Ashley Grevelink u0749357
Assignment 08 Analysis
October 26, 2016

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

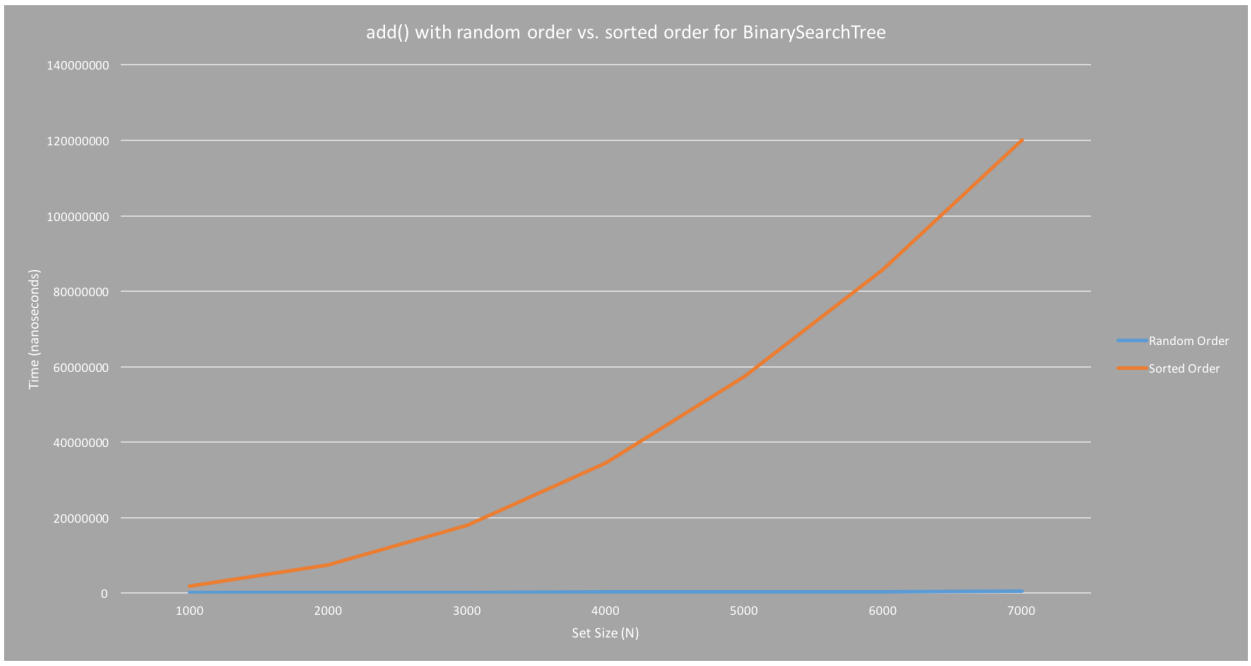
They call him Colton Haacke but his code is anything but hacky. He submitted.

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

It would be awesome to work together again. Colton is all-around great.

3. Experiment on add() for BST with sorted order vs. random order

We compared runtimes of invoking the contains() method on two different types of a BinaryTreeSet. First, we added N items to a BST in sorted order. This represents our sorted BST. Then we added N items to a different BST in random order. This represents our random BST. We separately timed how long it would take to execute calling contains() on each item of N of each BST. Our iteration count was 500 times to account for randomness. We plotted N from 1000 to 7000, stepping up by 1000 each time for a total of 7 data points per BST.



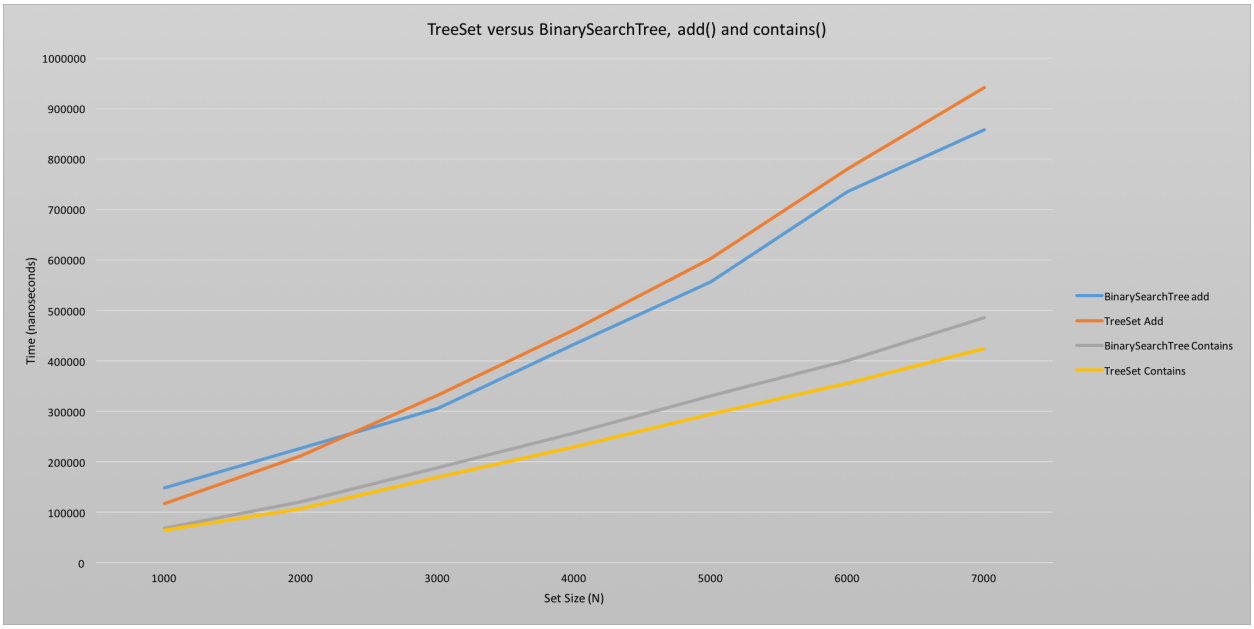
The sorted BinarySearchTree took orders of magnitude more time to execute than the random BST. This is because our BST was not required to be balanced, and adding to it in sorted order caused a heavily unbalanced tree. In this case, it was right-heavy. The random BST had a greater chance of being balanced because due to pseudorandom shuffling, the median word had a greater chance of being added first, as the root node.

4. Experiment on addAll() and contains(), TreeSet versus BinarySearchTree

We compared runtimes of invoking the contains() method on different types of trees, including our own BinarySearchTree and Java's TreeSet. We gathered four plots of data: two invoking add() and two invoking contains() on our two data structure types.

First, we added N items to a BinarySearchTree in random order. This represents our BST. Then we added the same items in the same order thanks to pseudorandom number generators to a TreeSet. This represents our random BST. We separately timed how long it would take to execute calling addAll() on the BST and the TreeSet. Our iteration count was 500 times to account for randomness. We plotted N from 1000 to 7000, stepping up by 1000 each time for a total of 7 data points per data set.

Then, we used our random BST and random TreeSet to call contains(). Because we had to call contains() in a for-loop, we separately timed an empty for-loop and subtracted the time for this process from the total time of calling contains() over 500 iterations. Once again, we plotted N from 1000 to 7000, stepping up by 1000 each time for a total of 7 data points per data set.



When analyzing the add() methods, BinarySearchTree performs better. This is because TreeSet takes extra time and processing to balance its tree as it adds. This often requires finding and swapping nodes, which results in a longer runtime. When adding to a BST, we must traverse the tree to add, but no swapping is required.

However, when it comes to contains(), TreeSet performs better. Because of the balanced nature of the TreeSet, calling contains() theoretically does not have to search as many nodes as an unbalanced tree of the same size. The unbalanced BST could have a much greater height than a TreeSet and require many more recursive operations to search for an item.

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 a BST is constructed by inserting words in alphabetical order, it will inevitably become right-heavy. To avoid this problem, we could build a balancing property into our add() method. If add() included balancing, this would require added complexity of searching for an swapping nodes. Some cheaper solutions might include choosing the first word that begins with 'm' or 'n' because these may be a relative median for the set. If the dictionary is incomplete or does not sequence from A to Z, however, it proves this solution is not very robust or generic. Perhaps the best would be choosing a random item to insert first. Adding a random item, or middle-of-three offers a better chance at a relatively median item becoming the root node.

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

5-6 hours