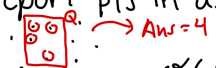
Range Trees I

Recap:

- **kd-Tree:** General-purpose data structure for pts in $\mathbb{R}^d$
- **Orthogonal range query:** Count/report pts in axis-aligned rect.



- kd-Tree: **Counting:** $O(\sqrt{n})$ time
  Report: $O(k + \sqrt{n})$ time

No. of pts reported

Can we do better?

Range Trees:

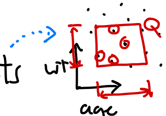
- Space is $O(n \log^{d-1} n)$
- Query time:
  - **Counting:** $O(\log^d n)$
  - **Reporting:** $O(k + \log^d n)$

→ In $\mathbb{R}^2$: $\log^2 n$ **much better** than $\sqrt{n}$ for large n

→ Range trees are **more limited**

Layering: Combining search structures

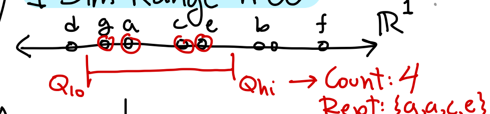
- Suppose you want to answer a **composite query** w. multiple criteria:
- Medical data: Count subjects
  - **Age range:** $a_{lo} \leq age \leq a_{hi}$
  - **Weight range:** $w_{lo} \leq weight \leq w_{hi}$



- Design a data structure for each criterion **individually**
- **Layer** these structures together to answer full query

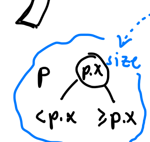
→ **Multi-Layer Data Structures**

1-Dim Range Tree:



Approach:

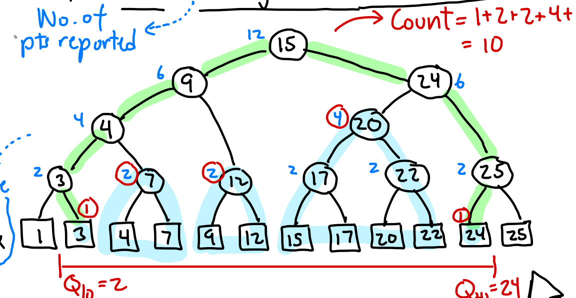
- Balanced BST (eg. AVL, RB, ...)
- Assume **extended tree**
- Each node p stores no. of entrie in subtree: **p.size**



Call this a **1-Dim Range Tree:**

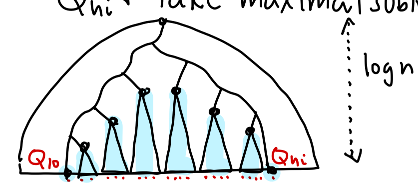
Claim: A 1-Dim range tree with n pts has space $O(n)$ and answers 1-D range count/rept queries in time $O(\log n)$ (or $O(k + \log n)$)

No. of pts reported



Canonical Subsets:

- **Goal:** Express answer as disjoint union of subsets
- **Method:** Search for Q_{lo} + Q_{hi} + take maximal subtrees



Range Trees II

Recursive helper:

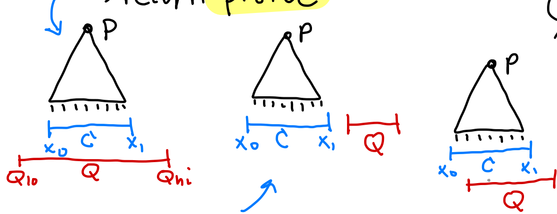
int range1Dx(Node p,
  Intv $Q=[Q_{lo},Q_{hi}]$, Intv $C=[x_0,x_1]$)

initial call: range1Dx(root, Q, C_0)

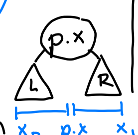
Cases:

p is external:
- if $p.pt.x \in Q \rightarrow 1$ else $\rightarrow 0$

p is internal:
- $C \subseteq Q \Rightarrow$ all of p's pts lie within query
  $\rightarrow$ return p.size

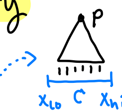


- C is disjoint from Q $\Rightarrow$ none of p's pts lie in Q
  $\rightarrow$ return 0
- Else partial overlap
  $\rightarrow$ Recurse on p's children + trim the cell



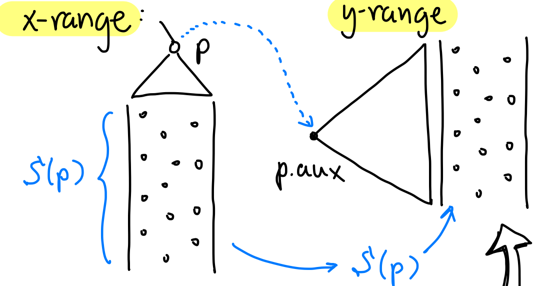
More details:

Given a 1-D range tree T:
- Let $Q=[Q_{lo},Q_{hi}]$ be query interval
- For each node p, define interval cell $C=[x_0,x_1]$ s.t. all pts of p's subtree lie in C



- Root cell: $C_0=[-\infty,+\infty]$

```
int range1Dx(Node p,
    Intv Q, Intv C=[x0,x1]) {
  if (p is external)
    └return p.pt.x∈Q  T→ 1, F→ 0
  else if (C ⊆ Q) return p.size
  else if (Q+C disjoint) return 0
  else return:
    range1Dx(p.left, Q, [x0, p.x])
  + range1Dx(p.right, Q, [p.x, x1])
```



2-D Range Searching:

- "Layer" a range tree for x with range tree for y
- For each node $p \in$ 1D-x tree, let S(p) = set of pts in p's subtree
- Def: p.aux: A 1D-y tree for S(p)

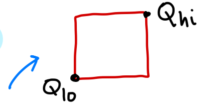
Analysis:

Lemma: Given a 1-D range tree with n pts, given any interval Q, can compute $O(\log n)$ subtrees whose union is answer to query.

Thm: Given 1-D range tree... can answer range queries in time $O(\log n)$ (+k to report)

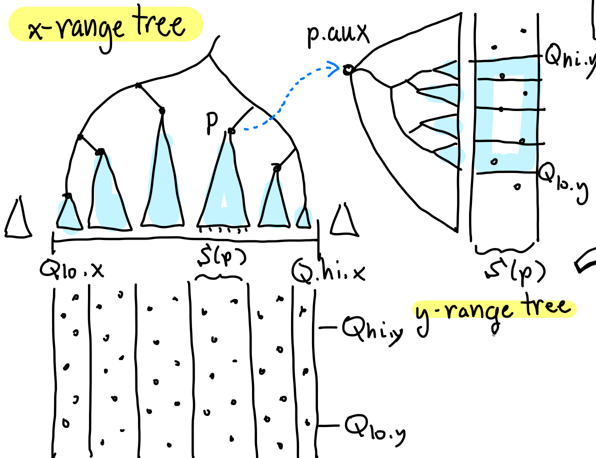
Range Trees III

Answering Queries?



Given query range
$Q = [Q_{lo}.x, Q_{hi}.x] \times [Q_{lo}.y, Q_{hi}.y]$

- Run range1Dx to find all subtrees that contribute
- For each such node p, run range1Dy on p.aux
- Return sum of all result



Intuition: The x-layer finds subtrees p contained in x-range + each aux tree filters based on y.

2D Range Tree:

- Construct 1D range tree based on x coords for all pts
- For each node p:
  - Let $S(p)$ be pts of p's tree
  - Build 1D range tree for $S(p)$ based on y → p.aux
- Final structure is union of x-tree + (n-1) y-trees

Higher Dimensions?

- In d-dim space, we create d-layers
- Each recurses one dim lower until we reach 1-d search
- Time is the product:
  $\underbrace{\log n \cdot \log n \cdots \log n}_{d} = O(\log^d n)$

Analysis: The 1D x search takes of $O(\log n)$ time + generates $O(\log n)$ calls to 1D y search
⇒ Total: $O(\log n \cdot \log n) = O(\log^2 n)$

```
int range2D(Node p, Rect Q, Intv C=[x0,x1]){
  if (p is external) return p.pt ∈ Q ? (T→1, F→0)
  else if (Q.x contains C){   // C ⊆ Q's x-projection
    [y0,y1] = [-∞,+∞]          // init y-cell
    return range1Dy(p.aux, Q, [y0,y1])
  } else if (Q.x is disjoint of C) return 0
  else                         // partial x-overlap
    return range2D(p.left, Q, [x0, p.x])
         + range2D(p.right, Q, [p.x, x1])
}
```

Analysis:

Invoked $O(\log n)$ times - once per maximal subtree

Invoked $O(\log n)$ times - once for each ancestor of max subtree