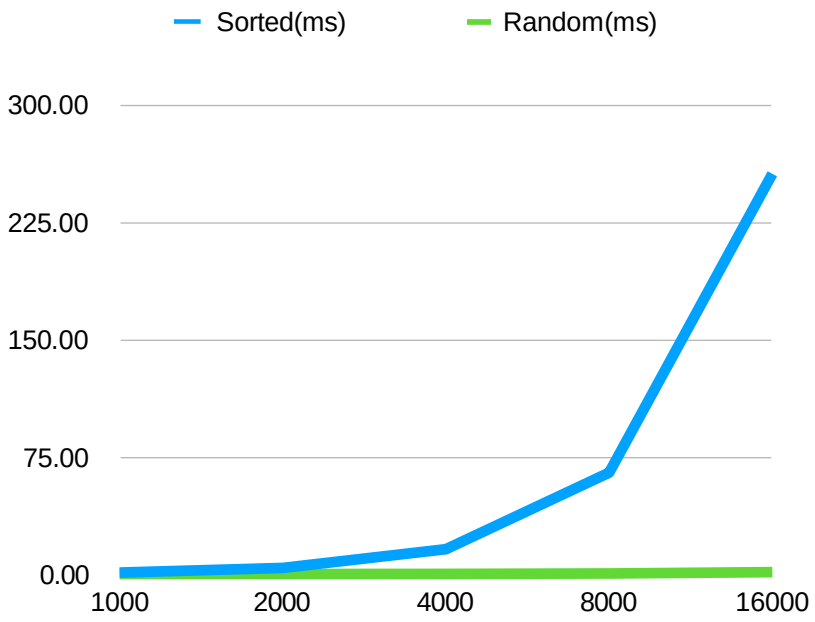
1. Explain how the order that items are inserted into a BST affects the construction of the tree, and how this construction affects the running time of subsequent calls to the add, contains, and remove methods.

The order in which items are inserted into a Binary Search Tree (BST) affects its shape. If items are inserted in a random order, the tree is likely balanced, leading to faster operations with a runtime of O(log n). However, if items are inserted in sorted order, the tree becomes unbalanced, making operations slower with a runtime of O(n).A balanced tree improves the efficiency of add, contains, and remove, while an unbalanced tree significantly slows them down.

2. Design and conduct an experiment to illustrate the effect of building an N- item BST by inserting the N items in sorted order versus inserting the N items in a random order. Carefully describe your experiment, so that anyone reading this document could replicate your results.

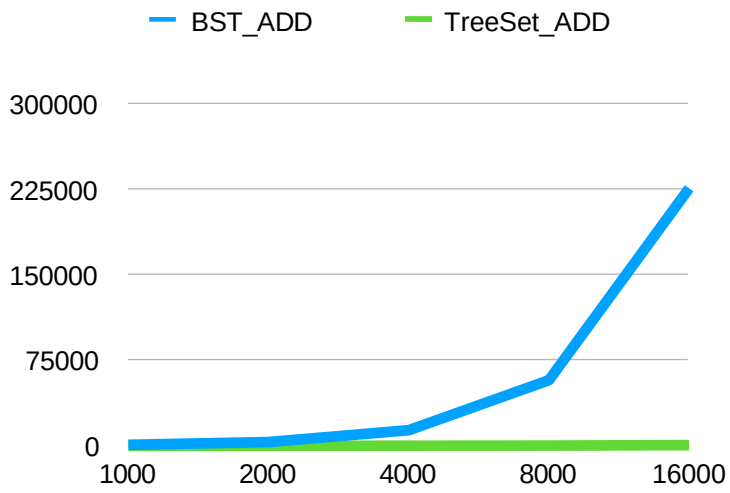
To compare BST construction performance, I tested inserting integers in both sorted order and random order. I used data sizes ranging from 1,000 to 16,000 elements, running 100 trials for each size to ensure accuracy. The results showed that random insertion was much faster than sorted insertion. For example, with 16,000 elements, sorted insertion took about 205ms, while random insertion only took 10ms. This demonstrates that random insertion helps keep the tree balanced, which is crucial for efficient BST construction.

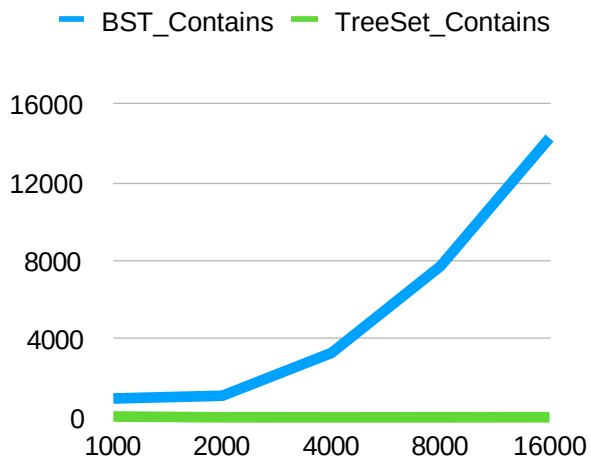
3. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as your interpretation of the plots, are critical.



4. Design and conduct an experiment to illustrate the differing performance in a BST with a balance requirement and a BST that is allowed to be unbalanced. Use Java's TreeSet as an example of the former and your BinarySearchTree as an example of the latter. Carefully describe your experiment, so that anyone reading this document could replicate your results.
5. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as your interpretation of the plots, are critical.

   I conducted an experiment testing the add and contains operations. I used sorted numbers to create a worst-case scenario for the unbalanced BST, with data sizes ranging from 1,000 to 32,000 elements. I performed 1,000 random searches for the contains operation and repeated the tests 100 times for accuracy. The results showed that the TreeSet, which maintains balance, performed better, with faster insertion and search times. At 32,000 elements, the BST took about 800ms to insert, while the TreeSet only took 22ms.





6. 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.)

   Yes, a BST allows for fast searches because it uses binary search, which is efficient for finding words. By comparing the characters of the word, the tree can quickly determine if it should go left or right, making it easy to find a word in the dictionary.

7. 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?

   Inserting words in alphabetical order into a BST will cause the tree to become highly unbalanced, essentially turning it into a linked list. We can use a balanced tree structure, like an AVL tree or a Red-Black tree, which automatically rebalances itself during insertions and deletions.