**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.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3}

Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11

**Code**

import java.util.\*;

public class Kadanesalgo {

    static int outputl(int[] arr) {

        int result = arr[0];

        int maximum = arr[0];

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

            maximum = Math.max(maximum + arr[i], arr[i]);

            result = Math.max(result, maximum);

        }

        return result;

    }

    public static void main(String[] args) {

        int[] arr = { -2, -4, 5 };

        System.out.println(outputl(arr));

    }

}

**Output**



**Time Complexity**

**O(n)**

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

Input: arr[] = {-2, 6, -3, -10, 0, 2}

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10) = 180

**Code**

public class Productarray {

    public static void main(String[] args) {

        int[] arr = { -2, 6, -3, -10, 0, 2 };

        int m = arr[0];

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

            int sum = 1;

            for (int j = i; j < arr.length; j++) {

                sum \*= arr[j];

                m = Math.max(m, sum);

            }

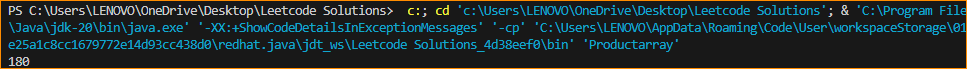
        }

        System.out.println(m);

    }

}

**Output**



**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.

Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0

Output : 4

**Code**

public class Arraym {

    public static void main(String[] args) {

        int[] arr = { 3, 5, 7, 9, 11 };

        int target = 9;

        int result = binarySearchm(arr, target);

        if (result == -1) {

            System.out.println("Element not found");

        } else {

            System.out.println("Element found at index " + result);

        }

    }

    static int binarySearchm(int[] arr, int target) {

        int left = 0;

        int right = arr.length - 1;

        while (left <= right) {

            int mid = (right + left) / 2;

            if (arr[mid] == target) {

                return mid;

            }

            if (arr[left] <= arr[mid]) {

                if (arr[left] <= target && target < arr[mid]) {

                    right = mid - 1;

                } else {

                    left = mid + 1;

                }

            } else {

                if (arr[mid] < target && target <= arr[right]) {

                    left = mid + 1;

                } else {

                    right = mid - 1;

                }

            }

        }

        return -1;

    }

}

**Output**



**Time Complexity**

**O(log n)**

**4 . Container with Most Water Input**:

arr = [1, 5, 4, 3] ,Output: 6

Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

**Code**

public class Waterproblem {

    public static void main(String[] args) {

        int[] arr = { 1, 5, 4, 3 };

        int i = 0;

        int j = arr.length - 1;

        int result = 0;

        while (i < j) {

            int just = Math.min(arr[i], (arr[j])) \* (j - i);

            if (arr[i] > arr[j]) {

                j--;

            } else {

                i++;

            }

            result = Math.max(result, just);

        }

        System.out.print(result);

}

}

**Output**



**Time Complexity**

**O(n)**

**5. Find the Factorial of a large number**

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

**Code**

import java.math.BigInteger;

public class Factorial {

    public static BigInteger factorial(BigInteger n) {

        if (n.equals(BigInteger.ZERO) || n.equals(BigInteger.ONE)) {

            return BigInteger.ONE;

        } else {

            return n.multiply(factorial(n.subtract(BigInteger.ONE)));

        }

    }

    public static void main(String[] args) {

        BigInteger number = new BigInteger("50");

        BigInteger result = factorial(number);

        System.out.println("Factorial of " + number + " is: " + result);

    }

}

**Output**



**Time Complexity**

**O(n)**

**6 . Trapping Rainwater Problem** - 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.

Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

**Code**

public class Rainwater {

    public static void main(String[] args) {

        int[] arr = { 3, 0, 1, 0, 4, 0, 2 };

        int result = trap(arr);

        System.out.println("Trapped water: " + result);

    }

    public static int trap(int[] height) {

        int result = 0;

        int[] maxi = new int[height.length];

        int[] mini = new int[height.length];

        int a = 0;

        maxi[0] = a;

        for (int i = 1; i < height.length; i++) {

            if (a < height[i - 1]) {

                a = height[i - 1];

            }

            maxi[i] = a;

        }

        mini[height.length - 1] = height[height.length - 1];

        int temp = height[height.length - 1];

        for (int i = height.length - 2; i >= 0; i--) {

            if (temp < height[i + 1]) {

                temp = height[i + 1];

            }

            mini[i] = temp;

        }

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

            int res = Math.min(mini[i], maxi[i]) - height[i];

            if (res >= 0) {

                result += res;

            }

        }

        return result;

    }

}

**Output**



**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.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3 Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

**Code**

import java.util.Arrays;

public class ChocolateDistribution {

    public static void main(String[] args) {

        int[] arr = { 7, 3, 2, 4, 9, 12, 56 };

        int m = 3;

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

    }

    public static int findMinDiff(int[] arr, int m) {

        int n = arr.length;

        if (m == 0 || n == 0) {

            return 0;

        }

        Arrays.sort(arr);

        if (n < m) {

            return -1;

        }

        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;

    }

}

**Output**



**Time Complexity**

**O(n logn)**

**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.

Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]

Output: [[1, 4], [6, 8], [9, 10]]

Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

**Code**

import java.util.ArrayList;

import java.util.Arrays;

public class Mergeintervals {

    public static void main(String[] args) {

        int[][] intervals = { { 1, 3 }, { 2, 4 }, { 6, 8 }, { 9, 10 } };

        ArrayList<int[]> mergelist = new ArrayList<>();

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

        for (int i = 0; i < intervals.length - 1; i++) {

            if (intervals[i + 1][0] <= intervals[i][1]) {

                int lasend = Math.max(intervals[i + 1][1], intervals[i][1]);

                intervals[i + 1][0] = Math.min(intervals[i + 1][0], intervals[i][0]);

                intervals[i + 1][1] = lasend;

            } else {

                mergelist.add(new int[] { intervals[i][0], intervals[i][1] });

            }

        }

        mergelist.add(new int[] { intervals[intervals.length - 1][0], intervals[intervals.length - 1][1] });

        int[][] result = mergelist.toArray(new int[mergelist.size()][]);

        System.out.println("Merged intervals:");

        for (int[] interval : result) {

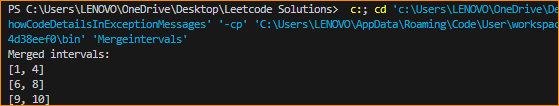
            System.out.println("[" + interval[0] + ", " + interval[1] + "]");

        }

    }

}

**Output**



**Time Complexity**

**O(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.

Input: {{0, 0, 0}, {0, 0, 1}}

Output: {{0, 0, 1}, {1, 1, 1}}

**Code**

public class Boolmatrix {

    public static void main(String[] args) {

        int[][] matrix = {

                { 1, 0, 0, 1 },

                { 0, 0, 1, 0 },

                { 0, 0, 0, 0 }

        };

        int rows = matrix.length;

        int cols = matrix[0].length;

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < cols; j++) {

                if (matrix[i][j] == 1) {

                    int ind = i - 1;

                    while (ind >= 0) {

                        if (matrix[ind][j] != 1) {

                            matrix[ind][j] = -1;

                        }

                        ind--;

                    }

                    ind = i + 1;

                    while (ind < rows) {

                        if (matrix[ind][j] != 1) {

                            matrix[ind][j] = -1;

                        }

                        ind++;

                    }

                    ind = j - 1;

                    while (ind >= 0) {

                        if (matrix[i][ind] != 1) {

                            matrix[i][ind] = -1;

                        }

                        ind--;

                    }

                    ind = j + 1;

                    while (ind < cols) {

                        if (matrix[i][ind] != 1) {

                            matrix[i][ind] = -1;

                        }

                        ind++;

                    }

                }

            }

        }

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < cols; j++) {

                if (matrix[i][j] < 0) {

                    matrix[i][j] = 1;

                }

            }

        }

        System.out.println("The Final Matrix is:");

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < cols; j++) {

                System.out.print(matrix[i][j] + " ");

            }

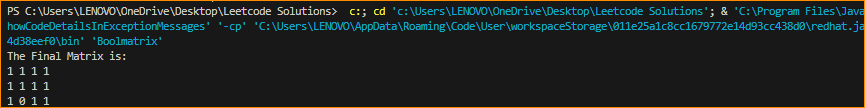
            System.out.println();

        }

    }

}

**Output**



**Time Complexity**

**O((n×m)×(n+m))**

**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.

Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }}

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

**Code**

public class SpiralMatrix {

    public static void printSpiral(int[][] matrix) {

        if (matrix == null || matrix.length == 0) {

            return;

        }

        int top = 0;

        int bottom = matrix.length - 1;

        int left = 0;

        int right = matrix[0].length - 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[][] matrix = {

                { 1, 2, 3, 4, 5, 6 },

                { 7, 8, 9, 10, 11, 12 },

                { 13, 14, 15, 16, 17, 18 }

        };

        System.out.println("Spiral order of matrix1:");

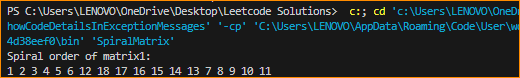
        printSpiral(matrix);

        System.out.println();

    }

}

**Output**



**Time Complexity**

**O(m\*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.

Input: str = “((()))()()”

Output: Balanced

**Code**

import java.util.\*;

public class Parenthesis {

    public static void main(String[] args) {

        String a = "((()))()()";

        Stack<Character> stack1 = new Stack<>();

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

            char ch = a.charAt(i);

            if (ch == '(') {

                stack1.push(ch);

            } else if (ch == ')') {

                if (stack1.isEmpty()) {

                    System.out.println("Not balanced");

                }

                stack1.pop();

            }

        }

        if (stack1.isEmpty()) {

            System.out.println("Balanced");

        } else {

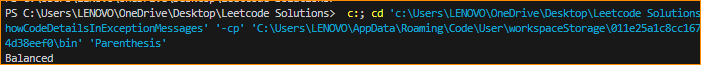
            System.out.println("Not balanced");

        }

    }

}

**Output**



**Time Complexity**

**O(n)**

**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.

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

**Code**

import java.util.Arrays;

public class Anagrams {

    public static void main(String[] args) {

        String a = "listen";

        String b = "silent";

        char[] c = a.toCharArray();

        char[] d = b.toCharArray();

        Arrays.sort(c);

        Arrays.sort(d);

        if (Arrays.equals(c, d)) {

            System.out.println("Anagrams");

        } else {

            System.out.println("Not anagrams");

        }

    }

}

**Output**



**Time Complexity**

**O(n log 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.

Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

**Code**

public class Palindrome {

    public static void main(String[] args) {

        String s = "forgeeksskeegfor";

        String res = "";

        int resLen = 0;

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

            int l = i, r = i;

            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

                if ((r - l + 1) > resLen) {

                    res = s.substring(l, r + 1);

                    resLen = r - l + 1;

                }

                l--;

                r++;

            }

            l = i;

            r = i + 1;

            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

                if ((r - l + 1) > resLen) {

                    res = s.substring(l, r + 1);

                    resLen = r - l + 1;

                }

                l--;

                r++;

            }

        }

        System.out.println(res);

    }

}

**Output**



**Time Complexity**

**O(n²)**

**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”.

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee Explanation: “gee” is the longest common prefix in all the given strings.

**Code**

import java.util.Arrays;

public class Commonprefix {

    public static void main(String[] args) {

        String[] arr = { "geeksforgeeks", "geeks", "geek", "geezer" };

        if (arr.length == 0) {

            System.out.println("-1");

        }

        Arrays.sort(arr);

        String first = arr[0];

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

        int minLength = Math.min(first.length(), last.length());

        int i = 0;

        while (i < minLength && first.charAt(i) == last.charAt(i)) {

            i++;

        }

        String commonPrefix = first.substring(0, i);

        if (commonPrefix.isEmpty()) {

            System.out.println("-1");

        } else {

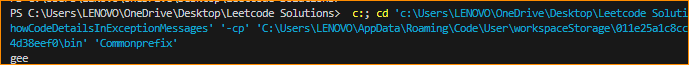
            System.out.println(commonPrefix);

        }

    }

}

**Output**



**Time Complexity**

**O(n log n \* m)**

**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.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

**Code**

import java.util.Stack;

public class Middleelement {

    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);

        stack.push(6);

        System.out.println("Original Stack: " + stack);

        deleteMiddle(stack, stack.size());

        System.out.println("Stack after deleting middle element: " + stack);

    }

    public static void deleteMiddle(Stack<Integer> stack, int size) {

        if (stack.isEmpty() || size == 0) {

            return;

        }

        int middleIndex = size / 2;

        deleteMiddleHelper(stack, middleIndex);

    }

    private static void deleteMiddleHelper(Stack<Integer> stack, int currentIndex) {

        if (currentIndex == 0) {

            stack.pop();

            return;

        }

        int topElement = stack.pop();

        deleteMiddleHelper(stack, currentIndex - 1);

        stack.push(topElement);

    }

}

**Output**



**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. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [ 4 , 5 , 2 , 25 ]

Output:

4 –> 5 5 –> 25 2 –> 25 25 –> -1

Explanation: Except 25 every element has an element greater than them present on the right side

**Code**

import java.util.Stack;

public class Nge {

    public static void main(String[] args) {

        int[] arr = { 34, 56, -9, 0 };

        printNGE(arr);

    }

    public static void printNGE(int[] arr) {

        int n = arr.length;

        int[] nge = new int[n];

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

        for (int i = n - 1; i >= 0; i--) {

            while (!stack.isEmpty() && stack.peek() <= arr[i]) {

                stack.pop();

            }

            if (stack.isEmpty()) {

                nge[i] = -1;

            } else {

                nge[i] = stack.peek();

            }

            stack.push(arr[i]);

        }

        for (int i = 0; i < n; i++) {

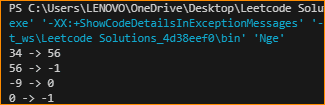
            System.out.println(arr[i] + " -> " + nge[i]);

        }

    }

}

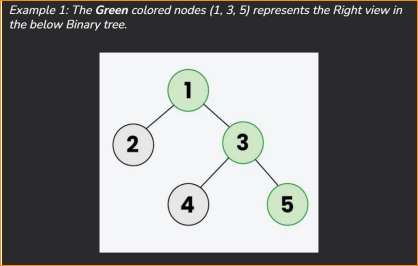
**Output**



**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.LinkedList;

import java.util.Queue;

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int item) {

        val = item;

        left = right = null;

    }

}

public class Rightviewofabinarytree {

    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.right.right = new TreeNode(7);

        System.out.println("Right view of the binary tree:");

        printRightView(root);

    }

    public static void printRightView(TreeNode root) {

        if (root == null) {

            return;

        }

        Queue<TreeNode> queue = new LinkedList<>();

        queue.add(root);

        while (!queue.isEmpty()) {

            int n = queue.size();

            for (int i = 1; i <= n; i++) {

                TreeNode node = queue.poll();

                if (i == n) {

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

                }

                if (node.left != null) {

                    queue.add(node.left);

                }

                if (node.right != null) {

                    queue.add(node.right);

                }

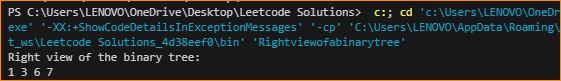
            }

        }

    }

}

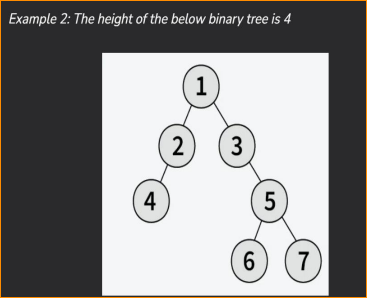
**Output**



**Time Complexity**

**0(n)**

**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**

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int item) {

        val = item;

        left = right = null;

    }

}

public class Maximumdepthofbinarytree {

    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.left = new TreeNode(6);

        root.right.right = new TreeNode(7);

        System.out.println("The maximum depth or height of the binary tree is: " + maxDepth(root));

    }

    public static int maxDepth(TreeNode root) {

        if (root == null) {

            return 0;

        }

        int leftDepth = maxDepth(root.left);

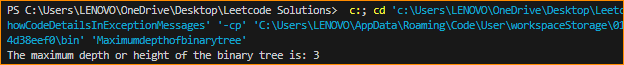
        int rightDepth = maxDepth(root.right);

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

    }

}

**Output**



**Time Complexity**

**O(n)**