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

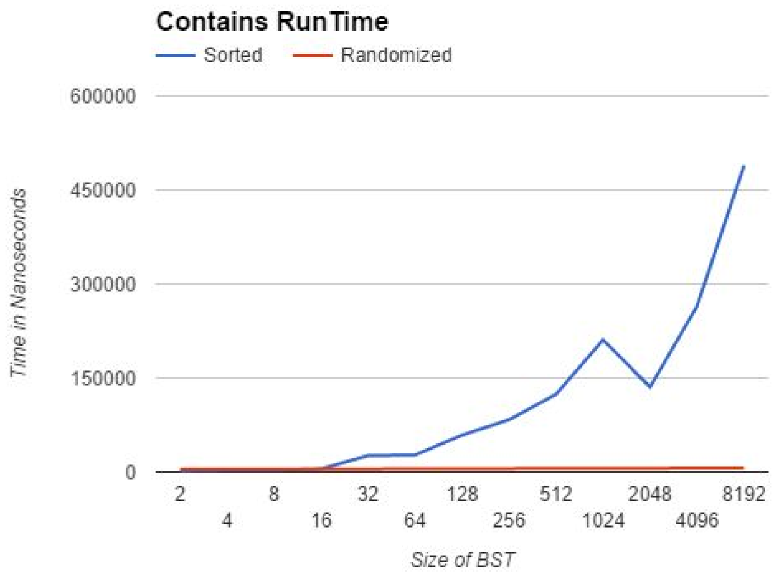
Mike (Miao) Yu was my programming partner. I will be submitting the source code of our program.

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

He was great and totally knew what he was doing. He helped me find a handful of mistakes that I couldn’t find no matter what I did, and he also helped suggest new ideas when I had no idea what I was doing. Yes, I would 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:

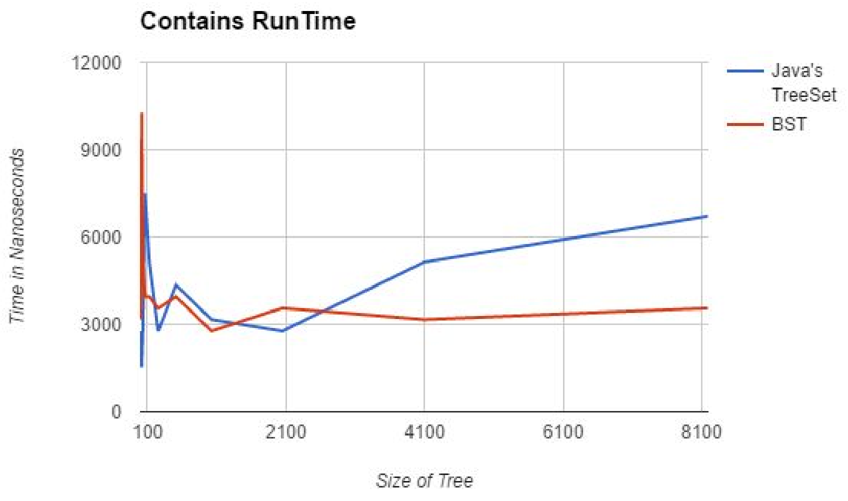
1. Add N items to a BST in sorted order, then record the time required to invoke the contains method for each item in the BST.
2. Add the same N items to a new BST in a random order, then record the time required to invoke the contains method for each item in the new BST. (Due to the randomness of this step, you may want to perform it several times and record the average running time required.)
3. 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.

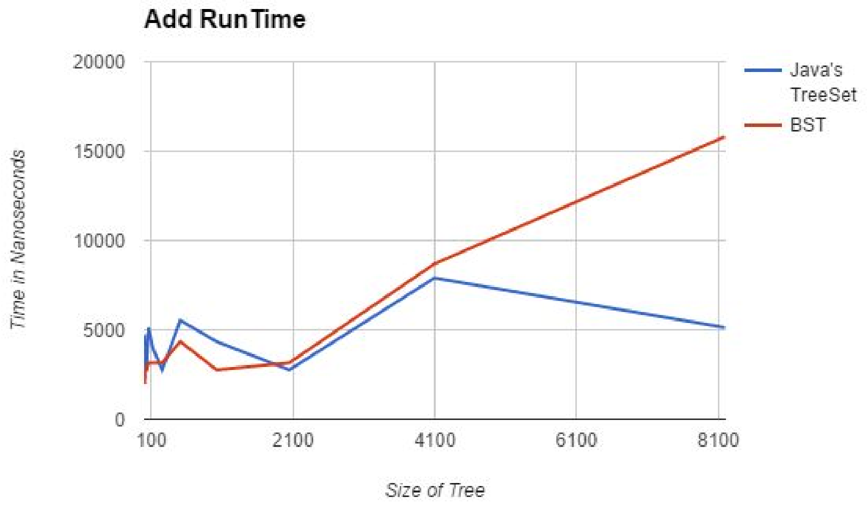


For the run time of the contains method, the code was significantly more efficient if the inputs were randomized rather than sorted. This was because, by the nature of binary search trees, the time cut by dividing a set into multiple sets of “halves” is completely negated if none of the values are split up, thus leading to an efficiency of O(N), which is roughly reflected in the blue line of the graph (barring some outliers). Meanwhile, when the line is randomized, the efficiency of the contains method becomes much closer to O(1) because the method will be taking full advantage of the binary search.

4A. 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.

1. Add N items to a TreeSet in a random order and record the time required to do this.
2. Record the time required to invoke the contains method for each item in the TreeSet.
3. Add the same N items (in the same random order) as in #1 to a Binary Search Tree and record the time required to do this.
4. Record the time required to invoke the contains method for each item in the BinarySearchTree.
5. Let one line of the plot be the running times found in #1 for each N in the range [0, 10000], incrementing by 100 each time. Let the other line of the plot be the running times found in the #3 for each N in the same range as above.
6. 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.)





4B. Analysis Portion

For the run time of the contains method, the code was surprisingly slightly more efficient using my own BST class. I expected Java's Tree method to be more efficient than my BST, because it takes what we're trying to achieve with the randomized order inputs (taking full advantage of the benefits of splitting the datasets up into traversable branches) and amplifies it by using some algorithm to continuously further sort the values. I am not sure why this wasn't the case based on my data. I wonder if they would follow my prediction more closely had I averaged out the values of numerous trials. In any case, because both took in the inputs in a randomized order, both were fairly efficient, showing time efficiencies of somewhere between O(log N) and (O1).

For the run time of the add method, the code was fortunately a bit more efficient using Java's treeset class compared to using my BST class. This makes sense because Java's treeset class continually sorts the values inputted to it to make sure the tree is balanced, which is the goal randomizing the input values is meant to emulate. The efficiency of the add method for my BST class is close to O(N), while the efficiency of the add method for Java's treeset class is a bit closer to O(log N), which is preferable. In either case, as the inputs were randomized, the add methods of both data structures were fairly efficient regardless of their class details.

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.

If we insert words in alphabetical order to a Binary Search Tree, this will cause problems in that it will be a right-heavy tree. In fact, it would form a linear "tree" with items only to the right. This would cause a runtime of O(N) instead of the near-O(logN), which is what we want. To fix this problem, we need to take the median of the list of arrays, then keep inputting each median of both sides of the median found previously, taking the left side of the "median" if there is an even number of strings left.

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

We spent around fifteen hours on this assignment.