**AVL Group assignment**

Populate a tree via a text file (input.txt) Make sure that after every insert, the tree is balanced. At the end, display the tree in level format. Make sure to include the height and the balance factor of every node in your output. Redirect the display to an output file (output.txt)

Hint:

//I will not accept any other algorithm

//This is not a recursive algorithm

node \* rebalance(node \*node){

node->height = max(height(node->left), height(node->right)) + 1;

int balance = getBalance(node); //node->left - node->right

/\*do rotations as necessary

If Left heavy outside : return rightRotate(node);

If right heavy outside: return leftRotate(node);

If left heavy inside: left rotation first, right rotation 2nd, return top node

node->left = leftRotate(node->left);

return rightRotate(node);

if right heavy inside: right rotation first, left rotation 2nd, return top node

node->right = rightRotate(node->right);

return leftRotate(node);

if no rotation, return node

\*/

}

//non-tail recursive algorithm because of rebalance

node\* insert(node\* node, int key)

{

//recursive Code for inserting a node

//When insert happens set height to 0 for the node

if (node == NULL)

return(newNode(key));

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

node=rebalance(node); //update heights and rebalance

return node;

}

node \*leftRotate(node \*x){

struct node \*y=x->right;

//add more code to rotate to the left, update heights for x and y

//return y

}

node \*rightRotate(node \*x){

struct node \*y=x->left;

//add more code to rotate to the right, update heights for x and y

//return y

}