Range Searching: (Data structure problem)
- Given a point set \( P = \{ p_1, \ldots, p_n \} \subset \mathbb{R}^d \)
- Given a class of shapes (e.g., rectangles, balls, triangles, halfspaces)
- Build a data structure so that:
  - Given any query region \( Q \) from the class, quickly identify the points of \( P \) in \( Q \)

- Points (data structure) is (relatively) static
- Queries must be answered fast! (sublinear time)

\[ P + \text{shape = rectangles} \]
What types of Queries?

- **Emptiness**: Any pts of \( P \) in \( Q \)?

- **Counting**: How many? \( |P \cap Q| \)

- **Weighted count**: Each \( p \in P \) has weight \( w(p) \). Return total weight \( \sum_{p \in P \cap Q} w(p) \)

- **Semigroup weight**: Any commutative + associative function of wts:

  \[ \text{Eg. Max-query: } \max_{p \in P \cap Q} w(p) \]

- **Reporting**: List the pts of \( P \cap Q \)

- **Top-k**: List just the highest \( k \) pts of \( P \cap Q \) based on weights
**Complexity Bounds:**

**Space:** Total space needed to store points + data structure

**Query time:** Time needed to answer a query

**Construction time:** Time to build structure

**Common:** (Space bound) \( O(\log n) \)

**Gold standard:**
\( O(n) \) space  
\( O(\log n) \) query time  
\( O(n \log n) \) constr. time

Many geometric structures are \textbf{inferior} w.r.t. space:
\( O(n \log^2 n) \)  
\( O(n \log^d n) \) in \( \mathbb{R}^d \)  
\( O(n^2) \)

or \textbf{Query time}:
\( O(\log^2 n) \)  
\( O(\sqrt{n}) \)  
\( O(n^{1-\frac{1}{d}}) \) in \( \mathbb{R}^d \)
Orthogonal Range Queries:

Query region is \textbf{axis-aligned rectangle}

Eg. Given pts \(a, b \in \mathbb{R}^d\) s.t. \(a_i < b_i \ \forall i\)

\[
Q(a, b) = \{ p \in \mathbb{R}^d \mid a_i \leq p_i \leq b_i \}
\]

= \([a_1, b_1] \times \ldots \times [a_d, b_d]\)

Common in database queries:

How many patients with age \(\in [25, 35]\)
weight \(\in [100, 200]\)
blood pressure \(\in [80, 120]\)
General approach to answering range queries:

- Too slow to count pts one by one
- Too much space to precompute answer to every possible query

- Canonical subsets:
  Carefully select an (ideally small) collection of subsets of $P$ so that the answer to any query can be formed as (disjoint) union of a small number of subsets.

Example: 1-dimensional range query

$P = p_1 < p_2 < \ldots < p_n$ in $\mathbb{R}$

- Store $P$ as leaves of a balanced tree
- Leaves of each subtree form canonical set
The answer to any 1-dim range query can be expressed as the disjoint union of $O(\log n)$ canonical subsets.

**Example:** $Q = [x_{lo}, x_{hi}] = [2, 23]$

$P \cap Q = \{3, 5\} \cup \{1, 3, 11\} \cup \{9, 12, 14, 15\} \cup \{17, 20\} \cup \{22\}$

- Cover the range with maximal subtrees
- Take union of the associated canonical subsets
- $O(\log n)$ subtrees always suffice.
- $O(n)$ nodes $\Rightarrow$ $O(n)$ canonical subsets

Compose the Answer to Query from Subsets:

Counting query: Node stores # of leaves
Weighted count: Node stores total weight of leaves
Max query: Node stores max of all weights in leaves

Can answer queries in $O(\log n)$ time by combining subtree results (assuming you can identify the canonical subsets for query and precompute info.)
**kd-Trees**: A natural generalization of 1-d trees to higher dim

1-d tree, 2-d tree, ..., k-d tree

Jon Bentley (1975)

Numerous variants - we present one

- Assume have large bounding box \( B \) containing \( P \)

- Recursively split space by axis-orthogonal hyperplane

**cutting dimension**: which axis

**cutting value**: where to cut

Spatial subdivision

Tree structure

**Cell**: Each tree node represents a rectangular region
Design choices:
- Where are points stored?
  - internal nodes (used for splitting)
  - external nodes (leaves)
  - Permits more flexibility in where to split
- How is cutting dim chosen?
  - alternate: $x,y,x,y,...$ or $x,y,z,x,y,z,...$
  - select based on point distribution
- How is cutting value chosen?
  - median (balanced height)
  - mid pt (geom. balanced)

Our structure:
- Points stored at leaves (external nodes)
- Alternate splitting axes
- Split at median
**Construction:**

Tree can be built in \( O(n \log n) \) time

\[
T(n) = n + 2T\left(\frac{n}{2}\right) \quad \text{find median splitting coord recursively build subtrees}
\]

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

**Slight improvement:** Presort the points \( d \) times into \( d \) lists - one for each coordinate + cross-link entries

- Faster in practice

**Space:** \( O(n) \)

- \( n \) leaves (one per point)
- \( (n-1) \) internal nodes
- \( O(1) \) info per per node

**Range Search:**

Key: If node's cell does not overlap \( Q \rightarrow \) Don't visit

If node's cell completely in \( Q \rightarrow \) count all its pts
Algorithm: Weighted range count in kd-tree

\[
\text{range-count}(\text{Rect } Q, \text{ KdNode } u) \\
\text{if ( } u \text{ is leaf) } \\
\quad \text{if ( } u.\text{point} \in Q \text{) return } u.\text{point}.\text{weight} \\
\quad \text{else return } 0 \\
\text{else ( } u \text{ is internal) } \\
\quad \text{if ( } u.\text{cell} \cap Q = \emptyset \text{) } \\
\quad \quad \text{return } 0 \text{ (no overlap) } \\
\quad \text{else if ( } u.\text{cell} \subseteq Q \text{) } \\
\quad \quad \text{return } u.\text{weight} \text{ (total weight) } \\
\quad \text{else } \\
\quad \quad \text{return range-count}(Q, u.\text{left}) + \text{range-count}(Q, u.\text{right})
\]
Example:

Query Time:

Thm: Given a height-balanced kd-tree in $\mathbb{R}^2$ using alternating splitting axes, orthogonal counting queries can be answered in $O(\sqrt{n})$ time.

[Reporting queries in time $O(k + \sqrt{n})$, where $k = \#$ of points reported.]

Proof: Query rectangle bounded by 4 lines

We’ll show that each line stabs

$\leq \sqrt{n}$ cells of tree $\Rightarrow O(4\sqrt{n})$
Key: Because we alternate cutting dim for every 2 levels of tree, any axis parallel line can stab at most 2 out of 4 grandchild cells.

Since we use balanced splitting:
- Parent has \( n \) pts
- Child has \( \frac{n}{2} \) pts
- Grandchild has \( \frac{n}{4} \) pts

\[ T(n) = 2T\left(\frac{n}{4}\right) + 1 \]

- Recurse on 2 of 4 grandchildren
- Constant time per cell

\[ = O(\sqrt{n}) \quad [\text{see lect. notes for details}] \]