WORKSHEET-6
FULLSTACK DEVELOPMENT

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

A.
```
class Node {
   int data;
   Node next;
   Node(int data) {
      this.data = data;
      this.next = null;
   }
}
class SortedLinkedList {
   Node head;
   public void insert(int data) {
      Node newNode = new Node(data);
 if (head == null) {
         head = newNode;
         return;
      }
 if (data < head.data) {
         newNode.next = head;
         head = newNode;
         return;
      }
Node current = head;
      while (current.next != null && current.next.data < data) {
         current = current.next;
      }
newNode.next = current.next;
      current.next = newNode;
   }
   public void printList() {
      Node current = head;
      while (current != null) {
         System.out.print(current.data + " -> ");
         current = current.next;
      }
      System.out.println("null");
   }
   public static void main(String[] args) {
      SortedLinkedList list = new SortedLinkedList();
      list.insert(5);
      list.insert(3);
      list.insert(8);
      list.insert(2);
      list.insert(7);
      list.printList();  // Output should be: 2 -> 3 -> 5 -> 7 -> 8 -> null
   }
}
```

Q2**. Write a java program to compute the height of the binary tree.**

A.

```
class Node {
    int value;
    Node left;
    Node right;
    public Node(int value) {
        this.value = value;
        left = null;
        right = null;
    }
}
class BinaryTree {
    Node root;
    public int height() {
        return height(root);
    }
    private 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 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.height();
        System.out.println("Height of the binary tree: " + height);
    }
}
```

Q3. **Write a java program to determine whether a given binary tree is a BST or not.**

```
A. class Node {
   int value;
   Node left;
   Node right;
   public Node(int value) {
      this.value = value;
      left = null;
      right = null;
   }
}
class BinaryTree {
   Node root;
   public int height() {
      return height(root);
   }
   private 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 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.height();
      System.out.println("Height of the binary tree: " + height);
   }
}
```

Q4. **Write a java code to Check the given below expression is balanced or not . (using stack)**
{{[[(())]]}}

```
A. import java.util.Stack;
public class BalanceChecker {
   public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char c : expression.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 = "{ { [ [ ( ( ) ) ] ] } }";
        boolean isBalanced = isBalanced(expression);
        System.out.println("Is the expression balanced? " + isBalanced);
    }
}
```

Q5. **Write a java program to Print left view of a binary tree using queue.**

```
A.   import java.util.Queue;
import java.util.LinkedList;
class Node {
    int value;
    Node left;
    Node right;
    public Node(int value) {
        this.value = value;
        left = null;
        right = null;
    }
}
public class LeftViewPrinter {
    public static void printLeftView(Node root) {
        if (root == null) {
            return;
        }
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            System.out.print(queue.peek().value + " ");
            for (int i = 0; i < levelSize; i++) {
                Node node = queue.poll();
                if (node.left != null) {
                    queue.add(node.left);
                }
                if (node.right != null) {
                    queue.add(node.right);
                }
            }
        }
    }
    public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Left view of the binary tree: ");
        printLeftView(root);
    }
}
```