**Coding practice Problems:** 9.11.2024   
 **Name: VAITHIYANATHAN T   
 Dept :CSE**  
**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** :   
  
class Solution {

public int maxSubArray(int[] nums) {

int maxi = Integer.MIN\_VALUE;

int sum = 0;

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

sum += nums[i];

if (sum > maxi) {

maxi = sum;

}

if (sum < 0) {

sum = 0;

}

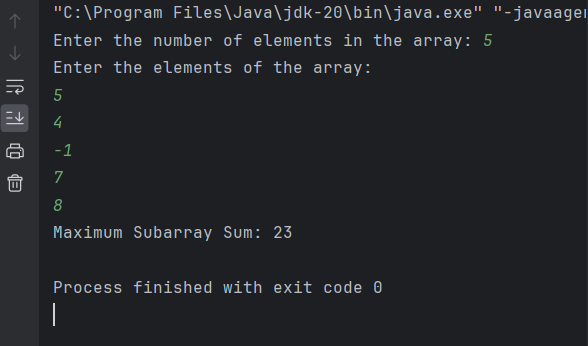
}

return maxi;

}

}

**TC - O(N)**

**SC - O(1)  
  
OUTPUT:**  
  


========================================================================

**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**  
CODE :   
class Solution {

public int maxProduct(int[] nums) {

int n = nums.length;

int pre = 1;

int suf =1;

int m = Integer.MIN\_VALUE;

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

{

if(pre==0){

pre=1;

}

else if(suf==0){

suf=1;

}

pre = pre\*nums[i];

suf = suf\*nums[n-i-1];

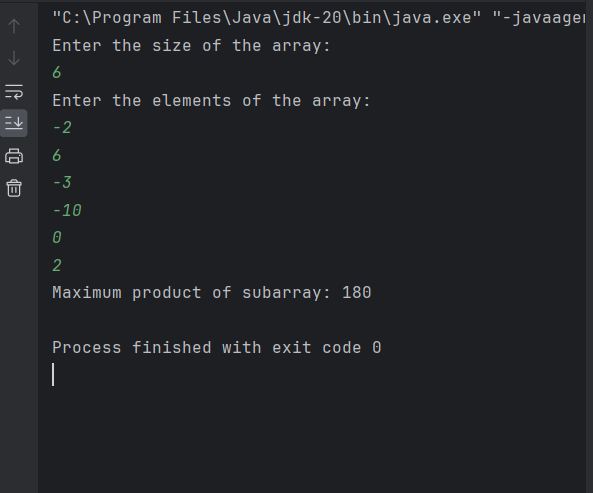
m = Math.max(m,Math.max(pre,suf));

}

return m;

}

}  
  
**TC- O(n)  
SC -O(1)  
Output:**



========================================================================

**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   
CODE:**

class Solution {

public int search(int[] nums, int t) {

int low = 0;

int high = nums.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (nums[mid] == t) {

return mid;

}

if (nums[low] <= nums[mid]) { // Left half is sorted

if (t >= nums[low] && t <= nums[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

} else { // Right half is sorted

if (t >= nums[mid] && t <= nums[high]) {

low = mid + 1;

} else {

high = mid - 1;

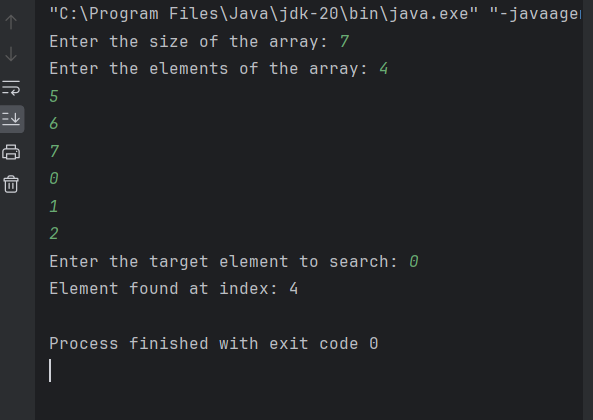
}

}

}

return -1; // Target not found

}

}  
**TC: O(log n) SC :O(1)**  


========================================================================

**4. Container with Most Water  
Given n non-negative integers m, m, ... where each represents a point at**

**coordinate (i, ' n ' vertical lines are drawn such that the two endpoints of line i**

**is at (i, at•) 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.**

**CODE :**class Solution {

public int maxArea(int[] h) {

int low = 0;

int high = h.length-1;

int res = 0;

while(low<high){

res = Math.max(res,(high-low)\*Math.min(h[low],h[high]));

if (h[low]<h[high]){

low+=1;

}

else{

high-=1;

}

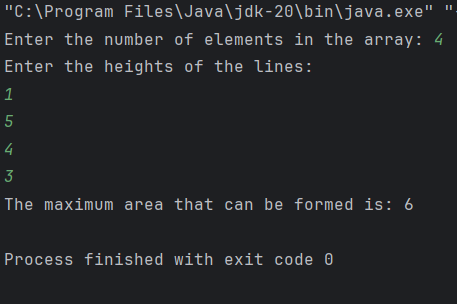
}

return res;

}

}

**TC - O(N)   
SC -O(1)**



========================================================================

**5. Find the Factorial of a large number Input: 100 Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00 Input: 50 Output: 30414093201713378043612608166064768844377641568960512000000000000  
  
CODE :**  
class Solution {

public BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

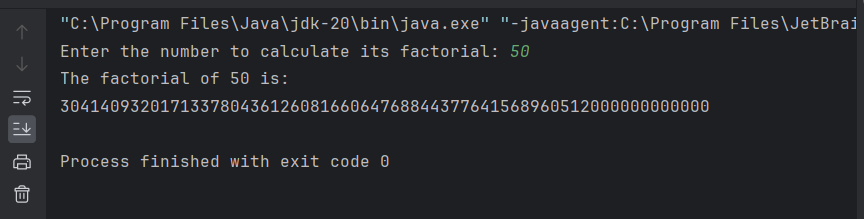
for (int i = 2; i <= n; i++) {

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

}

return result;

}

**TC :O()  
SC:O()**  
  
  


========================================================================

**6.Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain. Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image. Input: arr[] = {3, 0, 2, 0, 4} Output: 7 Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units. Input: arr[] = {1, 2, 3, 4} Output: 0 Explanation : We cannot trap water as there is no height bound on both sides Input: arr[] = {10, 9, 0, 5} Output: 5 Explanation : We trap 0 + 0 + 5 + 0 = 5**  
**CODE:**  
class Solution {

public int trap(int[] h) {

int low = 0;

int high = h.length-1;

int res = 0;

int left\_max = h[low];

int right\_max = h[high];

while(low<high){

if (h[low] < h[high]){

low+=1;

left\_max = Math.max(left\_max,h[low]);

res+=left\_max - h[low];

}

else{

high-=1;

right\_max = Math.max(right\_max,h[high]);

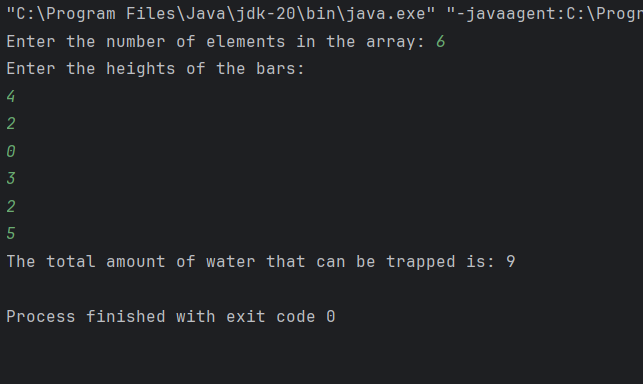
res+=right\_max - h[high];

}

}

return res;

}

}  
  
**TC - O(n)  
SC - 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**

**CODE:**import java.util.Arrays;

class Solution {

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

if (n < m) {

return -1;

}

Arrays.sort(arr);

long minDiff = Long.MAX\_VALUE;

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

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

minDiff = Math.min(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

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

int m = 3;

Solution sol = new Solution();

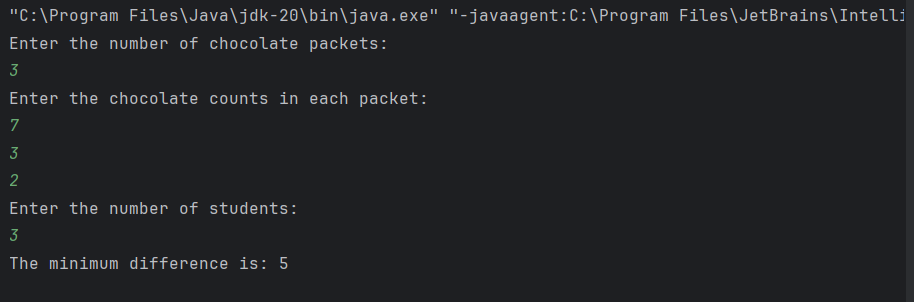
long result = sol.findMinDiff(arr, arr.length, m);

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

}

}

**TC -O(N LOG N)**

**SC -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].**   
  
**CODE:**class Solution {

public void mergeIntervals(int[][] intervals, int n) {

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

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

merged.add(intervals[0]);

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

int[] last = merged.get(merged.size() - 1);

int[] curr = intervals[i];

if (last[1] >= curr[0]) {

last[1] = Math.max(last[1], curr[1]);

} else {

merged.add(curr);

}

}

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

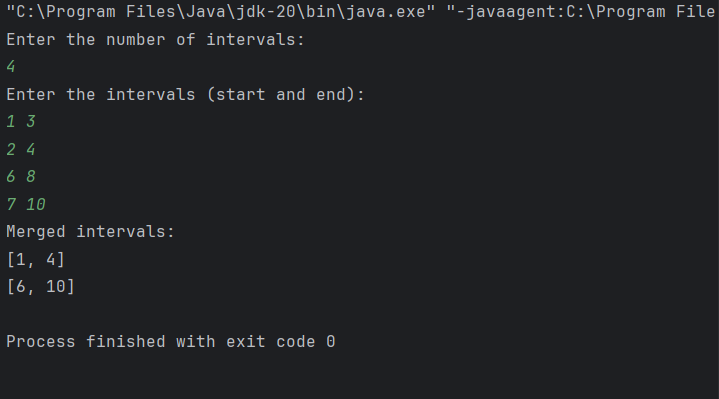
for (int[] interval : merged) {

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

}

}

**TC -O(N LOG N)**

**SC-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 :**  
 public void modifyMatrix(int[][] mat, int M, int N) {

boolean[] rows = new boolean[M];

boolean[] cols = new boolean[N];

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

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

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

rows[i] = true;

cols[j] = true;

}

}

}

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

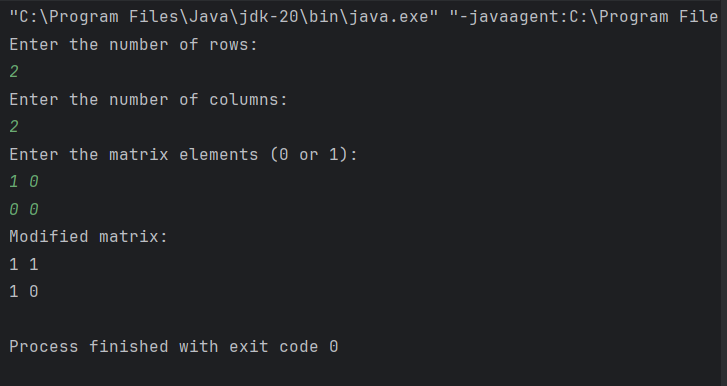
for (int j = 0; j < N; j++) {

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

mat[i][j] = 1;

}

**TC - O(M\*N)**

**SC - O(M)+O(N)  
  
**

========================================================================

**10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form. Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10 Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}} Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11 Explanation: The output is matrix in spiral format.**   
**CODE:** public void printSpiral(int[][] matrix, int m, int n) {

int top = 0, bottom = m - 1, left = 0, right = n - 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] + " ");

}

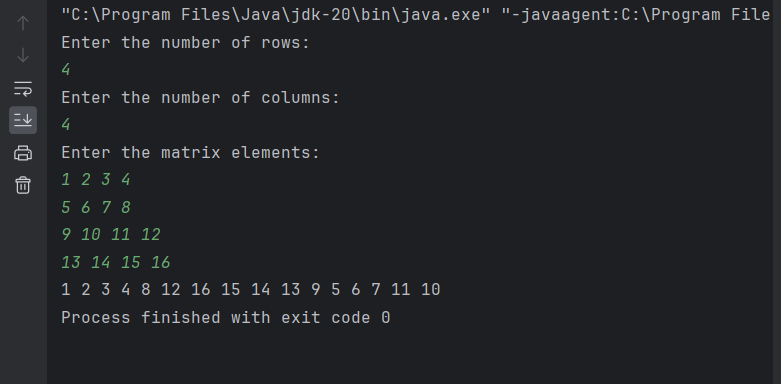
left++;

}

}

}

**TC - O(M\*N)**

**SC - O(1)   
  
**

========================================================================**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:** public String checkBalanced(String str) {

int count = 0;

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

char ch = str.charAt(i);

if (ch == '(') {

count++;

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

count--;

}

if (count < 0) {

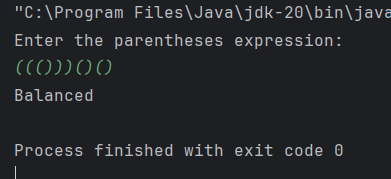
return "Not Balanced";

}

}

return count == 0 ? "Balanced" : "Not Balanced";

}

**TC - O() SC -O()**  
  
  
========================================================================  
  
**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 anagram**s  
  
**CODE:**  
 public boolean areAnagrams(String s1, String s2) {

if (s1.length() != s2.length()) {

return false;

}

int[] freq = new int[26];

for (int i = 0; i < s1.length(); i++) {

freq[s1.charAt(i) - 'a']++;

freq[s2.charAt(i) - 'a']--;

}

for (int count : freq) {

if (count != 0) {

return false;

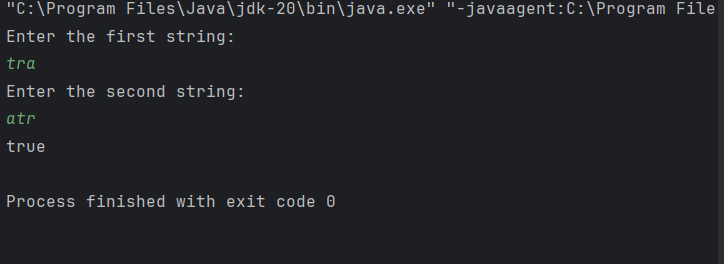
}

}

return true;

}

TC - O(n)

SC -O(1)  
  


========================================================================

**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 String longestPalindrome(String s) {

int n = s.length(), start = 0, maxLen = 1;

boolean[][] dp = new boolean[n][n];

for (int i = 0; i < n; i++) dp[i][i] = true;

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

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

int j = i + len - 1;

if (s.charAt(i) == s.charAt(j)) {

dp[i][j] = len == 2 || dp[i + 1][j - 1];

if (dp[i][j] && len > maxLen) {

start = i;

maxLen = len;

}

}

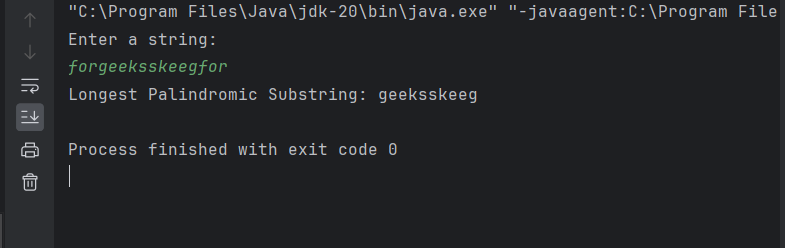
}

}

return s.substring(start, start + maxLen);

}

TC - O(n\*\*2)

SC -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;

class Solution {

public String longestCommonPrefix(String[] arr, int n) {

if (n == 0) return "-1";

Arrays.sort(arr);

String first = arr[0];

String last = arr[n - 1];

int i = 0;

while (i < first.length() && i < last.length()) {

if (first.charAt(i) != last.charAt(i)) break;

i++;

}

String prefix = first.substring(0, i);

return prefix.isEmpty() ? "-1" : prefix;

}

public static void main(String[] args) {

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

int n = arr.length;

Solution sol = new Solution();

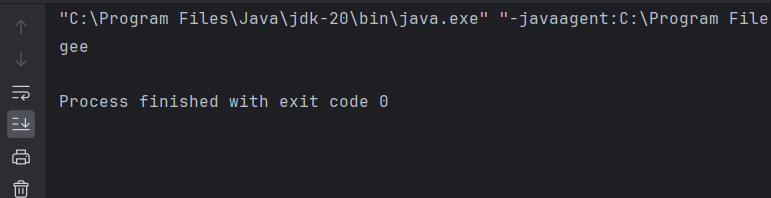
String result = sol.longestCommonPrefix(arr, n);

System.out.println(result);

}

}

**TC - O( N log N + M)  
SC- O(1)**

**Output:**  
  


========================================================================

**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.Stack;

class Solution {

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

if (size % 2 == 0) {

size++;

}

int middleIndex = size / 2;

int currentIndex = 0;

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

while (!stack.isEmpty()) {

int element = stack.pop();

if (currentIndex != middleIndex) {

tempStack.push(element);

}

currentIndex++;

}

while (!tempStack.isEmpty()) {

stack.push(tempStack.pop());

}

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.push(5);

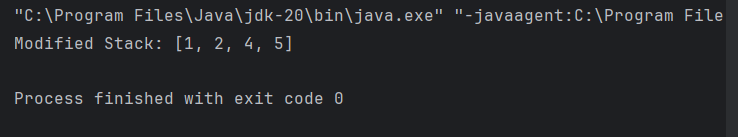
Solution sol = new Solution();

sol.deleteMiddle(stack, stack.size());

System.out.println("Modified Stack: " + stack);

}

}

**TC -O(n)+O(n)  
SC O(n)**  
  
**Output:**  
  
  
  
======================================================================== **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**

**CODE :**   
 public void nextGreaterElement(int[] arr, int n) {

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

int[] result = new int[n];

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

result[i] = -1;

}

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

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

result[stack.pop()] = arr[i];

}

stack.push(i);

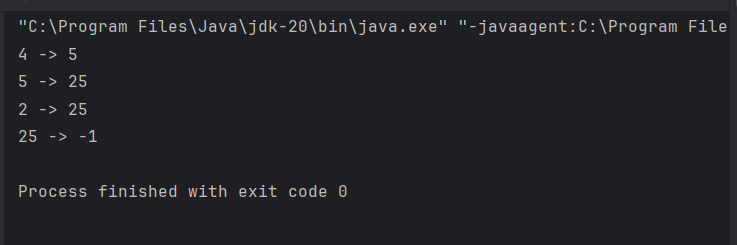
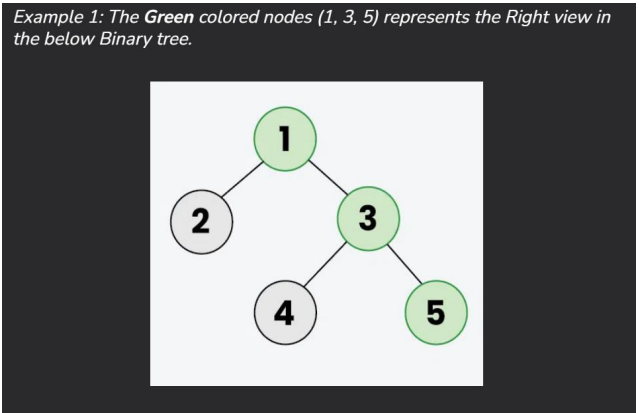
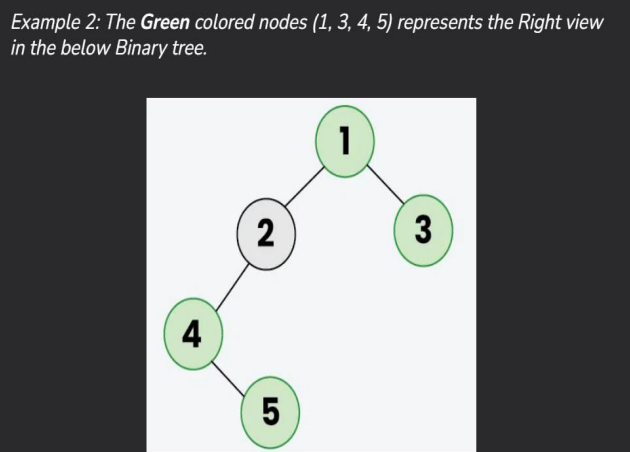
}

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

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

}

}

**TC- O(n)  
SC-O(n)**  
  
========================================================================  
  
**19. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.**  
**Code :**   
import java.util.\*;

class TreeNode {

int value;

TreeNode left, right;

TreeNode(int val) {

value = val;

left = right = null;

}

}

public class Main {

public List<Integer> getRightView(TreeNode root) {

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

exploreRight(root, rightView, 0);

return rightView;

}

public void exploreRight(TreeNode current, List<Integer> rightView, int depth) {

if (current == null) {

return;

}

if (depth == rightView.size()) {

rightView.add(current.value);

}

exploreRight(current.right, rightView, depth + 1);

exploreRight(current.left, rightView, depth + 1);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.right.left = new TreeNode(4);

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

Main tree = new Main();

List<Integer> rightView = tree.getRightView(root);

System.out.print("Right side view of the tree: ");

for (int val : rightView) {

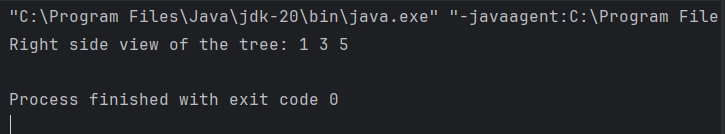
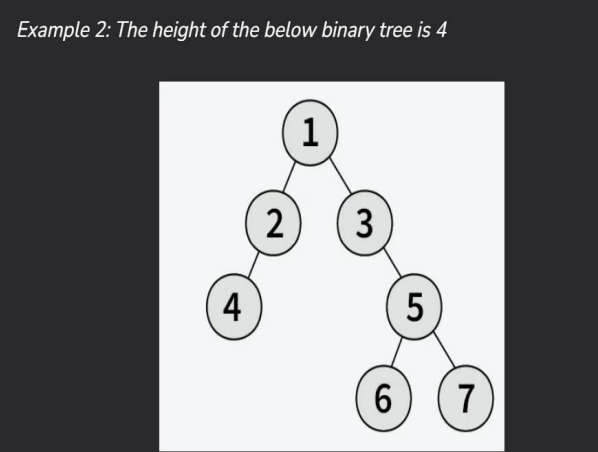
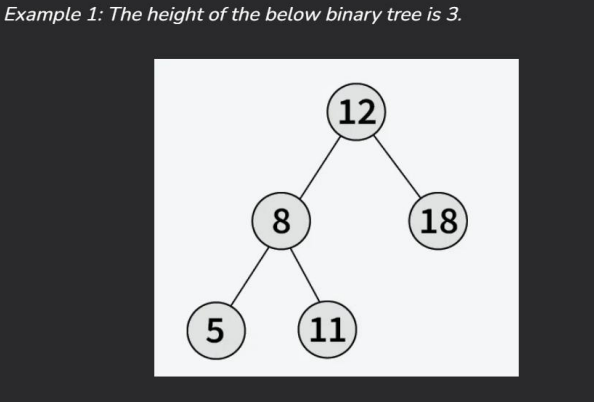
System.out.print(val + " ");

}

System.out.println();

}

}

**TC - O(N)  
SC-O(H+ N):   
OUTPUT:**  
  
  
  
  
  
  
  
========================================================================  
 **20. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.**  
  
  


**Code :**   
class Node {

int value;

Node left, right;

Node(int x) {

value = x;

left = right = null;

}

}

public class Main {

public int findMaxDepth(Node root) {

if (root == null) {

return 0;

}

int leftDepth = findMaxDepth(root.left);

int rightDepth = findMaxDepth(root.right);

return Math.max(leftDepth, rightDepth) + 1;

}

public static void main(String[] args) {

Node rootNode = new Node(1);

rootNode.left = new Node(2);

rootNode.right = new Node(3);

rootNode.left.left = new Node(4);

rootNode.right.left = new Node(5);

rootNode.right.right = new Node(6);

rootNode.right.right.left = new Node(7);

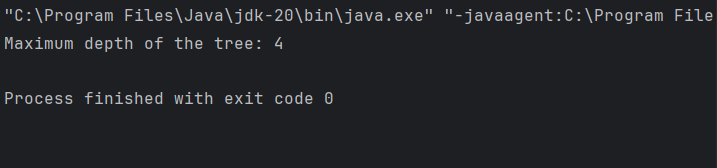
Main tree = new Main();

int maxDepth = tree.findMaxDepth(rootNode);

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

}

}

**TC -O(N)  
SC -O(H)  
  
  
output :**  
  
  
========================================================================