1. Maximum Subarray Sum – Kadane"s Algorithm:

```
import java.util.*;
public class tUf {
    public static long maxSubarraySum(int[] arr, int n) {
        long maxi = Long.MIN_VALUE; // maximum sum
        long sum = 0;
        int start = 0;
        int ansStart = -1, ansEnd = -1;
        for (int i = 0; i < n; i++) {
            if (sum == 0) start = i; // starting index
            sum += arr[i];
            if (sum > maxi) {
                maxi = sum;
                ansStart = start;
                ansEnd = i;
            }
            // If sum < 0: discard the sum calculated
            if (sum < 0) {
                sum = 0;
            }
        }
        //printing the subarray:
        System.out.print("The subarray is: [");
        for (int i = ansStart; i <= ansEnd; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.print("]n");
        // To consider the sum of the empty subarray
        // uncomment the following check:
```

```
        //if (maxi < 0) maxi = 0;
        return maxi;
    }
    public static void main(String args[]) {
        int[] arr = { -2, 1, -3, 4, -1, 2, 1, -5, 4};
        int n = arr.length;
        long maxSum = maxSubarraySum(arr, n);
        System.out.println("The maximum subarray sum is: " + maxSum);
    }
}
```

OUTPUT:

The subarray is: [4 -1 2 1]The maximum subarray sum is: 6

TIME COMPLEXITY :O(N)

SPACE COMPLEXITY: O(1)

2. Maximum Product Subarray :

```
import java.util.*;
public class Kadan{
public static int MaxArr(int arr[]) {
    int l=arr.length;
    int pre=1,post=1;
    int res=Integer.MIN_VALUE;
    for(int i=0;i<l;i++){
        if(pre==0) pre=1;
        if(post==0) post=1;
        pre*=arr[i];
        post*=arr[l-i-1];
        res=Math.max(res, Math.max(pre,post));

    }
```

```
    return res;
   }
   public static void main(String[] args) {
    int arr[]={-2, 6, -3, -10, 0, 2};
    int ans=MaxArr(arr);
    System.out.println("The Maximum product of subarray is"+ans);
   }
}
```

OUTPUT:

The Maximum product of subarray is: 180

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

3. Search in a sorted and rotated Array

```
import java.util.*;
public class GFG {
    // Search in a pivoted sorted array
    public static int pivotedSearch(List<Integer> arr, int key) {
        int low = 0, high = arr.size() - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            // Case 1: Find key
            if (arr.get(mid) == key)
                return mid;
            // Case 2: Left half is sorted
            if (arr.get(mid) >= arr.get(low)) {
                if (key >= arr.get(low) && key < arr.get(mid))
                    high = mid - 1;
                else
                    low = mid + 1;
```

```
            }
            // Case 3: Right half is sorted
            else {
                if (key > arr.get(mid) && key <= arr.get(high))
                    low = mid + 1;
                else
                    high = mid - 1;
            }
        }
        return -1; // Key not found
    }
    public static void main(String[] args) {
        List<Integer> arr1 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);
        int key1 = 0;
        int result1 = pivotedSearch(arr1, key1);
        System.out.println("The index of given key element is:"+ result1); // Output: 4    }
}
```

OUTPUT:

The index of given key element is:8

TIME COMPLEXITY:O(LOG N)

SPACE COMPLEXITY:O(1)

4. Container with Most Water :

```
import java.util.*;
class Solution {
    public int maxArea(int[] height) {
        int m=0;
        int left=0;
        int right=height.length-1;
        while(left < right){
```

```
            m=Math.max(m,(right-left)*Math.min(height[left],height[right]));
        if(height[left]<height[right]){
            left++;
        }
        else{
        right--;}}
        return m;
    }
 public static void main(String[] args) {
        int[] arr1 ={ 1,8,6,2,5,4,8,3,7};
        int result1 = MaxArea(arr1);
        System.out.println("The max is:"+ result1); // Output: 49
}
```

OUTPUT:

The max is :6

TIME COMPLEXITY:O(log n)

SPACE COMPLEXITY:O(1)

5. Find the Factorial of a large number

```
import java.math.BigInteger;
import java.util.Scanner;
public class Example {
    static BigInteger factorial(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[]) throws Exception
    {
        int N = 100;
        System.out.println("Factorial of given number is:"+factorial(N));
    }
}
```

OUTPUT:

Factorial of given number is:

933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

6.Trapping Rainwater Problem

```
class Solution {
    public satic int trap(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int leftMax = height[left];
        int rightMax = height[right];
        int water = 0;
        while (left < right) {
            if (leftMax < rightMax) {
                left++;
                leftMax = Math.max(leftMax, height[left]);
                water += leftMax - height[left];
            } else {
                right--;
                rightMax = Math.max(rightMax, height[right]);
                water += rightMax - height[right];
            }
```

```
        }
        return water;
    }
public static void main(String args[]){
    int[] arr={0,1,0,2,1,0,1,3,2,1,2,1};
    int a=trap(arr);
    System.out.println("The unit of water will be : "+a);
}
}
```

OUTPUT:

The unit of water will be : 6

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

7. Chocolate Distribution Problem

```
import java.util.*;
public class One{
 public static int Chocolates(int arr[],int key){
   Arrays.sort(arr);
   return arr[key-1]-arr[0];
}
public static void main(String args[]){
 int[] newa={7, 3, 2, 4, 9, 12, 56};
 int k=3;
 int val=Chocolates(newa,k);
 System.out.println(val);
}
}
```

OUTPUT:

2

TIME COMPLEXITY:O(N log N)

SPACE COMPLEXITY:O(1)

8. Merge Overlapping Intervals

```
import java.util.*;
class Solution {
    public static int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
        List<int[]> merged = new ArrayList<>();
        int[] prev = intervals[0];
        for (int i = 1; i < intervals.length; i++) {
            int[] interval = intervals[i];
            if (interval[0] <= prev[1]) {
                prev[1] = Math.max(prev[1], interval[1]);
            } else {
                merged.add(prev);
                prev = interval;
            }
        }
        merged.add(prev);
        return merged.toArray(new int[merged.size()][]);
    }
public static void main(String args[]){
 int[][] newa={{1,3},{2,6},{8,10},{15,18}};
 int[][] val = merge(newa);
 for(int[] i:val){
            System.out.println(Arrays.toString(i));
 }
}
```

```
}
```

OUTPUT:

Merged Intervals:

[1, 6]

[8, 10]

[15, 18]

TIME COMPLEXITY:O(N log N)

SPACE COMPLEXITY:O(1)

9. A Boolean Matrix Question

```
import java.util.Arrays;
class One {
    public static int[][] setone(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        boolean firstRowHasZero = false;
        boolean firstColHasZero = false;
        for (int c = 0; c < cols; c++) {
            if (matrix[0][c] == 1) {
                firstRowHasZero = true;
                break;
            }
        }
        for (int r = 0; r < rows; r++) {
            if (matrix[r][0] == 1) {
                firstColHasZero = true;
                break;
            }
        }
        for (int r = 1; r < rows; r++) {
```

```
    for (int c = 1; c < cols; c++) {
        if (matrix[r][c] == 1) {
            matrix[r][0] = 1;
            matrix[0][c] = 1;
        }
    }
}
for (int r = 1; r < rows; r++) {
    if (matrix[r][0] == 1) {
        for (int c = 1; c < cols; c++) {
            matrix[r][c] = 1;
        }
    }
}
for (int c = 1; c < cols; c++) {
    if (matrix[0][c] == 1) {
        for (int r = 1; r < rows; r++) {
            matrix[r][c] = 1;
        }
    }
}
if (firstRowHasZero) {
    for (int c = 0; c < cols; c++) {
        matrix[0][c] = 1;
    }
}
if (firstColHasZero) {
    for (int r = 0; r < rows; r++) {
        matrix[r][0] = 1;
    }
}
    return matrix;
```

```
    }
    public static void main(String[] args) {
        int[][] matrix = {
          {1, 0, 0, 1},
           {0, 0, 1, 0},
          {0, 0, 0, 0}
        };
        int[][] ans = setone(matrix);
            for (int[] row : ans) {
            System.out.println(Arrays.toString(row));
        }
    }
}
```

OUTPUT:

[1, 1, 1, 1]

[1, 1, 1, 1]

[1, 0, 1, 1]

TIME COMPLEXITY: O(M*N)

SPACE COMPLEXITY: O(1)

10. Print a given matrix in spiral form

```
import java.util.*;
class One {
    public static List<Integer> spiralOrder(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int x = 0;
        int y = 0;
        int dx = 1;
        int dy = 0;
```

```
        List<Integer> res = new ArrayList<>();
        for (int i = 0; i < rows * cols; i++) {
            res.add(matrix[y][x]);
            matrix[y][x] = -101; // the range of numbers in matrix is from -100 to 100
            // Change direction when hitting the boundary or visited cell
              if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) || matrix[y+dy][x+dx] ==
-101) {
                int temp = dx;
                dx = -dy;
                dy = temp;
            }
            x += dx;
            y += dy;
        }
        return res;
    }
    public static void main(String[] args) {
       int[][] matrix ={{1,    2,   3,   4},
                        {5,    6,   7,   8},
                        {9,   10,  11,  12},
                       {13,  14,  15,  16 }} ;
        List<Integer> result = spiralOrder(matrix);
        System.out.println("Spiral Order:");
        System.out.println(result);
    }
}
```

OUTPUT:

```
Spiral Order:
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]
```

TIME COMPLEXITY: O(M*N)

SPACE COMPLEXITY: O(M*N)

11.Check if given Parentheses expression is balanced or not

```
import java.util.Scanner;
import java.util.Stack;
class One {
    public static String isBalanced(String str) {
        Stack<Character> stack = new Stack<>();
        for (char ch : str.toCharArray()) {
            if (ch == '(') {
                stack.push(ch);
            }
            else if (ch == ')') {
                if (stack.isEmpty()) {
                    return "Not Balanced";
                }
                stack.pop(); // Pop the matching opening parenthesis
            }
        }
        return stack.isEmpty() ? "Balanced" : "Not Balanced";
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
       System.out.print("Enter a string of parentheses: ");
        String str = scanner.nextLine();
       System.out.println("Output: " + isBalanced(str));
    }
}
```

OUTPUT:

Enter a string of parentheses: ()()()

Output: Balanced

Enter a string of parentheses: ()(((()()(((()

Output: Not Balanced

TIME COMPLEXITY: O(N)

SPACE COMPLEXITY: O(N)

12. Check if two Strings are Anagrams of each other

```
import java.util.Scanner;
import java.util.Arrays;
class Solution {
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();
        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the first string: ");
        String s1 = scanner.nextLine();
        System.out.print("Enter the second string: ");
        String s2 = scanner.nextLine();
        if (areAnagrams(s1, s2)) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
        scanner.close();
```

```
    }
}
```

OUTPUT:

Enter the first string: hsgduew

Enter the second string: hjgdhj

false

TIME COMPLEXITY:O(N log N)

SPACE COMPLEXITY:O(N)

13.Longest Palindromic Substring

```
import java.util.Scanner;
class Solution {
    public static String longestPalindrome(String str) {
        if (str == null || str.length() == 0) {
            return "";
        }
        String longest = "";
        for (int i = 0; i < str.length(); i++) {
            String oddPalindrome = expandAroundCenter(str, i, i);    // Odd length palindrome
            String evenPalindrome = expandAroundCenter(str, i, i + 1); // Even length palindrome
            if (oddPalindrome.length() > longest.length()) {
                longest = oddPalindrome;
            }
            if (evenPalindrome.length() > longest.length()) {
                longest = evenPalindrome;
            }
        }
        return longest;
    }
    private static String expandAroundCenter(String str, int left, int right) {
```

```
        while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
            left--;
            right++;
        }
        return str.substring(left + 1, right);
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        String result = longestPalindrome(str);
        System.out.println("Longest Palindromic Substring: " + result);
        scanner.close();
    }
}
```

OUTPUT:

Enter a string: kjaswygdnjdjjjjjjjjbdsv

Longest Palindromic Substring: jjjjjjjj

TIME COMPLEXITY:O(N^2)

SPACE COMPLEXITY:O(1)

14.Longest Common Prefix using Sorting

```
import java.util.Arrays;
class Solution {
    public static String longestCommonPrefix(String[] arr) {
        if (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) {
        String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr1));
    }
}
```

OUTPUT:

Longest Common Prefix: gee

TIME COMPLEXITY:O(N LOG N)

SPACE COMPLEXITY:O(1)

15. Delete middle element of a stack

```
import java.util.Stack;
public class DeleteMiddleElement {
    public static void deleteMiddle(Stack<Integer> stack, int currentIndex, int size) {
        if (currentIndex == size / 2) {
            stack.pop();
            return;
        }
        int topElement = stack.pop();
        deleteMiddle(stack, currentIndex + 1, size);
```

```
        stack.push(topElement);
    }
    public static void deleteMiddle(Stack<Integer> stack) {
        int size = stack.size();
        deleteMiddle(stack, 0, size);
    }
    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("Original Stack: " + stack);
        deleteMiddle(stack);
        System.out.println("Stack after deleting middle element: " + stack);
        Stack<Integer> stack2 = new Stack<>();
        stack2.push(1);
        stack2.push(2);
        stack2.push(3);
        stack2.push(4);
        stack2.push(5);
        stack2.push(6);
        System.out.println("Original Stack: " + stack2);
        deleteMiddle(stack2);
        System.out.println("Stack after deleting middle element: " + stack2);
    }
}
```

OUTPUT:

Original Stack: [1, 2, 3, 4, 5]

Stack after deleting middle element: [1, 2, 4, 5]

Original Stack: [1, 2, 3, 4, 5, 6]

Stack after deleting middle element: [1, 2, 4, 5, 6]

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY: O(1)

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

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

```
import java.util.Stack;
public class NextGreaterElement {
    public static void printNextGreater(int[] arr) {
        int n = arr.length;
        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            // Pop elements from the stack while they are smaller or equal to arr[i]
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            if (!stack.isEmpty()) {
                System.out.println(arr[i] + " --> " + stack.peek());
            } else {
                System.out.println(arr[i] + " --> -1");
            }
            stack.push(arr[i]);
        }
    }
    public static void main(String[] args) {
        int[] arr1 = {4, 5, 2, 25};
        System.out.println("Next Greater Elements for arr1:");
        printNextGreater(arr1);
    }
}
```

OUTPUT:

Next Greater Elements for arr1:

25 --> -1

2 --> 25

5 --> 25

4 --> 5

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY: O(N)

17.Print Right View of a Binary Tree

```
import java.util.ArrayList;
import java.util.List;
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) {
        val = x;
    }
}
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        rightView(root, result, 0);
        return result;
    }
    public void rightView(TreeNode curr, List<Integer> result, int currDepth) {
        if (curr == null) {
            return;
        }
```

```
        if (currDepth == result.size()) {
            result.add(curr.val);
        }
        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);
        root.left.left.left = new TreeNode(7);
        Solution solution = new Solution();
        List<Integer> rightView = solution.rightSideView(root);
        System.out.println("Right side view of the binary tree: " + rightView);
    }
}
```

OUTPUT:

Right side view of the binary tree: [1, 3, 6, 7]

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(H)

18. Maximum Depth or Height of Binary Tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
```

```
  TreeNode(int x) {
      val = x;
    }
}
public class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        return Math.max(left, right) + 1;
    }
    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);
        root.left.left.left = new TreeNode(7);
        Solution solution = new Solution();
        int depth = solution.maxDepth(root);
        System.out.println("Maximum depth of the binary tree: " + depth);
    }
}
```

OUTPUT:

Maximum depth of the binary tree: 4

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(N)