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.

```
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);
    }
}
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.

```
import java.util.Scanner;
class BTree
{
        public static void main(String arg[])
        {
        Scanner sc=new Scanner(System.in);
        System.out.println("enter number of nodes in a binary tree :");
        int nodes=sc.nextInt();
        int c=0;
        int n=nodes;
        while(nodes!=1)
        {
        nodes=nodes/2;
        c++;
        }
           System.out.println("height of a binary tree :"+c);
          System.out.println("height of a binary tree in worst case :"+(n-1));


        }
```

```
}
```

Output: enter number of nodes in a binary tree :10

height of a binary tree : 2

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

```
// Java implementation for the above approach
import java.io.*;
class GFG {
static class node {
	int data;
	node left, right;
}
/* Helper function that allocates a new node with the given data and NULL left and right
pointers. */
static node newNode(int data)
{
	node Node = new node();
	Node.data = data;
	Node.left = Node.right = null;

	return Node;
}
static int maxValue(node Node)
{
	if (Node == null) {
	return Integer.MIN_VALUE;
	}
	int value = Node.data;
	int leftMax = maxValue(Node.left);
	int rightMax = maxValue(Node.right);
	return Math.max(value, Math.max(leftMax, rightMax));
}
static int minValue(node Node)
{
```

```
        if (Node == null) {
        return Integer.MAX_VALUE;
        }
        int value = Node.data;
        int leftMax = minValue(Node.left);
        int rightMax = minValue(Node.right);
        return Math.min(value, Math.min(leftMax, rightMax));
}
/* Returns true if a binary tree is a binary search tree*/
static int isBST(node Node)
{
        if (Node == null) {
        return 1;
        }
        /* false if the max of the left is > than us */
        if (Node.left != null
                && maxValue(Node.left) > Node.data) {
        return 0;
        }



        /* false if the min of the right is <= than us */
        if (Node.right != null && minValue(Node.right) < Node.data) {
        return 0;
        }
        /* false if, recursively, the left or right is not a * BST*/
        if (isBST(Node.left) != 1 || isBST(Node.right) != 1) {
        return 0;
        }
        /* passing all that, it's a BST */
        return 1;
}
public static void main(String[] args)
{
        node root = newNode(4);
```

```
        root.left = newNode(2);

        root.right = newNode(5);


        // root->right->left = newNode(7);

        root.left.left = newNode(1);

        root.left.right = newNode(3);


        // Function call

        if (isBST(root) == 1) {

        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)

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

```
import java.util.Stack;

public class BalancedParentheses {

  public static void main (String args []) {

    System.out.println (balancedParentheses ("{ { [ [ ( ( ) ) ] ) } }"));

  }

  public static boolean balancedParentheses (String s) {

    Stack<Character> stack = new Stack<Character> ();

    for (int i = 0; i < s.length (); i++) {

      char c = s.charAt (i);

      if (c == ' [' || c == ' (' || c == ' {' ) {
```

```
        stack.push (c);
      } else if (c == ']') {
        if (stack.isEmpty () || stack.pop () != ' [') {
          return false;
        }
      } else if (c == ')') {
        if (stack.isEmpty () || stack.pop () != ' (') {
          return false;
        }
      } else if (c == '}') {
        if (stack.isEmpty () || stack.pop () != ' {') {
          return false;
        }
      }
    }
    return stack.isEmpty ();
  }
}
```

Output : false

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

```
/* Class containing left and right child of current node and key value*/
class Node {
         int data;
         Node left, right;
         public Node(int item)
         {        data = item;
                  left = right = null;
         }
```

```
}

/* Class to print the left view */

class BinaryTree {

        Node root;

        static int max_level = 0;

        // recursive function to print left view

        void leftViewUtil(Node node, int level)

        {       // Base Case

                if (node == null)

                        return;

                // If this is the first node of its level

                if (max_level < level) {

                        System.out.print(node.data + " ");

                        max_level = level;

                }

                // Recur for left and right subtrees

                leftViewUtil(node.left, level + 1);

                leftViewUtil(node.right, level + 1);

        }

        // A wrapper over leftViewUtil()

        void leftView(){

                max_level = 0;

                leftViewUtil(root, 1);

        }

        /* testing for example nodes */

        public static void main(String args[])

        {       /* creating a binary tree and entering the nodes */

                BinaryTree tree = new BinaryTree();
```

```java
            tree.root = new Node(10);

            tree.root.left = new Node(2);

            tree.root.right = new Node(3);

            tree.root.left.left = new Node(7);

            tree.root.left.right = new Node(8);

            tree.root.right.right = new Node(15);

            tree.root.right.left = new Node(12);

            tree.root.right.right.left = new Node(14);

            tree.leftView();

    }}
```

Output : 10 2 7 14