**DSA TRAINING**

**NAME:** Dhuvarrakesh K

**Department:** IT

**Date:** 09/11/24

**Register Number:** 22IT024

**1.Maximum Subarray Sum – Kadane‟s Algorithm:**

Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

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

Output: 11

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

Input: arr[] = {-2, -4}

Output: –2

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

**Code:**

import java.util.Scanner;

public class SubArray{

public static int maxSubArray(int[] arr){

int x=arr[0];

int sum=arr[0];

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

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

if(sum>x){

x=sum;

}

}

return x;

}

public static void main(String[] args){

Scanner scanner=new Scanner(System.in);

int arr0=maxSubArray(new int[] {2, 3, -8, 7, -1, 2, 3});

int arr1=maxSubArray(new int[] {-2, -4});

int arr2=maxSubArray(new int[] {5, 4, 1, 7, 8});

System.out.println("Output: "+arr0);

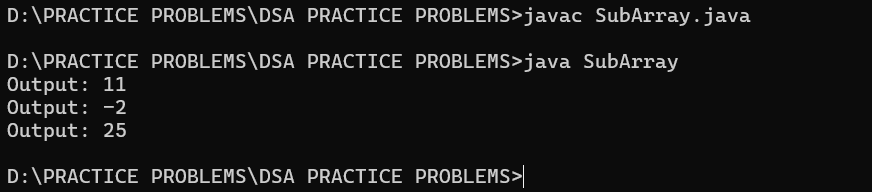
System.out.println("Output: "+arr1);

System.out.println("Output: "+arr2);

scanner.close();

}

}  
 **Output:**

****

**Time Complexity:O(n)**

**Space 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

**Code:**

import java.util.Scanner;

public class SubArray1{

public static int maxSubArray(int[] arr){

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

return 0;

}

int maxProduct = arr[0];

int minProduct = arr[0];

int result = arr[0];

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

result = Math.max(result, maxProduct);

}

return result;

}

public static void main(String[] args){

Scanner scanner=new Scanner(System.in);

int arr0=maxSubArray(new int[] {-2, 6, -3, -10, 0, 2});

int arr1=maxSubArray(new int[] {-1, -3, -10, 0, 60});

System.out.println("Output: "+arr0);

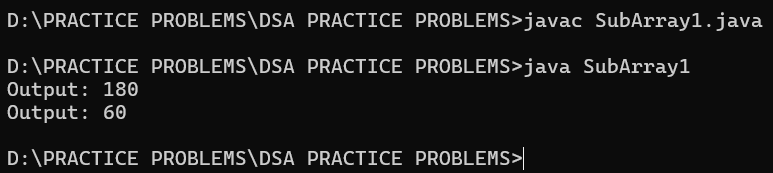
System.out.println("Output: "+arr1);

scanner.close();

}

}

**Output:**

****

**Time Complexity:O(n)**

**Space Complexity: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

**Code:**

import java.util.Scanner;

public class RotatedArray {

public static int binsearch(int[] arr, int key) {

int left = 0;

int right = arr.length - 1;

while (left <= right) {

int mid = (left + right) / 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;

}

public static void main(String[] args) {

int[] arr0 = {4, 5, 6, 7, 0, 1, 2};

int key0 = 0;

int[] arr1 = {4, 5, 6, 7, 0, 1, 2};

int key1 = 3;

int[] arr2 = {50, 10, 20, 30, 40};

int key2 = 10;

System.out.println("Output: " + binsearch(arr0, key0));

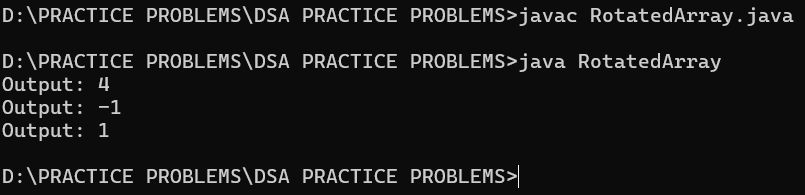
System.out.println("Output: " + binsearch(arr1, key1));

System.out.println("Output: " + binsearch(arr2, key2));

}

}

**Output:**



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

**Space Complexity:O(1)**

**4.Container with Most Water**

Given n non-negative integers a1, a2, ..., an where each represents a point at

coordinate (i, ai). 'n' vertical lines are drawn such that the two endpoints of line I is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

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

Note: You may not slant the container.

Input: arr = [1, 5, 4, 3]

Output: 6

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

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

Output: 12

Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**Code:**

import java.util.Scanner;

public class maxArea{

public static int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int maxArea = 0;

while (left < right) {

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

maxArea = Math.max(maxArea, currentArea);

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

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

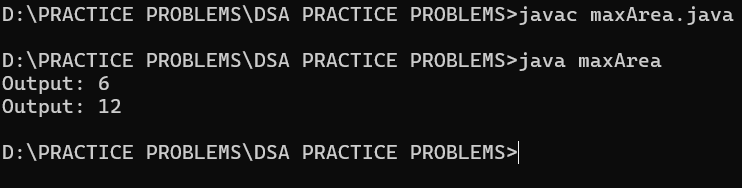
System.out.println("Output: " + maxArea(new int[] {1, 5, 4, 3}));

System.out.println("Output: " + maxArea(new int[] {3, 1, 2, 4, 5}));

}

}

**Output:**

****

**Time Complexty:O(n)**

**Space Complexity:O(1)**

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

Input: 100

Output: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**Code:**

import java.util.Scanner;

import java.math.BigInteger;

public class Factorial {

public static BigInteger fact(BigInteger n) {

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

return BigInteger.ONE;

} else if (n.compareTo(BigInteger.ONE) > 0) {

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

} else {

return BigInteger.valueOf(-1);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

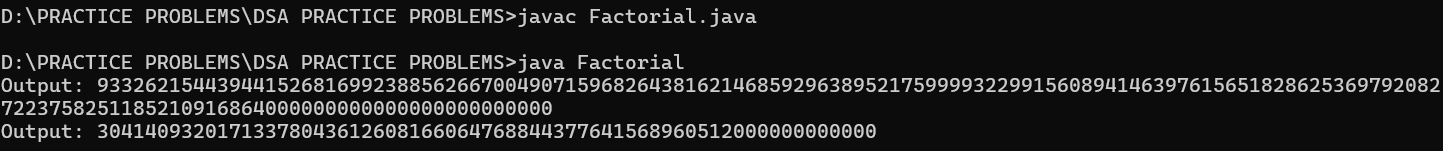
System.out.println("Output: " + fact(BigInteger.valueOf(100)));

System.out.println("Output: " + fact(BigInteger.valueOf(50)));

}

}

**Output:**

****

**Time Complexity:O(n)**

**Space Complexity:O(n)**

**6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.**

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

Output: 10

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

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

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

public class TrappedWater {

public static int trap(int[] height) {

int n = height.length;

if (n == 0) return 0;

int[] left = new int[n];

int[] right = new int[n];

int storedWater = 0;

left[0] = height[0];

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

left[i] = Math.max(left[i-1], height[i]);

}

right[n - 1] = height[n - 1];

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

right[i] = Math.max(right[i + 1], height[i]);

}

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

int minHeight = Math.min(left[i], right[i]);

storedWater += minHeight - height[i];

}

return storedWater;

}

public static void main(String[] args) {

int[] height1 = {3, 0, 1, 0, 4, 0, 2};

TrappedWater solution1 = new TrappedWater();

int trappedWater1 = TrappedWater.trap(height1);

System.out.println("Total trapped water: "+trappedWater1);

int[] height2 = {3, 0, 2, 0, 4};

TrappedWater solution2 = new TrappedWater();

int trappedWater2 = TrappedWater.trap(height2);

System.out.println("Total trapped water: "+trappedWater2);

int[] height3 = {1, 2, 3, 4};

TrappedWater solution3 = new TrappedWater();

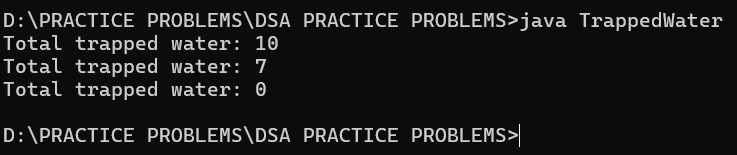
int trappedWater3 = TrappedWater.trap(height3);

System.out.println("Total trapped water: "+trappedWater3);

}

}

**Output:**

****

**Time Complexity:O(n)**

**Space Complexity:O(n)**

**7.Chocolate Distribution Problem**

Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

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

Output: 2

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

**Code:**

import java.util.Arrays;

public class Chocolate {

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

if (m == 0 || n == 0 || m > n) return 0;

Arrays.sort(arr);

int minDiff = Integer.MAX\_VALUE;

for (int i = 0; i + m - 1 < n; i++) {

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

minDiff = Math.min(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

Chocolate solution = new Chocolate();

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

int m = 3;

int n = arr.length;

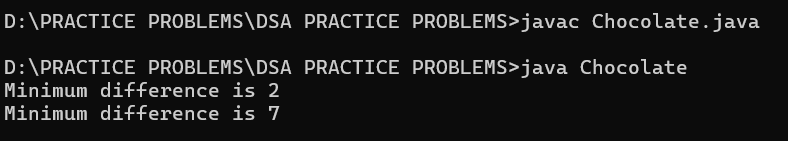
int result = solution.findMinDiff(arr, n, m);

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

}

}

**Output:**

****

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

**Space Complexity:O(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.ArrayList;

import java.util.Arrays;

import java.util.List;

public class MergeInt {

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

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

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

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

int[] currentInterval = intervals[0];

merged.add(currentInterval);

for (int[] interval : intervals) {

int currentEnd = currentInterval[1];

int nextStart = interval[0];

int nextEnd = interval[1];

if (currentEnd >= nextStart) {

currentInterval[1] = Math.max(currentEnd, nextEnd);

} else {

currentInterval = interval;

merged.add(currentInterval);

}

}

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

}

public static void main(String[] args) {

MergeInt solution = new MergeInt();

int[][] intervals1 = {{1, 3}, {2, 6}, {8, 10}, {15, 18}};

int[][] result1 = solution.merge(intervals1);

System.out.print("Merged Intervals 1: ");

for (int[] interval : result1) {

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

}

System.out.println();

int[][] intervals2 = {{1, 4}, {4, 5}};

int[][] result2 = solution.merge(intervals2);

System.out.print("Merged Intervals 2: ");

for (int[] interval : result2) {

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

}

System.out.println();

int[][] intervals3 = {{1, 4}, {5, 6}};

int[][] result3 = solution.merge(intervals3);

System.out.print("Merged Intervals 3: ");

for (int[] interval : result3) {

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

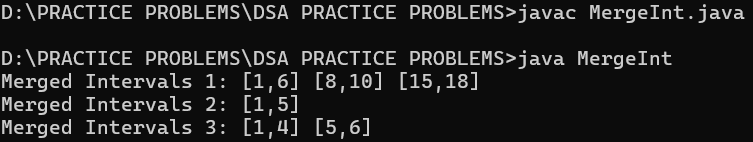
}

System.out.println();

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**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}}

**Code:**

class BoolMatrix {

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

int r = matrix.length;

int c = matrix[0].length;

boolean[] rco = new boolean[r];

boolean[] cco = new boolean[c];

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

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

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

rco[i] = true;

cco[j] = true;

}

}

}

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

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

if (rco[i] || cco[j]) {

matrix[i][j] = 1;

}

}

}

}

public static void main(String[] args) {

int[][] matrix1 = {{1, 0}, {0, 0}};

System.out.println("Input:");

printMatrix(matrix1);

BoolMatrix.booleanMatrix(matrix1);

System.out.println("Output:");

printMatrix(matrix1);

int[][] matrix2 = {{0, 0, 0}, {0, 0, 1}};

System.out.println("\nInput:");

printMatrix(matrix2);

BoolMatrix.booleanMatrix(matrix2);

System.out.println("Output:");

printMatrix(matrix2);

int[][] matrix3 = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};

System.out.println("\nInput:");

printMatrix(matrix3);

BoolMatrix.booleanMatrix(matrix3);

System.out.println("Output:");

printMatrix(matrix3);

}

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

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

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

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

}

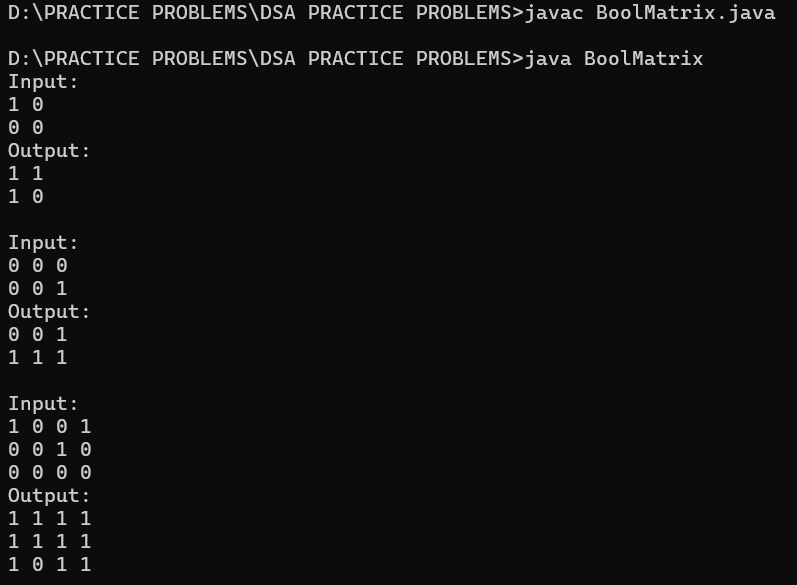
System.out.println();

}

}

}

**Output:**

****

**Time Complexity:O(r\*c)**

**Space Complexity:O(r+c)**

**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.ArrayList;

import java.util.List;

class SpiralMatrix {

public List<Integer> spiralOrder(int[][] matrix) {

ArrayList<Integer> a = new ArrayList<Integer>();

int rows = matrix.length, cols = matrix[0].length;

int left = 0, right = cols - 1, top = 0, bottom = rows - 1;

while (left <= right && top <= bottom) {

for (int i = left; i <= right; i++) {

a.add(matrix[top][i]);

}

top++;

for (int i = top; i <= bottom; i++) {

a.add(matrix[i][right]);

}

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--) {

a.add(matrix[bottom][i]);

}

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--) {

a.add(matrix[i][left]);

}

left++;

}

}

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}

};

SpiralMatrix sm = new SpiralMatrix();

List<Integer> result1 = sm.spiralOrder(matrix1);

for (int num : result1) {

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

}

System.out.println();

int[][] matrix2 = {

{1, 2, 3, 4, 5, 6},

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

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

};

List<Integer> result2 = sm.spiralOrder(matrix2);

for (int num : result2) {

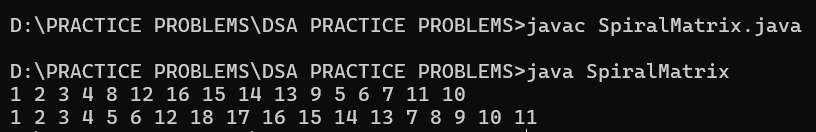
System.out.print(num + " ");

}

}

}

**Output:**

****

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

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

**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.Stack;

public class Parentheses {

public static String result(String str) {

Stack<Character> st1 = new Stack<>();

Stack<Character> st2 = new Stack<>();

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

if (I == ')' && !st1.isEmpty() && st1.peek() == '(') {

st1.pop();

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

st2.push(')');

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

st1.push('(');

}

}

return (st1.isEmpty() && st2.isEmpty()) ? "Balance" : "Not Balanced";

}

public static void main(String[] args) {

String str1 = "((()))()()";

System.out.println("Output 1: " + result(str1));

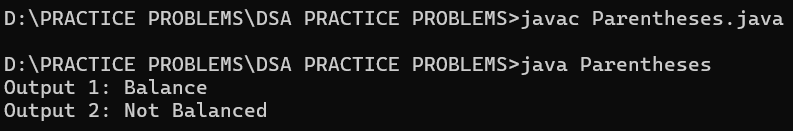
String str2 = "())((())";

System.out.println("Output 2: " + result(str2));

}

}

**Output:**

****

**Time Complexity:** O(n)

**Space 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 = “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.

**Code:**

import java.util.Arrays;

class Anagram {

public boolean isAnagram(String s, String t) {

if(s.length()!=t.length()){

return false;

}

char[] c1 = s.toCharArray();

char[] c2 = t.toCharArray();

Arrays.sort(c1);

Arrays.sort(c2);

for(int i = 0; i < c1.length; i++){

if(c1[i] != c2[i]){

return false;

}

}

return true;

}

public static void main(String[] args) {

Anagram solution = new Anagram();

String s1 = "allergy";

String s2 = "allergic";

System.out.println(solution.isAnagram(s1, s2));

s1 = "g";

s2 = "g";

System.out.println(solution.isAnagram(s1, s2));

s1 = "geeks";

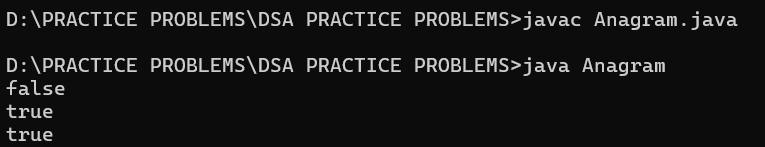
s2 = "kseeg";

System.out.println(solution.isAnagram(s1, s2));

}

}

**Output:**

****

**Time Complexity:** O(nlogn)

**Space 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:**

public class LongPalindrome {

public String longestPalindrome(String s) {

if (s.length() <= 1) {

return s;

}

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

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

String odd = expandFromCenter(s, i, i);

String even = expandFromCenter(s, i, i + 1);

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

maxStr = odd;

}

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

maxStr = even;

}

}

return maxStr;

}

private String expandFromCenter(String s, int left, int right) {

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

left--;

right++;

}

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

}

public static void main(String[] args) {

LongPalindrome sol = new LongPalindrome();

String str1 = "forgeeksskeegfor";

System.out.println("Output 1: " + sol.longestPalindrome(str1));

String str2 = "Geeks";

System.out.println("Output 2: " + sol.longestPalindrome(str2));

String str3 = "abc";

System.out.println("Output 3: " + sol.longestPalindrome(str3));

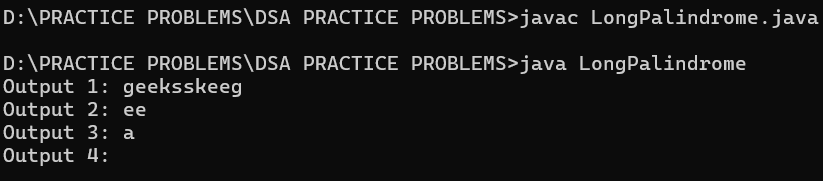
String str4 = "";

System.out.println("Output 4: " + sol.longestPalindrome(str4));

}

}

**Output:**

****

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

**Space Complexity:**O(1)

**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.Arrays;

public class LongestCommonPrefix {

public String longestCommonPrefix(String[] strs) {

Arrays.sort(strs);

String s1 = strs[0];

String s2 = strs[strs.length - 1];

int idx = 0;

while (idx < s1.length() && idx < s2.length()) {

if (s1.charAt(idx) == s2.charAt(idx)) {

idx++;

} else {

break;

}

}

return (idx == 0) ? "-1" : s1.substring(0, idx);

}

public static void main(String[] args) {

LongestCommonPrefix solution = new LongestCommonPrefix();

String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};

System.out.println("Output 1: " + solution.longestCommonPrefix(arr1));

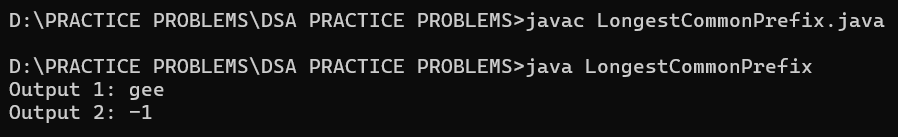
String[] arr2 = {"hello", "world"};

System.out.println("Output 2: " + solution.longestCommonPrefix(arr2));

}

}

**Output:**

****

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

**Space Complexity:** O(m log m)

**17.Delete middle element of a stack**

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

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

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

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

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

**Code:**

import java.util.\*;

public class DeleteMiddle {

public static void deleteMiddle(Stack<Integer> st, int n, int current) {

if (current == n / 2) {

st.pop();

return;

}

int x = st.pop();

deleteMiddle(st, n, current + 1);

st.push(x);

}

public static void main(String[] args) {

Stack<Integer> st1 = new Stack<>();

st1.push(1);

st1.push(2);

st1.push(3);

st1.push(4);

st1.push(5);

int n1 = st1.size();

deleteMiddle(st1, n1, 0);

List<Integer> result1 = new ArrayList<>();

while (!st1.isEmpty()) {

result1.add(st1.pop());

}

Collections.reverse(result1);

System.out.println("Output 1: " + result1);

Stack<Integer> st2 = new Stack<>();

st2.push(1);

st2.push(2);

st2.push(3);

st2.push(4);

st2.push(5);

st2.push(6);

int n2 = st2.size();

deleteMiddle(st2, n2, 0);

List<Integer> result2 = new ArrayList<>();

while (!st2.isEmpty()) {

result2.add(st2.pop());

}

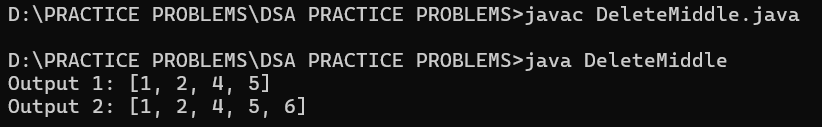
Collections.reverse(result2);

System.out.println("Output 2: " + result2);

}

}

**Output:**

****

**Time Complexity:**O(n)

**Space Complexity**:O(n)

**18. Next Greater Element (NGE) for every element in given Array**

Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

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

Output: 4 5 2 –> 5 –> 25 –> 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 –> 7 -1 –> 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 NGE {

public int[] nextGreaterElement(int[] nums1, int[] nums2) {

int[] output = new int[nums1.length];

for (int i = 0; i < nums1.length; i++) {

int set = 0;

for (int j = 0; j < nums2.length; j++) {

if (nums1[i] == nums2[j]) {

set = 1;

}

if (set == 1 && nums2[j] > nums1[i]) {

output[i] = nums2[j];

break;

} else if (set == 1 && (j == nums2.length - 1)) {

output[i] = -1;

}

}

}

return output;

}

public static void main(String[] args) {

NGE nge = new NGE();

int[] nums1 = {4, 5, 2, 25};

int[] nums2 = {4, 1, 2, 3, 5, 25};

int[] result1 = nge.nextGreaterElement(nums1, nums2);

for (int num : result1) {

System.out.println(num);

}

int[] nums3 = {13, 7, 6, 12};

int[] nums4 = {13, 6, 7, 12};

int[] result2 = nge.nextGreaterElement(nums3, nums4);

for (int num : result2) {

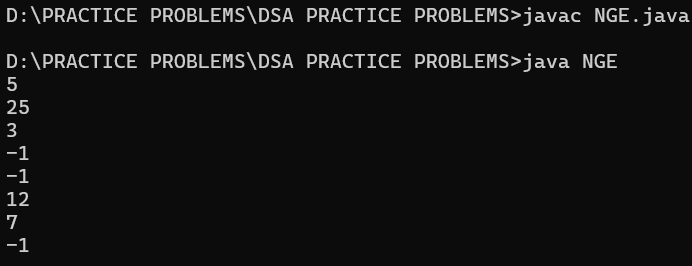
System.out.println(num);

}

}

}

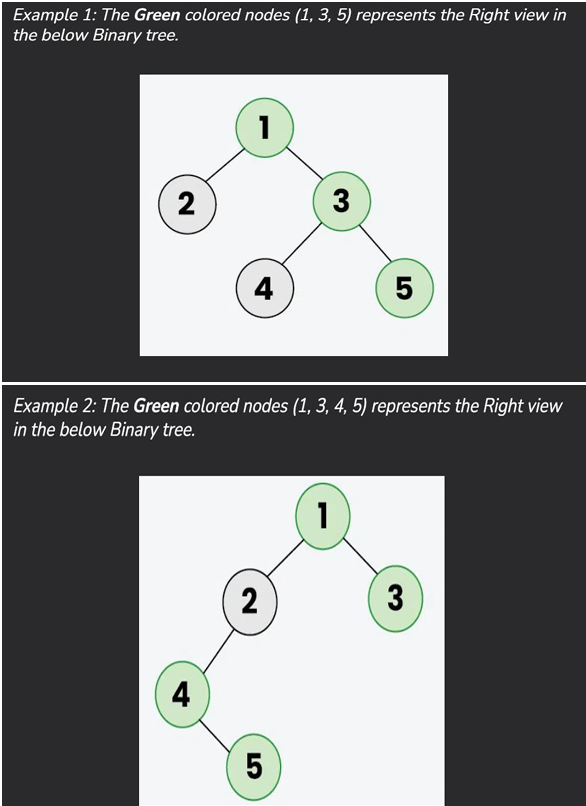
**Output:**

****

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

import java.util.LinkedList;

import java.util.Queue;

class Node {

int data;

Node left;

Node right;

Node(int val) {

data = val;

left = null;

right = null;

}

}

class Rview {

public static void rightView(Node root) {

if (root == null) {

return;

}

Queue<Node> q = new LinkedList<>();

q.add(root);

while (!q.isEmpty()) {

int size = q.size();

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

Node front = q.poll();

if (i == size - 1) {

System.out.print(front.data + " ");

}

if (front.left != null) {

q.add(front.left);

}

if (front.right != null) {

q.add(front.right);

}

}

}

}

public static void main(String[] args) {

Node root = new Node(12);

root.left = new Node(8);

root.right = new Node(18);

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

root.right.left = new Node(17);

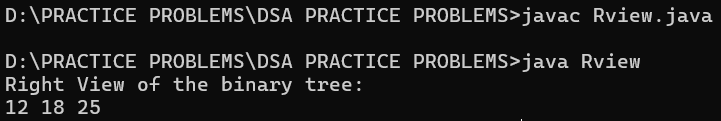
root.right.right = new Node(25);

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

rightView(root);

}

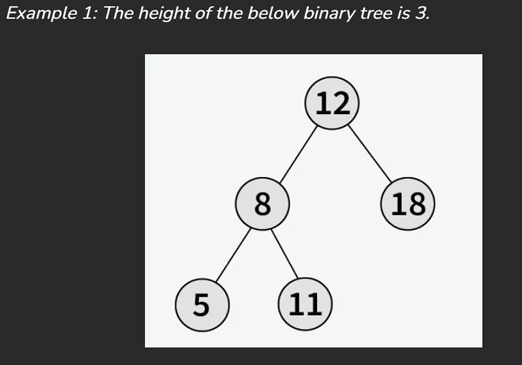
}

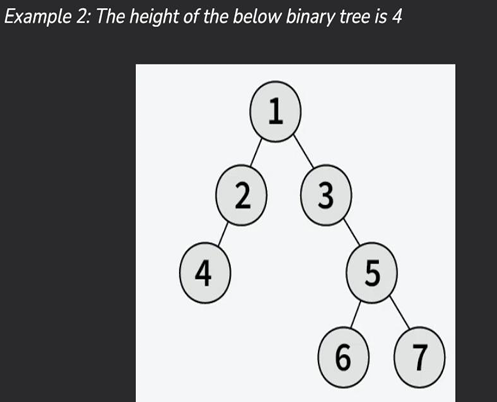
**Output:** ****

**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:**

import java.util.LinkedList;

import java.util.Queue;

class Node {

int data;

Node left;

Node right;

Node(int val) {

data = val;

left = null;

right = null;

}

}

public class MaxHeight {

public static int maxDepth(Node root) {

if (root == null) {

return 0;

}

Queue<Node> q = new LinkedList<>();

int level = 0;

q.add(root);

while (!q.isEmpty()) {

int size = q.size();

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

Node front = q.poll();

if (front.left != null) {

q.add(front.left);

}

if (front.right != null) {

q.add(front.right);

}

}

level++;

}

return level;

}

public static void main(String args[]) {

Node root = new Node(12);

root.left = new Node(8);

root.right = new Node(18);

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

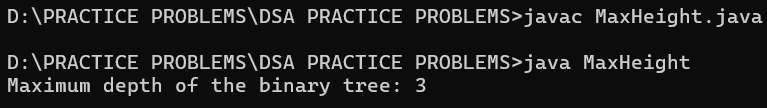
int depth = MaxHeight.maxDepth(root);

System.out.println("Maximum depth of the binary tree: " + depth);

}

}

**Output:**

****

**Time Complexity:O(n)**

**Space Complexity:O(n)**