Range counting

Count (or enumerate) objects in a given range (many times)

1D:

2D:
1D: \[ \underbrace{\cdots \cdots} \quad k=6 \quad \underbrace{\cdots \cdots} \]

USE ARRAY: \( O(\log n) \) to place \( L,R \) → to count.
\( O(k + \log n) \) to enumerate/report.

2 problems:
- doesn’t generalize to 2D (no array)
- not dynamic ... insert, delete data: \( O(n) \)
Store size of subtree in each node.
$ID: \quad \overbrace{\langle \ldots \rangle_{L}^{\ldots}}^{k = 6} \underbrace{\langle \ldots \rangle_{R}^{\ldots}}_{\text{count 1}}$
$k = 6$

- □ $\Rightarrow$ count 1
- ○ $\Rightarrow$ count subtree
$O(\log n)$ nodes visited

- 2 paths root→leaf
- 1 neighbor off path per node

○: always "inside"
×: always "outside"
k-d tree (k=2) recursively cut median, alternating $\uparrow \leftrightarrow$

**2D:**

- First cut
- First node
2D:

until every cell has 1 point
Building a 2-d tree

at each level we compute medians. (disjoint) $\Sigma = O(n)$

$O(n \log n)$
3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing

We will only query the crossing leaves explicitly.
3 Types of Leaves (Rectangles)

- Inside
- Outside
- Crossing

Algorithm:

step 1: compare □ to the 2 rectangles of 1st partition
intersects both sides of tree
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

Step 2: compare to 4 rectangles (children)
3 TYPES OF LEAVES (RECTANGLES)
- inside
- outside
- crossing

Step 3: Finally, one rectangle doesn't overlap. Ignore it and its children.
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

Level 4:
- no comparison of \( \square \) to \( \square \)
- ignore new external boxes: \( \square \)
- found internal \( \square \)

\( \rightarrow \) count total points inside & ignore in next levels
3 Types of Leaves (Rectangles)

- inside
- outside
- crossing

Similar to 1D,
- we implicitly count #points in maximally contained rectangles
  & ignore points in exterior rectangles
  (rectangle ~ subtree)
3 Types of Leaves (Rectangles)

- Inside
- Outside
- Crossing

Similar to 1D,
- We implicitly count #points in maximally contained rectangles & ignore points in exterior rectangles.

Unlike 1D,
- Can branch a lot in the balanced BST.
  
  Worst case work \(\approx O(logn) + (#\text{leaves reached})\)

\( (#\text{leaves} \approx #\text{internal}) \)
3 TYPES OF LEAVES (RECTANGLES)

- inside
- outside
- crossing

Dominating factor in worst case work:  \# crossing leaves

Other leaves are either

- in an ignored rectangle
- already counted (ancestor)

represented by nodes adjacent to paths to cr. leaves
3 Types of Leaves (Rectangles)
- inside □
- outside □
- crossing □

Dominating factor in worst case work: \# crossing leaves

Other leaves are either
- in an ignored rectangle
- already counted (ancestor)

\# crossings ≤ 4 * (\text{max\# rectangles stabbed by a line})
2 Examples of Stabbing a 2-d Tree w/ a Line.

Every horizontal node has one subtree untouched.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node: must visit both children.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node: must visit both children.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node must visit both children.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node: must visit both children.

For vertical stabbing line, opposite: h-nodes ↔ v-nodes.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node: must visit both children.

For vertical stabbing line, opposite: h-nodes ↔ v-nodes.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.

Every vertical node: must visit both children.

For vertical stabbing line, opposite: h-nodes ↔ v-nodes.
2 examples of stabbing a 2-d tree w/ a line.

Every horizontal node has one subtree untouched.
Every vertical node: must visit both children.

For vertical stabbing line, opposite: h-nodes $\leftrightarrow$ v-nodes.
1 point per rectangle
$\sqrt{n}$ stabbed

Subdivisions not shown
must visit both subtrees

horizontal stab

guaranteed to skip one subtree

Notice we don't care if \( \times \) means that a rectangle is in or out. Just counting leaves
1 vertical stab
wlog

Never visit more than 2 grandchildren of a visited node.

Compress every other level in tree: explore 2 of 4 sub-trees/children
- Both trees: explore $\frac{1}{2}$ of subtrees
  
  : $O(1)$ work per node

- Branching factor change analysis:
  
  $S(n) \leq 2 \cdot S\left(\frac{n}{4}\right) \leq 2 \cdot 2 \cdot S\left(\frac{n}{16}\right) \leq 2 \cdot 2 \cdot 2 \cdot S\left(\frac{n}{64}\right) \leq \cdots \leq 2^k \cdot S\left(\frac{n}{4^k}\right)$

  ...ends when $k \sim \log_4 n \Rightarrow S(n) \leq 2^{\log_4 n} \cdot S(1) = O(\sqrt{n})$
So any rectangle can intersect only $O(\sqrt{n})$ edges of a 2-d tree.

- 2D range query (counting points in $\square$): $O(\sqrt{n})$ time.
- Reporting: $O(k + \sqrt{n})$

Updating the 2-d tree (just a note)

$\rightarrow$ each branch represents median (by $x$ or by $y$)

$\rightarrow$ +/- one point shifts median over by 1

$\rightarrow$ can be complicated to rebalance

$\rightarrow$ but possible

$\rightarrow$ works in dimension $k$
Another intuitive idea: search $X$, then search $Y$. 
Another intuitive idea: search $X$, then search $Y$. 

Diagram:

- Search $X$ (top of the diagram).
- Search $Y$ (bottom of the diagram).
Another intuitive idea: search X, then search Y

Expensive to Y-sort every possible X-range

$O(n \log n) \cdot O(n^2)$
Remember: any $x$-range can be described using $O(\log n)$ nodes in a tree.

For each node, store a new tree, sorting all contents of subtree by $y$.

(tree of trees)

$\text{size of aux. trees} = \frac{n}{1} + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + 8 \cdot \frac{n}{8} + \ldots + n \cdot 1 \approx n \log n \text{ space}$

-OR-

Every point can only be represented in $O(\log n)$ secondary trees
- Building the tree of trees
  - Build primary tree, by \( x \). \( \rightarrow O(n \log n) \) time
  - Build aux. trees starting from leaves: mergesort
    
    Otherwise, each aux tree separately

\[
\begin{align*}
\text{(root)}: & \quad n \log n + 2 \left( \frac{n}{2} \log \frac{n}{2} \right) + 4 \left( \frac{n}{4} \log \frac{n}{4} \right) + \cdots + n \cdot \alpha(1) \\
\text{(leaves)}: & \quad n \log^2 n
\end{align*}
\]
• Query
  - Search X: identify $O(\log n)$ nodes
  - Search Y: in $O(\log n)$ aux. trees. $\{ O(\log^2 n) $
  - Union the secondary searches
    $O(\log n) + O(\log \frac{n}{2}) + O(\log \frac{n}{4}) + \ldots + O(1)$

• Updating
  $O(\log n)$ primary nodes affected; $O(\log^2 n)$ total

For each □ check if y-range is ok.
For each ○ check y-range of aux.tree
## Range Counting

<table>
<thead>
<tr>
<th>Type</th>
<th>Size of Structure</th>
<th>Query Time</th>
<th>Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>k=2 k-d tree</td>
<td>$\Theta(n)$</td>
<td>$O(\sqrt{n})$</td>
<td>Complicated</td>
</tr>
<tr>
<td>Tree of trees</td>
<td>$\Theta(n\log n)$</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log^2 n)$</td>
</tr>
<tr>
<td>Dimension k $&gt; 3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-d tree</td>
<td>$O(kn)$</td>
<td>$O(kn^{1-1/k})$</td>
<td>Complicated</td>
</tr>
<tr>
<td>(tree of) $k^{k-1}$ trees</td>
<td>$\Theta(n\log^{k-1} n)$</td>
<td>$O(\log^{k-1} n)$</td>
<td>$O(\log^{k-1} n)$</td>
</tr>
</tbody>
</table>