FULL STACK WEB DEVELOPMENT - WORKSHEET

1. **Write a Java program that inserts a node into its proper sorted position in a linked list.**

```
class Node {
   int data;
   Node next;

   Node(int data) {
      this.data = data;
      this.next = null;
   }
}

public class SortedLinkedList {
   Node head;

   // Function to insert a node into its proper sorted position
   void insert(int data) {
      Node newNode = new Node(data);
      if (head == null || data < head.data) {
         newNode.next = head;
         head = newNode;
      } else {
         Node current = head;
         while (current.next != null && current.next.data < data) {
            current = current.next;
         }
         newNode.next = current.next;
         current.next = newNode;
      }
```

```
    }

    // Function to display the linked list
    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }

    public static void main(String[] args) {
        SortedLinkedList list = new SortedLinkedList();
        list.insert(5);
        list.insert(10);
        list.insert(2);
        list.insert(8);

        System.out.println("Sorted Linked List:");
        list.display();
    }
}
```

2. Write a Java program to compute the height of the binary tree.

```
class Node {
   int data;
   Node left, right;

   Node(int item) {
      data = item;
      left = right = null;
   }
}

public class BinaryTreeHeight {
   Node root;

   // Function to find the height of a binary tree
   int height(Node root) {
      if (root == null) {
         return 0;
      } else {
         int leftHeight = height(root.left);
         int rightHeight = height(root.right);

         return Math.max(leftHeight, rightHeight) + 1;
      }
   }

   public static void main(String[] args) {
      BinaryTreeHeight tree = new BinaryTreeHeight();
      tree.root = new Node(1);
      tree.root.left = new Node(2);
      tree.root.right = new Node(3);
      tree.root.left.left = new Node(4);
      tree.root.left.right = new Node(5);
```

```
        System.out.println("Height of the binary tree is: " +
tree.height(tree.root));
    }
}
```

3. **Write a Java program to determine whether a given binary is BST or not.**

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class BinaryTreeIsBST {
    Node root;

    // Function to check if a binary tree is a BST
    boolean isBST(Node root, int min, int max) {
        if (root == null) {
            return true;
        }

        if (root.data <= min || root.data >= max) {
            return false;
        }

        return isBST(root.left, min, root.data) && isBST(root.right, root.data,
max);
    }
```

```
    public static void main(String[] args) {
        BinaryTreeIsBST tree = new BinaryTreeIsBST();
        tree.root = new Node(2);
        tree.root.left = new Node(1);
        tree.root.right = new Node(3);

        if (tree.isBST(tree.root, Integer.MIN_VALUE, Integer.MAX_VALUE)) {
            System.out.println("The binary tree is a BST.");
        } else {
            System.out.println("The binary tree is not a BST.");
        }
    }
}
```

4. **Check if the expression is balanced using stack.**

```
import java.util.Stack;

public class BalancedExpression {
    // Function to check if an expression is balanced
    static boolean isBalanced(String expr) {
        Stack<Character> stack = new Stack<>();

        for (char ch : expr.toCharArray()) {
            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            } else if (ch == ')' && !stack.isEmpty() && stack.peek() == '(') {
                stack.pop();
            } else if (ch == ']' && !stack.isEmpty() && stack.peek() == '[') {
```

```java
                stack.pop();
            } else if (ch == '}' && !stack.isEmpty() && stack.peek() == '{') {
                stack.pop();
            } else {
                return false; // Unmatched closing bracket
            }
        }

        return stack.isEmpty(); // If the stack is empty, expression is balanced
    }

    public static void main(String[] args) {
        String expression = "{ { [ [ ( ( ) ) ] ] } }";

        if (isBalanced(expression)) {
            System.out.println("The expression is balanced.");
        } else {
            System.out.println("The expression is not balanced.");
        }
    }
}
```

5. Write a Java program to print left view of a binary que using Que.

```
import java.util.LinkedList;
import java.util.Queue;

class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

public class LeftViewBinaryTree {
    Node root;

    // Function to print left view of a binary tree
    void leftView() {
        if (root == null)
            return;

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int n = queue.size();

            for (int i = 1; i <= n; i++) {
                Node temp = queue.poll();

                if (i == 1)
                    System.out.print(temp.data + " ");
```

```
            if (temp.left != null)
                queue.add(temp.left);

            if (temp.right != null)
                queue.add(temp.right);
        }
    }
}

public static void main(String args[]) {
    LeftViewBinaryTree tree = new LeftViewBinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Left view of binary tree is ");
    tree.leftView();
}
}
```