Can we do better?

Range Trees:
- Space is $O(n \log^d n)$
- Query time:
  - Counting: $O(\log^d n)$
  - Reporting: $O(k + \log^d n)$
In $\mathbb{R}^2$: $\log^2 n$ much better than $\Omega(n)$ 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: $a_{i_0} \leq age \leq a_{i_1}$
    - Weight range: $w_{i_0} \leq weight \leq w_{i_1}$
- Design a data structure for each criterion individually
- Layer these structures together to answer full query
→ Multi-Layer Data Structures

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

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

Call this a 1-Dim Range Tree:

1-Dim Range Tree:
- Goal: Express answer as disjoint union of subsets
- Method: Search for $Q_{i_0} + Q_{i_1}$ + take maxima/subtrees

Canonical Subsets:
- Design a data structure for each criterion individually
- Layer these structures together to answer full query
→ Multi-Layer Data Structures
Recursive helper:  
\[ \text{int range1Dx}(\text{Node } p, \text{Intv } Q = [Q_h, Q_w], \text{Intv } C = [x_0, x_1]) \]

Initial call:  
\[ \text{range1Dx}(\text{root}, Q, C) \]

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
    - Recurse on \( p \)'s children + trim the cell

More details:
Given a 1-D range tree \( T \):
- Let \( Q = [Q_h, Q_w] \) 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: \( C_o = [-\infty, +\infty] \)

2-D Range Searching:
- Layer a range tree for \( x \) with range tree for \( y \)
- For each node \( p \) \( \in \) 1-D \( x \) tree, let 
  \[ S(p) = \text{set of pts in } p \)'s subtree \]
- Def: \( p_{aux} \): A 1-D \( y \) 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 \( O(\log n) \) \( \rightarrow \) \( (k \text{ to report}) \)
Answering Queries?
Given query range
$Q = [Q_{lo,x}, Q_{hi,x}] \times [Q_{lo,y}, Q_{hi,y}]$
- Run range1Dx to find all subtrees that contribute
  - For each such node p
    - run range1Dy on p.aux
- Return sum of all result

x-range tree

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 \to p.aux
- Final structure is union of x-tree + (n-1) y-trees

y-range tree

Higher Dimensions?
- In d-dim space, we create d-layers
- Each recrusrs one dim lower until we reach 1-d search
- Time is the product:
  $\log n \cdot \log n \ldots \log n = O(\log^d n)$

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

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

int range2D(Node p, Rect Q, Intv C=[x_0,x_1])
if (p is external) return p.pt in Q
else if (Q.x contains C) \COMMENT{C \subseteq Q.x \text{-projection}}
  \begin{align*}
  [y_0, y_1] &= [-\infty, +\infty] \COMMENT{init y-cell} \\
  \text{return range1Dy}(p.aux, Q, [y_0, y_1])
  \end{align*}
else if (Q.x is disjoint of C) return 0
else \COMMENT{partial x-overlap}
  return range2D(p.left, Q, [x_0, p.x])
  + range2D(p.right, Q, [p.x, x_1])

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