STUDENT NAME : Jesse Watson

- What is the time complexity of each method (**corresponding to a command**) in your implementation? Reflect on the worst-case time complexity represented in Big O notation.
    - **PrintPreOrder** function is O(n^2).
        - Preorder_helper(node, b) is O(n) due to it checking every node in the tree.
        - Constructing the order string and going through the vector as well is O(n^2)in the worst case due to the linear transversal of the vector ”b” and then concatenating what is in vector “b” into “order” giving us O(n^2).
    - **Insert** Function Is O(logn)
        - This is due to the tree rebalancing the nodes as it is being constructed, making it easier to insert future nodes
    - **Remove** Function Is O(logn)
        - Again, this is O(logn) because the nodes are being balanced as the tree is being created, making it easier to remove nodes whether that be with 1,2 or even zero children
    - **Search(ID)** Function is O(n)
        - This is the log(n) due to the breadth first search that is taken to transverse the tree to search for the desired node which is my fault.I should use a depth first search to take advantage of the AVL rebalance but I already had code on Traversing through a tree so I just used that instead.
    - **Search(String)** Function is O(n)
        - Same as the last one but just with finding a String
    - **Printinorder** function is O(n).
        - This time complexity is due to visiting all the nodes exactly node once
        - The reason why this transversal isnt n^2 like preorder due to not having to concatenate a string that's within the the for loop
    - **printPostorder** function is O(n).
        - This time complexity is due to visiting all the nodes exactly node once
        - The reason why this transversal isnt n^2 like preorder due to not having to concatenate a string that's within the the for loop
    - **printLevelCount** function is O(n).
        - Breadth and Depth Search are the same in terms of time complexity but printLevelCount does go through every node leading us to O(n)
    - **removeInorder *N*** function is O(n).
        - So it does following operations which is inorder traversal (O(n)),secondly accessing element within the vector that was made by the traversal (O(1)),Thirdly Searching for node that was within the inorder traversal vector in the AVL tree itself (O(n)), Finally Removing the node and rebalancing the tree using ‘RemoveHelper2’ is O(log n)

- What did you learn from this assignment, and what would you do differently if you had to start over? [2 points]

- I learned that depth Breadth search have the same time complexity
- I wished I would of started sooner