DSA practice problems:

Name: Naresh M

Dept: AI&DS

Reg NO: 22AD072

Date : 09.11.24

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.

Input: arr[] = {-2, -4} Output: –2

Explanation: The subarray {-2} has the largest sum -2.

**Code:**

import java.util.\*;

public class maximumSubarraySum{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("enter the size of an array");

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int max\_curr=arr[0];

int max\_global=arr[0];

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

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

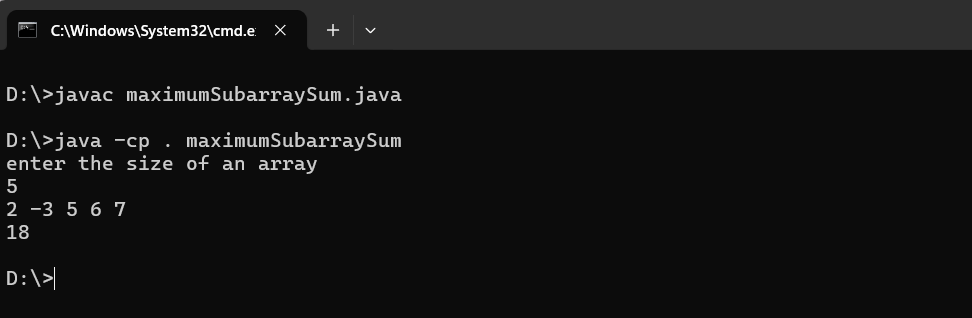
max\_global=Math.max(max\_global,max\_curr);

}

System.out.println(max\_global);

}

}

****

**Time Complexity: O(n)**

1. 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

Input: arr[] = {-1, -3, -10, 0, 60} Output: 60 Explanation: The subarray with maximum product is {60}.

**Code:**

import java.util.\*;

public class maximumProductSubarrray{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("enter the size of an array:");

int n=sc.nextInt();

System.out.println("enter the elements:");

int [] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int maxProduct = arr[0];

int minProduct = arr[0];

int res= arr[0];

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

int num = arr[i];

if (num < 0) {

int temp = maxProduct;

maxProduct = minProduct;

minProduct = temp;

}

maxProduct = Math.max(num, maxProduct \* num);

minProduct = Math.min(num, minProduct \* num;

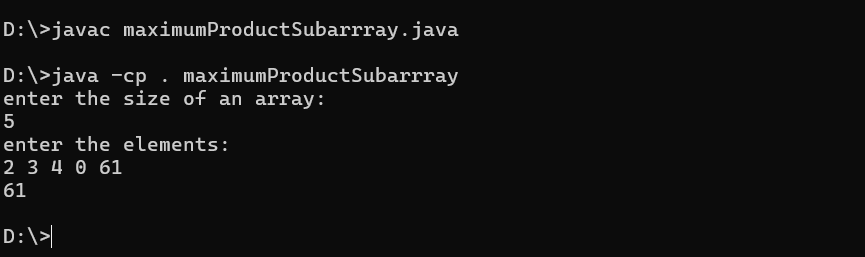
res= Math.max(res, maxProduct);

}

System.out.println(res);

}

}



Time Complexity: O(n)

1. **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

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

key = 3 Output : -1

**code:**

import java.util.\*;

public class searchInRotatedSortedArray {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("enter the size:");

        int n=sc.nextInt();

        System.out.println("enter target element");

        int target=sc.nextInt();

        int [] arr=new int[n];

        System.out.println("enter the elements of the array");

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        int l=0,r=n-1;

        boolean found = false;

        while (l<=r){

            int  mid=(l+r)/2;

            if (target==arr[mid]){

                System.err.println(mid);

                found = true;

            }

          if (arr[l]<=arr[mid]){

                if(target>arr[mid] || target<arr[l]){

                    l=mid+1;

                }else{

                    r=mid-1;

                }

            }

            else{

                if(target<arr[mid] || target>arr[r]){

                    r=mid-1;

                }else{

                    l=mid+1;

                }

            }

        }

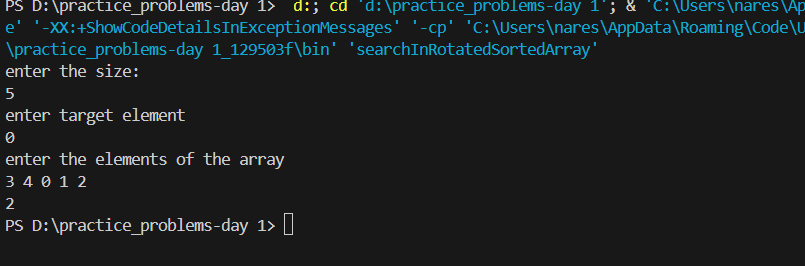
        if (!found) {

            System.out.println("-1");

        }

    }

}

****

**Time complexity:**

**O(n)**

**4.container with most water**

Given n non-negative integers a1, a2,..., an, where each represents a point at coordinate (i, a₁). 'n' vertical lines are drawn such that the two endpoints of line i is at (i, a₁) 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.

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

Input: arr = [3, 1, 2, 4, 5] Output: 12

Explanation: 5 and 3 are distance 4 apart.

So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**Code:**

import java.util.Scanner;

public class containerWithMostWater {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("enter the size of an array:");

        int n=sc.nextInt();

        System.out.println("enter the elements:");

        int [] arr=new int[n];

        int maxArea=0;

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        int l=0,r=n-1;

        while(l<r){

            int area=(r-l)\*Math.min(arr[l],arr[r]);

            maxArea = Math.max(maxArea, area);

            if(arr[l]<arr[r]){

                l++;

            }else{

                r--;

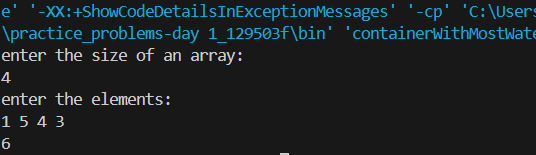
            }

        }

        System.out.println(maxArea);

    }

}



**Time complecity: O(n)**

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

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**Code:**

import java.math.BigInteger;

import java.util.\*;

public class factorialOfLargeNumbers {

    public static BigInteger factorial(int n) {

        if (n == 0 || n == 1) {

            return BigInteger.ONE;

        } else {

            BigInteger result = BigInteger.ONE;

        for (int i = 2; i <= n; i++) {

            result = result.multiply(BigInteger.valueOf(i));

        }

        return result;

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

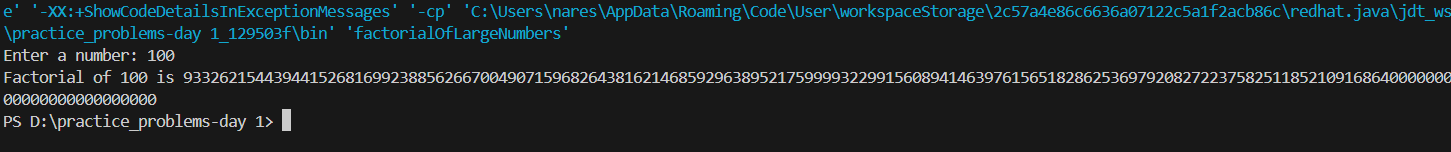
        int n = scanner.nextInt();

        scanner.close();

        BigInteger result =factorial(n);

        System.out.println("Factorial of "+ n +" is 100"+result);

    }

}

**Time complexity: O(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. Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image.

**Code:**

import java.util.Scanner;

public class trappingRainWater {

    public 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) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the size of the array: ");

        int n = scanner.nextInt();

        int[] height = new int[n];

        System.out.println("Enter the elements:");

        for (int i = 0; i < n; i++) {

            height[i] = scanner.nextInt();

        }

        trappingRainWater solution = new trappingRainWater();

        int result = solution.trap(height);

        System.out.println("The total water trapped: " + result);

        scanner.close();

    }

}

**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;

import java.util.Scanner;

class chocolateDistribution {

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

        if (m == 0 || arr.length < m) {

            return -1;

        }

        Arrays.sort(arr);

        int minDiff = Integer.MAX\_VALUE;

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

            int diff = arr[i + m - 1] - arr[i];

            minDiff = Math.min(minDiff, diff);

        }

        return minDiff;

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the size of array : ");

        int n=sc.nextInt();

        System.out.println("Enter the elements in array : ");

        int[] arr=new int[n];

        for(int i=0;i<n;i++)

        {

            arr[i]=sc.nextInt();

        }

        System.out.println("Enter the number of students : ");

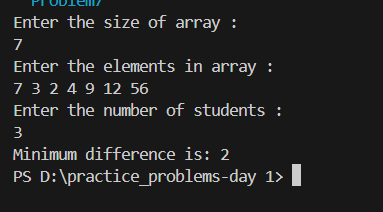
        int m=sc.nextInt();

        int result = findMinDiff(arr, m);

        System.out.println("Minimum difference is: " + result);

    }

}



**Time complexity: O(nlogn)**

**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]].

Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]] Output: [[1, 6], [7, 8]] Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

**Code:**import java.util.\*;

class mergedIntervals{

    public int[][] merge(int[][] intervals) {

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

        List<int[]> output = new ArrayList<>();

        int[] currentInterval = intervals[0];

        output.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;

                output.add(currentInterval);

            }

        }

        return output.toArray(new int[output.size()][]);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of intervals: ");

        int n = scanner.nextInt();

        int[][] intervals = new int[n][2];

        System.out.println("Enter the intervals in the format [start end]:");

        for (int i = 0; i < n; i++) {

            System.out.print("Interval " + (i + 1) + ": ");

            intervals[i][0] = scanner.nextInt();

            intervals[i][1] = scanner.nextInt();

        }

        mergedIntervals solution = new mergedIntervals();

        int[][] mergedIntervals = solution.merge(intervals);

        System.out.println("\nMerged intervals:");

        for (int[] interval : mergedIntervals) {

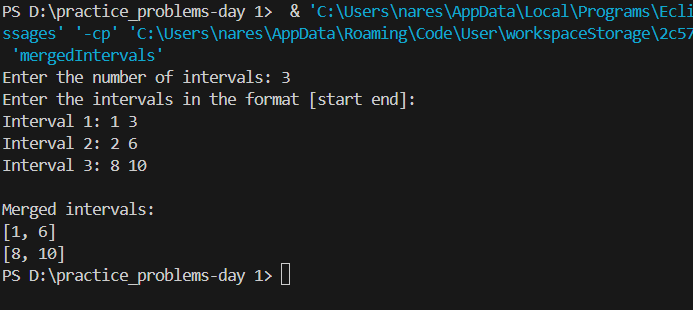
            System.out.println(Arrays.toString(interval));

        }

        scanner.close();

    }

}

****

**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: {{1, 0}, {0, 0}}

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

**Code:**

public class booleanMatrix{

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

        int numberOfRows = matrix.length;

        int numberOfCols = matrix[0].length;

        int[] rows = new int[numberOfRows];

        int[] cols = new int[numberOfCols];

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

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

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

                    rows[i] = 1;

                    cols[j] = 1;

                }

            }

        }

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

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

                if (rows[i] == 1 || cols[j] == 1) {

                    matrix[i][j] = 1;

                }

            }

        }

    }

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

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

            for (int j = 0; j < matrix[0].length; j++) {

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

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        int[][] matrix = {

            {1, 0, 0, 1},

            {0, 0, 1, 0},

            {0, 0, 0, 0},

        };

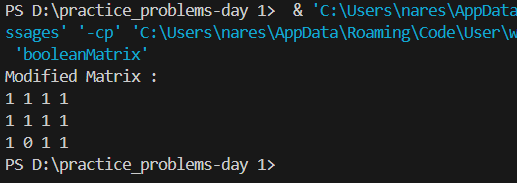
        System.out.println("Modified Matrix :");

        modifyMatrix(matrix);

        printMatrix(matrix);

    }

}



**Time Complexity: O (m x 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.

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:**

import java.util.\*;

public class spiralMatrix {

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++) {

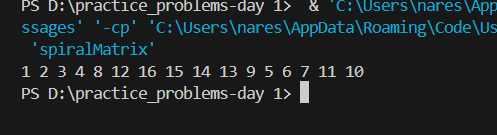
System.out.print(ans.get(i) + " ");

}

System.out.println();

}

}



**Time Complexity : O(m x 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 balancedParenthesis {

    public static boolean isBalanced(String s){

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

            for(char ch:s.toCharArray()){

                if (ch == '(') {

                    stack.push(ch);

                } else if (ch == ')'){

                    if (stack.isEmpty()) {

                        return false;

                    }

                    stack.pop();

                }

            }

            return stack.isEmpty() ;

        }

        public static void main(String[] args) {

            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter the parentheses string: ");

            String str = scanner.nextLine();

            if (isBalanced(str)) {

            System.out.println("Balanced");

        } else {

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

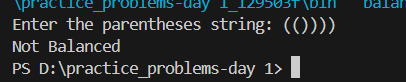
        }

        scanner.close();

    }

}



****

**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 strings have the same characters with the same frequency. So, they are anagrams.

**Code:**

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the first string: ");

String a = sc.nextLine();

System.out.println("Enter the second string: ");

String b = sc.nextLine();

if (a.length() != b.length()) {

System.out.println("The strings are not anagrams.");

return;

}

Map<Character, Integer> map = new HashMap<>();

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

map.put(a.charAt(i), map.getOrDefault(a.charAt(i), 0) + 1);

}

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

if (!map.containsKey(b.charAt(i)) || map.get(b.charAt(i)) == 0) {

System.out.println("The strings are not anagrams.");

return;

}

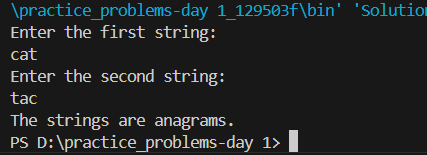
map.put(b.charAt(i), map.get(b.charAt(i)) - 1);

}

System.out.println("The strings are anagrams.");

}

}

****

**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.

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:**

import java.util.Scanner;

public class longestPalindromicSubstring {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a string: ");

String s = sc.nextLine();

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++;

}

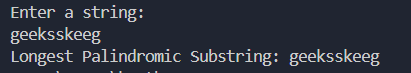
}

System.out.println("Longest Palindromic Substring: " + res);

sc.close();

}

}

****

**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”.

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

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

**Code:**

import java.util.\*;

public class longestCommonPrefix{

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of Strings : ");

int n=sc.nextInt();

String[] arr=new String[n];

System.out.println("Enter the Strings : ");

for(int i=0;i<n;i++)

{

arr[i]=sc.next();

}

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

System.out.println(-1);

}

Arrays.sort(arr);

String first = arr[0];

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

StringBuilder commonPrefix = new StringBuilder();

for (int i = 0; i < Math.min(first.length(), last.length()); i++) {

if (first.charAt(i) == last.charAt(i)) {

commonPrefix.append(first.charAt(i));

} else {

break;

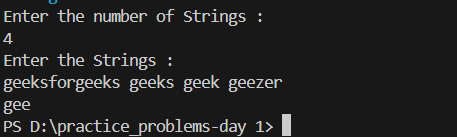
}

}

System.out.println(commonPrefix.length() > 0 ? commonPrefix.toString() : "-1");

}

}

****

**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.\*;

class deleteMiddleElementStack{

    public void deleteMiddle(Stack<Integer> stack, int currentIndex, int middleIndex) {

        if (currentIndex == middleIndex) {

            stack.pop();

            return;

        }

        int top = stack.pop();

        deleteMiddle(stack, currentIndex + 1, middleIndex);

        stack.push(top);

    }

    public void deleteMiddleElement(Stack<Integer> stack) {

        int size = stack.size();

        if (size == 0) return;

        int middleIndex = size / 2;

        deleteMiddle(stack, 0, middleIndex);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        deleteMiddleElementStack sol = new deleteMiddleElementStack();

        System.out.print("Enter the number of elements in the stack: ");

        int n = scanner.nextInt();

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

        System.out.println("Enter the elements of the stack:");

        for (int i = 0; i < n; i++) {

            stack.push(scanner.nextInt());

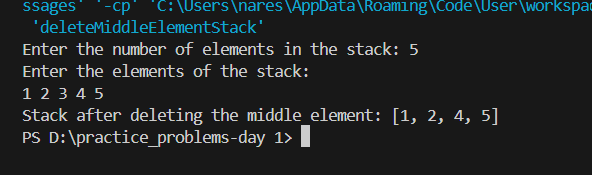
        }

        sol.deleteMiddleElement(stack);

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

        scanner.close();

    }}

}

**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 exists, 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;

import java.util.\*;

public class NGE {

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();

}

nge[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

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

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

}

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the size of the array : ");

int n=sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array : ");

for (int i = 0; i < n; i++)

{

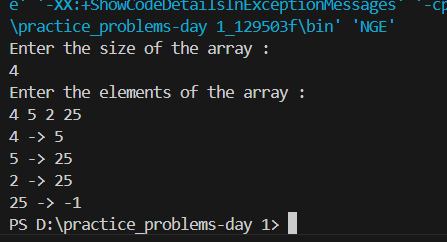
arr[i]=sc.nextInt();

}

printNGE(arr);

}

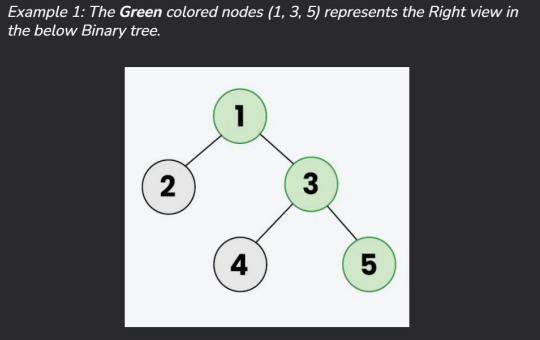
}

****

**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.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

class rightViewBinaryTree {

public List<Integer> rightSideView(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

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);

rightViewBinaryTree solution = new rightViewBinaryTree ();

List<Integer> rightViewList = solution.rightSideView(root);

System.out.println("Right View of the Binary Tree:");

for (Integer val : rightViewList)

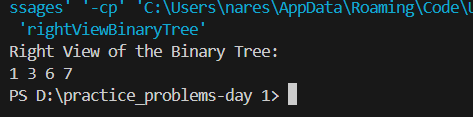
{

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

}

}

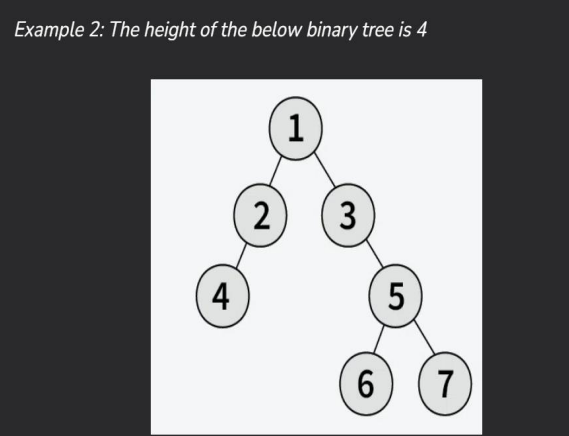
}

****

**Time Complexity:** O(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;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

class maximumDepthBinaryTree {

public int maxDepth(TreeNode root) {

if (root == null) {

return 0;

}

int lh = maxDepth(root.left);

int rh = maxDepth(root.right);

return 1 + Math.max(lh, rh);

}

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);

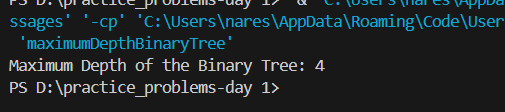
maximumDepthBinaryTree solution = new maximumDepthBinaryTree ();

int maxDepth = solution.maxDepth(root);

System.out.println("Maximum Depth of the Binary Tree: " + maxDepth);

}

}

****

**Time Complexity:**  O(n)