- insert = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- removeById = O(log n) where n is the number of nodes in a tree as the recursive calls split the amount of nodes to be searched in half each time
- searchById = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- searchByName = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- printInorder = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- printPreorder = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- printPostorder = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- printLevelCount = O(n) where n is the number of nodes in a tree as the recursive calls touch every node
- removeInorder = O(n log n) where n is the number of nodes in a tree as the recursive calls touch every node, and the intertwined function (removeById) splits the amount of nodes (n) to be searched in half each time (once that function is called)

The worst case time complexity is O(n log n) for the removeInorder function. The removeInorder recursive calls (where n is the number of nodes in a tree) make the function O(n) to find the correct inorder node to remove. Then, the function removeById is called with the id passed in from the correct inorder node found. The removeById function is an O(log n) operation because it cuts the searching it has to do in half each recursive call (where n is the number of nodes in a tree), based on if the id passed in is greater than or less than the current node. Therefore, the worst case time complexity of a function in this program is O(n * log n) ~ O(n log n).

I learned from this assignment that meticulous planning and a thorough understanding of the directions is very important before starting a project. When I first started this project, I didn't consider how I was going to update the heights after the rotations occurred. This was messing up my balance factor when I was trying to calculate it based on the height. The next problem I came across was that I didn't truly understand what was meant by prioritizing replacing a removed node with its inorder successor. I had to look at an AVL tree visualizer to fully understand this concept. Both these mistakes took me a long time to figure out. If I had to start over I would prioritize understanding the project and what all of its methods are supposed to do. I would also implement the three traversals first because they are crucial for testing test cases. I was also very familiar with these traversals so it would’ve been a very easy implementation that would’ve really helped in testing. I started testing after I had completed the insert function and found a lot of errors. If I had implemented the traversals and then periodically tested my insert function as I made it, I think I would’ve saved a lot of time.