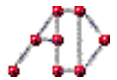
Orthogonale Bereichssuche

1. 1-dim Bereichssuche: 1-dim Range Trees

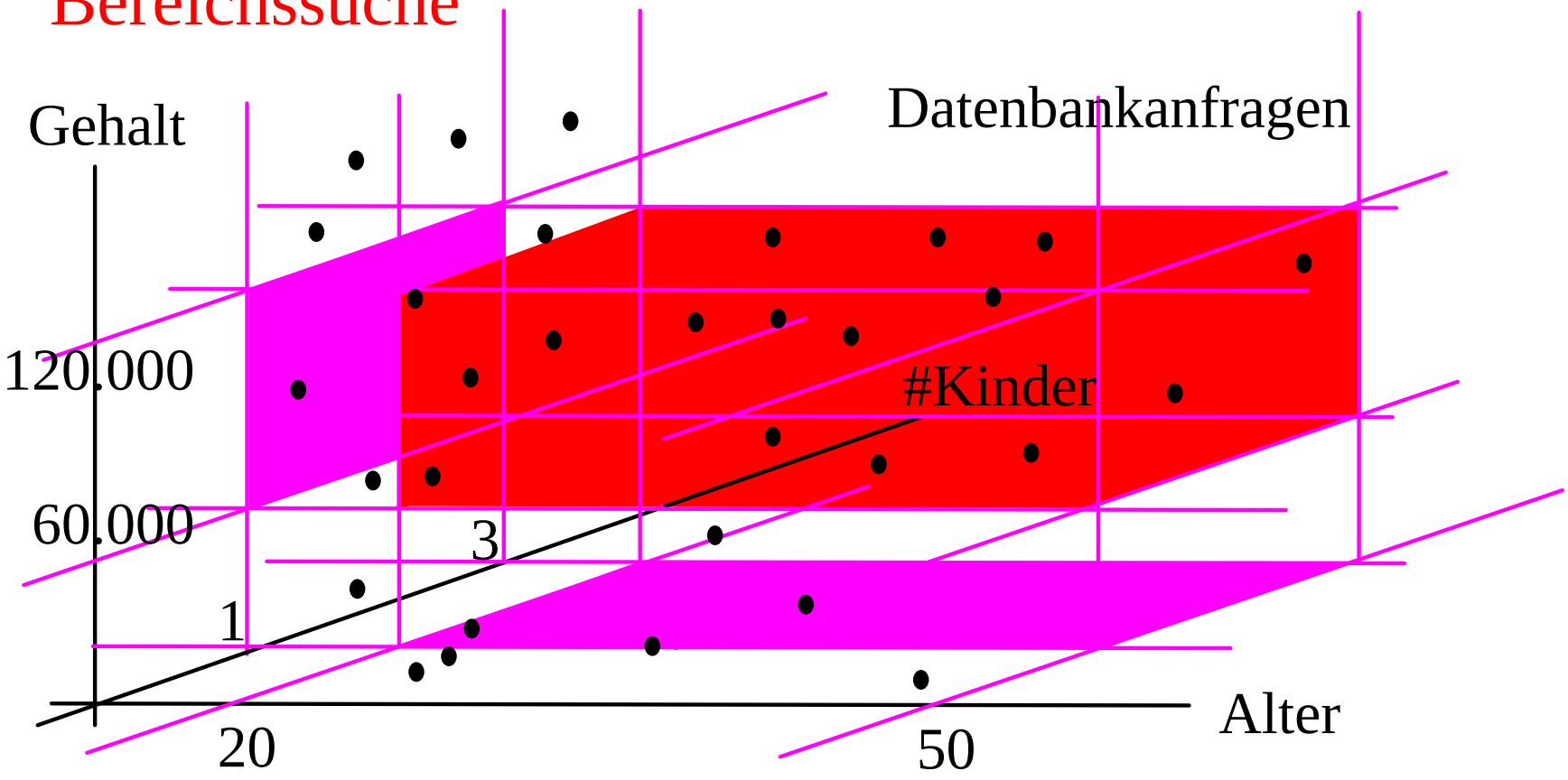
2. 2-dim Bereichssuche: k-d-Bäume

3. 2-dim Bereichssuche: 2-dim Range Trees

Bereichssuche in höheren Dimensionen



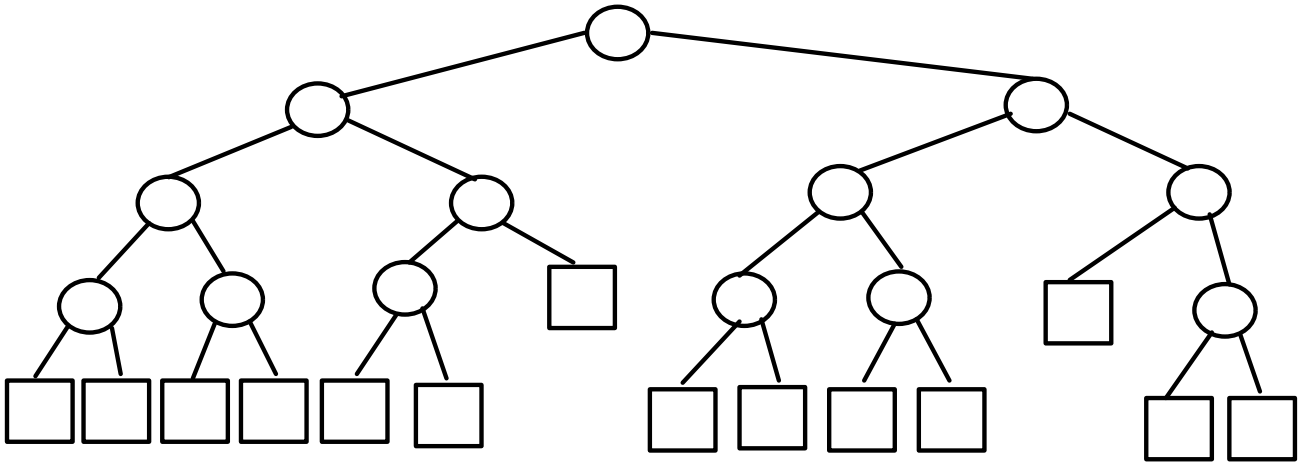
Bereichssuche



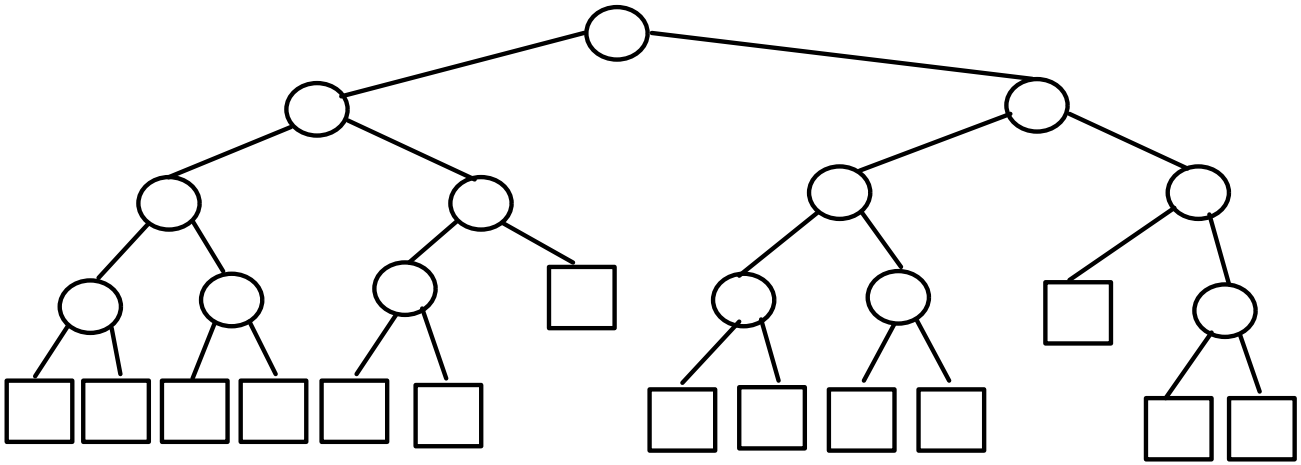
Gegeben: Menge von Datenbankpunkten
orthogonaler Anfragebereich P

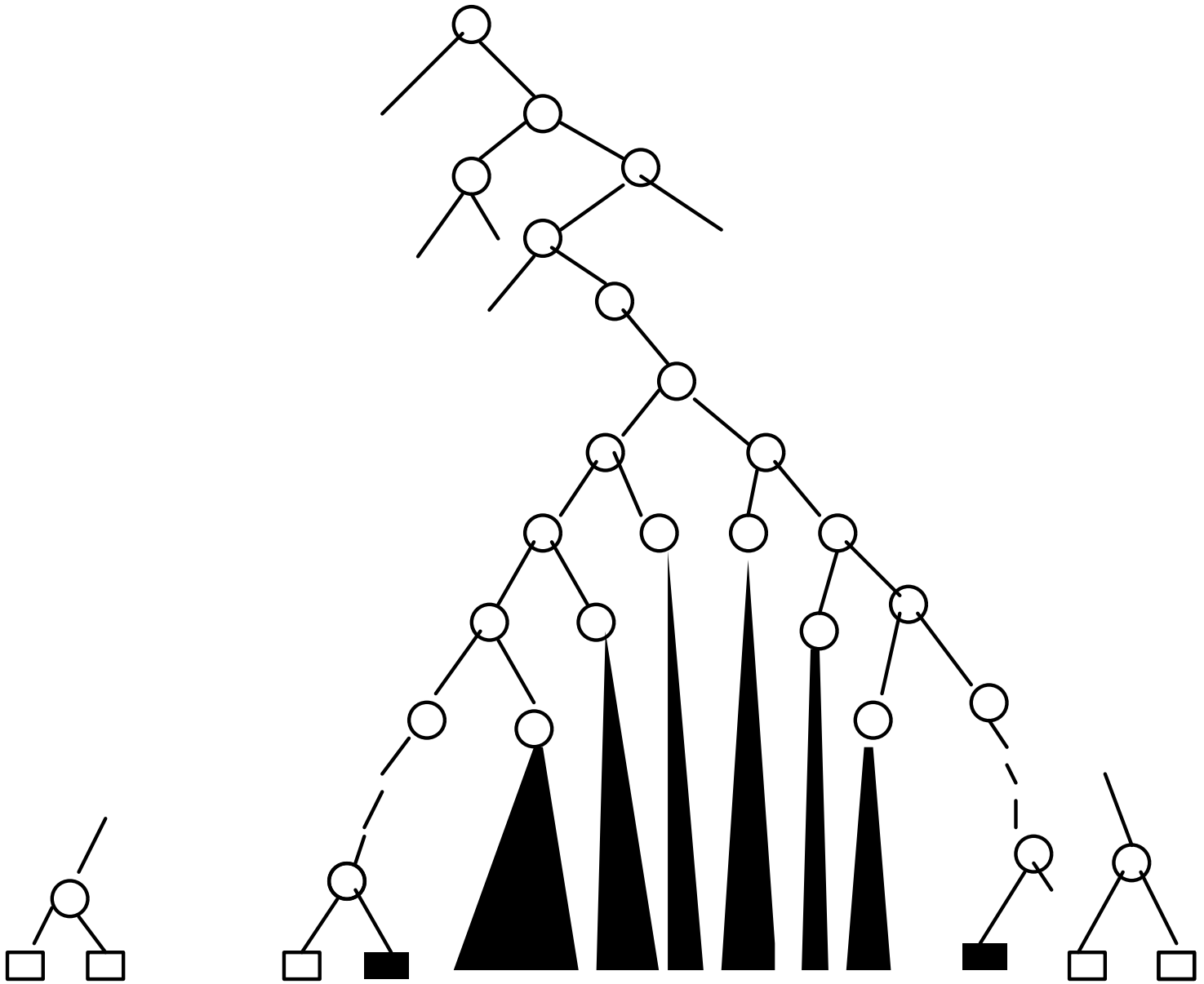
Gesucht: Knoten die in P liegen

Binärer Suchbaum (1-dim Fall)



Binärer Blattsuchbaum



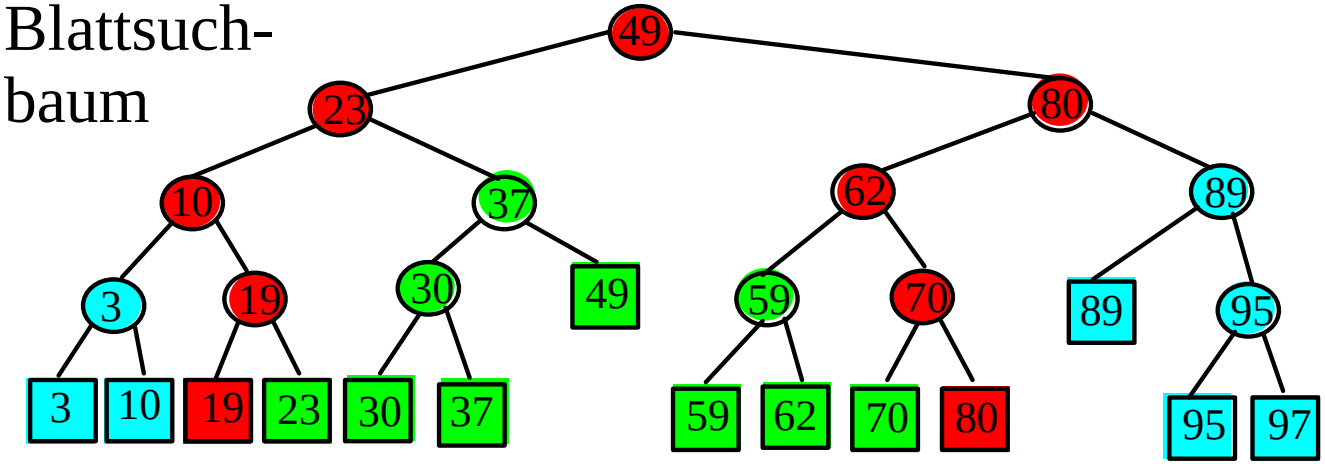


Eindimensionale Bereichssuche

binärer Blattsuch-baum

SplitNode

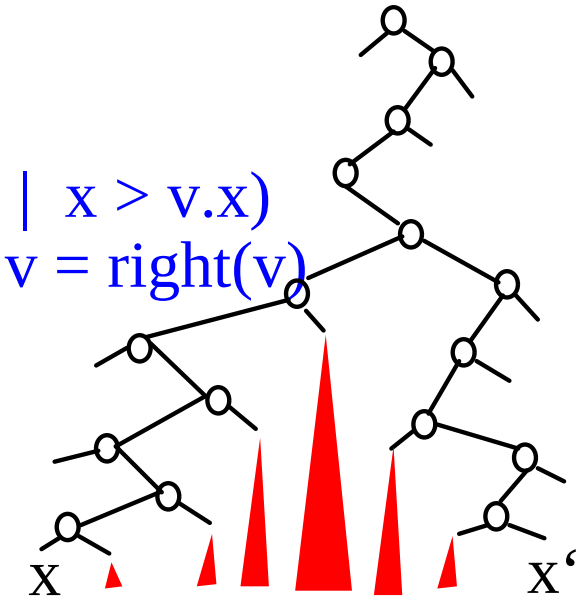
Anfrage: [18,77]




```
FindSplitNode(T,x,x')
    v=root(T)
    while not Leaf(v) & (x' <= v.x  |  x > v.x)
        if (x' <= v.x) v = left(v) else v = right(v)
    return v
```

Laufzeit: O(log n)

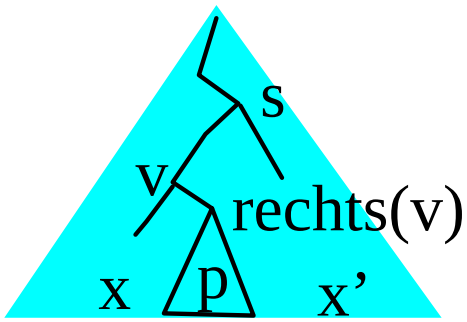
Beachte: O(log n) ausgewählte Teilbäume



Algorithmus eindimensionale Bereichssuche

1-dimRangeQuery (T,[x,x'])

```
v_split= FindSplitNode(T,[x,x'])
if (Leaf(v_split) & v_split in R) write v_split, return
v = lchild(v_split)
while (not Leaf(v))
    if (x <= v.x) write Subtree(rchild(v)), v = lchild(v)
    else v = rchild(v)
if (v in R) write v
v = rchild(v_split) ...
```

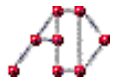


Satz: Laufzeit O(log n+k)n=#Punkte, k=#Ausgabepunkte

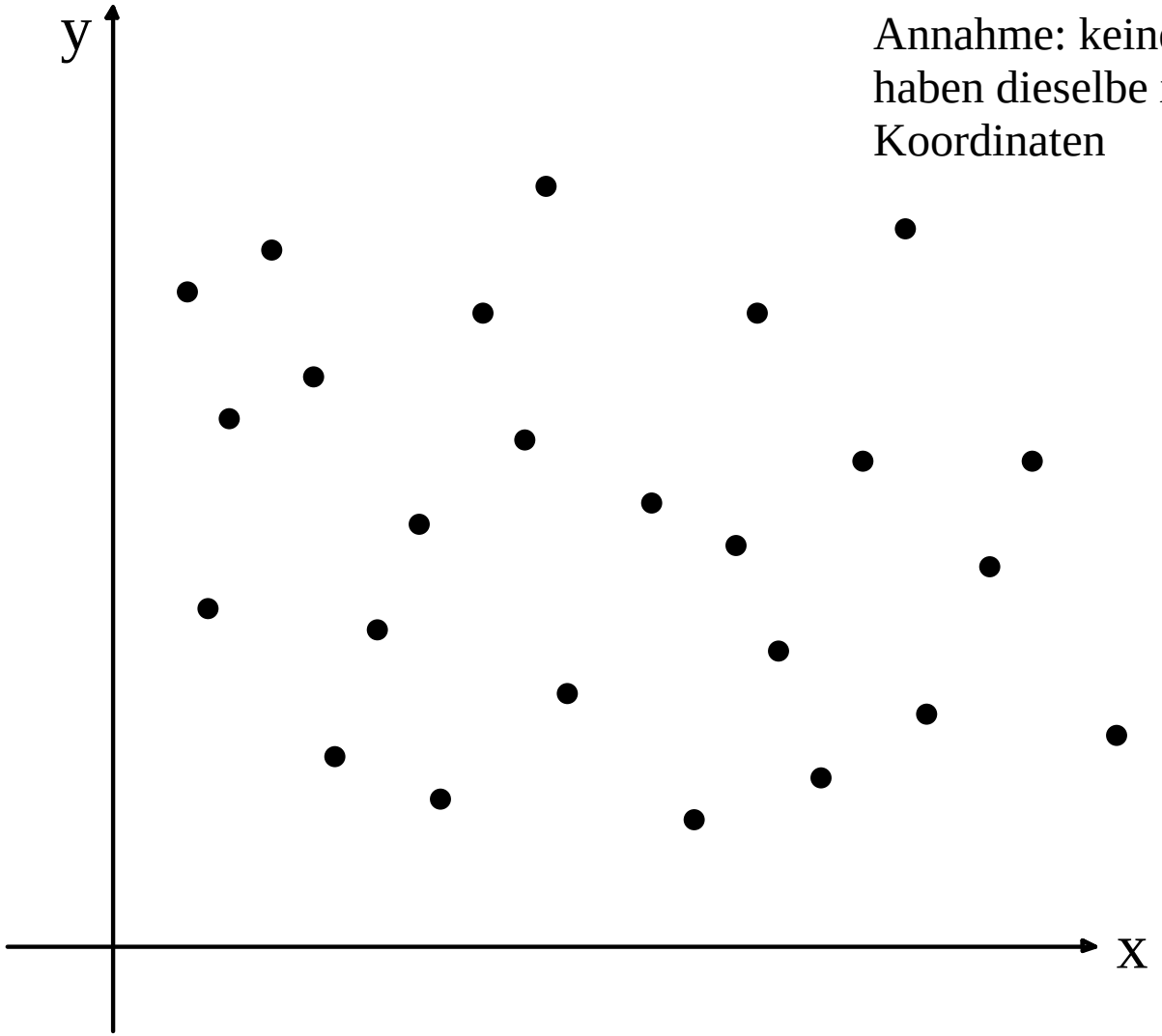
Bew: FindSplitNode: O(log n), Blattsuche O(log n)
#grüne Knoten O(k), da #innere Knoten in O(k)

Zusammenfassung:

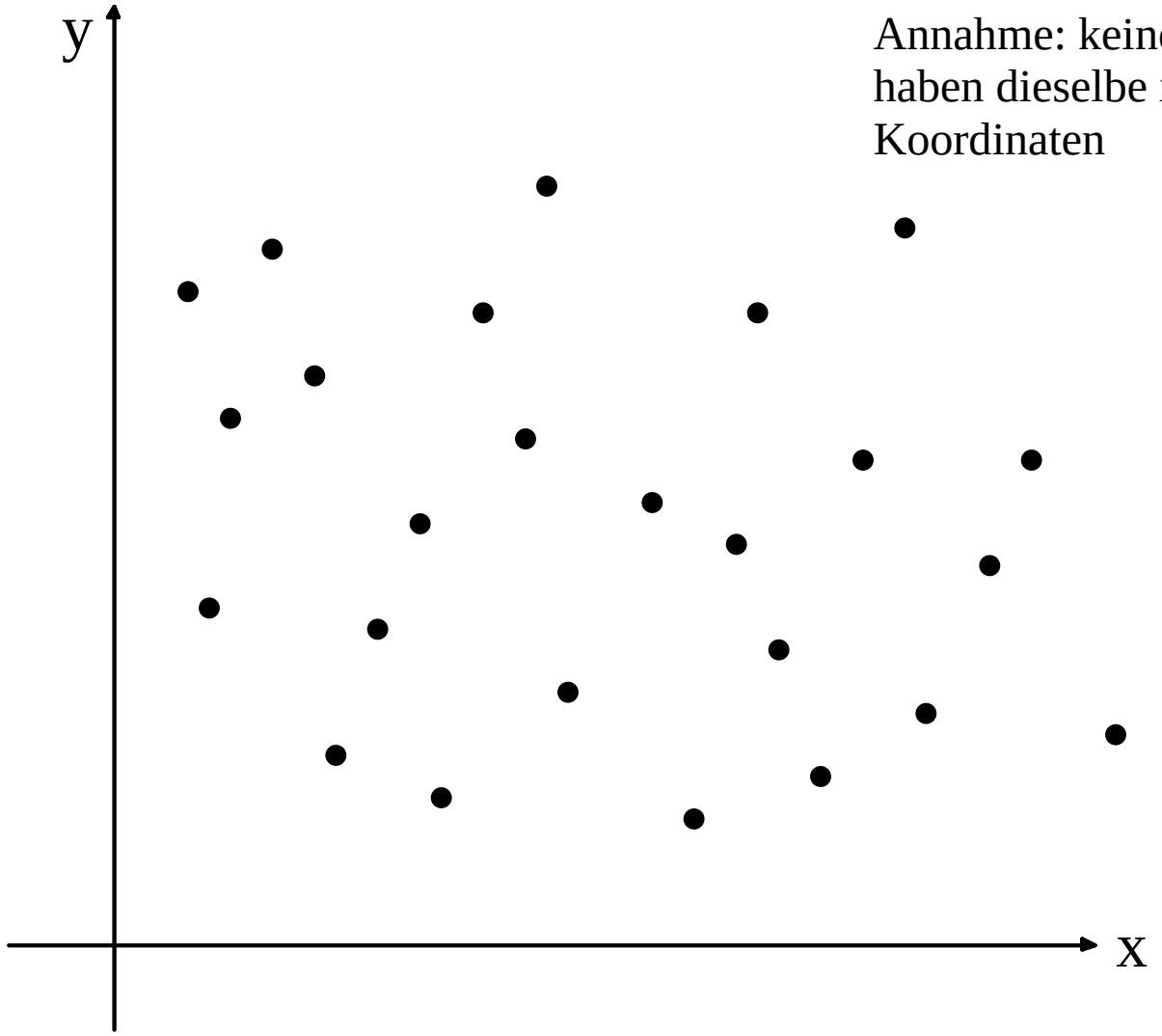
Sei P eine Menge von n Punkten im 1-dim Raum. Dann kann P in einem balancierten (Blatt-) Suchbaum gespeichert werden, der in Zeit O(n log n) und Platz O(n) konstruiert werden kann und so daß sich alle k in einem gegebenen Anfragebereich liegenden Punkte in Zeit O(log n + k) berichten lassen.



Zweidimensionale Bereichssuche

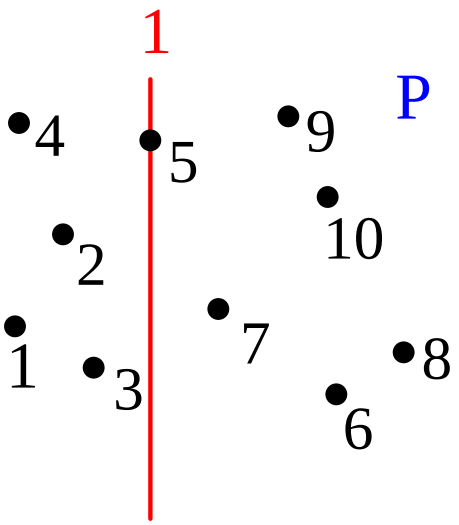


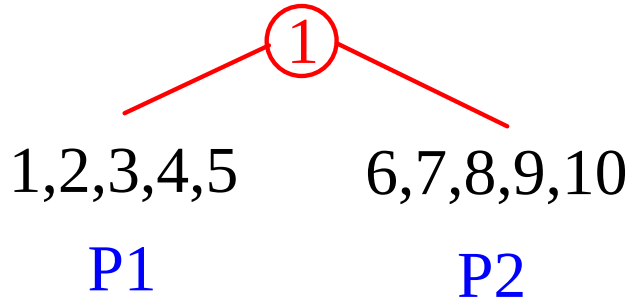
Annahme: keine zwei Punkte haben dieselbe x- oder y-Koordinaten

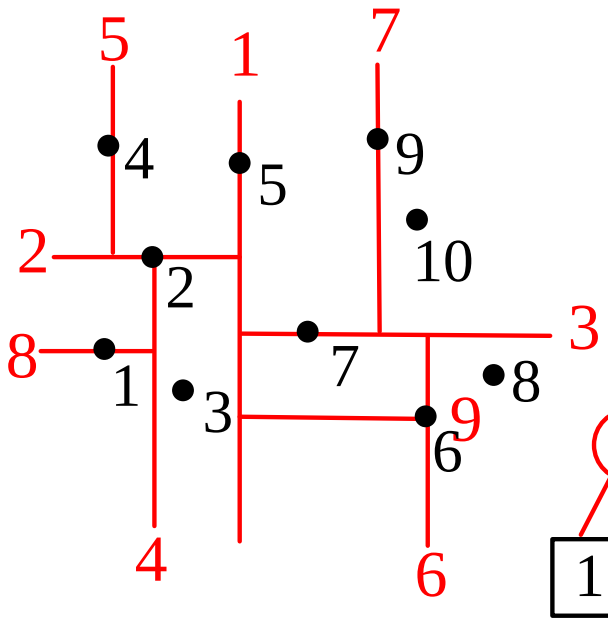


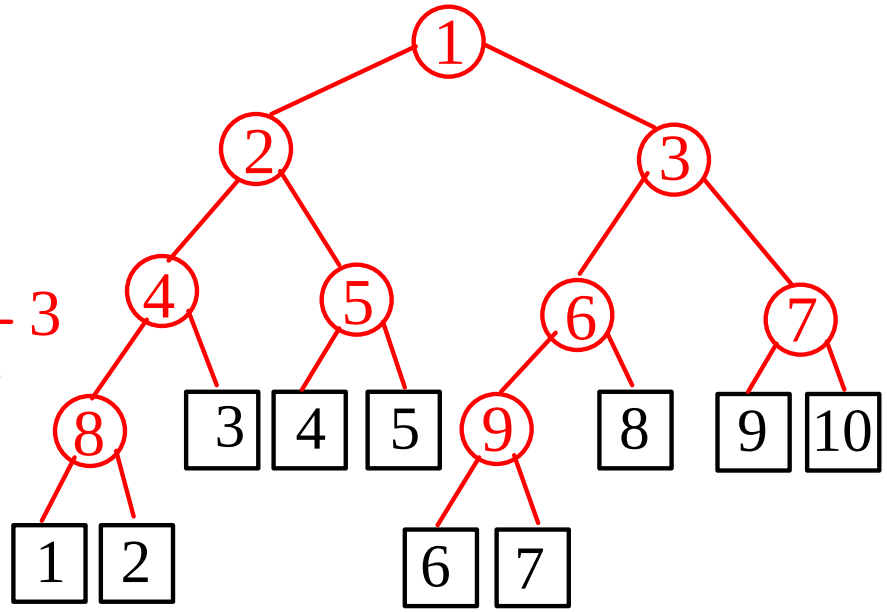
Annahme: keine zwei Punkte haben dieselbe x- oder y-Koordinaten

Konstruktion kd-Baum









Algorithmus Kd-Baum

BuildTree(P,depth)

```
if (|P| =1) return Leaf(p)
if (depth gerade) splitte P in P1,P2 via vertikalem Median
else splitte P in P1,P2 via horizontalem Median
v1=BuildTree(P1,depth+1)
v2=BuildTree(P2,depth+1)
return (v1,Median,v2)
```

Satz: O(n) Platz und O(n log n) Zeit

Bew: Platz: Binärbaum mit n Leaves.

Zeit: $T(n) = O(1), \quad n=1$

$= O(n)+2T(n/2), \quad n>1$

$T(n) <= cn + 2T(n/2)$

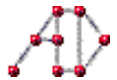
$<= cn + 2(cn/2+T(n/4))$

$<= 2cn + T(n/4)$

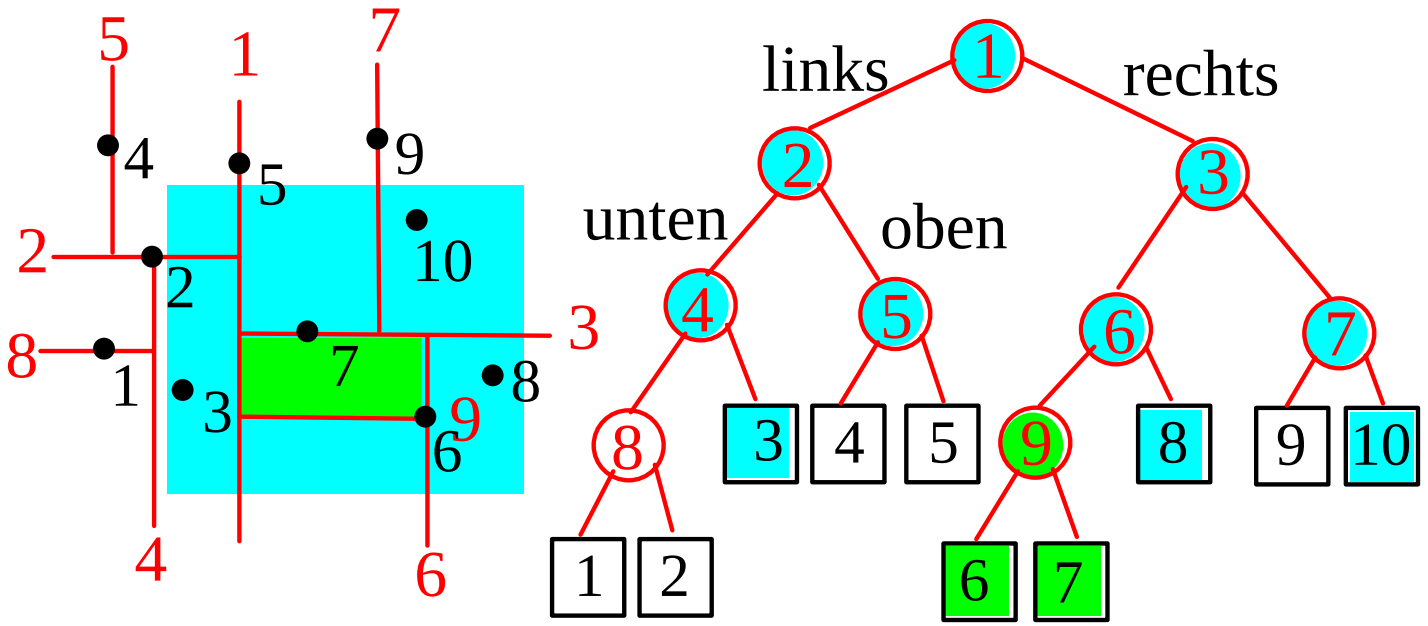
$= O(n \log n)$

Tiefe:

$O(n \log n)$



Suche in einem kd-Baum

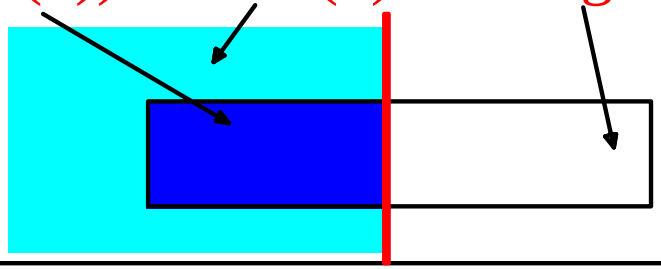


Regionen:

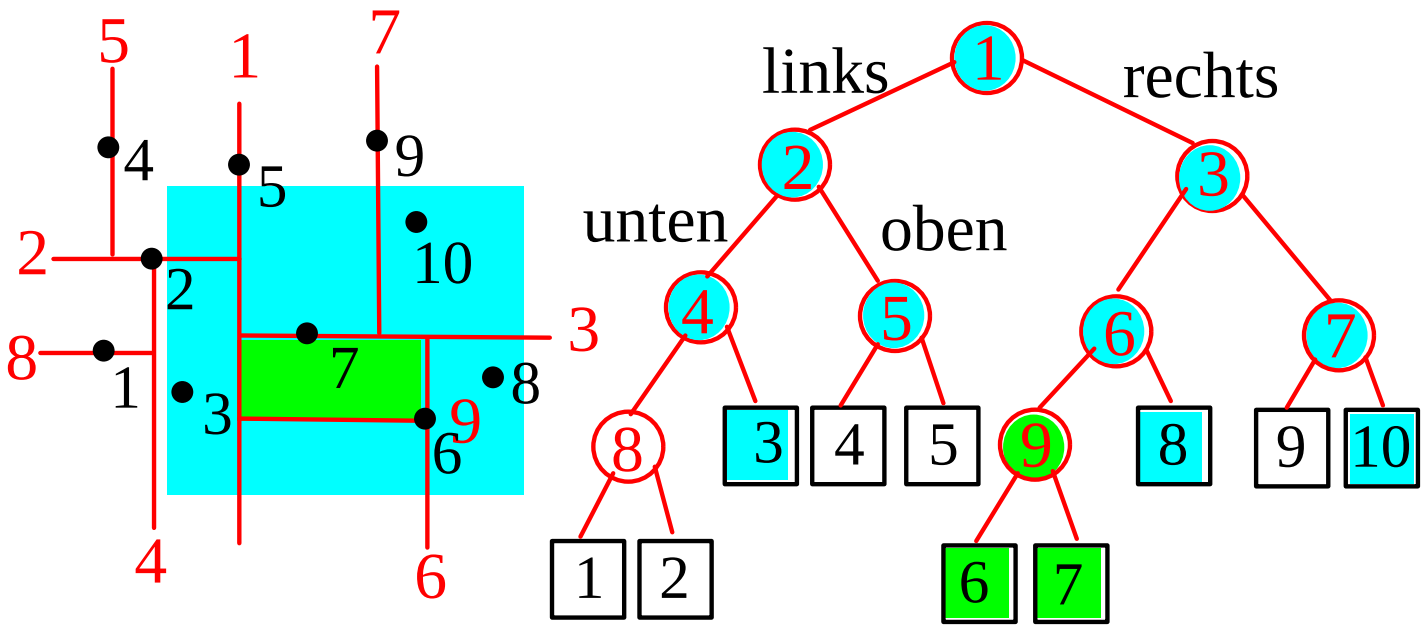
Region(5) ist links von 1 und oberhalb von 2

Inkrementell gilt (hier gerade Tiefe):

Region(left(v)) = links(v) ∩ Region(v)



Algorithmus Suche im 2-d-Baum



SearchTree(v,R)

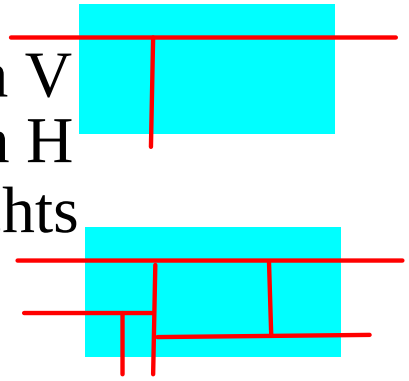
```
if (Leaf(v) & v in R) write v, return
if (Region(left(v)) in R) write Subtree(left(v)), return
if (Region(left(v)) ∩ R <> {}) SearchTree(left(v),R)
if (Region(right(v)) in R) write Subtree(right(v)), return
if (Region(right(v)) ∩ R <> {}) SearchTree(right(v),R)
```


Algorithmus Suche in Kd-Baum

Satz: Eine orthogonale Bereichsanfrage in einem kd-Baum mit n Punkten kann in Zeit $O(\sqrt{n} + k)$ beantwortet werden. (k=#Punkte im Anfragebereich)

Sei B = # blauer, G = # grüner Knoten

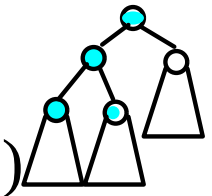
Bew: G(n)=O(k).

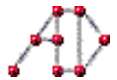


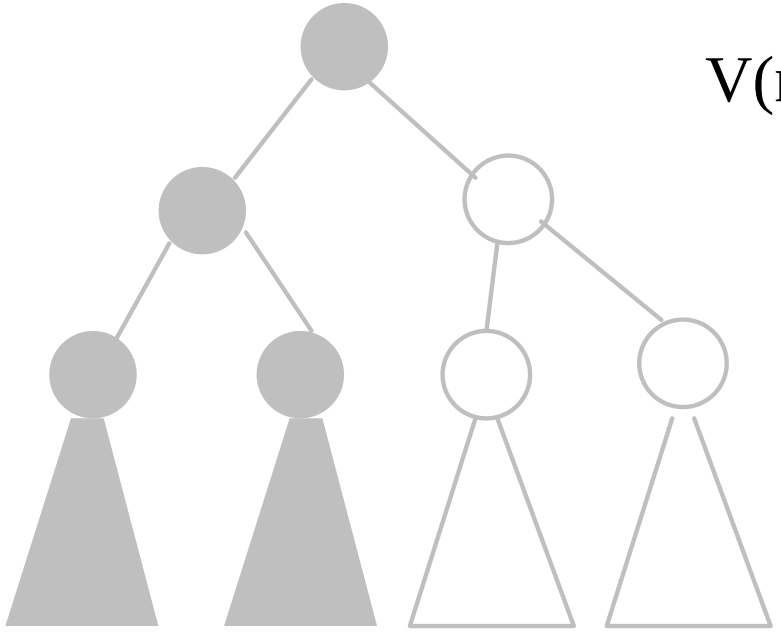
B(n) <= # vertikaler Schnittregionen V
+ # horizontaler Schnittregionen H
Bel. l schneidet root(T) links od. rechts
Es gilt die folgende Rekursion

V(n) = O(1), n=1
= 2+2V(n/4), n>1

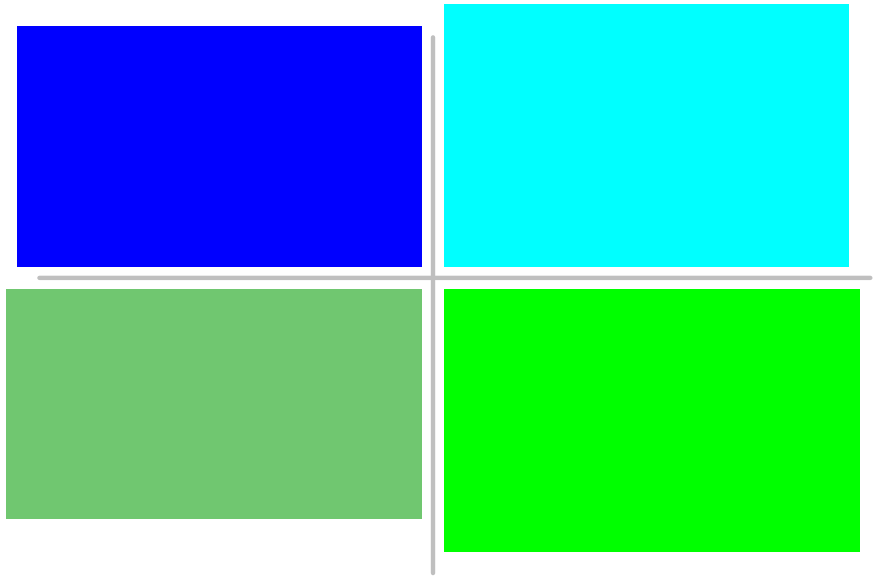
V(n) = 2 + 4 + 8 + 16 + ... + 2^log_4 n
= 2 + 4 + 8 + 16 + ... + $\sqrt{n}$ = O($\sqrt{n}$)







V(n)=#Regionen in k-d-Baum mit n Punkten, die von vertikaler Gerade geschnitten werden.

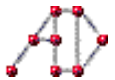


V(1)=1

V(n)=2+2V(n/4)

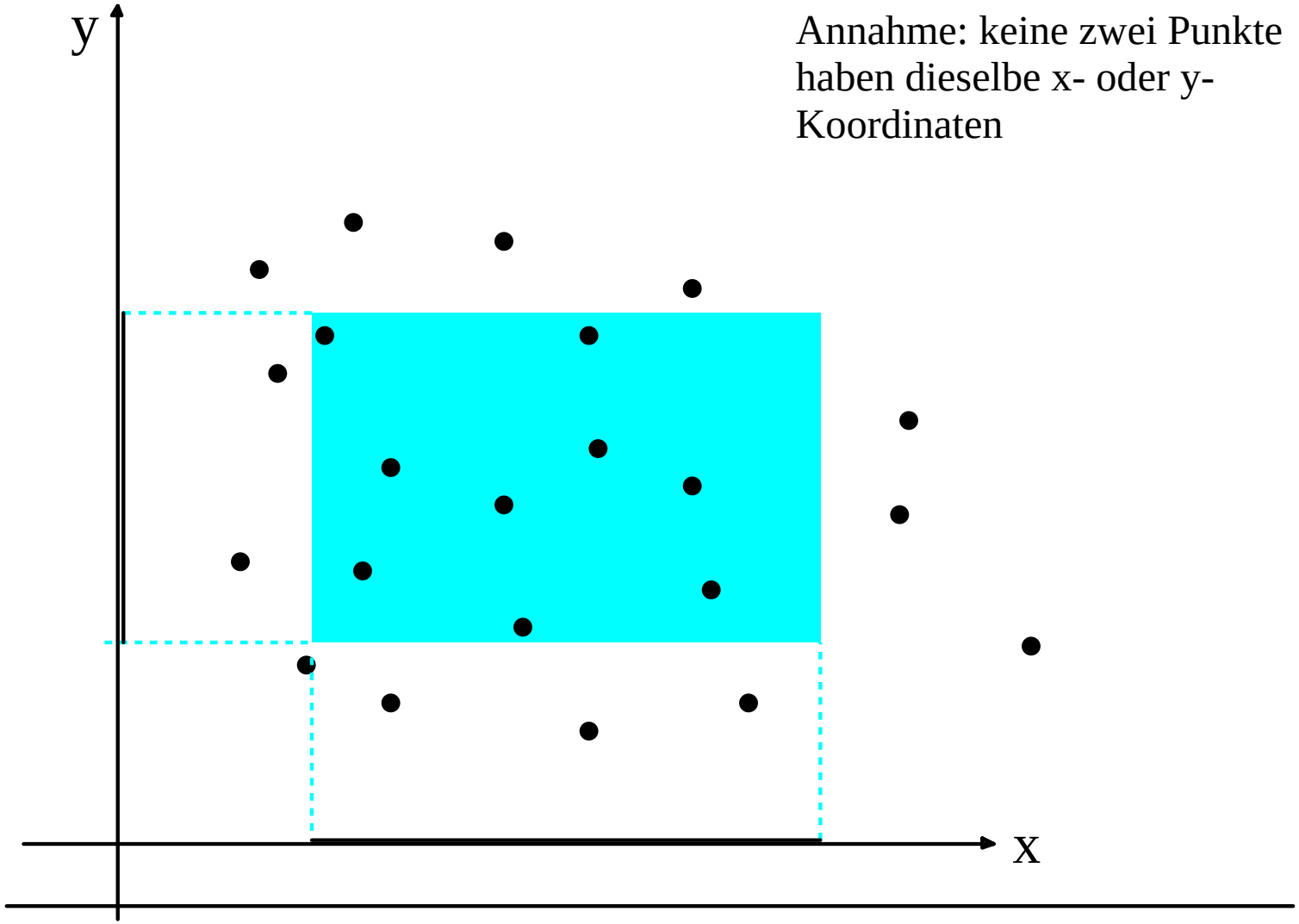
Zusammenfassung

Ein k-d-Baum zur Speicherung einer Menge von n Punkten in der Ebene kann in Zeit O(n log n) und Platz O(n) konstruiert werden. Für einen beliebig gegebenen Anfragebereich können alle im Bereich liegenden k Punkte in Zeit O(n lon n +k) berichtet werden.



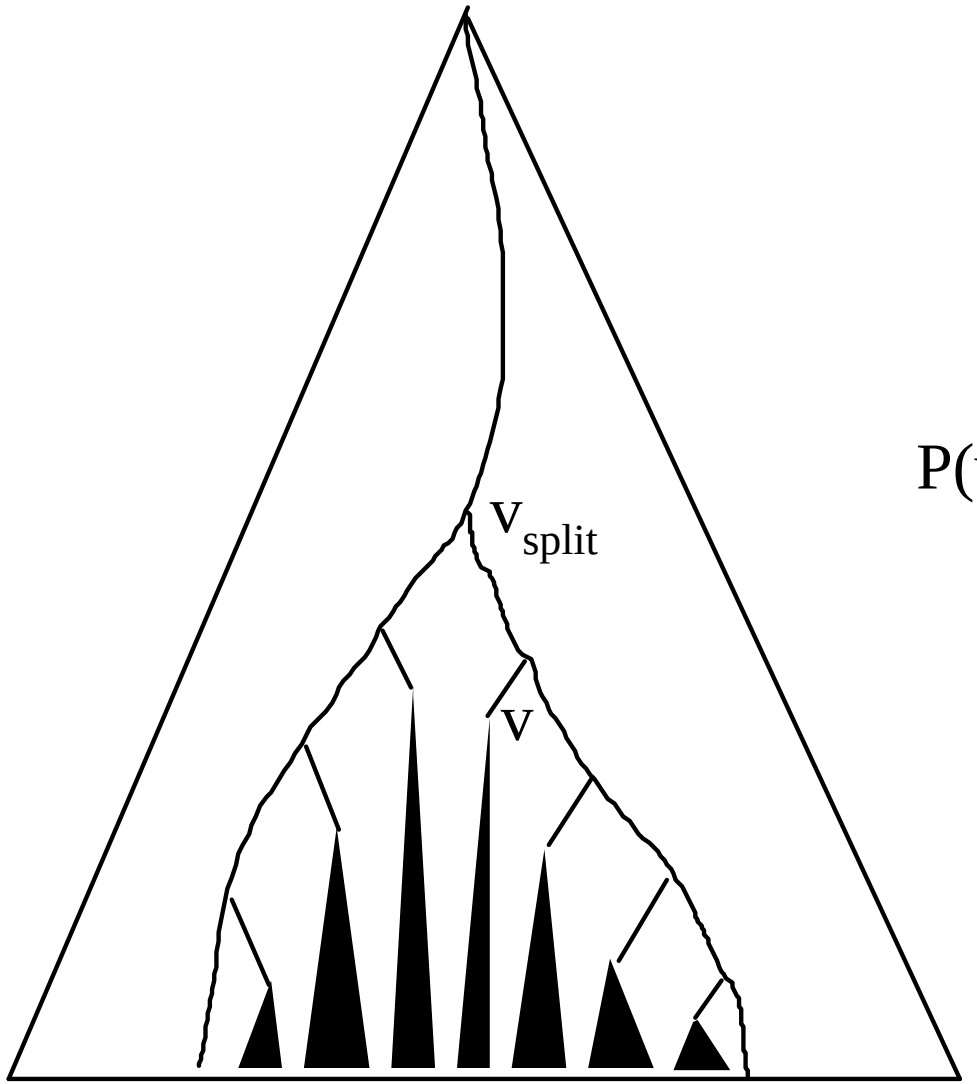
Bereichsbäume (Range Trees)

Zweidimensionale Bereichssuche



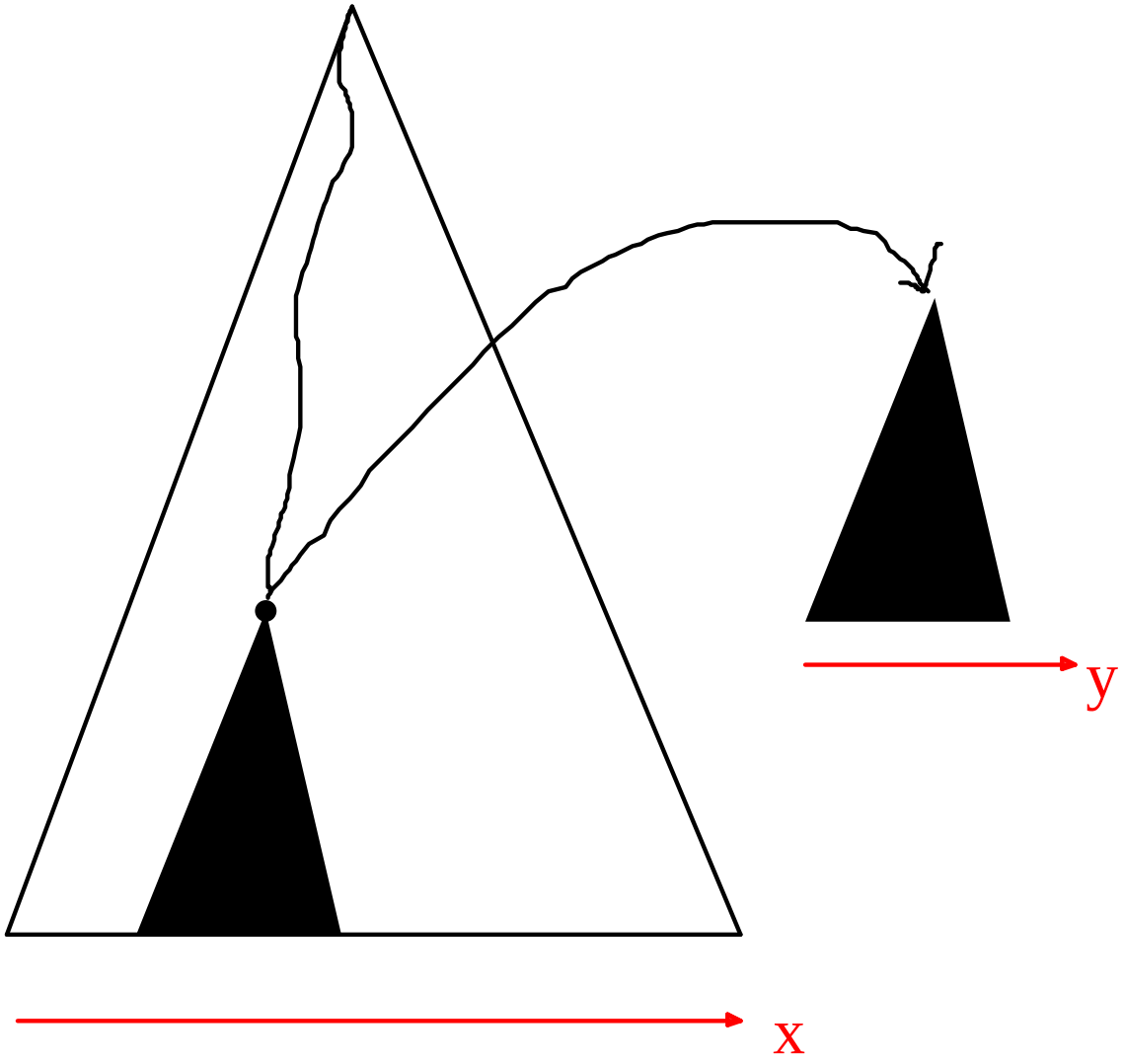
Annahme: keine zwei Punkte haben dieselbe x- oder y-Koordinaten

Kanonische Punktmenge eines Knotens



P(v)=Menge der Punkte im Teilbaum von v

assoziierter Baum



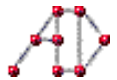
Bereichsbaum für Menge P

(1) Baum für die erste Schicht:

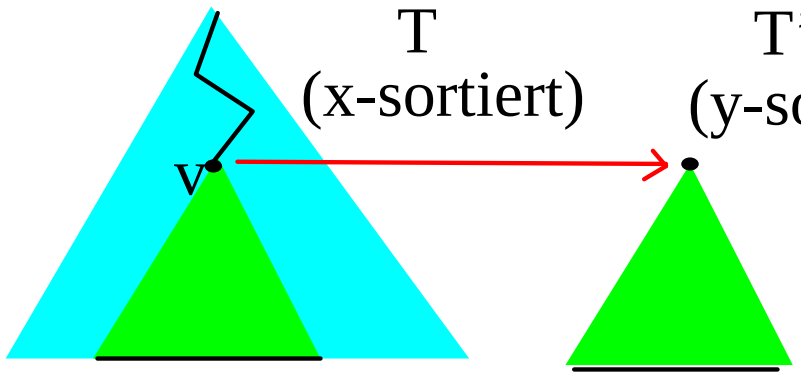
Balancierter Suchbaum T für die Punkte in P
in x-Reihenfolge

(2) Assoziierte Bäume auf der zweiten Schicht:

Für jeden Knoten v von T wird die kanonische Punktmenge von v in einem nach den y-Koordinaten sortierten balancierten Suchbaum $T_{assoc}(v)$ gespeichert.

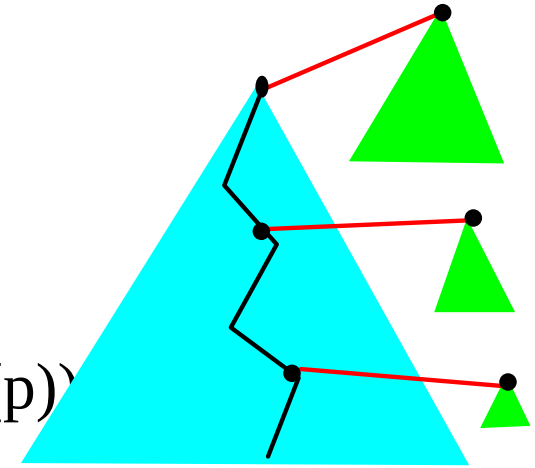


Konstruktion Bereichsbaum



T'(v) für alle Knoten in T
(y-sortiert) speichert gesamte
Punktinformation

Alle Knoten y-vorsortiert

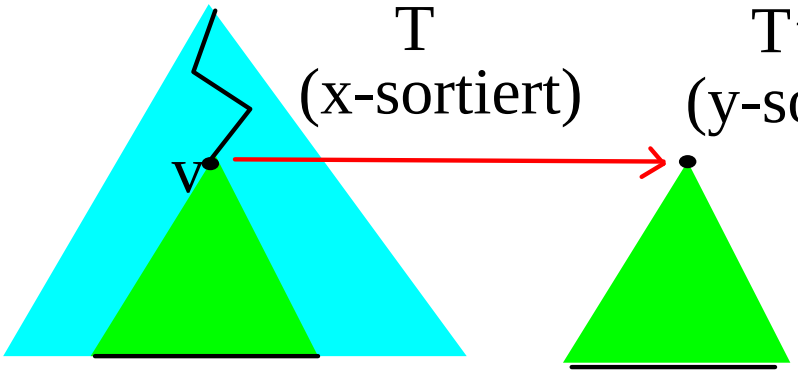


BuildRangeTree(P)

1. Konstruiere assoziierten Baum T'
   für die Wurzel von P
2. if (|P| = 1) return Leaf(p),T'(Leaf(p))

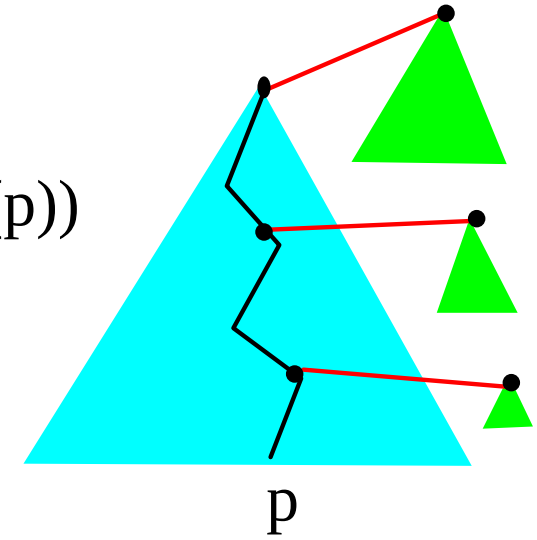
   sonst: Splitte P in P1, P2 via Median x
   v1 = BuildRangeTree(P1)
   v2 = BuildRangeTree(P2)
3. speichere x in, leftChild(v)=v1, rightChild(v)=v2,
   assoziiere T' mit v

Konstruktion Bereichsbaum



T'(v) für alle Knoten in T
(y-sortiert) speichert gesamte Punktinformation

Alle Knoten y-vorsortiert



BuildRangeTree(P)

if (|P| = 1) return Leaf(p),T'(Leaf(p))
Splitte P in P1, P2 via Median x
v1 = BuildRangeTree(P1)
v2 = BuildRangeTree(P2)
return (v1,x,v2),T'(v1,x,v2)

Satz: O(n log n) Platz und Zeit

Bew: Jeder Knoten in T ist maximal O(log n) mal in T'
=> Platz O(n log n)
Aufbau des binären Suchbaumes T' (bottom-up)
in Linearzeit, da y -vorsortiert => Zeit O(n log n)

Suche im Bereichsbaum

RangeQuery(T,[x,x']x[y,y'])

```
v_split = FindSplitNode(T,x,x')
if (Leaf(v_split) & v_split in R) write v, return
v = left(v_split)
while not (Leaf(v))
if (x<=v.x) 1-dim RangeQuery(T'(right(v)),[y,y'])
                v = left(v)
else v = right(v)
if (v in R) write v
v = right(v_split) ...
```

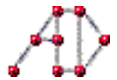
Satz: O(log^2 n + k) Zeit.

Bew: Die Zeit für die eindimensionale Suche an v ist durch O(log n + kv) beschränkt.

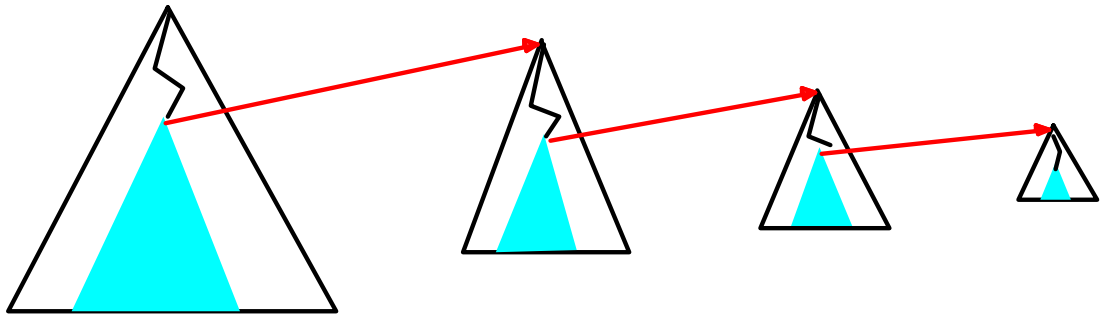
$k = \sum_v kv$ und Suchpfad zu x und x' $O(\log n)$ =>

$T(n) = \sum_v O(\log n + kv) = k + \sum_v O(\log n)$

$= O(\log^2 n + k)$



Höhere Dimensionen



Zeitbedarf für Konstruktion:

- $T(2,n) = O(n \log n)$

$T(d,n) = O(n \log n) + O(\log n)\, T(d-1,n)$

$\Rightarrow T(d,n) = O(n \log^{d-1} n)$

Zeitbedarf für Bereichsanfrage (ohne Ausgabe):

$Q(2,n) = O(\log^2 n)$

$Q(d,n) = O(\log n) + O(\log n)\, Q(d-1,n)$

$\Rightarrow Q(d,n) = O(\log^d n)$

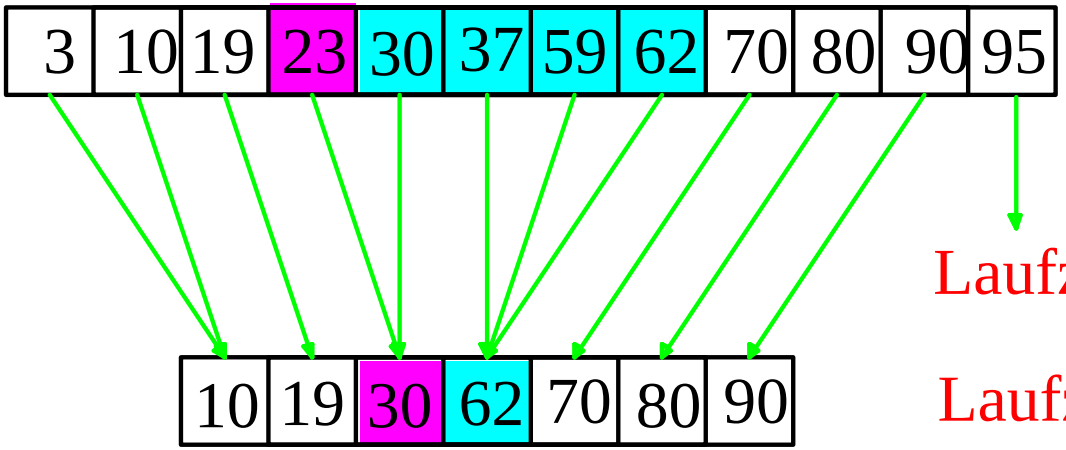
Suche in Untermengen

Gegeben: Zwei geordnete Arrays A1 und A2.
Schlüssel(A2)$\subset$ Schlüssel(A1)
Anfrage [x,x']

Gesucht: Alle Elemente e in A1 und A2 mit
x <= Schluessel(e) <= x'

Idee: Zeiger zwishen A1 und A2.

Beispielanfrage: [20:65]



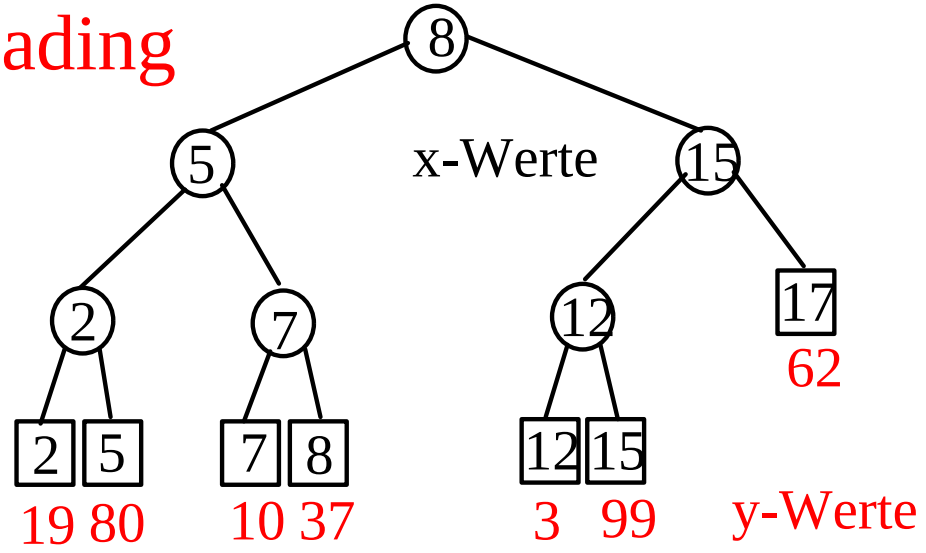
Laufzeit: O(log n+k)

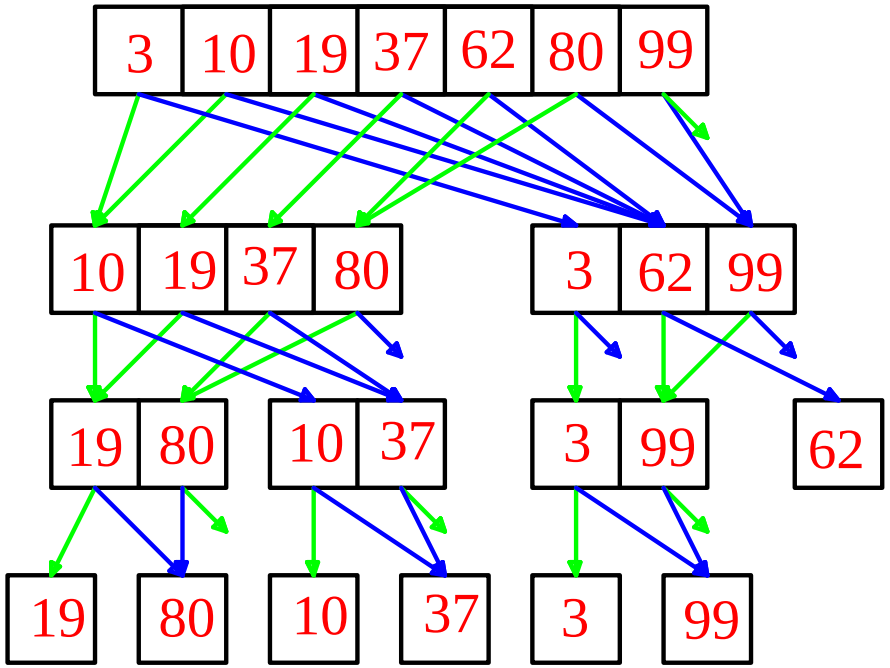
Laufzeit: O(1+k)

Fractional Cascading

Idee: $P1 \subset P$

$P2 \subset P$





Satz: Anfragezeit kann auf $O(\log n + k)$ gedrückt werden.

Bem: In d-Dimension Ersparnis um ein log-Faktor