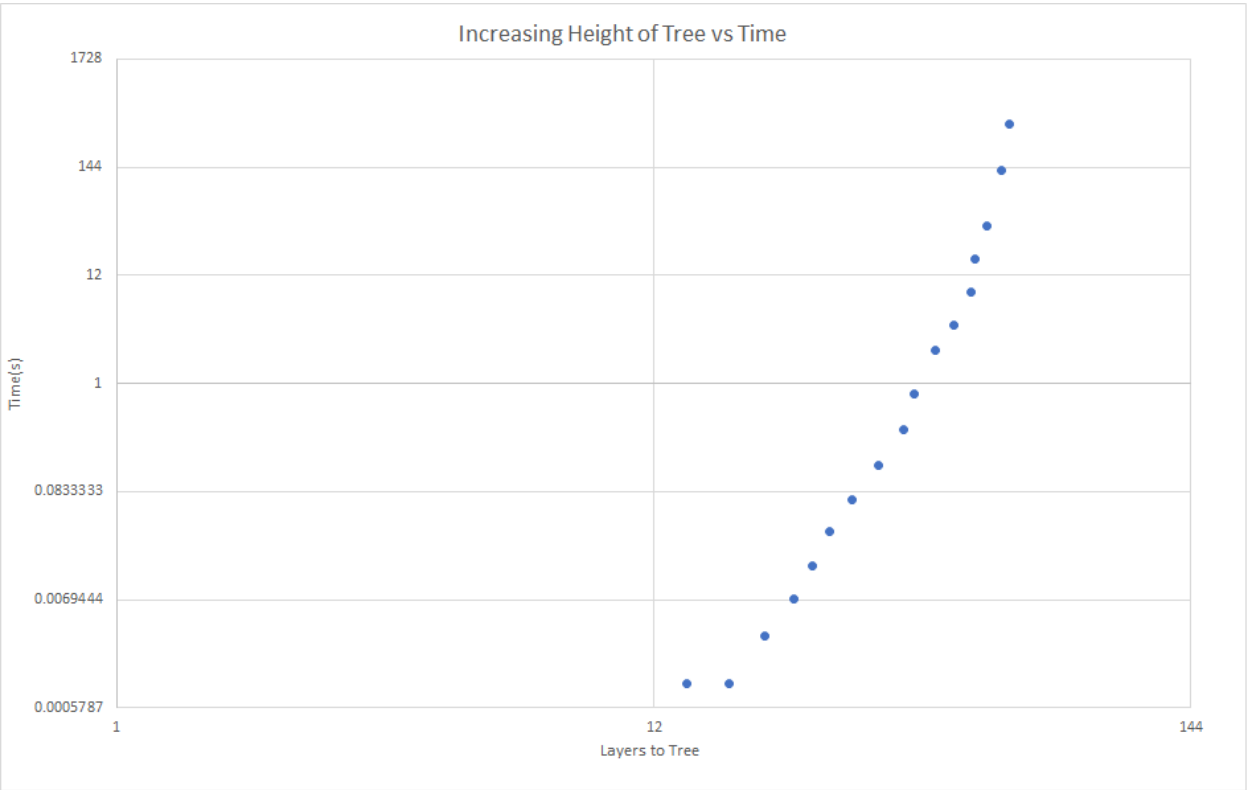
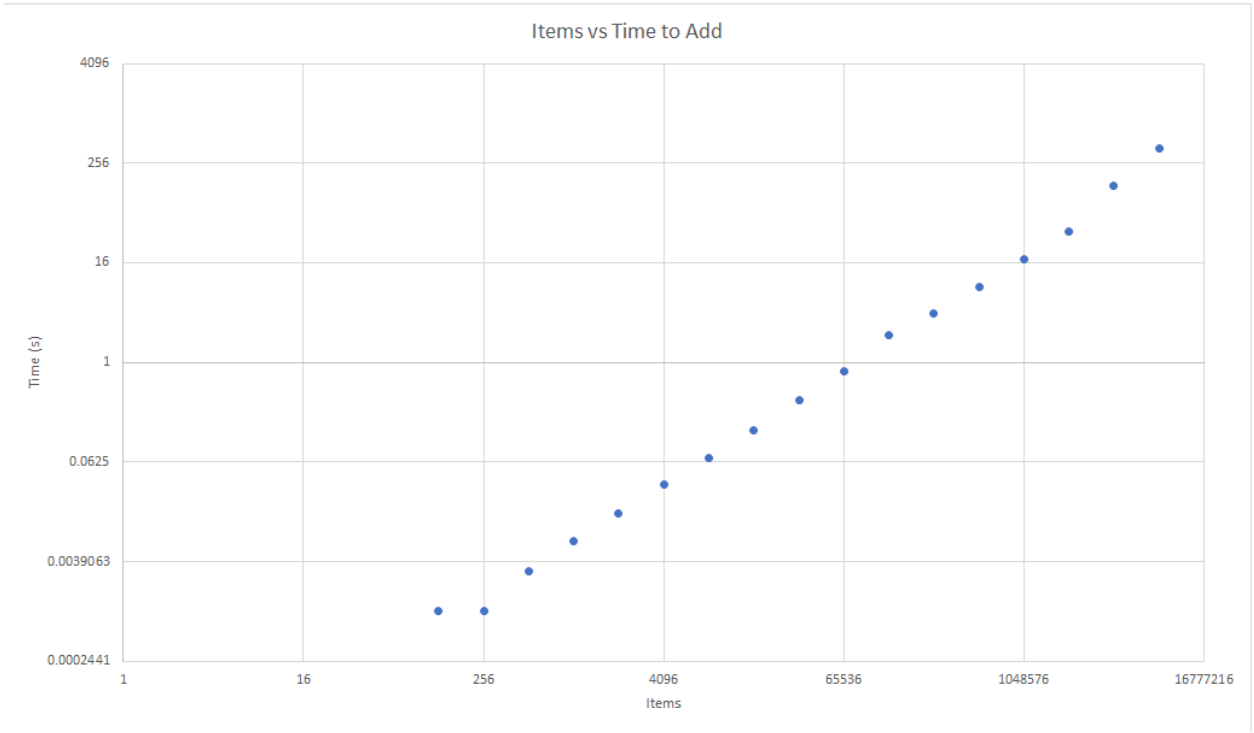
Intro:

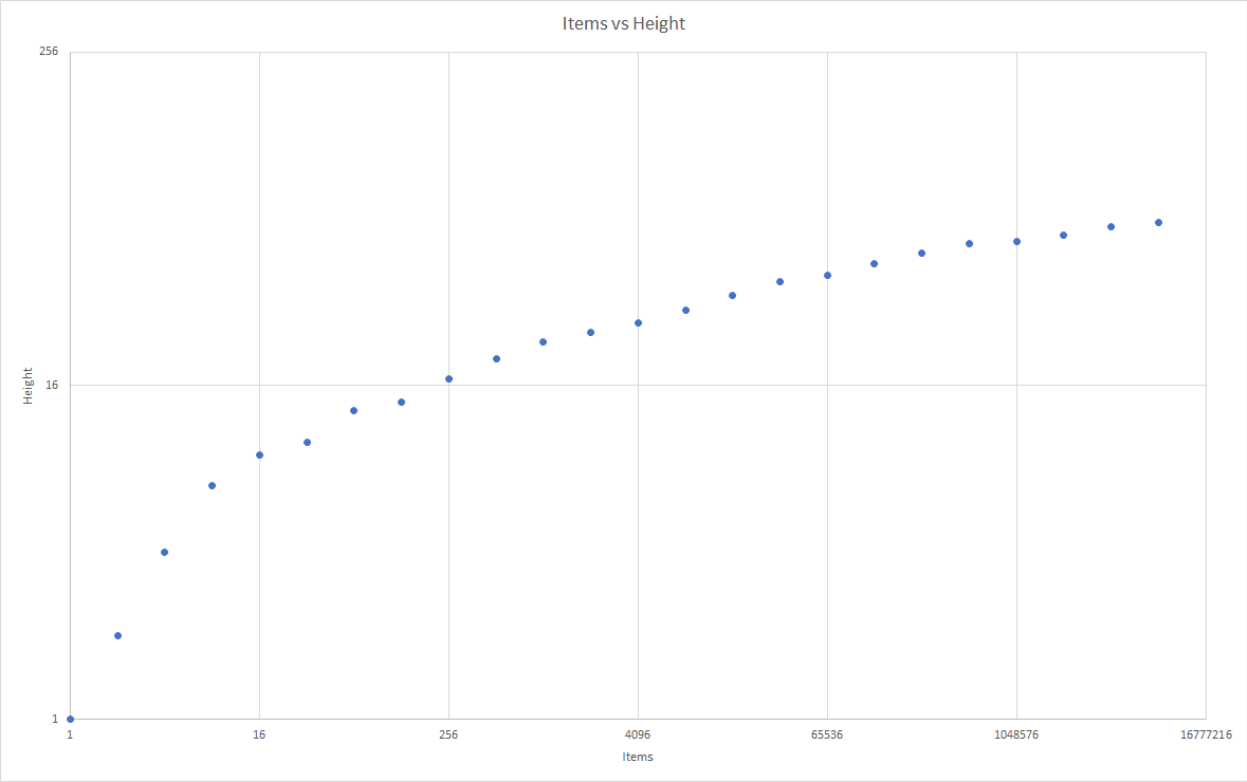
The Binary Search Tree was interesting to implement because I was somewhat familiar with trees because of the Heap Sort I wrote for the sorting assignment. Apparently my Heap Sort was not a correct or good version, but I still learned about trees when writing the code for the sort. So I tried to connect the BST in this assignment to the Heap Sort, which also used Binary Trees (but I think the Binary Tree used in that is slightly different). However, I still found some difficulty in understanding the use of recursion in the methods as that required a lot of mind-bending to understand. I'm not entirely sure I have a one-hundred percent grasp on it, but I wrote all the functions with almost no code from the internet (evident by the fact my deletion is very ugly, and I wasn't confident enough that it worked to benchmark).

Data:





Here I benchmarked how long it took to add so many layers to a tree, and also how long it took to simply add *n* items to it. Every point represents a tree with 2^*i* elements in it, and corresponds to a time in seconds that it took to create that tree. This plot shows how many layers, or the height, that tree had. The number of layers could depend on the randomness of the numbers generated, so if I used a different seed for different tests than the heights could be different. For instance, if the tree had three items in it, then it could either have a height of two or three.



Here is a graph comparing the number of items to the height of the tree. I was very surprised with how well this trend fit with a logarithmic graph, and I was expecting for there to be a lot more outliers or for it to be more scattered overall. Overall I thought this was a cool graph to look at, and I was scared my height function was horribly wrong but I think this proves that it's not.

Conclusions:

Overall, I was not expecting a data structure that uses recursion to be faster than other ones such as ArrayList or LinkedList. I can understand it being faster for certain operations, though, like containing or finding a specific value. Instead of having to go through all *n* items, the tree makes it possible to get to a value much quicker as it knows where the value you want is based on how it compares to other values. Implementing a delete function was harder than I expected, and I'm still unhappy with how mine turned out so I will most likely finish it this

weekend out of spite and to prove to myself I can do it. Also in conclusion, I think that Trees are my favorite with ArrayList coming in second. I think LinkedList is disgusting.