1. Time Complexity of Functions
    a. **remove()** works in O(logn) time complexity because the tree remains balanced, and thus it can locate the node by traversing either left or right in each progression.
    b. **insert()** also works in O(logn) time because the balance of the tree allows for insertion to take shortcuts when searching for the correct placement of the node. The other functions within the insert function, like checking for valid names/ids or rotating
    c. **printPostorder()** works in O(n) because it must visit every node
    d. **printPreorder()** also works in O(n) because it must visit every node in the tree
    e. **printPostorder()** also works in O(n) because it has to visit every node in the tree
    f. **printLevelCount()** also works in O(n) time because the way that I implemented it is similar to finding the height of a BST. It must traverse all the way down to both sides of the tree (or subtree) to get the height.
    g. **removeInorder(N**) works in O(n^2) because it needs to traverse the entire trees, and store the nodes up to N in a vector. I then need to traverse the vector of N nodes, so it is N * N.
    h. **search()** works in O(n) because it has to traverse the entire tree in worst-case. I used a preorder traversal to implement this function for both searching names and id's.
2. This assignment made the rotations much more clear on how they work. I also think that it helped me compartmentalize the functionality of my code. You can't really get away with messy code or shortcuts for this project, and there are a lot of situations that need to be broken down into subproblems. I wished I reviewed test cases on my whiteboard before starting the coding process. Not doing this caused quite a bit of confusion for me in the middle of project, especially with the insert and remove functions. But after going over the rotations on Stepik and watching youtube/lecture videos on how rotations can be visualized, it was much easier to understand the implementation. Overall, breaking it down into mini tasks was the most beneficial tactic for progressing in implementing the trees functionality and to do it over , I would focus on having a better understanding of how these mini tasks fit into the overall process.