FULL STACK DEVELOPMENT – WORKSHEET A

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

Ans1-

```
// Code for adding a Node in a sorted Linked list at its Sorted place

        class Node
        {
          int data;         //data that we will store in the Nodes
          Node next;     //next is a pointer that will be used to point at next Node


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


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


class Main
{
   // function to print a given linked list
   public static void printList(Node head)
   {
      Node ptr = head;     //ptr refers to pointer,setting the initial value of ptr to head for traversingLL
      while (ptr != null)    //traversing the LL until the next node has value null
      {
         System.out.print(ptr.data + " —> ");
         ptr = ptr.next;
```

```
        }

        System.out.println("null");
    }
    /*Function to insert a given node at its correct sorted position into a given list sorted in increasing order*/
    public static Node sortedInsert(Node head, Node newNode)
    {
        // special cases
        //special case 1: If the Linked List has 0 nodes or head = null, thus new node will be our head
       /*special case 2: If data stored in head node is greater or equals to new node, so the correct order of the new node will be before head, thus new node will be the new head*/
        if (head == null || head.data >= newNode.data)
        {
            newNode.next = head;
            head = newNode;
            return head;
        }

        // locate the node before the point of insertion
        Node current = head;
        while (current.next != null && current.next.data < newNode.data) {
            current = current.next;   //traversing the LL until we find the correct position of new node
        }

        newNode.next = current.next;   //placing the new node at its position
        current.next = newNode;

        return head;
    }

    public static void main(String[] args)
```

```
  {
    // input keys
    int[] keys = {2, 4, 6, 8};

    // points to the head node of the linked list
    Node head = null;

    // construct a linked list
    for (int i = keys.length - 1; i >= 0; i--) {
      head = new Node(keys[i], head);
    }

    head = sortedInsert(head, new Node(5));
    head = sortedInsert(head, new Node(9));
    head = sortedInsert(head, new Node(1));

    // print linked list
    printList(head);
  }
}
```

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

Ans2-
```
class Node {
   int data;
   Node left, right;
   public Node(int item) {
      data = item;   //data in a node
      left = right = null;  //initializing the values of nodes
   }
}

class BinaryTree {
   Node root;
   public BinaryTree() {
      root = null;
   }

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

         if (leftHeight > rightHeight)
            return (leftHeight + 1);
         else
            return (rightHeight + 1);
      }
   }

   public static void main(String[] args) {
```

```
        BinaryTree tree = new BinaryTree();

        // Construct the binary tree
        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 tree is : " + tree.height(tree.root));
    }
}
```

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

Ans3- // Java program to check if a given tree is BST.

```
class BST{


   static int prev = Integer.MIN_VALUE;
   /* A binary tree node has data, pointer to
   left child and a pointer to right child */
   static class Node {
      int data;
      Node left, right;


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


   // Utility function to check if Binary Tree is BST
   static boolean isBSTUtil(Node root)
   {
      // traverse the tree in inorder fashion and
      // keep track of prev node
      if (root != null) {
         if (!isBSTUtil(root.left))
            return false;


         // Allows only distinct valued nodes
```

```
        if (root.data <= prev)
            return false;

        // Initialize prev to current
        prev = root.data;

        return isBSTUtil(root.right);
    }

    return true;
}

// Function to check if Binary Tree is BST
static boolean isBST(Node root)
{
    return isBSTUtil(root);
}

public static void main(String[] args)
{
    Node root = new Node(5);
    root.left = new Node(2);
    root.right = new Node(15);
    root.left.left = new Node(1);
    root.left.right = new Node(4);

    if (isBST(root))
```

```
            System.out.print("Is BST");
        else
            System.out.print("Not a BST");
    }
}
```

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

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

Ans4-

```
import java.util.Stack;


public class BalancedExpressionChecker {


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


        for (int i = 0; i < expression.length(); i++) {
            char currentChar = expression.charAt(i);


            if (currentChar == '{' || currentChar == '[' || currentChar == '(') {
                stack.push(currentChar);
            } else if (currentChar == '}' || currentChar == ']' || currentChar == ')') {
                if (stack.isEmpty()) {
                    return false;
                }
                char topChar = stack.pop();
                if ((currentChar == '}' && topChar != '{') ||
                        (currentChar == ']' && topChar != '[') ||
                        (currentChar == ')' && topChar != '(')) {
                    return false;
                }
            }
        }


        return stack.isEmpty();
```

```
    }
    public static void main(String[] args) {
        String expression = "{{[[(())]]}}";
        boolean isBalanced = checkIfBalanced(expression);
        if (isBalanced) {
            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.

Ans5-

```
import java.util.*;
class Node {
   int data;
   Node left, right;


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


class BinaryTree {
   Node root;


   // Print the left view of the binary tree using a queue
   void leftView() {
      if (root == null)
         return;


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


      while (!queue.isEmpty()) {
         // Get the size of the current level
         int levelSize = queue.size();
```

```
        // Print the leftmost node of the current level
        System.out.print(queue.peek().data + " ");

        // Add the children of the current level to the queue
        for (int i = 0; i < levelSize; i++) {
            Node temp = queue.poll();
            if (temp.left != null)
                queue.add(temp.left);
            if (temp.right != null)
                queue.add(temp.right);
        }
    }
}

// Driver code
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.right.left = new Node(5);
    tree.root.right.right = new Node(6);

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

}