Haoran Chen

U1060286

Assignment 8 Analysis

1. **Who is your programming partner? Which of you submitted the source code of your program?**

   My partner is Eduardo Ortiz. I will submit the code.

2. **Evaluate your programming partner. Do you plan to work with this person again?**

   He's a very clever guy and he could focus on what we are doing. He is very good at testing and looking for bugs. I'd like to work with him again.

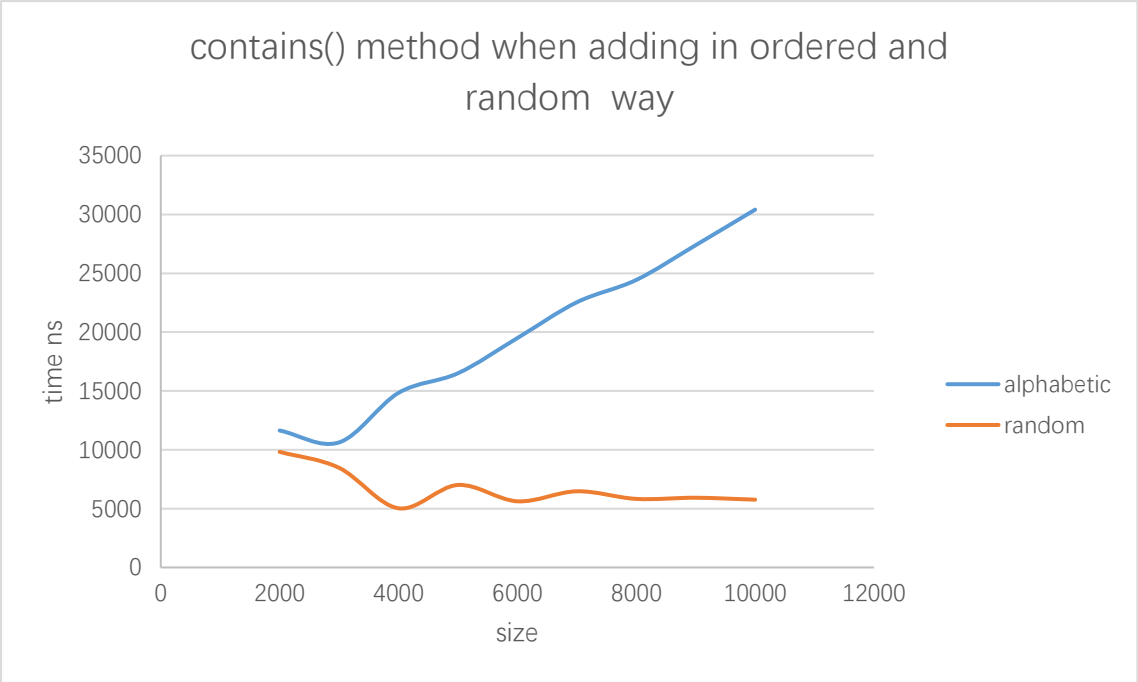
3. Design and conduct an experiment to illustrate the effect of building an N-item BST by inserting the N items in sorted order versus inserting the N items in a random order. Carefully describe your experiment, so that anyone reading this document could replicate your results. Submit any code required to conduct your experiment with the rest of your program and make sure that the code is well-commented. 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 is critical. One suggestion for your experiments is:

- Add N items to a BST in sorted order, then record the time required to invoke the contains method for each item in the BST.

- Add the same N items to a new BST in a random order, then record the time required to invoke the contains method for each item in the new BST. (Due to the

randomness of this step, you may want to perform it several times and record the average running time required.)

- Let one line of the plot be the running times found in #1 for each N in the range [1000, 10000] stepping by 100. (Feel free to change the range, as needed, to complement your machine.) Let the other line of the plot be the running times found in #2 for each N in the same range.



I generated two separate arraylists for two situations, one is sorted order and one is random. For the random list, I generated a list of number from 1 to the size and then shuffled it. In the test, I called addAll() method for an empty tree and time the contains method to the largest number in the tree, which is size. Then I just did the timing test for different sizes. On the graph, the blue line shows that when the list is added in sorted order, as it will produce a heavy right tree as everything added is the right child of the parent, the time complexity of searching an element is O(N), and the line shows my result. For the random added tree, the line does not show what I expected. When I

opened the .dot file for 25 elements, I found that after I shuffled the list I needed, as it is really random, the height of the largest number could be really small, even 1. Thus, the timing result couldn't reach O(logN), which is the Big(O) for contains() method of balanced tree. The shuffled order cannot produce a balanced tree because it doesn't grantee the first element to be added to be the middle of the sorted list.

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 (http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) as an example of the former and your BinarySearchTree as an example of the latter. Java's TreeSet is an implementation of a BST which automatically re-balances itself when necessary. Your BinarSearchTree class is not required to do this. Carefully describe your experiment, so that anyone reading this document could replicate your results.Â Submit any code required to conduct your experiment with the rest of your program and make sure that the code is well-commented. 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 is critical. â ¨One suggestion for your experiments is:

- Add N items to a TreeSet in a random order and record the time required to do this.

- Record the time required to invoke the contains method for each item in the TreeSet.

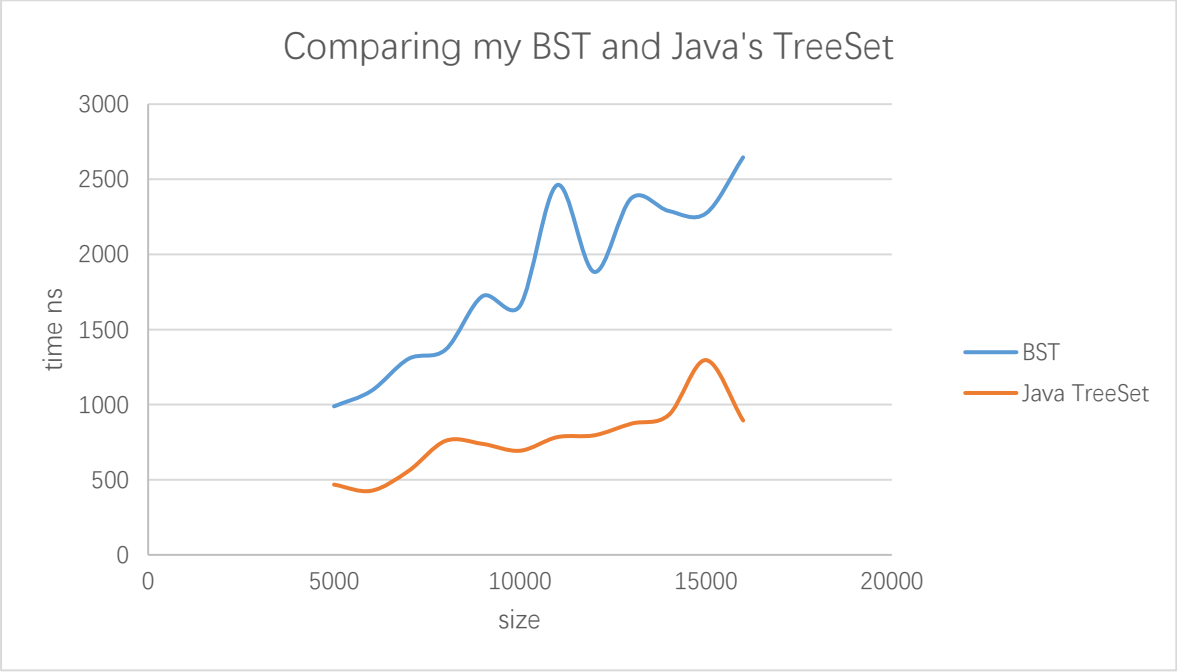
- Add the same N items (in the same random order) as in #1 to a BinarySearchTree

and record the time required to do this.

- Record the time required to invoke the contains method for each item in the BinarySearchTree.

- Let one line of the plot be the running times found in #1 for each N in the range [1000, 10000] stepping by 100. (Feel free to change the range, as needed, to complement your machine.) Let the other line of the plot be the running times found in the #3 for each N in the same range as above.

- Let one line of a new plot be the running times found in #2 for each N in the same range as above. Let the other line of plot be the running times found in #4 for each N in the same range. (You can combine the plots in the last two steps, if the y axes are similar.)



For the second experiment, I used the same idea of timing random order in #3. Besides doing addAll() and contains() to my BST, I called the same methods for a TreeSet as Java's TreeSet also has these two methods. To make sure I use the same random list for

same size, I did the timing test together for both trees. In this case, the timing results are a little bit larger than doing one of them.

Again, for my BST, it will not reach O(logN) as I expect because the tree will seldom be balanced. The range of the line plotted should be lower than O(N), but higher than O(logN). The line is not linear because sometimes the height of largest number is very small but sometimes it is very big, Then the time cost to search for it is varied by the root node picked by shuffling the list. For Java's TreeSet, as it is balanced, I can fully expect my result to be O(logN). On my graph, the first several points are kind of strange because of the cpu and other noise produced by my computer. The middle part is good for O(logN), and when the size reaches 13,000 the time taken is significantly increased, maybe because it reaches some threshold and it need to random again. Therefore for my graph, although it is not an O(logN) as we learned before, the time taken still increased in small amounts.

5. 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? Explain what you could do to fix the problem.

When we add all the words in alphabetical order, it will generate a heavy right tree, as every time we want to add something, it will be added as right child, no left child will be appearing in the tree. In this case when we want to search for a word, the Big(O) of it should be O(N), which is not efficient. The way to solve it is to use a balanced binary search tree instead of an unbalanced tree. In this case, the Big(O) for searching should be O(logN). However, if I don't have a chance to get a balanced tree, then I will do

something like “binary search” for the whole document, which means I will add the middle one of the dictionary first, then the middle of first half, middle of first quarter… In this case, the tree is kind of balanced as well.

6. How many hours did you spend on this assignment?

About 5 hours.