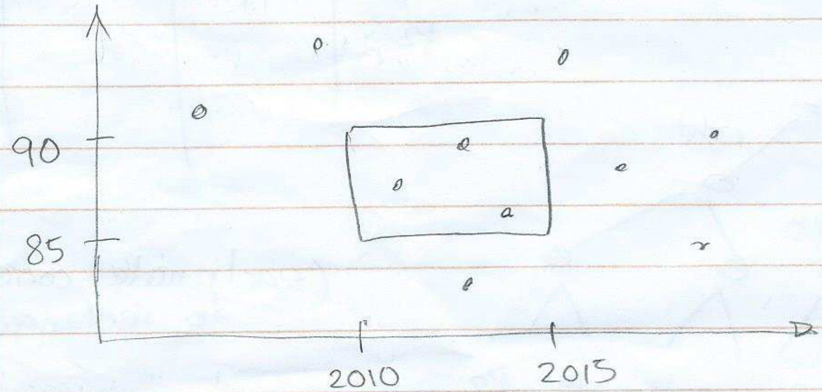
Orthogonal Range Searching

store n pts in $\mathbb{R}^d$

s.t. given query range q, find pts inside q

axis-aligned rectangle/box

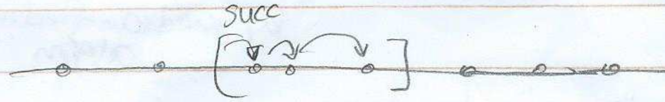


static case
(no insert/delete)

online queries
(not known in advance)

diff. versions – report all, decide emptiness, count,
sum of weights, max weight,
(more advanced: median weight, closest pair,
report all distinct colors, ...)

1D:



space $S(n) = O(n)$

preprocessing time $P(n) = O(n \log n)$ by sorting

query time $Q(n) = O(\log n)$ for emptiness

$O(\log n + k)$ for reporting
(k = output size)

$O(\log n)$ for counting

(optimal in comparison model, but can do better in RAM ...)

2D? query type

general (4-sided)

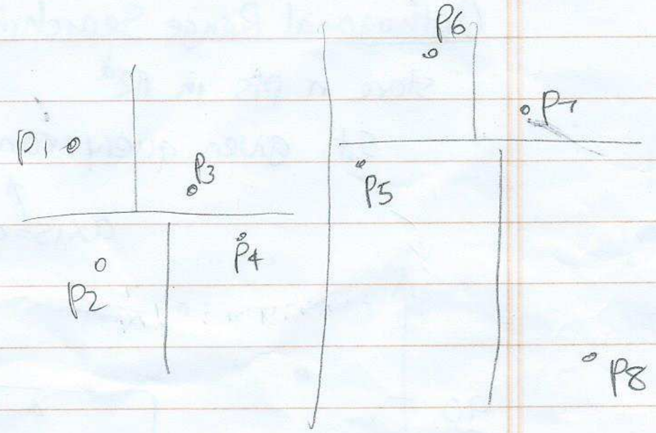
3-sided

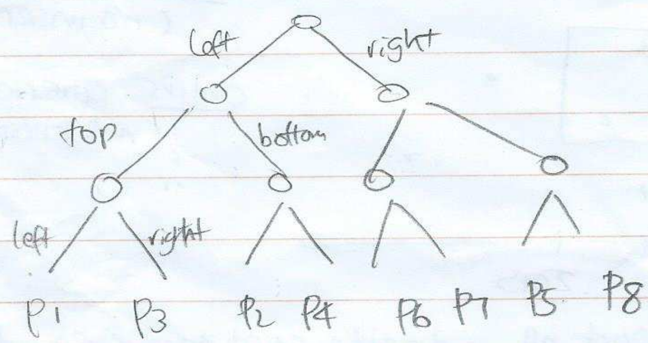
2-sided ("dominance")

1-sided

Method 0: "k-d Tree"

divide by median x
then median y, ...
alternate





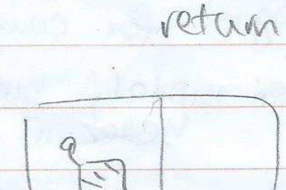
(each node corresponds to rectangular cell)

Space $S(n) = \boxed{O(n)}$

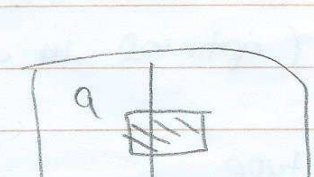
preproc $P(n) = 2P(\frac{n}{2}) + O(n) \Rightarrow \boxed{O(n \log n)}$

by median finding alg'm

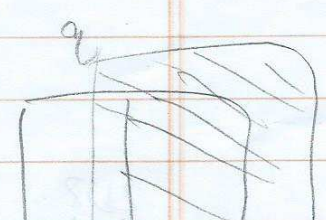
query alg'm, given rectangle q: // say, counting
 if q does not intersect node's cell
  return 0
 else if q completely contains cell
  return # pts in cell
 else recurse in both children
  return sum



recurse left

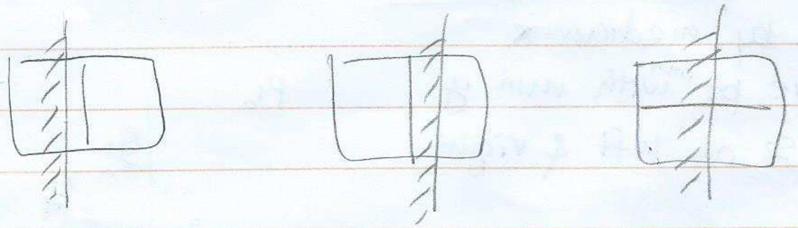


recurse on both



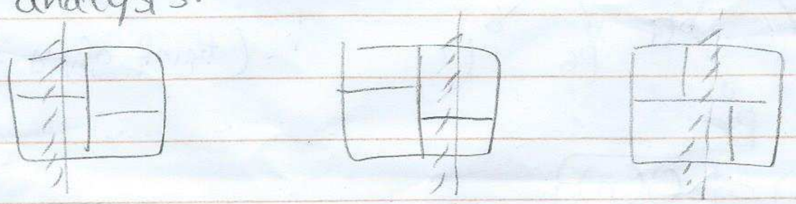
recurse on left

Analysis for 1-sided query:



$$Q_1(n) \le 2\,Q_1\left(\tfrac{n}{2}\right) + O(1) \Rightarrow O(n) \text{ bad!}$$

better analysis:



$$Q_1(n) \le 2\,Q_1\left(\tfrac{n}{4}\right) + O(1)$$

$$\Rightarrow O\left(n^{\log_4 2}\right) = \boxed{O(\sqrt{n})}$$

<u>Cor</u> any horizontal/vertical line intersects $O(\sqrt{n})$ cells of k-d tree.

Analysis for general 4-sided query:

$$Q(n) = O(\#\text{ cells visited})$$

$$= O(\#\text{ cells crossing } \partial q)$$

boundary has 4 line segs.

$$\le 4 \cdot O(\sqrt{n}) = \boxed{O(\sqrt{n})}$$

(+k for reporting)

Higher-d: $Q_1(n) \le 2^{d-1}\,Q_1\left(\frac{n}{2^d}\right) + O(1)$

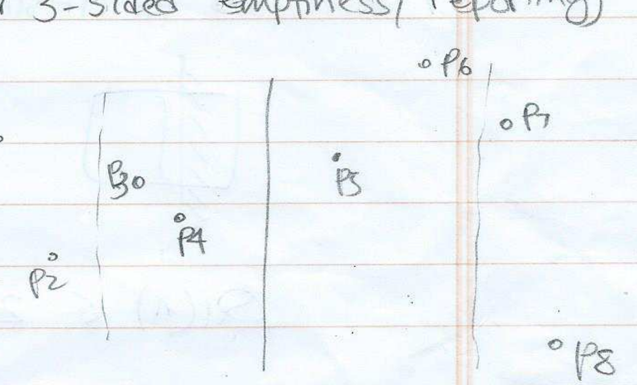
$$\Rightarrow O\left(n^{\frac{d-1}{d}}\right) = \boxed{O\left(n^{1-1/d}\right)}$$

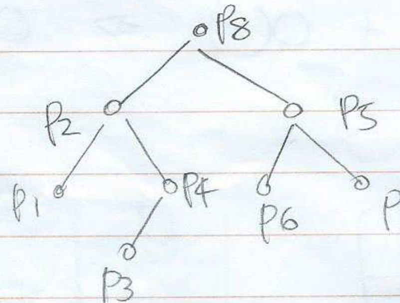
Method 1: Priority Search Tree (for 3-sided emptiness/ reporting)

divide by median x

remove pt p_{min} with min y

recurse on left & right





(think of y as "priority" → heap)

$S(n) = \boxed{O(n)}$

$P(n) = 2P(n/2) + O(n) \Rightarrow \boxed{O(n \log n)}$

query alg'm, given 3-sided rectangle q: (unbd'd from below)

if (q is below p_{min}) return

if (p_{min} is in q) report p_{min}

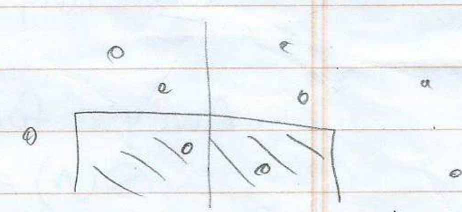
if (q left of median x)

recurse left

else if (q right of median x)

recurse right

else recurse on both

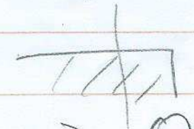


Analysis: for 1-sided, unbdd from below

$Q_1(n) = O(1)$ for emptiness

(+k for reporting, since each node visited reports 1 pt)

for 2-sided



$$Q_2(n) = \begin{cases} Q_2(\frac{n}{2}) + O(1) \\ Q_2(\frac{n}{2}) + \overset{O(1)}{\cancel{Q_1(1)}} + O(1) \end{cases} \Rightarrow O(\log n)$$

for 3-sided:

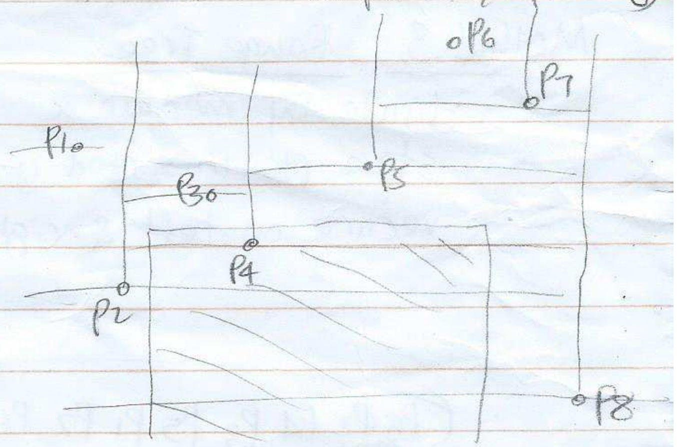
$$Q_3(n) \le \max \begin{cases} Q_3(\frac{n}{2}) + O(1) \\ 2\, \underbrace{Q_2(\tfrac{n}{2})}_{O(\log n)} + O(1) \end{cases}$$

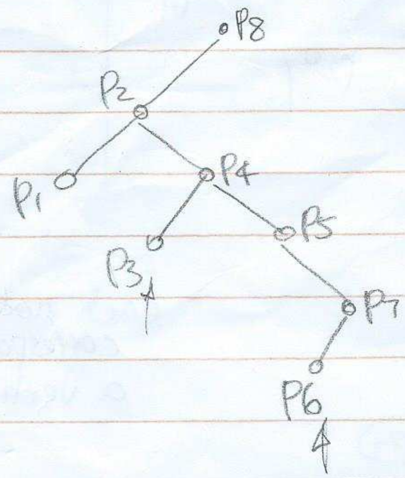
$$\Rightarrow \boxed{O(\log n)}$$

(+k for reporting)

Method 2: Cartesian Tree (also for 3-sided emptiness/reporting)

remove pt p_{min} with min y
divide by x at p_{min}
recurse on left & right





tree is not balanced!

$S(n) = O(n)$

$P(n) = O(n \log n)$ how?

query alg'm:

to report one pt insided 3-sided rectange q:

let p_i = x-successor of q's left side

p_j = x-predecessor of q's right side

find lowest common ancestor (LCA) of p_i, p_j

solvable in O(1) time for any binary tree (e.g. Bender & Farach-Colton '00) to report all pts inside q:

find one

recurse left & right

$\Rightarrow$ $\boxed{O(\boxed{1+k})}$ time if given succ/pred

$\boxed{O(\log n + k)}$ time else

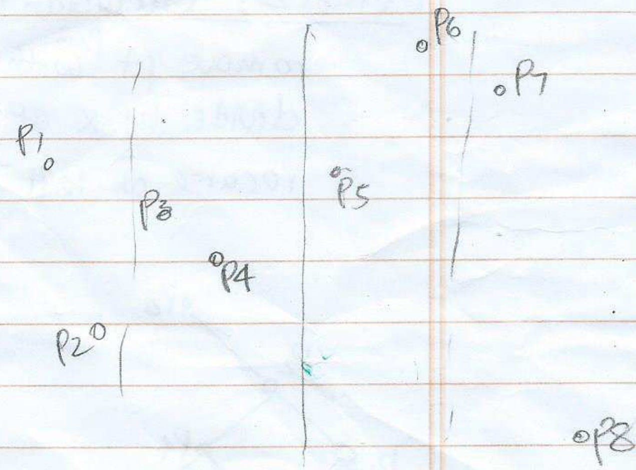
[note - doesn't generalize to counting or 4-sided ...]

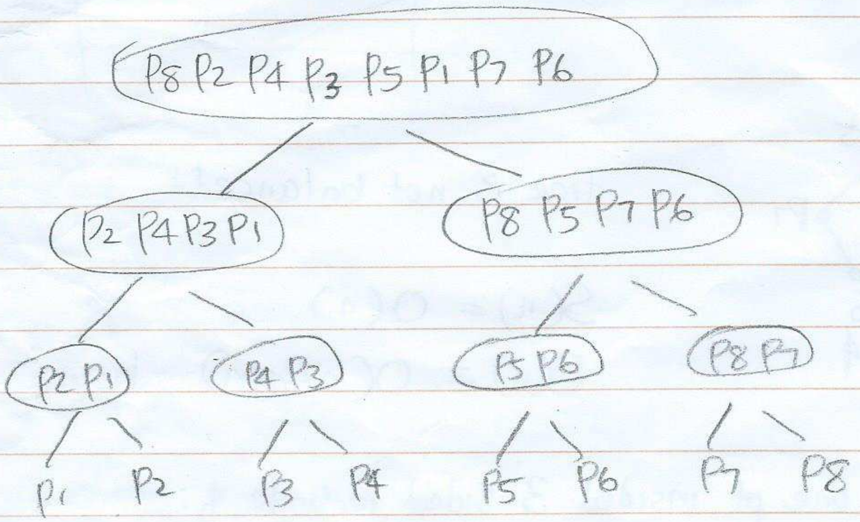
<u>Method 3</u>: <u>Range Tree</u>

divide by median x

store pts in sorted y-order

recurse on left & right





each node corresponds to a vertical <u>slab</u>

$S(n) = 2S(\frac{n}{2}) + O(n) \Rightarrow \boxed{O(n\log n)}$

$P(n) = 2P(\frac{n}{2}) + O(n\log n) \Rightarrow \boxed{O(n\log^2 n)}$

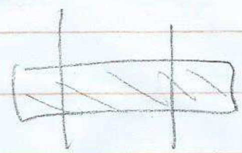
reduce to $O(n)$ by pre-sorting

query alg'm, given rectangle q: // say, counting

if (q doesn't intersect node's slab σ) return 0

if (q cuts completely across σ)

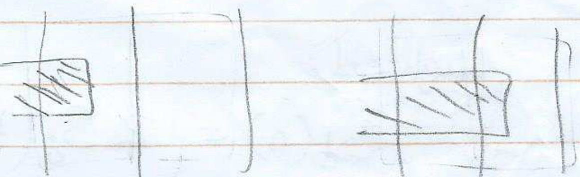
do binary search on y-sorted list



recurse on both children

return sum

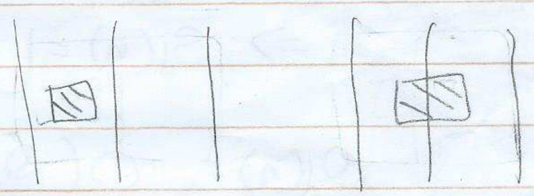
Analysis for 3-sided:



$$Q_3(n) \le Q_3\left(\frac{n}{2}\right) + \underbrace{O(\log n)}_{\text{binary search}}$$

$$\Rightarrow O(\log^2 n)$$

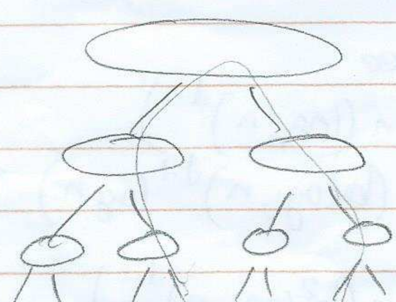
Analysis for 4-sided:



$$Q_4(n) \le \max \begin{cases} Q_4(n/2) + O(1) \\ 2\underbrace{Q_3(n/2)}_{O(\log^2 n)} + O(1) \end{cases}$$

$$\Rightarrow O(\log n + \log^2 n) = \boxed{O(\log^2 n)}$$

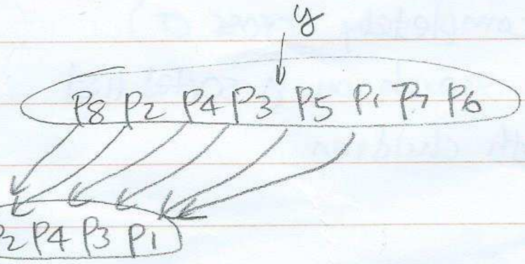
(+k for reporting)



binary search at $\le 2\log n$ nodes

$$\Rightarrow O(\log^2 n)$$

query time can be reduced to $\boxed{O(\log n)}$ (+k for report)
by adding pointers between parent list & child list



only need one initial binary search at root

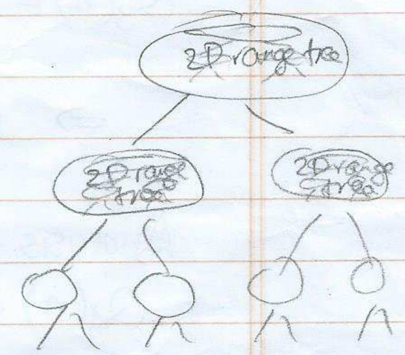
Higher d:

$$S_d(n) = 2\,S_d\left(\frac{n}{2}\right) + S_{d-1}(n)$$

$$\Rightarrow S_d(n) = O(S_{d-1}(n)\log n)$$

$$\Rightarrow S_d(n) = \boxed{O(n \log^{d-1} n)}$$

3D rangetree



"multi-level" data structure

$$Q_d(n) = O(Q_{d-1}(n) \log n)$$

$$\Rightarrow Q_d(n) = \boxed{O(\log^{d-1} n)} \quad (+k)$$

[lower bd in semigroup model, ptr machine model, ...]

Trade-offs:

degree-b range tree

$$S(n) = O(n(\log_b n)^{d-1})$$

$$Q(n) = O((b \log_b n)^{d-1} \log n)$$

e.g. $b = n^{\varepsilon/d}$

$$S(n) = O(n)$$

$$Q(n) = O(n^{\varepsilon})$$

or

$$S(n) = O(n(b^2 \log_b n)^{d-1})$$

$$Q(n) = O((\log_b n)^{d-1} \log n)$$

$b = n^{\varepsilon/2d}$

$$S(n) = O(n^{1+\varepsilon})$$

$$Q(n) = O(\log n)$$