**09/11/2024** **DSA Technical Training Practice 1**

-Rithikka J AI&DS

Question 1:

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.

Program:

```
import java.util.*;

public class MaximumSubarraySum{
public static void main(String[]args){
Scanner sc = new Scanner(System.in);
System.out.println("Enter the arrary:");
String input = sc.nextLine();
String[] n = input.split(" ");
int[] arr = new int[n.length];
for (int i = 0; i < n.length; i++) {
arr[i] = Integer.parseInt(n[i]);
}
int maxSum = maxSubarraySum(arr);
System.out.println(maxSum);
sc.close();
}
public static int maxSubarraySum(int[] arr) {
int maxEnd = arr[0];
int res = arr[0];
for (int i = 1; i < arr.length; i++) {
maxEnd = Math.max(arr[i], maxEnd + arr[i]);
res = Math.max(res, maxEnd);
}
    return res;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java MaximumSubarraySum
Enter the arrary:
2 3 -8 7 -1 2 3
11

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java MaximumSubarraySum
Enter the arrary:
5 4 1 7 8
25
```

Time complexity: O(n)

Question 2:

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

Program:

```
import java.util.*;

public class MaximumSubarrayProduct{
public static void main(String[]args){
Scanner sc = new Scanner(System.in);
System.out.println("Enter the arrary:");
String input = sc.nextLine();
String[] n = input.split(" ");
int[] arr = new int[n.length];
for (int i = 0; i < n.length; i++) {
arr[i] = Integer.parseInt(n[i]);
}
int maxProduct = maxSubarrayProduct(arr);
System.out.println(maxProduct);
sc.close();
}
public static int maxSubarrayProduct(int[] arr) {
int maxEnd = arr[0];
int minEnd = arr[0];
int res = arr[0];
for (int i = 1; i < arr.length; i++) {
if(arr[i]<0){
int temp = maxEnd;
maxEnd = minEnd;
minEnd = temp;
}
maxEnd = Math.max(arr[i], maxEnd * arr[i]);
minEnd = Math.min(arr[i], minEnd * arr[i]);
res = Math.max(res, maxEnd);
}
return res;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java MaximumSubarrayProduct
Enter the arrary:
-2 6 -3 -10 0 2
180

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java MaximumSubarrayProduct
Enter the arrary:
-1 -3 0 -10 60
60
```

Time complexity: O(n)

Question 3:

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

Program:

```
import java.util.*;
public class SearchInRotatedArray{
public static void main(String[]args){
int[] arr1={4,5,6,7,0,1,2};
int key1=0;
System.out.println("Test Case 1: " + search(arr1, key1));
int[] arr2 = {4, 5, 6, 7, 0, 1, 2};
int key2 = 3;
System.out.println("Test Case 2: " + search(arr2, key2));
int[] arr3 = {50, 10, 20, 30, 40};
int key3 = 10;
System.out.println("Test Case 3: " + search(arr3, key3));
}
public static int search(int[] arr,int key){
int left=0,right=arr.length - 1;
while(left<=right){
int mid=left+(right-left)/2;
if(arr[mid]==key){
return mid;
}
if(arr[left]<=arr[mid]){
if(arr[left]<=key && key <arr[mid]){
right=mid - 1;
}
else{
left= mid+1;
}
}
else{
if(arr[mid]<key && key<= arr[right]){
left=mid+1;
}
else{
right=mid-1;
}
}
}
return -1;

}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java SearchInRotatedArray
Test Case 1: 4
Test Case 2: -1
Test Case 3: 1
```

Time complexity: O(log n)

Question 4:

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

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

***Note:*** You may not slant the container.

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

Program:

```
import java.util.*;
public class Solution {
public static void main(String[] args) {
int[] arr1 = {1, 5, 4, 3};
int[] arr2 = {3, 1, 2, 4, 5};
System.out.println("Max area for arr1: " + maxArea(arr1));
System.out.println("Max area for arr2: " + maxArea(arr2));
}
 public static int maxArea(int[] height) {
int maxArea = 0;
int left = 0;
int right = height.length - 1;
while (left < right) {
int width = right - left;
int currentHeight = Math.min(height[left], height[right]);
int currentArea = width * currentHeight;
maxArea = Math.max(maxArea, currentArea);
if (height[left] < height[right]) {
left++;
}
else
{
right--;
}
}
return maxArea;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java Solution
Max area for arr1: 6
Max area for arr2: 12
```

Time complexity: O(n)

Question 5:

5. Find the Factorial of a large number
   Input: 100
   Output:
   933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000000

   Input: 50
   Output: 30414093201713378043612608166064768844377641568960512000000000000

Program:

```
import java.math.BigInteger;
import java.util.Scanner;

public class Factorial5 {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number:");
int n = sc.nextInt();
if (n == 0 || n == 1) {
System.out.println("Factorial is 1");
}
else {
BigInteger fact = BigInteger.ONE;
for (int i = 2; i <= n; i++) {
fact = fact.multiply(BigInteger.valueOf(i));
}
System.out.println("Factorial is " + fact);
}
sc.close();
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java Factorial5
enter the number
50
Factorial is30414093201713378043612608166064768844377641568960512000000000000

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java Factorial5
enter the number
100
Factorial is933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

Time complexity: O(n)

Question 6:

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

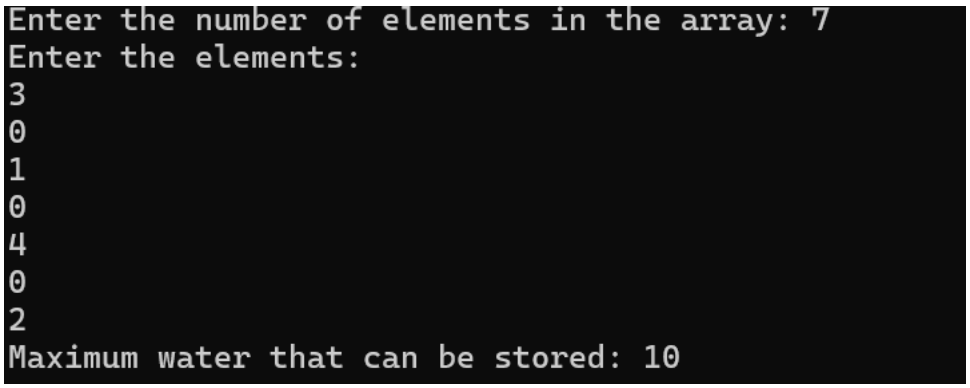
Input: arr[] = {10, 9, 0, 5}
Output: 5
Explanation : We trap 0 + 0 + 5 + 0 = 5

Program:

```
import java.util.Scanner;

public class Rain {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of elements in the array: ");
int n = sc.nextInt();
int[] arr = new int[n];
System.out.println("Enter the elements:");
for (int i = 0; i < n; i++) {
arr[i] = sc.nextInt();
}
int maxWaterStored = maxWater(arr);
System.out.println("Maximum water that can be stored: " + maxWaterStored);
}
static int maxWater(int[] arr) {
int n = arr.length;
if (n <= 2) return 0;
int[] leftMax = new int[n];
int[] rightMax = new int[n];
leftMax[0] = arr[0];
for (int i = 1; i < n; i++) {
leftMax[i] = Math.max(leftMax[i - 1], arr[i]);
}
rightMax[n - 1] = arr[n - 1];
for (int i = n - 2; i >= 0; i--) {
rightMax[i] = Math.max(rightMax[i + 1], arr[i]);
}
int res = 0;
for (int i = 1; i < n - 1; i++) {
res += Math.min(leftMax[i], rightMax[i]) - arr[i];
}
return res;
}
}
```

Output:




Time complexity: O(n)

Question 7:

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.

Program:

```
import java.util.*;
public class Chocolate {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the array elements :");
String input = sc.nextLine();
String[] inputArr = input.split(" ");
int[] arr = new int[inputArr.length];
for (int i = 0; i < inputArr.length; i++) {
arr[i] = Integer.parseInt(inputArr[i]);
}
System.out.print("Enter no.of students: ");
int m = sc.nextInt();
int result = findMinDiff(arr, m);
if (result == -1) {
System.out.println("Not enough packets to distribute.");
}
else
{
System.out.println("Minimum difference is: " + result);
}
}
static int findMinDiff(int[] arr, int m) {
if (m == 0 || arr.length == 0) return 0;
if (arr.length < m) return -1;
Arrays.sort(arr);
int minDiff = Integer.MAX_VALUE;
for (int i = 0; i + m - 1 < arr.length; i++) {
int diff = arr[i + m - 1] - arr[i];
if (diff < minDiff) {
minDiff = diff;
}
}
return minDiff;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java Chocolate
Enter the array elements :
7 3 2 4 9 12 56
Enter no.of students: 3
Minimum difference is: 2

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java Chocolate
Enter the array elements :
7 3 4 9 12 56
Enter no.of students: 5
Minimum difference is: 9
```

Time complexity: O(n logn)

Question 8:

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]].

Program:

Output:

Time complexity:

Question 9:

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}}

Program:

```
import java.util.*;
public class BooleanMatrix {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the number of rows and columns:");
int rows = sc.nextInt();
int cols = sc.nextInt();
int[][] mat = new int[rows][cols];
System.out.println("Enter the matrix elements (0 or 1):");
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
mat[i][j] = sc.nextInt();
}
}
modifyMatrix(mat);
System.out.println("Modified Matrix:");
printMatrix(mat);
}
static void modifyMatrix(int[][] mat) {
int rows = mat.length;
int cols = mat[0].length;
boolean rowFlag = false, colFlag = false;
for (int j = 0; j < cols; j++) {
if (mat[0][j] == 1) {
rowFlag = true;
break;
}
}
for (int i = 0; i < rows; i++) {
if (mat[i][0] == 1) {
colFlag = true;
break;
}
}
for (int i = 1; i < rows; i++) {
for (int j = 1; j < cols; j++) {
if (mat[i][j] == 1) {
mat[0][j] = 1;
mat[i][0] = 1;
}
}
}
for (int i = 1; i < rows; i++) {
for (int j = 1; j < cols; j++) {
if (mat[0][j] == 1 || mat[i][0] == 1) {
mat[i][j] = 1;
}
}
}
if (rowFlag) {
for (int j = 0; j < cols; j++) {
mat[0][j] = 1;
}
}
if (colFlag) {
for (int i = 0; i < rows; i++) {
mat[i][0] = 1;
}
}
}
static void printMatrix(int[][] mat) {
for (int[] row : mat) {
for (int elem : row) {
System.out.print(elem + " ");
}
System.out.println();
}
}
}
```

Output:

```
Enter the number of rows and columns:
3
3
Enter the matrix elements (0 or 1):
0 0 1
1 0 1
1 1 0
Modified Matrix:
1 1 1
1 1 1
1 1 1

C:\Users\jrith\OneDrive\Desktop\Data structure
Enter the number of rows and columns:
2
2
Enter the matrix elements (0 or 1):
1 0
0 1
Modified Matrix:
1 1
1 1
```

Time complexity: O(M*N)

Question 10:

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.

Program:

```
import java.util.*;
public class SpiralMatrix {
public static void main(String[] args) {
int[][] matrix1 = {
{1, 2, 3, 4},
{5, 6, 7, 8},
{9, 10, 11, 12},
{13, 14, 15, 16}
};
int[][] matrix2 = {
{1, 2, 3, 4, 5, 6},
{7, 8, 9, 10, 11, 12},
{13, 14, 15, 16, 17, 18}
};
System.out.println("Spiral Order of Matrix 1:");
printSpiral(matrix1);
System.out.println("Spiral Order of Matrix 2:");
printSpiral(matrix2);
}

public static void printSpiral(int[][] matrix) {
int top = 0, bottom = matrix.length - 1;
int left = 0, 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++;
}
}
System.out.println();
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java SpiralMatrix
Spiral Order of Matrix 1:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Spiral Order of Matrix 2:
1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
```

Time complexity:O(M*N)

Question 13:

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

Program:

```
import java.util.*;

public class BalancedParentheses {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter a string of parentheses:");
String str = sc.nextLine();
System.out.println("Input: " + str);
System.out.println("Output: " + (isBalanced(str) ? "Balanced" : "Not Balanced"));
}

public static boolean isBalanced(String str) {
Stack<Character> stack = new Stack<>();
for (char ch : str.toCharArray()) {
if (ch == '(') {
stack.push(ch);
} else if (ch == ')') {
if (stack.isEmpty() || stack.pop() != '(') {
return false;
}
}
}
return stack.isEmpty();
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Tra
Enter a string of parentheses:
)(()))(())((
Input: )(()))(())((
Output: Not Balanced

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Tra
Enter a string of parentheses:
(((()))()()()
Input: (((()))()()()
Output: Balanced
```

Time complexity: O(n)

Question 14:

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.

Program:

```
import java.util.Arrays;
import java.util.Scanner;

public class AnagramCheck {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the first string:");
String s1 = sc.nextLine();
System.out.println("Enter the second string:");
String s2 = sc.nextLine();
System.out.println("Input: " + s1 + ", " + s2);
System.out.println("Output: " + areAnagrams(s1, s2));
}

public static boolean areAnagrams(String s1, String s2) {
if (s1.length() != s2.length()) {
return false;
}
char[] s1Array = s1.toCharArray();
char[] s2Array = s2.toCharArray();
Arrays.sort(s1Array);
Arrays.sort(s2Array);
return Arrays.equals(s1Array, s2Array);
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java AnagramCheck
Enter the first string:
geeks
Enter the second string:
keesg
Input: geeks, keesg
Output: true

C:\Users\jrith\OneDrive\Desktop\Data structure and algorithm\Training dsa>java AnagramCheck
Enter the first string:
rithi
Enter the second string:
adrit
Input: rithi, adrit
Output: false
```

Time complexity: O(n logn)

## Question 15:

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: ""

## Program:

```
public class LongestPalindrome {
public static void main(String[] args) {
String str1 = "forgeeksskeegfor";
String str2 = "Geeks";
String str3 = "abc";
String str4 = "";

System.out.println("Input: " + str1);
System.out.println("Output: " + longestPalindrome(str1));

System.out.println("Input: " + str2);
System.out.println("Output: " + longestPalindrome(str2));

System.out.println("Input: " + str3);
System.out.println("Output: " + longestPalindrome(str3));

System.out.println("Input: " + str4);
System.out.println("Output: " + longestPalindrome(str4));
}

public static String longestPalindrome(String s) {
int n = s.length();
if (n == 0) return "";
boolean[][] dp = new boolean[n][n];
int start = 0, maxLength = 1;
for (int i = 0; i < n; i++) {
dp[i][i] = true;
}
for (int i = 0; i < n - 1; i++) {
if (s.charAt(i) == s.charAt(i + 1)) {
dp[i][i + 1] = true;
start = i;
maxLength = 2;
}
}

for (int length = 3; length <= n; length++) {
for (int i = 0; i < n - length + 1; i++) {
int j = i + length - 1;
if (dp[i + 1][j - 1] && s.charAt(i) == s.charAt(j)) {
dp[i][j] = true;
if (length > maxLength) {
start = i;
maxLength = length;
}
}
}
}
return s.substring(start, start + maxLength);
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop
Input: forgeeksskeegfor
Output: geeksskeeg
Input: Geeks
Output: ee
Input: abc
Output: a
```

Time complexity: O(n^2)

Question 16:

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.

Program:

```
public class LongestCommonPrefix {
public static void main(String[] args) {
String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
String[] arr2 = {"hello", "world"};

System.out.println("Input: " + String.join(", ", arr1));
System.out.println("Output: " + longestCommonPrefix(arr1));

System.out.println("Input: " + String.join(", ", arr2));
System.out.println("Output: " + longestCommonPrefix(arr2));
}

public static String longestCommonPrefix(String[] strs) {
if (strs == null || strs.length == 0) {
return "-1";
}
String prefix = strs[0];
for (int i = 1; i < strs.length; i++) {
while (strs[i].indexOf(prefix) != 0) {
prefix = prefix.substring(0, prefix.length() - 1);
if (prefix.isEmpty()) {
return "-1";
}
}
}
return prefix;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data struct
Input: geeksforgeeks, geeks, geek, geezer
Output: gee
Input: hello, world
Output: -1
```

Time complexity: O(M*N)

Question 17:

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]

Program:

```
import java.util.*;

public class DeleteMiddleElement {
public static void main(String[] args) {
Stack<Integer> stack1 = new Stack<>();
stack1.push(1);
stack1.push(2);
stack1.push(3);
stack1.push(4);
stack1.push(5);

System.out.println("Original Stack: " + stack1);
deleteMiddle(stack1, stack1.size());
System.out.println("Stack after deleting middle element: " + stack1);

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, stack2.size());
System.out.println("Stack after deleting middle element: " + stack2);
}

public static void deleteMiddle(Stack<Integer> stack, int size) {
if (stack.isEmpty() || size == 0) {
return;
}
int middle = size / 2;
deleteMiddleUtil(stack, middle);
}

private static void deleteMiddleUtil(Stack<Integer> stack, int middle) {
if (middle == 0) {
stack.pop();
return;
}
int temp = stack.pop();
deleteMiddleUtil(stack, middle - 1);
stack.push(temp);
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structure and alg
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)

Question 18:

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

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

Program:

```
import java.util.*;


public class NextGreaterElement {
public static void main(String[] args) {
int[] arr1 = {4, 5, 2, 25};
int[] arr2 = {13, 7, 6, 12};

System.out.println("Input: " + Arrays.toString(arr1));
printNextGreaterElements(arr1);

System.out.println("Input: " + Arrays.toString(arr2));
printNextGreaterElements(arr2);
}

public static void printNextGreaterElements(int[] arr) {
int n = arr.length;
int[] nge = new int[n];
Stack<Integer> stack = new Stack<>();

for (int i = 0; i < n; i++) {
while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
nge[stack.pop()] = arr[i];
}
stack.push(i);
}

while (!stack.isEmpty()) {
nge[stack.pop()] = -1;
}

for (int i = 0; i < n; i++) {
System.out.println(arr[i] + " -> " + nge[i]);
}
}
}
```

Output:

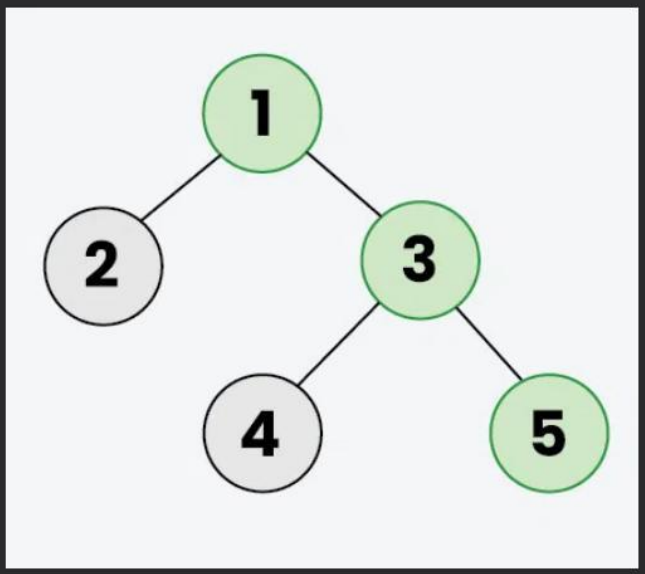
```
C:\Users\jrith\OneDrive
Input: [4, 5, 2, 25]
4 -> 5
5 -> 25
2 -> 25
25 -> -1
Input: [13, 7, 6, 12]
13 -> -1
7 -> 12
6 -> 12
12 -> -1
```

Time complexity: O(n)

Question 19:

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.

*Example 1: The **Green** colored nodes (1, 3, 5) represents the Right view in the below Binary tree.*



Program:

```
import java.util.*;

class Node {
int data;
Node left, right;

Node(int x) {
data = x;
left = right = null;
}
}

public class RightViewBinaryTree {
public static void main(String[] args) {
Node root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.right.left = new Node(4);
root.right.right = new Node(5);

List<Integer> rightView = rightView(root);
System.out.println("Right view of the Binary Tree: " + rightView);
}

public static List<Integer> rightView(Node root) {
List<Integer> result = new ArrayList<>();
if (root == null) return result;

Queue<Node> queue = new LinkedList<>();
queue.offer(root);

while (!queue.isEmpty()) {
int levelSize = queue.size();
Node current = null;

for (int i = 0; i < levelSize; i++) {
current = queue.poll();
if (current.left != null) queue.offer(current.left);
if (current.right != null) queue.offer(current.right);
}
result.add(current.data);
}
return result;
}
}
```

Output:

```
C:\Users\jrith\OneDrive\Desktop\Data structur

C:\Users\jrith\OneDrive\Desktop\Data structur
Right view of the Binary Tree: [1, 3, 5]
```

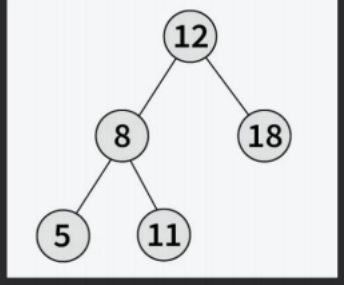
Time complexity: O(n)

Question 20:

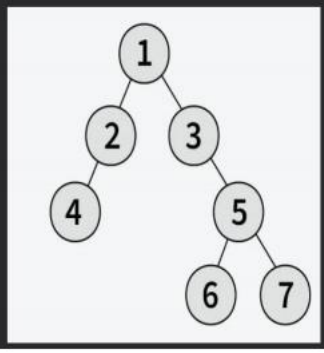
20. Maximum Depth or Height of Binary Tree

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.

*Example 1: The height of the below binary tree is 3.*




*Example 2: The height of the below binary tree is 4*



Program:

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

TreeNode(int val) {
this.val = val;
left = right = null;
}
}

public class MaxDepthBinaryTree {
public static void main(String[] args) {
TreeNode root1 = new TreeNode(1);
root1.left = new TreeNode(2);
root1.right = new TreeNode(3);
root1.left.left = new TreeNode(4);
root1.left.right = new TreeNode(5);

TreeNode root2 = new TreeNode(1);
root2.left = new TreeNode(2);
root2.right = new TreeNode(3);
root2.left.left = new TreeNode(4);
root2.left.right = new TreeNode(5);
root2.left.left.left = new TreeNode(6);

System.out.println("Max depth of tree 1: " + maxDepth(root1));
System.out.println("Max depth of tree 2: " + maxDepth(root2));
}

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:

```
C:\Users\jrith\OneDrive\Desktop\Data s
Max depth of tree 1: 3
Max depth of tree 2: 4
```

Time complexity:O(n)