Justin Galin
Complexity Analysis
**Disclaimer:

Every function works with at least one helper function, but all these helper functions either call the function that actually performs the action and are thus the same time complexity as the actual function or the helper function is of time O(1) since they contain no loops or recursive calls (Rotation functions)

- Insert
  - O(log(n)) where n is the number of nodes in the tree already since the node traversal up a binary tree take log(n) time
- Remove
  - O(log(n)) where n is the number of nodes in the tree already since we are essentially making use of a binary search since the AVL tree is relatively balanced and ordered , I do make use of a while loop to find the successor for the case where the deleted node has 2 children but it only occurs once and is O(logn) time so it does not increase the Big O classification of the algorithm.
- SearchID
  - O(n) time where n is the number of nodes in the tree since I used in order traversal to implement this function and if the searched for node is the last one it will have to iterate through all n nodes to find it.
- SearchName
  - O(n) time where n is the number of nodes in the tree, pre order traversal was used for this algorithm, so if the searched for node was the last node the program would have to iterate through all n nodes to get to it.
- printInOrder
  - O(n) time where n is the number of nodes in the tree, in order traversal so you have to visit every node
- printPreOrder
  - O(n) time where n is the number of nodes in the tree, pre order traversal so you have to visit every node
- printPostOrder
  - O(n) time where n is the number of nodes in the tree, post order traversal so you have to visit every node
- printLevelCount
  - O(1) time since all I do is access the height attribute of the root value of the tree which is stored in memory and only takes O(1) time to access
- removeInOrder
  - O(n) time where n is the number of nodes in the tree, n time since this is an in order traversal, I do make use of a while loop to find the successor for the case where the deleted node has 2 children but it only occurs once and is O(logn) time so it does not increase the Big O classification of the algorithm.

What I learned

I learned more of what goes into making a large-scale project that is rigorously tested and has a lot of possible test cases. I underestimated how many edge cases functions like insert and remove would have and ended up being more pressed for time than I expected to be. If I had to start over I would have started a bit earlier and made sure I fully understood how recursion worked and how to make use of it as that was a big pain point at the beginning of the project for me.