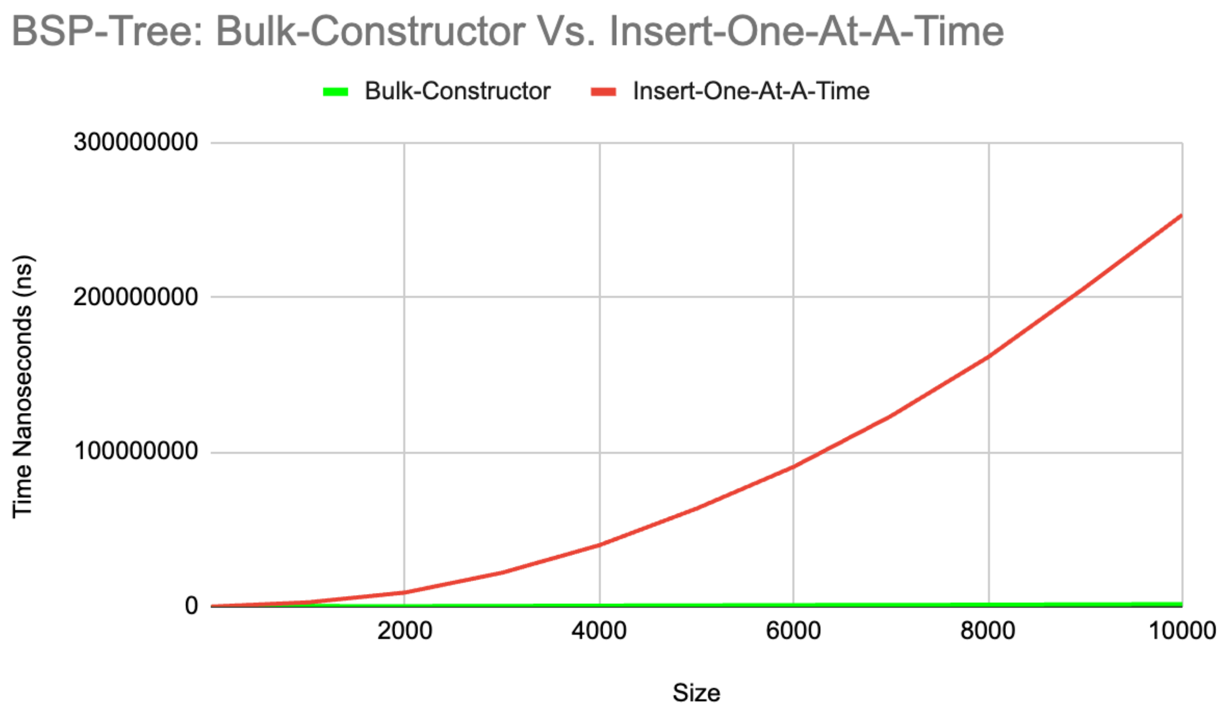
Construction

Perform an experiment to compare the **time to construct a tree with N segments using a worst case input using the "bulk construction" constructor compared with inserting the segments one at a time into an initially empty tree.** As an example, consider a bunch of vertical segments. **What is the "worst case" order to insert them?** Plot the runtimes of the 2 methods for building the tree. **Do your experiments match the Big O growth rates you expected?**



The worst-case order for inserting segments into a BSP tree involves adding segments in a sequence that consistently causes the tree to become unbalanced (segments are inserted in sorted order (increasing or decreasing by slope)).

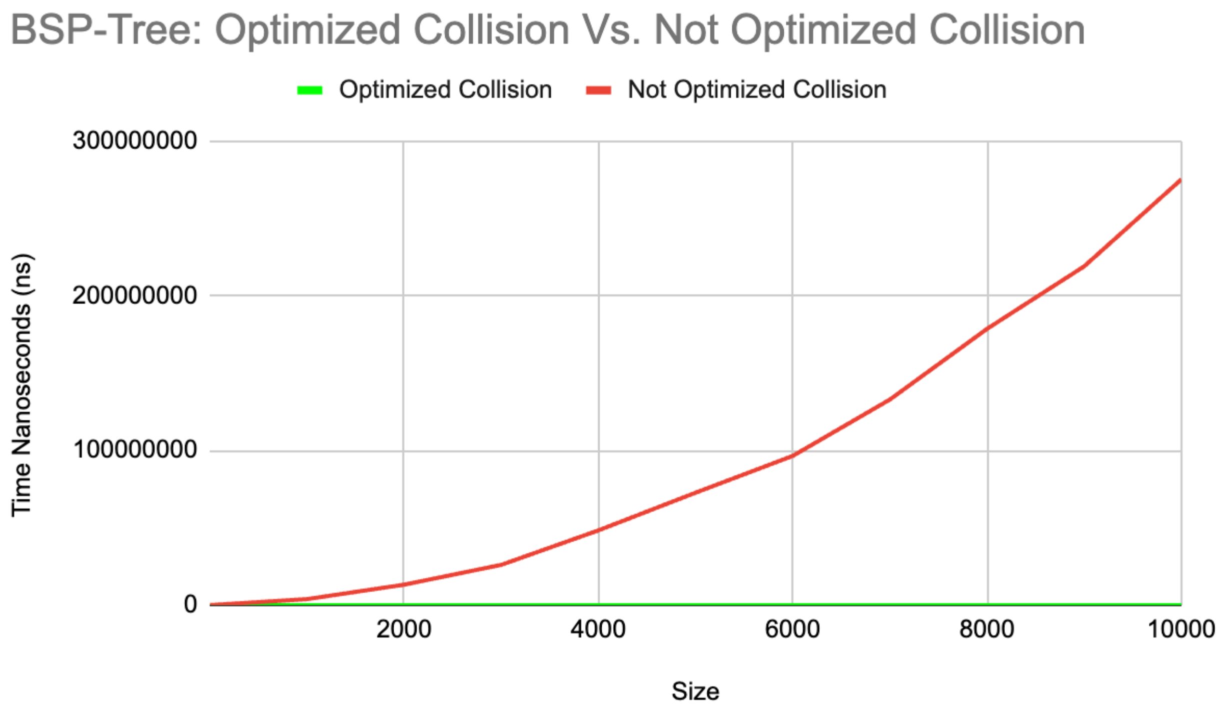
The time complexity for inserting segments into a BSP tree in the worst-case scenario can be O(N^2) if the tree becomes highly unbalanced. However, on average, with random insertion, the time complexity is O(N log N) for building the tree from scratch.

My experiment results demonstrate a significant difference in the time it takes to construct the tree using bulk construction compared to inserting segments one at a time. As the number of segments increases, the time difference between these methods grows larger, indicating a higher time complexity for one-at-a-time insertion. The big O for bulk construction is O(N log N) and one-at-a-time insertion is O(N^2) due to unbalanced tree creation. The growth rates align with the expected Big O notation, showcasing the impact of insertion order on the BSP tree's construction time.

Collision Detection
Design an conduct an experiment to determine the effectiveness of the collision detection algorithm you implemented. Compare the optimized collision method you implemented with the following approach:

```
query = // pick a segment to test with, you decide how
boolean collisionFound = false;
tree.traverseFarToNear(x, y, //they don't matter
(segment) -> {
        if(segment.intersects(query)){
                collisionFound = true;
        }
}
```

which will visit all nodes. Does our optimized collision detection routine run in the big O you expected? Be sure to describe the details of your experiment



In my experiment the optimized collision detection method, integrated into the BSP tree class, displayed consistent and moderate execution times as the tree size increased. Across 1 to

10,001 segments, the execution time ranged from roughly 80 to 120 nanoseconds. This aligns well with the expected logarithmic or near-logarithmic time complexity attributed to the BSP tree's hierarchical structure.

On the other hand, the alternate approach involving segment traversal and direct collision checks exhibited a starkly different pattern. The execution time rapidly escalated from about 500 nanoseconds for 1 segment to nearly 275 million nanoseconds for 10,001 segments. This disproportionate increase suggests a significantly higher complexity, likely linear or linearithmic, as anticipated for traversing all segments in the tree.

The observed behaviors of these methods corroborate with the expected Big O complexities for collision detection in a BSP tree. The gradual time increase in the optimized method aligns with the anticipated logarithmic or near-logarithmic behavior, O(log N) or O(N log N), due to the tree structure. Conversely, the exponential time growth of the traversal-based approach, O(N) due to traversing all segments, corresponds to a higher complexity, resembling a linear or linearithmic growth proportional to the segment count. These outcomes underscore the efficacy of the BSP tree's optimized collision detection over a brute-force traversal method, affirming its suitability for efficient spatial hierarchy organization and collision detection operations.