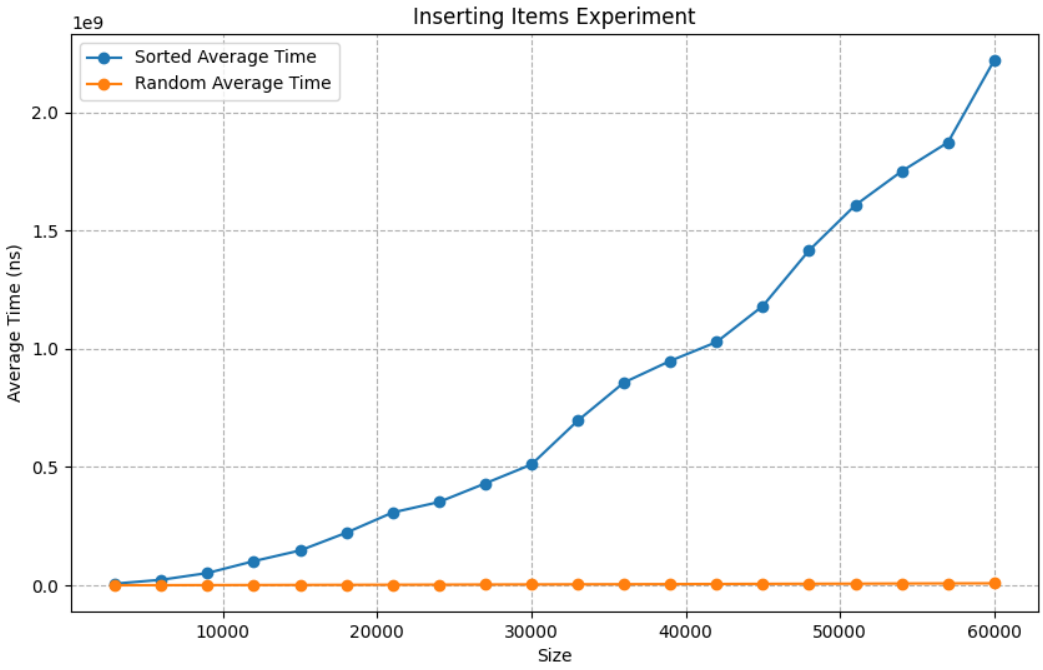
Assignment 6 Analysis
Ray Ding

1. **Explain how the order that items are inserted into a BST affects the construction of the tree, and how this construction affects the running time of subsequent calls to the add, contains, and remove methods.**

The order in which items are inserted into a BST significantly affects its construction, which in turn affects the performance of the add, contains and remove methods.

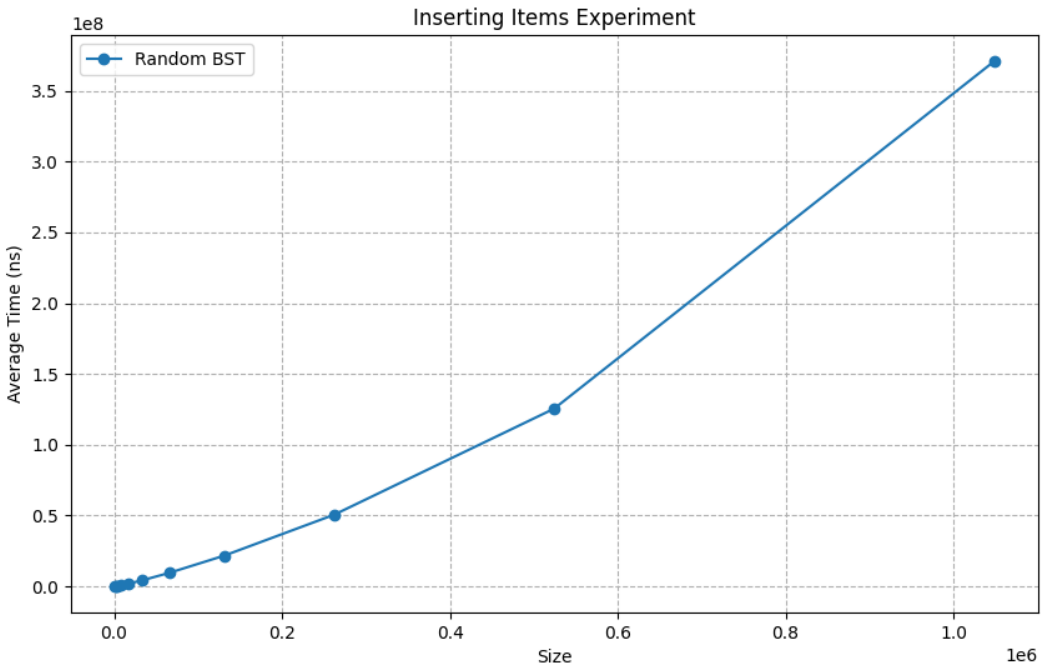
The following figure shows the running time of the call for contains for each item in the sorted and randomly ordered BST.



Due to the stack overflow issue for the testing of sorted BST, I tested the running time for N in the range 3,000 to 60,000. As shown by the blue curve, the time complex for the contains method for each item in the sorted BST is $O(N^2)$. This is because that the sorted BST becomes skewed. It resembles a linked list rather than a tree, with each node having only one child. In such cases, the height of the tree becomes N, leading to inefficient operations. For the contains method, it needs to traverse each node like in a linear search of a linked list. The time complexity is O(N). With the for loop iterate N times for all the items, the time complexity for

the test becomes $O(N^2)$. The orange curve shows the running time for randomly ordered BST, which is much faster compared to that of the sorted BST.

The next figure only displays the running time of the call for contains for each item in the randomly ordered BST. The data N is in the range from 2^{10} to 2^{20}. The curve clearly shows the O(NlogN) time complexity. For a random ordered BST, nodes are distributed evenly on both sides of the tree. In a balanced BST, the height of the tree is approximately logN. With the for loop iterates N times for all the items, the time complexity becomes O(NlogN).



2. BST experiment suggestion — self-balancing BST vs. regular BST

The figure below demonstrates the experiment result for TreeSet (self-balancing BST) and regular BST (randomly ordered BST). N is in the range [10000, 200000] stepping by 10000. Both curves show the O(NlogN) time complexity. The behavior of these two is almost identical for small datasets, while the TreeSet is faster for the contains method for each item at large dataset. The reason of the difference is due to the nature of the TreeSet and randomly ordered BST. A randomly ordered BST tends to be more balanced than one filled with sorted data, while it won't be as optimally balanced as a TreeSet. At large dataset, the difference is clearer.

3. **Discuss whether a BST is a good data structure for representing a dictionary. If you think that it is, explain why. If you think that it is not, discuss other data structure(s) that you think would be better. (Keep in mind that for a typical dictionary, insertions and deletions of words are infrequent compared to word searches.) Many dictionaries are in alphabetical order. What problem will it create for a dictionary BST if it is constructed by inserting words in alphabetical order? What can you do to fix the problem?**

BST is not a good data structure for representing a dictionary. Since most dictionaries are in alphabetical order, the sorted BST degrade into a structure akin to a linked list, leading to O(N) time complexity for searches, insertions and deletions. For most typical dictionary applications where fast lookup is a priority and order is not critical, hash tables or tries are generally more efficient and practical. Hash tables offer O(1) average time complexity for search, insertion and deletion operation.