Dylan Long
GatorAVL Project

Time Complexity Analysis

- **insertNode (Insert NAME ID)**
  O(log n): The insertion in an AVL tree involves a single path traversal from root to leaf, then at most a constant number of rotations to maintain the balance. The height of an AVL tree is O(log n), making the worst-case complexity O(log n).

- **removeNode (Remove ID)**
  O(log n): Similar to insertion, removal involves a single path traversal and a constant number of rotations for rebalancing. The height of the tree dictates the time complexity, which is O(log n) in the worst case.

- **searchId (Search ID)**
  O(log n): Searching for an ID involves traversing a path from the root to a leaf or a node with a matching ID, with the tree's height being O(log n).

- **searchName (Search NAME)**
  O(n): Requires traversing the entire tree to find all nodes with the given name, making it O(n) since every node must be visited.

- **printInOrder, printPreOrder, printPostOrder**
  O(n): Each of these functions performs a full traversal of the tree, visiting each node once, resulting in a time complexity of O(n).

- **printLevelCount**
  O(n): Involves a full traversal of the tree to determine the height of each node, thus the complexity is O(n).

- **removeInOrder (RemoveInorder N)**
  O(n): This operation requires an in-order traversal to find the Nth node, which is O(n), followed by removal, which is O(log n). However, the traversal dominates the complexity, making it O(n).

Reflections

- From this assignment, I've learned the importance of understanding the balance and traversal mechanisms of AVL trees, which directly affect the efficiency of various operations. The balance factor and rotation techniques are crucial for maintaining optimal time complexities.
- If I were to start over, I'd better my approach by integrating unit testing early in the development process to more efficiently debug and validate the code. I would also optimize the search and removal algorithms in operations like searchName and removeInOrder in order to improve overall efficiency. This combined focus on unit testing and algorithm optimization would lead to a more robust and efficient implementation.