Daniel Shmul

Project 1: Gator AVL

<u>Computational Complexity:</u>

Insert

- Best: O(n log n) where n is the number of nodes in the tree because after insertion the function calls a helper updateBalance to calculate the balance for each node in the tree and this is done by the helper getHeight which progresses from a node to its leaves.
- Average: O(n log n) where n is the number of nodes in the tree because after insertion the function calls a helper updateBalance to calculate the balance for each node in the tree and this is done by the helper getHeight which progresses from a node to its leaves.
- Worst: O(n log n) where n is the number of nodes in the tree because after insertion the function calls a helper updateBalance to calculate the balance for each node in the tree and this is done by the helper getHeight which progresses from a node to its leaves.

Remove

- Best: O(log n) where n is the number of nodes in the tree because the function recursively splits the tree in half every call.
- Average: O(log n) where n is the number of nodes in the tree because the function recursively splits the tree in half every call.
- Worst: O(log n) where n is the number of nodes in the tree because the function recursively splits the tree in half every call.

Search NAME

- Best: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Average: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Worst: O(n) where n is the number of nodes in the tree as the recursive call touches every node.

Search ID

- Best: O(1) when the node being search is the root.
- Average: O(log n) where n is the number of nodes in the tree as the recursive function splits the tree in half each call.
- Worst: O(log n) where n is the number of nodes in the tree as the recursive function splits the tree in half each call.

Print Traversal (preorder, inorder, postorder)

- Best: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Average: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Worst: O(n) where n is the number of nodes in the tree as the recursive call touches every node.

Print Level Count

- Best: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Average: O(n) where n is the number of nodes in the tree as the recursive call touches every node.
- Worst: O(n) where n is the number of nodes in the tree as the recursive call touches every node.

Remove Inorder

- Best: O(n) where n is the number of nodes in the tree because inorderVector recursively touches every node. The remove operation itself is O(log n).
- Average: O(n) where n is the number of nodes in the tree because inorderVector recursively touches every node. The remove operation itself is O(log n).
- Worst: O(n) where n is the number of nodes in the tree because inorderVector recursively touches every node. The remove operation itself is O(log n).

What did you learn from this assignment and what would you do differently if you had to start over?

Since every operation is recursive by nature, this assignment significantly increased my experience and comfort with recursive functions. Additionally, it also solidified my understanding of self-balancing trees and the corresponding operations by having to solve each one individually. If I were to start over, I would attempt to optimize the time complexity of these operations. Theoretically the insertion and deletion operations can have a worst-case time complexity of O(log n), but because I have helper functions that recursively call every node within one another my computational complexity suffered. This could be achieved by adding an attribute to every node such as height, which wouldn't have to be calculated for every node by an insertion call as my implementation does with balance. Other than this, I would separate the code into a source and header file in order to help with organization and cleanliness.