Justin Beauchesne

Gator AVL Project Analysis:

Time Complexity:

Notes:

- Majority of the necessary functions are wrappers to the main functionality due to the need to keep certain features of the AVl class private.
- n is the number of nodes in the tree

Insert (insertHelper): O(logn) * (O(n) + O(n) +O(1)) = O(logn * (2n + 1)) = O(nlogn)

- Log(n) since the bst is self balancing the worst case is logn for insertion, although if not including rotations that balance the tree the worst case would have been O(n), done recursively.
  - O(2n) for the maxHeight helper function runs twice to find the height of the left and right sub-trees thus checking every node recursively.
  - O(1) for rotations both rotateLeft and rotateRight
- Logn * (2n +1) since the max height and rotations occur nested inside the insert recursion

Print Traversals: preorder, inorder, and postorder: O(3n) + O(n) =O(4n) = O(n)

- O(3n) for the 2 recursive transversal and insertion into vector
  - O(n) for insertion into container
  - O(n) for 2 recursive calls
- O(n) for print function that iterates through entire container

remove(int): O(2logn) + O(logn) = O(3logn) = O(logn)

- O(2logn) from the helper function that recursively finds the node matching the ID number. This helper happens twice worst case if the node has two children and the main remove function is called again recursively, this recall can only happen once
- O(logn) for the helper function that iteratively finds the successor of a node, this successor is used for the recall of the remove function mentioned above which prevents multiple recalls since the successor only has at most 1 child.

removeInorder(name): O(n) + O(n) + O(logn) = O(n)

- O(n) for inorder helper that returns a vector of all nodes inorder
- O(n) for search of Nth node in vector at worst case
- O(logn) from previous remove function (i just reused the the other remove function although i know its less efficient this way)

printLevelCount: O(n)

- O(n) for the maxHeight helper function that is called once but checks every branch/ node

search(ID): O(logn)

O(logn) for the use of helper function to find the node in balanced tree

search(name): O(2n + logn) + O(n) = O(n)

- O(2n+logn) for the helper function of the preorder
  - O(2n) twice recursive call
  - O(logn) insertion into queue

- O(n) for iteration through elements in queue, given that worst case every element was added to queue

Reflection:

Overall, I'd say that my understanding of pointers and pointer logic has improved a lot. This improved both from the rotation logic and recursive deletion logic. Although if I were to do things differently I would have looked over the slides more thoroughly because I was originally unaware of pseudo code that was provided to help me better understand the assignment. This would have saved me a lot of time and debugging since the resources where already there from some parts.