09-11-2024

JAVA PROBLEMS – (SET - 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.

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

Output: 25

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

SOLUTION:

```
import java.util.*;
class Kadane{
	public static void main(String[] args){
		Scanner sc = new Scanner(System.in);
		int[] a=new int[30];
		int val = sc.nextInt();
		for(int i=0;i<val;i++){
			a[i] = sc.nextInt();
		}
		int sum = 0;
		int totalsum = Integer.MIN_VALUE;
		for(int i=0;i<val;i++){
			sum+=a[i];
			totalsum = Math.max(totalsum,sum);
			if(sum<0){
				sum=0;
			}
		}
		System.out.println(totalsum);
	}
```

}

OUTPUT:

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

D:\OOPS\09-11-2024 practice java>javac Kadane.java

D:\OOPS\09-11-2024 practice java>java Kadane
2
-2
-4
-2

D:\OOPS\09-11-2024 practice java>
```

Time Complexity = O(n)

Space Complexity = O(n) // because of taking user input otherwise O(1)

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}.

SOLUTION:

```
import java.util.*;

class KadaneProd{
	public static int maxproduct(int[] arr, int n){
		// if array length <= 0 just return the value 0;
		if(arr.length<=0) return 0;
		int maxproduct = arr[0];
		int minproduct = arr[0];
```

```
        int total = arr[0];
        for(int i=1;i<n;i++){
            if(arr[i]<0){
                int temp = maxproduct;
                maxproduct = minproduct;
                minproduct = temp;
            }
            maxproduct = Math.max(arr[i]*maxproduct, arr[i]);
            minproduct = Math.min(arr[i]*minproduct, arr[i]);
            total = Math.max(total, maxproduct);
        }
        return total;
}
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int[] a = new int[30];
        int val = sc.nextInt();
        for(int i=0;i<val;i++){
            a[i] = sc.nextInt();
        }
        System.out.println(KadaneProd.maxproduct(a,val));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac KadaneProd.java

D:\OOPS\09-11-2024 practice java>java KadaneProd
6
-2
6
-3
-10
0
2
180

D:\OOPS\09-11-2024 practice java>java KadaneProd
5
-1
-3
-10
0
60
60
```

Time Complexity = O(n)

Space Complexity = O(n) // because of taking user input otherwise O(1)

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

SOLUTION:

```
import java.util.*;

class Search{

        public static int find(int[] arr, int k){

                int l=0;
```

```
        int h=arr.length-1;
        while(l<=h){
            int mid = (l+h)/2;
            if(arr[mid] == k) return mid;
            // left sorted
            if(arr[l]<=arr[mid]){
                if(arr[l]<=k && arr[mid]>k)  h = mid-1;
                else l = mid+1;
            }
            //right sorted
            else{
                if(k>arr[mid] && k<=arr[h]) l = mid+1;
                else h = mid-1;
            }
        }
        return -1;
    }
    public static void main(String[] args){
        int arr1[] = {4, 5, 6, 7, 0, 1, 2};
        int key1 = 0;
        int arr2[] = {4, 5, 6, 7, 0, 1, 2 };
        int key2 = 3;
        int arr3[] = {50, 10, 20, 30, 40};
        int key3 = 10;
        System.out.println(find(arr1,key1));
        System.out.println(find(arr2,key2));
        System.out.println(find(arr3,key3));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac Search.java

D:\OOPS\09-11-2024 practice java>java Search
4
-1
1
```

Time Complexity = O(n)

Space Complexity = O(1)

4. Container with Most Water

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

SOLUTION:

```
import java.util.*;
class ContainerWithWater{
        public static int find(int[] arr){
                int l=0; int r=arr.length-1;
```

```
            int length = 0, breadth = 0, area = Integer.MIN_VALUE;
            while(l<=r){
                length = Math.min(arr[r],arr[l]);
                breadth = r - l;
                area = Math.max(area, length*breadth);
                while(l<=r && arr[l]<=length)  l++;
                while(l<=r && arr[r]<=length)  r--;
            }
            return area;
        }
        public static void main(String[] args){
            int arr1[] = {1, 5, 4, 3};
            int arr2[] = {3,1,2,4,5};
            System.out.println(find(arr1));
            System.out.println(find(arr2));
        }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac ContainerWithWater.java

D:\OOPS\09-11-2024 practice java>java ContainerWithWater
6
12
```

Time Complexity: O(N)

Space Complexity: O(1)

5. Find the Factorial of a large number

Input: 100
Output:33262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
Input: 50
Output: 30414093201713378043612608166064768844377641568960512000000000000

SOLUTION:

```
import java.math.BigInteger;

import java.util.Scanner;
```

```
public class Factorial {
    static BigInteger factorial(int N) {
        BigInteger f = BigInteger.ONE;
        for (int i = 2; i <= N; i++)
            f = f.multiply(BigInteger.valueOf(i));
        return f;
    }

    // Driver method
    public static void main(String args[]) {
        int N1 = 100;
        int N2 = 50;
        BigInteger r1 = factorial(N1);
    BigInteger r2 = factorial(N2);
        System.out.println("Factorial of " + N1 + " is:");
        System.out.println(r1);
    System.out.println("Factorial of " + N2 + " is:");
        System.out.println(r2);

    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac Factorial.java

D:\OOPS\09-11-2024 practice java>java Factorial
Factorial of 100 is:
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
Factorial of 50 is:
30414093201713378043612608166064768844377641568960512000000000000
```

Time Complexity: O(N)

Space Complexity: O(1)

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

SOLUTION:

```
class Traprainwater {
    public static int trap(int[] height) {
        int total = 0;
        int lmax = 0, rmax = 0;
        int l = 0, r = height.length - 1;
        while (l < r) {
            if (height[l] <= height[r]) {
                if (lmax > height[l]) {
                    total += lmax - height[l];
                } else {
                    lmax = height[l];
                }
                l = l + 1;
            } else {
                if (rmax > height[r]) {
                    total += rmax - height[r];
                } else {
                    rmax = height[r];
                }
                r = r - 1;
            }
        }
        return total;
    }

    public static void main(String[] args) {
        int[] arr1 = {3, 0, 1, 0, 4, 0, 2};
        int[] arr2 = {3, 0, 2, 0, 4};
```

```
        int[] arr3 = {1, 2, 3, 4};
        int[] arr4 = {10, 9, 0, 5};
        System.out.println("Input: arr[] = {3, 0, 1, 0, 4, 0, 2}");
        System.out.println("Output: " + trap(arr1));
        System.out.println("Input: arr[] = {3, 0, 2, 0, 4}");
        System.out.println("Output: " + trap(arr2));
        System.out.println("Input: arr[] = {1, 2, 3, 4}");
        System.out.println("Output: " + trap(arr3));
        System.out.println("Input: arr[] = {10, 9, 0, 5}");
        System.out.println("Output: " + trap(arr4));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac Traprainwater.java

D:\OOPS\09-11-2024 practice java>java Traprainwater
Input: arr[] = {3, 0, 1, 0, 4, 0, 2}
Output: 10
Input: arr[] = {3, 0, 2, 0, 4}
Output: 7
Input: arr[] = {1, 2, 3, 4}
Output: 0
Input: arr[] = {10, 9, 0, 5}
Output: 5
```

Time Complexity: O(N)

Space Complexity: O(1)

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

SOLUTION:

```
import java.util.Arrays;

class ChocolateDistribution {
    static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            // calculate difference of current window
            int diff = arr[i + m - 1] - arr[i];
            // if current difference is smaller then update the minimum difference
            if (diff < minDiff)
                minDiff = diff;
        }
        return minDiff;
    }

    public static void main(String[] args) {
        int[] arr1 = {7, 3, 2, 4, 9, 12, 56};
        int m1 = 3;
        System.out.println("Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3");
        System.out.println("Output: " + findMinDiff(arr1, m1));
        int[] arr2 = {7, 3, 2, 4, 9, 12, 56};
        int m2 = 5;
        System.out.println("Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5");
        System.out.println("Output: " + findMinDiff(arr2, m2));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac ChocolateDistribution.java

D:\OOPS\09-11-2024 practice java>java ChocolateDistribution
Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3
Output: 2
Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5
Output: 7
```

Time Complexity: O(N)

Space Complexity: O(1)

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].

SOLUTION:

```
import java.util.*;


class MergeIntervals{
    public static int[][] merge(int[][] intervals) {
        if(intervals.length <= 1) return intervals;
        Arrays.sort(intervals, (i1, i2) -> Integer.compare(i1[0], i2[0]));
        List<int[]> l = new ArrayList<>();
        for(int i = 0; i < intervals.length; i++){
            if(l.size() == 0 || intervals[i][0] > l.get(l.size() - 1)[1]){
                l.add(intervals[i]);
            } else {
                l.get(l.size() - 1)[1] = Math.max(intervals[i][1], l.get(l.size() - 1)[1]);
            }
        }
        return l.toArray(new int[l.size()][]);
    }
```

```
    public static void main(String[] args) {
        int[][] arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        int[][] t1 = merge(arr1);
    for(int[] i : t1)   System.out.println(Arrays.toString(i));
    System.out.println();
        int[][] arr2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
        int[][] t2 = merge(arr2);
    for(int[] i : t2)   System.out.println(Arrays.toString(i));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac MergeIntervals.java

D:\OOPS\09-11-2024 practice java>java MergeIntervals
[1, 4]
[6, 8]
[9, 10]

[1, 6]
[7, 8]
```

Time Complexity: O(N*logN) + O(N)

Space 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}}
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}}

SOLUTION:

```
class SetMatrixOnes {
    public static void setones(int[][] mat) {
        boolean firstrow = false;
        boolean firstcol = false;
```

```
    int n = mat.length;
    int m = mat[0].length;
    // Check if the first row or first column needs to be zeroed
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(mat[i][j] == 1) {
                if(i == 0) firstrow = true;
                if(j == 0) firstcol = true;
                mat[i][0] = 1;
                mat[0][j] = 1;
            }
        }
    }
    // Zero out the interior cells based on markers
    for(int i = 1; i < n; i++) {
        for(int j = 1; j < m; j++) {
            if(mat[0][j] == 1 || mat[i][0] == 1) {
                mat[i][j] = 1;
            }
        }
    }
    // Zero out the first row if needed
    if(firstrow) {
        for(int j = 0; j < m; j++) mat[0][j] = 1;
    }

    // Zero out the first column if needed
    if(firstcol) {
        for(int i = 0; i < n; i++) mat[i][0] = 1;
    }
}
public static void main(String[] args) {
    int[][] arr1 = {{1, 0}, {0, 0}};
    setones(arr1);
    printMatrix(arr1);
System.out.println();
```

```
        int[][] arr2 = {{0, 0, 0}, {0, 0, 1}};
        setones(arr2);
        printMatrix(arr2);
    System.out.println();

        int[][] arr3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};
        setones(arr3);
        printMatrix(arr3);
    System.out.println();
    }
    public static void printMatrix(int[][] mat) {
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[i].length; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac SetMatrixOnes.java

D:\OOPS\09-11-2024 practice java>java SetMatrixOnes
1 1
1 0

0 0 1
1 1 1

1 1 1 1
1 1 1 1
1 0 1 1
```

Time Complexity: O(N*M)

Space Complexity: O(1)

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.

SOLUTION:

```
import java.util.*;


class SpiralMatrix {
    public static List<Integer> spiralOrder(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        ArrayList<Integer> a = new ArrayList<>();
        // l denotes left , r denotes right, b denotes bottom, t denotes top
        int l = 0;
        int r = n - 1;
        int t = 0;
        int b = m - 1;
        while (l <= r && t <= b) {
            for (int i = l; i <= r; i++) {
                a.add(matrix[t][i]);
            }
            t++;
            for (int i = t; i <= b; i++) {
                a.add(matrix[i][r]);
            }
            r--;
            if (t <= b) {
                for (int i = r; i >= l; i--) {
                    a.add(matrix[b][i]);
                }
                b--;
            }
            if (l <= r) {
                for (int i = b; i >= t; i--) {
```

```
                a.add(matrix[i][l]);
            }
            l++;
        }
    }
    return a;
  }
  public static void main(String[] args) {
    int[][] matrix1 = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
        {13, 14, 15, 16}
    };
    System.out.println(spiralOrder(matrix1));
  System.out.println();
    int[][] matrix2 = {
        {1, 2, 3, 4, 5, 6},
        {7, 8, 9, 10, 11, 12},
        {13, 14, 15, 16, 17, 18}
    };
    System.out.println(spiralOrder(matrix2));
  }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac SpiralMatrix.java

D:\OOPS\09-11-2024 practice java>java SpiralMatrix
[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

[1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]
```

Time Complexity: O(N*M)

Space Complexity: O(N*M)

11. 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

SOLUTION:

```
import java.util.*;

class BalancedParanthesis {
    public static boolean isValid(String s) {
        Stack<Character> st = new Stack<Character>();
        for (char it : s.toCharArray()) {
            if (it == '(' || it == '[' || it == '{')
                st.push(it);
            else {
                if (st.isEmpty()) return false;
                char ch = st.pop();
                if ((it == ')' && ch == '(') || (it == ']' && ch == '[') || (it == '}' && ch == '{')) continue;
                else return false;
            }
        }
        return st.isEmpty();
    }

    public static void main(String[] args) {
        String str1 = "((()))()()";
        if (isValid(str1)) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }

        String str2 = "())((())";
        if (isValid(str2)) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not Balanced");
        }
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac BalancedParanthesis.java

D:\OOPS\09-11-2024 practice java>java BalancedParanthesis
Balanced
Not Balanced
```

Time Complexity: O(N)

Space Complexity: O(N)

12.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.

SOLUTION:

```
import java.util.*;

class Anagrams{

    static boolean areAnagrams(String s1, String s2) {

        // Create a hashmap to store character frequencies
        HashMap<Character, Integer> charCount = new HashMap<>();

        // Count frequency of each character in string s1
        for (char ch : s1.toCharArray())
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);

        // Count frequency of each character in string s2
        for (char ch : s2.toCharArray())
            charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);

        // Check if all frequencies are zero
        for (var pair : charCount.entrySet()) {
            if (pair.getValue() != 0) {
                return false;
            }
        }

        // If all conditions satisfied, they are anagrams
        return true;
    }
```

```
        public static void main(String[] args) {
            String s1 = "geeks";
            String s2 = "kseeg";
            System.out.println(areAnagrams(s1, s2) ? "true" : "false");
                System.out.println();
                String s3 = "allergy";
            String s4 = "allergic";
            System.out.println(areAnagrams(s3, s4) ? "true" : "false");
    System.out.println();
                String s5 = "g";
            String s6 = "g";
            System.out.println(areAnagrams(s5, s6) ? "true" : "false");


        }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac Anagrams.java

D:\OOPS\09-11-2024 practice java>java Anagrams
true

false

true
```

Time Complexity: O(N+M)

Space Complexity: O(K)

13. 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: “”

SOLUTION:

```
import java.util.*;

class LongPalindromeOfString {
    public static String longestPalindrome(String s) {
        String ans = "";

        if(s.length() <= 1) return s;

        for(int i = 1; i < s.length(); i++) {
            // odd-length palindrome check
            int l = i;
            int h = i;
            while(l >= 0 && h < s.length() && s.charAt(l) == s.charAt(h)) {
                l--;
                h++;
            }
            String n = s.substring(l + 1, h);
            if(n.length() > ans.length()) ans = n;

            // even-length palindrome check
            l = i - 1;
            h = i;
            while(l >= 0 && h < s.length() && s.charAt(l) == s.charAt(h)) {
                l--;
                h++;
            }
            n = s.substring(l + 1, h);
            if(n.length() > ans.length()) ans = n;
        }
        return ans;
    }

    public static void main(String[] args) {

        System.out.println(longestPalindrome("forgeeksskeegfor"));
        System.out.println();
        System.out.println(longestPalindrome("Geeks"));
        System.out.println();
        System.out.println(longestPalindrome("abc"));
        System.out.println();
        System.out.println(longestPalindrome(""));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac LongPalindromeOfString.java

D:\OOPS\09-11-2024 practice java>java LongPalindromeOfString
geeksskeeg

ee

b
```

Time Complexity: O(N^2)

Space Complexity: O(1)

14. 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.

SOLUTION:

```
import java.util.*;
class LongCommonPref {
    public static String longestCommonPrefix(String[] strs) {
        if (strs.length == 0) return "";
        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);
            }
        }
        return prefix.isEmpty() ? "-1" : prefix;
    }

    public static void main(String[] args) {
        String[] arr1 = { "geeksforgeeks", "geeks", "geek", "geezer" };
        System.out.println(longestCommonPrefix(arr1));
        String[] arr2 = { "hello", "world" };
        System.out.println(longestCommonPrefix(arr2));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac LongCommonPref.java

D:\OOPS\09-11-2024 practice java>java LongCommonPref
gee
-1
```

Time Complexity: O(N*M)

Space Complexity: O(1)

15. 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]

SOLUTION:

```
import java.util.*;

class DeleteMidOfStack {

    static void deleteMid(Stack<Character> st) {
        int n = st.size();
        Stack<Character> tempSt = new Stack<>();
        int count = 0;

        // Move first n/2 elements to tempSt
        while (count < n / 2) {
            tempSt.push(st.pop());
            count++;
        }

        // Remove middle element
        st.pop();

        while (!tempSt.isEmpty()) {
            st.push(tempSt.pop());
        }
    }

    public static void main(String[] args) {
        Stack<Character> st1 = new Stack<>();
        st1.push('1');
        st1.push('2');
        st1.push('3');
```

```
        st1.push('4');
        st1.push('5');
        deleteMid(st1);

        ArrayList<Character> result1 = new ArrayList<>();
        while (!st1.isEmpty()) {
            result1.add(st1.pop());
        }

        System.out.print("Stack[] = [");
        for (int i = result1.size() - 1; i >= 0; i--) {
            System.out.print(result1.get(i));
            if (i > 0) System.out.print(", ");
        }
        System.out.println("]");
        System.out.println();

        Stack<Character> st2 = new Stack<>();
        st2.push('1');
        st2.push('2');
        st2.push('3');
        st2.push('4');
        st2.push('5');
        st2.push('6');
        deleteMid(st2);

        ArrayList<Character> result2 = new ArrayList<>();
        while (!st2.isEmpty()) {
            result2.add(st2.pop());
        }

        System.out.print("Stack[] = [");
        for (int i = result2.size() - 1; i >= 0; i--) {
            System.out.print(result2.get(i));
            if (i > 0) System.out.print(", ");
        }
        System.out.println("]");
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac DeleteMidOfStack.java

D:\OOPS\09-11-2024 practice java>java DeleteMidOfStack
Stack[] = [1, 2, 4, 5]

Stack[] = [1, 2, 4, 5, 6]
```

Time Complexity: O(N)

Space Complexity: O(N)

16. 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

SOLUTION:

```
import java.util.*;

public class NextGreaterElement {
   static class stack {
      int top;
      int items[] = new int[100];

      // Stack functions to be used by printNGE
      void push(int x)
      {
         if (top == 99) {
            System.out.println("Stack full");
         }
         else {
            items[++top] = x;
         }
      }

      int pop()
      {
         if (top == -1) {
            System.out.println("Underflow error");
            return -1;
```

```
        }
        else {
            int element = items[top];
            top--;
            return element;
        }
    }

    boolean isEmpty()
    {
        return (top == -1) ? true : false;
    }
}

/* prints element and NGE pair for all elements of arr[] of size n */
static void printNGE(int arr[], int n)
{
    int i = 0;
    stack s = new stack();
    s.top = -1;
    int element, next;

    s.push(arr[0]);

    for (i = 1; i < n; i++) {
        next = arr[i];

        if (s.isEmpty() == false) {

            // if stack is not empty, then
            // pop an element from stack
            element = s.pop();

            /* If the popped element is smaller than next, then a) print the pair b) keep popping while elements
are smaller and stack is not empty */
            while (element < next) {
                System.out.println(element + " --> "
                                   + next);
                if (s.isEmpty() == true)
                    break;
                element = s.pop();
            }
            if (element > next)
                s.push(element);
        }
        s.push(next);
    }

    /* After iterating over the loop, the remaining elements in stack do not have the next greater element,
so print -1 for them */
    while (s.isEmpty() == false) {
```

```
            element = s.pop();
            next = -1;
            System.out.println(element + " -- " + next);
        }
    }
    public static void main(String[] args)
    {
         int arr1[] = {4, 5, 2, 25};
        int arr2[] = {13, 7, 6, 12};

        // First input and output
        printNGE(arr1, arr1.length);
        System.out.println();

        // Second input and output
        printNGE(arr2, arr2.length);
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac NextGreaterElement.java

D:\OOPS\09-11-2024 practice java>java NextGreaterElement
4 --> 5
2 --> 25
5 --> 25
25 --> -1

6 --> 12
7 --> 12
12 --> -1
13 --> -1
```

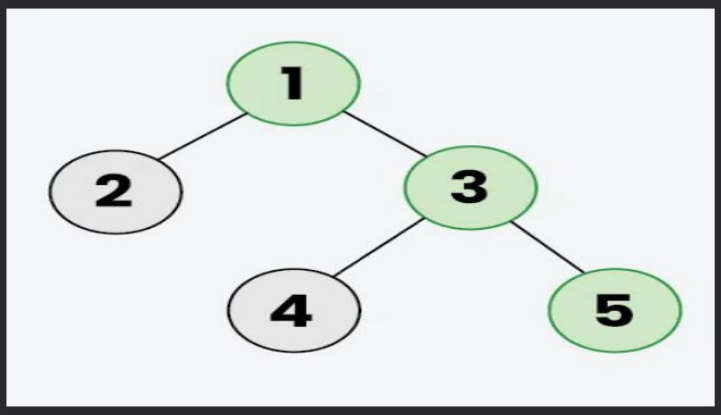
Time Complexity: O(N)

Space Complexity: O(N)

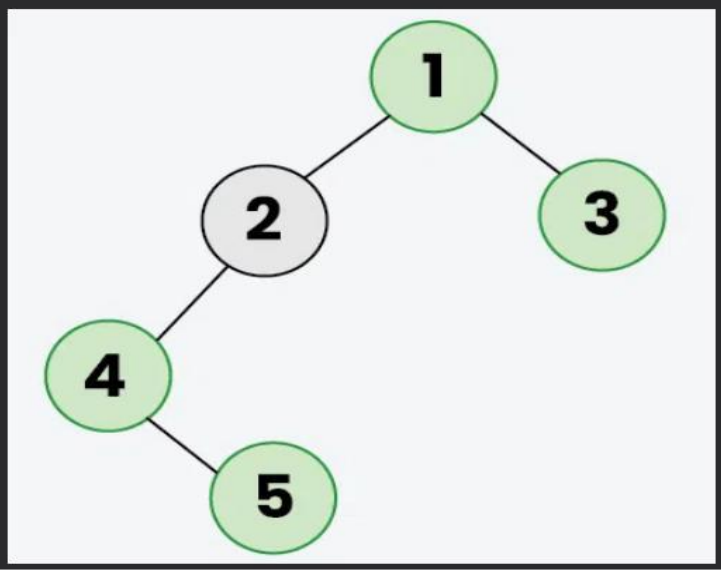
17. Print Right View of a Binary Tree

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.*




*Example 2: The **Green** colored nodes (1, 3, 4, 5) represents the Right view in the below Binary tree.*




SOLUTION:

```
import java.util.*;
public class RightViewBT {
    // TreeNode class should be defined inside the Main class
    static class TreeNode {
        int data;
        TreeNode left, right;
        // Constructor for the TreeNode class
        public TreeNode(int item) {
            data = item;
            left = right = null;
        }
    }

    // Main method to execute the program
    public static void main(String[] args) {
```

```
        RightViewBT tree = new RightViewBT();

        // Creating the binary tree
        tree.root = new TreeNode(1);
        tree.root.left = new TreeNode(2);
        tree.root.right = new TreeNode(3);
        tree.root.left.left = new TreeNode(4);
        tree.root.left.right = new TreeNode(5);
        tree.root.right.right = new TreeNode(6);
        tree.root.left.left.left = new TreeNode(7);

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

    TreeNode root;

    // Function to print the right view of the tree
    public void rightView() {
        if (root == null) {
            return;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        // Loop to traverse the tree level by level
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            TreeNode currentNode = null;

            // Process each node in the current level
            for (int i = 0; i < levelSize; i++) {
                currentNode = queue.poll();

                // Add left and right children of the node to the queue
                if (currentNode.left != null) {
                    queue.add(currentNode.left);
                }
                if (currentNode.right != null) {
                    queue.add(currentNode.right);
                }
            }

            // The last node processed at the current level is the rightmost node
            System.out.print(currentNode.data + " ");
        }
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac RightViewBT.java

D:\OOPS\09-11-2024 practice java>java RightViewBT
Right view of the binary tree:
1 3 6 7
```

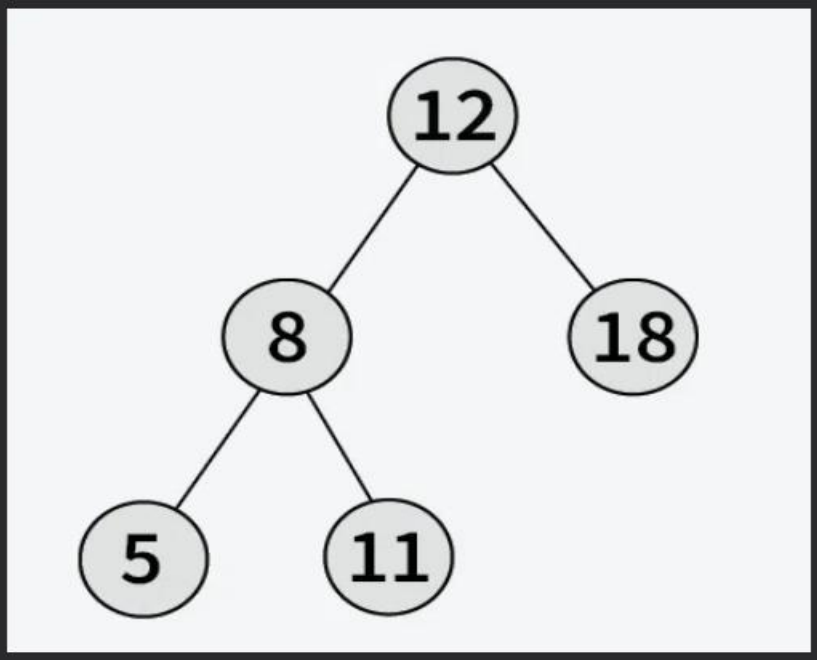
Time Complexity: O(N)

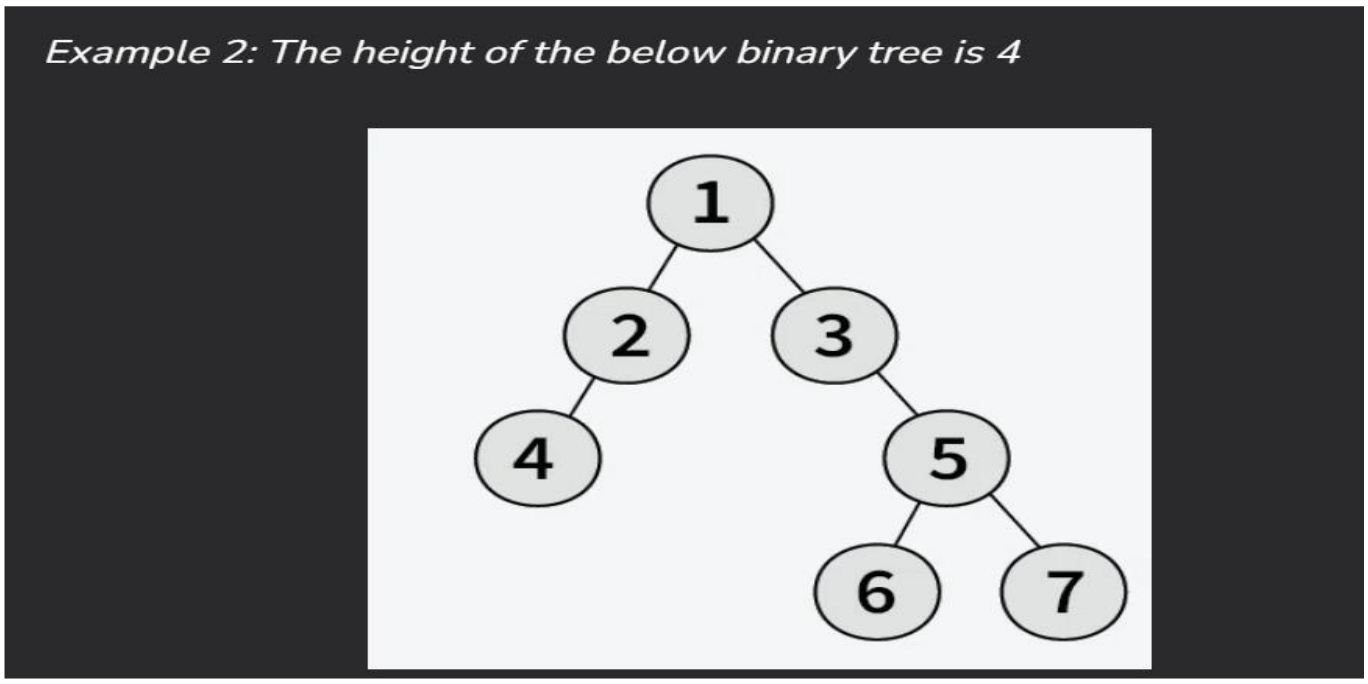
Space Complexity: O(H)

18. Maximum Depth or Height of Binary Tree

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.*






SOLUTION:

```
import java.util.*;
class Node {
    int data;
    Node left, right;
    Node(int val) {
        data = val;
        left = null;
        right = null;
    }
}
class MaxDepthBT {
    static int maxDepth(Node node) {
        if (node == null)
            return 0;

        // compute the depth of left and right subtrees
        int lDepth = maxDepth(node.left);
        int rDepth = maxDepth(node.right);

        return Math.max(lDepth, rDepth) + 1;
    }

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

        System.out.println(maxDepth(root));

        Node root1 = new Node(1);
```

```
        root1.left = new Node(2);
        root1.right = new Node(3);
        root1.left.left = new Node(4);
        root1.right.right = new Node(5);
        root1.right.right.left = new Node(6);
        root1.right.right.left = new Node(7);
        System.out.println(maxDepth(root1));
    }
}
```

OUTPUT:

```
D:\OOPS\09-11-2024 practice java>javac MaxDepthBT.java

D:\OOPS\09-11-2024 practice java>java MaxDepthBT
3
4
```

Time Complexity: O(N)

Space Complexity: O(H)