Orthogonal range searching

Input: \( n \) points in \( d \) dimensions
- e.g., representing a database of \( n \) records each with \( d \) numeric fields

Query: Axis-aligned box (in 2D, a rectangle)
- Report on the points inside the box:
  - Are there any points?
  - How many are there?
  - List the points.

Goal:
Preprocess points into a data structure to support fast queries
- Primary goal: Static data structure
- In 1D, we will also obtain a dynamic data structure supporting insert and delete

1D range searching

In 1D, the query is an interval:

First solution using ideas we know:
- Interval trees
- Represent each point \( x \) by the interval \([x, x]\).
- Obtain a dynamic structure that can list \( k \) answers in a query in \( O(k \lg n) \) time.

Goal:
Obtain a dynamic structure that can list \( k \) answers in a query in \( O(k + \lg n) \) time.

Second solution using ideas we know:
- Sort the points and store them in an array
- Solve query by binary search on endpoints.
- Obtain a static structure that can list \( k \) answers in a query in \( O(k + \lg n) \) time.

New solution that extends to higher dimensions:
- Balanced binary search tree
- New organization principle:
  Store points in the leaves of the tree.
- Internal nodes store copies of the leaves to satisfy binary search property:
  Node \( x \) stores in \( key[x] \) the maximum key of any leaf in the left subtree of \( x \).
Example of a 1D range tree

Example of a 1D range query

Pseudocode, part 1: Find the split node

Pseudocode, part 2: Traverse left and right from split node
Analysis of 1D-RANGE-QUERY

**Query time:** Answer to range query represented by $O(lg n)$ subtrees found in $O(lg n)$ time. Thus:
- Can test for points in interval in $O(lg n)$ time.
- Can count points in interval in $O(lg n)$ time if we augment the tree with subtree sizes.
- Can report the first $k$ points in interval in $O(k + lg n)$ time.

**Space:** $O(n)$

**Preprocessing time:** $O(n lg n)$

2D range trees

Store a primary 1D range tree for all the points based on x-coordinate. Thus in $O(lg n)$ time we can find $O(lg n)$ subtrees representing the points with proper x-coordinate. How to restrict to points with proper y-coordinate?

2D range trees

**Idea:** In primary 1D range tree of x-coordinate, every node stores a secondary 1D range tree based on y-coordinate for all points in the subtree of the node. Recursively search within each.

Analysis of 2D range trees

**Query time:** In $O((lg n)^2)$ time, we can represent the answer to range query by $O((lg n)^2)$ subtrees. Total cost for reporting $k$ points: $O(k + (lg n)^2)$.

**Space:** The secondary trees at each level of the primary tree together store a copy of the points. Also, each point is present in each secondary tree along the path from the leaf to the root. Either way, we obtain that the space is $O(n lg n)$.

**Preprocessing time:** $O(n lg n)$

d-dimensional range trees

Each node of the secondary y-structure stores a tertiary z-structure representing the points in the subtree rooted at the node, etc.

**Query time:** $O(k + (lg n)^2)$ to report $k$ points.

**Space:** $O(n (lg n)^{d-1})$

**Preprocessing time:** $O(n (lg n)^{d-1})$

Best data structure to date:

**Query time:** $O(k + (lg n)^{d-1})$ to report $k$ points.

**Space:** $O(n (lg n / lg lg n)^{d-1})$

**Preprocessing time:** $O(n (lg n)^{d-1})$