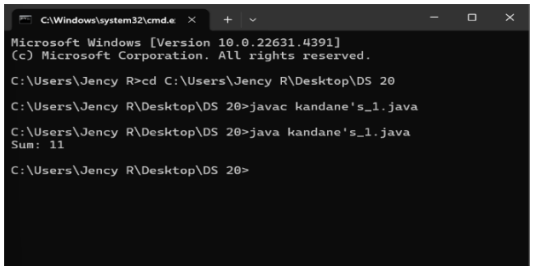
## Coding Question Practice test- 01

### 1. Maximum Subarray Sum – Kadane‟s Algorithm:

```
import java.util.*;
class MaxArray{
    public static int maxSubarraySum(int[] arr) {
      int result = arr[0];
      int maxEnding = arr[0];
      for(int i=0;i<arr.length;i++) {
        maxEnding = Math.max(maxEnding + arr[i], arr[i]);
        result = Math.max(result, maxEnding);
      }
       return result;
    }

    public static void main(String []args) {
            int[] arr = {-2, -4};
            System.out.println("Sum: " + maxSubarraySum(arr));
    }
}
```

OUTPUT:

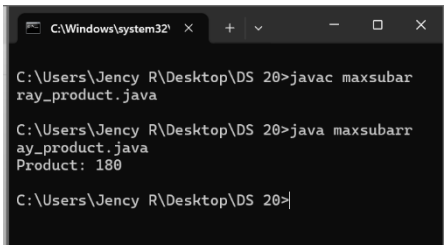


Time Complexity : O(n)

2. **Maximum Product Subarray**

```
import java.util.*;
class MaxArray{
        public static int maxSubarrayProduct(int[] arr) {
          int result = arr[0];
          int maxEnding = arr[0];
          int minEnding = arr[0];
          for(int i=1;i<arr.length;i++) {
            int temp = maxEnding;
            maxEnding = Math.max(Math.max(arr[i], maxEnding * arr[i]),
minEnding * arr[i]);
          minEnding = Math.min(Math.min(arr[i], temp * arr[i]),
minEnding * arr[i]);
            result = Math.max(result, maxEnding);
          }
        return result;
        }

        public static void main(String []args) {
                int[] arr =  {-1, -3, -2, -5};
                System.out.println("Product: " + maxSubarrayProduct(arr));
        }
}
```

OUTPUT:



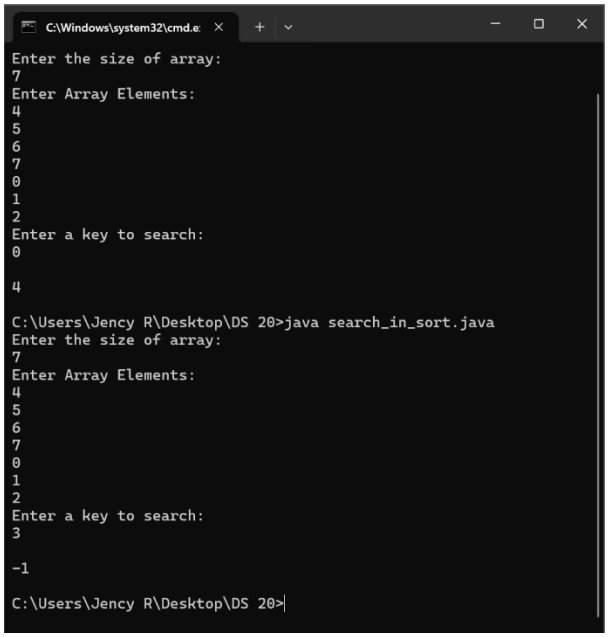

Time Complexity : O(n)

**3. Search in a sorted and rotated Array**

```
import java.util.*;
class SortedArray {
    public static void main(String []args){
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the size of array: ");
        int n = s.nextInt();
        System.out.println("Enter Array Elements: ");
        int[] arr = new int[n];
        for(int i=0;i<n;i++){
            arr[i] = s.nextInt();
        }
        System.out.println("Enter a key to search: ");
        int key = s.nextInt();
        int i = 0;
        int index = -1;
        for(int num : arr){
            if(num==key){
                index = i;
                break;
            }
            i++;
        }
    System.out.println(" ");
    System.out.println(index);
```

```
    }
}
```

OUTPUT:



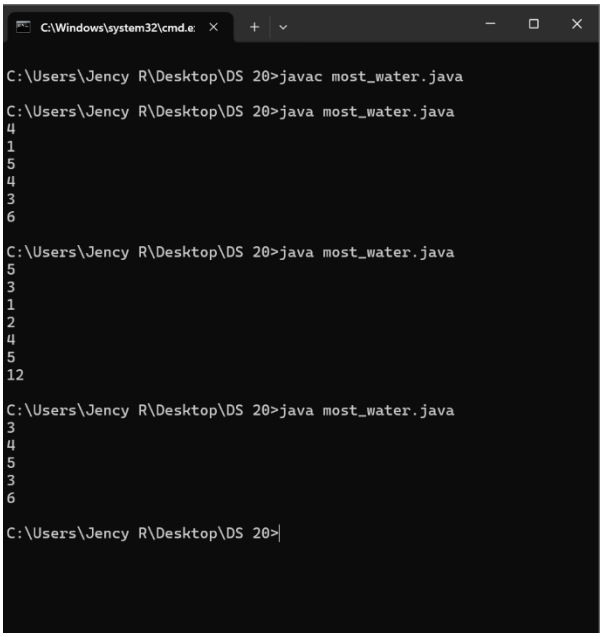

Time Complexity: O(n)

**4. Container with Most Water**

```
import java.util.*;
class MostWater{
        public static void main(String[]args){
                Scanner s = new Scanner(System.in);
                int n = s.nextInt();
                int[] arr = new int[n];
                for(int i=0; i<n; i++){
                        arr[i] = s.nextInt();
                }
                int max = 0;
                int area = 0;
```

```
            for(int i=0; i<n-1; i++){
                for(int j=i+1; j<n;j++){
                    int d = Math.abs(i-j);
                    int h = Math.min(arr[i],arr[j]);
                    area = h * d;
                    if(max<area){
                        max = area;

                    }
                }
            }
            System.out.println(max);
        }
}
```

OUTPUT:

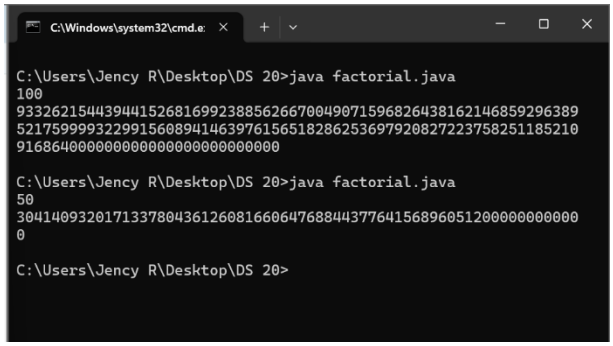



Time Complexity: O(n^2)

## 5. Find the Factorial of a large number

```
import java.util.*;
import java.math.BigInteger;
class Factorial{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        BigInteger res = BigInteger.ONE;
        int n = s.nextInt();
        for(int i=2;i<=n;i++){
            res = res.multiply(BigInteger.valueOf(i));
        }
        System.out.println(res);
    }
}
```

OUTPUT:



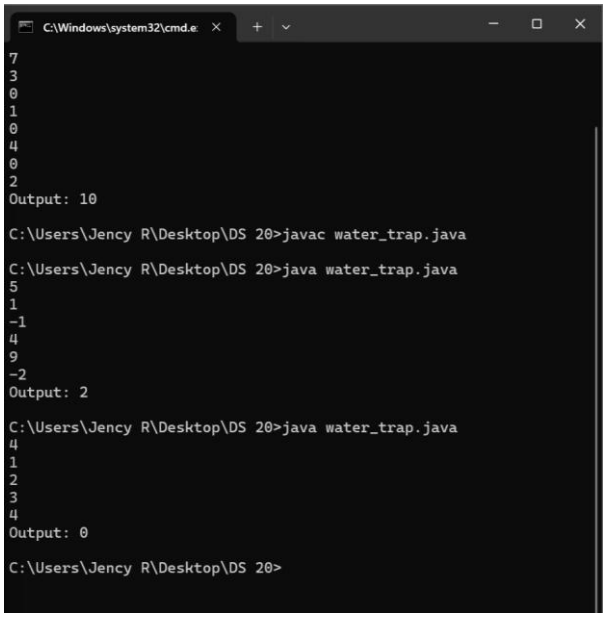

Time Complexity: O(n)

## 6. Trapping Rainwater Problem

```
import java.util.*;
```

```
class ChocolateDistribution{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();
        int m = s.nextInt();
        int[] arr = new int[n];
        int[] subarr = new int[m];
        for(int i=0; i<n ; i++){
            arr[i] = s.nextInt();
        }

        arr.sort();
        for(int i = 0;i<m-1;i++){
            subarr.append(arr[i]);
        }

        int lar = Math.max(subarr);
        int sma = Math.min(subarr);
        int diff = lar - sma;

        System.out.println(diff);
    }
}
```

OUTPUT:



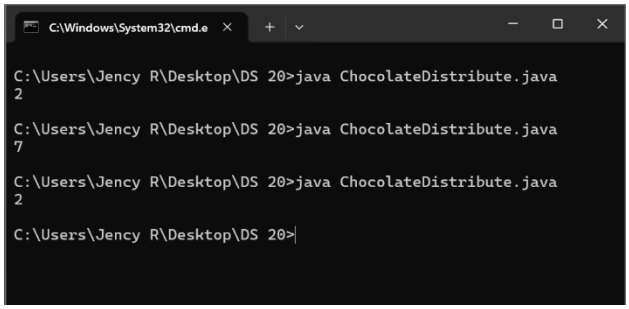

Time Complexity: O(n)

**7. Chocolate Distribution Problem**

```
import java.util.*;

public class ChocolateDistribution {
    public static int findMinDiff(int[] arr, int n, int m) {
        if (m == 0 || n == 0) {
            return 0;
        }

        if (n < m) {
            return -1;
        }
        Arrays.sort(arr);
```

```
        int mdiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            mdiff = Math.min(mdiff, diff);
        }

        return mdiff;
    }
    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        int n = arr.length;
        int result = findMinDiff(arr, n, m);
        System.out.println(result);
    }
}
```

OUTPUT:




Time Complexity: O(n logn)

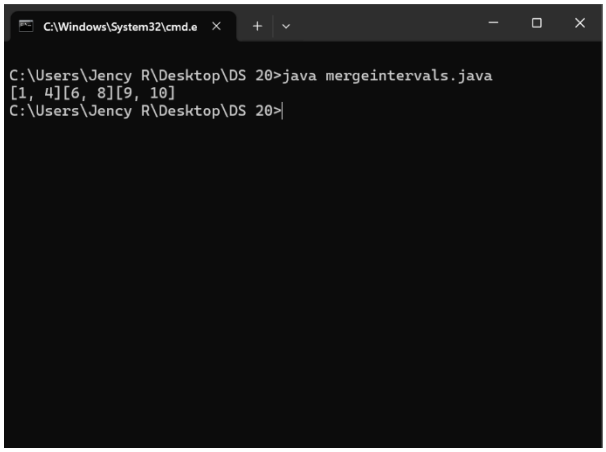
**8. Merge Overlapping Intervals**

```
import java.util.*;
```

```
class Solution {
    public List<List<Integer>> merge(int[][] intervals) {
        if (intervals.length == 0) {
            return new ArrayList<>();
        }
        List<List<Integer>> m = new ArrayList<>();
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        int[] current = intervals[0];
        for (int i = 1; i < intervals.length; i++) {
            int[] interval = intervals[i];
            if (current[1] >= interval[0]) {
                current[1] = Math.max(current[1], interval[1]);
            } else {
                m.add(Arrays.asList(current[0], current[1]));
                current = interval;
            }
        }
        m.add(Arrays.asList(current[0], current[1]));
        return m;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[][] intervals = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        List<List<Integer>> result = solution.merge(intervals);
```

```
        for (List<Integer> interval : result) {
            System.out.print(interval);
        }
    }
}
```

OUTPUT:




Time Complexity: O(n logn)

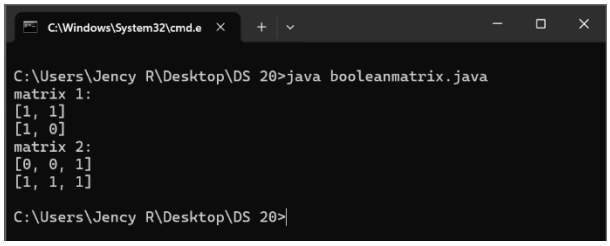
**9. Boolean Matrix Question**

```
import java.util.Arrays;
class Solution {
    public static void modifyMatrix(int[][] mat) {
        int M = mat.length;
        int N = mat[0].length;
        boolean[] row = new boolean[M];
        boolean[] col = new boolean[N];
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
```

```
                if (mat[i][j] == 1) {
                        row[i] = true;
                        col[j] = true;
                }
                }
        }

        for (int i = 0; i < M; i++) {
                for (int j = 0; j < N; j++) {
                if (row[i] || col[j]) {
                        mat[i][j] = 1;
                }
                }
        }
}

public static void main(String[] args) {
        int[][] mat1 = { {1, 0}, {0, 0} };
        int[][] mat2 = { {0, 0, 0}, {0, 0, 1} };

        modifyMatrix(mat1);
        modifyMatrix(mat2);

        System.out.println("matrix 1:");
        for (int[] row : mat1) {
                System.out.println(Arrays.toString(row));
```

```
            }

            System.out.println("matrix 2:");
            for (int[] row : mat2) {
                System.out.println(Arrays.toString(row));
            }
        }
    }
```

OUTPUT:




Time Complexity: O(mXn)

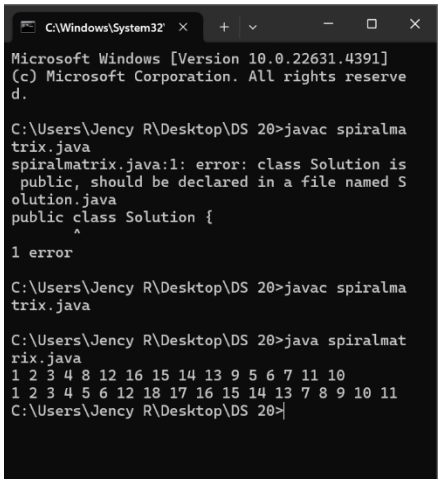
**10. Print a given matrix in spiral form**

```
class Solution {

    public static void printSpiral(int[][] m) {
        if (m == null || m.length == 0 || m[0].length == 0) {
            return;
        }

        int top = 0, bottom = m.length - 1;
        int left = 0, right = m[0].length - 1;

        while (top <= bottom && left <= right) {
```

```
        for (int i = left; i <= right; i++) {
            System.out.print(m[top][i] + " ");
        }
        top++;

        for (int i = top; i <= bottom; i++) {
            System.out.print(m[i][right] + " ");
        }
        right--;

        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                System.out.print(m[bottom][i] + " ");
            }
            bottom--;
        }

        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                System.out.print(m[i][left] + " ");
            }
            left++;
        }
    }
}
```

```
    public static void main(String[] args) {
        int[][] matrix1 = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16}
        };

        int[][] matrix2 = {
            {1, 2, 3, 4, 5, 6},
            {7, 8, 9, 10, 11, 12},
            {13, 14, 15, 16, 17, 18}
        };



        printSpiral(matrix1);

        System.out.println("");
        printSpiral(matrix2);
    }
}
```

OUTPUT:



```
C:\Windows\System32
Microsoft Windows [Version 10.0.22631.4391]
(c) Microsoft Corporation. All rights reserve
d.

C:\Users\Jency R\Desktop\DS 20>javac spiralma
trix.java
spiralmatrix.java:1: error: class Solution is
 public, should be declared in a file named S
olution.java
public class Solution {
       ^
1 error

C:\Users\Jency R\Desktop\DS 20>javac spiralma
trix.java

C:\Users\Jency R\Desktop\DS 20>java spiralmat
rix.java
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
C:\Users\Jency R\Desktop\DS 20>
```

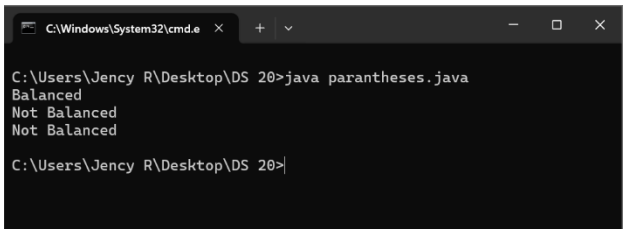
Time Complexity: O(mXn)

**11.Check if given Parentheses expression is balanced or not**

```
import java.util.Stack;


class Solution {
    public static String Balance(String str) {
        Stack<Character> st = new Stack<>();


        for (char ch : str.toCharArray()) {
            if (ch == '(') {
                st.push(ch);
            } else if (ch == ')') {
                if (st.isEmpty()) {
                    return "Not Balanced";
                }
                st.pop();
            }
```

```
        }

        return st.isEmpty() ? "Balanced" : "Not Balanced";
    }

    public static void main(String[] args) {
        String str1 = "((()))()()";
        String str2 = "())((())";
      String str3 = "(((((";

        System.out.println(Balance(str1));
        System.out.println(Balance(str2));
      System.out.println(Balance(str2));
    }
}
```

OUTPUT:




Time Complexity : O(n)

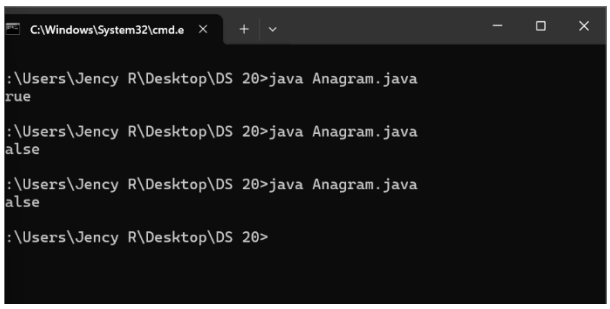
**12.Check if two Strings are Anagrams of each other**

```
import java.util.*;

class Solution {
```

```
public static boolean Anagram(String str1,String str2) {

 if(str1.length() != str2.length()){
        return false;
 }

 char[] a1 = str1.toCharArray();
 char[] a2 = str2.toCharArray();

 Arrays.sort(a1);
 Arrays.sort(a2);

 return Arrays.equals(a1, a2);
}

public static void main(String [] args) {
 String s1 = "geeks";
 String s2 = "keesg";

 if(Anagram(s1,s2)) {
        System.out.println("true");
 }
 else{
        System.out.println("false");
 }
}
```

}

OUTPUT:




Time Complexity: O(n logn)

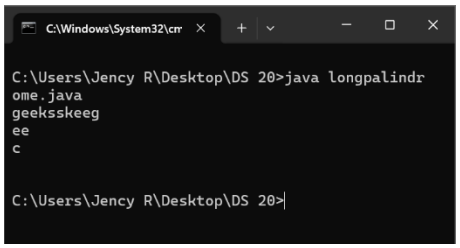
**13.Longest Palindromic Substring**

```
public class Solution {
    public static String longPalindrome(String str) {
        if (str == null || str.length() < 1) {
            return "";
        }

        int start = 0, end = 0;

        for (int i = 0; i < str.length(); i++) {
            int len1 = expandAroundCenter(str, i, i);
            int len2 = expandAroundCenter(str, i, i + 1);

            int len = Math.max(len1, len2);

            if (len > (end - start)) {
                start = i - (len - 1) / 2;
```

```
            end = i + len / 2;
        }
    }

    return str.substring(start, end + 1);
}

private static int expandAroundCenter(String str, int left, int right) {
    while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
        left--;
        right++;
    }
    return right - left - 1;
}

public static void main(String[] args) {
    String str1 = "forgeeksskeegfor";
    String str2 = "Geeks";
    String str3 = "abc";
    String str4 = "";

    System.out.println(longPalindrome(str1));
    System.out.println(longPalindrome(str2));
    System.out.println(longPalindrome(str3));
    System.out.println(longPalindrome(str4));
}
```

}

OUTPUT:



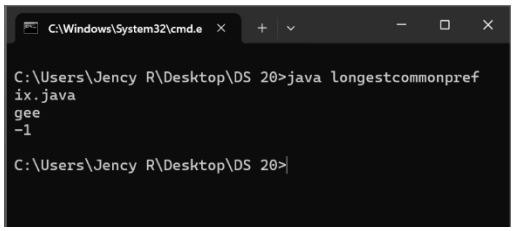

Time Complexity : O(n^2)

**14. Longest Common Prefix using Sorting**

```
 public class Solution {
public static String longestCommonPrefix(String[] arr) {
    if (arr == null || arr.length == 0) {
        return "-1";
    }

    String prefix = arr[0];

    for (int i = 1; i < arr.length; i++) {
        while (arr[i].indexOf(prefix) != 0) {
            prefix = prefix.substring(0, prefix.length() - 1);
            if (prefix.isEmpty()) {
                return "-1";
            }
        }
    }
```

```
        return prefix;
    }

    public static void main(String[] args) {
        String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
        String[] arr2 = {"hello", "world"};

        System.out.println(longestCommonPrefix(arr1));
        System.out.println(longestCommonPrefix(arr2));
    }
  }
```

OUTPUT:




Time Complexity: O(n logn + m)

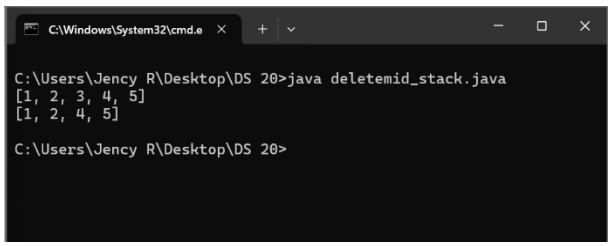
**15.Delete middle element of a stack**

```
import java.util.Stack;

class Solution {
  public void deleteMiddle(Stack<Integer> s) {
    int m = s.size() / 2;
    delete(s, m);
  }
```

```
private void delete(Stack<Integer> s, int m) {
    if (m == 0) {
        s.pop();
        return;
    }
    int top = s.pop();
    delete(s, m - 1);
    s.push(top);
}

public static void main(String[] args) {
    Stack<Integer> stack = new Stack<>();
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.push(5);

    System.out.println(stack);

    Solution solution = new Solution();
    solution.deleteMiddle(stack);

    System.out.println(stack);
}
```

}

OUTPUT :



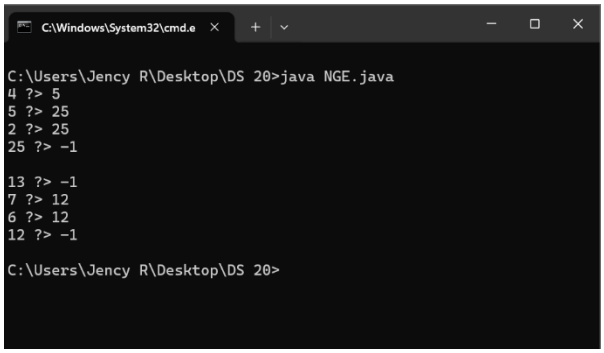

Time Complexity : O(n)

**16.Next Greater Element (NGE) for every element in given Array**

```
import java.util.Stack;


public class NextGreaterElement {
    public void printNextGreaterElements(int[] a) {
        Stack<Integer> stack = new Stack<>();
        int[] nge = new int[a.length];

        for (int i = a.length - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= a[i]) {
                stack.pop();
            }
            nge[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(a[i]);
        }

        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i] + " –> " + nge[i]);
        }
    }
```

```
    public static void main(String[] args) {
        NextGreaterElement nge = new NextGreaterElement();
        int[] arr1 = {4, 5, 2, 25};
        int[] arr2 = {13, 7, 6, 12};

        nge.printNextGreaterElements(arr1);
          System.out.println(" ");

        nge.printNextGreaterElements(arr2);
    }
}
```

OUTPUT:




Time Complexity: O(n)

**17.Print Right View of a Binary Tree**

```
import java.util.ArrayList;

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

```
    Node(int x) {
        data = x;
        left = right = null;
    }
}

public class BinaryTree {
    static void RecursiveRightView(Node root, int level,
                int[] maxLevel, ArrayList<Integer> result) {
        if (root == null) return;

        if (level > maxLevel[0]) {
            result.add(root.data);
            maxLevel[0] = level;
        }

        RecursiveRightView(root.right, level + 1, maxLevel, result);
        RecursiveRightView(root.left, level + 1, maxLevel, result);
    }

    static ArrayList<Integer> rightView(Node root) {
        ArrayList<Integer> result = new ArrayList<>();
        int[] maxLevel = new int[] {-1};
        RecursiveRightView(root, 0, maxLevel, result);
        return result;
    }
```

```
    static void printArray(ArrayList<Integer> arr) {
        for (int val : arr) {
            System.out.print(val + " ");
        }
        System.out.println();
    }

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

        ArrayList<Integer> result = rightView(root);
        printArray(result);
    }
}
```

OUTPUT:

```
Output
java -cp /tmp/ZnVzAlb8ys/BinaryTree
Maximum Depth of the Binary Tree: 4

=== Code Execution Successful ===
```

Time Complexity: O(n)

**18. Maximum Depth or Height of Binary Tree**

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

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

public class BinaryTree {

   public int maxDepth(Node root) {
      if (root == null) {
         return 0;
      }

      int leftDepth = maxDepth(root.left);
      int rightDepth = maxDepth(root.right);

      return Math.max(leftDepth, rightDepth) + 1;
   }

   public static void main(String[] args) {
      BinaryTree tree = new BinaryTree();
```

```
        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.left.left.left = new Node(6);

        int depth = tree.maxDepth(root);
        System.out.println(depth);
    }
}
```

OUTPUT:

```
java -cp /tmp/tJYLTLsvMX/BinaryTree
4

=== Code Execution Successful ===
```

Time Complexity: O(n)