Jai Patel

Insert Name ID: O(log n), where n is the number of nodes in the tree. The function has a for loop which checks if each character in the name is a real letter or a space. The time complexity of inserting into an AVL tree is O(log n). The function calls recursively and the rotation operations are constant. This makes it O(log n).

Remove ID: O(log n), where n is the number of nodes in the tree. The function calls itself recursively. Searching for a node in the tree takes O(log n) time. The node removal is constant but in the worst case it is O(log n) because of the scenario that you have to remove a node that has two children.

Search ID: O(log n), where n is the number of nodes in the tree. If you search by the number ID then the time complexity is O(log n). The tree is balanced and the function calls recursively.

Search Name: O(n), where the n is the number of nodes. When searching by name, the function is calling a helper function that goes through all the nodes and pushes all the nodes in a vector. Then it searches through the vector for the right name, and then goes through a for loop to go through the name and prints the id, which makes it O(n + n) which is O(n)

printInorder: O(n), where n is the number of nodes. This function calls another function that goes through the tree in the inorder way and puts the names into a vector. Then, it goes through the vector by a for loop to print the names which makes it O(n+n) which is O(n).

printPreorder:O(n), where n is the number of nodes. This function calls another function that goes through the tree in the preorder way and puts the names into a vector. Then, it goes through the vector by a for loop to print the names which makes it O(n+n) which is O(n).

printPostorder:O(n), where n is the number of nodes. This function calls another function that goes through the tree in the postorder way and puts the names into a vector. Then, it goes through the vector by a for loop to print the names which makes it O(n+n) which is O(n).

printLevelcount: O(1), where n is the number of nodes. This function is going through to count the level and it only prints the level of the tree which makes it O(1).

removeInorderN: O(n), where n is the number of nodes. The function calls a helper function which goes through the tree in the inorder traversal to find N which makes it O(n).

Reflection:
What did you learn from this assignment? -
I learned a deeper understanding of how an AVL tree works. I learned functions such as insert, search, and remove. I also learned how to manage different cases such as the name can only have letters.

What would you do differently if you have to start over? -
If I had to start over, I would look at different input scenarios before I start coding the AVL Tree. I would develop more test cases earlier so I can have less debugging near the end. I would also try to write out my conceptual logic on paper so I can visualize it better.