Assignment06 BST Analysis Document

Corinne Jones

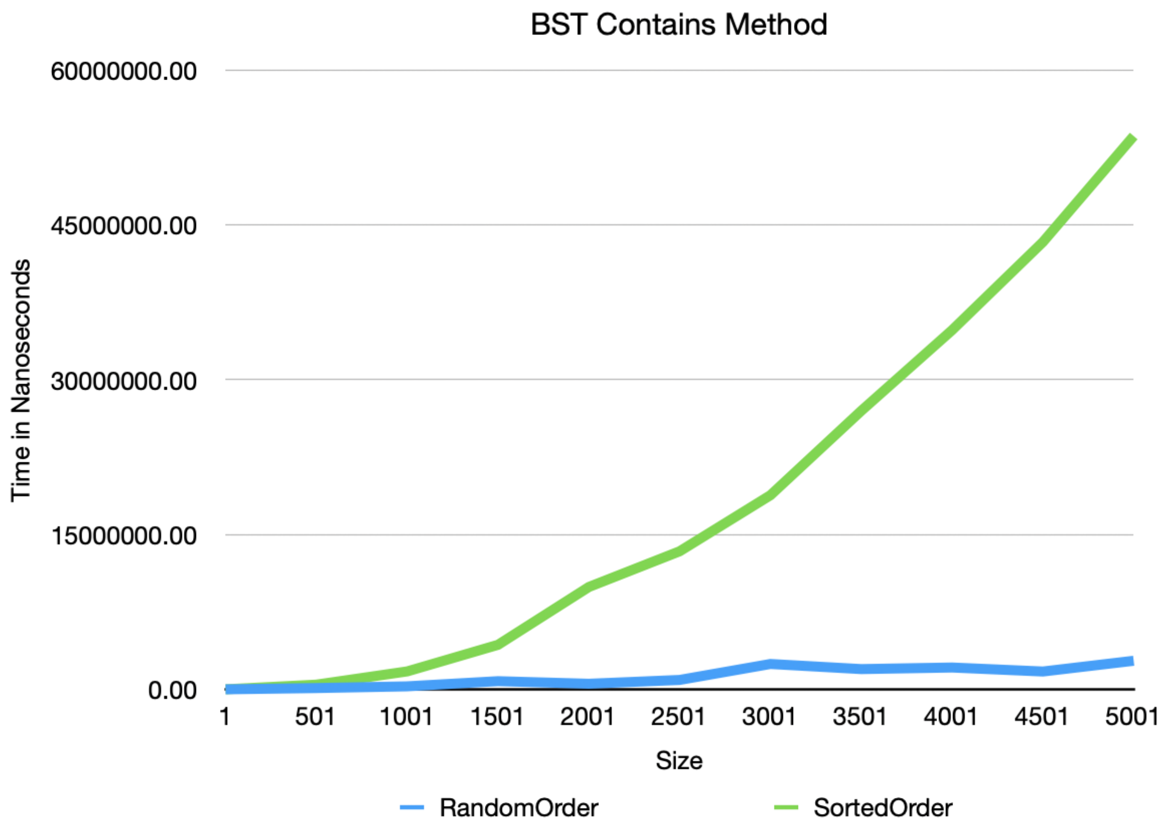
1. Explain how the order that items are inserted into a BST affects the construction of the tree, and how this construction affects the running time of subsequent calls to the add, contains, and remove methods.

The order items are inserted into a BST affects the construction by either making it balanced or imbalanced. In a skewed BST (i.e., imbalanced), the worst case scenario would involve every node being added to the right most node, or every node being added to the left most node, almost making a single linked list. (The runtime of this is O(N), which would happen everytime a call is made to the add, contain, and remove methods. When the BST is balanced, the nodes would be added in a way that makes a BST more full. When a BST is balanced, the runtime is O (lg N) for the add, contain, and remove methods. This is a significantly better runtime.

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.

The experiment I designed tested the BST contains() method on both a sorted and a random insertion of integers. The setup involved setting an int ITER_COUNT equal to 100, a variable long start time equal to the system.nanoTime(), and writing the results to a file. A for loop was created, setting int size=1, size<=5001, and size+=500. This creates an array beginning at size 1, ending at size 5,001, and increasing in size 500 increments. This size was determined by the length of time it took the program to run. A variable long totalTime was set =0. An array was then created up to the current array size in sorted order. This array was added to the binary search tree (BST1). The timer was started, and contains() was called to BST1, iterating through every index at the current size. After it was done iterating, the time was stopped and recorded. The array in sorted order was then randomly shuffled, making an array with the same numbers but in random order. This was then added to a second BST (BST2). The timer was reset to 0, and then the timer restarted, and contains() was called to BST2, iterating through every index. After it was done iterating, the time was stopped and recorded. Each time was recorded as the totalTime divided by the ITER_COUNT.

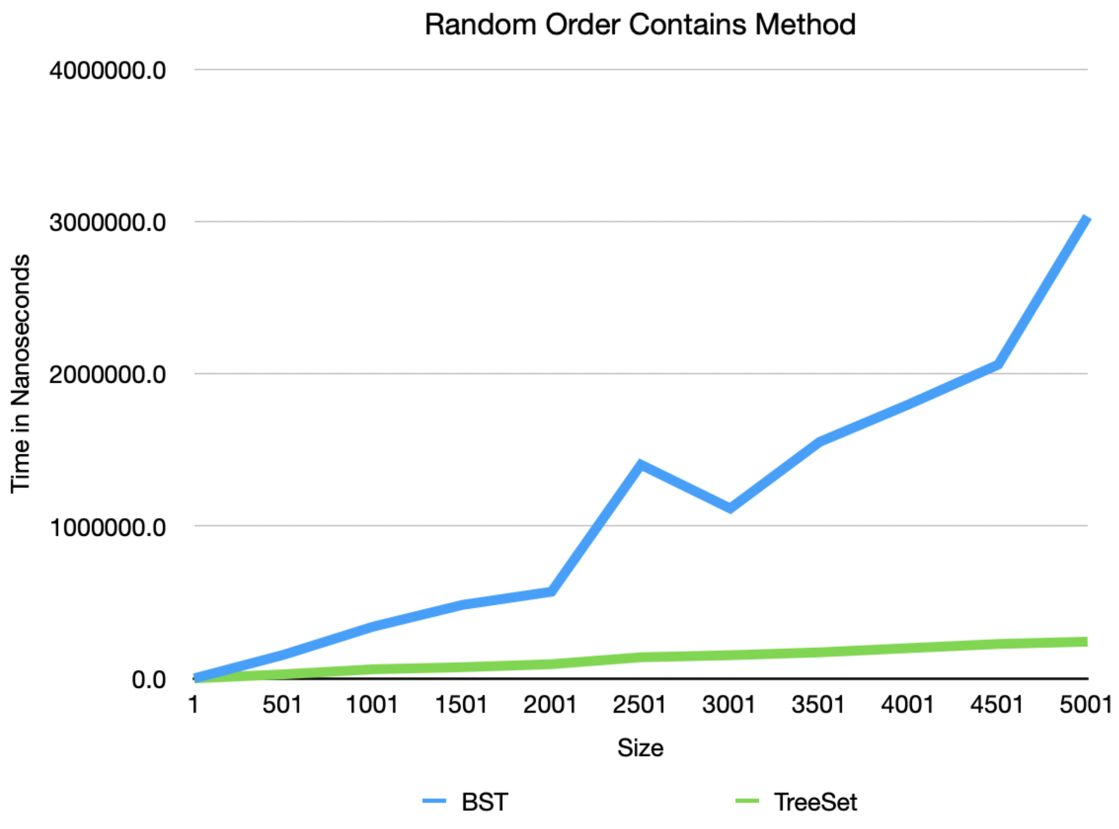
3. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as your interpretation of the plots, are critical.



This graph demonstrates that a sorted order results in a skewed tree which has the O(N) runtime. The random order allows the BST to be more balanced, resulting in an O(log N) runtime.

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. 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.

The experiment I designed tested the contains() method on a balanced (java TreeSet) and unbalanced (my implementation of binary search tree, BST) tree with a random assortment of integers. The setup involved setting an int ITER_COUNT equal to 100, a variable long start time equal to the system.nanoTime(), and writing the results to a file. A for loop was created, setting int size=1, size<=5001, and size+=500. This creates an array beginning at size 1, ending at size 5,001, and increasing in size 500 increments. A variable long totalTime was set =0. An array was then created up to the current array size in sorted order. The contents of this array were then shuffled, putting them in random order. This array was added to the binary search tree (BST). The timer was started, and contains() was called to BST, iterating through every index at the current size. After it was done iterating, the time was stopped and recorded. This same array, in random order, was then added to a java TreeSet. The timer was reset to 0, and then the timer restarted, and contains() was called to TreeSet, iterating through every index. After it was done iterating, the time was stopped and recorded. Each time was recorded as the totalTime divided by the ITER_COUNT.



This graph demonstrates that a balanced tree has a faster runtime than an unbalanced tree. The java tree set implementation is so much quicker than the BST due to balancing the items as they are added. This significantly decreases the height, allowing the method call to contains() to be much shorter than in a skewed tree. Although there is a random insertion of numbers to the BST, allowing it to be more balanced than when a sorted array list is inserted into it, it is not balanced, making the runtime longer.

5. 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.)

A BST is not a good data structure for representing dictionaries since they are typically in sorted order. The experiments I previously designed suggested that a BST does not work well with a sorted list of items since the runtime will be O(N). If you could balance the items when added, it would create O(log N) runtime. Another data structure would be a balanced tree, such as the java tree set.

6. 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 a dictionary in alphabetical order into a BST would essentially create a single linked list, since every “word” in the dictionary would be added to the right most node. In order to fix the problem, you could choose a new ADT such as a balanced tree. A balanced tree has a guaranteed runtime of O(log N), even in the worst case scenarios.