Name : Ramzy Izza Wardhana
NIM : 21/472698/PA/20322
Class : IUP CS-1

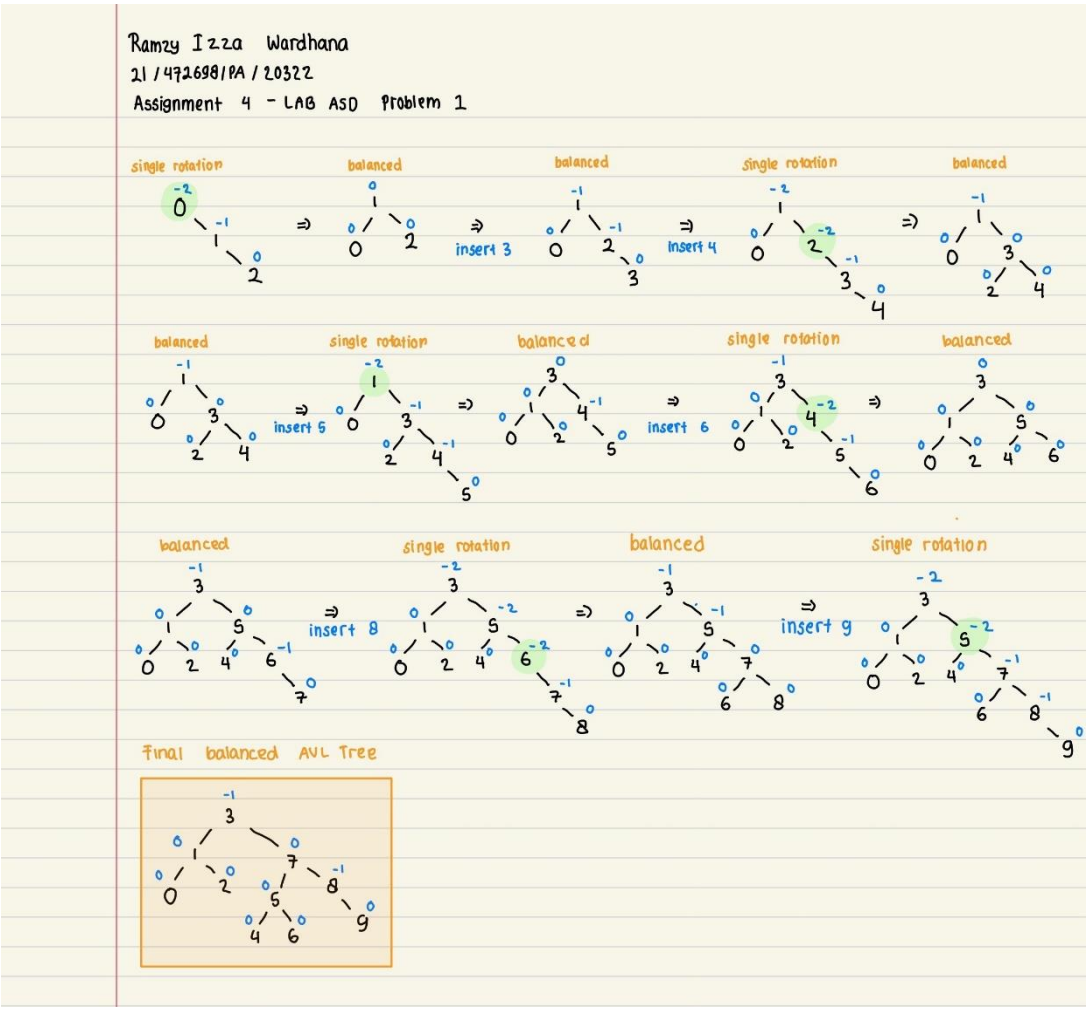
Assignment 4 – AVL Tree

Lab Algorithm & Data Structure CS1

1. Implement AVL from this chapter with data{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} in sequence. Draw the created AVL Tree!

   Source Code (Zip file) =

   https://drive.google.com/file/d/1zLZoSH-3yeo4Szq2t6Uv8seZfQAgeFES/view?usp=sharing



2. From Binary Search Tree implementation from chapter 3, create one method to check whether the Binary Search Tree is AVL Tree or not.

```
Input size of data: 10
Input 10 amount of data seperated by spaces: 0 1 2 3 4 5 6 7 8 9

[After] Constructured AVL Tree written in Preorder Traversal is:
3 1 0 2 7 5 4 6 8 9
It is Balanced BST / AVL Tree

[Before] Regular Binary Search Tree written in Preorder Traversal is:
0 1 2 3 4 5 6 7 8 9
It is Unbalanced on RHS of BST
PS C:\Users\themi\Downloads\Assignment 4 - AVL Tree Source Code>
```

Main.java

```java
public class Main{
    Run | Debug
    public static void main(String[]args){
        AVLTree AVL = new AVLTree();
        PreOrder order = new PreOrder();
        Scanner sc = new Scanner(System.in);

        System.out.print("Input size of data: ");
        int size = sc.nextInt();
        System.out.print("Input " + size + " amount of data seperated by spaces: ");

        for(int i = 0;i < size; i++){
            int input = sc.nextInt();
            AVL.root = AVL.insertion(AVL.root, input); //Construct AVL Tree - Balanced
            AVL.addNode(new Node(input)); //Construct Binary Search Tree - Unbalanced
        }
        //Question 1
        System.out.println("\n[After] Constructured AVL Tree written in Preorder Traversal is:  ");
        order.preOrder(AVL.root);
        //Question 2
        System.out.println("\n" + AVL.checkUnbalance(AVL.root)); //Check AVL Tree

        System.out.println("\n[Before] Regular Binary Search Tree written in Preorder Traversal is:  ");
        order.preOrder(AVL.root2);
        //Question 2
        System.out.println("\n" + AVL.checkUnbalance(AVL.root2)); //Check Binary SearchTree
    }
}
```

Preorder.java

```java
public class PreOrder {
    void preOrder(Node x){
        if(x != null){
            //Parent - Left Child - Right Child
            System.out.print(x.value + " "); //Parent
            preOrder(x.left); //left
            preOrder(x.right); //right
        }
        else return;
    }
}
```

Node.java

```java
public class Node{ //used for constructing the AVL Tree and BST
    int value, height;
    Node left, right;
    Node(int x){
        height = 1;
        value = x;
    }
    public int getValue(){
        return value;
    }
}
```

AVLTree.java

```java
public class AVLTree {
    //root 1 = AVL Tree | root 2 = Binary Search Tree
    Node root, root2;

    //return the height of tree
    int height(Node n){
        if(n == null) //check if the root has element or not
            return 0;
        return max(height(n.left), height(n.right)) + 1;
    }
    //return the max value of given integers
    int max(int a, int b){
        if(a > b) return a;
        else return b;
    }
```

```java
 //single right rotation algorithm
Node singleRight(Node a){
    Node b = a.left;  //set the left child
    Node t = b.right; //set the right child

    b.right = a;  //rotation process
    a.left = t;

    //max method to update the height value of new tree with respect to y and x
    a.height = max(height(a.left), height(a.right)) + 1;
    b.height = max(height(b.left), height(b.right)) + 1;

    return b; //return the updated value
}
//single left rotation algorithm
Node singleLeft(Node b){
    Node a = b.right; //set the right child
    Node t = a.left; //set the left child

    a.left = b; //rotation process
    b.right = t;

    //max method to update the height value of new tree with respect to y and x
    b.height = 1 + max(height(b.left), height(b.right));
    a.height = 1 + max(height(a.left), height(a.right));

    return a; //return the updated value
}
```

```
//inserting value to AVL Tree
Node insertion(Node n, int data){
    if(n == null) return (new Node(data)); //if theres no root, then create new AVL tree
    if(data < n.value) //if the inserted value is smaller then go left
        n.left = insertion(n.left, data); //repeat
    else if (data > n.value) //if inserted value is greater then go right
        n.right = insertion(n.right, data); //repeat
    else
        return n; //return the node

    n.height = 1 + max(height(n.left), height(n.right)); //computer the total height

    int balance = balanceFactor(n); //invoke balance factor and set to the balance variable

    //check to determine which rotation algorithm
    if(balance > 1 && data < n.left.value) //if balance factor +2 use single right rotation
        return singleRight(n); //left child is unbalanced
    if (balance < -1 && data > n.right.value) //if balance factor -2 use single left rotation
        return singleLeft(n); //right child is unbalanced
    //double rotation
    if(balance > 1 && data > n.left.value){ //if balance factor +2 and has different operator (-+)
        n.left = singleLeft(n.left); //implement double rotation
        return singleRight(n);
    }
    if(balance < -1 && data < n.right.value){ //if balance factor +2 and has different operator (-+)
        n.right = singleRight(n.right); //implement double rotation
        return singleLeft(n);
    }
    return n; //return updated AVL Tree
}
```

```
//compute the balance factor algorithm (left child & right child)
int balanceFactor(Node n){
    if(n == null) return 0; //if tree empty, return 0
    return height((n.left)) - height(n.right); //else, get the balance factor of nodes
}
//check AVL Tree or not
String checkUnbalance(Node n){
    if(balanceFactor(n) > 1) //if balance factor +2
        return "It is Unbalanced on LHS of BST";
    else if (balanceFactor(n) < -1) //if balance factor -2
        return "It is Unbalanced on RHS of BST";
    else
        return "It is Balanced BST / AVL Tree"; //if balance factor -1 < x < 1
}
//Regular binary search tree data insertion method
void addNode(Node n){
    if(root2 == null) //if tree empty, create a new root
        root2 = n; //set as root
    else
        insertNode(root2, n); //create as a child
}
```

```
void insertNode(Node parent, Node n){
    if(n.getValue() < parent.getValue()){ //if data < parent, go left
        if(parent.left == null) //if parent left null
            parent.left = n; //insert the data
        else
            insertNode(parent.left, n); //traverse to the left
    }
    else{
        if(parent.right == null) //if parent right null
            parent.right = n; //insert the data
        else
            insertNode(parent.right, n); //traverse to the right
    }
}
```