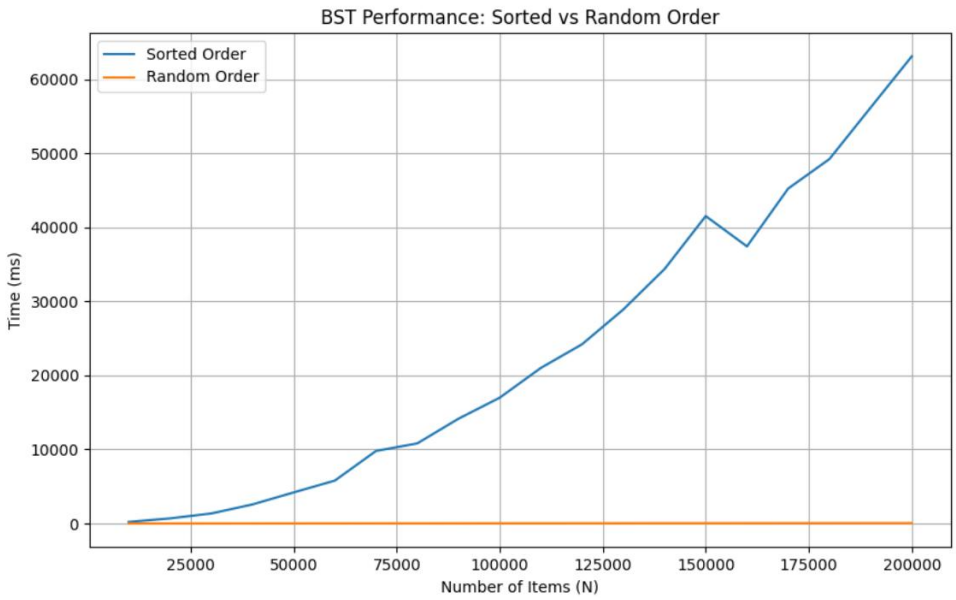
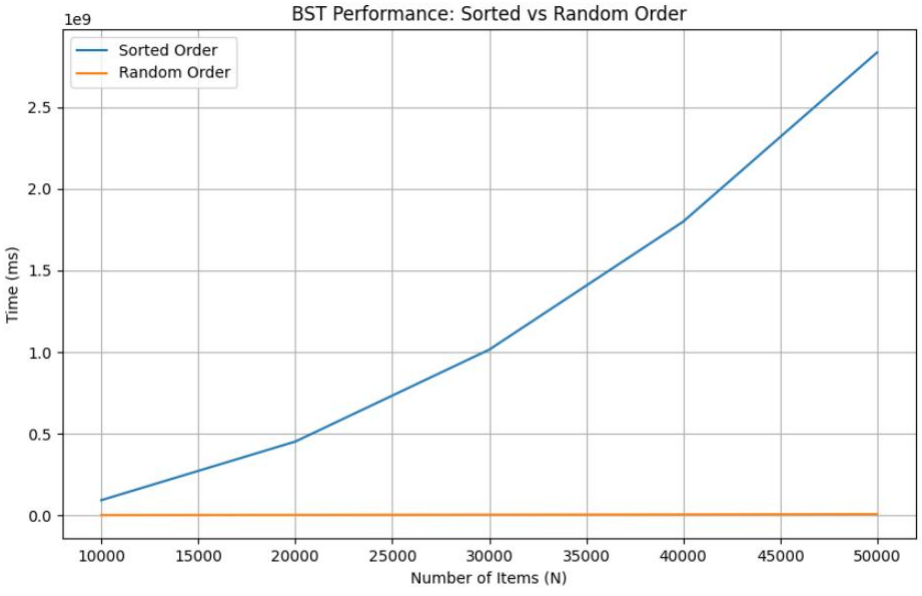
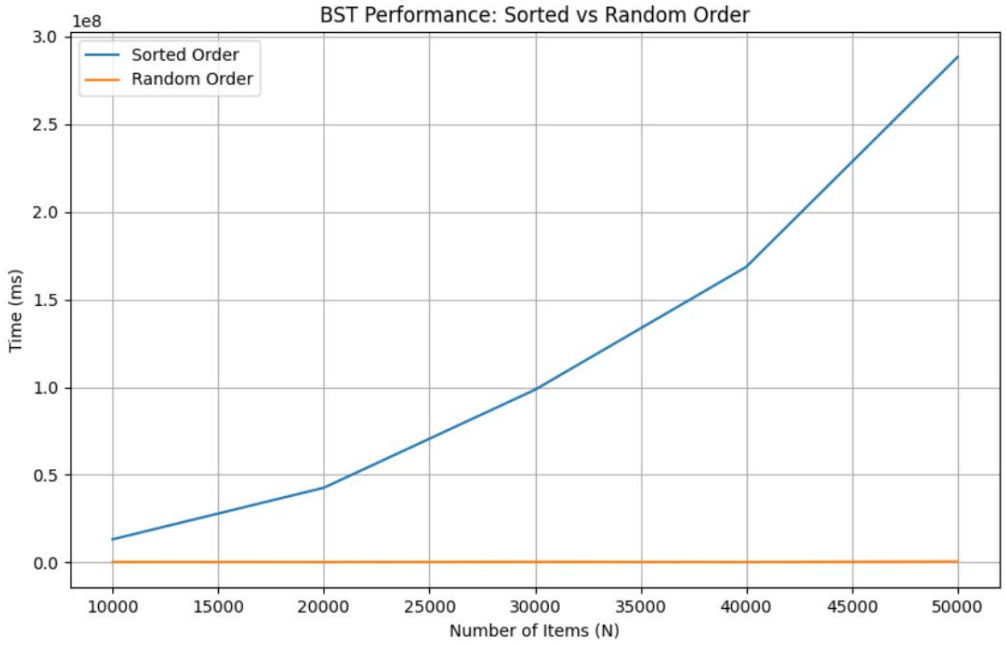
Add method of BST



Contain method of BST



Remove method of BST



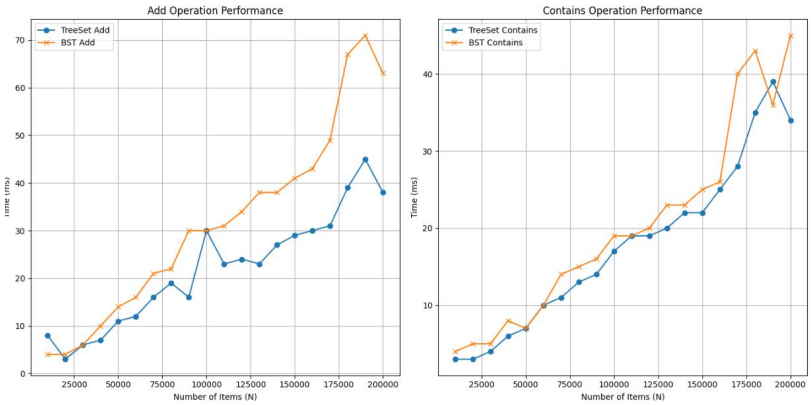
The blue line shows that performance deteriorates linearly as N increases, indicating that BST may degenerate into a linked list structure when sorting data is inserted. In this case, the height of the BST is proportional to the number of nodes, resulting in a time complexity of the operation approaching O(n).

The ideal BST should have a logarithmic height, and if this is not the case, it may be because the insertion algorithm did not balance the tree properly, or because the data is pre-sorted.

The random order is almost a straight line on the graph, which means that the performance of random inserts hardly changes with increasing N, which is unlikely in practice. This may indicate:

- The measured time granularity may not be fine enough to capture actual performance changes.
- The Y-axis range of the plot may be too wide, making logarithmic performance growth look like a straight line on the chart.
- Possible JVM optimizations, such as just-in-time compilation (JIT), may improve performance after executing the same code path multiple times.

BST & TreeSet



1. Add Operation Performance:

For the add operation, both TreeSet and BST show performance curves that increase with the number of elements. Initially, both have similar performance, but as the number of elements grows, the BST shows a significant decline in performance. This suggests that the BST may not maintain good balance with larger datasets, while the TreeSet, with its self-balancing properties, manages more data more effectively.

2. Contains Operation Performance:

For the contains operation, the performance curves for both increase with the number of items, but the TreeSet generally outperforms the BST. This underscores the importance of self-balancing mechanisms in maintaining operational efficiency, especially with larger data volumes.

3. **Reasons for the Differences:**

TreeSet is based on a red-black tree, which is a type of self-balancing binary search tree. It maintains balance through rotations and re-coloring operations, regardless of the order of insertion, which helps to ensure that operations have a time complexity of O(log n).
A regular BST does not have a built-in balancing mechanism, so its performance can be impacted by imbalances. If data inserted consecutively is ordered or near-ordered, it may degenerate into a long chain, causing the time complexity of operations to degrade to O(n).
In conclusion, the self-balancing feature of TreeSet provides a performance advantage in both add and contains operations, particularly when dealing with large datasets. While BST can maintain comparable performance with smaller datasets or less frequent operations, the lack of self-balancing can lead to significantly poorer performance as data volume increases. Choosing TreeSet or another self-balancing binary search tree might be more appropriate if you have specific performance targets or application contexts. However, a regular BST might still be a viable choice for smaller datasets or less frequent operations.