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 1.
```
class Node {
    int data;
    Node next;

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

class SortedLinkedList {
    Node head;

    public SortedLinkedList() {
        this.head = null;
    }

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

        if (head == null || head.data >= 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;
    }
  }

  public void display() {
    Node current = head;
    while (current != null) {
      System.out.print(current.data + " ");
      current = current.next;
    }
    System.out.println();
  }
}

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

    list.insert(5);
    list.insert(10);
    list.insert(3);
    list.insert(8);

    list.display(); // Output: 3 5 8 10
  }
}
```

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

Ans 2.

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

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

class BinaryTree {
    Node root;

    public BinaryTree() {
        this.root = null;
    }

    public int getHeight(Node node) {
        if (node == null) {
            return 0;
        } else {
            int leftHeight = getHeight(node.left);
            int rightHeight = getHeight(node.right);

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

```
        }
    }

    public int getHeight() {
        return getHeight(root);
    }
}

public class Main {
    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 height = tree.getHeight();
        System.out.println("Height of the binary tree is: " + height); // Output: Height of the binary tree is: 3
    }
}
```


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

Ans 3.

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

    public Node(int data) {
        this.data = data;
```

```
        this.left = null;
        this.right = null;
    }
}

class BinaryTree {
    Node root;

    public BinaryTree() {
        this.root = null;
    }

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

    private boolean isBSTUtil(Node node, int minValue, int maxValue) {
        if (node == null) {
            return true;
        }

        if (node.data < minValue || node.data > maxValue) {
            return false;
        }

        return isBSTUtil(node.left, minValue, node.data - 1)
                && isBSTUtil(node.right, node.data + 1, maxValue);
    }
}
```

```
public class Main {
    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 isBST = tree.isBST();
        System.out.println("Is the binary tree a BST? " + isBST); // Output: Is the binary tree a BST? true
    }
}
```

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

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

Ans 4.

```
import java.util.Stack;

public class Main {
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();

        for (char ch : expression.toCharArray()) {
            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            } else if (ch == ')' || ch == ']' || ch == '}') {
                if (stack.isEmpty()) {
                    return false;
```

```
            }

            char top = stack.pop();
            if ((ch == ')' && top != '(') || (ch == ']' && top != '[') || (ch == '}' && top != '{')) {
                return false;
            }
          }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        String expression = "{ { [ [ ( ( ) ) ] ) } }";
        boolean isBalanced = isBalanced(expression);
        System.out.println("Is the expression balanced? " + isBalanced); // Output: Is the expression
balanced? true
    }
}
```

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

Ans 5.

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

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

    public Node(int data) {
```

```
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

class BinaryTree {
    Node root;

    public BinaryTree() {
        this.root = null;
    }

    public void printLeftView() {
        if (root == null) {
            return;
        }

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

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                Node node = queue.poll();
                if (i == 0) {
                    System.out.print(node.data + " ");
                }
```

```
            if (node.left != null) {
                queue.add(node.left);
            }
            if (node.right != null) {
                queue.add(node.right);
            }
        }
    }
  }
}

public class Main {
    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);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        System.out.println("Left view of the binary tree:");
        tree.printLeftView(); // Output: Left view of the binary tree: 1 2 4
    }
}
```