Can we do better?

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

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

→ Range trees are more limited

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
  - Reporting: $O(k + \log n)$ time

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/report queries in time $O(\log n)$ (or $O(k + \log n)$)

Layering:
- Combining search structures
  - Suppose you want to answer a composite query w. multiple criteria:
    - Medical data: Count subjects
      - Age range: $18 \leq \text{age} \leq 35$
      - Weight range: $100 \leq \text{weight} \leq 200$
    - Design a data structure for each criterion individually
    - Layer these structures together to answer full query

→ Multi-Layer Data Structures

Range Trees I

1-Dim Range Tree:

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

→ Multi-Layer Data Structures

Canonical Subsets:
- Goal: Express answer as disjoint union of subsets
- Method: Search for $Q_i + Q_j$ that take maximal subsets

Approach:
- Balanced BST (e.g. AVL, RB, ...)
- Assume extended tree
- Each node $p$ stores no. of entries in subtree: $p.stats$
Recursive helper:
\[
\text{int range1Dx(Node p, 
\quad \text{Intv } Q=[Q_L, Q_R], \text{Intv } C=[x_L, x_R]) 
\]
\]
initial call: \(\text{range1Dx(root, Q, C_o)}\)

Cases:

- **p is external:**
  - if \(p.p.x \in Q\) → \(1\)
  - else → \(0\)

- **p is internal:**
  - \(C \subseteq Q\) ⇒ all of p's pts lie within query
  - \(\rightarrow \) return \(p\).size

- **C is disjoint from Q:**
  - none of p's pts lie in Q
  - \(\rightarrow \) return \(0\)

- **Else partial overlap**
  - Recurse on p's children
  + trim the cell

More details:

Given a 1-D range tree T:
- Let \(Q=[Q_L, Q_R]\) be query interval
- For each node \(p\), define interval cell \(C=[x_L, x_R]\)
  - s.t. all pts of p's subtree lie in \(C\)
  - Root cell: \(C_o=[-\infty, +\infty]\)

Range Trees II

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
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

2D Range Tree:
- Construct 1D range tree based on $x$ coordinate 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 $y \mapsto p.aux$
- Final structure is union of $x$-tree + $(n-1)$ $y$-trees

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

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:
  \[
  \log n \cdot \log n \cdot \ldots \cdot \log n = O(\log^d n)
  \]

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

Invoked $O(\log n)$ times - once per maximal subtree
Invoked $O(\log n)$ times - once for each ancestor of max subtree
Aside: Extended Binary Search Trees

Recall: Extension $\rightarrow$ replace null with special external node

Extended Binary Search Trees:
- Data (dictionary contents) stored in external nodes
- Internal nodes are just guideposts leading to actual data

Contents: $\{2,4,7,10,13,15,18\}$

Pros:
- Reduce space for "index" (internal nodes)
- Save space for unused null pointers

Cons:
- Coding is more complex (2 node types)

$$\text{find}(\text{Key } x, \text{Node } p)$$

```
if (p is external) return find(x, p.left)
else if (x == p.key) found!
else /* p is internal */
    if (x < p.key) return find(x, p.left)
    else return find(x, p.right)
```
Prog Assign 2 - Due Tue, 4/26 11:59 pm

class HBkdTree {
    private {
        root (KDNode)
        max Height Diff (int)
        size (# of pts)
        cell (Rected)
    }
    class KDNode {
        (x, y) - Point 2D
        LPoint point
        Label "DCA"
        children left, right
        height
        cell
        cutDim
    }
    Helpers (private)
    }
    public