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;

    // Constructor to create a new node
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class SortedLinkedList {
    Node head;

    // Method to insert a new node in sorted order
    public void insert(int data) {
        Node newNode = new Node(data);

        // If the list is empty or the new node should be inserted at the head
        if (head == null || head.data >= newNode.data) {
            newNode.next = head;
            head = newNode;
```

```
        return;
    }

    // Locate the node before the point of insertion
    Node current = head;
    while (current.next != null && current.next.data < newNode.data) {
        current = current.next;
    }

    newNode.next = current.next;
    current.next = newNode;
}

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

public static void main(String[] args) {
    SortedLinkedList list = new SortedLinkedList();
```

```
        list.insert(5);
        list.insert(2);
        list.insert(8);
        list.insert(1);
        list.insert(7);

        System.out.print("Sorted Linked List: ");
        list.printList();
    }
}
```

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

Ans: // Class representing a node in the binary tree

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

    // Constructor to create a new node
    TreeNode(int data) {
        this.data = data;
        left = right = null;
    }
}
```

```
// Class representing the binary tree
class BinaryTree {
    TreeNode root;

    // Method to compute the height of the binary tree
    int computeHeight(TreeNode node) {
        if (node == null) {
            return -1; // If you want to count height as the number of edges,
return -1
            // return 0; // If you want to count height as the number of nodes,
return 0
        }

        // Compute the height of each subtree
        int leftHeight = computeHeight(node.left);
        int rightHeight = computeHeight(node.right);

        // Return the larger height between the two subtrees plus 1 for the
current node
        return Math.max(leftHeight, rightHeight) + 1;
    }

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

        // Creating a binary tree
```

```java
        tree.root = new TreeNode(1);
        tree.root.left = new TreeNode(2);
        tree.root.right = new TreeNode(3);
        tree.root.left.left = new TreeNode(4);
        tree.root.left.right = new TreeNode(5);

        // Compute the height of the tree
        int height = tree.computeHeight(tree.root);
        System.out.println("Height of the binary tree: " + height);
    }
}
```

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

Ans: // Class representing a node in the binary tree

```java
class TreeNode {
    int data;
    TreeNode left, right;

    // Constructor to create a new node
    TreeNode(int data) {
        this.data = data;
        left = right = null;
```

```
    }
}

// Class representing the binary tree
class BinaryTree {
    TreeNode root;

    // Helper method to check if the tree is a BST
    boolean isBST(TreeNode node, Integer min, Integer max) {
        // An empty tree is a BST
        if (node == null) {
            return true;
        }

        // Check the current node's value against the min and max constraints
        if (min != null && node.data <= min) {
            return false;
        }
        if (max != null && node.data >= max) {
            return false;
        }

        // Recursively check the left and right subtrees with updated
constraints
        return isBST(node.left, min, node.data) &&
```

```
        isBST(node.right, node.data, max);
}

// Public method to start the BST check
boolean isBST() {
    return isBST(root, null, null);
}

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

    // Creating a binary tree
    tree.root = new TreeNode(4);
    tree.root.left = new TreeNode(2);
    tree.root.right = new TreeNode(5);
    tree.root.left.left = new TreeNode(1);
    tree.root.left.right = new TreeNode(3);

    // Check if the binary tree is a BST
    if (tree.isBST()) {
        System.out.println("The binary tree is a BST.");
    } else {
        System.out.println("The binary tree is not a BST.");
    }
}
```

}

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

{ { [[(( ))]) } }

Ans: import java.util.Stack;

```
public class BalancedExpression {
    // Method to check if the given expression is balanced
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();

        // Traverse the expression
        for (int i = 0; i < expression.length(); i++) {
            char ch = expression.charAt(i);

            // If the character is an opening bracket, push it onto the stack
            if (ch == '{' || ch == '[' || ch == '(') {
                stack.push(ch);
            }
            // If the character is a closing bracket
            else if (ch == '}' || ch == ']' || ch == ')') {
                // Check if the stack is empty, which means there's no matching
opening bracket
```

```java
            if (stack.isEmpty()) {
                return false;
            }

            // Pop the top of the stack and check if it matches the closing
bracket
            char top = stack.pop();
            if (!isMatchingPair(top, ch)) {
                return false;
            }
        }
    }

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

// Helper method to check if the brackets are a matching pair
private static boolean isMatchingPair(char opening, char closing) {
    return (opening == '{' && closing == '}') ||
           (opening == '[' && closing == ']') ||
           (opening == '(' && closing == ')');
}

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

// Class representing a node in the binary tree
class TreeNode {
    int data;
    TreeNode left, right;

    // Constructor to create a new node
    TreeNode(int data) {
        this.data = data;
```

```
        left = right = null;
    }
}

// Class representing the binary tree
class BinaryTree {
    TreeNode root;

    // Method to print the left view of the binary tree
    void printLeftView() {
        if (root == null) {
            return;
        }

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

        while (!queue.isEmpty()) {
            // Number of nodes at the current level
            int levelSize = queue.size();

            // Traverse all nodes of the current level
            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();
```

```java
                // Print the leftmost element at the level (i.e., the first element of
the level)
                if (i == 0) {
                    System.out.print(currentNode.data + " ");
                }

                // Add left child to the queue
                if (currentNode.left != null) {
                    queue.add(currentNode.left);
                }

                // Add right child to the queue
                if (currentNode.right != null) {
                    queue.add(currentNode.right);
                }
            }
        }
    }

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

        // Creating a binary tree
        tree.root = new TreeNode(1);
        tree.root.left = new TreeNode(2);
```

```
        tree.root.right = new TreeNode(3);
        tree.root.left.left = new TreeNode(4);
        tree.root.left.right = new TreeNode(5);
        tree.root.right.left = new TreeNode(6);
        tree.root.left.left.left = new TreeNode(7);

        // Print the left view of the binary tree
        System.out.print("Left view of the binary tree: ");
        tree.printLeftView();
    }
}
```