Orthogonal Range Searching

store $n$ pts in $\mathbb{R}^d$

st. given query range $q$, find pts inside $q$

axis-aligned rectangle/box

\[
\text{Static case (no insert/delete)}
\]

\[
\text{Online queries (not known in advance)}
\]

diff. versions – report all, decide emptiness, count,
sum of weights, max weight,
(more advanced: median, weight, closest pair,
report all distinct colors, ...)

1D:

space $S(n) = O(n)$

preprocessing time $P(n) = O(n \log n)$ by sorting
query time $Q(n) = O(\log n)$ for emptiness
$O(\log n + k)$ for reporting
$O(\log n)$ for counting
(optimal in comparison model but can do better in RAM...)

2D?

query type

general (4-sided)

3-sided

2-sided ("dominators")

1-sided
Method 0: "K-d Tree"

- divide by median x
- then median y...
- alternate

```
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
```

(each node corresponds to rectangular cell)

Space \( S(n) = \mathcal{O}(n) \)

preproc \( P(n) = 2P(\frac{n}{2}) + \mathcal{O}(n) \Rightarrow \mathcal{O}(n \log n) \)

by median-finding alg'm

query alg'm, given rectangle \( q \): // say counting
- if \( q \) does not intersect node's cell
  return 0
- else if \( q \) completely contains cell
  return # pts in cell
- else recurse in both children
  return sum

```
\[\text{recurse left}\]
```
```
\[\text{recurse on both}\]
```
```
\[\text{recurse on left}\]
```
Analysis for 1-sided query:

\[ Q_1(n) \leq 2Q_1(\frac{n}{2}) + O(1) \Rightarrow O(n) \text{ bad!} \]

Better analysis:

\[ Q_1(n) \leq 2Q_1(\frac{n}{4}) + O(1) \]

\[ \Rightarrow O(n^{100n^2}) = O(n^n) \]

Cor any horizontal/vertical line intersects \( O(n) \) cells of k-d tree.

Analysis for general 4-sided query:

\[ Q(4) = O(\# \text{ cells visited}) = O(\# \text{ cells crossing } \partial q) \leq 4O(n) = \frac{O(n)}{O(n)} \]

\[ \text{boundary has } k \text{ line segments} \]

\[ \text{Higher-d: } Q_1(n) \leq 2^{d-1}Q_1(\frac{n}{2^d}) + O(1) \]

\[ \Rightarrow O(n^{d-1}) = O(n^{1-\frac{1}{d}}) \]
Method 1: Priority Search Tree (for 3-sided emptiness/reporting)

- Divide by median x
- Remove pt with min y
- Recurse on left & right

\[ S(n) = 1 \text{ if } n \leq 1 \]
\[ P(n) = 2P(n/2) + O(n) \Rightarrow O(n \log n) \]

Query alg'm, given 3-sided rectangle \( q \): 
1. If \( q \) is below \( p_{\text{min}} \), return
2. If \( p_{\text{min}} \) is in \( q \), report \( p_{\text{min}} \)
3. If \( q \) left of median x, recurse left
4. Else if \( q \) right of median x, recurse right
5. Else recurse on both

Analysis: for 1-sided, unbound from below
\[ \Omega_1(n) = O(1) \text{ for emptiness} \]
(p. for reporting, since each node visited reports 1 pt)

For 2-sided
\[ \Omega_2(n) = \begin{cases} \Omega_2(q) + O(1) & \text{if } q \text{ is \leq } \frac{n}{2} \\ \Omega_2(n/2) + \Omega_2(n/2) + O(1) & \Rightarrow O(\log n) \end{cases} \]
for 3-sided:

\[ Q_3(n) \leq \max \left\{ \frac{Q_3(\frac{n}{2}) + O(1)}{2}, \frac{O(\log n)}{\alpha(\log n)} \right\} \]

\[ \Rightarrow O(\log n) \]

(+k for reporting)

**Method 2: Cartesian Tree** (also for 3-sided emptiness/reporting)

- remove pt \( p_{min} \) with \( \min x \)
- divide by \( x \) at \( p_{min} \)
- recurse on left & right

Tree is not balanced!

\[ S(n) = O(n) \]

\[ P(n) = O(n \log n) \] how?

**Query algorithm:**

- to report one pt inside a 3-sided rectangle \( q \):
  - let \( p_0 = x \)-successor of \( q \)'s left side
  - let \( p_1 = x \)-predecessor of \( q \)'s right side
  - find lowest common ancestor (LCA) of \( p_0, p_1 \)
  - solvable in \( O(1) \) time for any binary tree (e.g. Bender & Farach-Colton '00)
to report all pts inside q:
  find one
  recurse left & right

$\Rightarrow \begin{cases} 
O(1 + k) & \text{time if given succ/pred} \\
O(\log n + k) & \text{time else}
\end{cases}$

[note - doesn't generalize to counting or 4-sided ...]

Method 3: Range Tree

- divide by median x
- store pts in sorted y-order
- recurse on left & right

$S(n) = 2S(\frac{n}{2}) + O(n) \Rightarrow O(n \log n)$

$P(n) = 2P(\frac{n}{2}) + O(n \log n) \Rightarrow O(n \log^2 n)$

Each node corresponds to a vertical slab
query alg'm, given rectangle q: // say, counting
if (q doesn't intersect node's slab q) return 0
if (q cuts completely across 0)
do binary search on y-sorted list
recurse on both children
return Sum

Analysis for 3-sided:
$O_3(n) \leq O_3\left(\frac{n}{2}\right) + O(\log n) + \text{binary search}$
$\Rightarrow O(\log^2 n)$

Analysis for 4-sided:
$O_4(n) \leq \max \left\{ O_4\left(\frac{n}{2}\right) + O(1), 2O_3\left(\frac{n}{2}\right) + O(1), O(\log^2 n) \right\}$
$\Rightarrow O(\log n + \log^2 n) = O\left(\log^2 n\right)$
( + k for reporting)

binary search at ≤ 2 log n nodes
$\Rightarrow O(\log^2 n)$
query time can be reduced to $O(\log n)$ (+k for report) by adding pointers between parent list & child list

$P_1 P_2 P_4 P_3 P_5 P_6 P_7 \ldots$

only need one initial binary search at root

Higher d:

$S_d(n) = 2S_d\left(\frac{n}{2}\right) + S_{d-1}(n)$

$\Rightarrow S_d(n) = O\left(S_{d-1}(n) \log n\right)$

$\Rightarrow S_d(n) = O\left(n \log^{d-1} n\right)$

$Q_d(n) = O\left(Q_{d-1}(n) \log n\right)$

$\Rightarrow Q_d(n) = O\left(\log^{d-1} n\right) (+k)$

Trade-offs:

degree-b range tree

$S(n) = O\left(n \left(\log_b n\right)^{d-1}\right)$

$Q(n) = O\left(\left(\log_b n\right)^{d-1} \log n\right)$

or $S(n) = O\left(n \left(b^2 \log_b n\right)^{d-1}\right)$

$Q(n) = O\left(\left(\log_b n\right)^{d-1} \log n\right)$

3D range tree

"multi-level" data structure

[lower bd in semigroup model, ptr machine model,...]