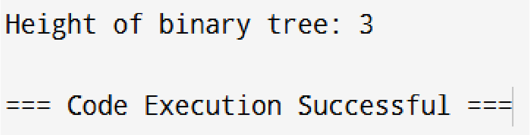
DAY 1 DSA PRACTICE PROBLEMS
NAME: HEMALALITHA M
DEPARTMENT: IT
REGISTER NO: 22IT036

20. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

CODE:

```
import java.util.LinkedList;
import java.util.Queue;
class HelloWorld {
   public static void main(String[] args) {
      Node root=new Node(12);
      root.left=new Node(8);
      root.right=new Node(18);
      root.left.left=new Node(5);
      root.left.right=new Node(11);
      Solution solution=new Solution();
      int height=solution.maxDepth(root);
      System.out.println("Height of binary tree: "+height);
   }
}
class Node{
   int data;
   Node left;
   Node right;
   Node(int val){
      data=val;
      left=null;
      right=null;
   }
}
class Solution{
   int maxDepth(Node root){
      if(root==null){
         return 0;
      }
      Queue<Node> q=new LinkedList<>();
      int level=0;
      q.add(root);
      while(!q.isEmpty()){
         int size=q.size();
         for(int i=0;i<size;i++){
            Node f=q.poll();
            if(f.left!=null){
               q.add(f.left);
            }
            if(f.right!=null){
                    q.add(f.right);
                }
            }
            level++;
        }
        return level;
    }
}
```

OUTPUT:



```
Height of binary tree: 3

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

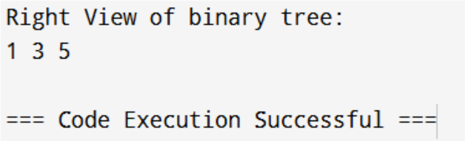
Time complexity: O(n)

19. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

CODE:

```
import java.util.List;
import java.util.ArrayList;
class HelloWorld {
    public static void main(String[] args) {
        TreeNode root=new TreeNode(1);
        root.left=new TreeNode(2);
        root.right=new TreeNode(3);
        root.right.left=new TreeNode(4);
        root.right.right=new TreeNode(5);
        Solution solution=new Solution();
        List<Integer> rightview=solution.rightsideview(root);
        System.out.println("Right View of binary tree: ");
        for(int node:rightview){
            System.out.print(node+" ");
        }
        System.out.println();
    }
}
class TreeNode{
    int data;
    TreeNode left;
    TreeNode right;
    TreeNode(int val){
        data=val;
        left=null;
        right=null;
    }
}
class Solution{
    public List<Integer> rightsideview(TreeNode root){
        List<Integer> res=new ArrayList<Integer>();
        rightview(root,res,0);
        return res;
    }
    public void rightview(TreeNode curr,List<Integer> res,int currdepth){
        if(curr==null){
            return;
        }
        if(currdepth==res.size()){
            res.add(curr.data);
        }
        rightview(curr.right,res,currdepth+1);
        rightview(curr.left,res,currdepth+1);
    }
}
```

OUTPUT:



```
Right View of binary tree:
1 3 5

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

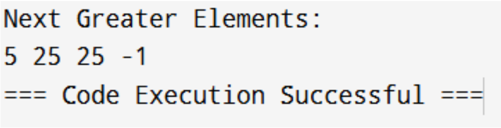
Time Complexity: O(n)

18. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element.

CODE:

```
import java.io.*;
import java.util.*;
class HelloWorld {
    public static void main(String[] args) {
        int arr[]={4,5,2,25};
        int arr2[]=nextgreater(arr);
        System.out.println("Next Greater Elements: ");
        for(int i=0;i<arr2.length;i++){
            System.out.print(arr2[i]+" ");
        }
    }
    public static int[] nextgreater(int[] arr){
        int n=arr.length;
        int nge[]=new int[n];
        Stack<Integer> st=new Stack<>();
        for(int i=n-1;i>=0;i--){
            while(!st.isEmpty() && st.peek()<=arr[i]){
                st.pop();
            }
            if(i<n){
                if(!st.isEmpty()){
                    nge[i]=st.peek();
                }else{
                    nge[i]=-1;
                }
            }
            st.push(arr[i]);
        }
        return nge;
    }
}
```

OUTPUT:



```
Next Greater Elements:
5 25 25 -1
=== Code Execution Successful ===
```

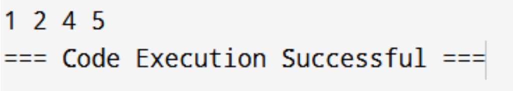
Time Complexity: O(n)

17. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

CODE:

```
import java.io.*;
import java.util.*;
class HelloWorld {
    public static void main(String[] args) {
        Stack<Character> st=new Stack<Character>();
        st.push('5');
        st.push('4');
        st.push('3');
        st.push('2');
        st.push('1');
        delmid(st,st.size(),0);
        while(!st.isEmpty()){
            char p=st.pop();
            System.out.print(p+" ");
        }
    }
    public static void delmid(Stack<Character> st, int n, int curr){
        if(st.empty() || curr==n) return;
        char x=st.pop();
        delmid(st,n,curr+1);
        if(curr!=n/2) st.push(x);
    }
}
```

OUTPUT:



```
1 2 4 5
=== Code Execution Successful ===
```

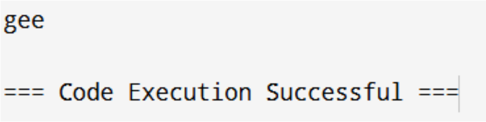
Time Complexity: O(n)

16. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

CODE:

```
class HelloWorld {
    public static String longcompre(String[] st){
        if(st==null || st.length==0) return "";
        String p=st[0];
        for(int i=1;i<st.length;i++){
            while(st[i].indexOf(p)!=0){
            p=p.substring(0,p.length()-1);
            if(p.isEmpty()) return "";
            }
        }
        return p;
    }
    public static void main(String[] args) {
        String arr[]={"geeksforgeeks","geeks","geek","geezer"};
        String st=longcompre(arr);
        System.out.println(st);
    }
}
```

OUTPUT:



```
gee

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

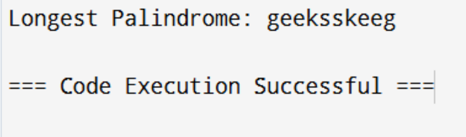
Time Complexity: O(n)

15. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

CODE:

```
import java.util.*;
class HelloWorld {
    public static String longestpalindrome(String st){
        if(st.length()<=1) return st;
        String xst=st.substring(0,1);
        for(int i=0;i<st.length()-1;i++){
            String odd=expand(st,i,i);
            String even=expand(st,i,i+1);
            if(odd.length()>xst.length()) xst=odd;
            if(even.length()>xst.length()) xst=even;
        }
        return xst;
    }
    public static String expand(String st, int left, int right){
        while(left>=0 && right<st.length() && st.charAt(left)==st.charAt(right)){
            left--;
            right++;
        }
        return st.substring(left+1,right);
    }
    public static void main(String[] args) {
        String st="forgeeksskeegfor";
        System.out.println("Longest Palindrome: "+longestpalindrome(st));
    }
}
```

OUTPUT:



```
Longest Palindrome: geeksskeeg

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

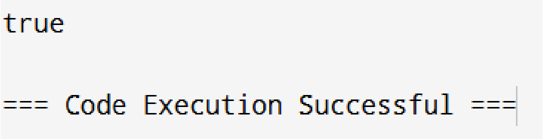
Time Complexity: O(n^2)

14. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

CODE:

```
import java.util.*;
class HelloWorld {
  public static boolean anagram(String s1,String s2){
    if(s1.length()!=s2.length()) return false;
    int[] f=new int[26];
    for(int i=0;i<s1.length();i++){
      f[s1.charAt(i)-'a']++;
    }
    for(int i=0;i<s2.length();i++){
      f[s2.charAt(i)-'a']--;
    }
    for(int i=0;i<26;i++){
      if(f[i]!=0) return false;
    }
    return true;
  }
  public static void main(String[] args) {
    String s1="geeks";
    String s2="kseeg";
    System.out.println(anagram(s1,s2));
  }
}
```

OUTPUT:



```
true

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

Time Complexity: O(n)

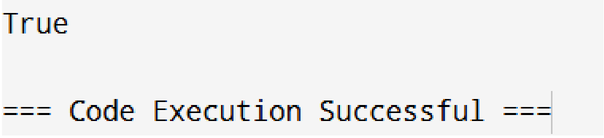
13. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

CODE:

```
import java.util.*;
class HelloWorld {
  public static boolean balance(String s){
    Stack<Character> st=new Stack<Character>();
    for(char i:s.toCharArray()){
      if(i=='('||i=='['||i=='{') st.push(i);
      else{
```

```
                if(st.isEmpty()) return false;
                char ch=st.pop();
                if((i==')'&&ch=='(')||(i==']'&&ch=='[')||(i=='}'&&ch=='{')) continue;
                else return false;
            }
        }
        return st.isEmpty();
    }
    public static void main(String[] args) {
        String s="()[{}()]";
        if(balance(s)==true)
        System.out.println("True");
        else
        System.out.println("False");
    }
}
```

OUTPUT:



```
True


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

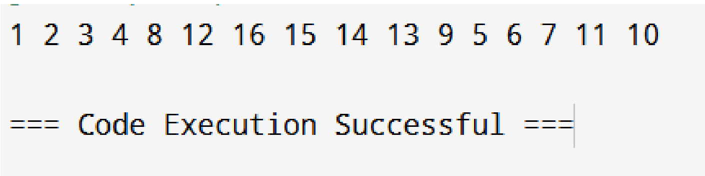
Time Complexity: O(n)

10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

CODE:

```
import java.util.*;
class HelloWorld {
    public static List<Integer> printSpiral(int[][] mat) {
        List<Integer> ans = new ArrayList<>();
        int n = mat.length;
        int m = mat[0].length;
        int top = 0, left = 0, bottom = n - 1, right = m - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++)
                ans.add(mat[top][i]);
            top++;
            for (int i = top; i <= bottom; i++)
                ans.add(mat[i][right]);
            right--;
            if (top <= bottom) {
                for (int i = right; i >= left; i--)
                    ans.add(mat[bottom][i]);
                bottom--;
            }
            if (left <= right) {
                for (int i = bottom; i >= top; i--)
                    ans.add(mat[i][left]);
                left++;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        int[][] mat = {{1, 2, 3, 4},
                       {5, 6, 7, 8},
                       {9, 10, 11, 12},
                       {13, 14, 15, 16}};
        List<Integer> ans = printSpiral(mat);
        for(int i = 0;i<ans.size();i++){
            System.out.print(ans.get(i) + " ");
        }
        System.out.println();
    }
}
```

OUTPUT:



```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

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

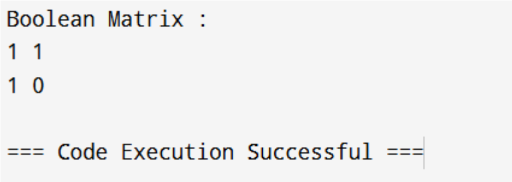
Time Complexity: O(m*n)

9. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

CODE:

```
import java.util.*;
class HelloWorld {
    public static void modifyMatrix(int mat[][])
    {
        boolean row_flag = false;
        boolean col_flag = false;
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[0].length; j++) {
                if (i == 0 && mat[i][j] == 1)
                    row_flag = true;
                if (j == 0 && mat[i][j] == 1)
                    col_flag = true;
                if (mat[i][j] == 1) {
                    mat[0][j] = 1;
                    mat[i][0] = 1;
                }
            }
        }
        for (int i = 1; i < mat.length; i++)
            for (int j = 1; j < mat[0].length; j++)
                if (mat[0][j] == 1 || mat[i][0] == 1)
                    mat[i][j] = 1;
        if (row_flag == true)
            for (int i = 0; i < mat[0].length; i++)
                mat[0][i] = 1;
    if (col_flag == true)
      for (int i = 0; i < mat.length; i++)
        mat[i][0] = 1;
  }
  public static void printMatrix(int mat[][])
  {
    for (int i = 0; i < mat.length; i++) {
      for (int j = 0; j < mat[0].length; j++)
        System.out.print(mat[i][j] + " ");
      System.out.println("");
    }
  }
  public static void main(String args[])
  {
    int mat[][] = { { 1, 0}, {0, 0 } };
    modifyMatrix(mat);
    System.out.println("Boolean Matrix :");
    printMatrix(mat);
  }
}
```

OUTPUT:



```
Boolean Matrix :
1 1
1 0

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

Time Complexity: O(m*n)

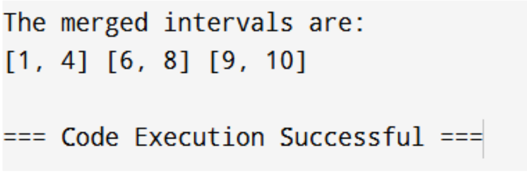
8. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

CODE:

```
import java.util.*;
class HelloWorld {
  public static List<List<Integer>> mergeOverlappingIntervals(int[][] arr) {
    int n = arr.length;
    Arrays.sort(arr, new Comparator<int[]>() {
      public int compare(int[] a, int[] b) {
        return a[0] - b[0];
      }
    });
    List<List<Integer>> ans = new ArrayList<>();
    for (int i = 0; i < n; i++) {
      if (ans.isEmpty() || arr[i][0] > ans.get(ans.size() - 1).get(1)) {
        ans.add(Arrays.asList(arr[i][0], arr[i][1]));
      }
      else {
        ans.get(ans.size() - 1).set(1,Math.max(ans.get(ans.size() - 1).get(1), arr[i][1]));
      }
    }
```

```
        return ans;
    }
    public static void main(String[] args) {
        int[][] arr = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        List<List<Integer>> ans = mergeOverlappingIntervals(arr);
        System.out.print("The merged intervals are: \n");
        for (List<Integer> it : ans) {
            System.out.print("[" + it.get(0) + ", " + it.get(1) + "] ");
        }
        System.out.println();
    }
}
```

OUTPUT:



```
The merged intervals are:
[1, 4] [6, 8] [9, 10]

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

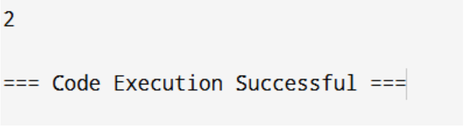
Time Complexity: O(n)

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

CODE:

```
import java.util.*;
class HelloWorld {
    public static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            if (diff < minDiff)
                minDiff = diff;
        }
        return minDiff;
    }
    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        System.out.println(findMinDiff(arr, m));
    }
}
```

OUTPUT:



```
2

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

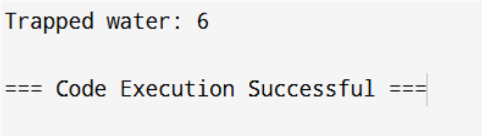
Time Complexity: O(n log n)

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

CODE:

```
import java.util.*;
class HelloWorld {
  public static int trap(int[] height) {
    int n = height.length;
    int left = 0, right = n - 1;
    int res = 0;
    int maxLeft = 0, maxRight = 0;
    while (left <= right) {
      if (height[left] <= height[right]) {
        if (height[left] >= maxLeft) {
          maxLeft = height[left];
        } else {
          res += maxLeft - height[left];
        }
        left++;
      } else {
        if (height[right] >= maxRight) {
          maxRight = height[right];
        } else {
          res += maxRight - height[right];
        }
        right--;
      }
    }
    return res;
  }
  public static void main(String args[]) {
    int arr[] = {0,1,0,2,1,0,1,3,2,1,2,1};
    System.out.println("Trapped water: " + trap(arr));
  }
}
```

OUTPUT:



```
Trapped water: 6

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

Time Complexity: O(n)

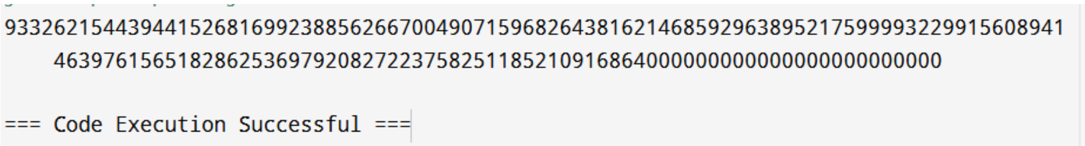
5. Find the Factorial of a large number

CODE:

```
import java.math.BigInteger;
import java.util.Scanner;
class HelloWorld {
  public static BigInteger fact(int n)
  {
    BigInteger f= new BigInteger("1");
    for (int i = 2; i <= n; i++)
```

```
        f = f.multiply(BigInteger.valueOf(i));
      return f;
   }
   public static void main(String[] args) {
      int n = 100;
      System.out.println(fact(n));
   }
}
```

OUTPUT:

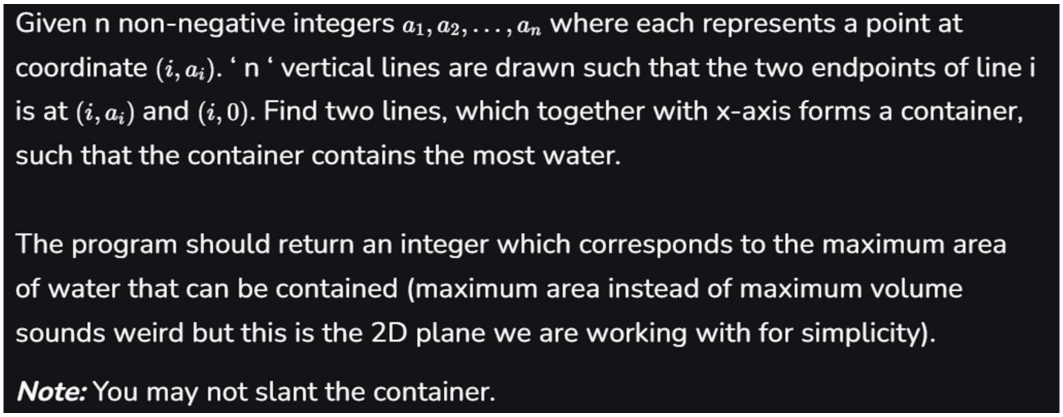


```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941
    463976156518286253697920827223758251185210916864000000000000000000000000

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

Time Complexity: O(n)

4. Container with Most Water



Given n non-negative integers $a_1, a_2, \ldots, a_n$ where each represents a point at coordinate (i, a_i). ‘ n ‘ vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

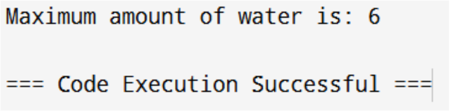
The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

CODE:

```
import java.util.*;
class HelloWorld {
   public static int area(int nums[]){
      int i=0,j=nums.length-1,maxi=Integer.MIN_VALUE;
      while(i<j){
         int w=(j-i)*Math.min(nums[i],nums[j]);
         maxi=Math.max(maxi,w);
         if(nums[i]<nums[j]) i++;
         else j--;
      }
      return maxi;
   }
   public static void main(String[] args) {
      int nums[] ={1,5,4,3};
      System.out.println("Maximum amount of water is: "+area(nums));
   }
}
```

OUTPUT:



```
Maximum amount of water is: 6

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

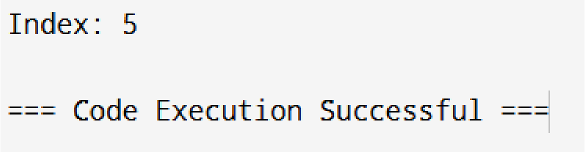
Time Complexity: O(n)

3. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1.

CODE:

```
import java.util.*;
class HelloWorld {
  public static int search(ArrayList<Integer> arr,int n,int k){
    int l=0,h=arr.size()-1;
    while(l<=h){
      int m=(l+h)/2;
      if(arr.get(m)==k) return m;
      if(arr.get(l)<=arr.get(m)){
        if(arr.get(l)<=k && k<=arr.get(m)){
          h=m-1;
        }else{
          l=m+1;
        }
      }else{
        if(arr.get(m)<=k && k<=arr.get(h)){
          l=m+1;
        }else{
          h=m-1;
        }
      }
    }
    return -1;
  }
  public static void main(String[] args) {
    ArrayList<Integer> arr=new ArrayList<>(Arrays.asList(4,5,6,7,0,1,2));
    int n=9,k=1;
    int ans=search(arr,n,k);
    System.out.println("Index: "+ans);
  }
}
```

OUTPUT:



```
Index: 5

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

Time Complexity: O(log n)

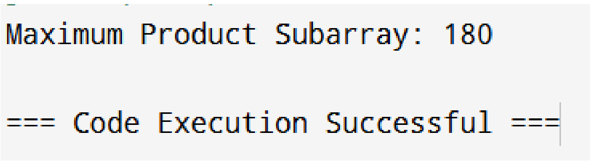
2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

CODE:

```
import java.util.*;
class HelloWorld {
  public static int maxproduct(int[] nums){
    int n=nums.length;
    int leftprod=1;
    int rightprod=1;
    int ans=nums[0];
```

```
        for(int i=0;i<n;i++){
            leftprod=leftprod==0?1:leftprod;
            rightprod=rightprod==0?1:rightprod;
            leftprod*=nums[i];
            rightprod*=nums[n-1-i];
            ans=Math.max(ans,Math.max(leftprod,rightprod));
        }
        return ans;
    }
    public static void main(String[] args) {
        int[] arr={-2,6,-3,-10,0,2};
        int ans=maxproduct(arr);
        System.out.println("Maximum Product Subarray: "+ans);
    }
}
```

OUTPUT:



```
Maximum Product Subarray: 180

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

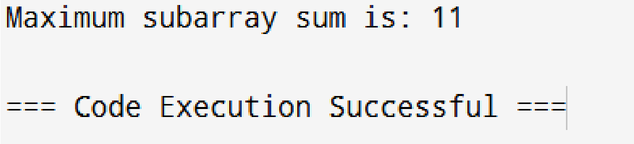
Time Complexity: O(n)

1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

CODE:

```
import java.util.*;
class HelloWorld {
    public static long maxSubarraySum(int[] arr, int n) {
        long maxi = Long.MIN_VALUE;
        long sum = 0;
        for (int i = 0; i < n; i++) {
            sum += arr[i];
            if (sum > maxi) {
                maxi = sum;
            }
            if (sum < 0) {
                sum = 0;
            }
        }
        return maxi;
    }
    public static void main(String args[]) {
        int[] arr = { 2, 3, -8, 7, -1, 2, 3};
        int n = arr.length;
        long maxSum = maxSubarraySum(arr, n);
        System.out.println("Maximum subarray sum is: " + maxSum);
    }
}
```

OUTPUT:



```
Maximum subarray sum is: 11


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

Time Complexity: O(n)