



Technische  
Universität  
Braunschweig



# Algorithmen und Datenstrukturen – Übung #6

Quicksort, Mediane, kd-Bäume

Arne Schmidt

06.02.20

# Euler Graph



The term "Euler graph" is sometimes used to denote a graph for which all vertices are of even degree (e.g., Seshu and Reed 1961)

## Eulerkreisproblem

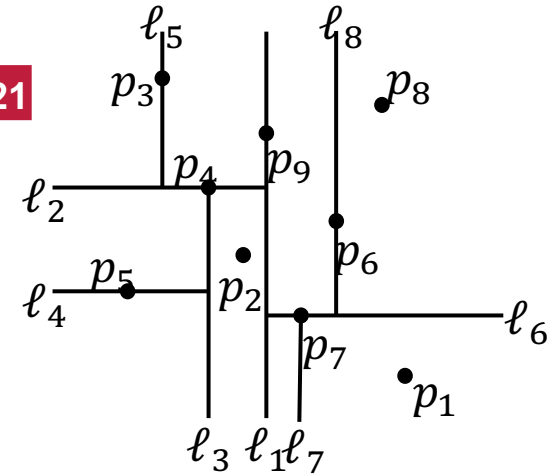
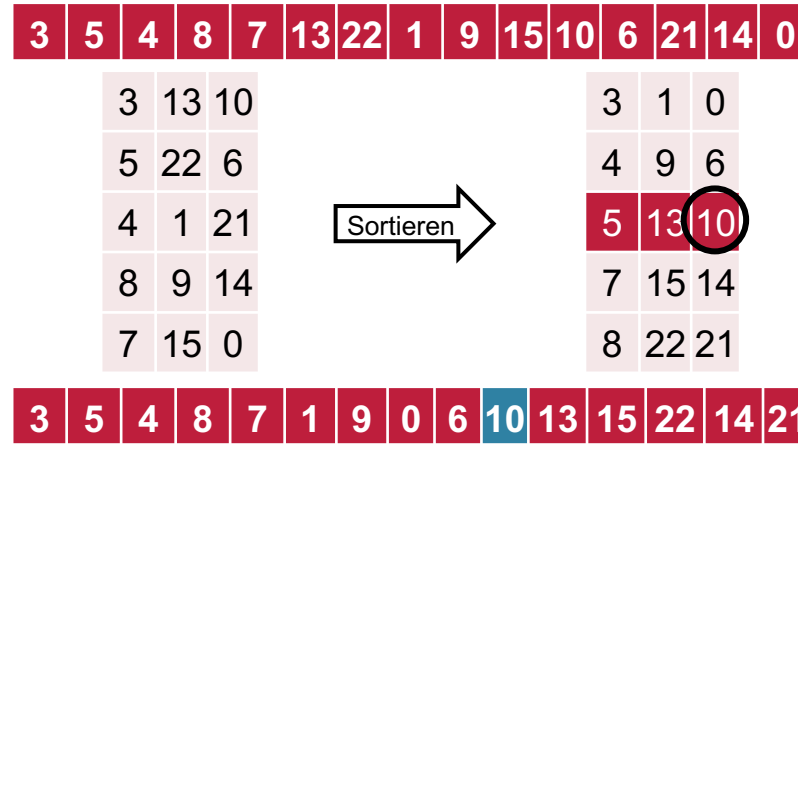
Ein **Eulerkreis** oder (geschlossener) **Eulerzug** (auch **Eulertour** oder **Eulersche Linie**

Ein *offener Eulerzug* (*Eulerpfad* oder auch *Eulerweg*) ist gegeben, wenn die Identität  $v$  welche jede Kante des Graphen genau einmal enthält.

Ein zusammenhängender Graph, der einen *Eulerkreis* besitzt, heißt *eulerscher Graph*

# Heute

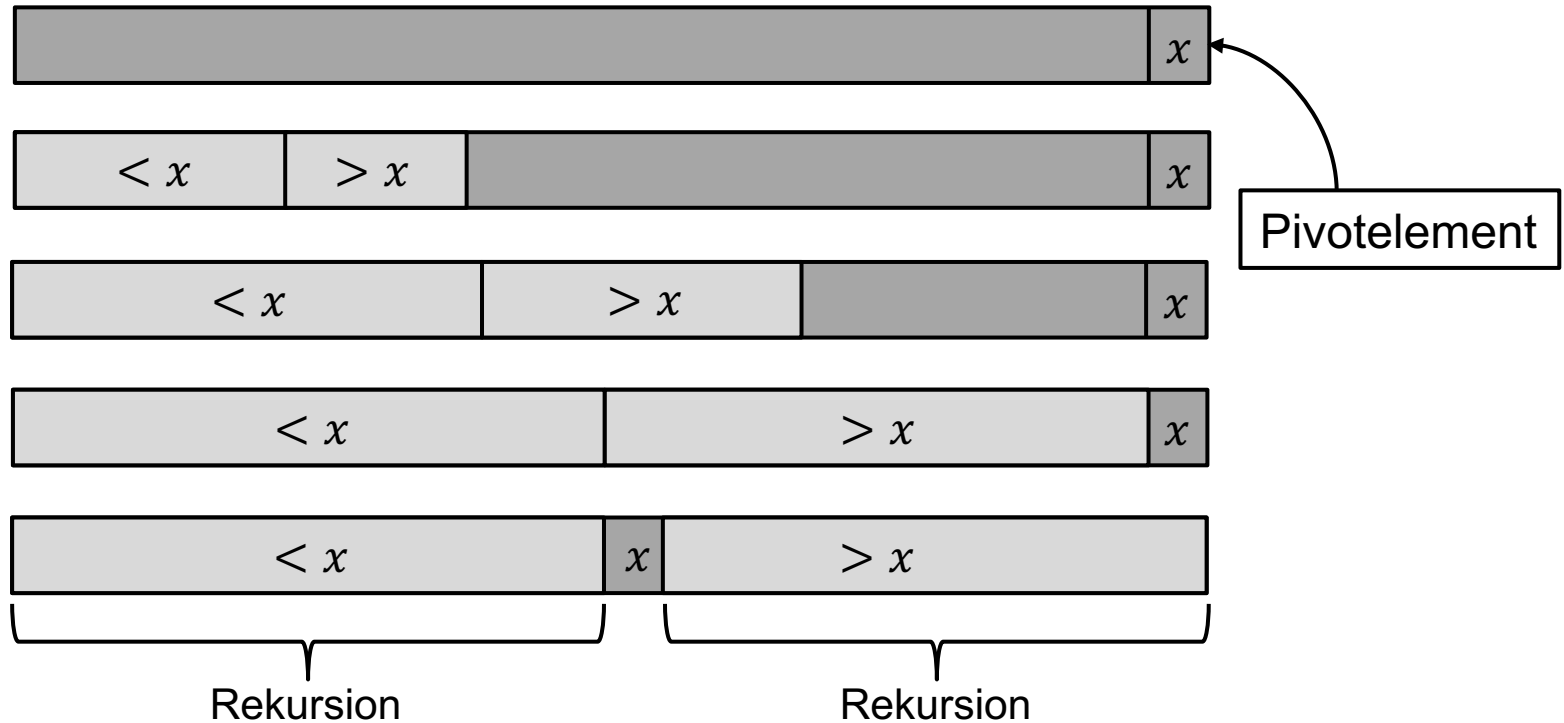
- Quicksort
- Mediane
  - Algorithmus
  - Laufzeit
- kd-Trees



# Bisherige Sortierverfahren

Algorithmus	Laufzeit	Idee
Mergesort	$O(n \log n)$	Sortiere Teilarrays und merge sie. Beginne mit kleinstem Teilarray (z.B. der Größe 1)
Quicksort	$O(n^2)$	Pivotisiere und sortiere rekursiv auf beiden Teilarrays weiter.
Selection (Notizen U5)	$O(n^2)$	Setze das $i$ -t kleinste Element in der $i$ -ten Iteration an die $i$ -te Stelle
Insertion (Notizen U5)	$O(n^2)$	Setze das $i$ -te Element des Arrays in $A[1] \dots A[i]$ an die richtige Stelle.
Radixsort	$O(nd)$	Sortiere Zahlen iterativ nach den $d$ Ziffern.
Bubblesort (Blatt 4)	$O(n^2)$	Iteriere $n$ -mal über das Array und vertausche falschstehende benachbarte Objekte.

# Quicksort – Prinzip



# Quicksort – Partition

8	0	5	4	6	3	9	2	1	7
---	---	---	---	---	---	---	---	---	---

# Quicksort – Partition

8	0	5	4	6	3	9	2	1	7
---	---	---	---	---	---	---	---	---	---

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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

# Quicksort – Partition

$i$	$j$										
		8	0	5	4	6	3	9	2	1	7

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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



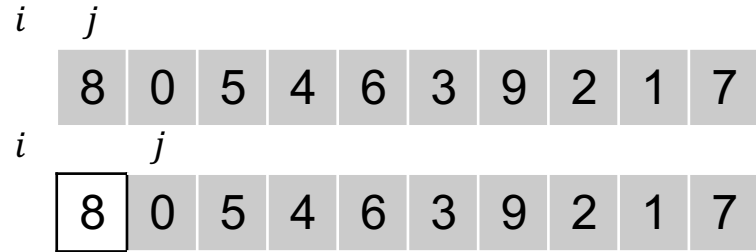
# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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



# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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



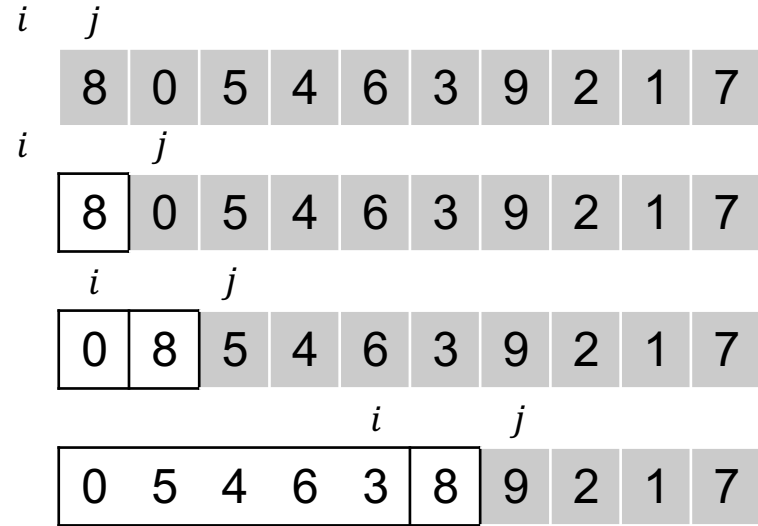
# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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



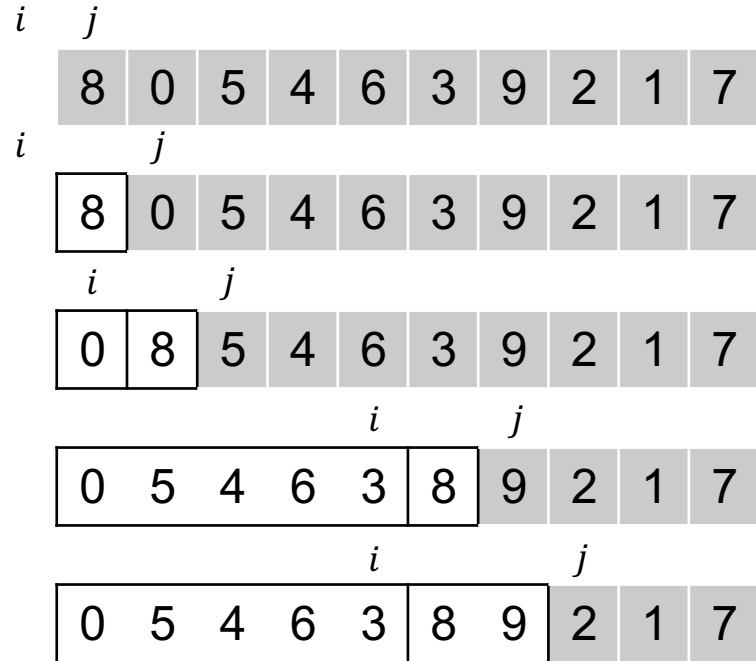
# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

- $i$ : Letzte Position mit Zahlen  $< x$
- $j$ : 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

$i$					$j$				
0	5	4	6	3	8	9	2	1	7

# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

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

$i$					$j$				
0	5	4	6	3	8	9	2	1	7
$i$						$j$			
0	5	4	6	3	2	9	8	1	7

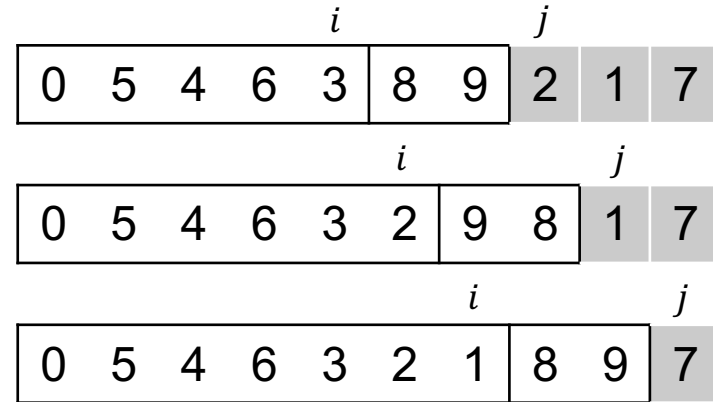
# Quicksort – Partition

Pivotelement  $x$ :

Letztes Element im Array

Zwei Zeiger:

- $i$ : Letzte Position mit Zahlen  $< x$
- $j$ : 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

$i$					$j$				
0	5	4	6	3	8	9	2	1	7

$i$							$j$		
0	5	4	6	3	2	9	8	1	7

$i$							$j$		
0	5	4	6	3	2	1	8	9	7

0	5	4	6	3	2	1	7	9	8
---	---	---	---	---	---	---	---	---	---



# 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:

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

# Mediane

# Mediane – Definition

Rang- $k$  Element  $m$  in  $X$ :

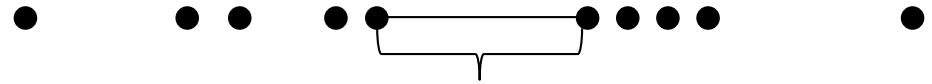
$$|\{x \in X: x \leq m\}| \geq k$$

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

Für einen Median  $m$  in  $X$  gilt:

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

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



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.

The More You Know

Der Median  $m$  minimiert

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

Der Durchschnitt  $D$  minimiert

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

<https://vignette.wikia.nocookie.net/dragonball/images/4/4b/Family-guy-the-more-you-know.jpg/revision/latest?cb=20140510200230>

# Mediane – Algorithmus (I)

Naive Idee: Sortieren und das Element an der  $k$ -te Stelle ausgeben.

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

Das geht besser!

Nutze die Idee von Quicksort: Pivotisiere und arbeite rekursiv auf **einem** Teil weiter.

Dazu stellen sich die Fragen:

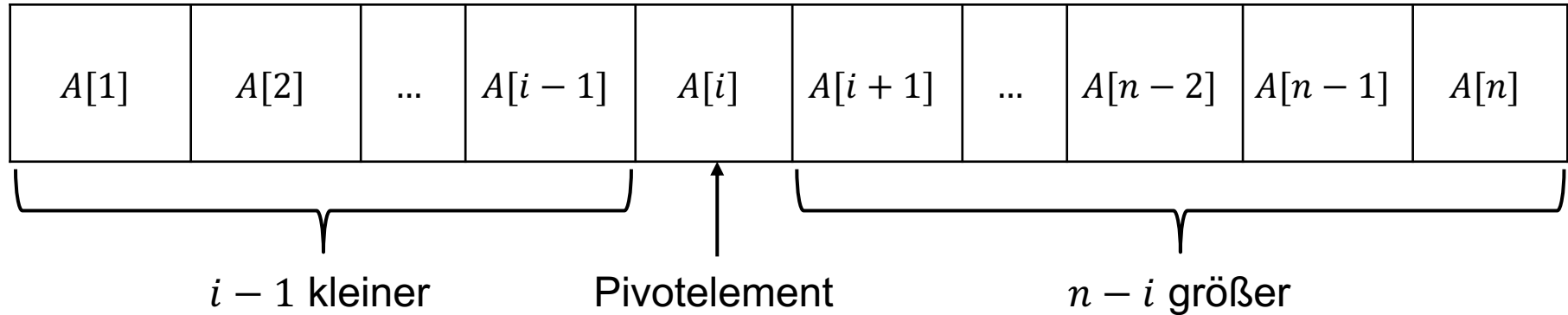
1. Auf welchem Teilarray geht es weiter?
2. Ist das schneller als  $\Theta(n \log n)$ ?

Schauen wir uns die Fragen genauer an!

# 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  $\Omega(n^2)$

Andere Idee:

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

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

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



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

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

# Mediane – Analyse

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

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

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

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

Median  $m$  ist in der  $\left\lfloor \frac{t}{2} \right\rfloor$ -ten Gruppe. Also sind mindestens  $3 \cdot \left\lfloor \frac{t}{2} \right\rfloor$  viele Elemente  $\leq m$ .

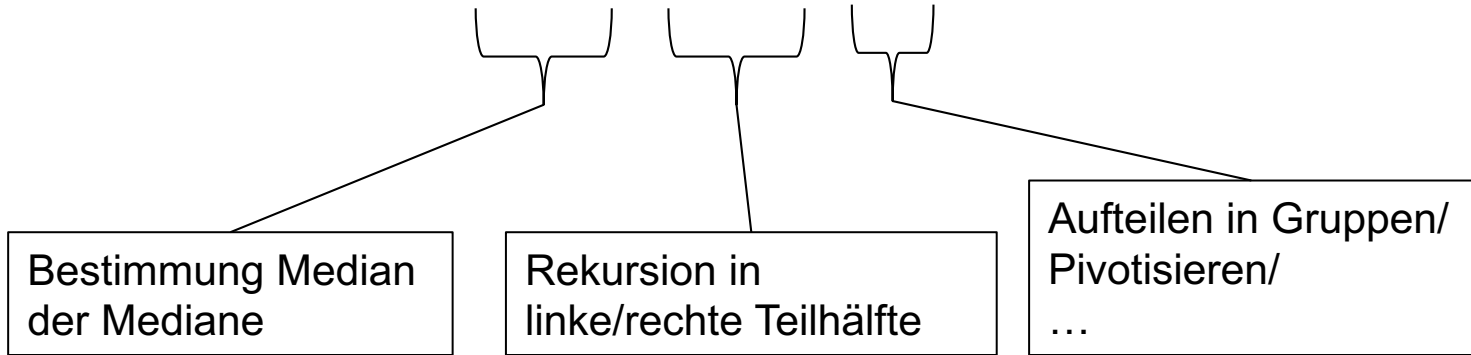
Damit gibt es maximal  $n - 3 \cdot \left\lfloor \frac{t}{2} \right\rfloor \leq n - \frac{3}{2}t \leq n - \frac{3}{2} \cdot \frac{n}{5} = \frac{7}{10}n$  Elemente größer als  $m$ .

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

# Mediane - Laufzeit

Wir haben also als Laufzeit:

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



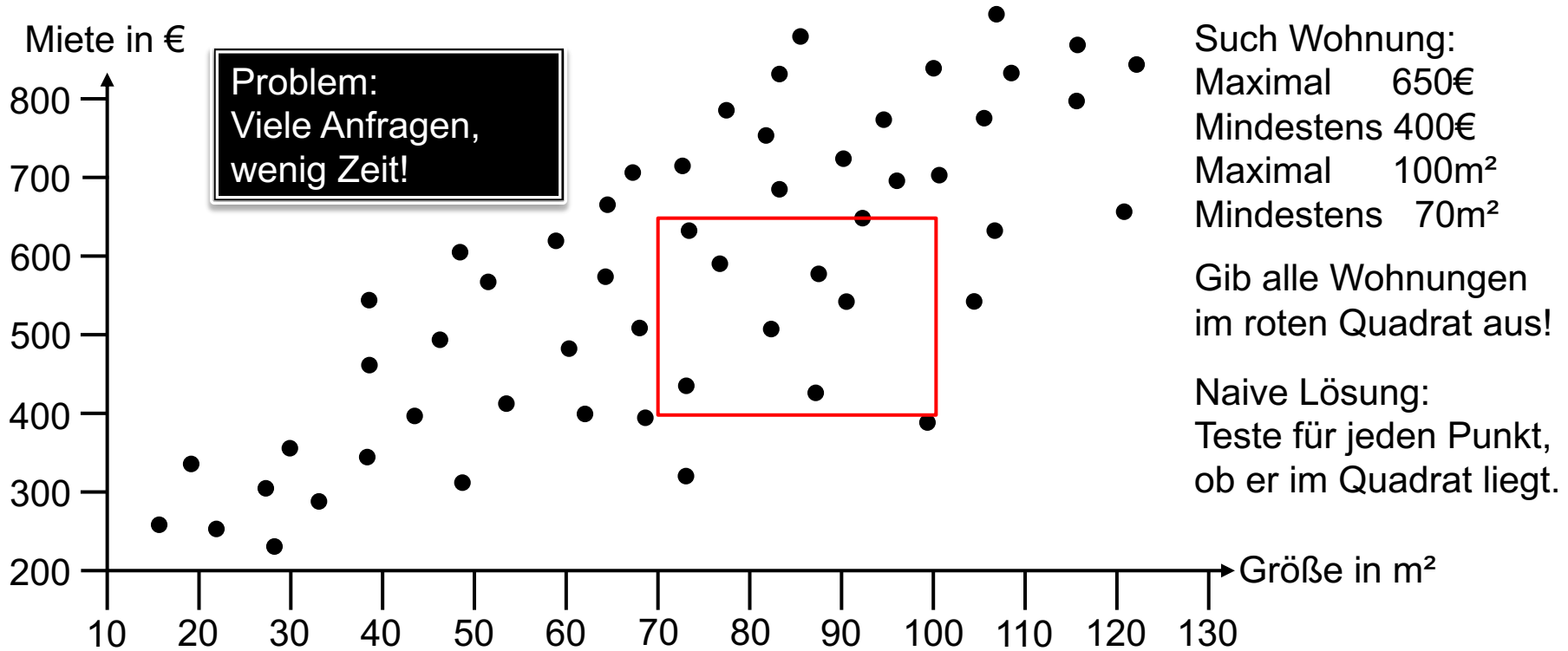
Mit dem Master-Theorem folgt:  $T(n) \in \Theta(n)$

Warum müssen das 5er Gruppen sein?



# kd-Bäume

# Motivation – Die Wohnungssuche



# kd-Bäume – Konstruktion/Preprocessing

Idee: Konstruiere binären Suchbaum

Trick: Suche abwechselnd nach

$x$ - und  $y$ -Koordinate.

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

*Beweis:*

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

(2) Rekursionsgleichung für die Zeit ist also:

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

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

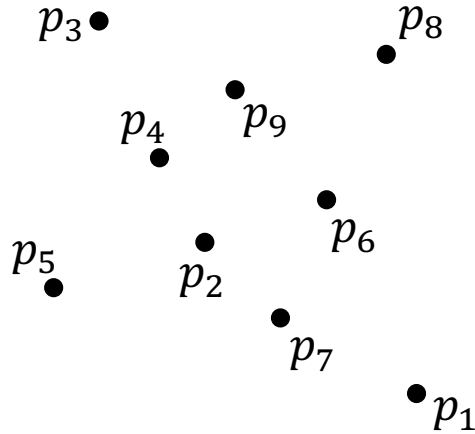
## Algorithmus BUILDKDTREE

Eingabe: Punktmenge  $P$ , Rekursionstiefe  $depth$

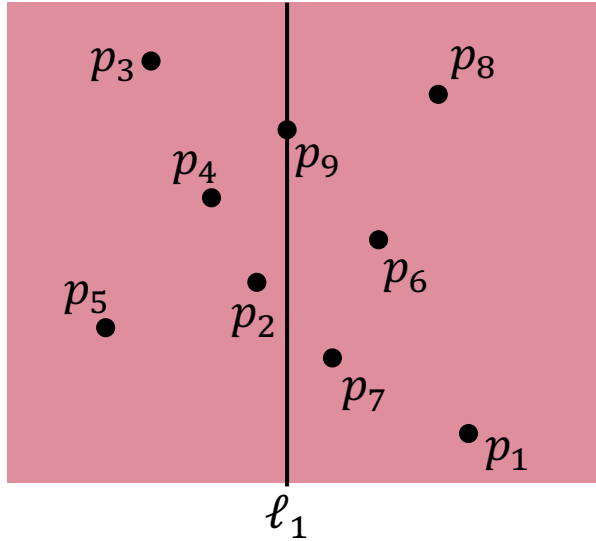
Ausgabe: Wurzel eines k-d-Baums

1. If ( $|P| = 1$ )
  - a. Return Blatt mit diesem Punkt
2. Else if ( $depth$  ist gerade)
  - a. Teile in zwei Teilmengen  $P_1 (\leq \ell)$  und  $P_2 (> \ell)$  an vertikaler Median-Linie  $\ell$
3. Else
  - a. Teile über horizontale Median-Linie  $\ell$
4. Setze  $v_{left} := \text{BUILDKDTREE}(P_1, depth+1)$
5. Setze  $v_{right} := \text{BUILDKDTREE}(P_2, depth+1)$
6. Erzeuge Knoten  $v$  für  $\ell$  mit  $v_{left}$  und  $v_{right}$  als Kinderknoten
7. Return  $v$

# kd-Bäume - Beispiel

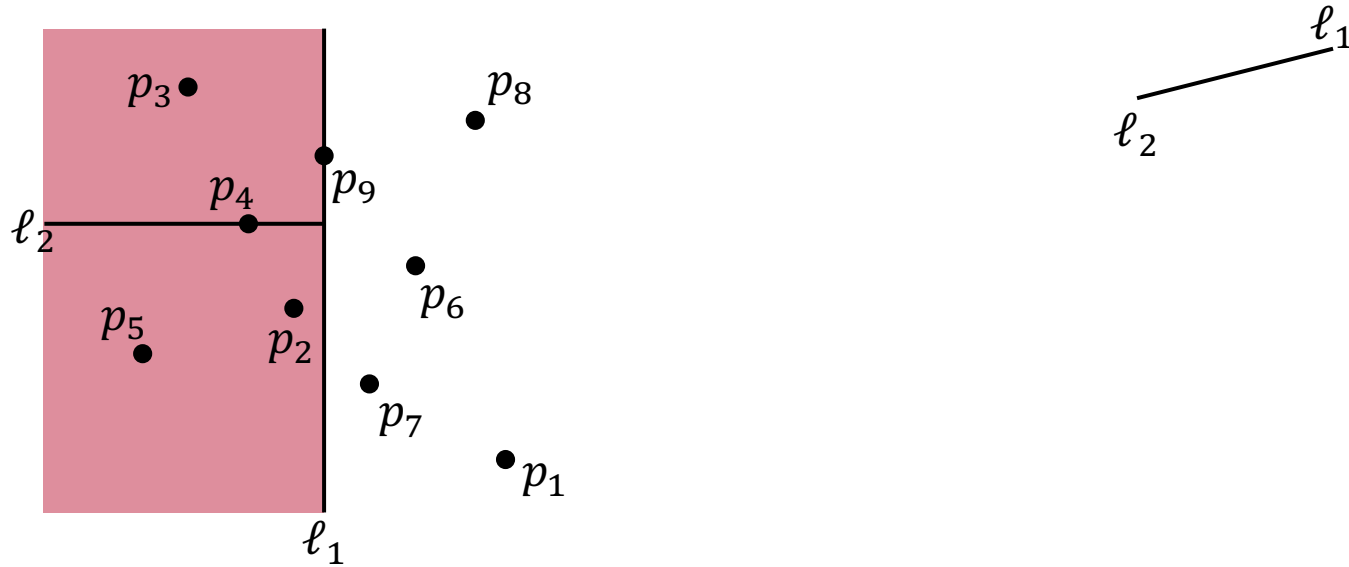


# kd-Bäume - Beispiel

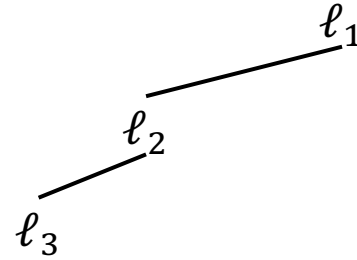
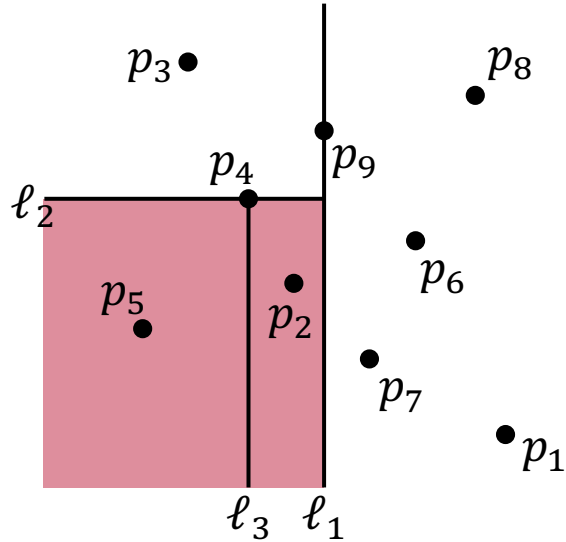


$\ell_1$

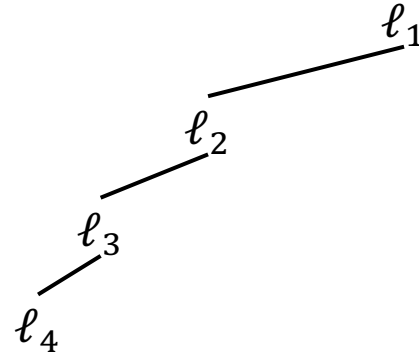
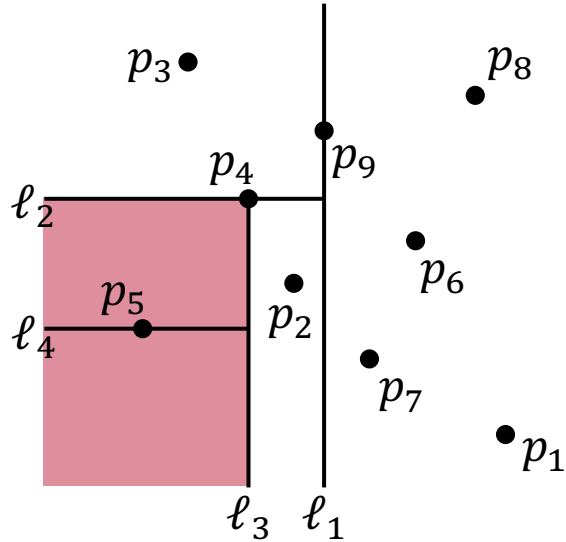
# kd-Bäume - Beispiel



# kd-Bäume - Beispiel

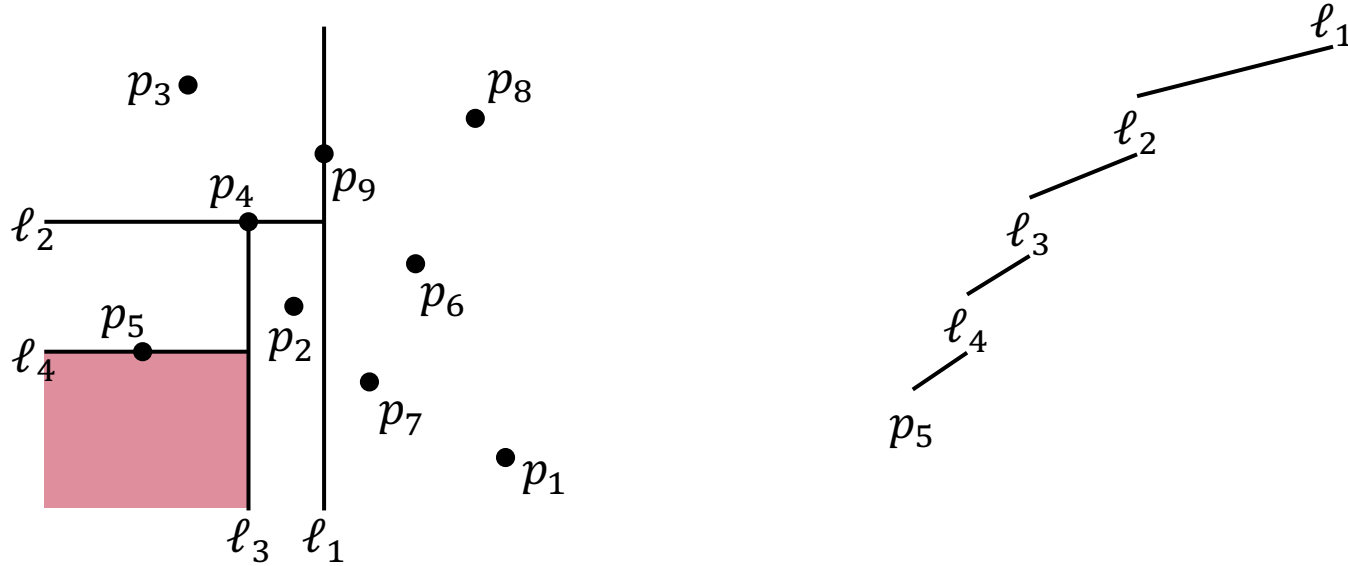


# kd-Bäume - Beispiel

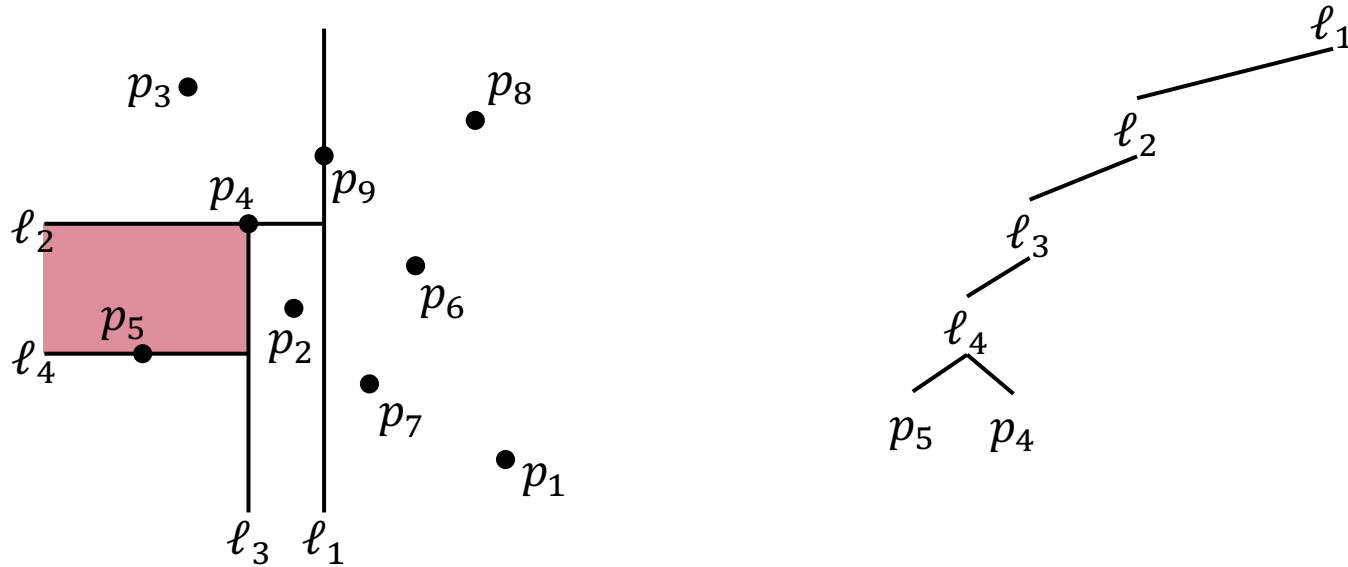




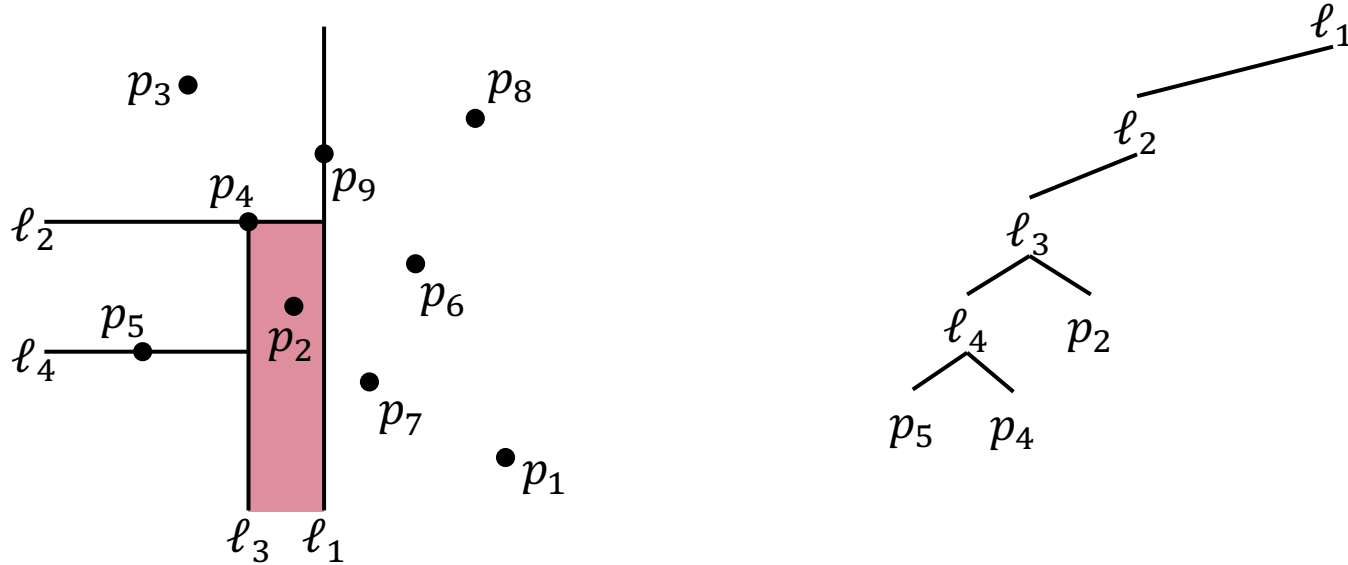
# kd-Bäume - Beispiel



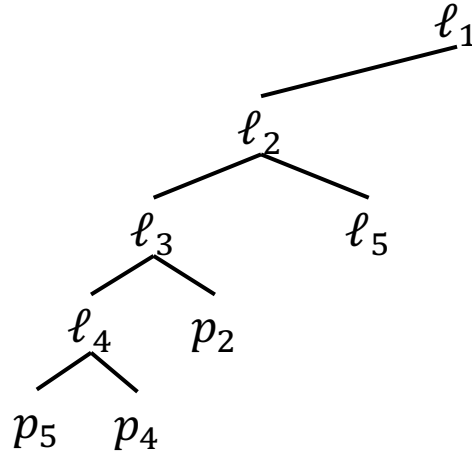
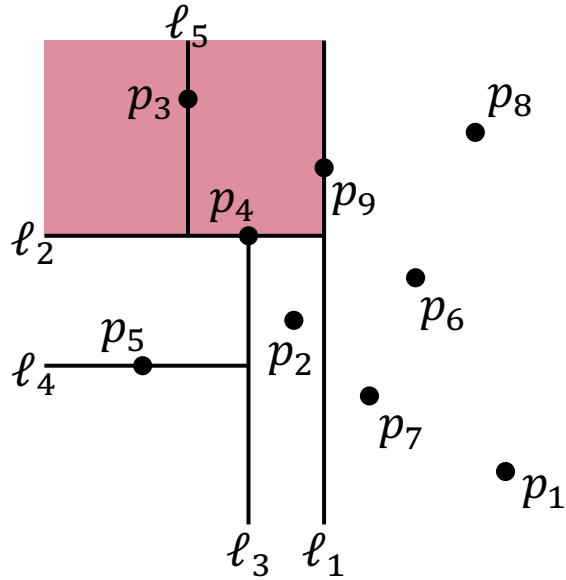
# kd-Bäume - Beispiel



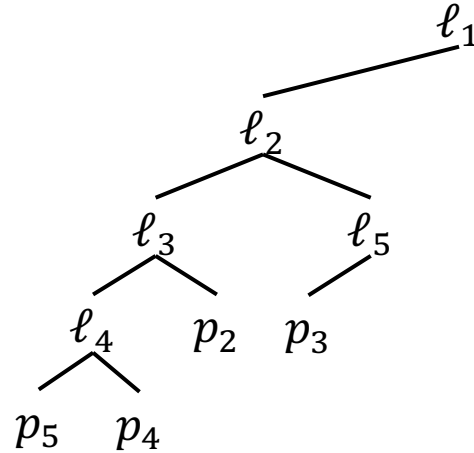
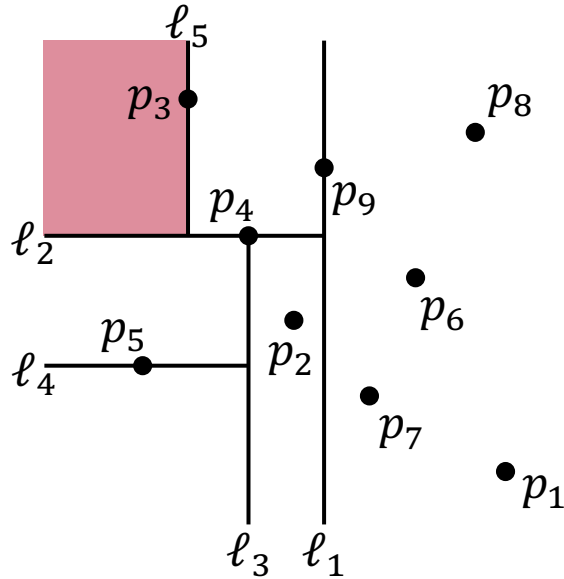
# kd-Bäume - Beispiel



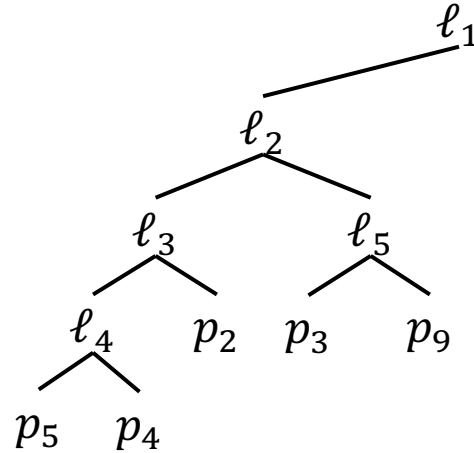
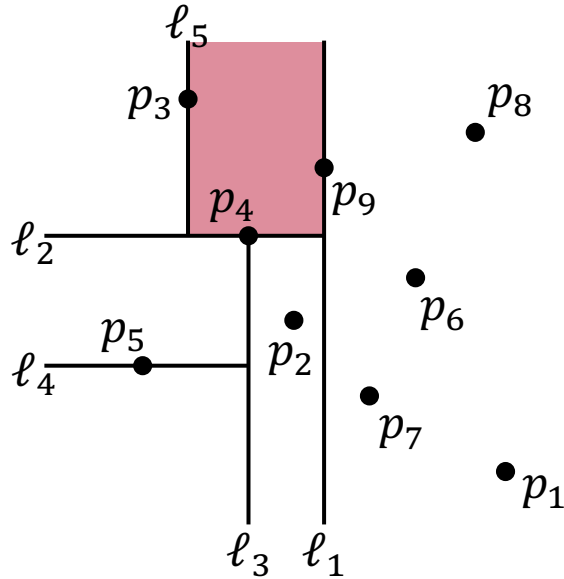
# kd-Bäume - Beispiel



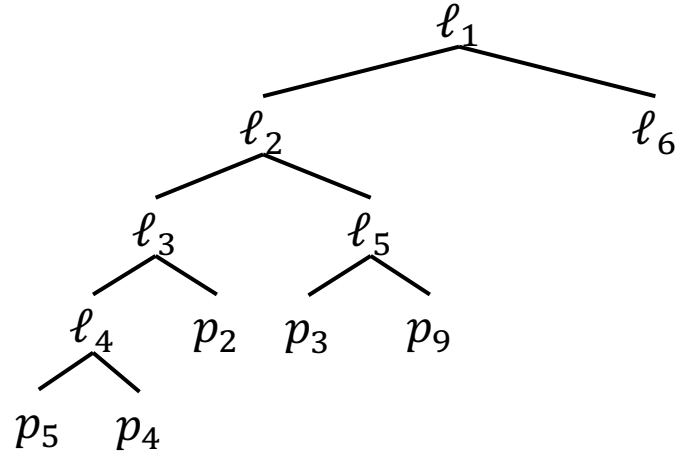
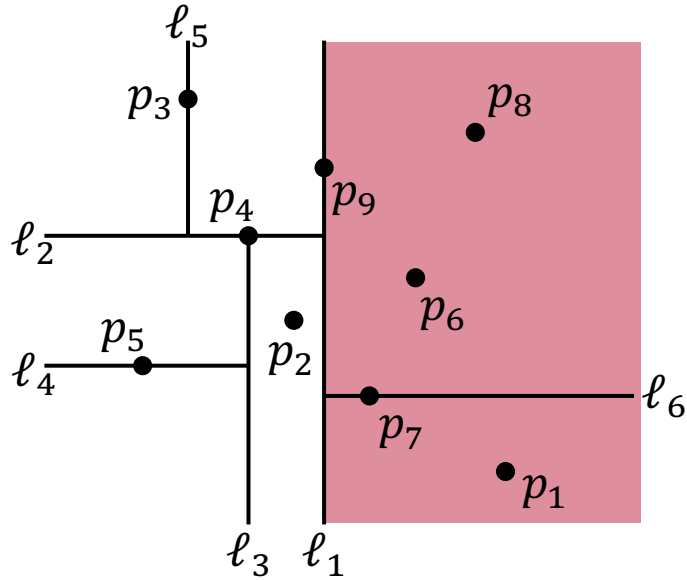
# kd-Bäume - Beispiel



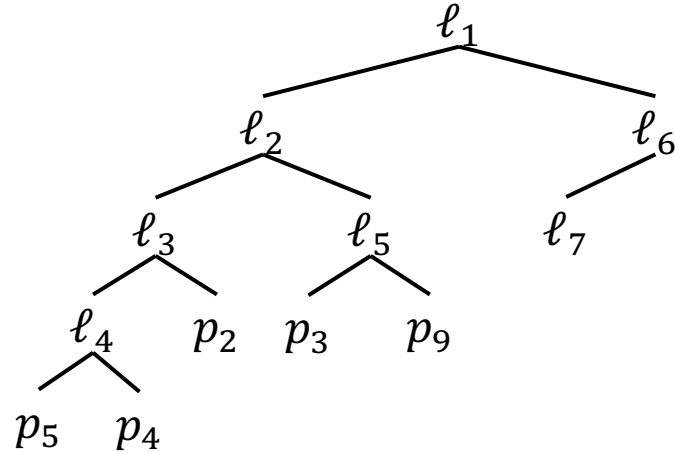
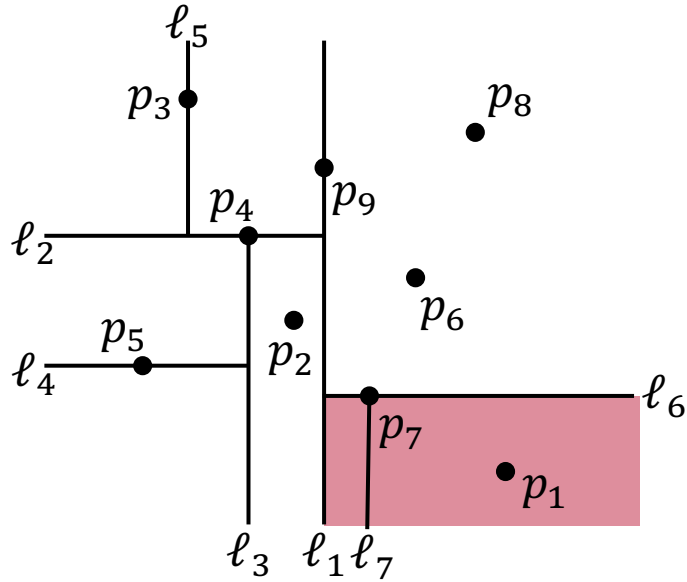
# kd-Bäume - Beispiel



# kd-Bäume - Beispiel

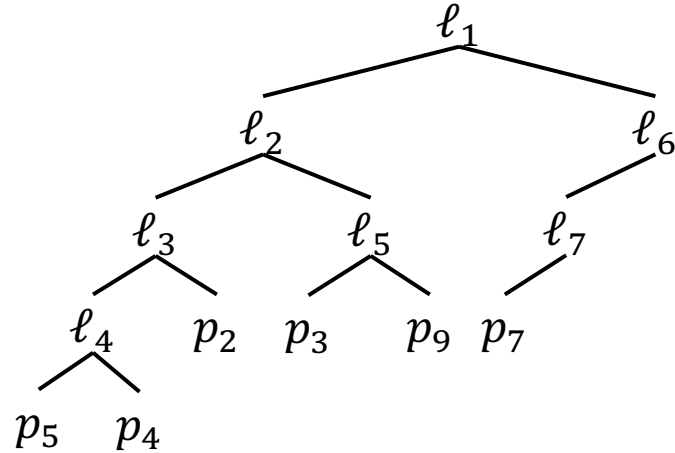
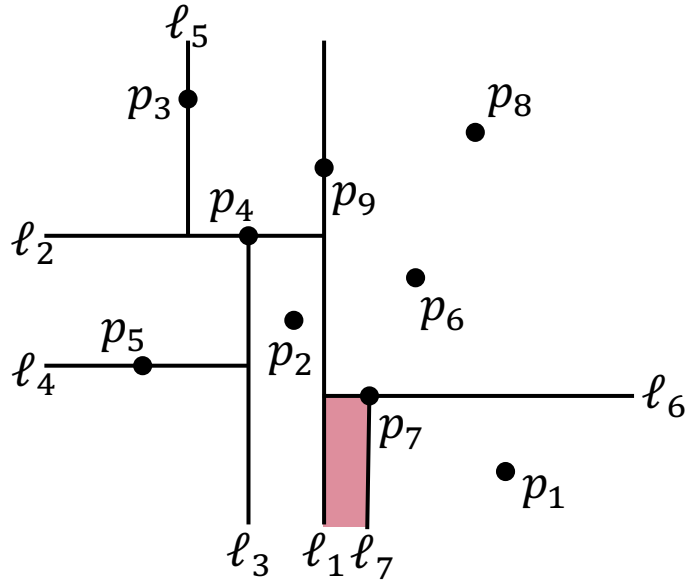


# kd-Bäume - Beispiel

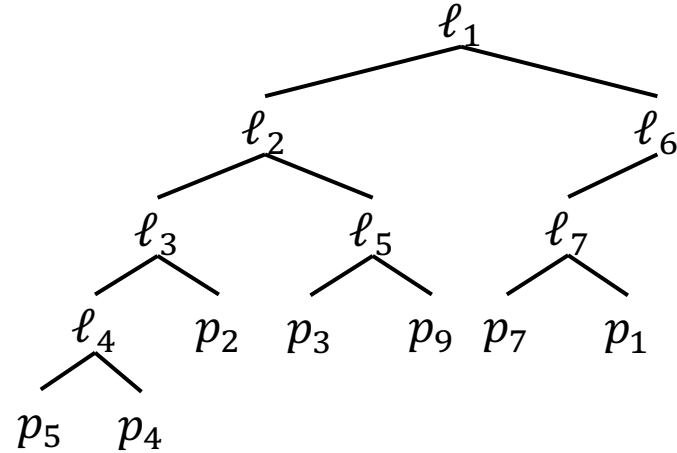
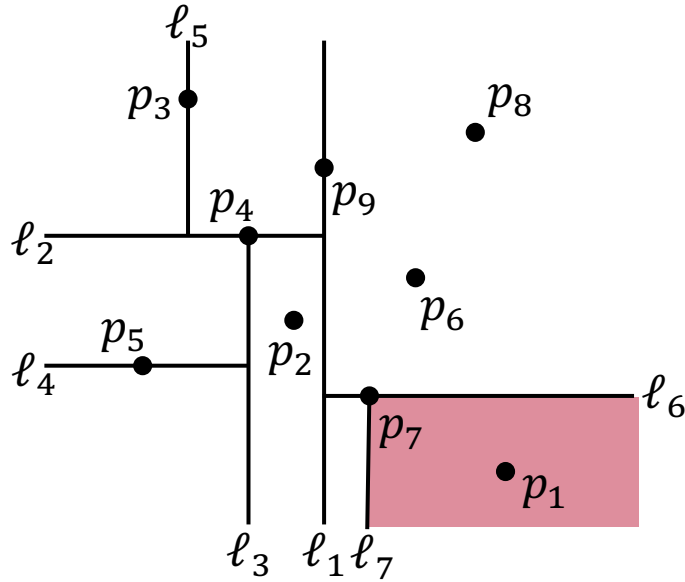




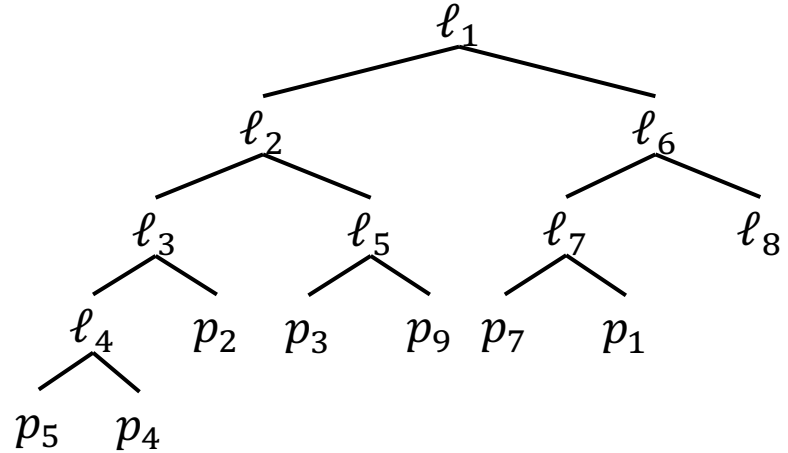
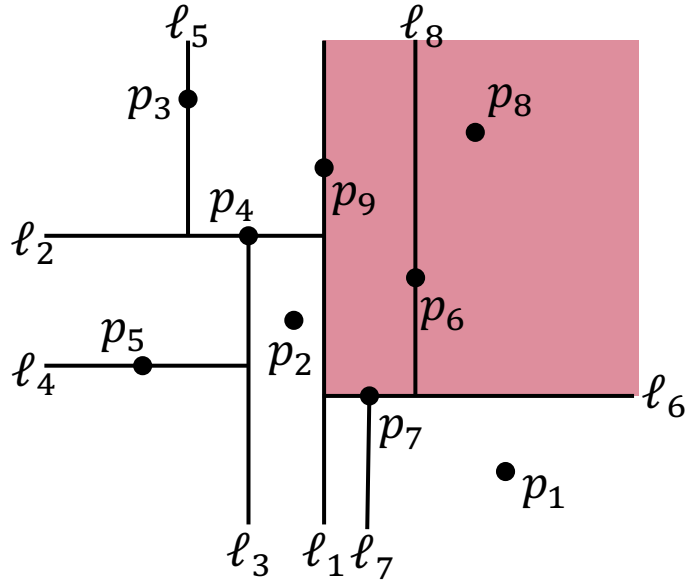
# kd-Bäume - Beispiel



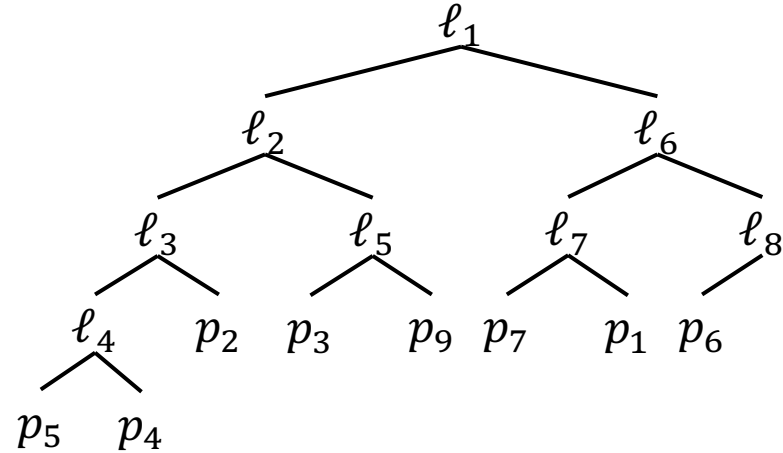
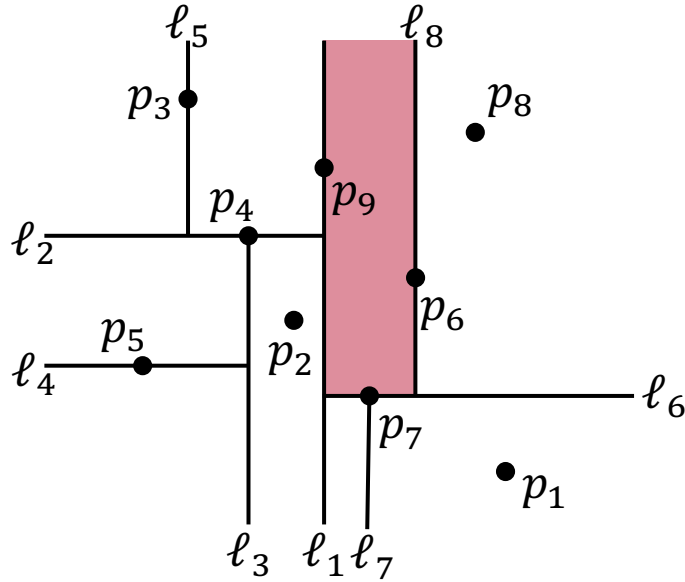
# kd-Bäume - Beispiel



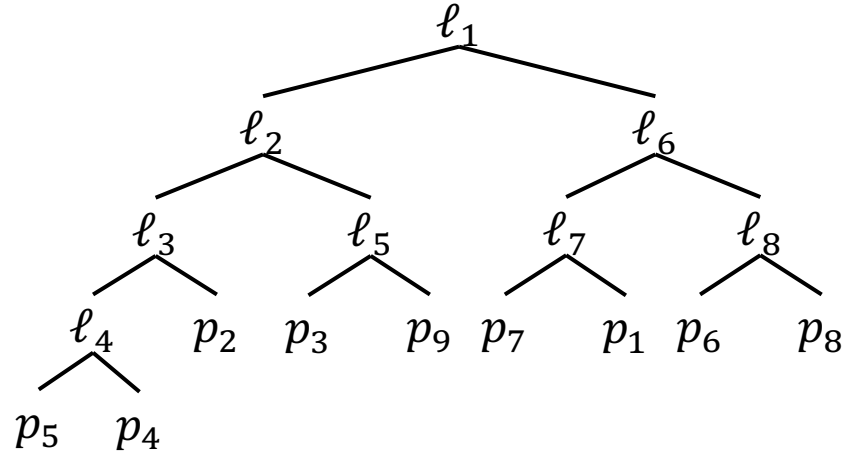
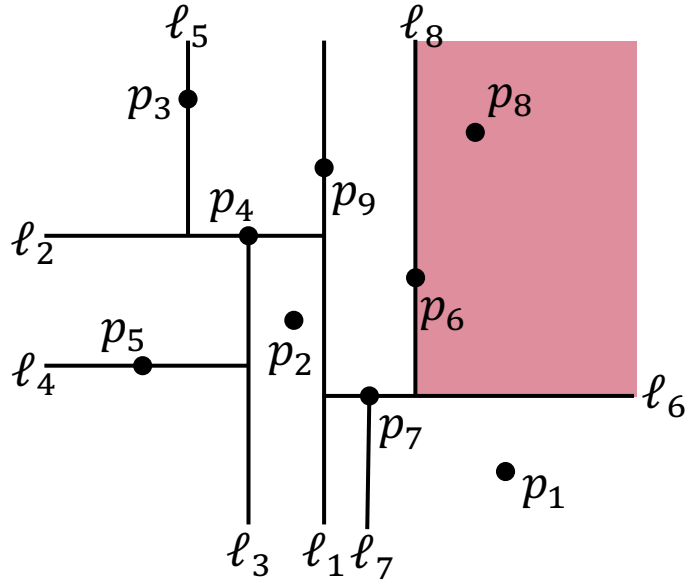
# kd-Bäume - Beispiel



# kd-Bäume - Beispiel



# kd-Bäume - Beispiel



# kd-Bäume – Search Query

Laufzeit ist  $O(\sqrt{n} + x)$ , wobei  $x$  die Anzahl an ausgegebenen Elementen ist. Das nennt sich *output-sensitive*, 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  $R$ .

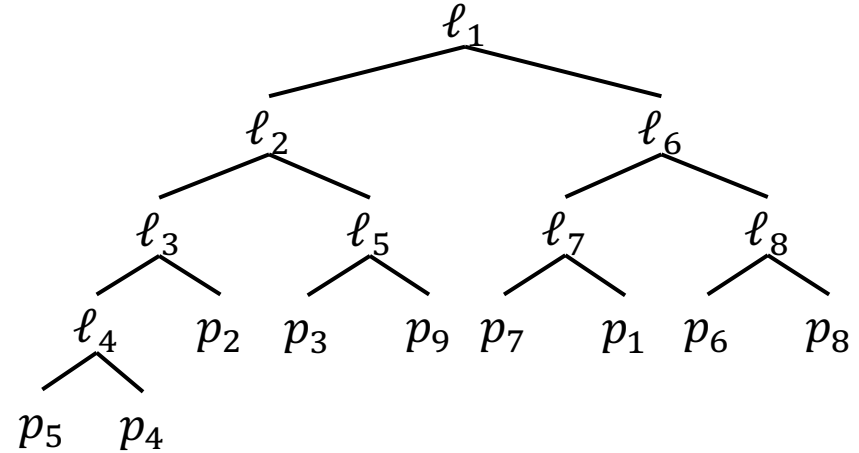
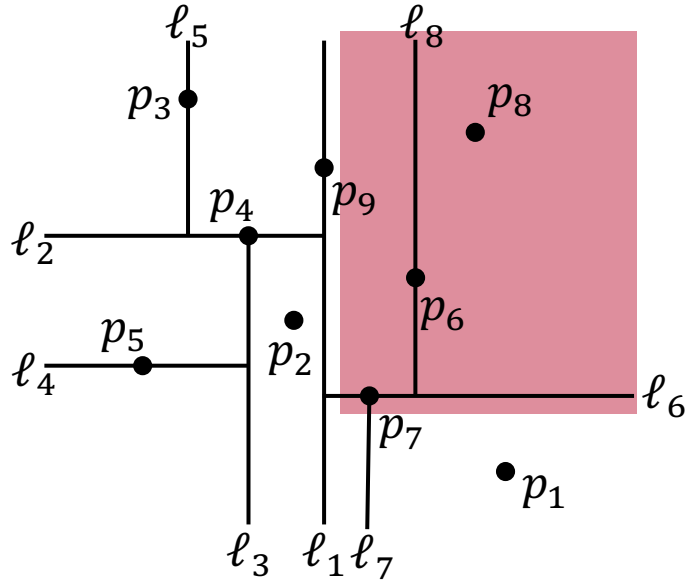
## Algorithmus SEARCHKDTREE

Eingabe: Wurzel  $v$  eines (Teil-)baums, Rechteck  $R$

Ausgabe: Knoten unterhalb  $v$ , die in  $R$  liegen

1. If ( $v$  ist ein Blatt)
  - a. Gib  $v$  aus, falls  $v$  in  $R$
2. Else
  - a. if (Region(lc( $v$ )) ganz in  $R$ )
    - i. REPORTSUBTREE(lc( $v$ ))
  - b. Else if (Region(lc( $v$ )) schneidet  $R$ )
    - i. SEARCHKDTREE(lc( $v$ ),  $R$ )
  - c. if (Region(rc( $v$ )) ganz in  $R$ )
    - i. REPORTSUBTREE(rc( $v$ ))
  - d. Else if (Region(rc( $v$ )) schneidet  $R$ )
    - i. SEARCHKDTREE(rc( $v$ ),  $R$ )

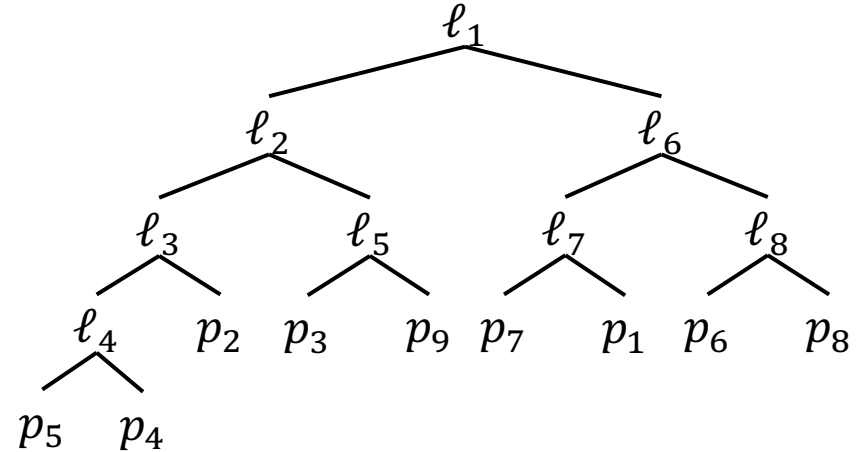
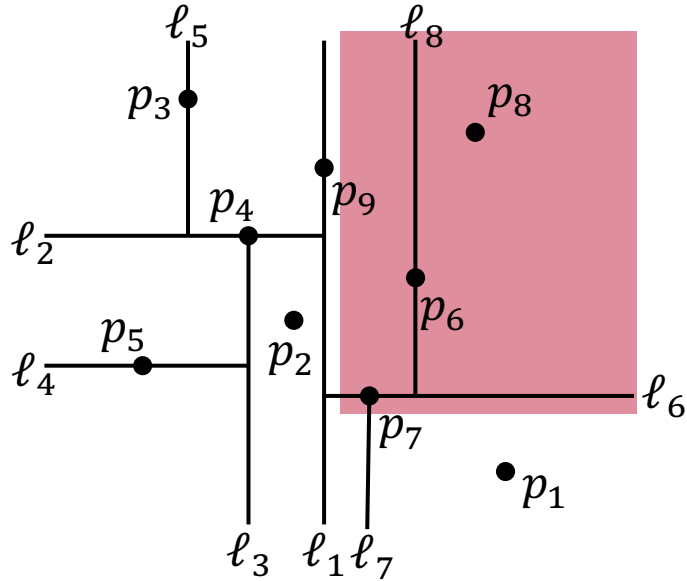
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:

# kd-Bäume - Beispiel

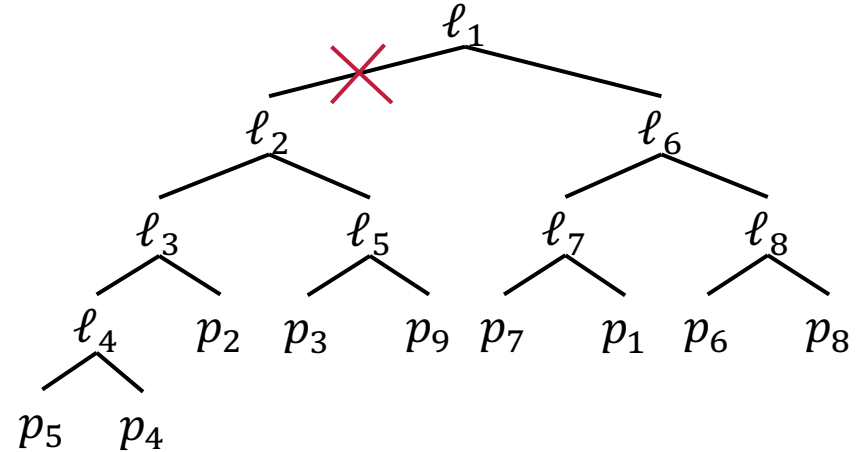
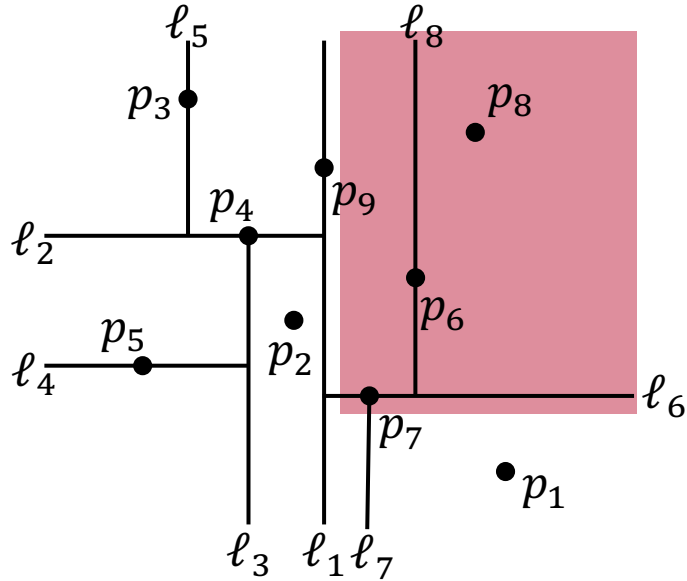


Welche Punkte liegen im Rechteck?

Antwort:



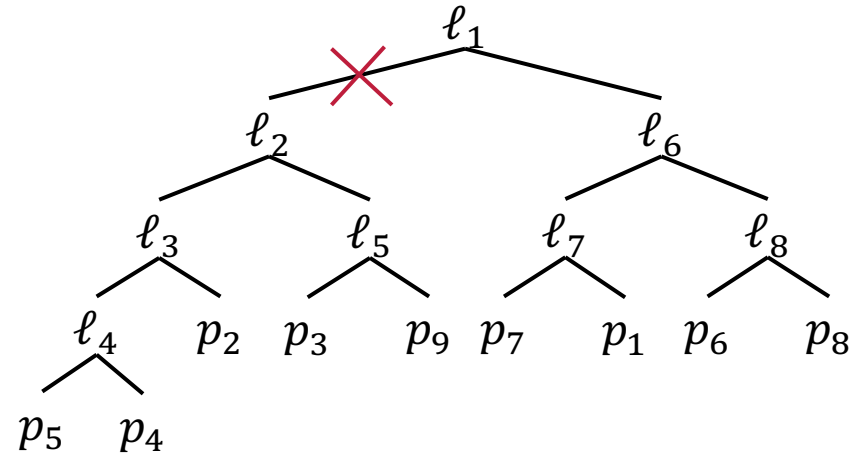
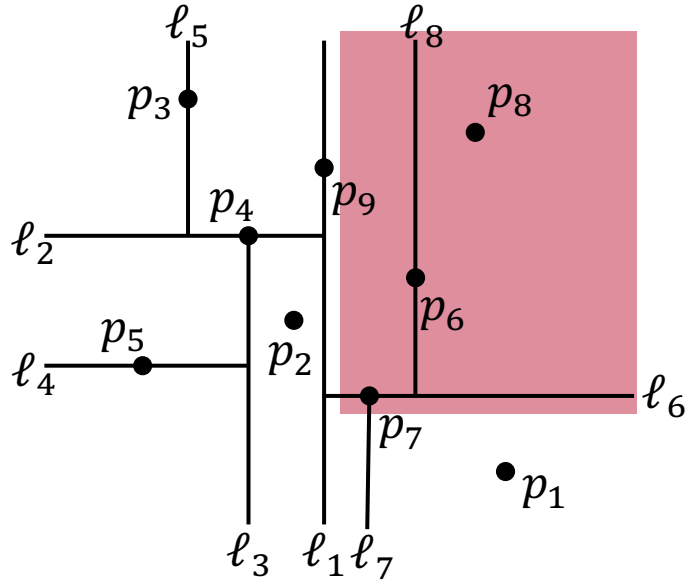
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:

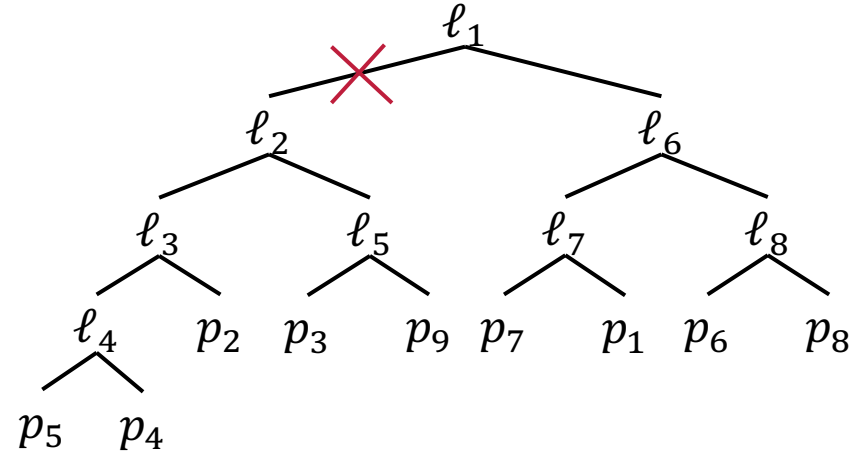
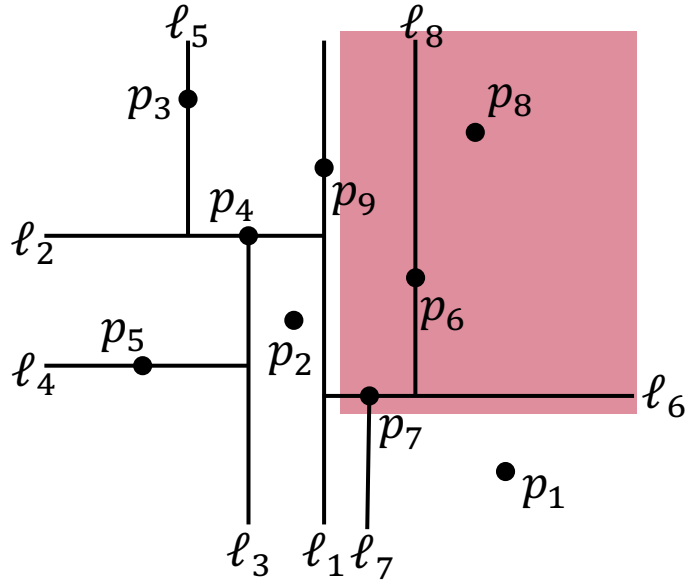
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:

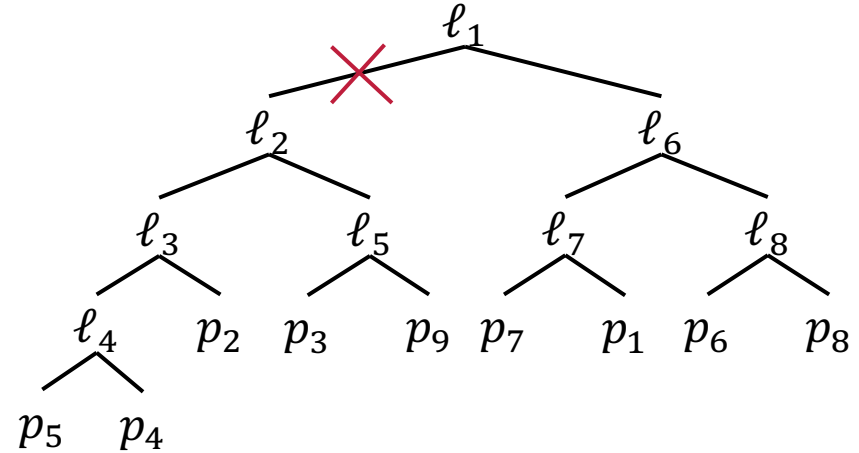
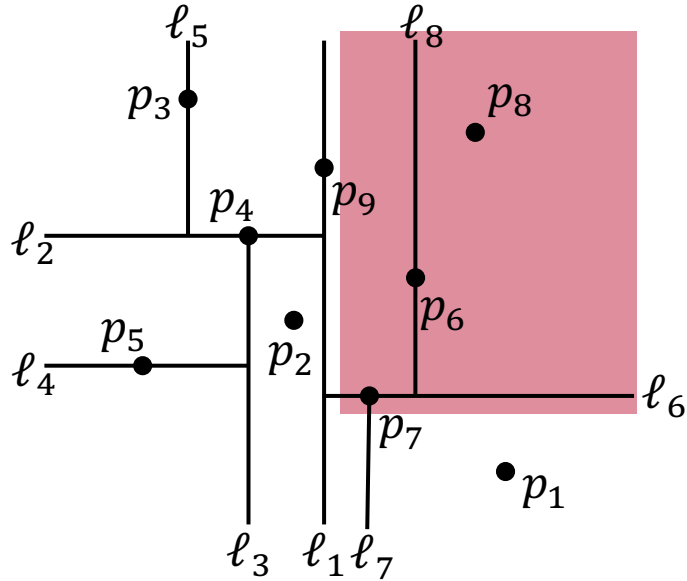
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:

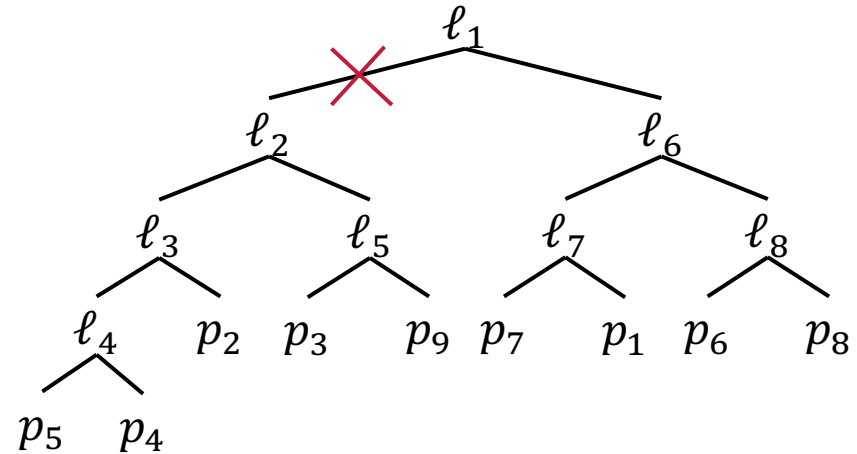
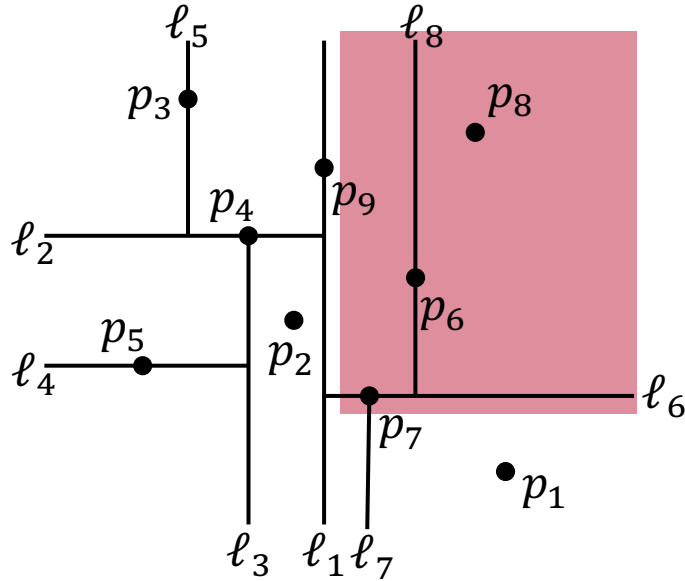
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:  $p_7$

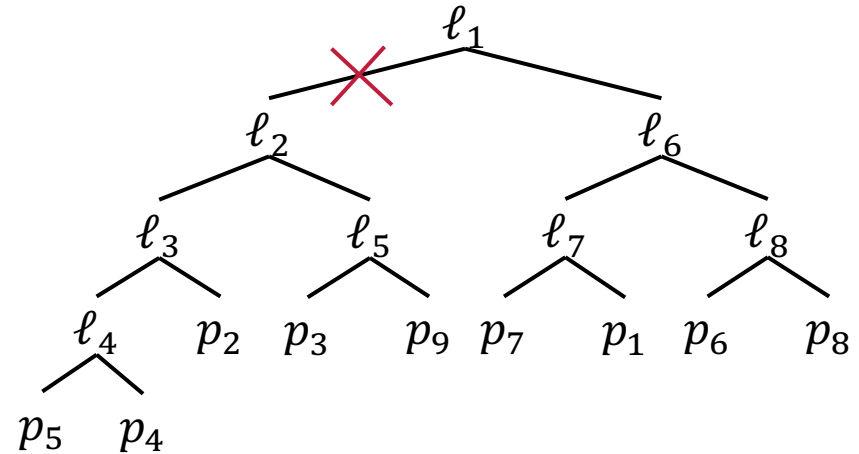
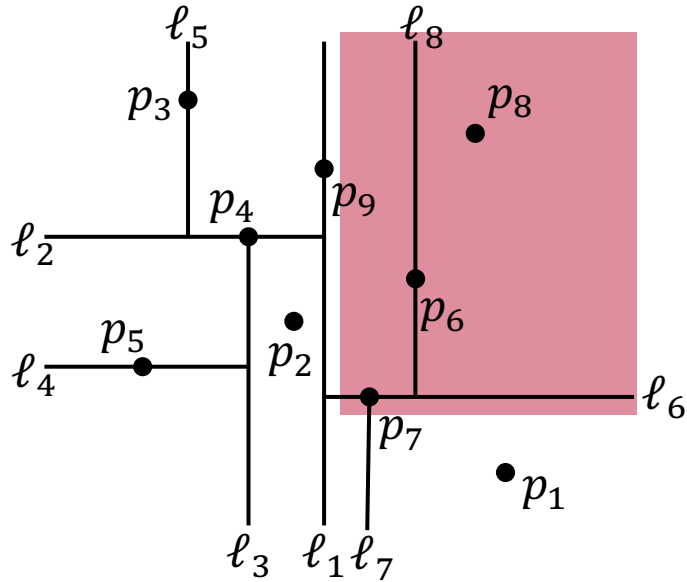
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:  $p_7$

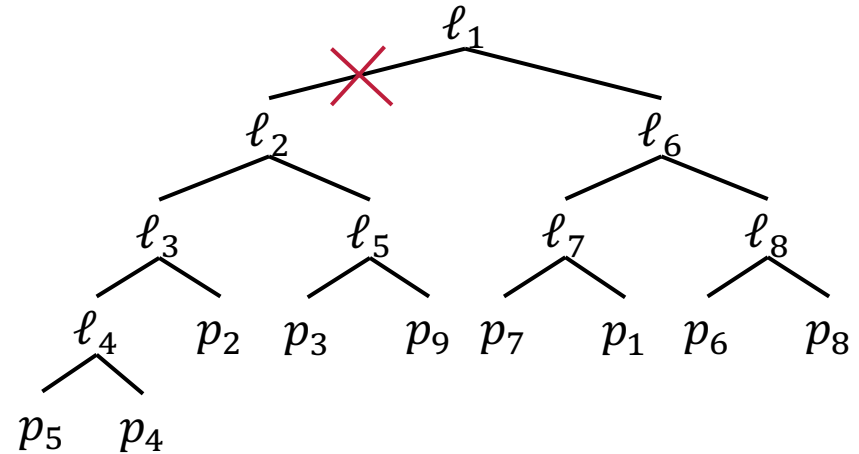
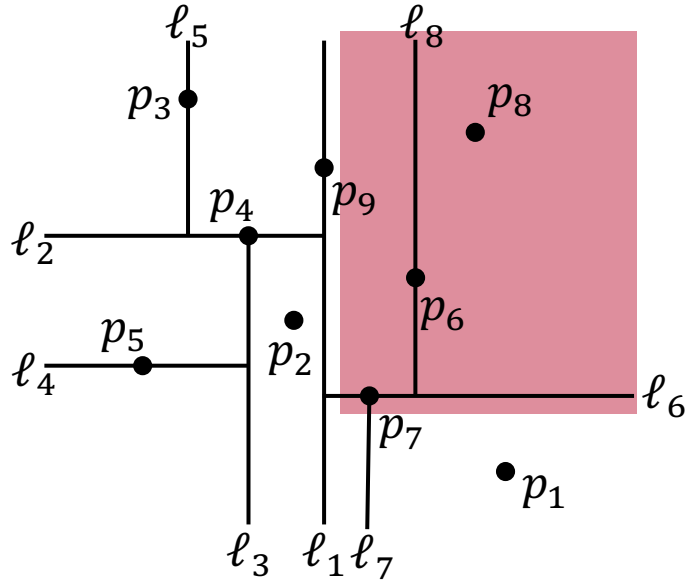
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:  $p_7$

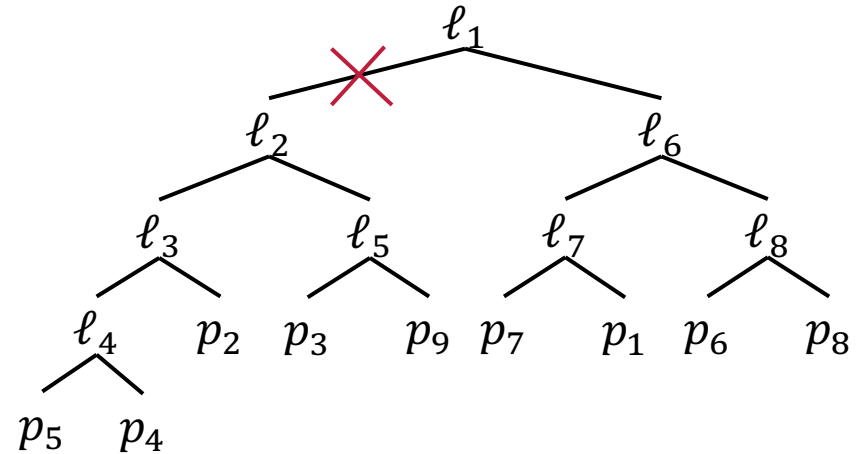
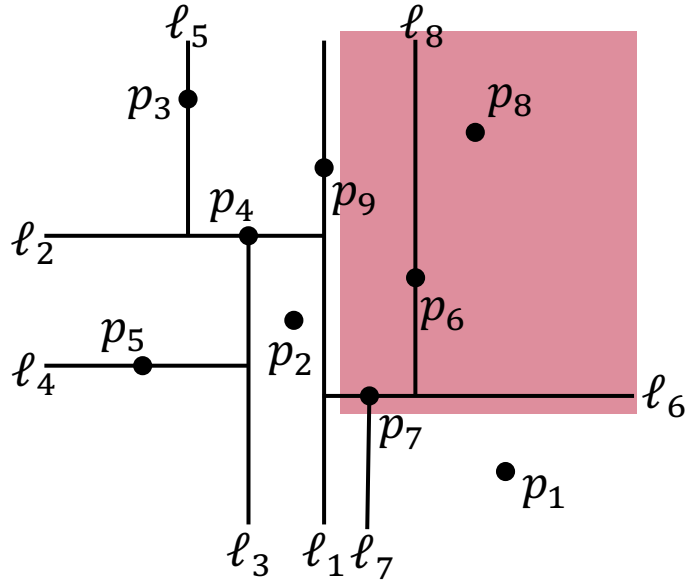
# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:  $p_7$   $p_6$

# kd-Bäume - Beispiel



Welche Punkte liegen im Rechteck?

Antwort:  $p_7$   $p_6$   $p_8$

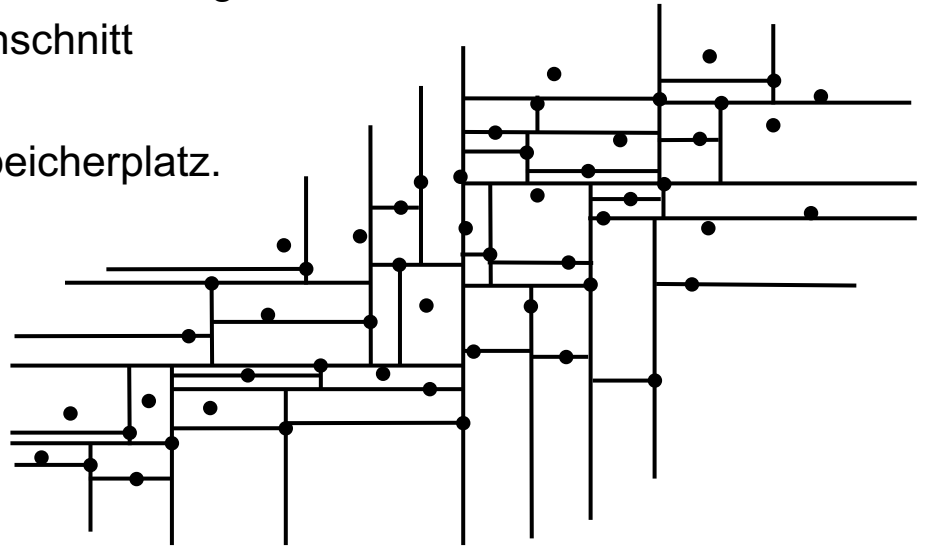


# kd-Bäume – Höhere Dimensionen

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.



# Ausblick

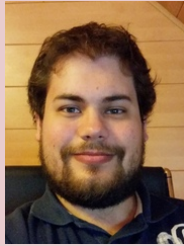
# Im Sommer...

...gibt es wieder zwei Veranstaltungen für den Bachelor-Bereich:

## Algorithmen und Datenstrukturen 2



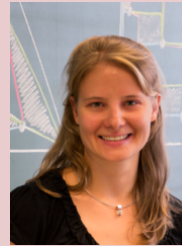
Prof. Sándor Fekete



Arne Schmidt

- Einführung in Komplexitätstheorie
- Heuristiken
- Exakte Methoden
- Approximationen
- Hashing

## Netzwerkalgorithmen



Linda Kleist



Christian Rieck

- Kostenminimale aufspannende Bäume
- Kürzeste Wege
- Maximale Flüsse
- Kardinalitätsmaximales Matching