**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.

Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

Code:

import java.util.\*;

public class MaxSubArrSum {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

int maxSum = Integer.MIN\_VALUE, currSum = 0;

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

currSum += arr[i];

if(currSum > maxSum){

maxSum = currSum;

}

if(currSum < 0){

currSum = 0;

}

}

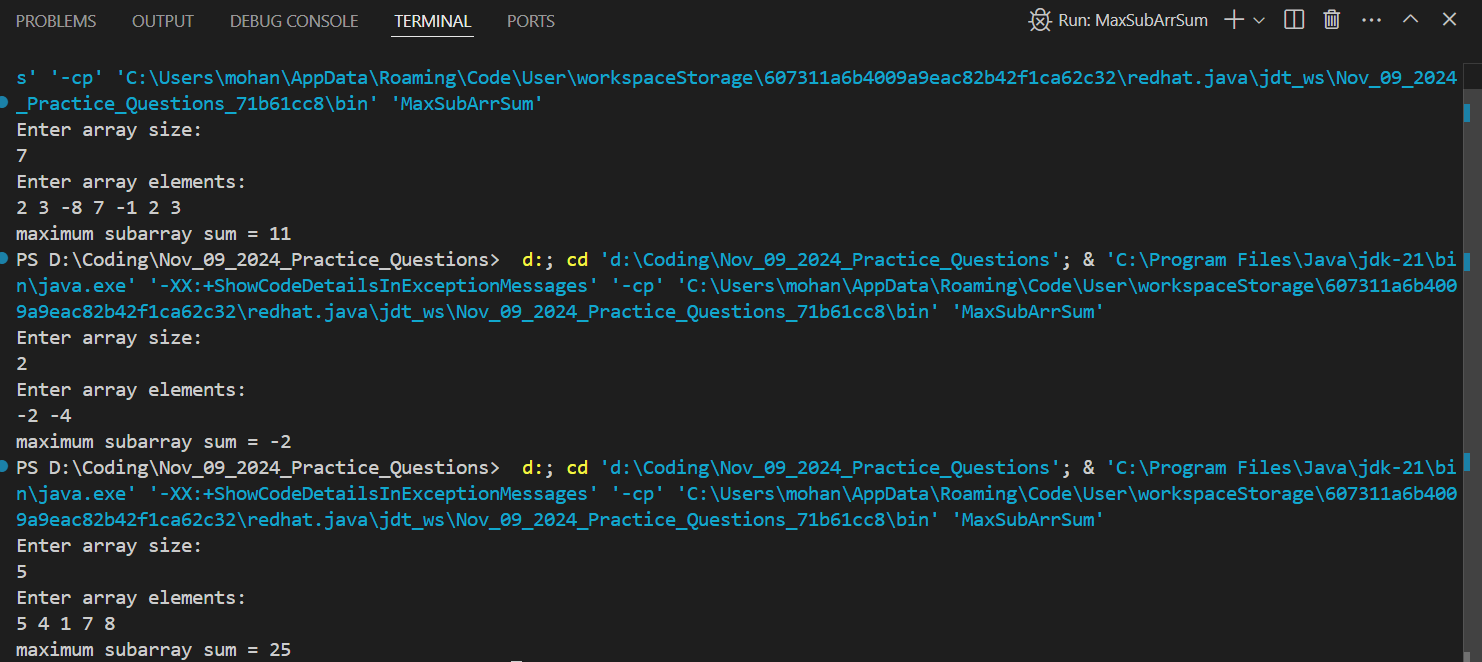
System.out.println("maximum subarray sum = " + maxSum);

sc.close();

}

}

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

Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

**Code:**

import java.util.\*;

public class MaxProductArr {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

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

sc.close();

}

static int max(int a, int b, int c) {

return Math.max(a, Math.max(b, c));

}

static int min(int a, int b, int c) {

return Math.min(a, Math.min(b, c));

}

static int maxProduct(int[] arr) {

int n = arr.length;

int currMax = arr[0];

int currMin = arr[0];

int maxProd = arr[0];

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

int temp = max(arr[i], arr[i] \* currMax, arr[i] \* currMin);

currMin = min(arr[i], arr[i] \* currMax, arr[i] \* currMin);

currMax = temp;

maxProd = Math.max(maxProd, currMax);

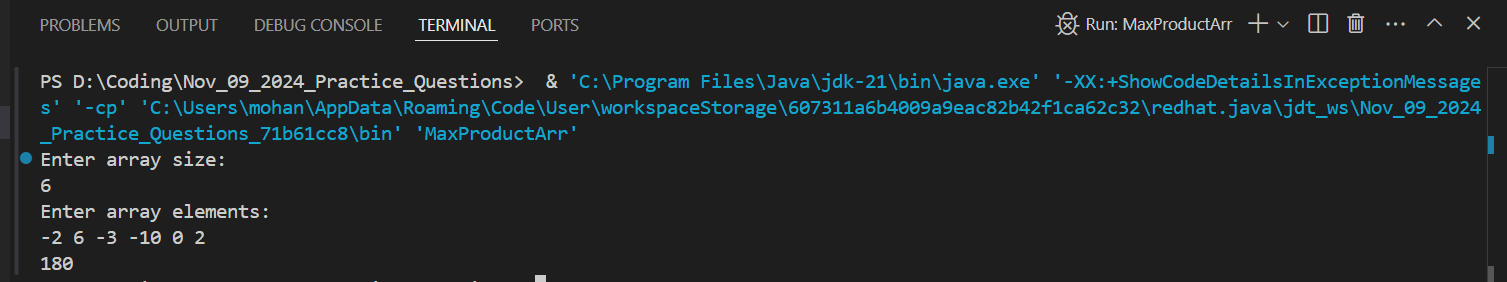
}

return maxProd;

}

}

**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

Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

Output : -1

Input : arr[] = {50, 10, 20, 30, 40}, key = 10

Output : 1

**Code:**

import java.util.\*;

public class SearchInRotatedSortedArr {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

System.out.println("Enter key to search:");

int key = sc.nextInt();

System.out.println(search(arr, key));

sc.close();

}

public static int search(int[] nums, int target) {

int pivot = findPivot(nums, 0, nums.length - 1);

int resIndex = 0;

if (pivot == -1) {

resIndex = binarySearch(nums, 0, nums.length - 1, target);

} else {

if (nums[pivot] == target) {

return pivot;

} else {

if (target >= nums[0]) {

resIndex = binarySearch(nums, 0, pivot - 1, target);

} else {

resIndex = binarySearch(nums, pivot + 1, nums.length - 1, target);

}

}

}

return resIndex;

}

private static int findPivot(int[] arr, int low, int high) {

if (high < low) {

return -1;

}

if (high == low) {

return low;

}

int mid = low + (high - low) / 2;

if (mid < high && arr[mid] > arr[mid + 1]) {

return mid;

}

if (mid > low && arr[mid] < arr[mid - 1]) {

return mid - 1;

}

if (arr[low] >= arr[mid]) {

return findPivot(arr, low, mid - 1);

}

return findPivot(arr, mid + 1, high);

}

private static int binarySearch(int[] arr, int low, int high, int target) {

if (high < low) {

return -1;

}

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

return mid;

}

if (target < arr[mid]) {

return binarySearch(arr, low, mid - 1, target);

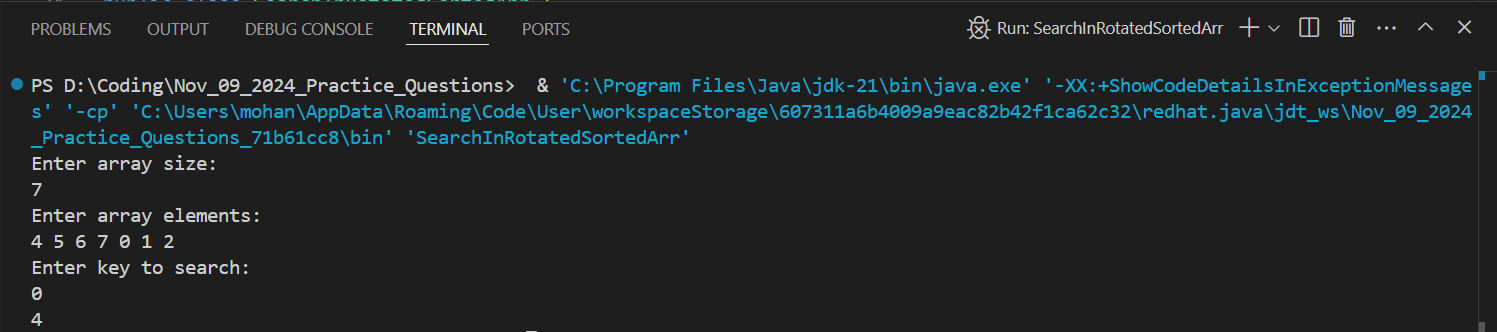
}

return binarySearch(arr, mid + 1, high, target);

}

}

**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

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.\*;

public class ContainerWithMostWater {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

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

sc.close();

}

public static int maxArea(int[] height) {

int maxWater = 0;

int left = 0, right = height.length - 1;

while (left < right) {

int water = Math.min(height[left], height[right]) \* (right - left);

maxWater = Math.max(maxWater, water);

if (height[left] < height[right])

left++;

else

right--;

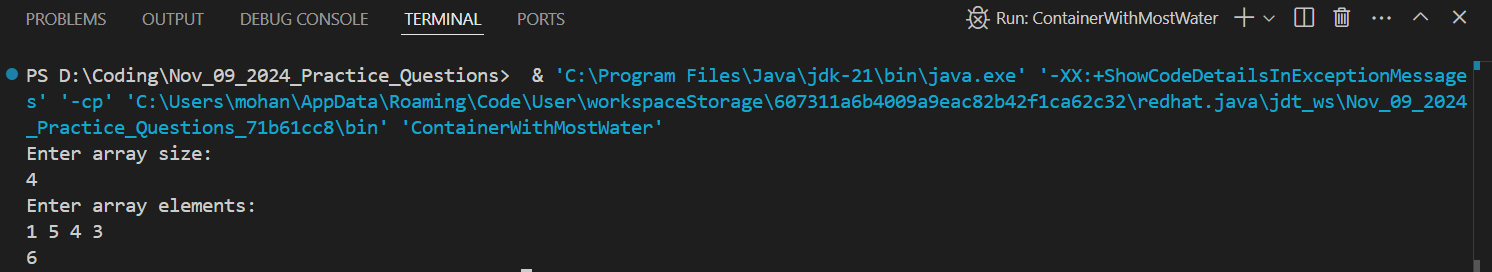
}

return maxWater;

}

}

Output:

  
  
Time Complexity: O(n)

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

Input: 100

Output: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**Code:**

import java.util.\*;

public class FactorialOfLargeNumber {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a large number:");

int n = sc.nextInt();

fact(n);

sc.close();

}

private static void fact(int n){

int[] res = new int[500];

res[0] = 1;

int resSize = 1;

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

resSize = mul(i, res, resSize);

}

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

System.out.print(res[i]);

}

}

private static int mul(int num, int[] res, int resSize){

int carry = 0;

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

int prod = res[i] \* num + carry;

res[i] = prod % 10;

carry = prod / 10;

}

while(carry != 0){

res[resSize] = carry % 10;

carry /= 10;

resSize++;

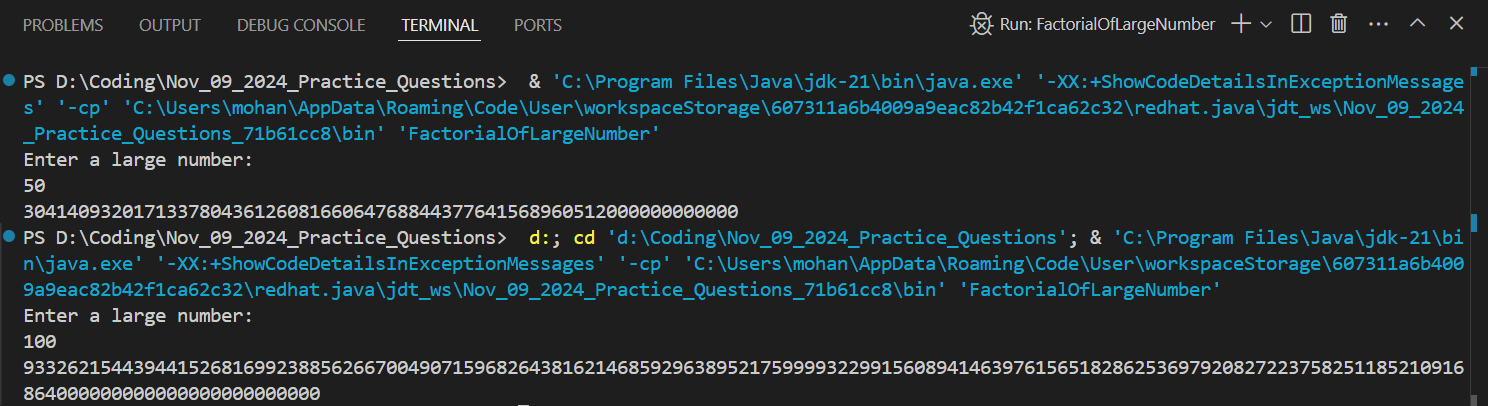
}

return resSize;

}

}

**Output:**

****

**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.

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

Output: 10

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

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

Output: 7

Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

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

Output: 0

Explanation : We cannot trap water as there is no height bound on both sides

Input: arr[] = {10, 9, 0, 5}

Output: 5 Explanation : We trap 0 + 0 + 5 + 0 = 5

**Code:**

import java.util.\*;

public class TrappingRainWater {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

int[] leftAssurity = new int[n];

int[] rightAssurity = new int[n];

leftAssurity[0] = -1;

rightAssurity[n-1] = -1;

int lmax = arr[0];

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

leftAssurity[i] = lmax;

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

}

int rmax = arr[n-1];

for(int i=n-2; i>=0; i--){

rightAssurity[i] = rmax;

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

}

int trappingWater = 0;

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

int minPossibleHeight = Math.min(leftAssurity[i], rightAssurity[i]);

if(arr[i] >= minPossibleHeight) continue;

else{

trappingWater += minPossibleHeight - arr[i];

}

}

System.out.println("Trapping Water = " + trappingWater);

sc.close();

}

}

**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.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5

Output: 7

Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

**Code:**

import java.util.\*;

public class ChocolateDistribution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter number of chocolates in each packet:");

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

arr[i] = sc.nextInt();

}

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

int m = sc.nextInt();

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

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

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

minDiff = Math.min(minDiff, diff);

}

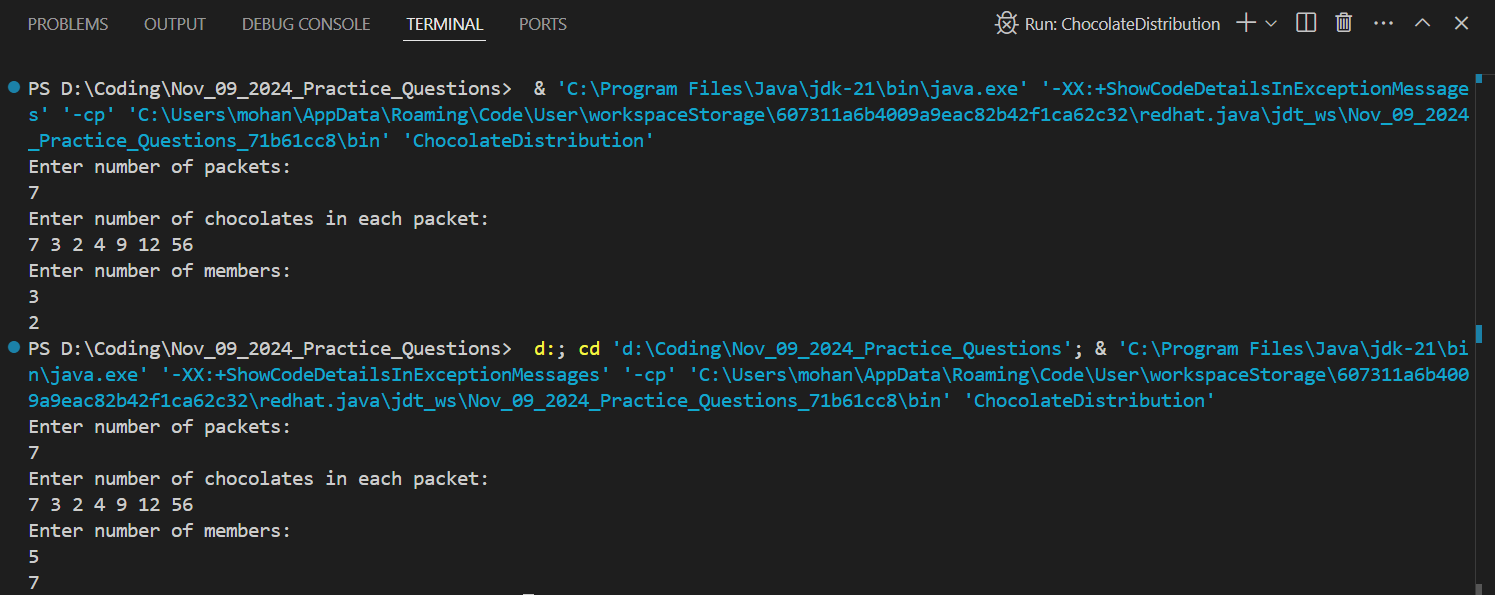
System.out.println(minDiff);

sc.close();

}

}

**Output:**

****

**Time Complexity:** O(n log 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.

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.\*;

public class MergeIntervals {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of intervals to be merged:");

int m = sc.nextInt();

int[] arr = new int[m \* 2];

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

for(int i=0; i<m\*2; i++){

arr[i] = sc.nextInt();

}

char[] helper = new char[m \* 2];

for(int i=0; i<m\*2; i++){

if((i & 1) == 0)

helper[i] = 's';

else

helper[i] = 'd';

}

for(int i=0; i<(m\*2)-1; i++){

for(int j=i+1; j<(m\*2); j++){

if(arr[i] > arr[j] || (arr[i] == arr[j] && helper[i] == 'd' && helper[j] == 's')){

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

char temp2 = helper[i];

helper[i] = helper[j];

helper[j] = temp2;

}

}

}

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

int ptr = 0;

for(int i=0; i<(m\*2); i++){

if(ptr == 0 && helper[i] == 's' || ptr == 1 && helper[i] == 'd')

res.add(arr[i]);

if(helper[i] == 's') ptr++;

else ptr--;

}

for(int elem : res){

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

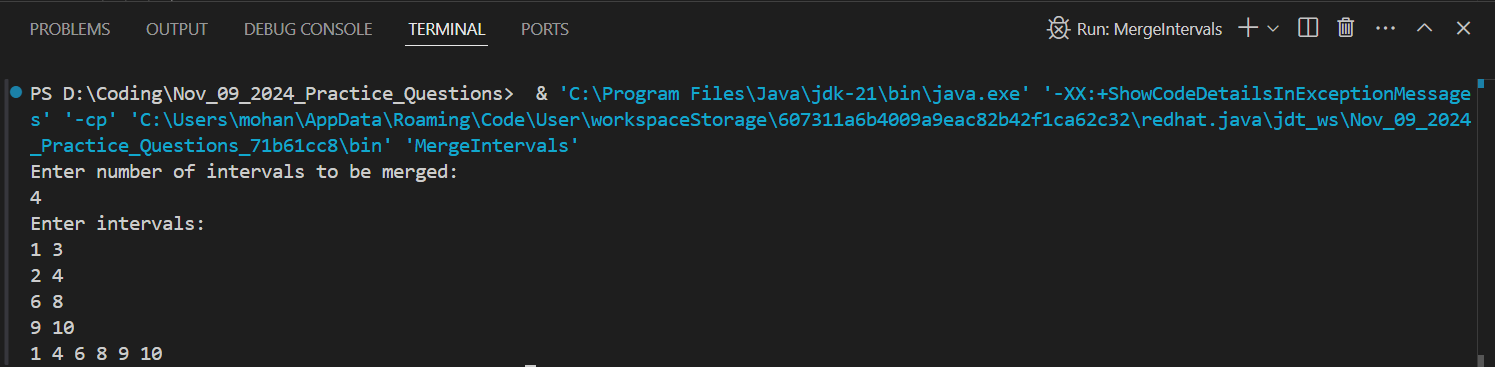
}

sc.close();

}

}

**Output:**

****

**Time Complexity: O(n^2)**

**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}}

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

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

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

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

**Code:**

import java.util.\*;

public class SetOnes {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter m: ");

int m = sc.nextInt();

System.out.println("Enter n: ");

int n = sc.nextInt();

int[][] matrix = new int[m][n];

System.out.println("Enter matrix elements(either 0 or 1):");

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

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

matrix[i][j] = sc.nextInt();

}

}

boolean fr = false, fc = false;

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

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

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

if(i == 0) fr = true;

if(j == 0) fc = true;

matrix[i][0] = 1;

matrix[0][j] = 1;

}

}

}

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

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

if(matrix[i][0] == 1 || matrix[0][j] == 1){

matrix[i][j] = 1;

}

}

}

if(fr){

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

matrix[0][j] = 1;

}

}

if(fc){

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

matrix[i][0] = 1;

}

}

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

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

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

}

System.out.println();

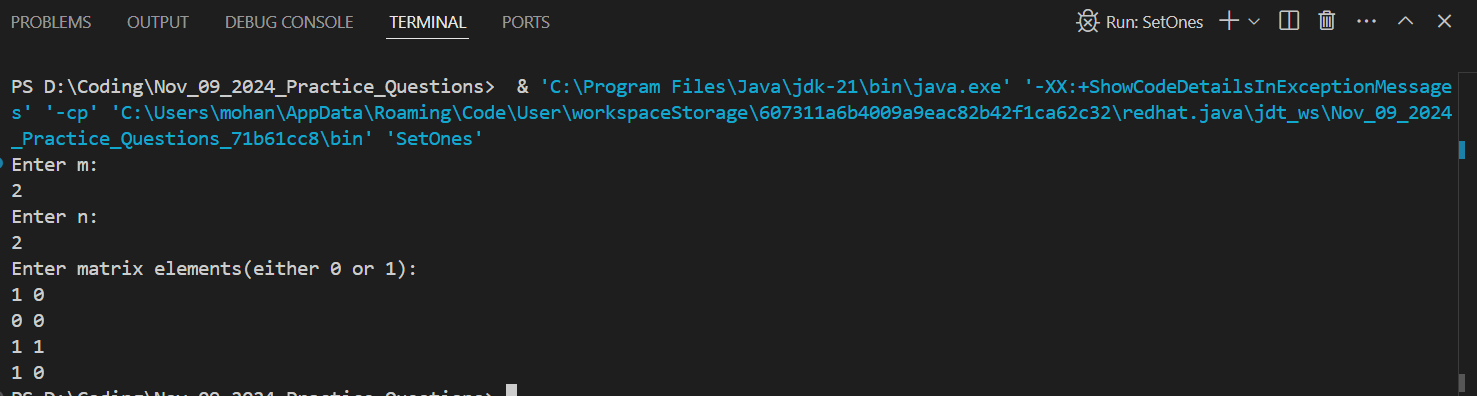
}

sc.close();

}

}

**Output:**



**Time Complexity: O(m \* 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

Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}}

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

Explanation: The output is matrix in spiral format

**Code:**

import java.util.\*;

public class SpiralMatrix {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter m:");

int m = sc.nextInt();

System.out.println("Enter n:");

int n = sc.nextInt();

int[][] matrix = new int[m][n];

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

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

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

matrix[i][j] = sc.nextInt();

}

}

int xSteps = m-1, ySteps = n, x = 0, y = 0; //xsteps and ysteps indicates no.of steps can be moved in row wise and col wise fashion

char dir = 'r'; //denominate right direction

while(xSteps > 0 || ySteps > 0){

if(dir == 'r'){

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

System.out.print(matrix[x][y++] + " ");

}

y--;

x++;

dir = 'd';

ySteps--;

}

else if(dir == 'd'){

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

System.out.print(matrix[x++][y] + " ");

}

x--;

y--;

dir = 'l';

xSteps--;

}

else if(dir == 'l'){

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

System.out.print(matrix[x][y--] + " ");

}

x--;

y++;

dir = 'u';

ySteps--;

}

else if(dir == 'u'){

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

System.out.print(matrix[x--][y] + " ");

}

x++;

y++;

dir = 'r';

xSteps--;

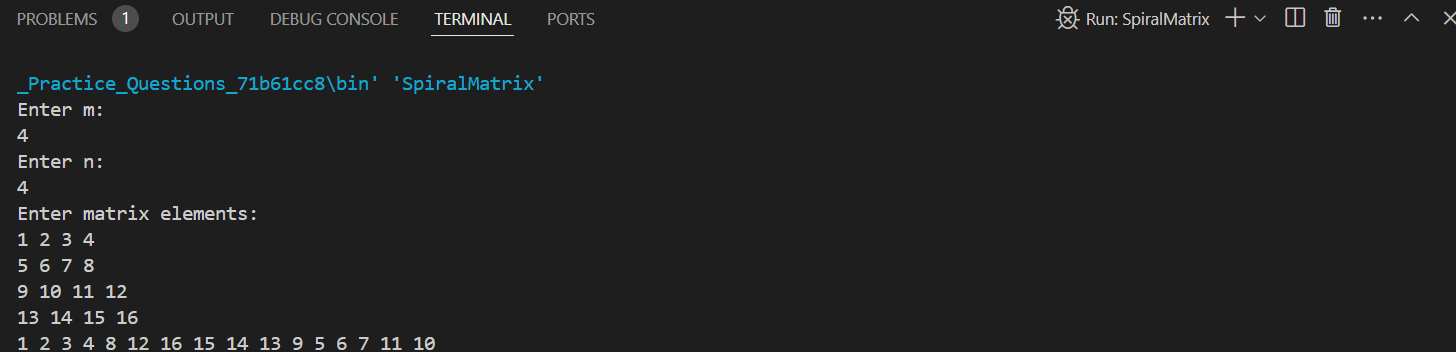
}

}

}

}

**Output:**

****

**Time Complexity: O(m \* n)**

**Space Complexity: O(m \* n)**

**11) Question is not there**

**12) Question is not there**

**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

Input: str = “())((())”

Output: Not Balanced

**Code:**

import java.util.\*;

public class BalancedParanthesis {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String str = sc.nextLine();

int ptr = 0;

boolean flag = true;

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

if(ch == '(')

ptr++;

else

ptr--;

if(ptr < 0){

flag = false;

break;

}

}

if(ptr != 0)

flag = false;

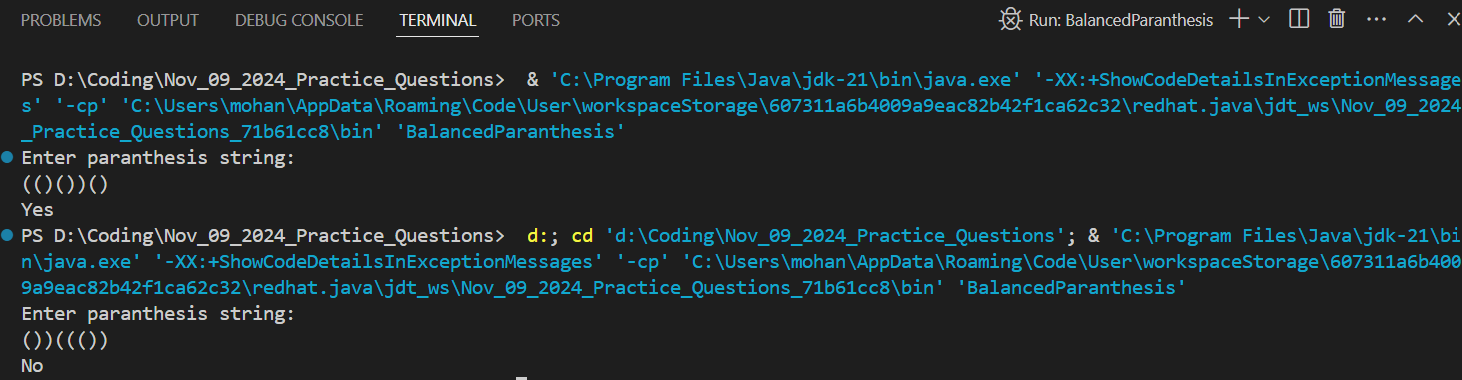
System.out.println(flag ? "Yes" : "No");

sc.close();

}

}

**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.

Input: s1 = “allergy” s2 = “allergic”

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams.

Input: s1 = “g”, s2 = “g”

Output: true

Explanation: Characters in both the strings are same, so they are anagrams.

**Code:**

import java.util.\*;

public class Anagrams {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter string1:");

String str1 = sc.nextLine();

System.out.println("Enter string2: ");

String str2 = sc.nextLine();

int[] alpha1 = new int[26];

int[] alpha2 = new int[26];

for(char ch : str1.toCharArray()){

alpha1[ch - 'a']++;

}

for(char ch : str2.toCharArray()){

alpha2[ch - 'a']++;

}

boolean flag = true;

for(int i=0; i<26; i++){

if(alpha1[i] != alpha2[i]){

flag = false;

break;

}

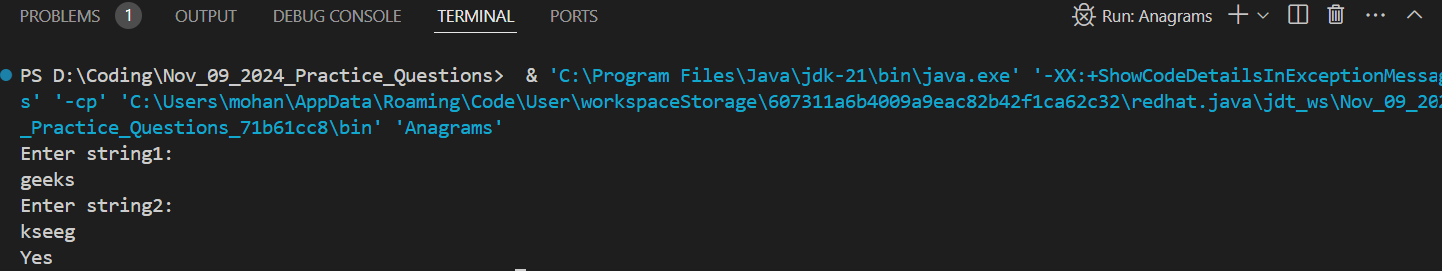
}

System.out.println(flag ? "Yes" : "No");

}

}

**Output:**

****

**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.

Input: str = “Geeks”

Output: “ee”

Input: str = “abc”

Output: “a”

Input: str = “”

Output: “”

**Code:**

import java.util.\*;

public class LongestPalindrome {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String str = "forgeeksskeegfor";

String maxStr = str.substring(0, 1);

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

String odd = expandCenter(str, i, i);

String even = expandCenter(str, i, i+1);

if(odd.length() > maxStr.length()){

maxStr = odd;

}

if(even.length() > maxStr.length()){

maxStr = even;

}

}

System.out.println(maxStr);

sc.close();

}

private static String expandCenter(String str, int left, int right){

while(left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)){

left--;

right++;

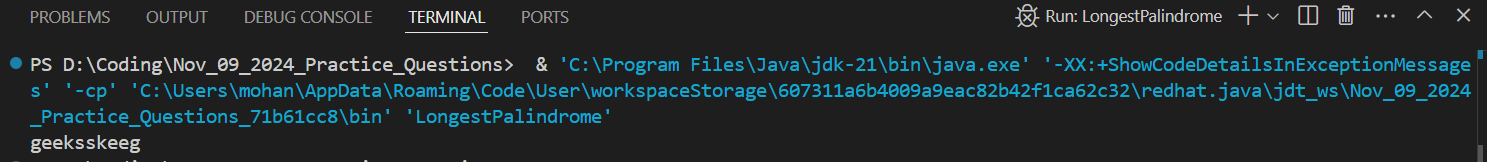
}

return str.substring(left+1, right);

}

}

Output:



**Time Complexity:** O(n^2)

**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.

Input: arr[] = [“hello”, “world”]

Output: -1

Explanation: There‟s no common prefix in the given strings.

**Code:**

import java.util.\*;

class Node{

Node[] links = new Node[26];

boolean flag = false;

boolean containsKey(char ch){

return links[ch - 'a'] != null;

}

void put(char ch, Node node){

links[ch - 'a'] = node;

}

Node get(char ch){

return links[ch - 'a'];

}

boolean isEnd(){

return flag;

}

void setEnd(){

flag = true;

}

}

class Trie{

Node root;

public Trie(){

root = new Node();

}

public int insert(String word){

Node currNode = root;

int reducedOverHead = 0;

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

char ch = word.charAt(i);

if(!currNode.containsKey(ch)){

currNode.put(ch, new Node());

}

else{

reducedOverHead++;

}

currNode = currNode.get(ch);

}

currNode.setEnd();

return reducedOverHead;

}

}

public class LongestCommonPrefix {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine();

String[] arr = new String[n];

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

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

arr[i] = sc.nextLine();

}

Trie trie = new Trie();

int minReducedOverHead = Integer.MAX\_VALUE;

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

int currReducedOverHead = trie.insert(arr[i]);

if(i != 0)

minReducedOverHead = Math.min(minReducedOverHead, currReducedOverHead);

}

if(minReducedOverHead == 0){

System.out.println("-1");

}

else{

System.out.println(arr[0].substring(0, minReducedOverHead));

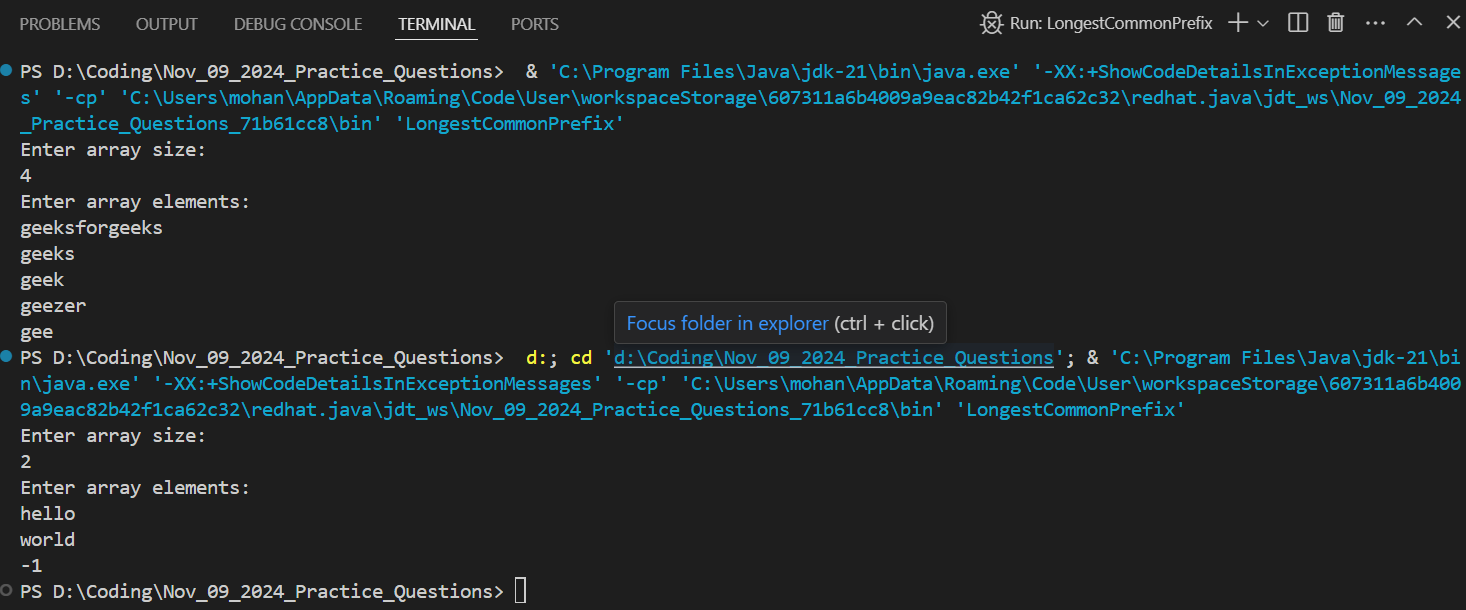
}

sc.close();

}

}

**Output:**

****

**Time Complexity: O(n \* m)** //n is the array length & m is each word length

**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]

**Input :** Stack[] = [1, 2, 3, 4, 5, 6]

**Output :** Stack[] = [1, 2, 4, 5, 6]

**Code**

import java.util.\*;

public class DeleteMidOfStack {

public static void main(String[] args) {

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

Scanner sc = new Scanner(System.in);

System.out.println("Enter stack size: ");

int n = sc.nextInt();

System.out.println("Enter stack Elements:");

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

st.push(sc.nextInt());

}

deleteMid(st, st.size(), 0);

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

while(!st.isEmpty()){

res.push(st.pop());

}

while(!res.isEmpty()){

System.out.print(res.pop() + " ");

}

sc.close();

}

private static void deleteMid(Stack<Integer> st, int n, int currIndex){

if(st.isEmpty() || currIndex == n)

return;

int currVal = st.pop();

deleteMid(st, n, currIndex + 1);

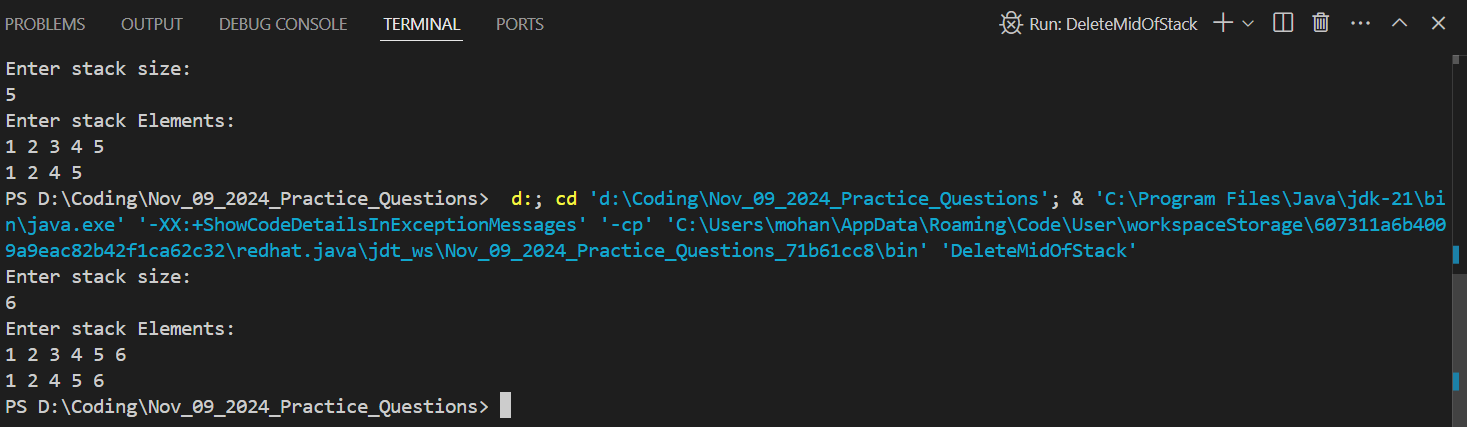
if(currIndex != n/2){

st.push(currVal);

}

}

} **Output:**

****

**Time Complexity:** O(n)

**Space Complexity:** O(n)

**18) Next Greater Element (NGE)**

for every element in given 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

**Input:** arr[] = [ 13 , 7, 6 , 12 ]

**Output:** 13 –> -1 7 –> 12 6 –> 12 12 –> -1

**Explanation:** 13 and 12 don’t have any element greater than them present on the right side

**Code:**

import java.util.\*;

public class NextGreater {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

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

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

arr[i] = sc.nextInt();

}

int res[] = new int[n];

Arrays.fill(res, -1);

findNextGreater(arr, res);

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

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

sc.close();

}

private static void findNextGreater(int[] arr, int[] res){

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

for(int i=arr.length-1; i>=0; i--){

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

st.pop();

}

if(!st.isEmpty()){

res[i] = st.peek();

}

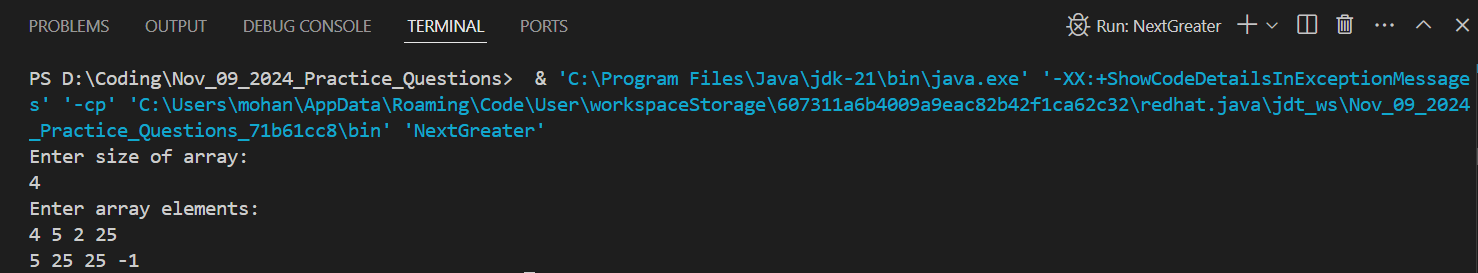
st.push(arr[i]);

}

}

}

**Output:**

****

**Time Complexity:** O(n)

**Space 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:**

class TreeNode {

int val;

TreeNode left;

TreeNode right;

public TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

public class RightViewBT {

static int maxDepth = 0;

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);

printRightView(root, 1);

}

public static void printRightView(TreeNode root, int depth){

if(root == null)

return;

if(depth > maxDepth){

System.out.println(root.val + " ");

maxDepth = depth;

}

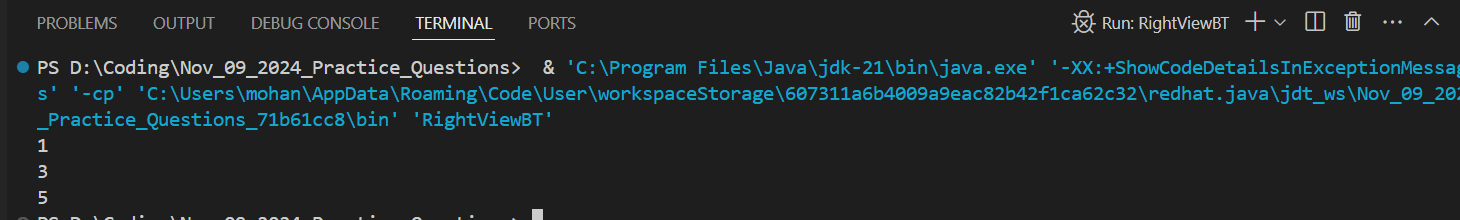
printRightView(root.right, depth+1);

printRightView(root.left, depth+1);

}

}

Output:



**Time Complexity:** O(n)

**Space 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;

public TreeNode(int val) {

this.val = val;

left = null;

right = null;

}

}

public class BTDepth {

static int maxDepth = 0;

public static void main(String[] args) {

TreeNode root = new TreeNode(12);

root.left = new TreeNode(8);

root.right = new TreeNode(18);

root.left.left = new TreeNode(5);

root.left.right = new TreeNode(11);

dfs(root, 1);

System.out.println(maxDepth);

}

public static void dfs(TreeNode root, int depth){

if(root == null)

return;

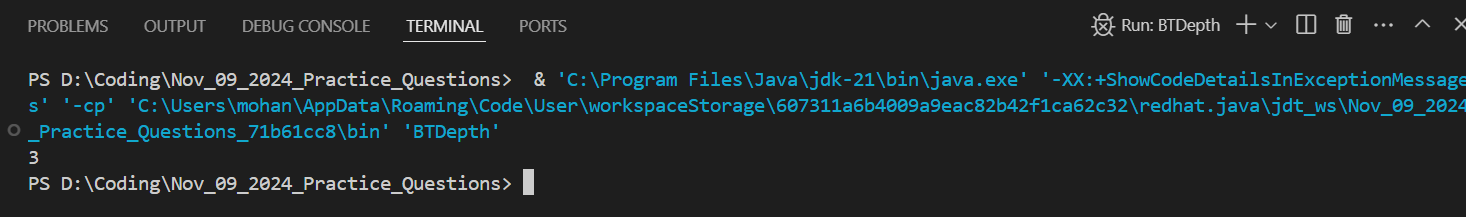
maxDepth = Math.max(maxDepth, depth);

dfs(root.left, depth+1);

dfs(root.right, depth+1);

}

}

Output:  
  


**Time Complexity:** O(n)

**Space Complexity:** O(n)