FULL STACK DEVELOPMENT – WORKSHEET – 6

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

Ans:

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

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

class linkedList {
    Node head;

    linkedList() {
        head = null;
    }

    void insert(int data) {
        Node newNode = new Node(data);

        if (head == null || head.data >= newNode.data) {
            newNode.next = head;
            head = newNode;
        } else {
            Node current = head;
                 while (current.next != null &&
     current.next.data < newNode.data) {
                 current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
    }

    void display() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        linkedList list = new linkedList();
        list.insert(5);
        list.insert(10);
        list.insert(2);
        list.insert(7);
        list.insert(1);

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

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

Ans:

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

   Node(int data) {
      this.data = data;
      left = null;
      right = null;
   }
}
class BinaryTree {
   Node root;
   BinaryTree() {
      root = null;
   }
   int height(Node node) {
      if (node == null) {
         return 0;
      } else {
         int leftHeight = height(node.left);
         int rightHeight = height(node.right);
         return Math.max(leftHeight, rightHeight) + 1;
      }
   }
   public static void main(String[] args) {
      BinaryTree tree = new BinaryTree();
      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);
      int treeHeight = tree.height(tree.root);
      System.out.println("Height of the binary tree is: " + treeHeight);
   }
}
```

Ques 3. Write a java program to determine whether a given binary tree is a BST or not.

Ans:

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

    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
```

```
class BinaryTree {
    Node root;

    BinaryTree() {
        root = null;
    }

    boolean isBST() {
        return isBSTUtil(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    boolean isBSTUtil(Node node, int min, int max) {
        if (node == null) {
            return true;
        }
        if (node.data < min || node.data > max) {
            return false;
        }
        return (isBSTUtil(node.left, min, node.data - 1) && isBSTUtil(node.right, node.data + 1,
max));
    }

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

        if (isBinarySearchTree) {

            System.out.println("The binary tree is a Binary Search Tree.");

        } else {

            System.out.println("The binary tree is not a Binary Search Tree.");

        }

    }

}
```

Ques 4. Write a java code to Check the given below expression is balanced or not . (using stack) { { [[(( ))]) } }

Ans:

```
public class BalancedExpression {

    static boolean isBalanced(String expr) {

        Stack<Character> stack = new Stack<>();

        for (char c : expr.toCharArray()) {

            if (c == '(' || c == '[' || c == '{') {

                stack.push(c);

            } else if (c == ')' || c == ']' || c == '}') {

                if (stack.isEmpty()) {

                    return false;

                }


                char top = stack.pop();

                if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) {

                    return false;

                }

            }

        }
        return stack.isEmpty();
    }
    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.");
        }
    }
}
```

Ques 5. Write a java program to Print left view of a binary tree using queue.

Ans:

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

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

class BinaryTree {
    Node root;

    BinaryTree() {
```

```
    root = null;
}

void leftView() {
    if (root == null)
        return;

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

    while (!queue.isEmpty()) {
        int size = queue.size();
        boolean isFirst = true;

        for (int i = 0; i < size; i++) {
            Node current = queue.poll();

            if (isFirst) {
                System.out.print(current.data + " ");
                isFirst = false;
            }

            if (current.left != null)
                queue.add(current.left);

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

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();

        // Creating a sample binary tree
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.right = new Node(4);
        tree.root.right.left = new Node(5);-
        tree.root.right.right = new Node(6);
        tree.root.right.left.left = new Node(7);

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