Can we do better?

Recap:
- kd-Tree: General-purpose data structure for pts in $\mathbb{R}^d$
- Orthogonal range query:
  - Count/report pts in axis-aligned rect.
  - $\text{Count} = O(\text{log} n)$
  - $\text{Report} = O(k + \text{log} n)$

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

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

Range trees are more limited

Layering: Combining search structures
- Suppose you want to answer a composite query with multiple criteria:
  - Medical data: Count subjects
    - Age range: $a_{i0} \leq \text{age} \leq a_{i1}$
    - Weight range: $w_{i0} \leq \text{weight} \leq w_{i1}$
- Design a data structure for each criterion individually
- Layer these structures together to answer full query

→ Multi-Layer Data Structures

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 $O(\text{log} n)$ (or $O(k + \text{log} n)$)

1-Dim Range Tree:
- Canonical Subsets:
  - Goal: Express answer as disjoint union of subsets
  - Method: Search for $Q_{i0} + Q_{i1}$

Approach:
- Balanced BST (e.g. AVL, RB, ...)
- Assume extended tree
- Each node $p$ stores no. of entries in subtree: $p$.size

Layering:
Combining search structures
Recursive helper:
\[
\text{int range1Dx(Node p, 
Intv Q = [Q_h, Q_k], Intv C = [x_o, x_1])} \\
\text{initial call: range1Dx(root, Q, Co)}
\]

More details:
- Given a 1D range tree T:
  - Let \( Q = [Q_h, Q_k] \) be query interval
  - For each node \( p \), define interval cell \( C = [x_o, x_1] \)
  - \( s.t. \) all pts of \( p \)'s subtree lie in \( C \)
  - Root cell: \( C = [-\infty, +\infty] \)

Cases:
\( p \) is external:
- if \( p.pt.x \in Q \rightarrow 1 \) else \( 0 \)
\( p \) is internal:
- \( C \leq Q \Rightarrow \text{all of } p \text{'s pts lie within query} \)
  \( \rightarrow \text{return } p \text{'s size} \)

Recurs:

\[
\text{int range1Dx(Node p, 
Intv Q, Intv C = [x_o, x_1])} \\
\text{if( } p \text{ is external) } \rightarrow 1 \\
\text{  return } p \text{.'s size} \\
\text{else if( } C \leq Q \text{) return } p \text{'s size} \\
\text{else if( } Q+G \text{disjoint) return } 0 \\
\text{else return:} \\
\text{  range1Dx(p.left, Q, [x_o, p.x])} \\
\text{  + range1Dx(p.right, Q, [p.x, x_1])}
\]

Analysis:

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

Thm: Given 1D range tree...can answer range queries in time \( O(\log n) \) \( \rightarrow (k \text{ to report}) \)
Answering Queries?

Given query range $Q = [Q_{lo,x}, Q_{hi,x}] \times [Q_{lo,y}, Q_{hi,y}]$
- Run range $1D_x$ to find all subtrees that contribute
  - For each such node $p$,
    - run range $1D_y$ 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 $x$ coord. for all pts
- For each node $p$:
  - Let $S(p)$ be pts of $pi$ tree
  - Build 1D range tree for $S(p)$ based on $y \rightarrow p.aux$
- Final structure is union of $x$-tree + $(n-1)$ $y$-trees

**Analysis:**
- The $1D_x$ search takes $O(\log n)$ time + generates $O(\log n)$ calls to $1D_y$ search
- Total: $O(\log n \cdot \log n) = O(\log^2 n)$

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:**
- Invoked $O(\log n)$ times once per maximal subtree
- Invoked $O(\log n)$ times once for each ancestor of max subtree
The image contains a mathematical equation and a diagram. The equation is:

\[
\sum_{x \in E} (1 + \log n) = n + n\log n
\]

For all y-trees, \(x\)-trees contribute to the \(y\)-tree together:

- Space: \(O(n \log^2 n)\)
- Query time: \(O(\log^3 n)\)
Range Tree Applications:
- Range trees can be applied to a variety of query problems

Methods:
- Minimization/Maximization
- Transform coordinates
- Adding new coordinates

Skewed rectangle query:

Given a set $P$ of $n$ pts in $IR^2$, a skewed rectangle is given by 2 pts $q^- = (x^-, y^-)$ and $q^+ = (x^+, y^+)$ and consists of pts in parallelogram with two vertical sides and two with slope $+1$ corners at $q^- + q^+$.

Return a count of the number of pts of $P$ inside the skewed rectangle.

NE Right Triangle Query

Given a set $P$ of $n$ pts in $IR^2$ and scalar $l > 0$, a NE triangle is a 45-45 right triangle with lower left corner at $q$ and side length $l$. Return a count of the number of pts of $P$ lying within the triangle.
3-sided Min Query

Return lowest in region
region \( x_0 \leq x \leq x_1, \ y \geq y_0 \)

Skewed rectangle query:

Given \( y_0 \) find the \( y \) in tree \( y \geq y_0 \)

Data structure:
- Build a range tree for \( x \)
- Aux. trees are range trees for \( y \) that support find Next Larger

Query Processing:
- Do 1D range search in main tree for interval \([x_0, x_1]\)
- For each maximal subtree in range, do find larger \((y_0)\)
- Return smallest of these \( y \)

Analysis:
- Same as 2D range tree
- Space: \( O(n \log n) \) Time: \( O(\log^2 n) \)

Transform coordinates to make orthogonal range query

\[ q^+ = (x^+, y^+) \]
\[ q^- = (x^-, y^-) \]

Line equation:

\[ y = x + (q^-_y - q^-_x) \]

\[ p_x + (q^-_y - q^-_x) \leq p_y \leq p_x + (q^+_y - q^+_x) \]

\[ q^-_y - q^-_x \leq p_y - p_x \leq q^+_y - q^+_x \]

Map each \( p = (p_x, p_y) \in \mathcal{P} \) to \( p' = (p_x, p_y') \in (p_x, p_y, p_x) \)

Let \( \mathcal{P}' \) be resulting set.

Build std. range tree for \( \mathcal{P}' \). Return ans. to query

\[ q^-_y - x \leq y \leq q^+_y - q^+_x \]

\[ q^-_x \leq x \leq q^+_x \]
**NE Right Triangle Query**

**Build a 3D range tree on \( P' \)**

**NE triangle query becomes:**

\[
\begin{align*}
q_x & \leq x \leq q_x + l \\
q_y & \leq y \leq q_y + l \\
q_x + q_y & \leq z \leq q_x + q_y + l
\end{align*}
\]

**Space:** \( O(n \log^2 n) \)

**Query time:** \( O(\log^3 n) \)

- Add new coord:
  \[
  z = x + y
  \]
- Map pts:
  \[
  p = (p_x, p_y) \rightarrow p' = (p_x, p_y, p_x + p_y)
  \]
- Let \( P' \) be resulting set
- Get a node’s height
  - Node p stores p.height but what if p == null?

```cpp
int getHeight(KDNode p) {
    if (p == null) return -1
    else return p.height
}
```

- Update node’s height

```cpp
void updateHeight(KDNode p) {
    p.height = 1 + max
    get Height(p.left),
    get Height(p.right))
}
```

- Similar to AVL insert
  - but pass in the cell
  - Return update subtree

```cpp
KDNode insert(LPoint pt, KDNode p, Rect cell) {
    if (p == null) return "new node... cutting dim based on cell"
    else if (p.point == pt) return "error - Duplicate!"
    else if (pt is on left side) p.left = insert(pt, p.left, left Part cell)
    else p.right = insert(... on right side)
    return rebalance(p)
}
```

getMin(i, pt1, pt2)
getMin(i, left, getMin(i, p, pt, right))

Utilities/Helpers:

- Insert Helper: