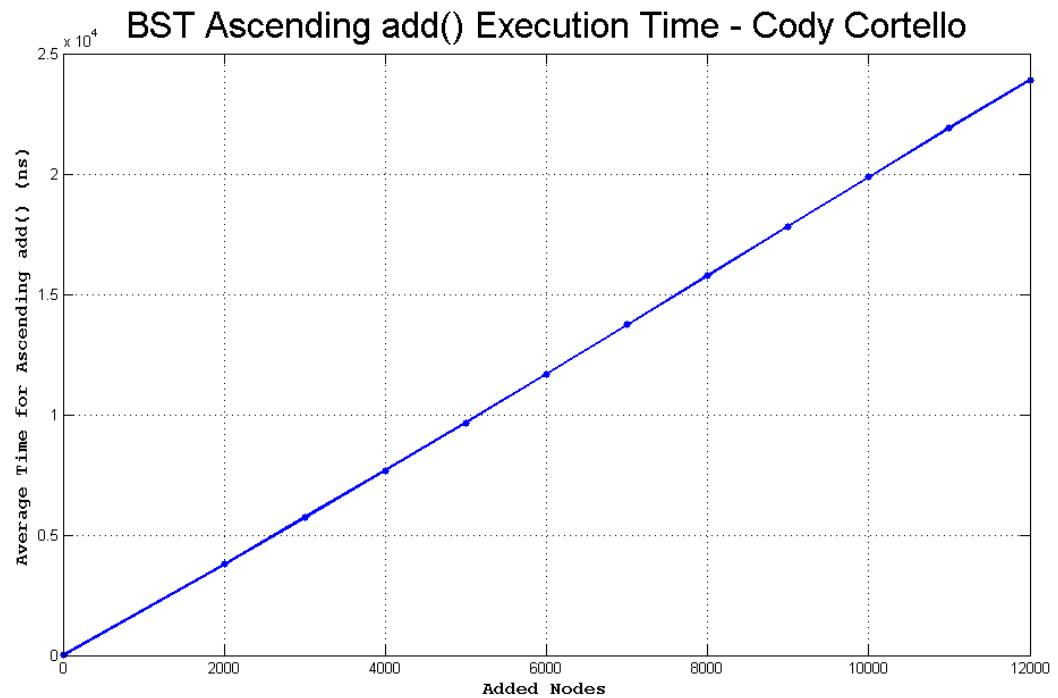


Assignment 6 Analysis

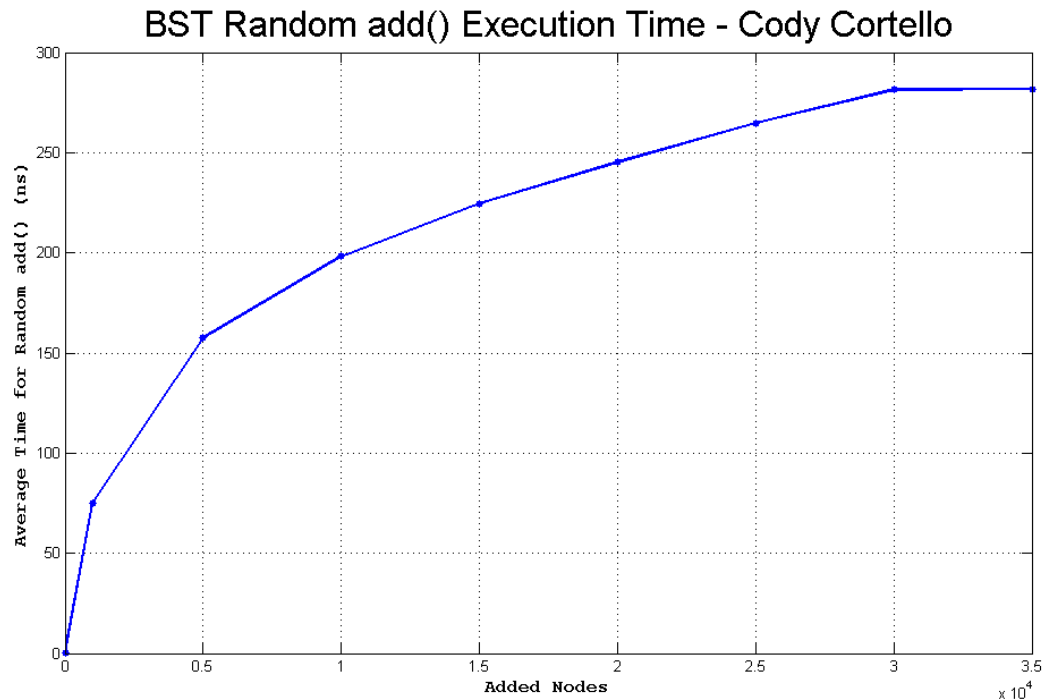
1. My programming partner is Casey Nordgran, but I will be submitting the source code for our project.
2. Casey is fun to work with, he understands Java and implementation very well, and he seems to have a great grasp of the concepts we're learning. I do plan on working with him again because he is a fantastic teammate as well as a knowledgeable one.
3. In order to measure the execution time of adding a sorted list of integers to an unbalanced BST I created an array of ascending integers with values equal to indices, and then used the timing code provided in class - and used in previous assignments - to find how long it takes to add each value to our BST. I did this several times for each size of n , found the average time (which was handled in the given timing code), and incremented n . Doing this through large values of n gave the following graph:



This graph is very clearly linear, which makes sense according to the expected shape of the substantiated BST. If solely ascending values are added to a BST with no self-balancing algorithm such as the one we implemented then each value will be the biggest value thus far, which will result in it being added as the rightmost node. The BST will then be functionally equivalent to a LinkedList (and will even look like one if drawn), though each add will have an execution time of n since every node has to be compared for each add.

A similar test was done for adding random integers instead of strictly ascending integers. The only code that had to be added was a permutation algorithm which would switch each element in the array with another element at a random index ten times. These ints were then added and timed with the existing code, resulting in the following

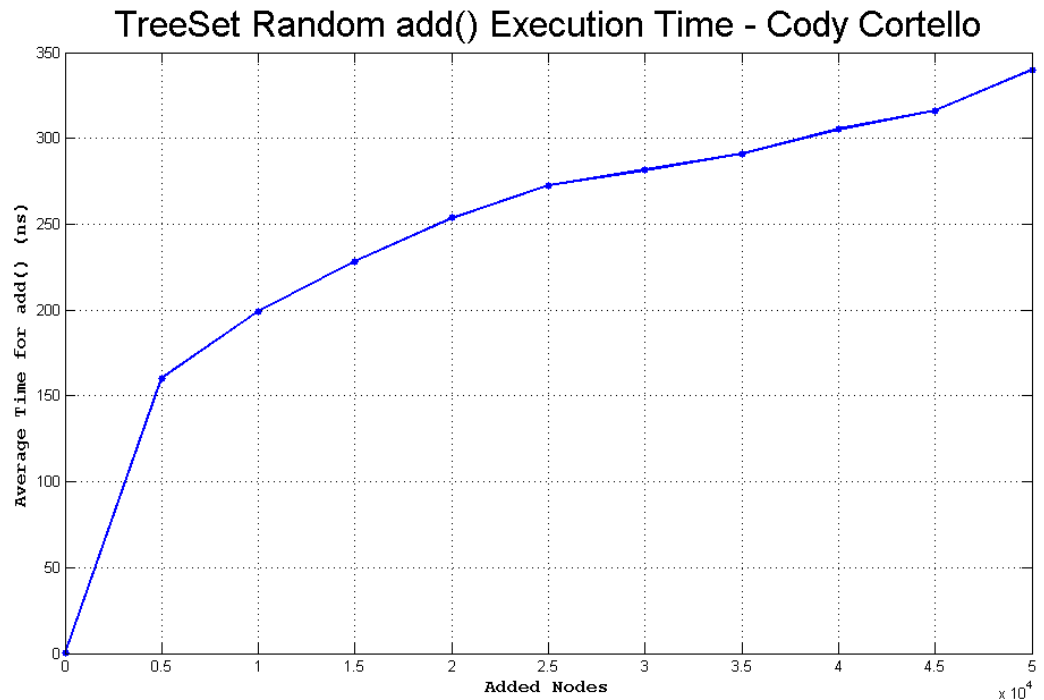
graph:



This graph quite clearly shows logarithmic execution time, which is what I would expect for adding random integers to a binary search tree. This result is due to the BST being near completely balanced – as a good randomization algorithm requires that neither left nor right directions are more heavily weighted at any point – and as such each random addition only requires a number of comparisons equal to the height of the tree, which is proportional to $\log(n)$.

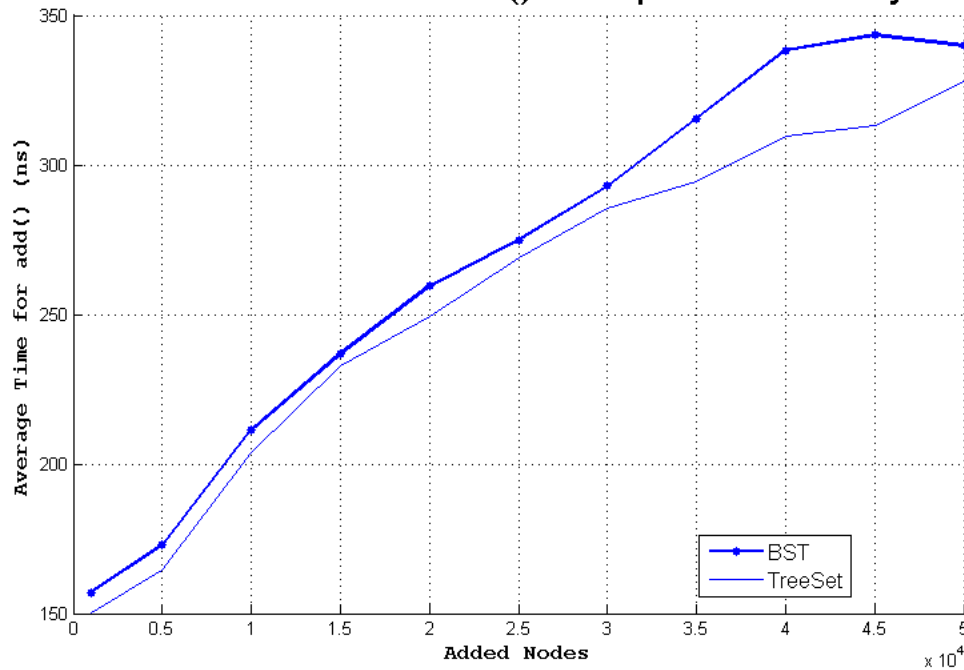
4. To test my implementation of a BST against Java's I only had to change the timed BST from `BinarySearchTree<Integer>` to `TreeSet<Integer>`, as the same method calls should be tested and the same functionality should be upheld in Java's `TreeSet`. This test was by

far the easiest to implement as changing the code to test it required change to two lines of code. Rerunning the timing on TreeSet produced the following graph:



Which, like the previous graph, is quite obviously logarithmic. Again, this is the expected result as the number of comparisons required to add a random integer to a BST is $\log(n)$. However, the real question is how much better a self-balancing BST such as TreeSet is in terms of execution time when compared to my BST implementation which has no self-balancing algorithm. To see the comparison I plotted the data values against each other and obtained the graph below:

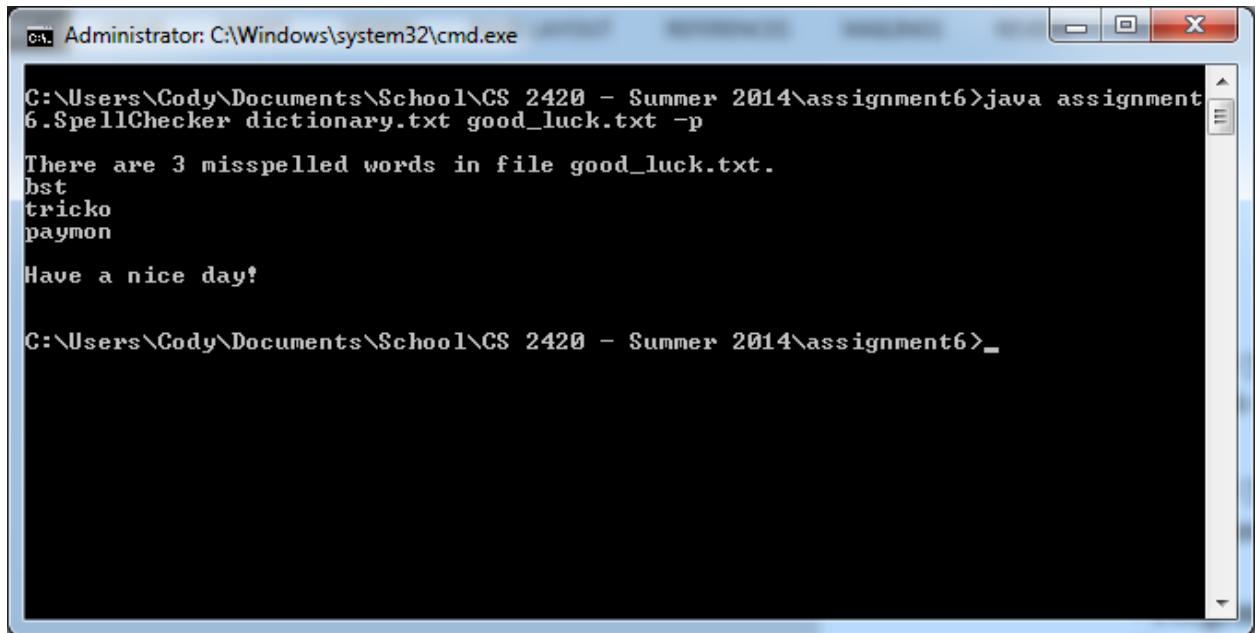
BST v. TreeSet Random add() Comparison - Cody Cortello



This shows that Java's TreeSet is only slightly better than my implementation, though the gap widens with higher values of n .

5. The problem with substantiating a dictionary using sorted-order entries is that the same situation is encountered as in problem 3 – that is, the BST created will be entirely linear and functionally equivalent to a LinkedList, and will have an execution time of $\frac{n}{2}$ for average random accesses. To fix the problem I would use a self-balancing BST. Using a randomly permuted dictionary file would also make the BST random, but it would necessitate creating randomly permuted dictionary files and the BST created wouldn't be completely balanced and the runtime would therefore be slower.

6.



```
Administrator: C:\Windows\system32\cmd.exe

C:\Users\Cody\Documents\School\CS 2420 - Summer 2014\assignment6>java assignment
6.SpellChecker dictionary.txt good_luck.txt -p

There are 3 misspelled words in file good_luck.txt.
bst
tricko
paymon

Have a nice day!

C:\Users\Cody\Documents\School\CS 2420 - Summer 2014\assignment6>_
```

7. Casey and I spent about thirteen hours total on this assignment.