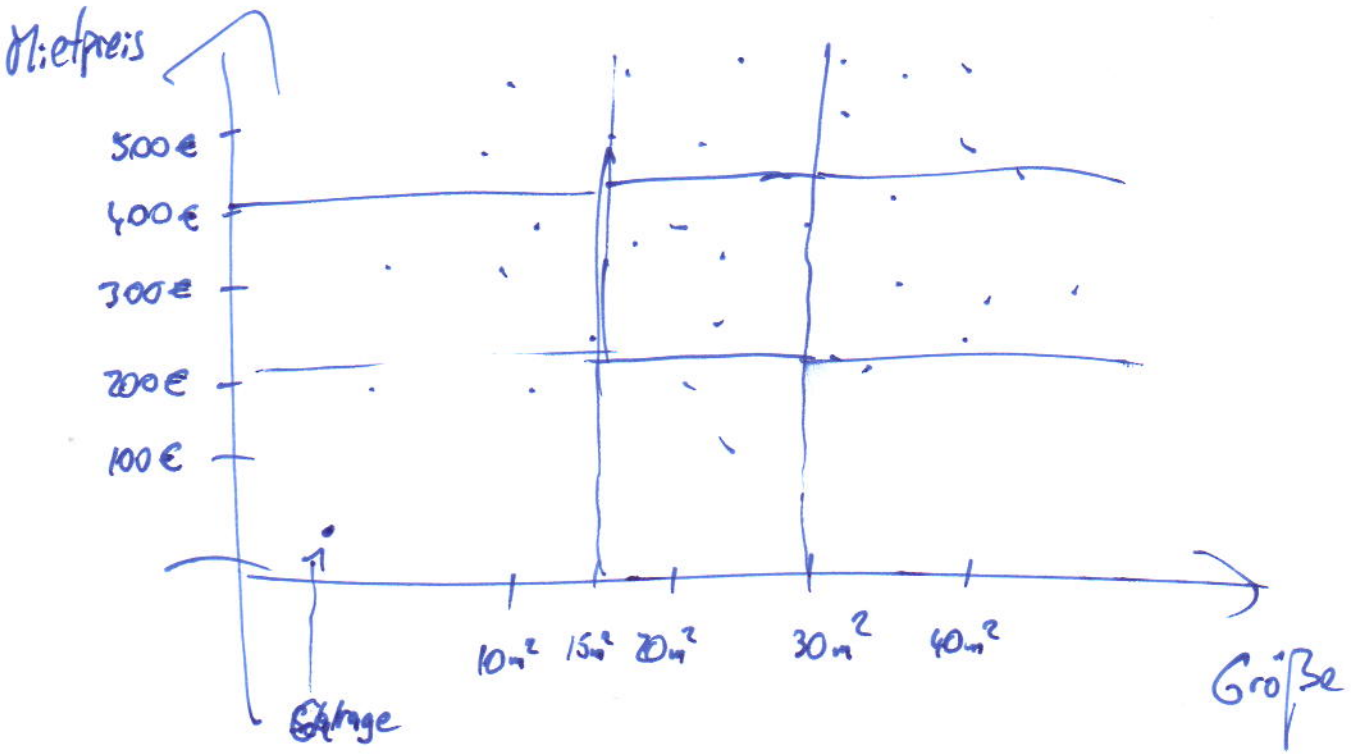
2.4 Bereichs-Lokalisierung von Punktmengen

2.4.1 Aufgabenstellung

Szenario: Große Punktmenge $P_i = (x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)})$

Gesucht: Alle Punkte in der Menge, die innerhalb der Grenzen $x_j^{(i)} \in [A_j, B_j]$ liegen:



Wichtige Punkte:

- Daten vorher bekannt
- Anfragen immer wieder
- Laufzeit $O(n)$ einfach, aber ggf. zu langsam
- Bei vielen Antworten dauert Antwort länger

Ideen:

- Preprocessing / Query
- Geeignete Datenstrukturen
- Ergebnissensitive Laufzeit

2.4.2 Dimension $d=1$

Siehe Übung!

- Binärer Suchbaum, Aufbau in $O(n \log n)$
- Abfrage- im Suchbaum, linke Grenze / rechte Grenze

  $O(\log n)$
- Ausgabe zwischen Grenzen

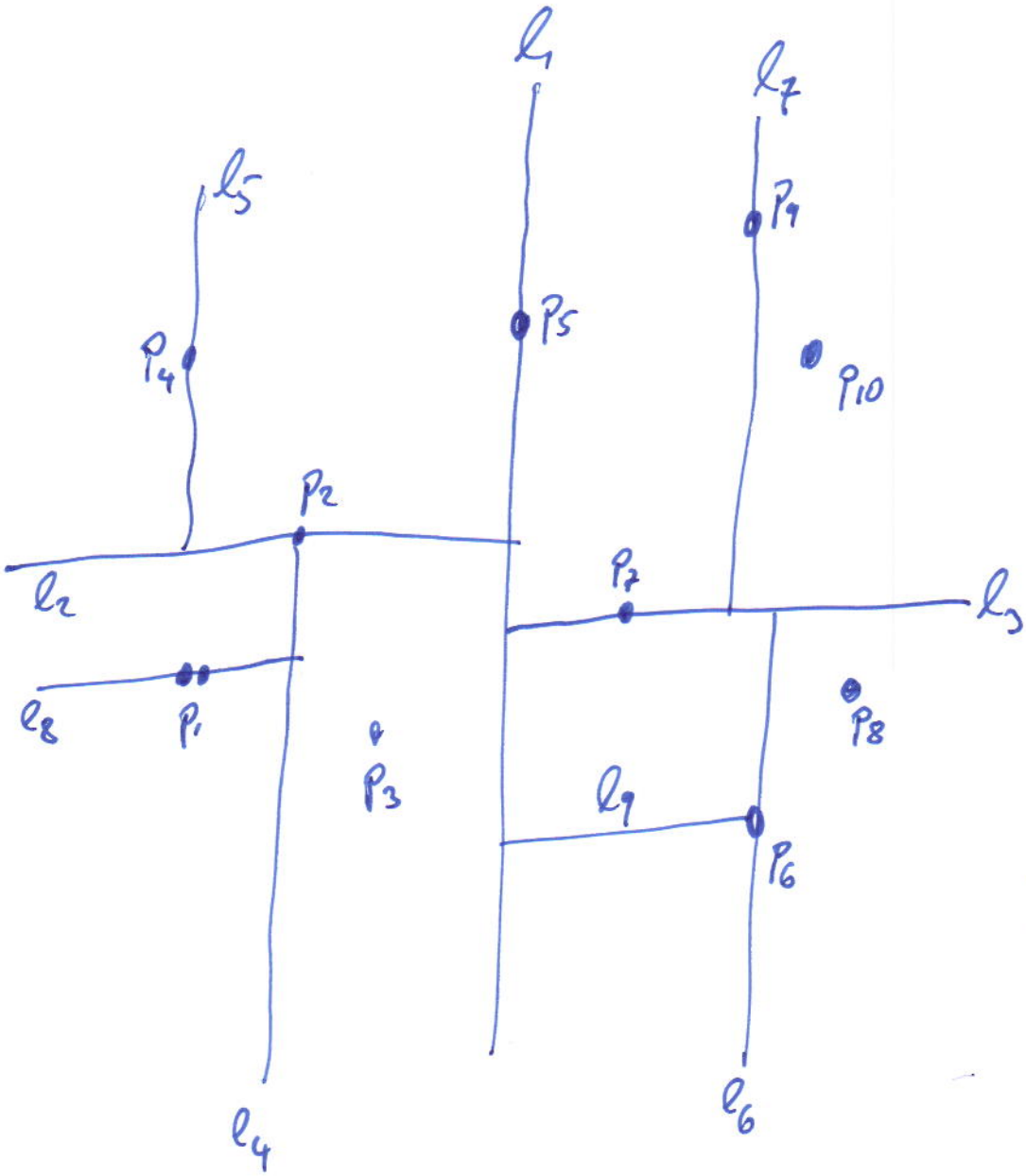
Gesamt $O(\log n + k)$

↑ output

2.4.3 Dimension $d=2$

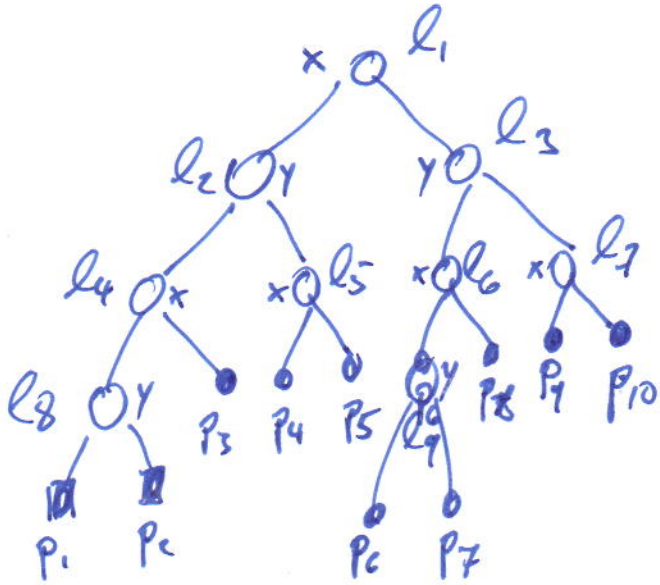
Problem: Ordnung in jeweiligen Dimensionen muss nicht übereinstimmen!

Wenn erst x_1-Grenzen abgefragt werden, weiß man danach nichts über x_2 → Binärer Suchbaum reicht nicht!



Abfragen:
$p_i \leq l_j$?

Lauter Medians!



Algorithmus 2.12 BuildKdTree (P, depth)

Eingabe: Punktemenge P, aktuelle Tiefe depth
Ausgabe: Wurzel eines kd-Baumes mit Punktemenge P

1 IF ($|P| \leq 1$)
  1.1 RETURN Blatt mit diesem Punkt
  1.2 ELSE IF depth gerade
    1.2.1 THEN Teile P durch vertikale Linie l in zwei Teilmengen P_1 ($\leq l$) und P_2 ($> l$) an Median-Linie l
  1.3 ELSE
    1.3.1 THEN Teile P durch horizontale (...)
2 v_{left} := BuildKdTree (P_1, depth + 1)
3 v_{right} := BuildKdTree (P_2, depth + 1)
4 Erzeuge Knoten v für l,
  setze v_{left} als linkes
  v_{right} als rechtes Kind
5 RETURN v.

Satz 2.13

Ein kd-Baum benötigt Zeit $O(n \log n)$ zum Bauen.

Beweis:

(1) Sortieren

(2) $T(n) = \begin{cases} O(1) & \text{für } n = 1 \\ O(n) + 2T(\lceil \frac{n}{2} \rceil) & \text{für } n > 1 \end{cases}$

ALGORITHMUS 2.14 SEARCH KDTREE (v, R)

Eingabe: Wurzel eines (Teil-) Baumes, Rechteck R
Ausgabe: Alle Knoten unterhalb v, die in R liegen.

(1) IF (v ist Blatt)
1.1 THEN Gib v aus, falls $v \in R$
1.2 ELSE IF (Region(lc(v)) ganz in R)
1.2.1 THEN REPORT SUBTREE (lc(v))
1.2.2 ELSE IF (Region(lc(v)) schneidet R)
1.2.2.1 THEN SEARCH KDTREE (lc(v), R)
1.3 IF (Region(rc(v)) ganz in R)
1.3.1 THEN REPORT SUBTREE (rc(v))
1.3.2 ELSE IF (Region(rc(v)) schneidet R)
1.3.2.1 THEN SEARCH KDTREE (rc(v), R)

SATZ 2.15

Laufzeit für Abfrage ist $O(\sqrt{n}+k)$,
k ist Ausgabegröße. ("Output-sensitive")

Beweis:

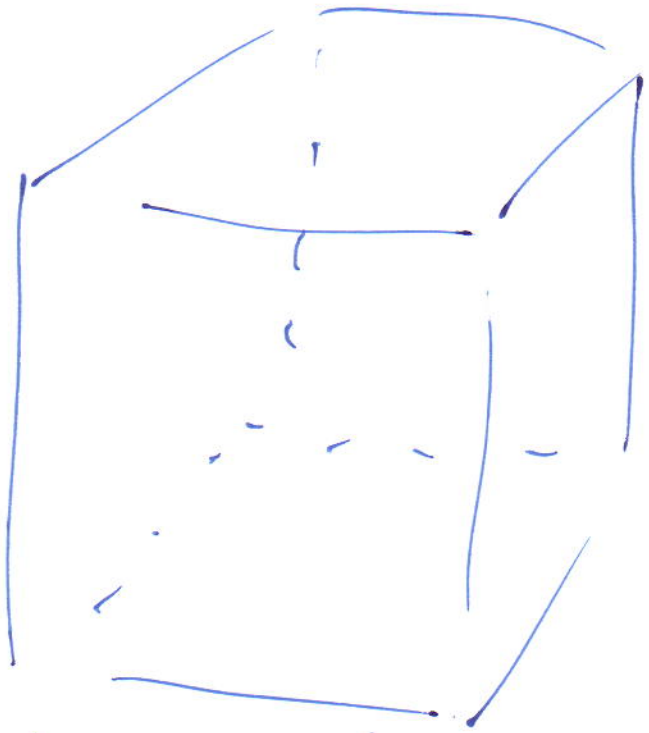
$$Q(n) = \begin{cases} O(1) & \text{für } n=1 \\ 2+2Q(n/4) & \text{für } n>1 \end{cases}$$

$\hookrightarrow \quad Q(n) = O(\sqrt{n})$

SATZ 2.16

In d (statt 2) Dimensionen erhält man $O\left(n^{1-\frac{1}{d}} + k\right)$.

Argument:



Oberfläche hat Dimension $d-1$
Volumen hat Dimension d $\Rightarrow$ $\frac{d-1}{d} = 1-\frac{1}{d}$

(Auch per Rekursion möglich!)