Kassahun Sanayew

1) The Insert command has a method of insert (string name, string ID). The time complexity of O(log n) is worst case because it maintains balance through rotations, this makes it so that the trees height stays logarithmitic.
2) Remove Command has a method of remote(string ID). The time complexity of O(log n) is worst case this is the searching for node and having to rebalance.
3) The Search (string name_or_ID) has a time complexity which is based on GatorID, and O(n) when searching through the name, the tree needs to be traversed.
4) Print level count printLevelCount(), this has a time complexity of O(1) this only needs to get the height of the node root.
5) Print Inorder, post order, preorder. Time complexity O(n) for each, since they need to visit all nodes in order.
6) Remove the inorder command which is removeInorder(int ), this has a time complexity of O(n) this is for going through the tree and the O(log n) for removing the node this would have an overall of O(n).

Reflection

This project helped me learn more about AVL tree balancing. This goes over rotating from the left to right, building out functions like deletion, insertion, and how they all work together to create the project. This gave me a better understanding of how to improve the algorithms that I use to be as good as possible. Next time I do this or something similar I would focus on creating a better validation system so that I could be able to handle input that is incorrect that ended up stalling the entire process quite a bit and writing out unit test as soon as possible as this would have saved time with debugging. This overall has reinforced the idea of self-balancing trees so that you can maintain the search times to be as efficient as possible which is key especially when dealing with more and more data.