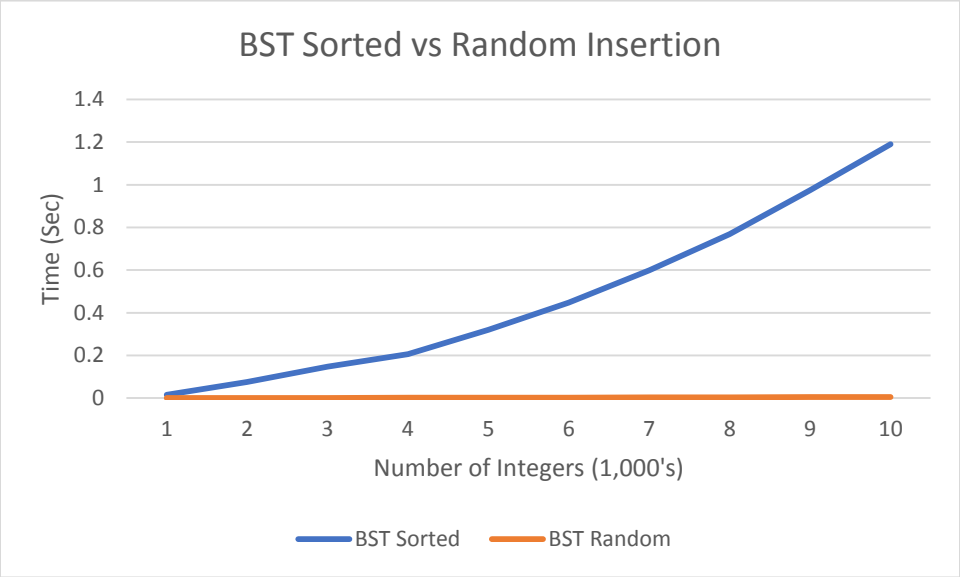
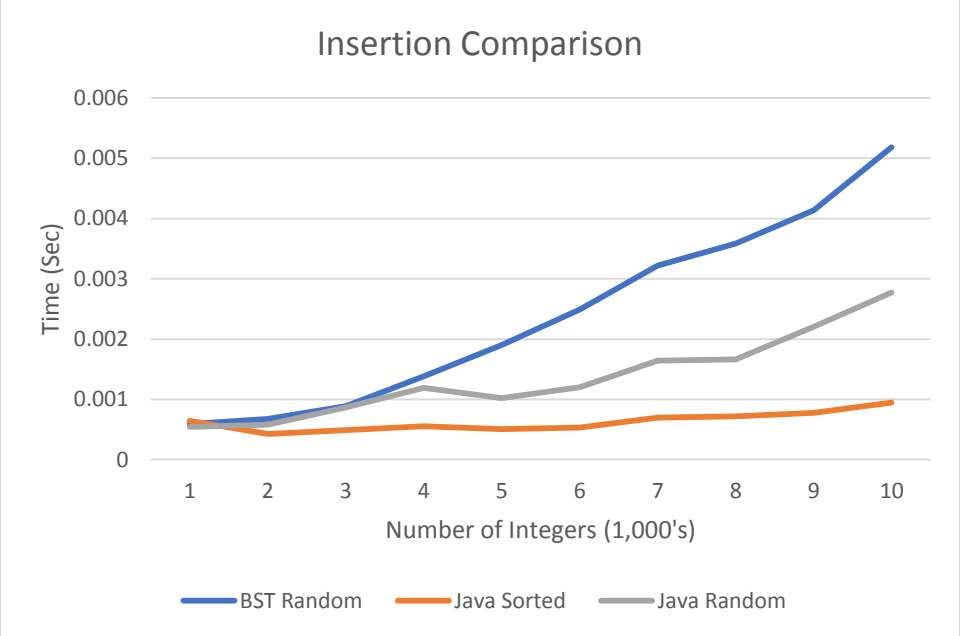
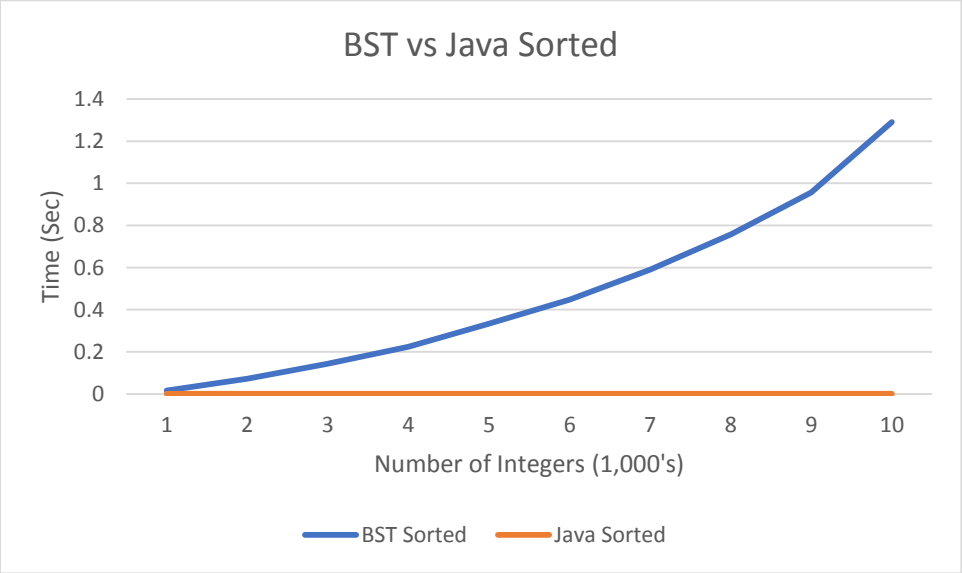
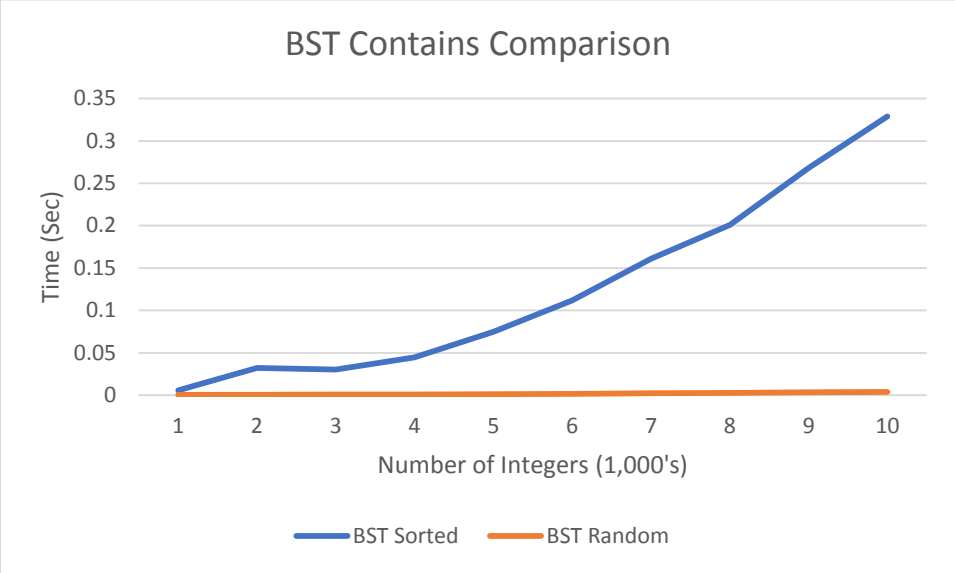
Brian Burton
03/02/17
CS 2420

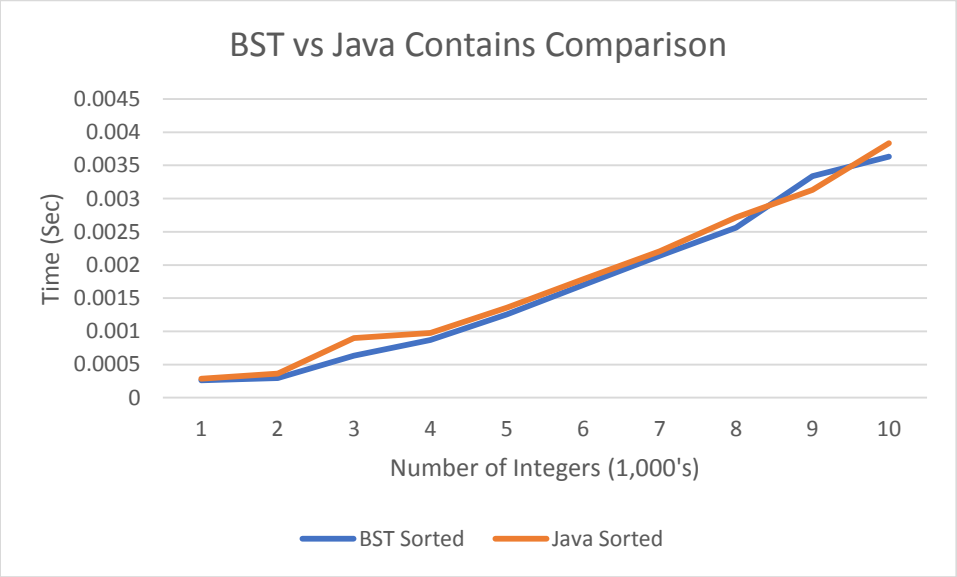
Analysis Document






BST vs Java Sorted
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Time (Sec)
1
2
3
4
5
6
7
8
9
10
Number of Integers (1,000's)
BST Sorted
Java Sorted


BST Contains Comparison
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Time (Sec)
1
2
3
4
5
6
7
8
9
10
Number of Integers (1,000's)
BST Sorted
BST Random



1. Overview of Project

I learned that Binary Search Trees are a more efficient data structure than a Linked List. This is because when you have a lot of data, the big O cost to add, remove, and find is log of N as long as the Tree is balanced. For a Linked Lists, the big O cost to add, remove, and find is N. The Binary Search Tree assignment also taught me how to use graphing tools so that you can better visualize what your code is doing.

2. Experiments and what the Graphs tell us

What I did in my experiments was test how long it took for my Binary Search Tree class, and Java's Tree class to add elements that were sorted and shuffled. What I learned from this was that my class was not as good as Java's, but it was close. The reason that there was a difference is because Java's class balances the tree while mine does not. That is why we see the slight difference between our times. This did confirm my theory about Binary Search Trees. I also figured that Java's class would be better.

I believe that the graphs show how you can add a lot of elements in a reasonable amount of time, as long as the tree is balanced. Searching and adding seem to have the same cost of log of N. I am not sure what the cost is to remove a node from a Binary Search Tree however. First you have to find the Node, so that would be Log of N, but then you have to remove that node then redirect the references. Therefore my guess is that removing takes a bit longer.

3. The Class Hierarchy

The Node and Binary Search Tree classes defiantly interact with each other. This is because a Binary Search Tree is made up of Nodes. Without Nodes you wouldn't have a Binary Search Tree. You would just have normal data. This is one way that the Node class supports the Binary Search Tree class. Another way it supports it is it allows the Binary Search Tree class to use abstraction. This allows the users to use the class without needing to fully understand what is happening. The Binary Search Tree supports the Sorted Set by allowing the Sorted Set data to be interacted with much faster than otherwise. It also allows the data to still be sorted.

4. Iteration vs Recursion

Binary Search Trees are naturally recursive because the data structure is always the same. Everything that is less than the root node goes to the left, everything greater to the right. They also only have a few operations that you can perform. This makes recursion oftentimes an easier rout. In most cases it was the best way to start at the bottom of the tree and work your way back up using the Stack. That being said, recursion was not always the answer.

We used recursion in every method and never used iteration. We could have used iteration however in most every method. An example is the add method. You could just use a couple of while loops to find the correct to add the new node and then add it. The reason that some methods are easily interchangeable is because of the way the Binary Search Tree is set up. There is only one way to add, remove and to find. The rules will never change. That makes it so you can go through the tree using recursion or iteration. There is one exception however, and that is the contains method. You are not able to go back up through the tree checking if an element is in it without using the Stack or an array that acts like a Stack.

5. Software Development Log

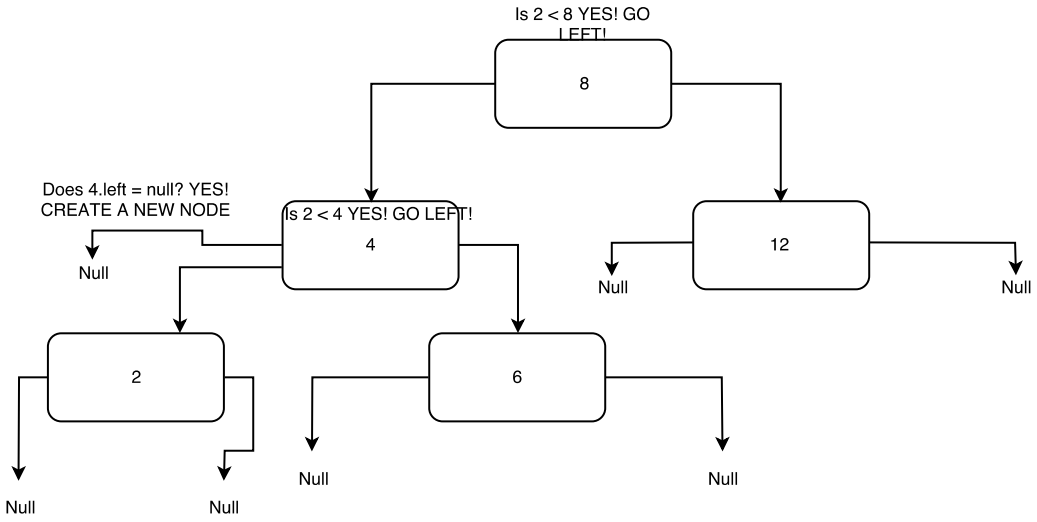
For this project, I probably spent around 12 hours. The most time consuming part was figuring out how to graph the Binary Search Tree and the remove method. Those were defiantly the hardest methods to implement for me. This was also the problems that took the most time to solve. In the future I am going to read the instructions better. Once I did that things began to make more sense. I did allocate enough time and did not wait until the last minute. Writing tests first also was I big help since it allowed us to see the problem, debug it, and then fix it faster in my opinion.

6. Thought Problems

The problem that you will come across when you add every word in the dictionary starting from the beginning and finishing at the end will be that you will have a right heavy tree. This would be not any better than a Linked List. The way that you could fix this problem is you could randomize all of the words in the dictionary before putting it into your Binary Search Tree. This way you would be able to create more of a balanced tree.

7. Algorithm Drawings

Add(2)



Remove(2)