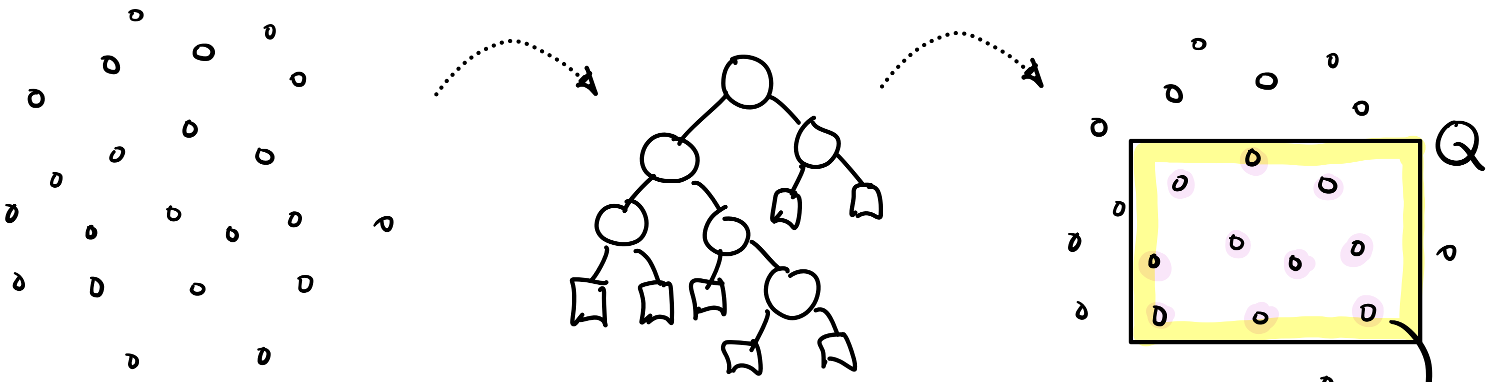
CMSC 754 - Computational Geometry
Lecture 15 - Orthogonal Range Trees

Recall: Range Search:

Given a set of n pts $P = \{p_1, \ldots, p_d\} \subseteq \mathbb{R}^d$, and class of shapes (range space) preprocess P to answer range queries: Given shape Q, count/report the pts in $P \cap Q$.



P + shape = rectangles

Data structure

Ans: 10 pts of P in Q

Last lecture: kd-trees

$O(n)$ space / $O(n \log n)$ build time
$O(\sqrt{n})$ query time (in $\mathbb{R}^2$)
$O(n^{1-1/d})$ in $\mathbb{R}^d$

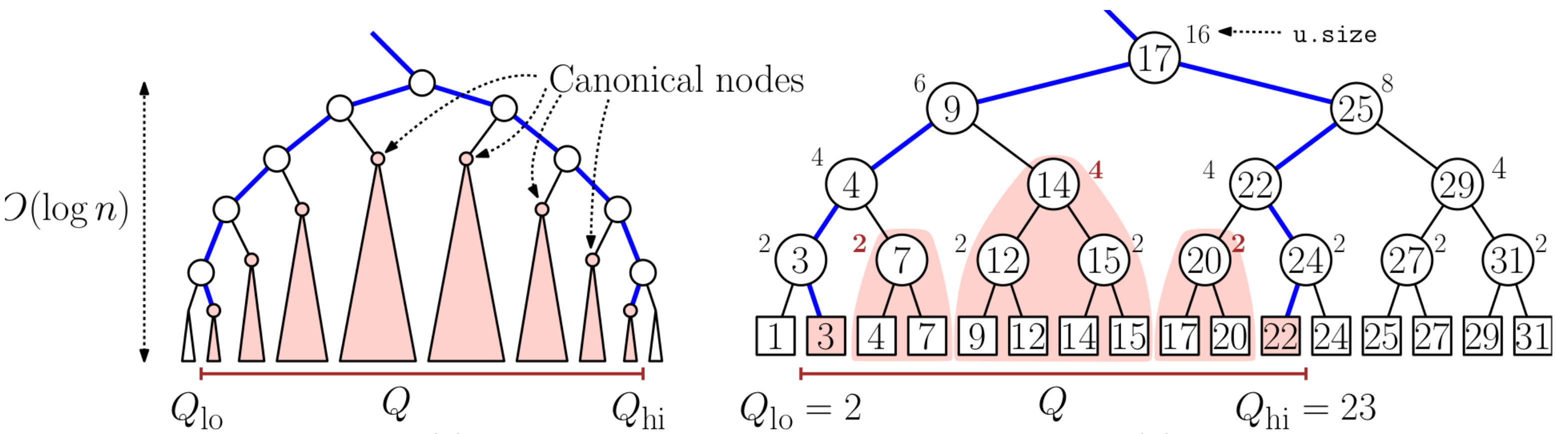
Today: Orthogonal Range Trees + Layered Data Structures

→ $O(\log^d n)$ query time

→ $O(n \log^{d-1} n)$ space

1-Dimensional Range Tree: (Review)

- Given set of scalars: $\underline{P} = \{p_1, \dots, p_n\} \subseteq \mathbb{R}$
- Store as leaves in balanced search tree
  - $\rightarrow \mathcal{O}(n)$ space
  - $\rightarrow \mathcal{O}(n \log n)$ construct. time
- Each node u stores num. of leaves: u.size
- Given query interval $Q = [Q_{lo}, Q_{hi}]$
  - Identify $\mathcal{O}(\log n)$ maximal subtrees that cover Q
  - Add up sizes for all these nodes



Query answer = $1 + 2 + 4 + 2 + 1 = 10$

Range counting algorithm:

Node u :

- **u.point** : point p_i (if u is leaf)
- **u.x** : split value (if u internal)
- **u.size** : # leaves (if u internal)
- **u.left**, **u.right** : children

range1Dx(Node u, Range Q, Interval $C = [x_0, x_1]$)

if (u is leaf)

  return $\begin{cases} 1 & \text{if } u.\text{point} \in Q \\ 0 & \text{o.w.} \end{cases}$

else if ($C \cap Q = \emptyset$) (no overlap)

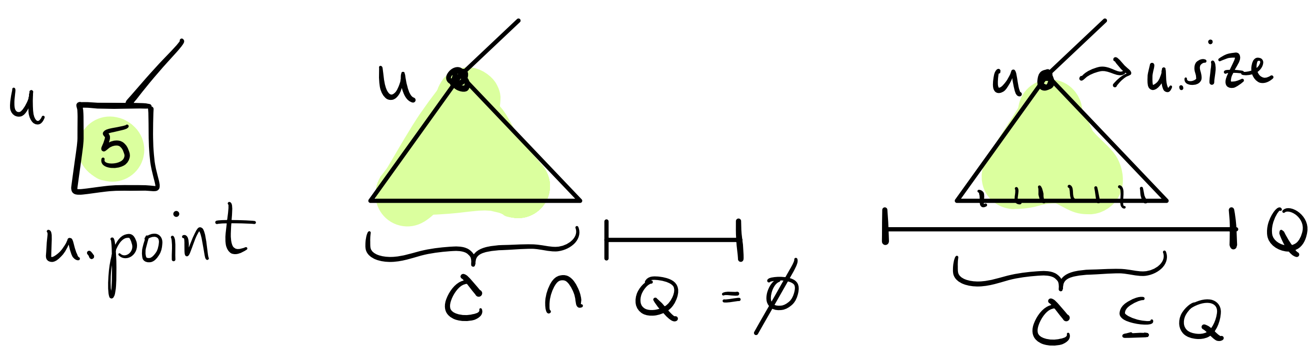
  return 0

else if ($C \subseteq Q$) (contained)

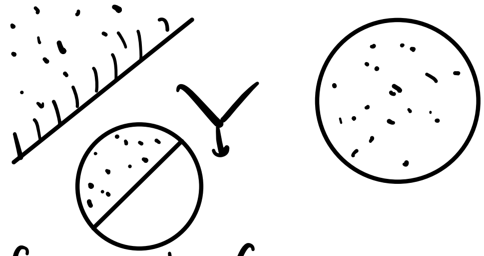
  return u.size

else (recurse)

  return range1Dx(u.left, Q, $[x_0, u.x]$)
    + range1Dx(u.right, Q, $[u.x, x_1]$)



Multi-Layered Structures:



Suppose your ranges are formed from **composing multiple (independent) queries**:

Eg. Find all patients of

- age between 25..35 : Q_1
- weight $\leq$ 200 lbs : Q_2
- blood pressure $\geq$ 100 : Q_3

Idea: Design a data structure for **each query type** + "**merge them**"

How to merge?

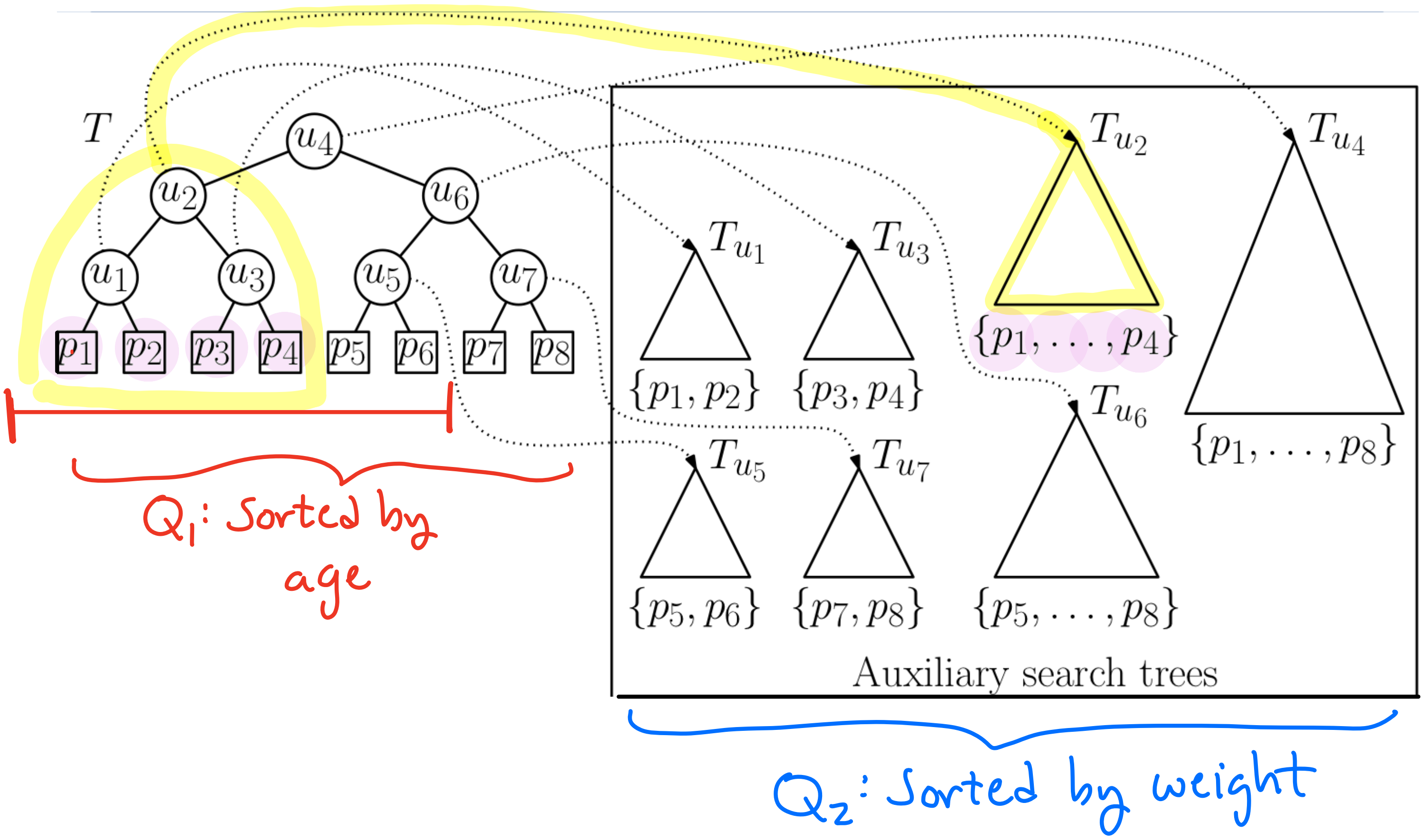
- Build **range structure for age** for P
  - $\Rightarrow$ **Canonical subsets**: $P_1, P_2, \ldots, P_m$
- For each P_i, build a **range structure for weight**
  - $\Rightarrow$ **Canonical subsets**: $P_{i1}, P_{i2}, \ldots$
- For each P_{ij}, build **range structure for blood pressure**

⋮

Multi-Layered Search Tree:

- Store data in leaves of tree
- Each node's canonical subset consist of it's leaves
- For each node, build a search tree for its canonical subset
  called its auxiliary tree

Example:



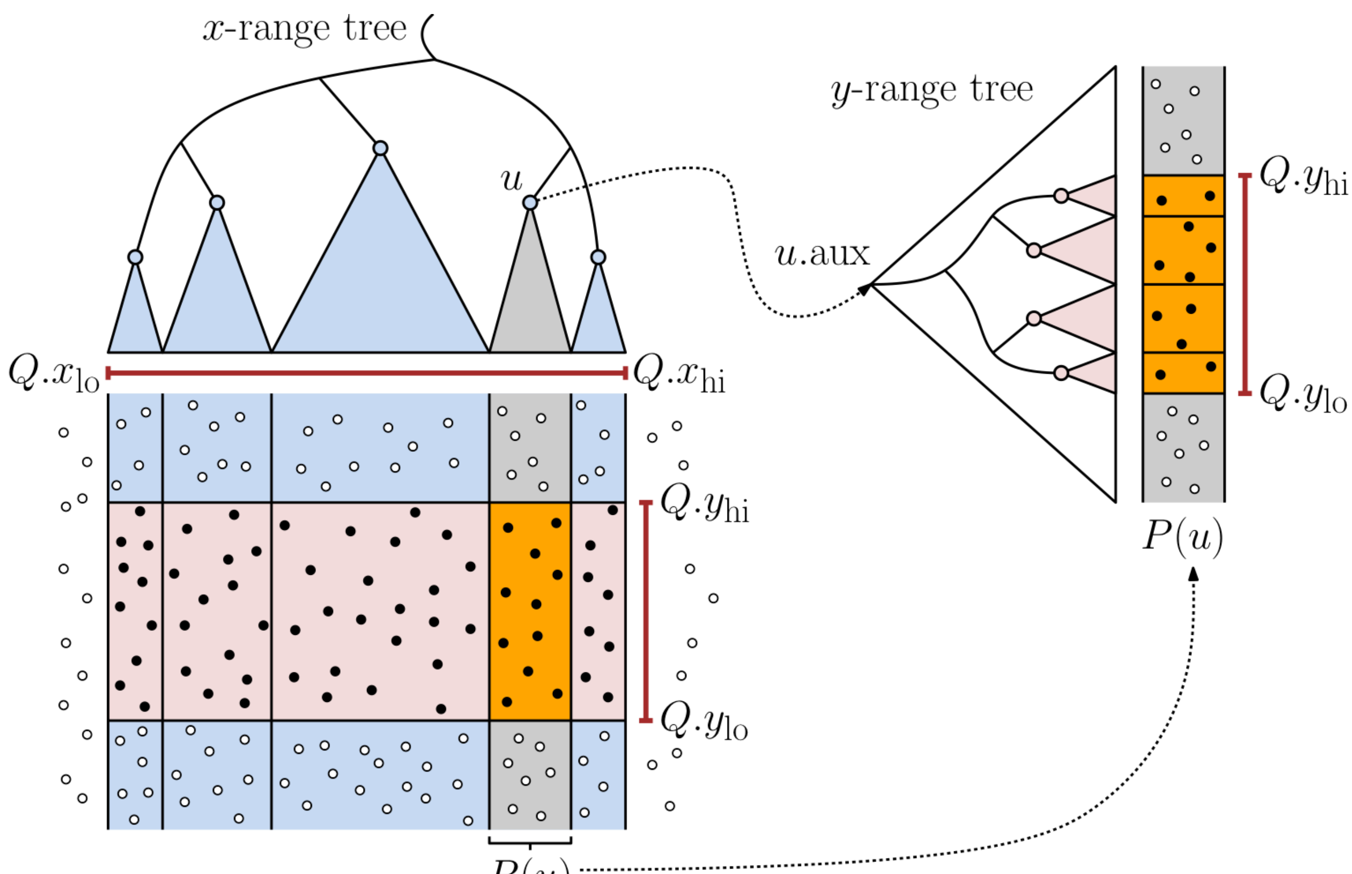
Orthogonal (2-d) Range Tree:

- Given points $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2$
- Build a 1-d range tree for P based on x only (data in leaves)

Main tree

- For each internal node u, let P(u) be points in its leaves (canon. subset)
  - Build a 1-d range tree for P(u) sorted by y-coords.

Aux. tree for u



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

- Apply 1-d search in main tree with query $[Q_{lo.x}, Q_{hi.x}]$ to identify $O(\log n)$ maximal subtrees
- For each root u of one of these max. subtrees apply 1-d search in u.aux with query $[Q_{lo.y}, Q_{hi.y}]$
- Return overall sum

```
range2D(Node u, Range Q, Interval C = [x0, x1])
  if (u is leaf)
    └ return { 1 if u.point ∈ Q
             { 0 o.w.
  else if (Q.x ∩ C = ∅)              (no x overlap)
    └ return 0
  else if (C ⊆ Q.x)                  (containment in x)
    └ return range1Dy(u.aux, Q, [-∞, +∞])
                                     search aux. tree
  else              (recurse)
    └ return range2D(u.left, Q, [x0, u.x])
           + range2D(u.right, Q, [u.x, x1])
```

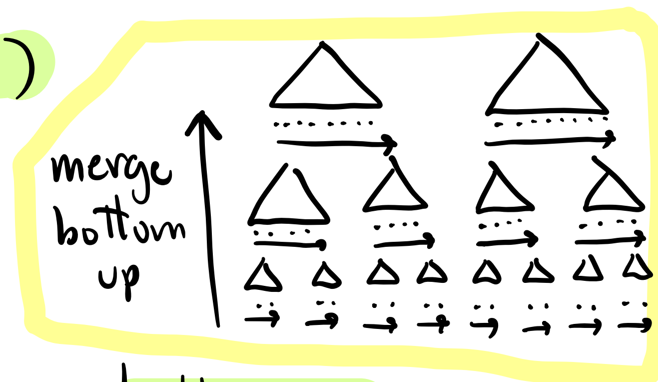
Space + Preprocessing Time:

- Since each node stores $O(1)$ data, total
  space = size of main tree + total size of aux. trees
- A tree with m leaves has size $O(m)$

$$\text{Space} = \underbrace{n}_{\text{main tree}} + \underbrace{\sum_u |P(u)|}_{u.\text{aux tree}}$$

- Main tree's height is $O(\log n)$
- Each leaf contributes a point to $u.\text{aux}$ for each of its ancestors
  - $\Rightarrow$ Each point appears in $O(\log n)$ aux. trees
  - $\Rightarrow \sum_u |P(u)| = O(n \log n)$

$\Rightarrow$ Total space is $O(n \log n)$



Construction time:

Naive: $O(n \log^2 n)$

Better: Build aux trees bottom-up
- Two child sets can be merged in linear time

$\Rightarrow O(n \log n)$

Query Time:

Main tree: $O(\log n)$ time

→ Identifies $O(\log n)$ maximal subtrees

- each has $\leq n$ points
- each searchable in $O(\log n)$ time

$\Rightarrow$ Total time $= O(\log n) \cdot O(\log n)$

$= O(\log^2 n)$

Thm: Using orthogonal range trees, 2-dim orthog. range (counting) queries can be answered in:

$O(n \log n)$ space

$O(n \log n)$ build time

$O(\log^2 n)$ query time → $+k$ for reporting

Thm: Using orthogonal range trees, d-dim orthog. range (counting) queries can be answered in:

$O(n \log^{d-1} n)$ space

$O(n \log^{d-1} n)$ build time

$O(\log^d n)$ query time → $+k$ for reporting

Can we do better?

You can shave off a $\log n$ factor for query times - Cascading Search

2-dim: $O(\log^2 n) \to O(\log n)$

d-dim: $O(\log^d n) \to O(\log^{d-1} n)$

(See latex notes)

Idea:

- Final aux trees can be stored as sorted arrays (trees not needed)
- Always searching for same values: Q.lo.y Q.hi.y
- Can exploit knowledge of answer in one array to find answer in another, without doing search from scratch.