Chris Grayston u0906710

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

Q1) Stephen Hogan u0813633. Stephen submitted the code for us.

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

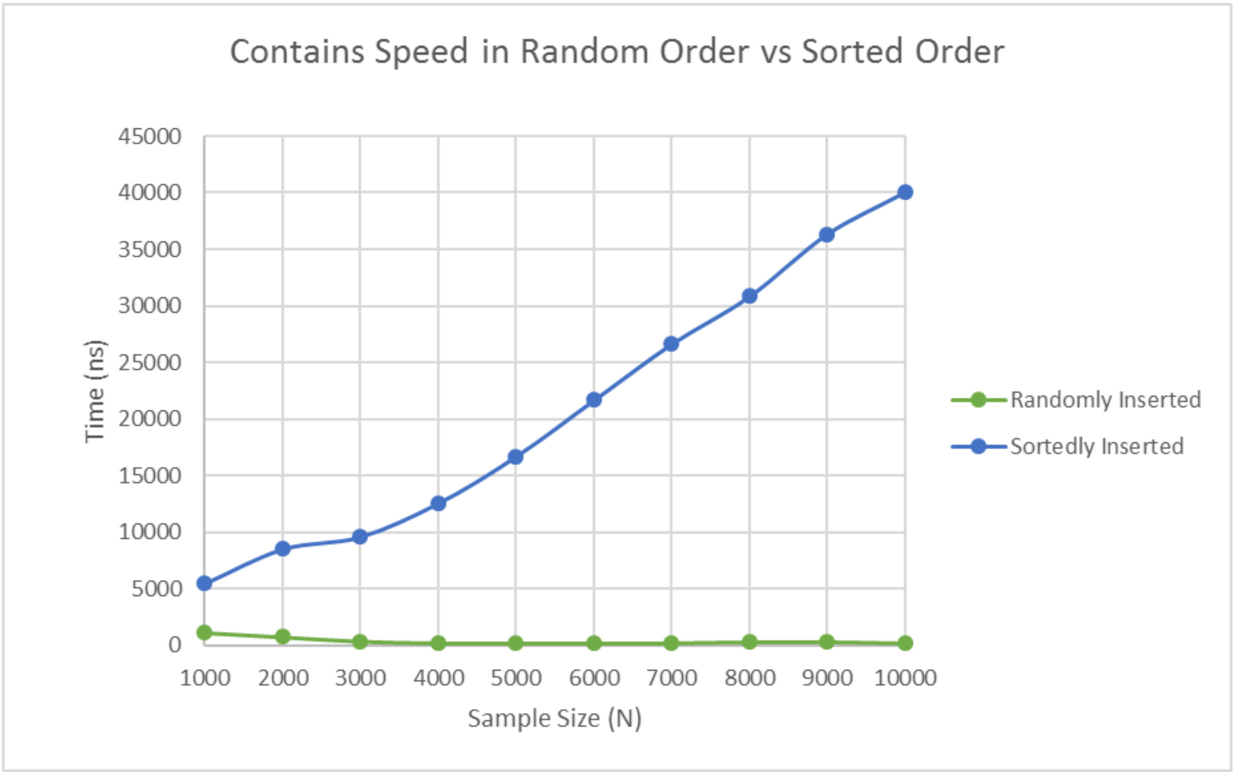
Q2) Stephen is a great partner as he texts me often to see where I am at my part of the coding so we can coordinate what we are going to do when we meet up later into the evenings. Yes, after I change programming partners next week I plan on working with Stephen again.

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.)

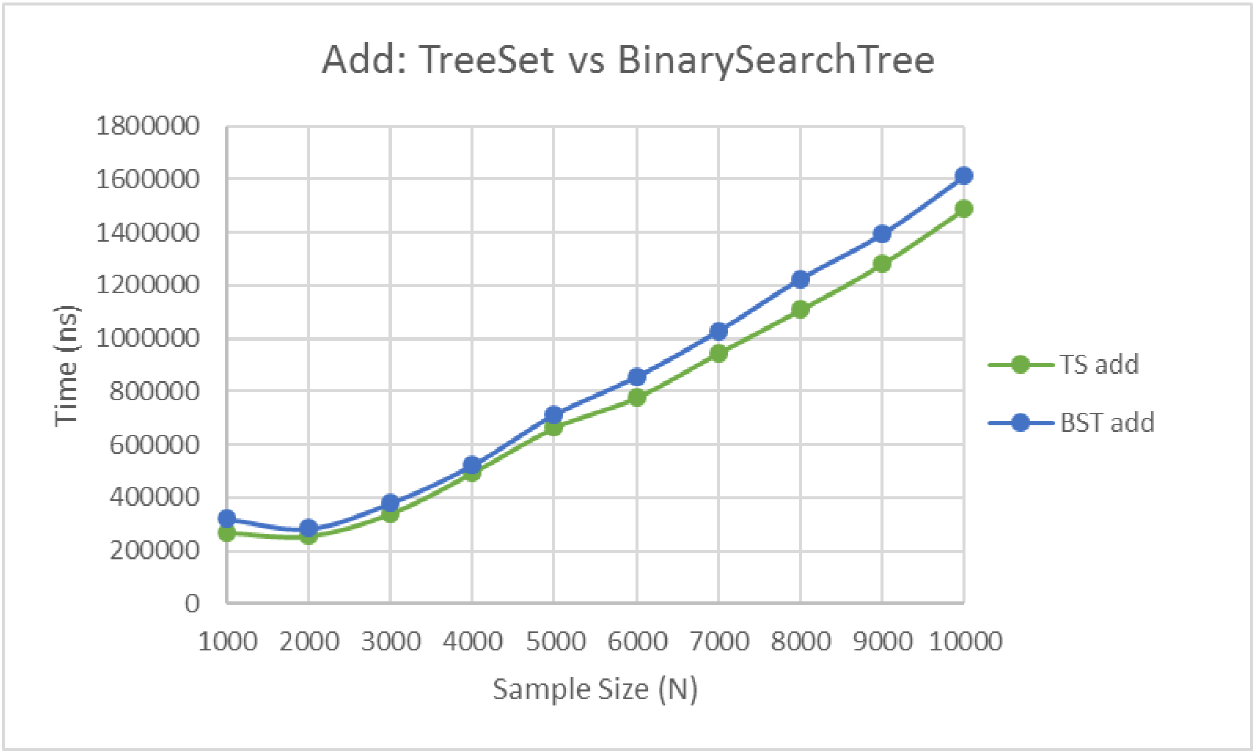
- Let one line of the plot be the running times found in #1 for each N in the range [1000, 10000] stepping by 100. (Feel free to change the range, as needed, to complement your machine.) Let the other line of the plot be the running times found in #2 for each N in the same range.



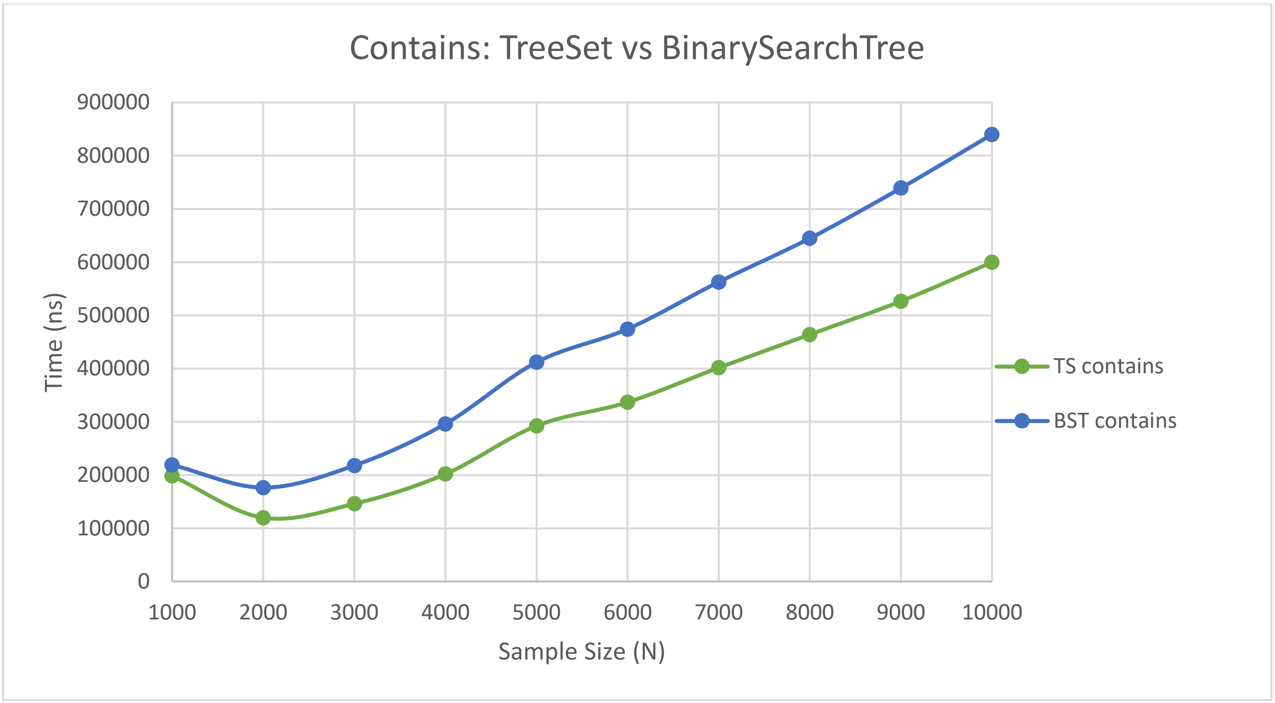
Q3 – Sorted Inserted (Blue Line)) By observing the graph above you can see that using contains on our sorted method seems to follow the path of Log(n). I think the reason the blue sorted line came out that was is I was running contains on the last item in the sample size. That was we had to go through all items the Binary Search Tree, this is what caused our line to resemble Big O(n).

Q3 – Randomly Inserted (Green Line)) As for the green line or our randomly inserted line, I think it came out looking almost constant time. The reason I believe the data creating the green line looked like this is because as I inserted numbers randomly into a Binary Tree of sizes 1,000 to 10,000 stepping by 1,000, sometimes we would get repeat numbers which won't be inserted into our Binary Tree. Therefor when running contains timing code on our randomly inserted Binary Tree there are most likely less numbers to have to sort through which results in a faster average runtime and a line that looks Big O (1).

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 for the Add and the Contains methods



Q4 – Add)
Java’s built in Tree Sort Add method (green line) ran very comparably to our Binary Search Tree Add method (blue line). I think the reason that the Tree Set ran just a little bit faster was because even though it does automatically rebalance itself when need be, this was written by a programmer who is a professional in this area therefore you can expect for right now that that their Tree Set might run a little bit faster than ours.



Q4 – Contains)
Java's built in Tree Sort Contains method (green line) ran noticeably quicker than our Binary Search Tree (blue line). The reason I think Java's Tree Set ran faster, although very similarly, was because of the fact that Tree Set automatically rebalances itself when it needs. Because of this you can bet that when contains is ran on a Tree Set that it will have an easier time locating if the item is contained in the set compared to our Binary Search Tree which could become unbalanced throughout all our adding.

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.

Q5)
Inserting words alphabetically into a BST will be a problem because it is slower compared to adding the words in randomly. Alphabetical search time Big O is something closer to O(n), while if it was randomly inserted the runtime would be closer to O (n log(n)). And for something as big as the dictionary that makes a big difference. So there for inserting words randomly into a BST will in the long run make spell checks and dictionary searches a lot faster.

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

Q6)
Around 11 hours.