ANALYZING TIME COMPLEXITIES FOR PROJECT 1

Rainier St. Fort

1. insert(const std::string& UFID, const std::string& name)

- **Operation**: Inserts a student into the AVL tree.
- **Steps**:
  - Calls isValidID(), which takes **O(1)** time.
  - Calls isValidName(), which takes **O(n)** time (with n being the length of the name string).
  - Calls isUniqueID(), which has a worst-case time complexity of **O(log n)**, since in the worst case, it must traverse the tree to check for uniqueness.
  - Calls the actual insertion operation, which has a time complexity of **O(log n)** for insertion in a balanced AVL tree.
  - Balancing the tree, using balanceNode(), also has a time complexity of **O(log n)** because rotations are localized and occur at most once for each level of the tree.

Total Time Complexity:

O(logn)

2. deleteNode(const std::string& UFID)

- **Operation**: Deletes a node by UFID from the AVL tree.
- **Steps**:
  - Calls searchByID() to check if the node exists. This takes **O(log n)** because the tree is balanced.
  - If found, calls deleteNode(), which also takes **O(log n)** since deletion in a balanced AVL tree involves searching for the node to delete.
  - After deletion, balancing the tree using balanceNode() takes **O(log n)** as well, as rotations occur at most once per level of the tree.

Total Time Complexity:

O(logn)

3. searchByID(const std::string& UFID)

- **Operation**: Searches for a student by UFID.
- **Steps**:
  - The search operation is a basic binary search in a balanced AVL tree, which takes **O(log n)** because it recursively divides the tree into halves.

Total Time Complexity:

O(logn)

4. searchByName(const std::string& name)

- **Operation**: Searches for a student by name.
- **Steps**:
  - Unlike searching by UFID, searching by name requires traversing the entire tree because there's no guarantee that names are in any specific order in an AVL tree.
  - This results in a time complexity of **O(n)** in the worst case, where n is the number of nodes in the tree, since you may need to check every node in the tree.

Total Time Complexity:

O(n)

5. deleteNthInorder(int N)

- **Operation**: Deletes the Nth student in an inorder traversal.
- **Steps**:
  - First, it calls collectUFIDs(), which performs an inorder traversal of the entire tree. This takes **O(n)** time, where n is the number of nodes in the tree.
  - After collecting the UFIDs, it deletes the Nth node, which is done by calling deleteNode(). As discussed earlier, deleteNode() takes **O(log n)**.

Total Time Complexity:

O(n)

All **traversal** functions (`inOrder()`, `preOrder()`, and `postOrder()`) visit every node exactly once, leading to a time complexity of **O(n)**.

The printLevelCount() function simply accesses the stored height of the root, resulting in **O(1)** time complexity.

Summary of Time Complexities

- **insert(const std::string& UFID, const std::string& name)**: **O(log n)**
- **deleteNode(const std::string& UFID)**: **O(log n)**
- **searchByID(const std::string& UFID)**: **O(log n)**
- **searchByName(const std::string& name)**: **O(n)**
- **deleteNthInorder(int N)**: **O(n)**

- **Inorder Traversal / Printing**: **O(n)**
- **printLevelCount()**: **O(1)**

REFLECTION

During this project I learned how meticulous you have to be in writing unit case tests, but also that writing unit tests can be very helpful in making sure you have the right code. The tests can point out to you logical errors in your code that may not pop up as errors when compiling. I learned a lot about trees as well (through struggle? Maybe) and about time complexities of functions used to make an AVL tree work. If I were to start this project over, I would make sure I made the UFID variable a string instead of an int. I almost tore my hair out trying to account for changing my functions to work with a string UFID instead of an int UFID.