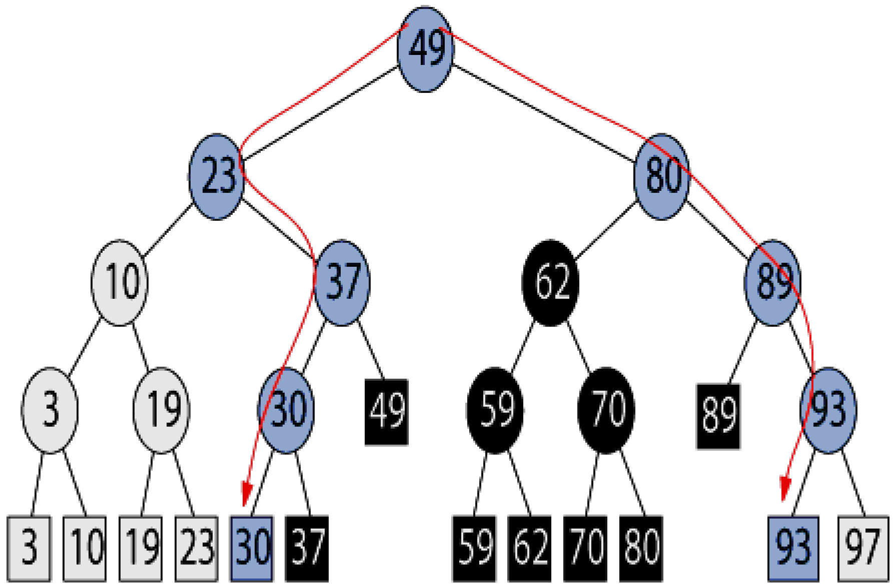
Orthogonal Range Searching

Part 2

Last section:

Kd-tree ⟶ $O(\sqrt{n} + k)$

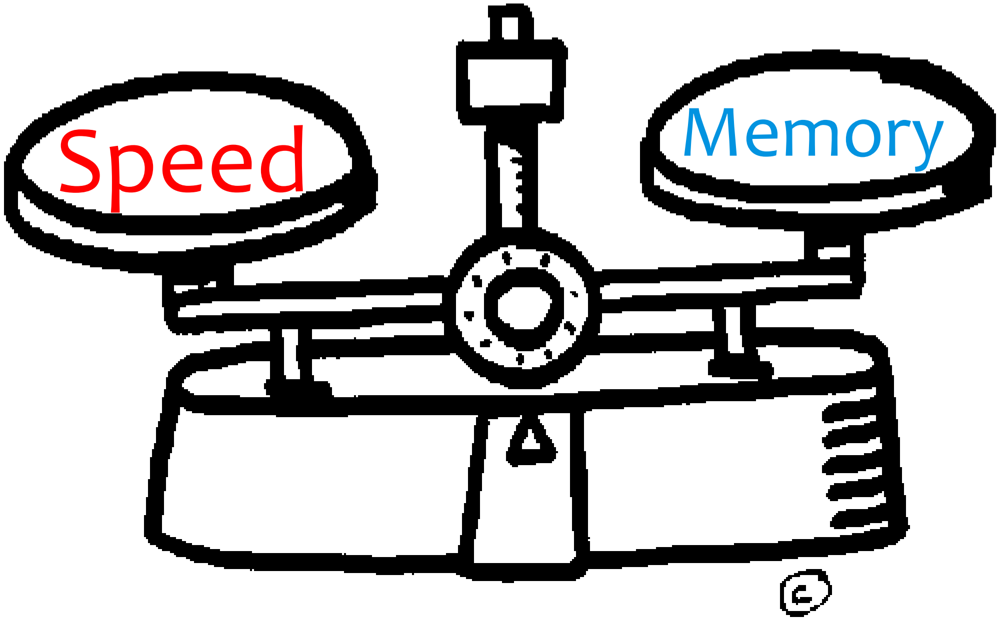


Search for a faster algorithm

Range tree → $O(log^2 n + k)$

Faster than Kd-tree but uses more storage:

$O(n) \rightarrow O(nlogn)$

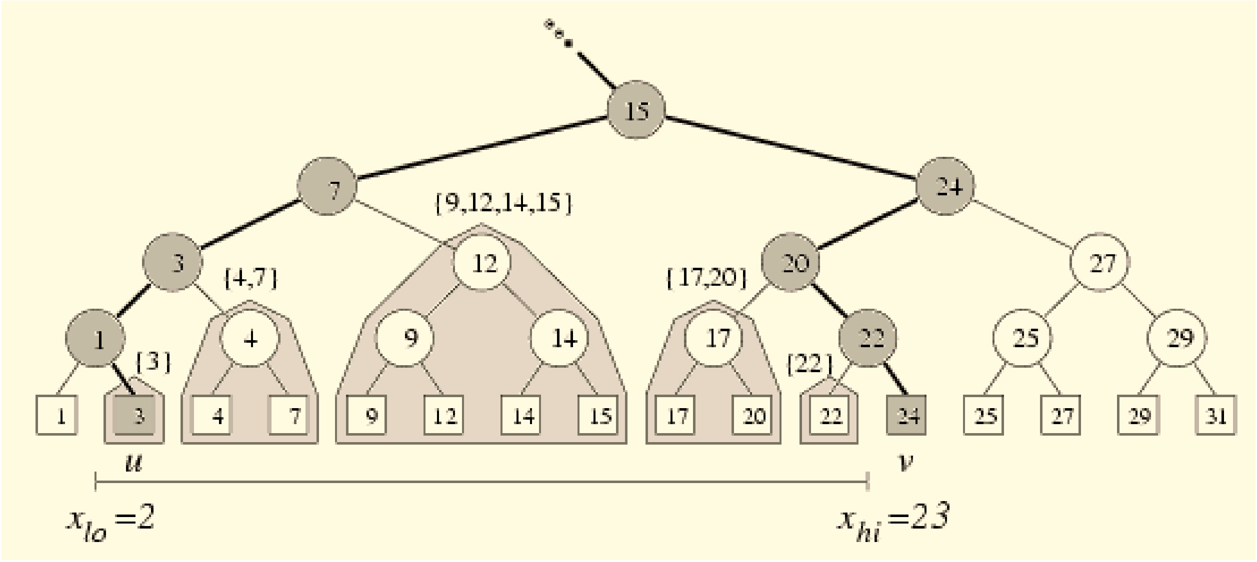


General schema:

$$Query\ range: [x: x'] \times [y: y']$$

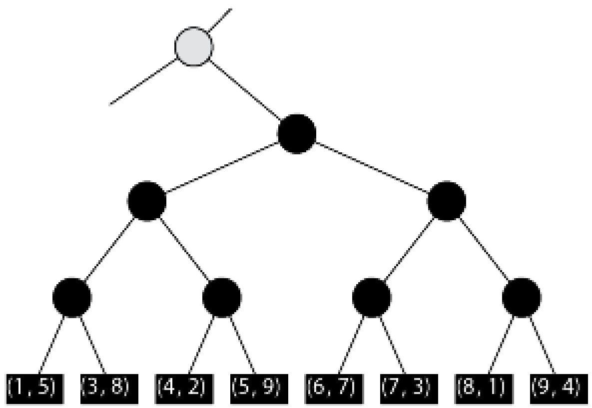
First a binary search tree on $[x: x']$ then we will care about y-component.

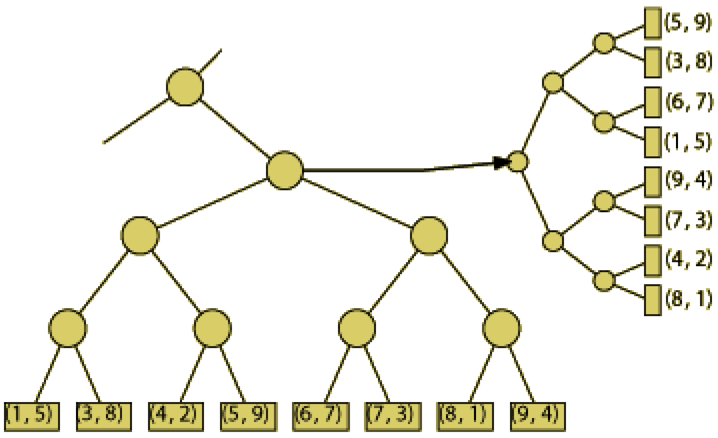
Canonical subset of *v*: Subset of points P at leaves of subtree of node *v: P(v)* : $O(logn)$



Points of P that lie in $[x: x']$ = ∪ disjoint $P(v)$

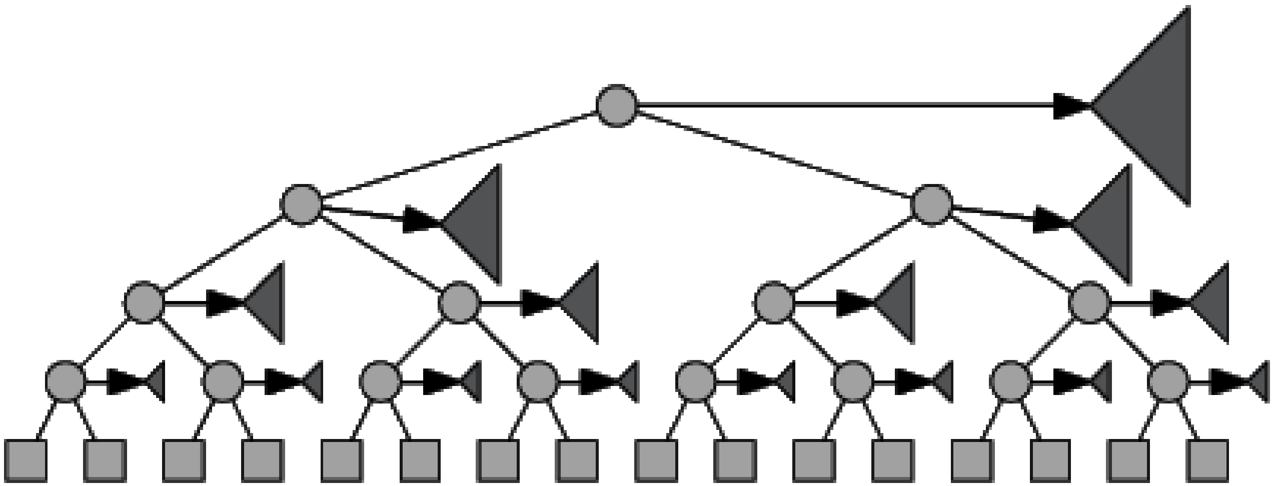
Those points laying in $[y: y']$ are important.





Every internal node stores a whole tree in an associated structure, on “y” component

Range Tree:



Construction

Algorithm BUILD2DRANGETREE(P)

1. Construct the associated structure: Build a binary search tree $\mathcal{T}_{\text{assoc}}$ on the set P_y of y-coordinates in P
2. **if** P contains only one point
3. **then** Create a leaf ν storing this point, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of ν.
4. **else** Split P into P_{left} and P_{right}, the subsets $\leq$ and $>$ the median x-coordinate x_{mid}
5. $\nu_{\text{left}} \leftarrow$ BUILD2DRANGETREE(P_{left})
6. $\nu_{\text{right}} \leftarrow$ BUILD2DRANGETREE(P_{right})
7. Create a node ν storing x_{mid}, make ν_{left} the left child of ν, make ν_{right} the right child of ν, and make $\mathcal{T}_{\text{assoc}}$ the associated structure of ν
8. **return** ν

Lemma 5.6:
A range tree on a set of n points in the plane requires $O(nlogn)$ storage.

Proof: 2 arguments

1. By level: on each level, any point is stored exactly once. So all associated trees on <u>one level</u> together have $O(n)$ size. Because the tree is of depth n, the storage will be $O(nlogn)$.
2. By point: for any point, it is stored in the associated structures of its search path. The depth of T is $O(logn)$. So it is stored in $O(logn)$ of them. As there are n points, the storage will be $O(nlogn)$.

Therefore the total amount of storage required is $O(nlogn)$.

The construction algorithm takes $O(nlog^2n)$ time

Lines 2 & 3

$$T(1) = O(1)$$

$$T(n) = 2.T\left({}^{n}/_{2}\right) + O(nlogn)$$

Line 1 : $O(nlogn)$
Lines 4 to 8:
Recursive relation

Which solves to $O(nlog^2n)$ time.

Suppose we pre-sort P on y component and whenever we split P into P_{Left} & P_{Right}, we keep the y-order.

For this sorted set we can build the associated structure in linear time.

Construction of Balanced Binary Search Tree on Sorted List takes $O(n)$ time, Since we can find the median in $O(1)$ time.

Therefore the adapted algorithm takes $O(nlogn)$ time, as sorting will take $O(nlogn)$ time either.

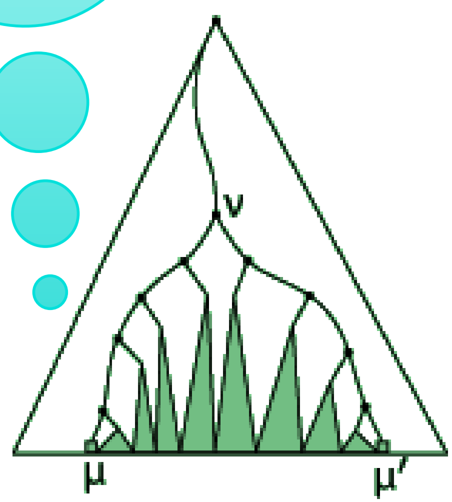
$$T(1) = O(1)$$

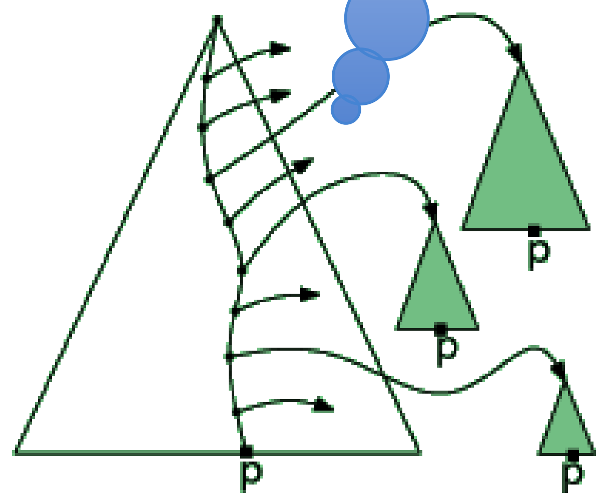
$$T(n) = 2.T(^n/_2) + O(n)$$

Which solves to $O(nlogn)$ time.

Selecting Canonical subsets that contain points of P which are in $[x, x']$

Performing 1DRangeQuery on associated structures of latter points,
Report Points of P which are in $[y, y']$





Algorithm 2DRANGEQUERY($\mathcal{T}, [x : x'] \times [y : y']$)

Input. A 2-dimensional range tree $\mathcal{T}$ and a range $[x : x'] \times [y : y']$.

Output. All points in $\mathcal{T}$ that lie in the range.

$\nu_{\text{split}} \leftarrow$ FINDSPLITNODE($\mathcal{T}, x, x'$)
if ν_{split} is a leaf
  then Check if the point stored at ν_{split} must be reported.
  else (∗ Follow the path to x and call 1DRANGEQUERY on the subtrees right of the path. ∗)
    $\nu \leftarrow lc(\nu_{\text{split}})$
    while ν is not a leaf
      do if $x \leqslant x_\nu$
        then 1DRANGEQUERY($\mathcal{T}_{\text{assoc}}(rc(\nu)), [y : y']$)
          $\nu \leftarrow lc(\nu)$
        else $\nu \leftarrow rc(\nu)$
    Check if the point stored at ν must be reported.
    Similarly, follow the path from $rc(\nu_{\text{split}})$ to x', call 1DRANGEQUERY with the range $[y : y']$ on the associated structures of subtrees left of the path, and check if the point stored at the leaf where the path ends must be reported.

Lemma 5.7: A query with an axis-parallel rectangle in a range tree storing n points takes $O(log^2 n + k)$ time, where k is the number of reported points.

We search in $O(logn)$ associated structures to perform a 1D range query.

- Each call takes $O(k_v + \log | T_{assoc}(v) |) = O(k_v + \log n)$ time.

Total Query Time = $O(\sum_v (k_v + logn)) = O(K + \log^2 n)$.

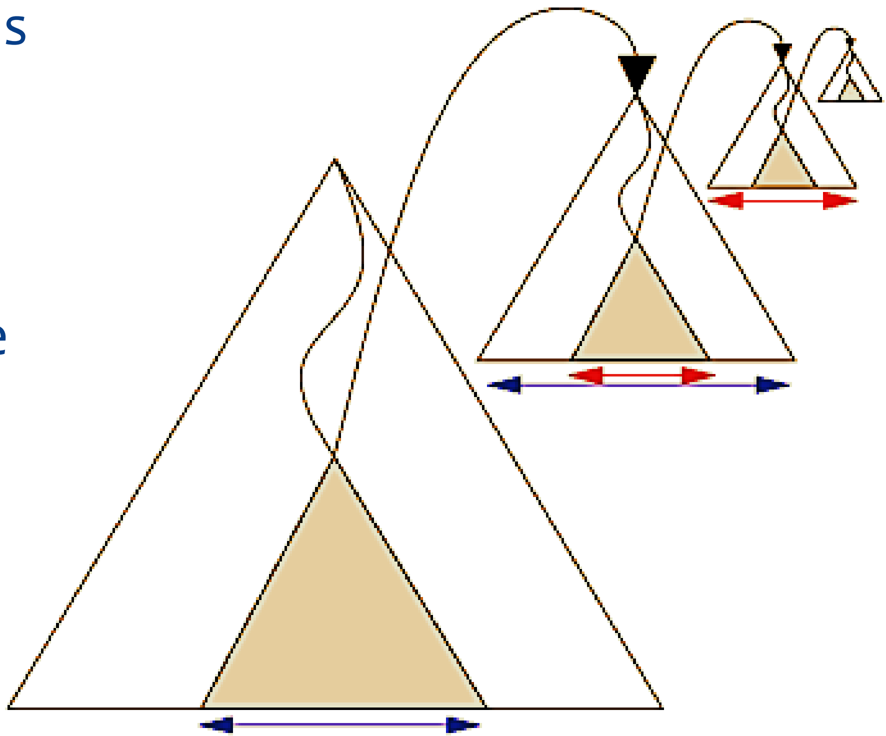
Theorem 5.8 *Let P be a set of n points in the plane. A range tree for P uses $O(n\log n)$ storage and can be constructed in $O(n\log n)$ time. By querying this range tree one can report the points in P that lie in a rectangular query range in $O(log^2 n + k)$ time, where k is the number of reported points.*

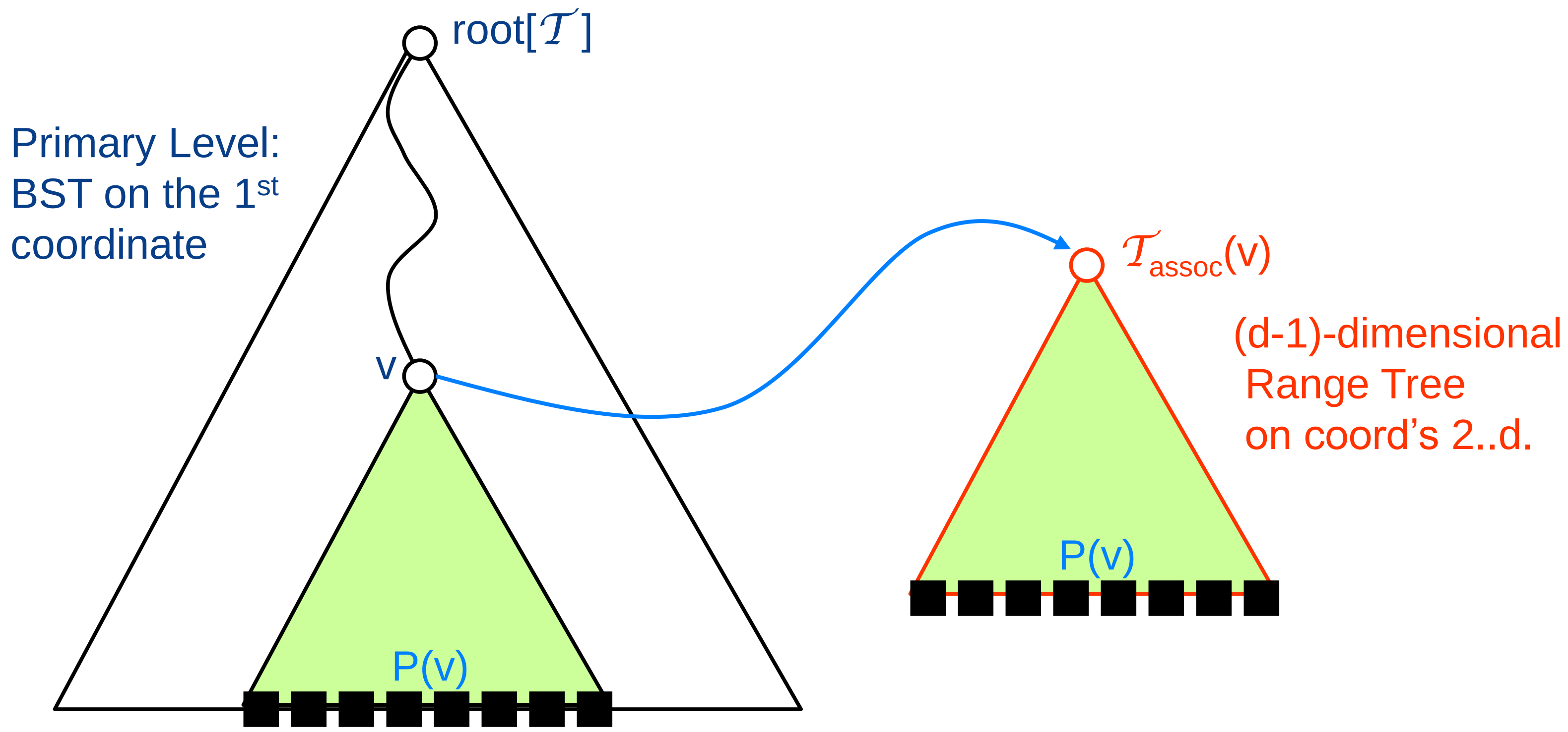
Comparing efficiency:

n	$\log n$	$\log^2 n$	$\sqrt{n}$
4	2	4	2
16	4	16	4
64	6	36	8
256	8	64	16
1024	10	100	32
4096	12	144	64
16384	14	196	128
65536	16	256	256
1M	20	400	1K

Higher-Dimensional Range Trees

A d-dimensional range tree has a main tree which is a one-dimensional balanced binary search tree on the first coordinate, where every node has a pointer to an associated structure that is a (d–1)-dimensional range tree on the other coordinates




root[$\mathcal{T}$]
Primary Level:
BST on the 1st
coordinate
v
P(v)
$\mathcal{T}_{assoc}$(v)
(d-1)-dimensional
Range Tree
on coord's 2..d.
P(v)

Theorem 5.9 :

- *Let P be a set of n points in d-dimensional space, where $d \geq 2$.*

A range tree for P uses $O(nlog^{d-1}n)$ storage and it can be constructed in $O(nlog^{d-1}n)$ time. One can report the points in P that lie in a rectangular query range in $O(log^{d}n + k)$ time, where k is the number of reported points.

Construction Time:	$T_d(n) = O(n\, log^{d-1}\, n)$
Space:	$S_d(n) = O(n\, log^{d-1}n)$
Query Time:	$Q_d(n) = O(K + \log^d n)$

$$\left.\begin{cases} T_d(n) = 2T_d\left(\frac{n}{2}\right) + T_{d-1}(n) + O(n) \\ T_2(n) = O(n\log n) \end{cases}\right\} \Rightarrow T_d(n) = O(n\log^{d-1} n)$$

$$\left.\begin{cases} S_d(n) = 2S_d\left(\frac{n}{2}\right) + S_{d-1}(n) + O(1) \\ S_2(n) = O(n\log n) \end{cases}\right\} \Rightarrow S_d(n) = O(n\log^{d-1} n)$$

$$\left.\begin{cases} Q_d(n) = O(K) + \hat{Q}_d(n) \\ \hat{Q}_d(n) = O(\log n) + O(\log n)\cdot \hat{Q}_{d-1}(n) \\ \hat{Q}_2(n) = O(\log^2 n) \end{cases}\right\} \Rightarrow \begin{cases} \hat{Q}_d(n) = O(\log^d n) \\ Q_d(n) = O(K + \log^d n) \end{cases}$$

General sets of points

Composite-Number Space: $(a|b)$: a & b real numbers.

We now define a total order on this space:

For composite numbers $(a|b)$ & $(a'|b')$

$$(a|b) < (a'|b') \Leftrightarrow a < a' or\ (a = a' \& b < b')$$

For every p in set P, define:

$$p \coloneqq (p_x, p_y) \rightarrow \hat{p} \coloneqq \left((p_x|p_y), (p_y|p_x)\right)$$

Now we define range $\hat{R}$ as below:

$$R \coloneqq [x: x'] \times [y: y'] \rightarrow \hat{R} \coloneqq [(x|-\infty): (x'|+\infty)] \times [(y|-\infty): (y'|+\infty)]$$

Lemma 5.10: Let p be a point and R a rectangular range. Then

$$p \in R \Leftrightarrow \hat{p} \in \hat{R}$$

Proof:

$$p \in R \Leftrightarrow x \leq p_x \leq x' \,\&\, y \leq p_y \leq y'$$

$$\Leftrightarrow (x|-\infty) \leq \left(p_x \middle| p_y\right) \leq (x'|+\infty) \,\&(y|-\infty) \leq \left(p_y \middle| p_x\right) \leq (y'|+\infty)$$

$$\Leftrightarrow \hat{p} \in \hat{R}$$

Therefore we can use $\hat{p}$ instead of p and $\hat{R}$ instead of R in order to conquer the degenerate case of points with same x- or y-components.