When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 25% of your Assignment 8 grade. Ensure that your analysis document addresses the following.

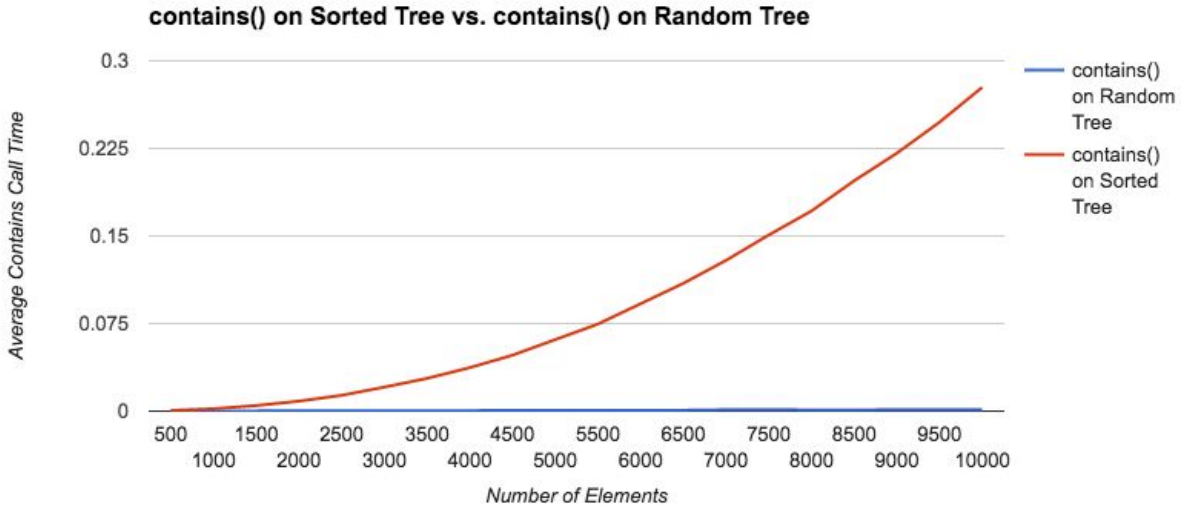
1. Who is your programming partner? Which of you submitted the source code of your program?
    Dallin Van Mondfrans, u0717113
    My programming partner Dallin submitted the code.

2. Evaluate your programming partner. Do you plan to work with this person again?
    He did good. Seems to know more then me. Would work again. I would say he is better at coding than me for sure. Also knew a bunch about timing.

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.



This experiment was run by creating two BinarySearchTrees. One tree was filled with each integer in order from 1 through N. The other tree was filled with each integer in a random order from 1 through N. Contains was called on each tree for each item 1 through N. The contains calls were averaged for each tree. Our expected result for the ordered tree was a predictable amount, because each pair will average to (½)N, which is O(N). Our expected result for the random tree is O(Log N). Actually more (log n)-ish, because we don't enforce balance. This did close to what we expected. Log N looks constant for these sizes, but the Sorted tree seems to tip above linear time. Probably due to some overhead in our methods.

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. 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:

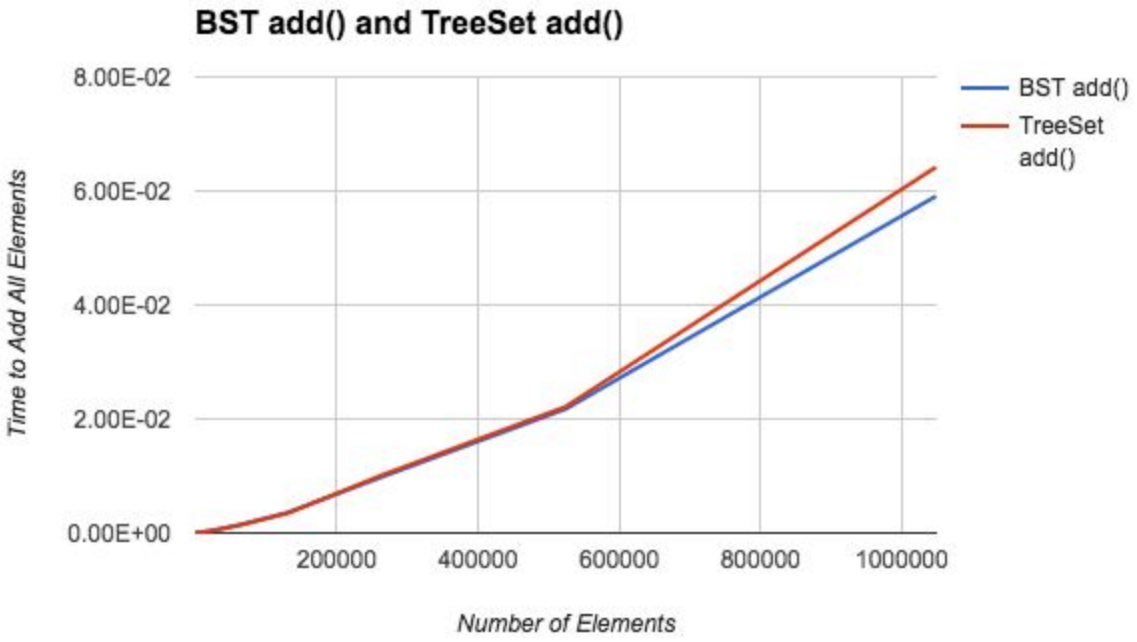
- Add N items to a TreeSet in a random order and record the time required to do this.

- 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.

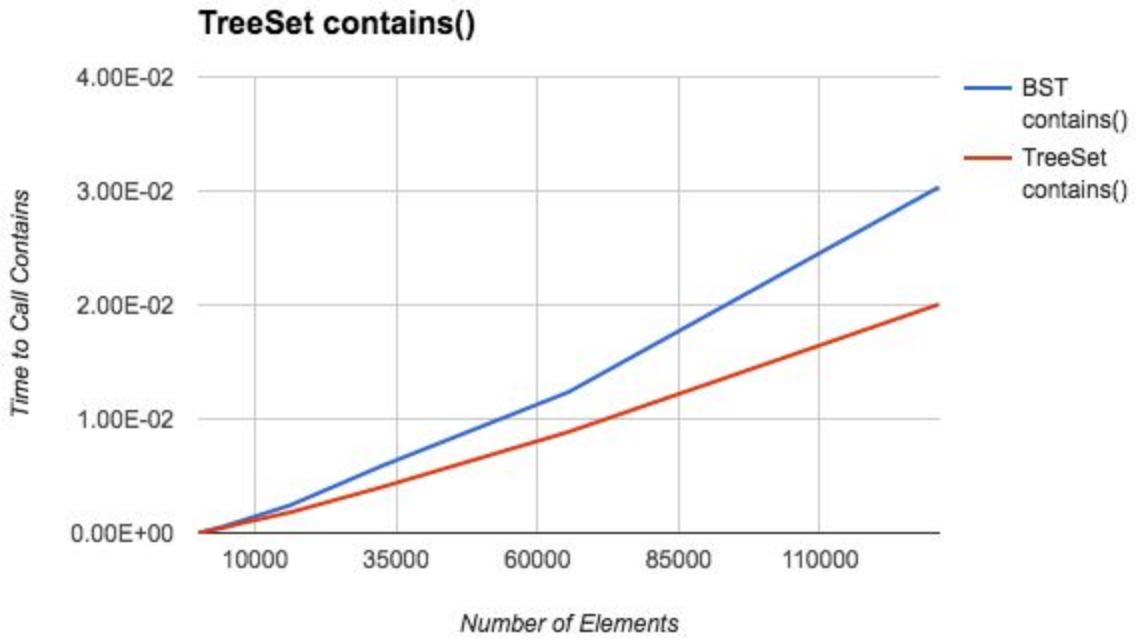
- Let one line of the plot be the running times found in #1 for each N in the range [1000, 10000] stepping by 100. (Feel free to change the range, as needed, to complement your machine.) Let the other line of the plot be the running times found in the #3 for each N in the same range as above.



This is a comparison of adding from 1 to N elements to our BinarySearchTree implementation compared to the Java TreeSet implementation. We just timed it for bigger and bigger amounts looking for variation. We expected ours to be slower, because we thought we would have more overhead since we are 2420 students, but it turns out balancing takes more effort than whatever inefficiency 2420 students add to BinarySearchTree.

- 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.)



This is a comparison of calling contains from 1 to N elements to our BinarySearchTree implementation compared to the Java TreeSet implementation. We just timed it for bigger and bigger amounts looking for variation. We expected ours to be slower, because Java was balancing their tree during the add step. We were not disappointed. Java is faster. By a noticeable and growing difference.

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.

Oh, the problem is if everything is in order it makes a BinarySingleLinkedList (yeah, you're welcome graders). And it turns out I can move the dictionary to an ArrayList and then use Collections.sort()! I learned this long after I was done randomizing short lists by hand. :-)

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

So many! I'm gonna say 25. I seriously think I could delete the whole project and reimplement it in 3 hours. I really hit a wall on this one that cost me a ton of time, but I learned a whole lot about Eclipse, BinarySearchTree, and debugging.

Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.
Upload your solution (.pdf only) here by 11:59pm on March 12.