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.

Ans

```
 // A Linked List Node
class Node
{
   int data;
   Node next;

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

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

class Main
{
   // Helper function to print a given linked list
   public static void printList(Node head)
   {
      Node ptr = head;
      while (ptr != 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 case for the head end
      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;
        }

        newNode.next = current.next;
        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);
    }
}
```

Output:-

```
1 —> 2 —> 4 —> 5 —> 6 —> 8 —> 9 —> null
```

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

Ans

```
    public class BinaryTree {

      //Represent the node of binary tree
      public static class Node{
        int data;
        Node left;
        Node right;

        public Node(int data){
  //Assign data to the new node, set left and right children to null
          this.data = data;
          this.left = null;
          this.right = null;
        }
      }

      //Represent the root of binary tree
      public Node root;
      public BinaryTree(){
        root = null;
      }

      //findHeight() will determine the maximum height of the binary tree
      public int findHeight(Node temp){
        //Check whether tree is empty
        if(root == null) {
```

```
     System.out.println("Tree is empty");
    return 0;
  }
  else {
    int leftHeight = 0, rightHeight = 0;

    //Calculate the height of left subtree
    if(temp.left != null)
      leftHeight = findHeight(temp.left);

    //Calculate the height of right subtree
    if(temp.right != null)
      rightHeight = findHeight(temp.right);

    //Compare height of left subtree and right subtree
    //and store maximum of two in variable max
    int max = (leftHeight > rightHeight) ? leftHeight : rightHeight;

    //Calculate the total height of tree by adding height of root
    return (max + 1);
  }
 }

public static void main(String[] args) {

  BinaryTree bt = new BinaryTree();
  //Add nodes to the binary tree
  bt.root = new Node(1);
  bt.root.left = new Node(2);
  bt.root.right = new Node(3);
  bt.root.left.left = new Node(4);
  bt.root.right.left = new Node(5);
  bt.root.right.right = new Node(6);
  bt.root.right.right.right= new Node(7);
  bt.root.right.right.right.right = new Node(8);

  //Display the maximum height of the given binary tree
```

```
        System.out.println("Maximum height of given binary tree: " + bt.findHeight(bt.root));
    }
```

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

Ans

```
    // Java program to check if a given tree is BST.
import java.io.*;

class GFG {
  /* A binary tree node has data, pointer to
  left child and a pointer to right child */
  public static class Node {
    public int data;
    public Node left, right;

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

  static Node prev;

  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 (prev != null && root.data <= prev.data)
        return false;

      prev = root;

      return isBSTUtil(root.right);
    }
    return true;
  }
```

```
    static Boolean isBST(Node root)
    {
        return isBSTUtil(root);
    }

    // Driver Code
    public static void main(String[] args)
    {
        Node root = new Node(3);
        root.left = new Node(2);
        root.right = new Node(5);
        root.left.left = new Node(1);
        root.left.right = new Node(4);

        // Function call
        if (isBST(root))
            System.out.println("Is BST");
        else
            System.out.println("Not a BST");
    }
}
```

Output

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.*;
public class Test {

    public static int check(String str)
    {
        Stack<Character> s = new Stack();
        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (c == '(') {
                s.push('(');
            }
            else if (c == ')') {
                if (s.isEmpty()) {
                    return 0;
                }
                else {
                    char p = s.peek();
                    if (p == '(') {
                        s.pop();
                    }
                    else {
                        return 0;
                    }
                }
            }
        }
        if (s.empty()) {
            return 1;
        }
        else {
            return 0;
        }
    }

    public static void main(String[] args)
    {
        String str = "()(())()";
        if (check(str) == 0) {
            System.out.println("Invalid");
        }
        else {
            System.out.println("Valid");
        }
    }
}
```

Output

```
Valid
```

Time complexity: O(N)
Auxiliary Space: O(1)

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

Ans

```
// Java Program to print the left view
import java.util.*;

class GFG {
```

```
// Binary Tree Node
static class Node {
  int data;
  Node left, right;

  public Node(int item)
  {
    data = item;
    left = right = null;
  }
};
// function to print the left view of binary tree
public static ArrayList<Integer> leftView(Node root)
{
  // Your code here
  ArrayList<Integer> ans = new ArrayList<>();

  if (root == null) {
    return ans;
  }

  Queue<Node> q = new LinkedList<>();
  q.add(root);
  q.add(null);
  boolean ok = true;

  while (!q.isEmpty()) {
    Node it = q.poll();
    if (it == null) {
      if (ok == false) {
        ok = true;
      }

      if (q.size() == 0)
        break;

      else {
        q.add(null);
      }
    }
    else {

      if (ok) {
        ans.add(it.data);
        ok = false;
      }

      if (it.left != null) {
```

```
                    q.add(it.left);
                }

                if (it.right != null) {
                    q.add(it.right);
                }
            }
        }

        return ans;
    }
    // driver code
    public static void main(String[] args)
    {
        Node root = new Node(10);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(7);
        root.left.right = new Node(8);
        root.right.right = new Node(15);
        root.right.left = new Node(12);
        root.right.right.left = new Node(14);

        ArrayList<Integer> vec = leftView(root);
        for (int x : vec) {
            System.out.print(x + " ");
        }
        System.out.println();
    }
}
```

Output

```
10 2 7 14
```

Time Complexity: O(N) where N is the total number of nodes.
Auxiliary Space: O(N) due to the space occupied by queue.