	22	21	

3	5	4	8	7	1	9	0	6	10	13	15	22	14	21

Mastertheorem

(Wir kennen es ja mittlerweile)

Satz 5.9 (Mastertheorem). *Sei* $T : \mathbb{N} \to \mathbb{R}$ *mit*

$$T(n) = \sum_{i=1}^{m} T(\alpha_i \cdot n) + \Theta(n^k), \tag{5.30}$$

wobei $\alpha_i \in \mathbb{R}$ *mit* $0 < \alpha_i < 1$, $m \in \mathbb{N}$ *und* $k \in \mathbb{R}$. *Dann gilt*

$$T(n) \in \begin{cases} \Theta(n^k) & \text{für } \sum_{i=1}^{m} \alpha_i^k < 1 \\ \Theta\left(n^k \log(n)\right) & \text{für } \sum_{i=1}^{m} \alpha_i^k = 1 \\ \Theta(n^c) & \text{mit } \sum_{i=1}^{m} \alpha_i^c = 1 \text{ für } \sum_{i=1}^{m} \alpha_i^k > 1 \end{cases} \tag{5.31}$$

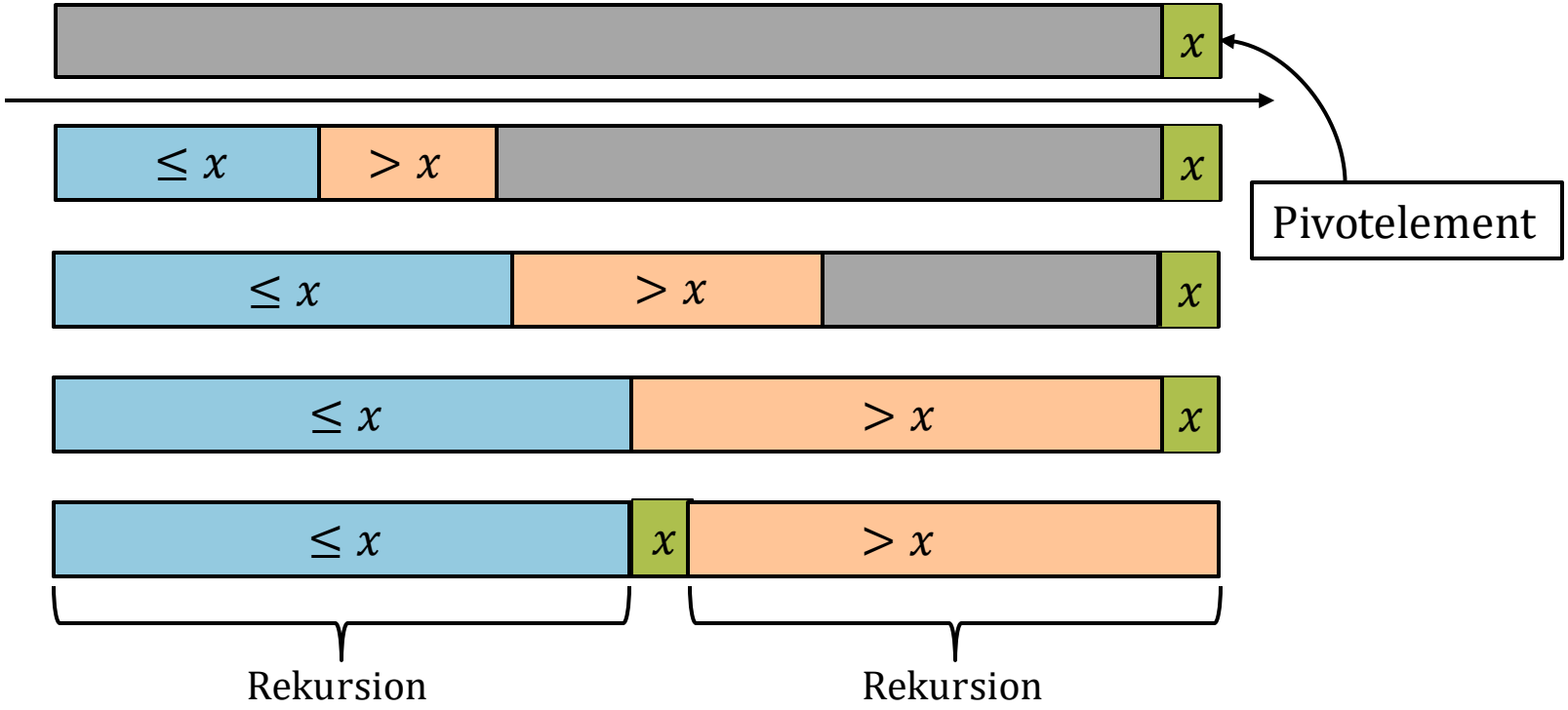
Bisherige Sortierverfahren

Algorithmus	Laufzeit	Idee

Bisherige Sortierverfahren

Algorithmus	Laufzeit	Idee
Mergesort	$O(n \log n)$	Sortiere Teilarrays und merge sie. Beginne mit kleinstem Teilarray (z.B. der Größe 1)
Bubblesort (P-Blatt 4)	$O(n^2)$	Iteriere n-mal über das Array und vertausche falsch-stehende benachbarte Objekte.
Heapsort (5. Große Übung)	$O(n \log n)$	Betrachte Array als Heap, entferne iterativ das größte Element und stelle die Heap-Eigenschaft wieder her.
Quicksort	$O(n^2)$	Pivotisiere und sortiere rekursiv auf beiden Teilarrays weiter.
Radixsort	$O\big(d(n + \mathrm{k})\big)$	Sortiere Zahlen iterativ nach d Ziffern mit Werten 1 bis k.
Selectionsort	$O(n^2)$	Setze das i-t kleinste Element in der i-ten Iteration an die i-te Stelle
Insertionsort	$O(n^2)$	Setze das i-te Element des Arrays in $A[1] \dots A[i]$ an die richtige Stelle.

Quicksort – Prinzip



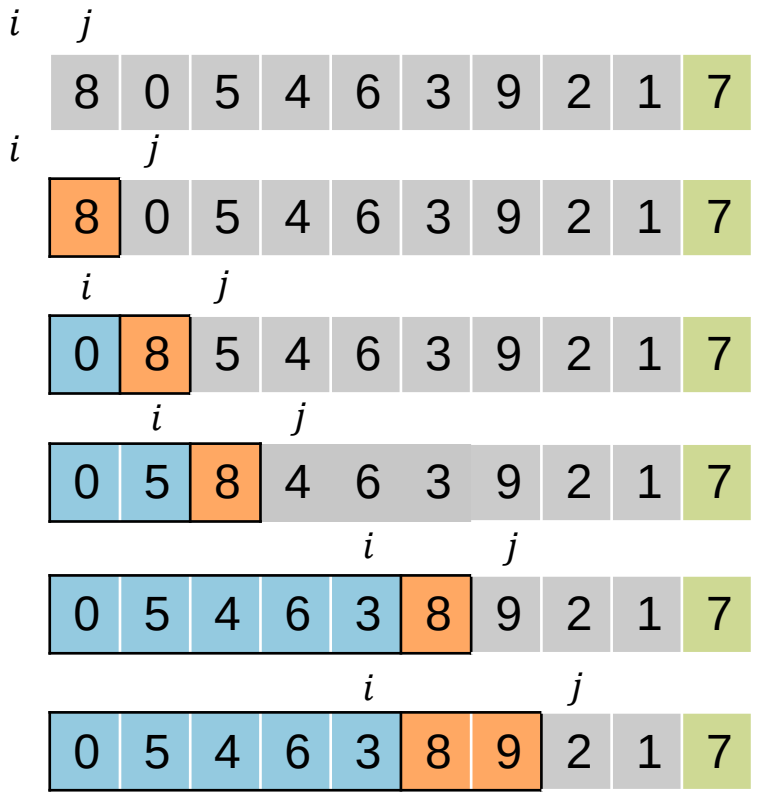
Quicksort – Partition

Pivotelement x

Letztes Element im Array

Zwei Zeiger

- i: Letzte Position mit Zahlen $\leq x$
- j: Erste Position mit nicht verglichenen Elementen



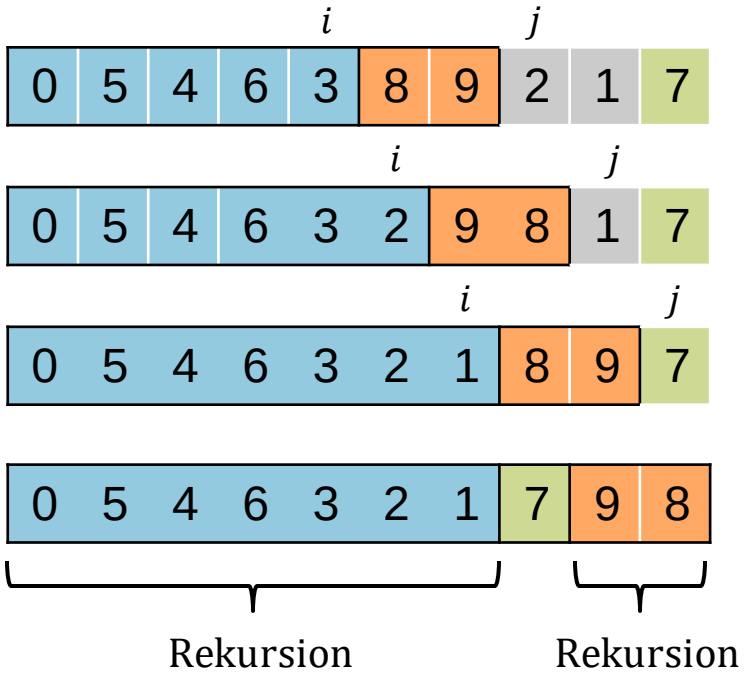
Quicksort – Partition

Pivotelement x

Letztes Element im Array

Zwei Zeiger

- i: Letzte Position mit Zahlen $< x$
- j: Erste Position mit nicht verglichenen Elementen



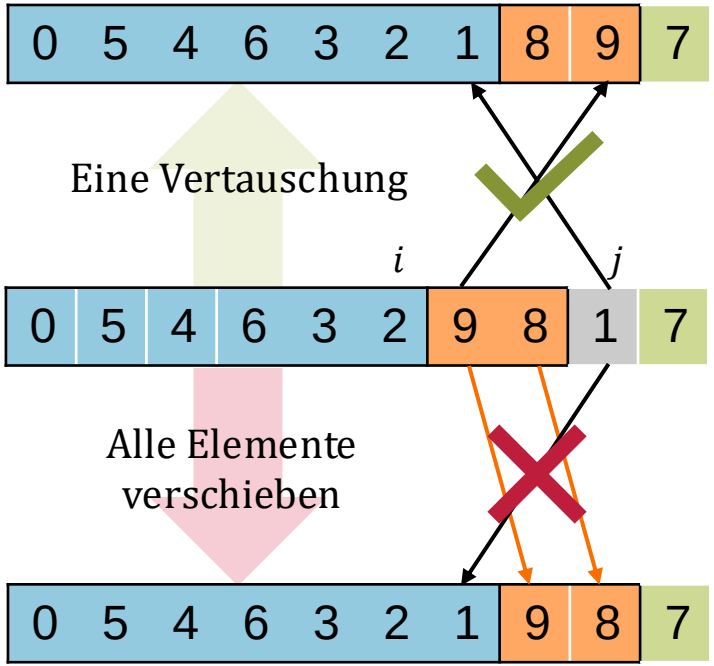
Quicksort – Details

- Innerhalb der rot und blau markierten Bereiche gibt es *keine* Sortierung! Und auch keine Garantie, dass die Elemente während des Partition-Aufrufs diese Reihenfolge behalten.
- Wir "verschieben" nicht den ganzen roten Bereich, sondern tauschen einzelne Paare, um lineare Laufzeit zu erreichen!

→ Pro Iteration reicht eine Vertauschung aus!

- Es gibt verschiedene Varianten zu pivotisieren, und daher verschiedene Versionen von Quicksort.

→ In dieser VL wird stets die hier gezeigte Variante genutzt.



Quicksort – Laufzeit

	Best-Case	Average-Case	Worst-Case
Quicksort			

Rekursionsgleichung Worst-Case:

$$T(n) =$$

Rekursionsgleichung Best-Case:

$$T(n) =$$

Quicksort – Laufzeit

	Best-Case	Average-Case	Worst-Case
Quicksort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$

Rekursionsgleichung Worst-Case:

$$T(n) = T(n-1) + \Theta(n) \Rightarrow T(n) \in \Theta\left(\sum_{i=1}^{n} i\right) = \Theta(n^2)$$

Rekursionsgleichung Best-Case:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \Rightarrow T(n) \in \Theta(n \log n)$$

Fragen?

Mediane

Mediane – Definition

Rang-k Element m in X:

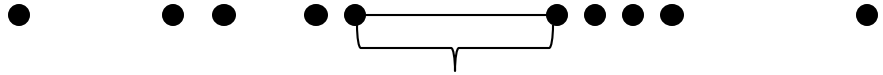
$$|\{x \in X : x \leq m\}| \geq k$$

und $$|\{x \in X : x \geq m\}| \geq n - k + 1$$

Für einen **Median** m in X gilt:

$$|\{x \in X : x < m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$

und $$|\{x \in X : x > m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$



Jeder Punkt in diesem Bereich ist ein Median!

Bei $X = \{1,2,3,4,5,6,7,8\}$ sind sowohl 4 als auch 5 ein Median.

Median vs Durchschnitt

Rang-k Element m in X:

$$|\{x \in X : x \leq m\}| \geq k$$

und $$|\{x \in X : x \geq m\}| \geq n - k + 1$$

Für einen **Median** m in X gilt:

$$|\{x \in X : x < m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$

und $$|\{x \in X : x > m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$

Bei $X = \{1,3,5,7,9\}$ ist die 5 der Median und 5 der Durchschnitt.

Bei $X = \{1,3,5,7,984\}$ ist die 5 der Median und 200 der Durchschnitt.

Der Median m minimiert

$$\sum_{x \in X} |x - m|$$

Der Durchschnitt D minimiert

$$\sum_{x \in X} (x - D)^2$$

Mediane – Algorithmus (I)

Naive Idee

Sortieren und das Element an der k-ten Stelle ausgeben

Laufzeit: $\Theta(n \log n)$

Geht das besser?

Nutze das Prinzip von Quicksort

Pivotisiere und arbeite rekursiv auf **einem** Teil weiter

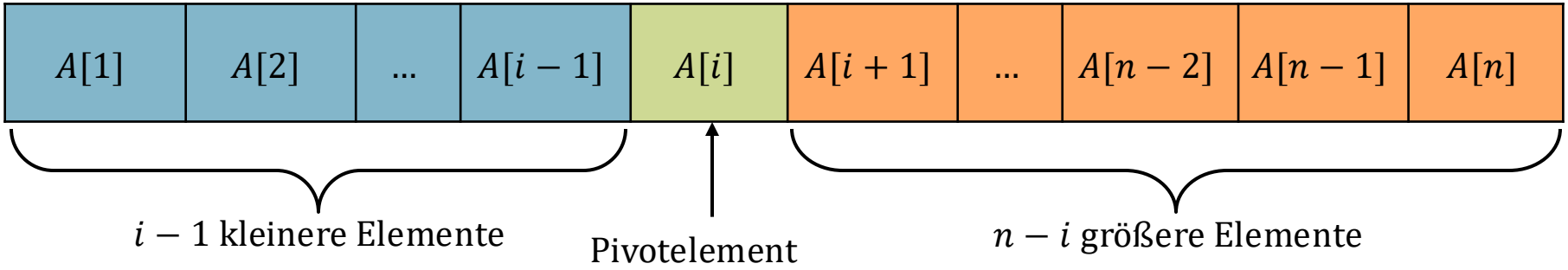
Dazu stellen sich die Fragen:

1. Auf welchem Teilarray geht es weiter?
2. Ist das schneller als $\Theta(n \log n)$?

Mediane – Algorithmus (II)

1. Auf welchem Teilarray geht es weiter?

Nach Pivotisierung:



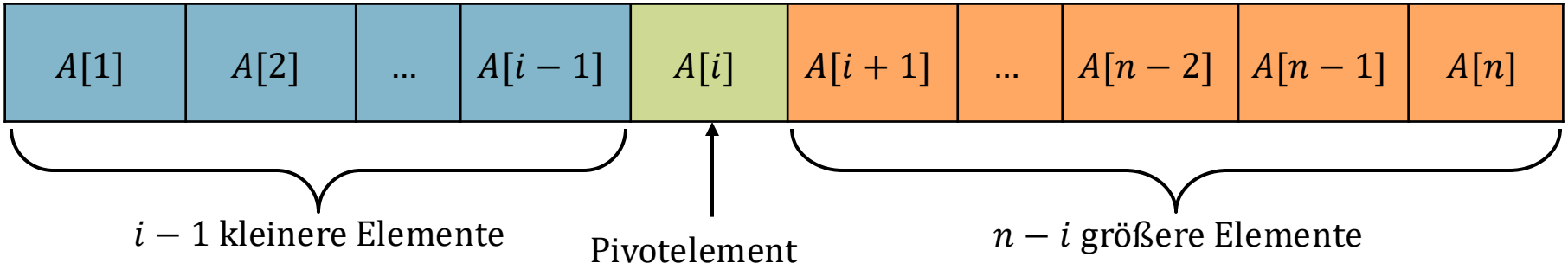
3 Fälle:

1. Falls $k < i$,
2. Falls $k = i$,
3. Falls $k > i$,

Mediane – Algorithmus (II)

1. Auf welchem Teilarray geht es weiter?

Nach Pivotisierung:



3 Fälle:

1. Falls $k < i$, suche im linken Teilarray nach dem k-ten Element.
2. Falls $k = i$, dann haben wir das k-te Element gefunden!
3. Falls $k > i$, suche im rechten Teilarray nach dem $(k - i)$-ten Element.

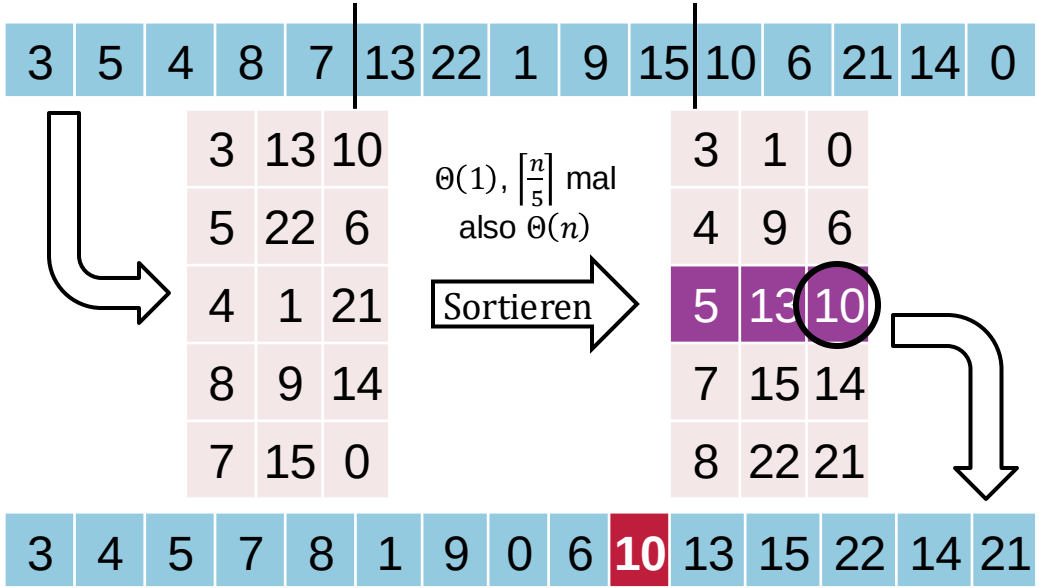
Mediane – Algorithmus (III)

2. Ist das schneller als $\Theta(n \log n)$?

Wie bei Quicksort kann größeres Teilarray $n - 1$ Elemente enthalten. Dadurch Laufzeit $\Omega(n^2)$

Idee: Wähle besseres Pivotelement

1. Teile Array in 5er Gruppen
2. Bestimme Median in jeder Gruppe
3. Bestimme Median der Mediane m
4. Benutze m als Pivotelement



Mediane – Analyse

Wie viele Zahlen gibt es, die größer/kleiner als m sind?

Sortiere die t 5er Gruppen **gedanklich** nach deren Median

Jeweils aufsteigend sortiert

$\leq m$	$\leq m$	$\leq m$	$\leq m$	m	$\geq m$	$\geq m$	$\geq m$	$\geq m$	$\geq m$

Mediane – Analyse

Wie viele Zahlen gibt es, die größer/kleiner als m sind?

Sortiere die t 5er Gruppen **gedanklich** nach deren Median

Jeweils aufsteigend sortiert

$\leq m$	$\leq m$	$\leq m$	$\leq m$	$\leq m$					
$\leq m$	$\leq m$	$\leq m$	$\leq m$	$\leq m$					
$\leq m$	$\leq m$	$\leq m$	$\leq m$	m	$\geq m$	$\geq m$	$\geq m$	$\geq m$	$\geq m$

Der rote Bereich enthält nur Elemente, die höchstens m sind.

Mediane – Analyse

Jeweils aufsteigend sortiert

$\leq m$	$\leq m$	$\leq m$	$\leq m$	$\leq m$					
$\leq m$	$\leq m$	$\leq m$	$\leq m$	$\leq m$					
$\leq m$	$\leq m$	$\leq m$	$\leq m$	m	$\geq m$	$\geq m$	$\geq m$	$\geq m$	$\geq m$

Der rote Bereich enthält nur Elemente, die höchstens m sind. Wie viele sind das?

Bei t Gruppen ist der Median m in der $\left\lceil \frac{t}{2} \right\rceil$-ten Gruppe.

Bei 5er Gruppen sind pro Gruppe mindestens 3 Elemente $\leq m$. (wenn der Median $\leq m$)

Entsprechend gibt es mindestens $3 \cdot \left\lceil \frac{t}{2} \right\rceil$ viele Elemente $\leq m$.

Damit gibt es maximal $n - 3 \cdot \left\lceil \frac{t}{2} \right\rceil \leq n - \frac{3}{2}t \leq n - \frac{3}{2} \cdot \frac{n}{5} = \frac{7}{10}n$ Elemente größer als m.

Analog: Maximal $\frac{7}{10}n$ Elemente kleiner als m.

Mediane – Laufzeit

Wir haben also als Laufzeit:

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)$$

Bestimmung Median der Mediane

Rekursion in linke/rechte Teilhälfte

Aufteilen in Gruppen / Sortieren je in $O(1)$ / Pivotisieren / …

Mediane - Laufzeit

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)$$

Mastertheorem

$$m = 2, \qquad k = 1, \qquad \alpha_1 = \frac{1}{5}, \alpha_2 = \frac{7}{10}$$

$$\sum_{i=1}^{2} \alpha_i^1 = \left(\frac{1}{5}\right)^1 + \left(\frac{7}{10}\right)^1$$

Mediane - Laufzeit

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)$$

Mastertheorem

$$m = 2, \qquad k = 1, \qquad \alpha_1 = \frac{1}{5}, \alpha_2 = \frac{7}{10}$$

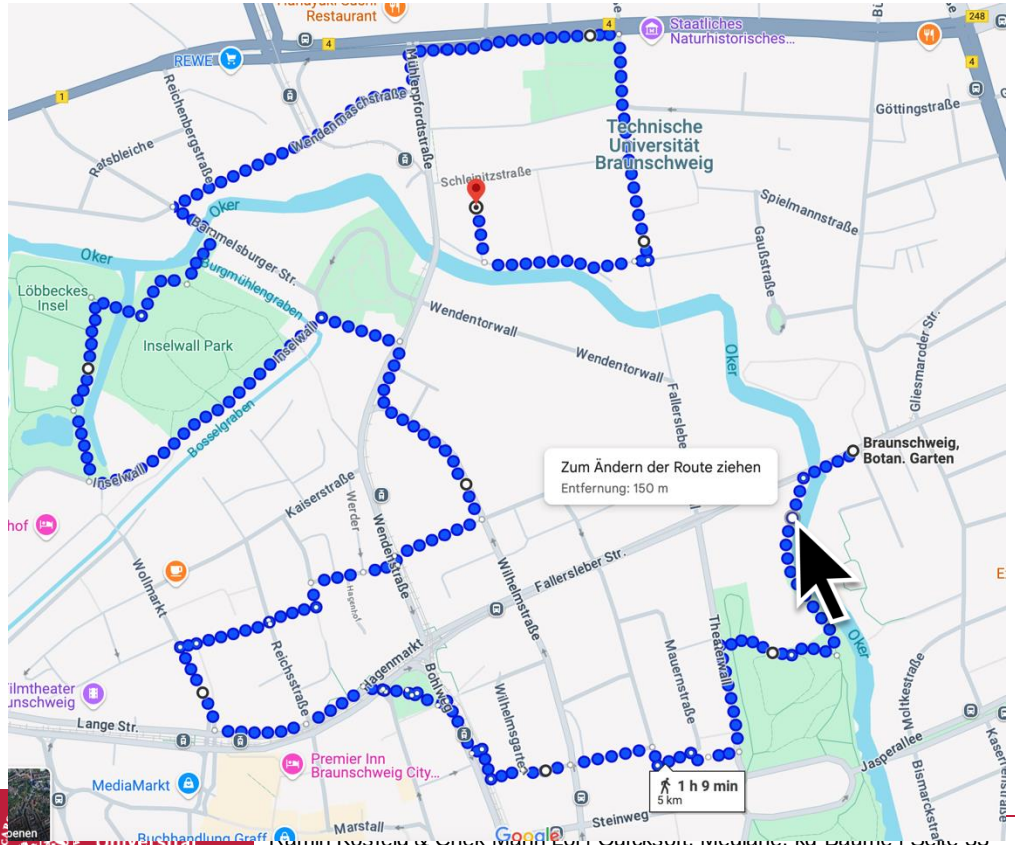
$$\sum_{i=1}^{2} \alpha_i^1 = \left(\frac{1}{5}\right)^1 + \left(\frac{7}{10}\right)^1 = \frac{2}{10} + \frac{7}{10} = \frac{9}{10} < 1 \quad \rightarrow \text{ Fall 1}$$

$$\Longrightarrow T(n) \in \Theta(n)$$

Fragen?

kd-Bäume

Motivation



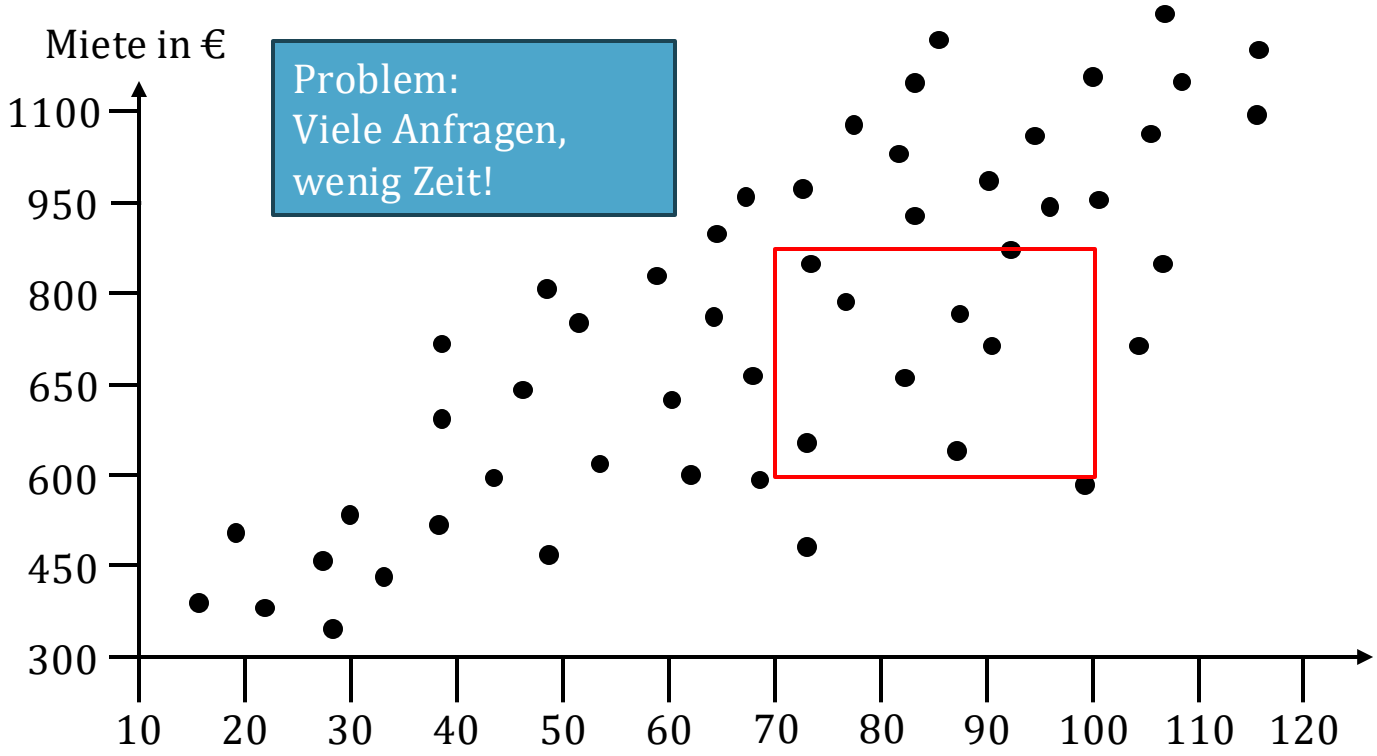
Das sind ja ganz schön viele Punkte ...

Wie genau finden die den nächsten Punkt zum Cursor?

Iterieren die wirklich einfach über die Punktmenge, oder machen die was Schlaueres?

Disclaimer: Ich habe keine Ahnung, was da tatsächlich passiert, so viele Punkte sinds ja doch nicht.

Motivation



Problem:
Viele Anfragen,
wenig Zeit!

Suche Wohnung:
Maximal 875€
Mindestens 600€
Maximal 100m²
Mindestens 70m²

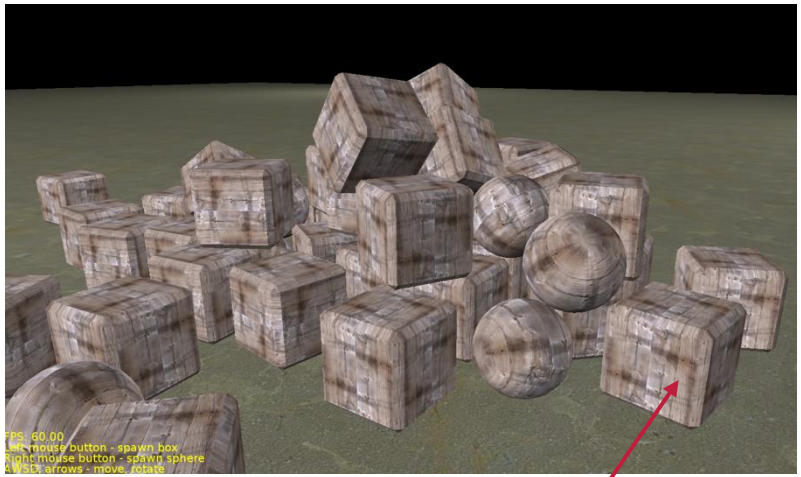
Gib alle Wohnungen
im roten Bereich aus!

Motivation

Arbeit mit Punktwolken, z.B. für Laserscans etc.

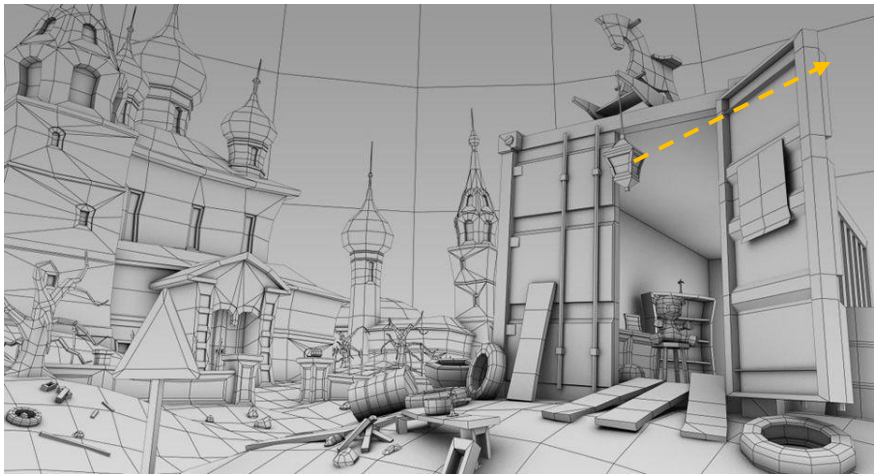
Motivation

Aufbauend auf solchen Grundideen kommt man dann zu Problemen wie …



Kollisionserkennung

z.B.: Muss ich *wirklich* prüfen, ob diese Kiste mit allen Kisten ganz hinten links kollidiert?



Ray Tracing

z.B.: Wenn ein Lichtstrahl hier nach vorne verläuft, muss ich dann echt für alle Dreiecke im hinteren Teil der Szene prüfen, ob wir den Strahl schneiden?

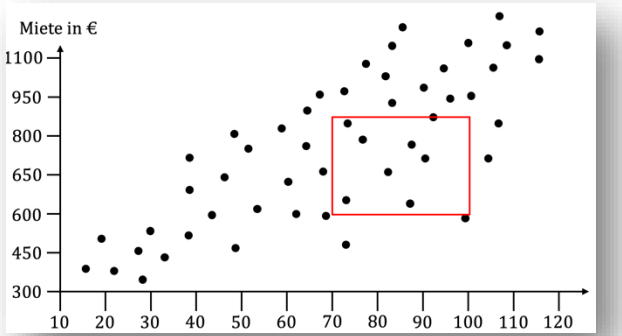
…und mehr!

Motivation

Wir stolpern immer wieder über Probleme der Art …

- Beantworte *schnell* Anfragen auf einer großen Menge von Punkten im Raum (2D, 3D, …)
  - Welcher Punkt in der Menge ist am nächsten zu einem gegebenen Punkt?
  - Finde *alle* Punkte in einem rechteckigen Bereich im Raum!

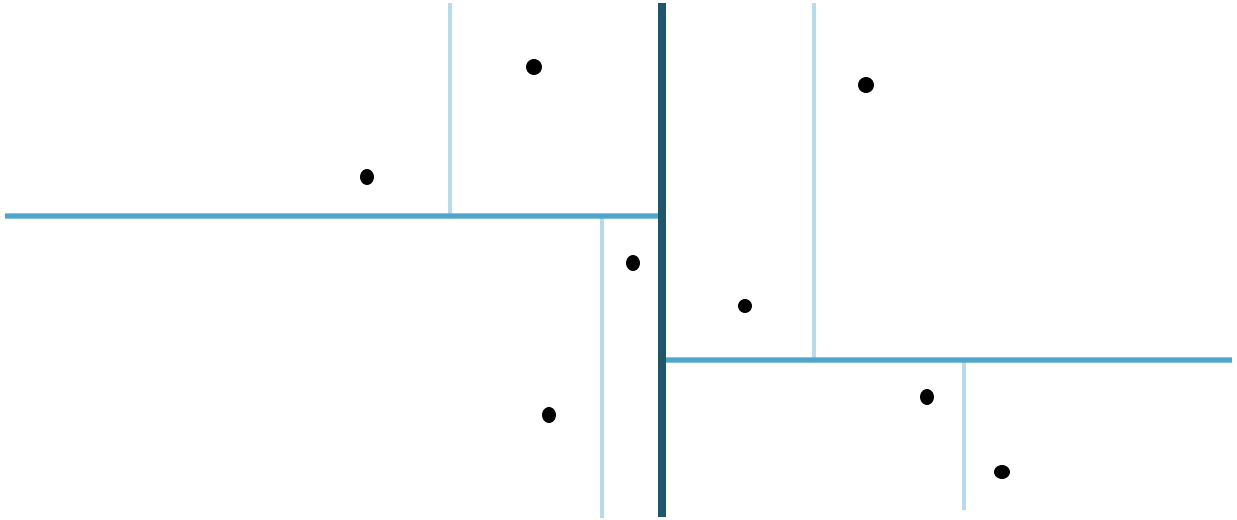
Für beide Probleme bietet sich die Nutzung von kd-Bäumen an.
Wir kümmern uns hier um die zweite Fragestellung.



Ideen?

Idee

- Teile den Raum immer wieder entlang einer Achse auf
- Konstruiere somit einen binären Suchbaum
- Wechsele die Achse nach jeder Unterteilung
- Teile den Raum entlang des Medians aller enthaltenen Punkte für gute Aufteilung

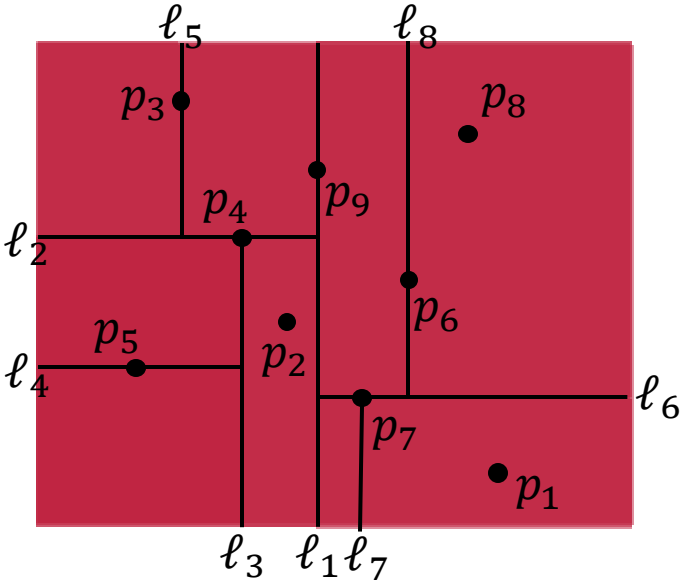


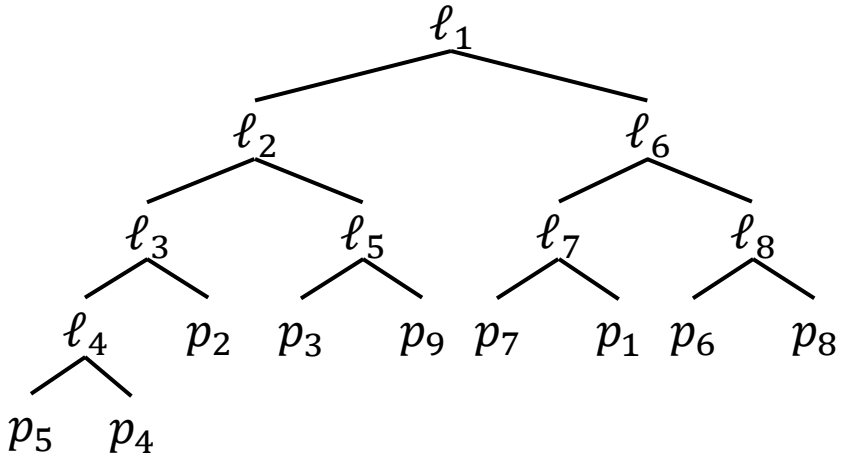
kd-Bäume – Konstruktion/Preprocessing

Algorithmus BUILDKDTREE
Eingabe: Punktmenge P, Rekursionstiefe *depth*
Ausgabe: Wurzel eines k-d-Baums

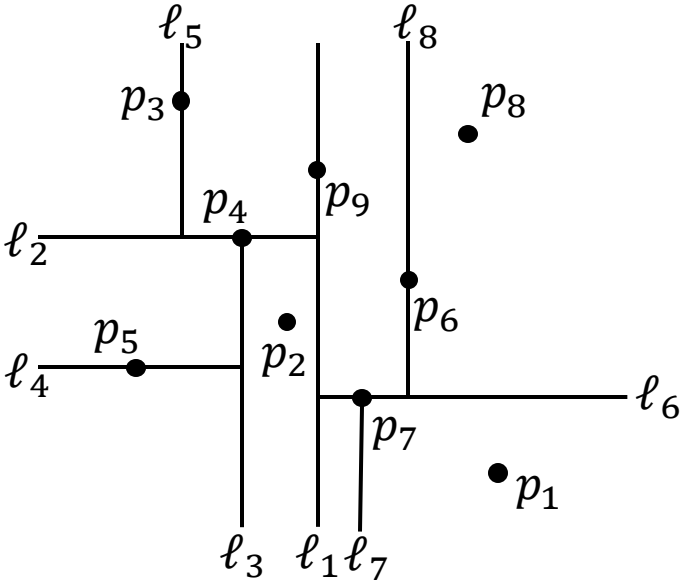
1. **if** ($|P| = 1$)
2. **return** Blatt mit diesem Punkt
3. **else if** (*depth* ist gerade)
4. Teile in zwei Teilmengen P_1 ($\leq \ell$) und P_2 ($> \ell$) an vertikaler Median-Linie ℓ
5. **else**
6. Teile in zwei Teilmengen P_1 ($\leq \ell$) und P_2 ($> \ell$) an horizontaler Median-Linie ℓ
7. Setze v_{left} := BUILDKDTREE(P_1,*depth+1)*
8. Setze v_{right} := BUILDKDTREE(P_2,*depth+1)*
9. Erzeuge Knoten v für ℓ mit v_{left} und v_{right} als Kinderknoten
10. **return** v

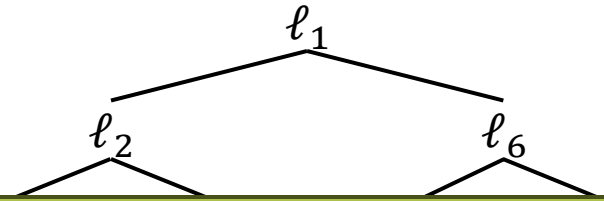
kd-Bäume – Beispiel





kd-Bäume – Beispiel





Laufzeit des Algorithmus ist $O(n \log n)$

Beweis

(1) Median-Linien können in $O(n)$ Zeit gefunden werden.

(2) Rekursionsgleichung für die Laufzeit ist also

$$T(n) = O(n) + 2T\left(\left\lceil \frac{n}{2} \right\rceil\right)$$

Nach Mastertheorem ist das $O(n \log n)$.

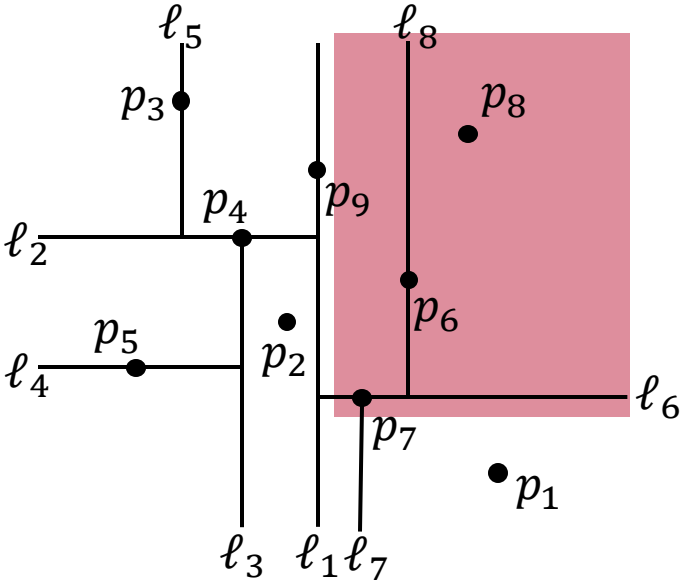
kd-Bäume – Search Query

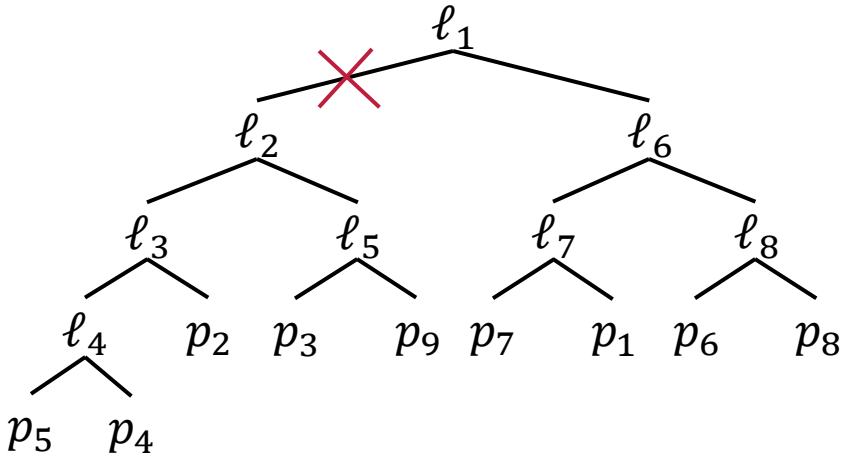
Algorithmus SEARCHKDTREE
Eingabe: Wurzel v eines (Teil-)baums, Rechteck R
Ausgabe: Knoten unterhalb v, die in R liegen

```
1.  if (v ist ein Blatt)
2.      Gib v aus, falls v in R
3.  else
4.      if (Region(l(v)) ganz in R)
5.          REPORTSUBTREE(l(v))
6.      else if (Region(l(v)) schneidet R)
7.          SEARCHKDTREE(l(v), R)
8.      if (Region(r(v)) ganz in R)
9.          REPORTSUBTREE(r(v))
10.     else if (Region(r(v)) schneidet R)
11.         SEARCHKDTREE(r(v), R)
```

REPORTSUBTREE(v)
gibt alle Punkte im gegebenen v Teilbaum aus

kd-Bäume – Beispiel





Welche Punkte liegen im Rechteck?

Antwort: p_7 p_6 p_8

kd-Bäume – Search Query

Laufzeit ist $O(\sqrt{n}+x)$, wobei x die Anzahl an ausgegebenen Elementen ist.

Das nennt sich *output-sensitiv*, d.h. die Laufzeit ist von der Größe des Outputs abhängig.

Beweisidee:

Wie viele *geschnittene* Regionen müssen betrachtet werden? Man kann die Rekursionsgleichung aufstellen:

$$Q(n) = 2 + 2Q\left(\frac{n}{4}\right).$$

Also $Q(n) \in O(\sqrt{n})$. So viele Elemente können geprüft werden, die nicht in R liegen. Die $O(x)$ sind die Elemente in R.

Algorithmus SEARCHKDTREE
Eingabe: Wurzel v eines (Teil-)baums, Rechteck R
Ausgabe: Knoten unterhalb v, die in R liegen

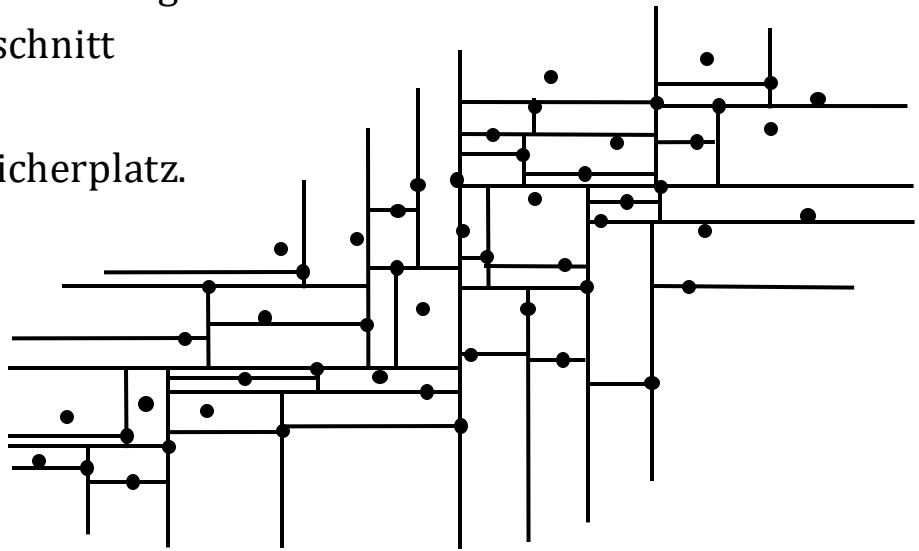
```
1.  if (v ist ein Blatt)
2.     Gib v aus, falls v in R
3.  else
4.     if (Region(l(v)) ganz in R)
5.        REPORTSUBTREE(l(v))
6.     else if (Region(l(v)) schneidet R)
7.        SEARCHKDTREE(l(v), R)
8.     if (Region(r(v)) ganz in R)
9.        REPORTSUBTREE(r(v))
10.    else if (Region(r(v)) schneidet R)
11.       SEARCHKDTREE(r(v), R)
```

kd-Bäume – Höhere Dimensionen

Bisher nur 1D und 2D betrachtet. Laufzeiten für $k \geq 2$ Dimensionen:

- BuildKDTree: $O(n \log n)$
- SearchKDTree: $O(n^{1-\frac{1}{k}} + x)$, um x Elemente auszugeben
- FindNearestNeighbor: $O(\log n)$ im Durchschnitt

Weiterer Vorteil: Sie benötigen nur $O(n)$ Speicherplatz.



Fragen?

Klausur

Klausur

Eckdaten

- 12.02.2025 von 8:00 Uhr bis 10:00
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)

Vorbereitung

- Alte Klausuren
- Hausaufgaben
- Tutorien (Tutoren fragen)
- Skript

… detailliertere Infos & Fragestunde rund um die Klausur:

GÜ nächste Woche :)

Gleich hier vorne zum Mitnehmen:

✨ Originale Altklausuren aus den letzten Jahren ✨

Nächstes Mal

Wiederholung und Fragestunde

… schon nächste Woche!