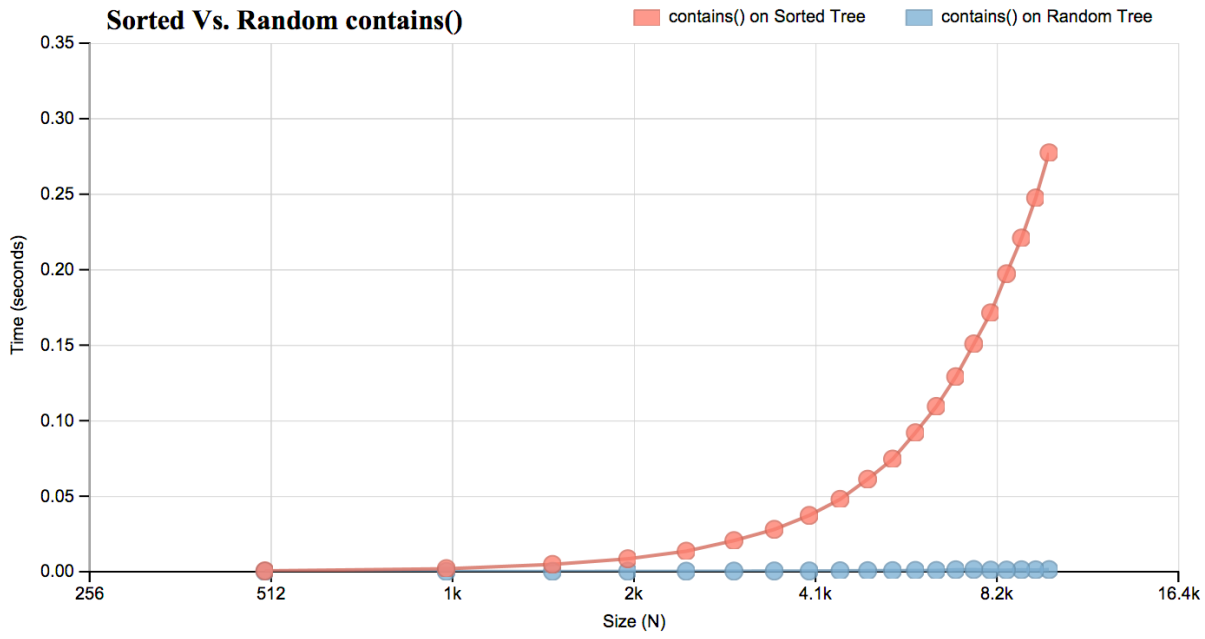
Dallin Van Mondfrans
u0717113
October 26, 2016

Assignment 8 Analysis

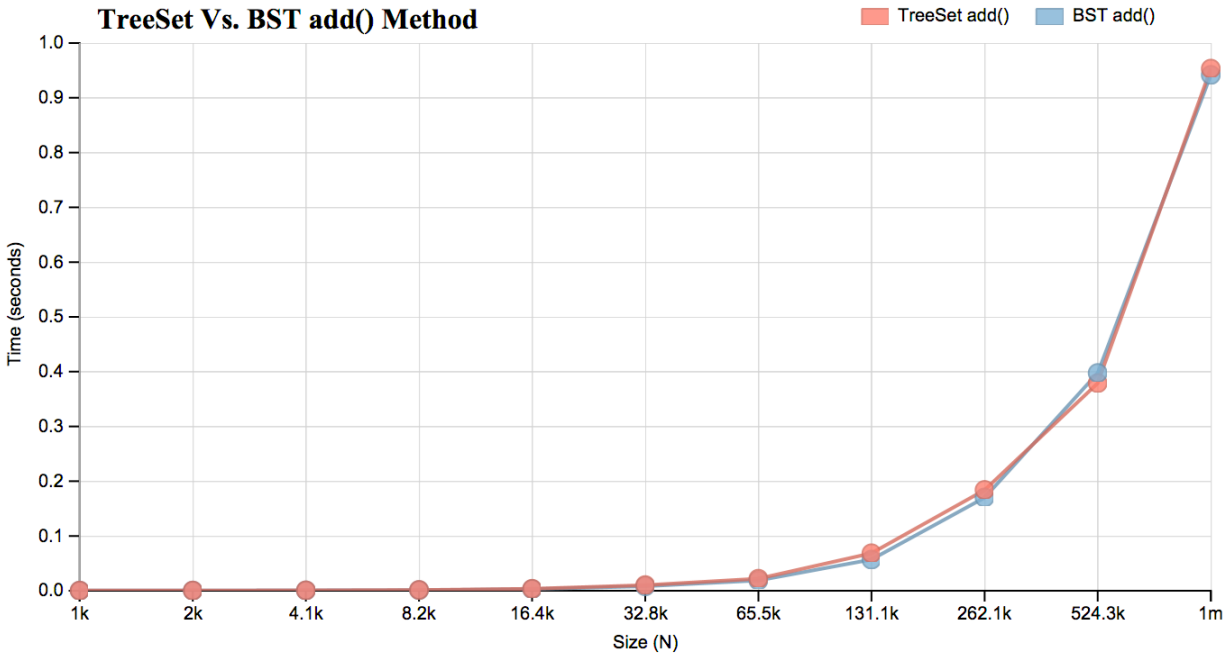
1. My programming partner for this assignment is Nicholas Kerr. I submitted the source code for our program.

2. Nicholas is a good partner. He was easy to work and coordinate with and contributed to the assignment. We both think similarly which I think helped us to communicate with each other more effectively. Yes, I plan on working on the next several paired programming assignment with Nicholas.

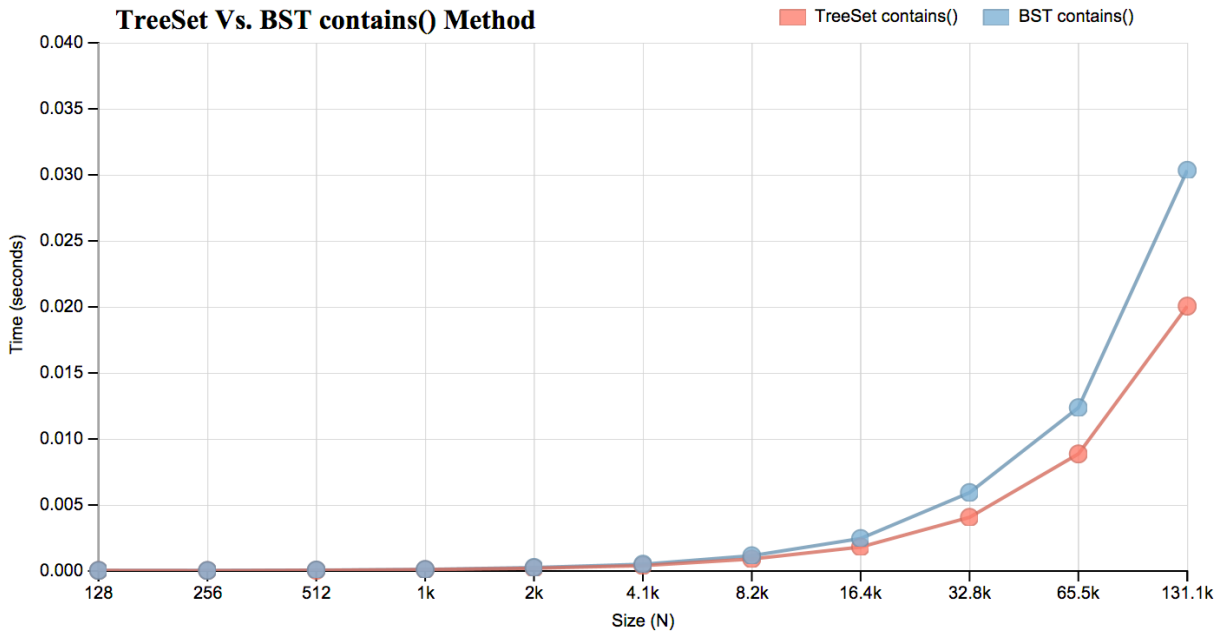
3. For our experiment in analyzing our BinarySearchTree data structure, we followed the suggested steps. First, we timed how long it took, on average, for the contains method to be run on our binary search tree when the data was added in sorted order. This was compared with the average time it took to run the contains method on a binary search tree whose data was added in random order. We did this for 20 sizes of data, starting from 500 and going to 10,000 in increments of 500. The graph below shows our results. We can see that when the data is added to the tree in sorted order, the time it takes to run the contains method increases dramatically. This is due to the fact that when the items are added in sorted order, they form, in essence, a linked list because each successive element is larger than the previous. Thus, the time it takes to search the data structure become linear, or O(N). In contrast, when the data is added in random order, the structure is roughly halved in each step of the search and we therefore see logarithmic behavior, or O(log N).



4. Similar to the previous experiment, we followed the suggested steps in order to compare Java's TreeSet data structure to our BinarySearchTree. To do this, we added N items to each of the trees and timed how long it took for different values of N. Unlike our binary search tree, TreeSet has a balancing requirement, meaning it restructures itself when needed so as to ensure that the tree remains mostly balanced. Thus, we expected to see our tree perform better than TreeSet. However, as can be seen in the graph below, we were surprised to find that the difference is negligible. I believe that there are two possible explanations for this unexpected result. First, our tree structure assuredly has some inefficiencies that may have kept it from performing optimally. Second, and likely more impactful, is that we added the items to the trees in random order. Trees generally stay fairly well balanced when items are added in random order, so it is likely that TreeSet didn't have to do much restructuring while adding the items. I suspect that our results would have conformed to expectations had we added the items in sorted



order. In addition to comparing the add method of the two tree structures, we also timed how long it took to run the contains method on the two structures on the same random number addition. Knowing that Java's TreeSet is guaranteed to be balanced, we expected it to out perform our binary search tree. The graph below validates this expectation, as we can see that when the size of our data, or N, becomes large, TreeSet begins to run the contains method faster than BinarySearchTree. Thus we see that using some kind of a balancing requirement for a tree structure enhances and optimizes performance.



5. The problem that adding words from a dictionary to a BST would create is that it would effectively create a linked list because each successive element added would be “larger” than the previous one. Thus, we would not achieve O(log N) behavior for the add, contains, and remove methods. One way to fix this issue would be to randomize the words contained in the dictionary and then add them to the BST in random order. Of course, using Java’s TreeSet would also fix this issue as it would restructure the tree so as to keep it balanced, thus achieving the expected logarithmic behavior.

6. I spent about 10 hours on this assignment.