Skyline Query Computation in Multidimensional Space

Sameer Mannava (210070073), Satyam Mishra (210040139)

1 Problem Statement

Let P be a set of n points in R^d. We say that a point p ϵ R^d dominates a point q ϵ Rd if for every coordinate i ϵ 1, 2, . . . , d, we have $p_i > q_i$. A point p is called a maximal point if no other point in P dominates it. The goal is to report all the maximal (or skyline) points in P.

- Design an algorithm to compute the skyline points in $O(n \log^d n)$ time. (Hint: Use range trees.)
- Improve the algorithm for the case when the input points lie in R^3, and compute the skyline points in $O(n \log n)$ time.

2 Part A: General d-Dimensional Skyline Algorithm

2.1 Algorithm Approach

For the general d-dimensional case, we implemented a Range Tree-based approach with the following components:

1. **Range Tree Construction**: A multidimensional range tree is built to efficiently query points that dominate a given point.
2. **Domination Queries**: For each point, we query the range tree to check if there exists any point that dominates it.
3. **Skyline Extraction**: Points that are not dominated by any other point form the skyline.

The range tree is implemented as a recursive data structure, where each node contains a split value for the current dimension and an associated structure for the next dimension. This allows for efficient orthogonal range queries.

2.2 Time Complexity Analysis

The theoretical time complexity of the range tree-based algorithm is $O(n \log^d n)$, where n is the number of points and d is the number of dimensions. This complexity can be broken down into several components:

1. **Range Tree Construction**: Building a d-dimensional range tree requires $O(n \log^{d-1} n)$ time.
2. **Domination Queries**: For each point, we perform a domination query that takes $O(\log^d n)$ time.
   - The query traverses at most $O(\log n)$ nodes at each dimension.

- At the leaf level (dimension d), the algorithm performs a standard BST query.
- Across all dimensions, this leads to $O(\log^d n)$ per query.

3. **Total Complexity**: With n points, each requiring a domination query, the total query time is $O(n \log^d n)$.

Hence, we get an overall complexity of $O(n \log^d n)$.

2.3 Pseudocode

Provided below is the pseudocode for the implemented algorithm:

Algorithm 1 BuildRangeTree

```
Require: $points$: Set of points, $d$: current dimension, $max\_d$: maximum dimension
Ensure: A range tree for the points
if $|points| \le 3$ then
    return RangeTreeLeaf($points$)
end if
if $d = max\_d - 1$ then
    $points\_sorted \leftarrow$ SortByDimension($points, d$)
    return Build1DBST($points\_sorted, d$)
end if
$points\_sorted \leftarrow$ SortByDimension($points, d$)
$n \leftarrow |points\_sorted|$
$median\_idx \leftarrow \lfloor n/2 \rfloor$
$median\_val \leftarrow points\_sorted[median\_idx][d]$
if AllPointsHaveSameValue($points\_sorted, d$) then
    if $d + 1 < max\_d$ then
        return BuildRangeTree($points\_sorted, d + 1, max\_d$)
    else
        return RangeTreeLeaf($points\_sorted$)
    end if
end if
Find proper split index (rightmost occurrence of $median\_val$)
$node \leftarrow$ RangeTreeNode($median\_val$)
$left\_points \leftarrow points\_sorted[0 \dots split\_idx]$
$right\_points \leftarrow points\_sorted[split\_idx + 1 \dots n - 1]$
if $left\_points \ne \emptyset$ then
    $node.left \leftarrow$ BuildRangeTree($left\_points, d, max\_d$)
end if
if $right\_points \ne \emptyset$ then
    $node.right \leftarrow$ BuildRangeTree($right\_points, d, max\_d$)
end if
if $d + 1 < max\_d$ then
    $node.assoc\_struct \leftarrow$ BuildRangeTree($points\_sorted, d + 1, max\_d$)
end if
return $node$
```

Algorithm 2 QueryDominatorExists

```
Require: $node$: Range tree node, $point$: Query point, $d$: current dimension, $max\_d$: maximum dimension
Ensure: True if a dominator exists, False otherwise
 1: if $node = null$ then
 2:     return False
 3: end if
 4: if $node$ is RangeTreeLeaf then
 5:     for each $p$ in $node.points$ do
 6:         if $p \neq point$ and $p$ dominates $point$ then
 7:             return True
 8:         end if
 9:     end for
10:     return False
11: end if
12: if $d = max\_d - 1$ then
13:     return Query1DBSTGreaterThan($node, point[d], d$)
14: end if
15: if $node.split\_val = null$ then
16:     if $node.assoc\_struct \neq null$ then
17:         return QueryDominatorExists($node.assoc\_struct, point, d + 1, max\_d$)
18:     end if
19:     return False
20: end if
21: $p\_coord \leftarrow point[d]$
22: if $p\_coord < node.split\_val$ then
23:     if $node.right \neq null$ and $node.right.assoc\_struct \neq null$ then
24:         if QueryDominatorExists($node.right.assoc\_struct, point, d + 1, max\_d$) then
25:             return True
26:         end if
27:     end if
28:     return QueryDominatorExists($node.left, point, d, max\_d$)
29: else
30:     return QueryDominatorExists($node.right, point, d, max\_d$)
31: end if
```

Algorithm 3 FindSkyline

```
Require: $points$: Set of points
Ensure: The skyline of $points$
 1: $unique\_points \leftarrow$ RemoveDuplicates($points$)
 2: $n \leftarrow |unique\_points|$
 3: $d \leftarrow$ Dimensionality($unique\_points[0]$)
 4: $root \leftarrow$ BuildRangeTree($unique\_points, 0, d$)
 5: $skyline \leftarrow \emptyset$
 6: for each $p$ in $unique\_points$ do
 7:     $is\_dominated \leftarrow$ QueryDominatorExists($root, p, 0, d$)
 8:     if not $is\_dominated$ then
 9:         $skyline \leftarrow skyline \cup \{p\}$
10:     end if
11: end for
12: return $skyline$
```

3 Part B: Specialized 3D Skyline Algorithm

3.1 Algorithm Approach

For the 3D case, we implemented a more efficient algorithm that leverages the specific properties of 3D data:

1. **Z-Coordinate Sorting**: Points are sorted by z-coordinate in descending order (and then by x and y in ascending order for tie-breaking).
2. **2D Skyline Maintenance**: As points are processed in z-order, a 2D skyline (considering only x and y coordinates) is maintained.
3. **Dominance Testing**: Each point is checked against the current 2D skyline to determine if it is dominated.
4. **Skyline Update**: Non-dominated points are added to both the 3D skyline result and the 2D skyline (after removing any 2D skyline points they dominate).

This approach leverages the fact that points with higher z-coordinates can only be dominated by points that also dominate them in the 2D projection.

3.2 Time Complexity

The theoretical time complexity of the 3D algorithm is $O(n \log n)$, where n is the number of points. This is a significant improvement over the general algorithm's $O(n \log^3 n)$ complexity for the 3D case.

The complexity breakdown is:

- Sorting: $O(n \log n)$
- Binary search insertions: $O(n \log n)$ across all points
- Dominance checks: $O(n \log n)$ in total

4 Experimental Time Complexity Analysis

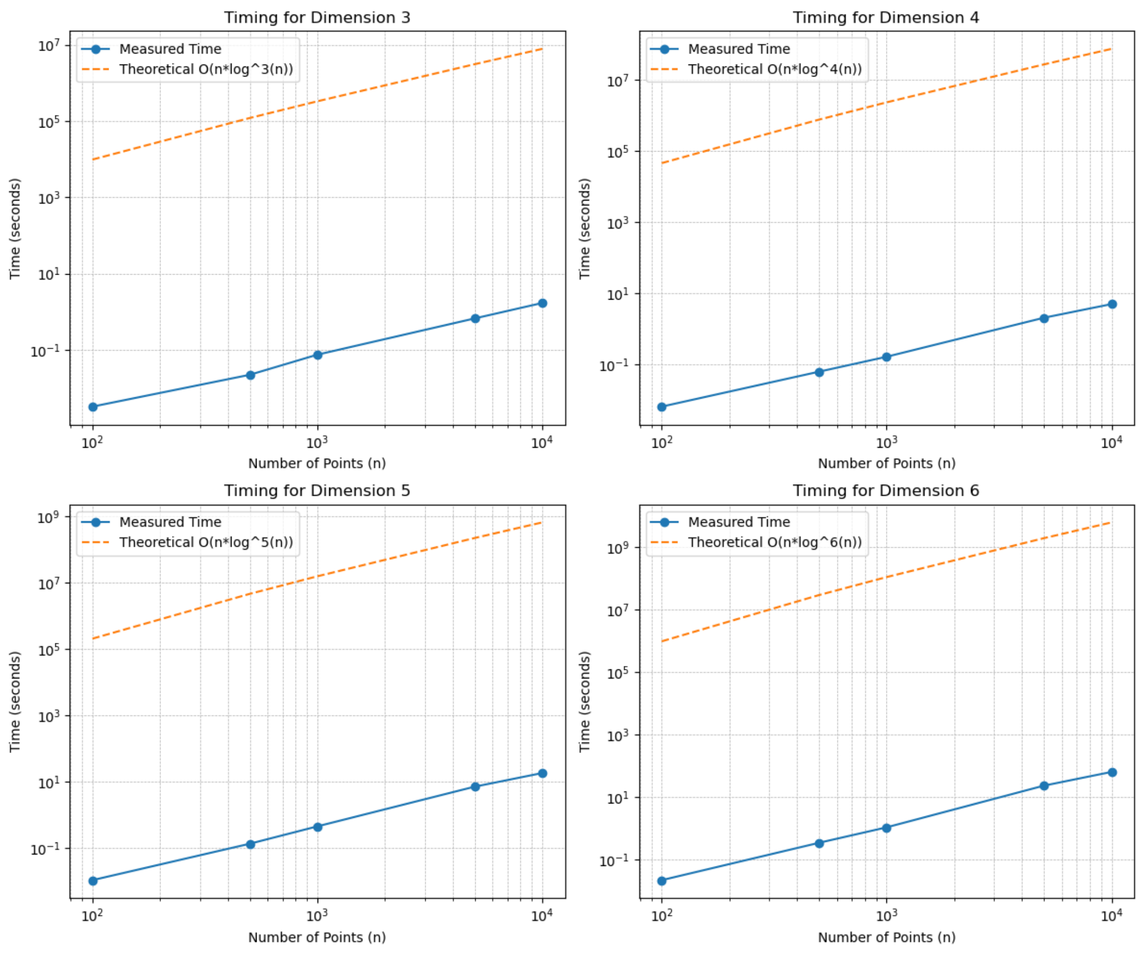

Figure 1: Timing tests for part A

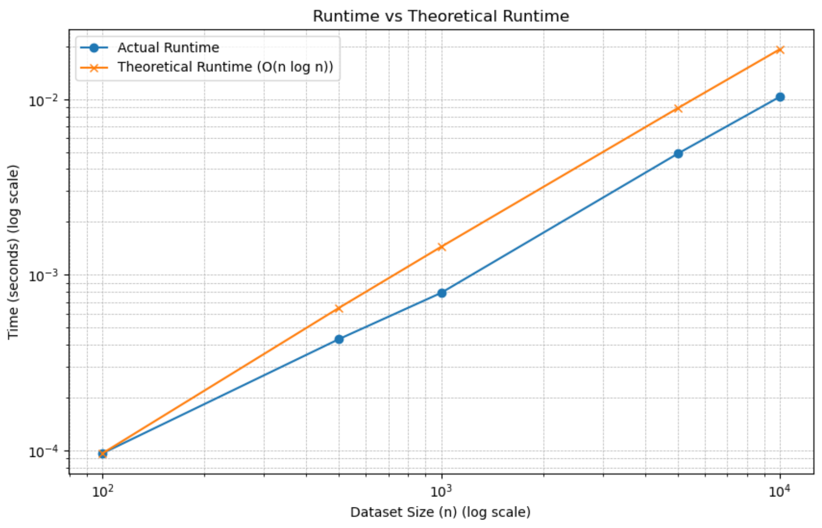

Figure 2: Timing tests for part B