Jordan Hanson
UFID: 9392-2594
jhanson1@ufl.edu

How to run the program

- Run command: make
- Run command: java avltree filename

How to clean up project

- Run command: make clean

Project Structure and Time Complexities

It is split up into three different files Node.java, Avl_Tree.java, and avltree.java.

Node.java:
This file contains the data structure that makes up the tree and holds the keys.

Node(int data) O(1) -> constructor for the Node class called with the data and sets all Node variables to either null or to the data passed in. Initially height is set to 1.

avltree.java
This file is the entry point for the program.

main(String[] args)-> handles all input and output operations as well as calling tree functions based on the input. All input is from a file which is read in using a filename passed into the program. All output is in a file named “output_file.txt” which is created every time the program is run. This function has a tree object that it performs operations on.

Avl_Tree.java
This file contains all the AVL tree operations.

Avl_Tree() -> O(1) time complexity. This function initializes the tree to an empty tree with a null root.

getHeight(Node nod) -> O(1) time complexity. Function gets the height of a node or returns 0 if node is null.

setHeight(Node nod) -> O(1) time complexity. Function sets the height of a node which is the max of its two children.

preOrder(Node current) -> O(n) time complexity. Returns the pre order traversal of the tree used for testing purposes.

levelOrder() -> O(n) time complexity. Returns the pre order traversal of the tree used for testing purposes.

insert(int key) -> O(log(n)) time complexity. Calls an insert helper function with the root and key.

insertRecursion(Node current,int key) -> O(log(n)) time complexity. Inserts key in the correct place and then balance the tree. Tree is balanced by the balance function.

balance(Node current) -> O(log(n)) time complexity. Balances the tree if needed at the current node based on height of children and calls the correct rotation function.

rRotate(Node root) -> O(1) time complexity. Performs a right rotate.

lRotate(Node root) -> O(1) time complexity. Performs a left rotate.

lrRotate(Node root) -> O(1) time complexity. Performs a left right rotate.

rlRotate(Node root) -> O(1) time complexity. Performs a right left rotate.

search(int key) -> O(log(n)) time complexity. Searches the tree for the correct key and returns null if it does not exist.

search(int key1, int key2) -> O(n) time complexity. Calls the searchRange function.

searchRange(Node root,int key1, int key2, List<Integer> keys) -> O(n) time complexity. Searches the tree for all nodes within the range by traversing the tree based on the key values.

biggest(Node current) -> O(log(n)) time complexity. finds the biggest node in the tree rooted at the current node.

delete(int key)-> O(log(n)) time complexity. Calls the recursive deleteNode function.
deleteNode(Node current, int val)->O(log(n)) time complexity. Deletes the node with key equal to val and replaces it with the largest value in the left subtree. The balance function is then called on every node encountered during delete.