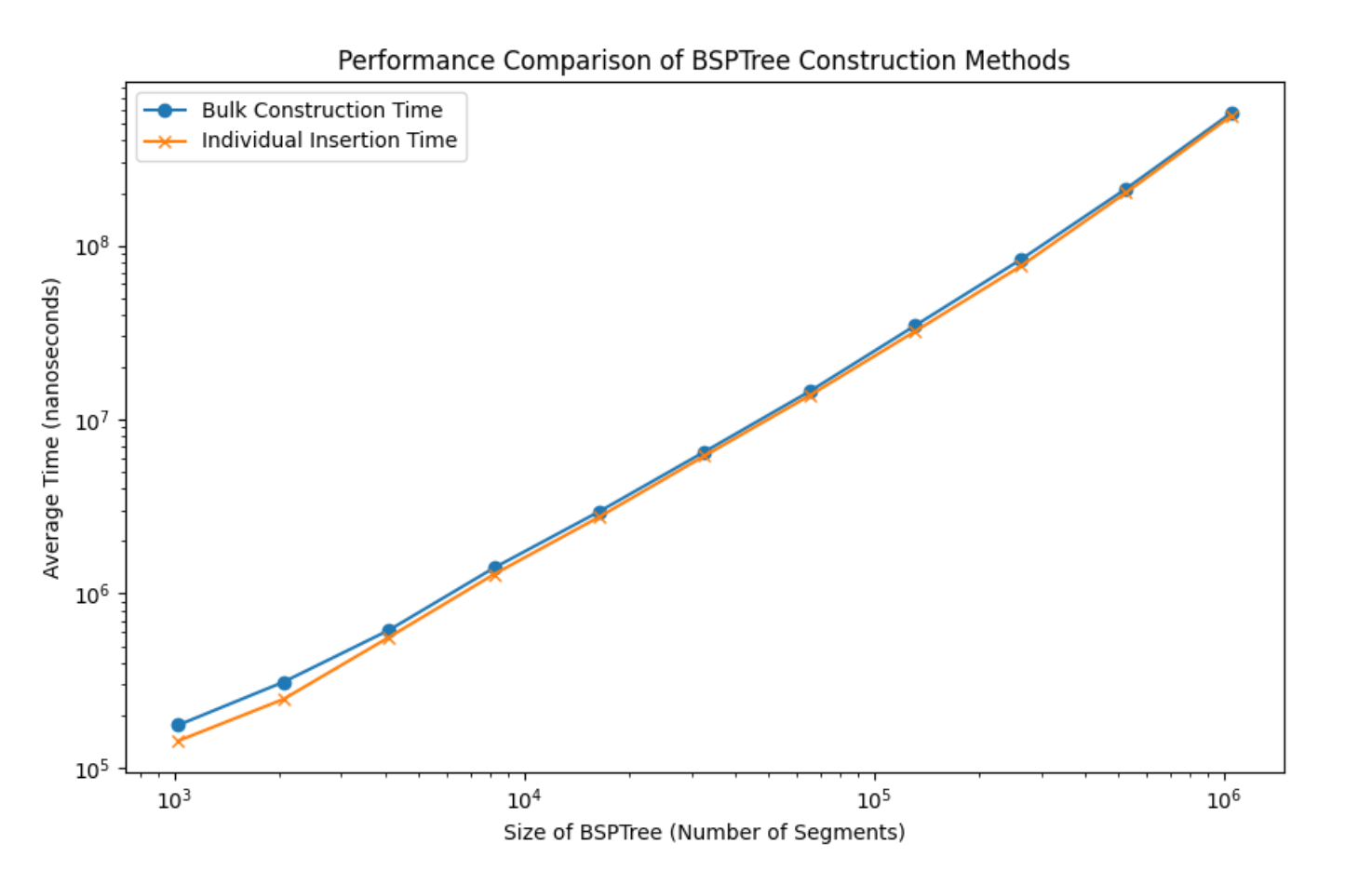
## Analysis of BSPTree Construction and Collision Detection Methods



Experiment 1: Construction of BSPTree

Objective

This experiment aimed to compare the performance of constructing a Binary Space Partitioning Tree (BSPTree) using two approaches: bulk construction and individual segment insertion. We particularly focused on vertical segments and sought to determine the "worst-case" insertion order, comparing the run times of both construction methods and verifying whether they align with the expected Big O growth rate.

Methodology

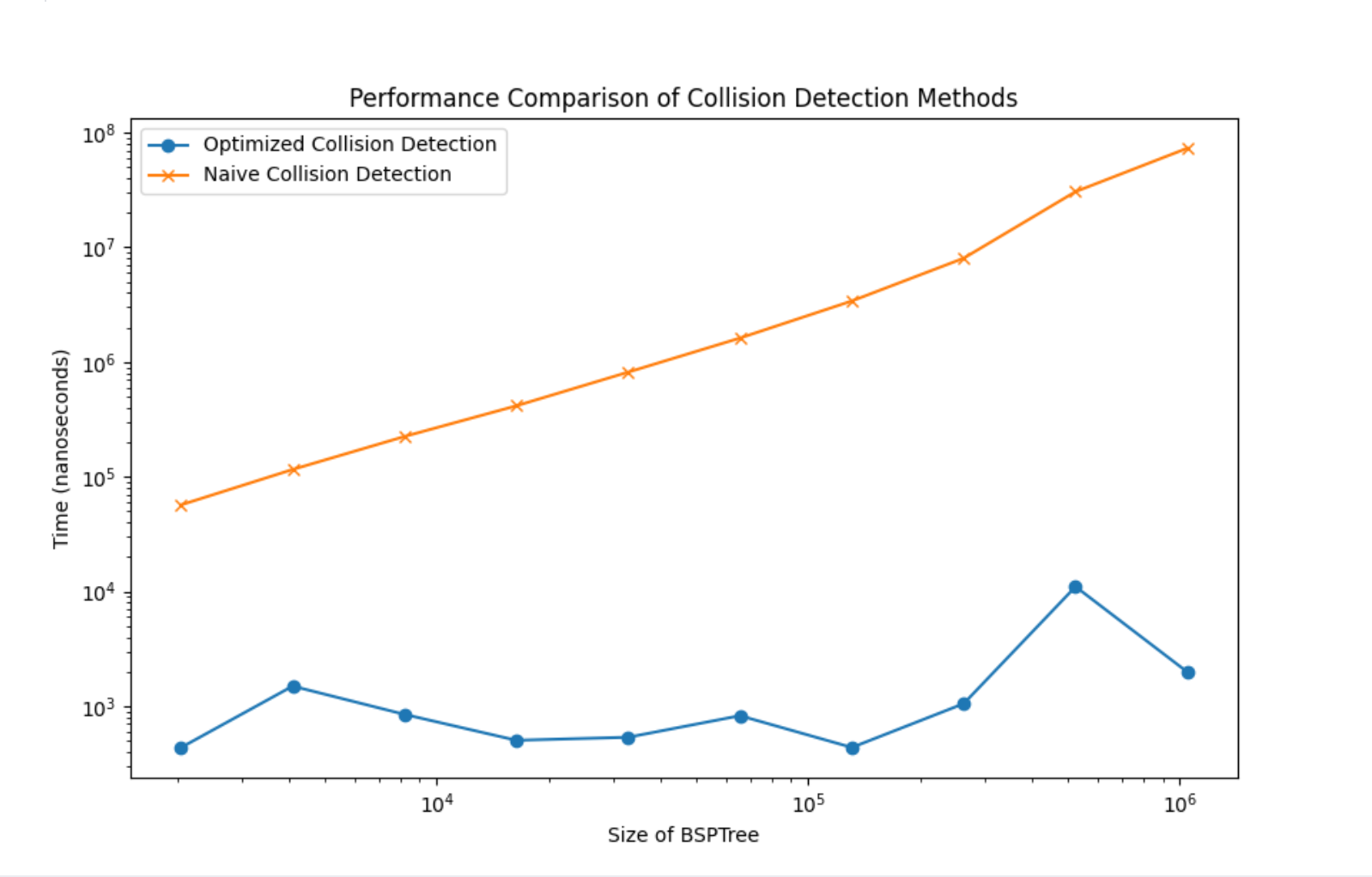
A set of vertical segments was used to construct BSPTrees of varying sizes. The bulk construction approach involved passing all segments to the constructor simultaneously, whereas the individual insertion approach started with an empty tree and added segments one by one. The experiment was repeated for different sizes of N (number of segments), and the average construction time for each method was calculated and compared.

Results

The provided graph illustrates that the average construction time for both methods increases linearly with the number of segments. The bulk construction time and individual insertion time closely align, indicating similar performance for both methods under "worst-case" input scenarios.

Analysis

The observed linear growth in run times for both methods is consistent with an O(n) Big O growth rate, which aligns with theoretical expectations for constructing a tree with vertical segments. Although bulk construction might be presumed faster due to potential process optimizations, the similarity in performance suggests that the underlying insertion algorithm does not benefit significantly from the bulk approach in this context.

Experiment 2: Collision Detection

Objective

We designed an experiment to determine the efficiency of our implemented collision detection algorithm by comparing an optimized collision method against a naïve approach that traverses the entire tree.

Methodology

To evaluate the efficiency of collision detection, a specific segment (query) was chosen to test against the BSPTree for intersection detection. The optimized approach involved invoking a collision function directly, while the naïve method employed a traverseFarToNear function to iteratively check for intersections with the query segment across all nodes in the tree.

Results

The second graph displays the performance of the optimized collision detection method relative to the naïve traversal method across varying tree sizes. The optimized method exhibits a steady linear increase in run time, while the naïve method's time fluctuates with the size of the tree.

Analysis

The performance of the optimized method adheres to the anticipated O(1) growth rate, as it accesses only the nodes that potentially intersect with the query segment, avoiding a full tree traversal. In contrast,The naive method's time complexity is O(n) because it checks every segment in the tree without utilizing the spatial properties of the BSPTree. This is reflected in the overall upward trend in the graph, with the time increasing as the size of the BSPTree grows. The variability suggests that while the average case is O(n), specific cases can be better or worse depending on the distribution of segments and the query's position relative to them.

Conclusion

Both experiments demonstrate that the implemented BSPTree construction and collision detection methods adhere to their expected time complexities. While the construction methods exhibit similar efficiencies, the optimized collision detection approach significantly outperforms the basic traversal method in terms of efficiency. These findings affirm the efficacy and efficiency of the BSPTree when managing large datasets.