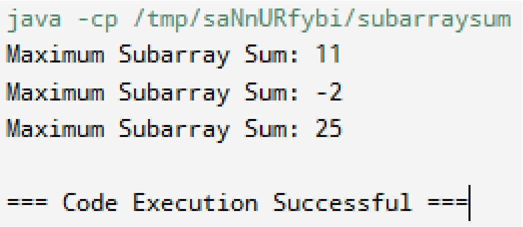
1. Maximum Subarray Sum – Kadane‟s Algorithm:

code:

```
package mypackage;

public class subarraysum {
    public int findMax(int[] arr) {
        int maxSoFar = arr[0];
        int maxEndingHere = arr[0];

        for (int i = 1; i < arr.length; i++) {
            maxEndingHere = Math.max(arr[i], maxEndingHere + arr[i]);
            maxSoFar = Math.max(maxSoFar, maxEndingHere);
        }

        System.out.println("Maximum Subarray Sum: " + maxSoFar);
        return maxSoFar;
    }

    public static void main(String[] args) {
        int[] arr1 = {2, 3, -8, 7, -1, 2, 3};
        int[] arr2 = {-2, -4};
        int[] arr3 = {5, 4, 1, 7, 8};

        subarraysum obj = new subarraysum();
        obj.findMax(arr1);
        obj.findMax(arr2);
        obj.findMax(arr3);
    }
}
```

output:



```
java -cp /tmp/saNnURfybi/subarraysum
Maximum Subarray Sum: 11
Maximum Subarray Sum: -2
Maximum Subarray Sum: 25

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

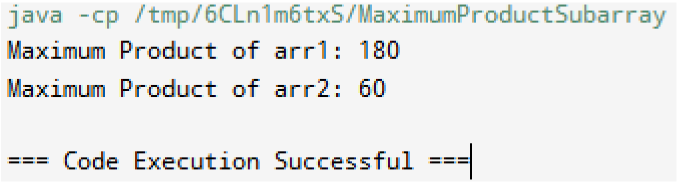
Time Complexity: O(n).

2. Maximum Product Subarray:

```
public class MaximumProductSubarray {
      public int maxProduct(int[] arr) {
            if (arr.length == 0) return 0;

            int maxProduct = arr[0];
```

```
                int minProduct = arr[0];
                int result = arr[0];

                for (int i = 1; i < arr.length; i++) {
                    int tempMax = maxProduct;

                    maxProduct = Math.max(arr[i], Math.max(maxProduct * arr[i], minProduct *
arr[i]));
                    minProduct = Math.min(arr[i], Math.min(tempMax * arr[i], minProduct *
arr[i]));
                    result = Math.max(result, maxProduct);
                }
                return result;
            }
        public static void main(String[] args) {
            int[] arr1 = {-2, 6, -3, -10, 0, 2};
            int[] arr2 = {-1, -3, -10, 0, 60};

            MaximumProductSubarray obj = new MaximumProductSubarray();
            System.out.println("Maximum Product of arr1: " + obj.maxProduct(arr1));
            System.out.println("Maximum Product of arr2: " + obj.maxProduct(arr2));
        }
    }
```

output:



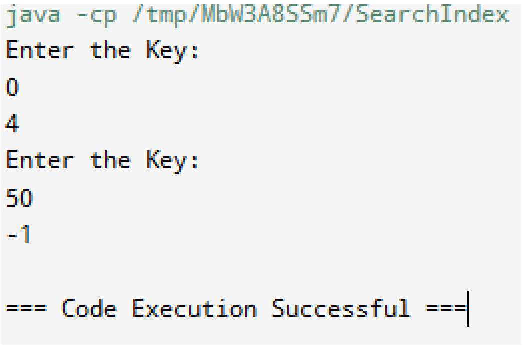
```
java -cp /tmp/6CLn1m6txS/MaximumProductSubarray
Maximum Product of arr1: 180
Maximum Product of arr2: 60

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

Time Complexity: O(n).

3.Search in a sorted and rotated Array:

```
import java.util.Scanner;

public class SearchIndex {
    public int Index(int[] arr) {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the Key:");
            int key = sc.nextInt();
            for(int i=0;i<arr.length;i++) {
                    if(arr[i]==key) {
                            return i;
                    }
            }
            return -1;
    }
    public static void main(String[] args) {
            int[] arr1 = {-2, 6, -3, -10, 0, 2};
            int[] arr2 = {-1, -3, -10, 0, 60};
             SearchIndex obj = new SearchIndex();
             System.out.println(obj.Index(arr1));
             System.out.println(obj.Index(arr2));

    }

}
```

Output:



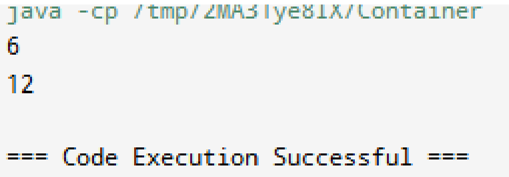
```
java -cp /tmp/MbW3A8SSm7/SearchIndex
Enter the Key:
0
4
Enter the Key:
50
-1

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

Time complexity: o(n).

4.Container with Most Water.

```
package mypackage;

public class Container {

   public int container(int[] arr){
      int left=0,right=arr.length-1,maxx=0;
      while(left<right){
         int width=right-left;
         int height=Math.min(arr[left],arr[right]);
         maxx=Math.max(maxx,width*height);
         if(arr[left]<arr[right]) left++;
         else right--;
      }
      return maxx;
   }

   public int suboptimal_container(int[] arr){
      int left=0,right=arr.length-1,maxx=-1;
      for(int i=0;i<arr.length/2;i++){
         left=0;
         while(left<right){
                maxx=Math.max(maxx,(right-left)*(Math.min(arr[left],arr[right])));
                left++;
            }right--;
        }return maxx;
    }


    public static void main(String[] args) {
        Container obj = new Container();
        int[] arr1={1, 5, 4, 3};
        int[] arr2 = {3, 1, 2, 4, 5};

        System.out.println(obj.container(arr1));
        System.out.println(obj.container(arr2));

    }
}
```

Output:



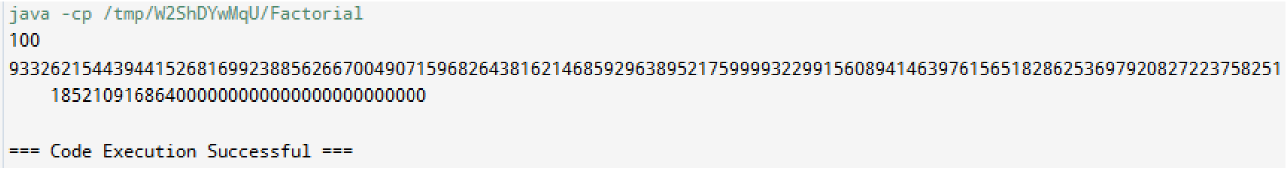
```
java -cp /tmp/ZMA3Tye81X/Container
6
12

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

Time Complexity: O(n^2).

5.Factorial:

```
import java.math.BigInteger;
import java.util.Scanner;

public class Factorial {
    public BigInteger Fact(int num) {
        BigInteger result = BigInteger.ONE;
        for (int i = 1; i <= num; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Factorial obj = new Factorial();
        System.out.println(obj.Fact(n));
    }
}
```

Output:



```
java -cp /tmp/W2ShDYwMqU/Factorial
100
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251
    185210916864000000000000000000000000

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

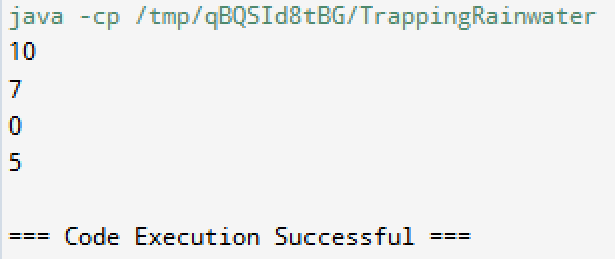
Time Complexity: O(n).

6.Trapping Rainwater Problem.

```
public class TrappingRainwater {
    public int trap(int[] height) {
        int n = height.length;
        int[] leftmax = new int[n];
        leftmax[0] = height[0];
        for (int i = 1; i < n; i++) {
            leftmax[i] = Math.max(leftmax[i - 1], height[i]);
        }

        int[] rightmax = new int[n];
        rightmax[n - 1] = height[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            rightmax[i] = Math.max(height[i], rightmax[i + 1]);
        }

        int trappedwater = 0;
        for (int i = 0; i < n; i++) {
            int waterlevel = Math.min(leftmax[i], rightmax[i]);
            trappedwater += waterlevel - height[i];
        }

        return trappedwater;
    }

    public static void main(String[] args) {
        TrappingRainwater solution = new TrappingRainwater();

        int[] height1 = {3, 0, 1, 0, 4, 0, 2};
        System.out.println(solution.trap(height1));

        int[] height2 = {3, 0, 2, 0, 4};
        System.out.println(solution.trap(height2));

        int[] height3 = {1, 2, 3, 4};
        System.out.println(solution.trap(height3));
```

```
            int[] height4 = {10, 9, 0, 5};
            System.out.println(solution.trap(height4));



        }
    }
```

Output:



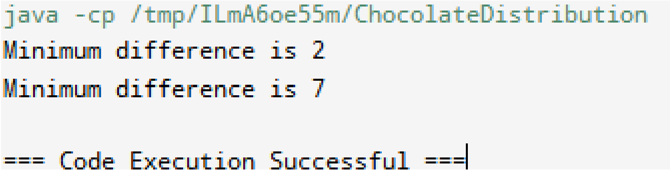
```
java -cp /tmp/qBQSId8tBG/TrappingRainwater
10
7
0
5

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

Time Complexity: O(n).

7. Chocolate Distribution Problem:

```
import java.util.Arrays;

public class ChocolateDistribution {

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

        Arrays.sort(arr);

        if (n < m) {
            return -1;
        }

        int minDifference = Integer.MAX_VALUE;

        for (int i = 0; i + m - 1 < n; i++) {
            int difference = arr[i + m - 1] - arr[i];
            minDifference = Math.min(minDifference, difference);
        }

        return minDifference;
    }

    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        int n = arr.length;

        System.out.println("Minimum difference is " + findMinDifference(arr, n, m));

        m = 5;
        System.out.println("Minimum difference is " + findMinDifference(arr, n, m));
    }
}
```

Output:



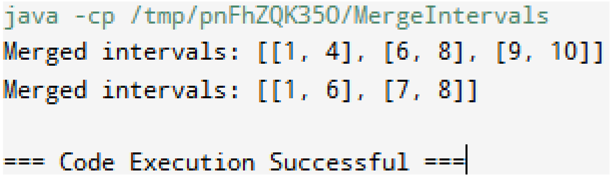
```
java -cp /tmp/ILmA6oe55m/ChocolateDistribution
Minimum difference is 2
Minimum difference is 7

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

Time Complexity:O(nlog(n)).

8. Merge Overlapping Intervals:

```
 import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MergeIntervals {

    public static int[][] merge(int[][] intervals) {
        if (intervals.length <= 1) {
            return intervals;
        }

        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

        List<int[]> merged = new ArrayList<>();
        int[] currentInterval = intervals[0];
        merged.add(currentInterval);

        for (int[] interval : intervals) {
            int currentEnd = currentInterval[1];
            int nextStart = interval[0];
            int nextEnd = interval[1];

            if (currentEnd >= nextStart) {
                currentInterval[1] = Math.max(currentEnd, nextEnd);
            } else {
                currentInterval = interval;
                merged.add(currentInterval);
            }
        }

        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {
        int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        System.out.println("Merged intervals: " + Arrays.deepToString(merge(intervals1)));

        int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
        System.out.println("Merged intervals: " + Arrays.deepToString(merge(intervals2)));
    }
}
```

Output:



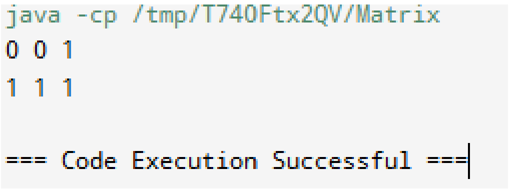
```
java -cp /tmp/pnFhZQK35O/MergeIntervals
Merged intervals: [[1, 4], [6, 8], [9, 10]]
Merged intervals: [[1, 6], [7, 8]]

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

Time Complexity:O(nlog(n))

9. A Boolean Matrix Question:

```
public class Matrix {
    public static void main(String[] args) {
        int[][] mat = {{0, 0, 0},
                    {0, 0, 1}};
        int[][] updat = new int[mat.length][mat[0].length];
        int n = mat.length;
        int m = mat[0].length;

        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                if (mat[i][j] == 1) {
                    for (int x = 0; x < m; x++) {
                        updat[i][x] = 1;
                    }
                    for (int x = 0; x < n; x++) {
                        updat[x][j] = 1;
                    }
                }
            }
        }
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                System.out.print(updat[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:



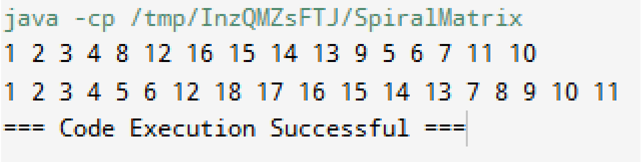
```
java -cp /tmp/T740Ftx2QV/Matrix
0 0 1
1 1 1

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

Time Complexity: O(n * m)

10. Print a given matrix in spiral form:

```
public class SpiralMatrix {
    public static void printSpiral(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;

        int top = 0, bottom = m - 1, left = 0, right = n - 1;

        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                System.out.print(matrix[top][i] + " ");
            }
            top++;

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

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

            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    System.out.print(matrix[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}
    };
    printSpiral(matrix1);

    System.out.println();

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

Output:



```
java -cp /tmp/InzQMZsFTJ/SpiralMatrix
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
=== Code Execution Successful ===
```

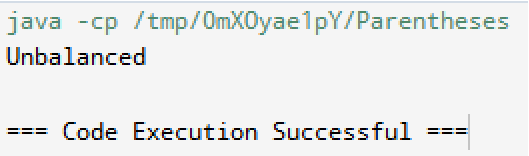
Time Complexity: O(m*n)

13.Check if given Parentheses expression is balanced or not:

```
import java.util.*;

public class Parentheses {

    public static void main(String[] args) {
//      String str1 = "((()))()()";
        String str1 = "())((())";
        Stack <Character>stack = new Stack<Character>();
        for(int i=0;i<str1.length();i++) {

            if(str1.charAt(i)==')' && stack.isEmpty()) {
                System.out.println("Unbalanced");
                return;
```

```
                        }
                        else if(str1.charAt(i)==')' && !stack.isEmpty()) {
                                stack.pop();
                        }
                        else if(str1.charAt(i)=='(') {
                                stack.push(str1.charAt(i));
                        }
                }
                if(stack.isEmpty()) {
                        System.out.println("Balanced");
                }else {
                        System.out.println("Unbalanced");
                }
        }

}
```

Output:



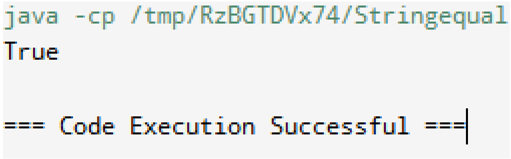
```
java -cp /tmp/OmXOyae1pY/Parentheses
Unbalanced

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

Time Complexity: O(n).

14.Check if two Strings are Anagrams of each other:

```
import java.util.Arrays;
public class Stringequal {

    public static void main(String[] args) {
            String s1 = "greeks";
            String s2 = "keegrs";
            char[] ch1 = s1.toCharArray();
            Arrays.sort(ch1);
            s1=new String(ch1);
            char[] ch2= s2.toCharArray();
            Arrays.sort(ch2);
            s2=new String(ch2);
            if(s1.equals(s2)) {
                    System.out.println("True");
            }
            else {
                    System.out.println("False");
            }
    }

}
```

Output:



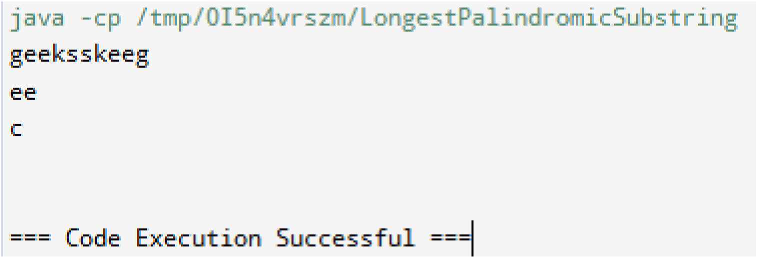
```
java -cp /tmp/RzBGTDVx74/Stringequal
True

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

Time Complexity: O(nlogn)

15. Longest Palindromic Substring:

```
public class LongestPalindromicSubstring {

    public static String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return "";

        int start = 0, end = 0;

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

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

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

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

    private static int expandAroundCenter(String s, int left, int right) {
        int L = left, R = right;

        while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {
            L--;
            R++;
        }

        return R - L - 1;
    }

    public static void main(String[] args) {
        System.out.println(longestPalindrome("forgeeksskeegfor"));
        System.out.println(longestPalindrome("Geeks"));
        System.out.println(longestPalindrome("abc"));
        System.out.println(longestPalindrome(""));
    }
}
```

Output:



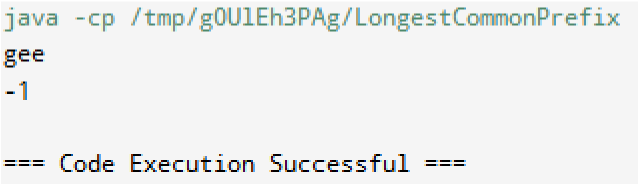
```
java -cp /tmp/0I5n4vrszm/LongestPalindromicSubstring
geeksskeeg
ee
c


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

Time Complexity: O(n^2).

16. Longest Common Prefix using Sorting:

```
import java.util.Arrays;

public class LongestCommonPrefix {

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

        Arrays.sort(arr);

        String first = arr[0];
        String last = arr[arr.length - 1];

        int i = 0;
        while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {
            i++;
        }

        if (i == 0) return "-1";

        return first.substring(0, i);
    }

    public static void main(String[] args) {
        System.out.println(longestCommonPrefix(new String[] {"geeksforgeeks", "geeks", "geek", "geezer"}));
        System.out.println(longestCommonPrefix(new String[] {"hello", "world"}));
    }
}
```

Output:



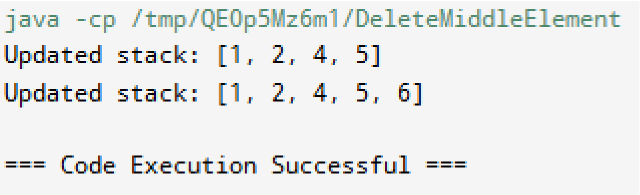
```
java -cp /tmp/g0UlEh3PAg/LongestCommonPrefix
gee
-1

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

Time Complexity:O(nlog(n)).

17. Delete middle element of a stack:

```
import java.util.Stack;

public class DeleteMiddleElement {

    public static void deleteMiddle(Stack<Integer> stack, int size, int current) {
        if (stack.isEmpty() || current == size) {
            return;
        }

        int top = stack.pop();
        deleteMiddle(stack, size, current + 1);

        if (current != size / 2) {
            stack.push(top);
        }
    }

    public static void deleteMiddleElement(Stack<Integer> stack) {
        int size = stack.size();
        if (size == 0) return;

        deleteMiddle(stack, size, 0);
    }

    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        stack1.push(1);
        stack1.push(2);
        stack1.push(3);
        stack1.push(4);
        stack1.push(5);

        deleteMiddleElement(stack1);

        System.out.println("Updated stack: " + stack1);  // Output: [1, 2, 4, 5]

        Stack<Integer> stack2 = new Stack<>();
        stack2.push(1);
        stack2.push(2);
        stack2.push(3);
        stack2.push(4);
        stack2.push(5);
        stack2.push(6);

        deleteMiddleElement(stack2);

        System.out.println("Updated stack: " + stack2);  // Output: [1, 2, 4, 5, 6]
    }
}
```

Output:



```
java -cp /tmp/QEOp5Mz6m1/DeleteMiddleElement
Updated stack: [1, 2, 4, 5]
Updated stack: [1, 2, 4, 5, 6]

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

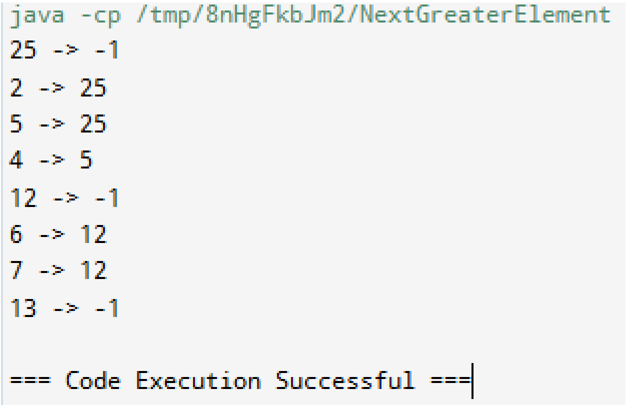
Time Complexity: O(n).

18. Next Greater Element (NGE) for every element in given Array:

```
import java.util.Stack;

public class NextGreaterElement {

  public static void printNextGreaterElement(int[] arr) {
    Stack<Integer> stack = new Stack<>();

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

      if (stack.isEmpty()) {
        System.out.println(arr[i] + " -> -1");
      } else {
        System.out.println(arr[i] + " -> " + stack.peek());
      }

      stack.push(arr[i]);
    }
  }
```

```
    public static void main(String[] args) {
        int[] arr1 = {4, 5, 2, 25};
        printNextGreaterElement(arr1);

        int[] arr2 = {13, 7, 6, 12};
        printNextGreaterElement(arr2);
    }
}
```

Output:



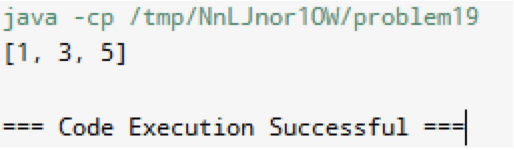
```
java -cp /tmp/8nHgFkbJm2/NextGreaterElement
25 -> -1
2 -> 25
5 -> 25
4 -> 5
12 -> -1
6 -> 12
7 -> 12
13 -> -1

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

Time Complexity:O(n)

19. Print Right View of a Binary Tree

```
import java.util.*;
class Node{
    int val;
    Node right;
    Node left;
    public Node(int val){
        this.val = val;
        right = left = null;
    }
}
public class problem19{
    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> arr = problem19.cal(root);
        System.out.println(arr);

    }
    public static ArrayList<Integer> cal(Node root){
        ArrayList<Node>arr = new ArrayList<>();
        arr.add(root);
        ArrayList<Integer>ans = new ArrayList<>();
        while(arr.size()!=0){
            int size = arr.size();
            for(int i = 0;i<size;i++){
                Node curr = arr.remove(0);
                if(i == size-1){
                    ans.add(curr.val);
                }
                if(curr.left!=null){
                    arr.add(curr.left);
                }
                if(curr.right!=null){
                    arr.add(curr.right);
                }
            }
        }
        return ans;
    }
}
```

Output:



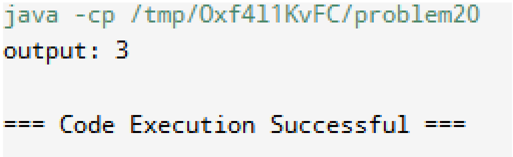
```
java -cp /tmp/NnLJnor1OW/problem19
[1, 3, 5]

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

Time Complexity: O(n).

20. Maximum Depth or Height of Binary Tree:

```
import java.util.*;
class Node{
    int val;
    Node right;
    Node left;
    public Node(int val){
        this.val = val;
        right = left = null;
    }
}
public class problem20{
    static int maxi = 0;
    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);
        problem20.cal(root,0);
        System.out.print("output: ");
        System.out.println(problem20.maxi);

    }
    public static void cal(Node root,int c){
        if(root == null){
            problem20.maxi = Math.max(c,problem20.maxi);
            return;
        }
        else{
            cal(root.left,c+1);
            cal(root.right,c+1);
        }
    }
}
```

Output:



```
java -cp /tmp/0xf411KvFC/problem20
output: 3

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

Time Complexity: O(n).