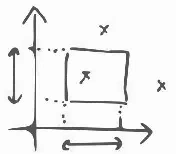
Orthogonal Range Searching.

Prob Given n points $p_1, \dots, p_n$, and a rectangular range query, find all the points inside the given range.

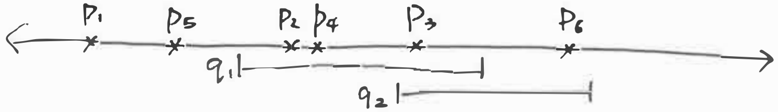


query that can be represented as the Cartesian product of ranges.

Naively, we can iterate through all points ... $O(n)$ alg.

$\Rightarrow$ Consider the setting where the points are fixed & range queries change. We want a sublinear query time.

I. The 1-dimensional case.



~~[illegible]~~ Consider the query $q = [x_1 ; x_2]$.

Approach #1. Arrays.

Alg Prep1

Sort the points.

Alg Query1

- Do binary search to find the smallest point p_i s.t. $p_i \geq x_1$
- Scan the array while outputting all points $< x_2$.

... easy to see that approach #1 uses $O(n \log n)$ prep + $O(\log n + k)$ query time, where k = # reported points.

>> Easy, but hard to generalize to higher dims.

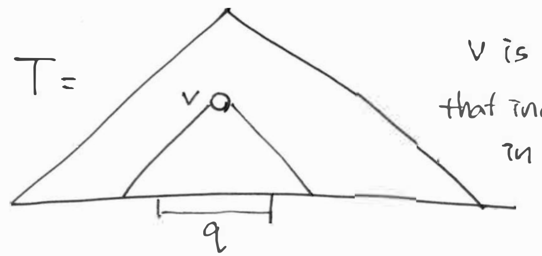
Approach #2. BST.

Alg Prep2

$T \leftarrow$ BST with points sorted w/ coords. & only at leaves.

Alg Query2

$v \leftarrow$ FindSplitNode(T, q)
report (v, q)



v is the lowest node that includes all nodes to report in the subtree rooted @ itself

Alg FindSplitNode(T, q)

```
v ← T.root.
while v is not a leaf and (x₂ ≤ v.x or x₁ > v.x)
    if x₂ ≤ v.x
        v ← v.left
    else v ← v.right
return v
```

Alg report(v, q)

if v is a leaf, report v if $v \in q$

else

  cur $\leftarrow$ v.left

  while cur is not a leaf

    if $x_1 \le$ cur.x

      report all leaves in the subtree rooted at cur.right

      cur $\leftarrow$ cur.left

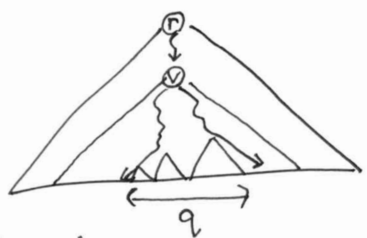
    else cur $\leftarrow$ cur.right

  if cur is a leaf, report cur if cur $\in q$.

↳ repeat once for the right symmetrically.

Analysis

Query 2 consists of two parts:



i) going down the BST

ii) reporting leaves in subtrees.

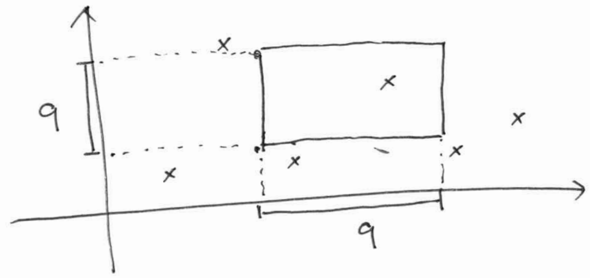
Assuming a balanced BST, i) takes $O(\log n)$ time

Since # internal nodes < # leaves, ii) takes $O(k)$ time

Overall, Query 2 runs in $O(\log n + k)$ time.

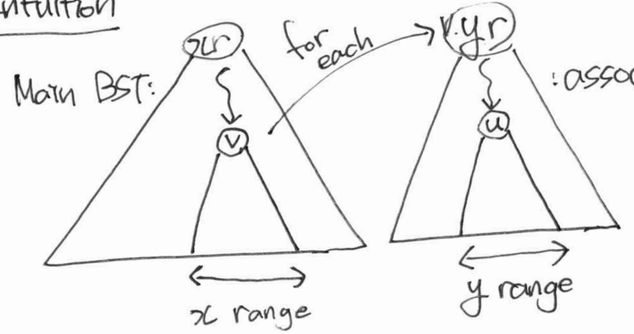
III. The 2-dimensional case.

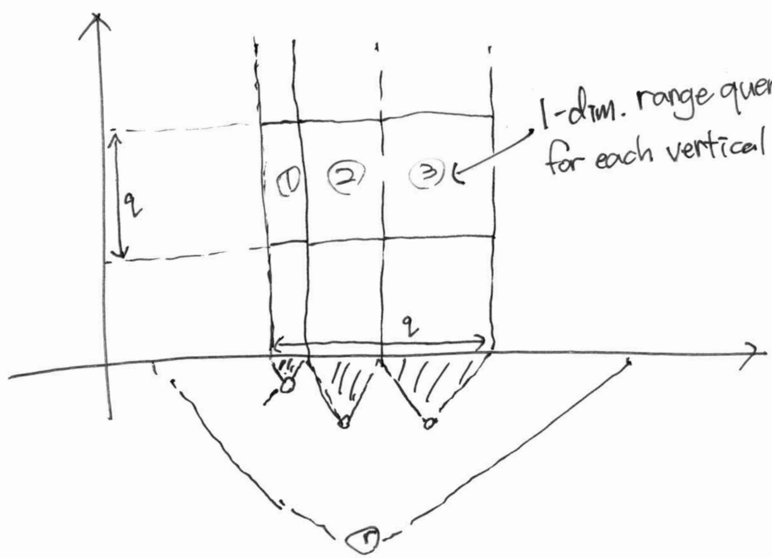
Approach #1. Range trees.



We extend the BST method to higher dims.

Intuition





Alg Construct Range Tree (P)

$P_x \leftarrow$ Sort points in P w/ x-coords

~~$P_y \leftarrow$ 〃 〃 〃 〃 y-coords~~

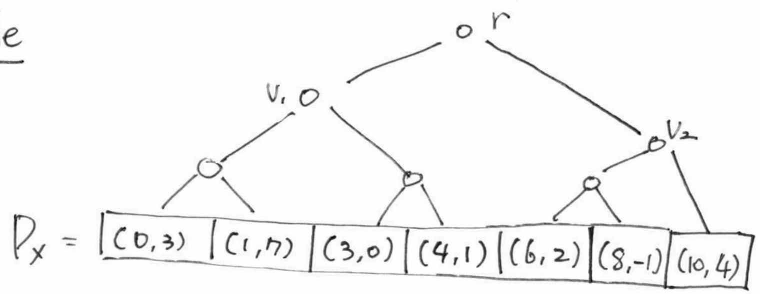
Structure the main BST using P_x so that it is balanced & full

For each node v in the main BST in bottom-up order do:

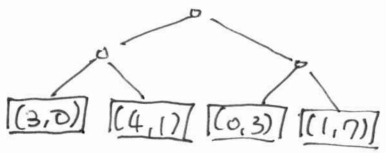
if v is a leaf, then v.assoc is a singleton BST w/ a single point.

o/w. merge v.left.assoc & v.right.assoc to construct v.assoc.

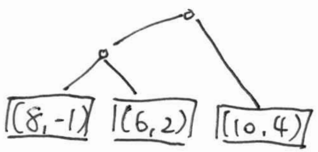
Example



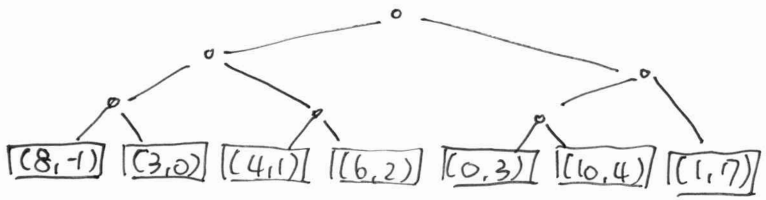
v_1.assoc:



v_2.assoc:



r.assoc:



i.e., merge like in mergesort.

Analysis The range tree of P can be constructed in $O(n\log n)$ time and uses $O(n\log n)$ space.

pf) 1. space. Each point p_i is stored only in the assoc. BST in a path from a leaf to the root. ∴ a point appears exactly once in the assoc. BST's of a given depth.

D ⇒ the overall storage of all the assoc. BSTs at level D is $O(n)$. Since the tree is balanced, the range tree uses $O(n\log n)$ storage.

2. time. Structuring the main BST takes $O(n)$ time. Sorting P takes $O(n\log n)$ time. Merging takes overall $O(n\log n)$ time using a similar analysis w/ merge sor

≫ We can answer axis-parallel rectangular range queries in $O(\log^2 n + k)$ time if we perform 1-dim. range queries for each BST.

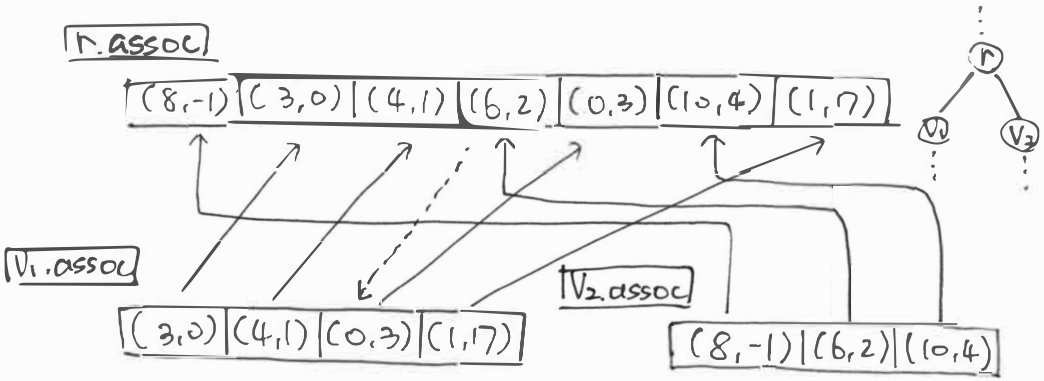
Remark. For points in $d \geq 2$-dimensional space, we can construct a range tree in $O(n\log^{d-1} n)$ time & space which can answer axis-parallel hyperrectangular queries in $O(\log^d n + k)$ time.

Approach #1-1. Range trees w/ fractional cascading.

~~Idea~~ Observation We do not need to maintain a BST for the associated structure at the final level.

Question How can we avoid searching at every associated structure at the final level?

≫ Return to the mergesort intuition.



Originally, we would have to search in both v_1.assoc & v_2.assoc. resulting in a $O(\log^2 n + k)$ RT. However, note that we can maintain pointers so that we only perform binary search @ the root & report consecutive points when necessary.

Example consider range query $[-1:7] \times [2:6]$.
We first find $(6,2)$ using binary search on r.assoc.
Since the x-range of v_1 is in $[-1:7]$, we report on v_1.
The least elt. in v_1.assoc. no smaller than $(6,2)$ is $(0,3)$, which is in the rect. However the next elt. $(1,7)$ is out-of-bounds, so we stop reporting [illegible]

Thm Let P be a set of n points in a d-dim. space, where $d \geq 2$. A (layered) range tree for P uses $O(n \log^{d-1} n)$ storage and it can be constructed in $O(n \log^{d-1} n)$ time. With the range tree, we can report the points in P that lie in a rectangular query range in $O(\log^{d-1} n + k)$ time, where k = # reported pts.

Approach #2. kd-trees.

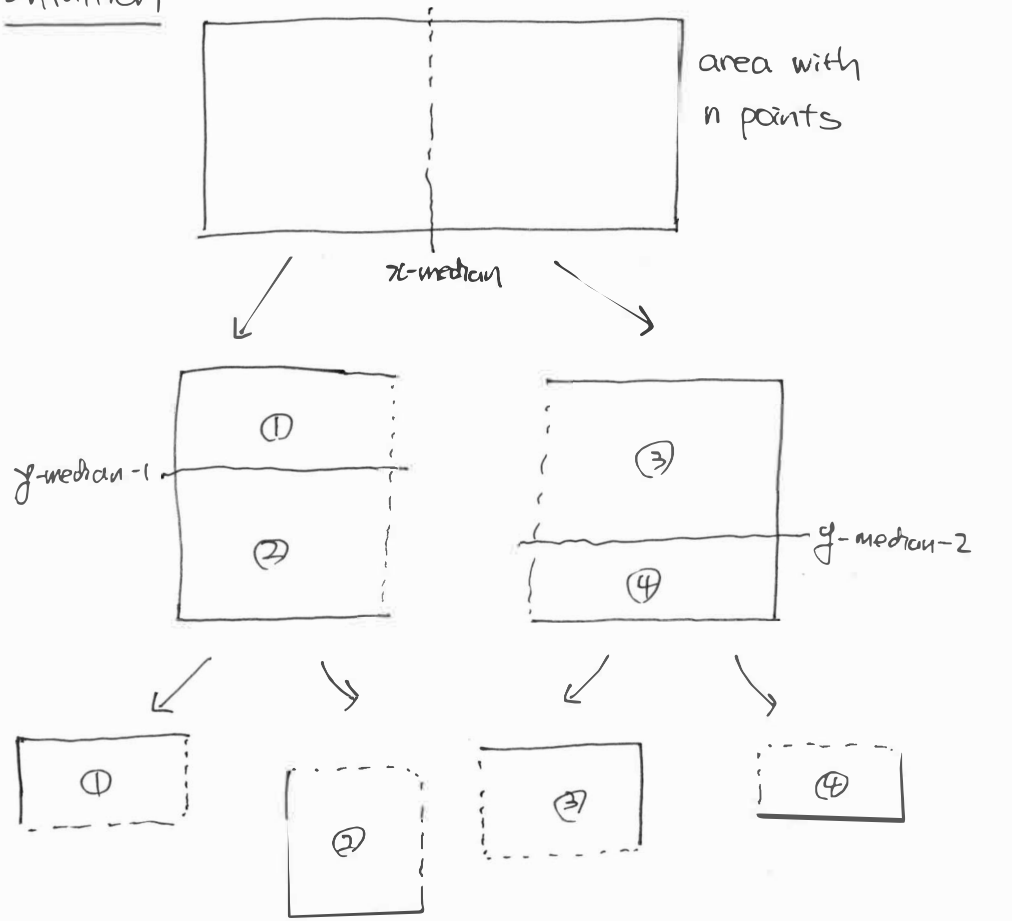
> We want a data structure w/ linear storage at the cost of querying time.

⇒ We can no longer "layer" structures for each dim.

How to structure a single BST?

Answer Alternate btwn splitting with X and splitting with Y.

Intuition



We report points in regions that are fully contained by the query rectangle.

+ We can extend to $d>2$ dimensions by using ~~medians along~~ the median along the $(D \bmod d)^{th}$ axis, where D is the depth in the kd-tree.