**Coding practice Problems (10/11/2024)**

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}

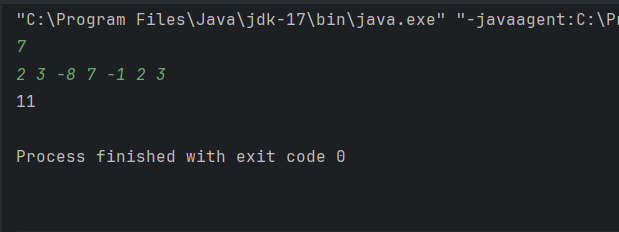
Output: –2

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

**Source Code:**

import java.util.\*;  
  
class MainClass {  
 public int max\_subarray(int[] arr){  
 int max = Integer.*MIN\_VALUE*;  
 int sum = 0;  
 for(int i=0; i<arr.length; i++){  
 sum+=arr[i];  
 if(sum>max){  
 max = sum;  
 }  
 if (sum<0){  
 sum = 0;  
 }  
 }  
 return max;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 MainClass c = new MainClass();  
 int res = c.max\_subarray(arr);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**



**Time Complexity**: O(N)

1. **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}

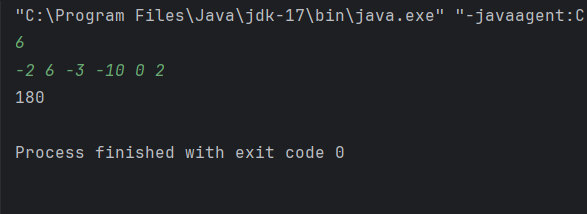
Output: 60

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

**SOURCECODE:**

import java.util.\*;  
  
class MainClass {  
 public int maxProdSubarray(int[] arr){  
 int l=1 ,r = 1;  
 int prod = arr[0];  
 for(int i=0; i<arr.length; i++){  
 l = l==0? 1:l;  
 r = r==0? 1:r;  
 l\*=arr[i];  
 r\*= arr[arr.length-1-i];  
 prod = Math.*max*(prod,Math.*max*(l,r));  
 }  
 return prod;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 MainClass c = new MainClass();  
 int res = c.maxProdSubarray(arr);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**

****

**Time Complexity :O(N)**

1. **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

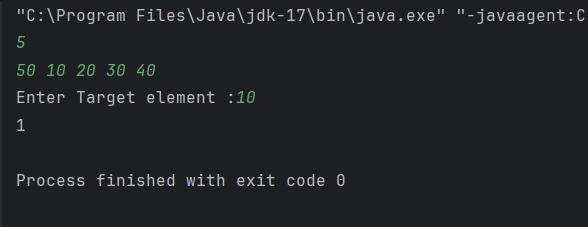
Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

Output : -1

**SOURCECODE:**

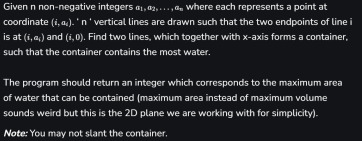
import java.util.\*;  
  
class MainClass {  
 public int search\_sort(int[] arr, int target){  
 int n = arr.length;  
 int low = 0;  
 int high = n-1;  
 while(low<high){  
 int mid = low+high-low/2;  
 if (target==arr[mid]){  
 return mid;  
 }  
 else if(arr[mid]>arr[low]){  
 if(target>=arr[low] && target<arr[mid]){  
 high = mid-1;  
 }  
 else{  
 low = mid+1;  
 }  
 }  
 else{  
 if(target<=arr[high] && target>arr[mid]){  
 low = mid+1;  
 }  
 else{  
 high = mid-1;  
 }  
 }  
 }  
 return -1;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 System.*out*.print("Enter Target element :");  
 int target = sc.nextInt();  
 MainClass c = new MainClass();  
 int res = c.search\_sort(arr,target);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(logN)**

1. **Container with Most Water**

****

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

Output: 6

Explanation:

5 and 3 are distance 2 apart. So the size of the base = 2.

Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

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

Output: 12

Explanation:

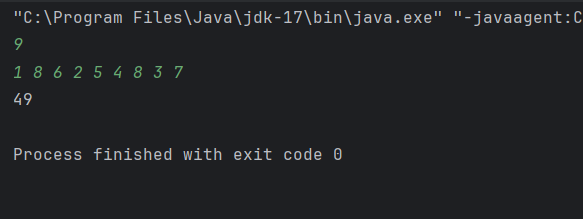
5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**SOURCECODE:**

import java.util.\*;  
  
class MainClass {  
 public int waterCapacity(int[] arr){  
 int water = 0;  
 int left = 0;  
 int right = arr.length-1;  
 while(left<right){  
 water = Math.*max*(water,Math.*min*(arr[left],arr[right])\*(right-left));  
 if(arr[left]<arr[right]){  
 left++;  
 }  
 else{  
 right--;  
 }  
 }  
 return water;  
 }  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 MainClass c = new MainClass();  
 int res = c.waterCapacity(arr);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(logN)**

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

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000 00

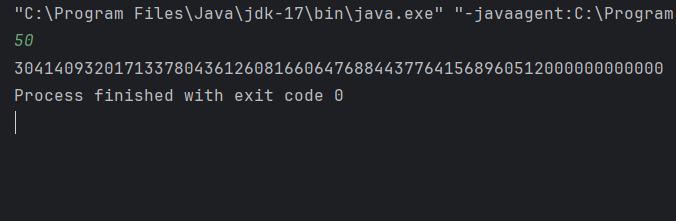
Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**SOURCECODE:**

import java.util.\*;  
  
class MainClass {  
 public void factorial(int n){  
 int[] res = new int[500];  
 res[0] = 1;  
 int re\_size = 1;  
 for(int i=2; i<=n; i++){  
 re\_size = *multiplication*(i,res,re\_size);  
 }  
 for(int i= re\_size-1; i>=0; i--){  
 System.*out*.print(res[i]);  
 }  
 }  
  
 public static int multiplication(int x, int[]res, int re\_size){  
 int carry = 0;  
 int n;  
 for(int i=0; i<re\_size; i++){  
 n = res[i]\*x+carry;  
 res[i] = n%10;  
 carry = n/10;  
 }  
 while(carry!=0){  
 res[re\_size] = carry%10;  
 carry/=10;  
 re\_size++;  
 }  
 return re\_size;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 MainClass c = new MainClass();  
 c.factorial(n);  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(Nlog(N!))**

1. **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}

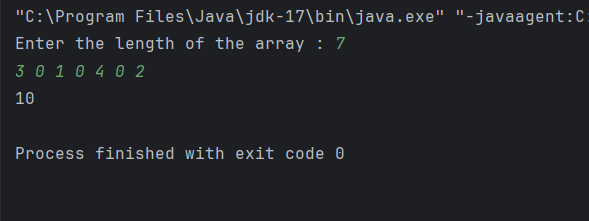
Output: 5

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

**SOURCECODE:**

import java.net.SocketOption;  
import java.util.\*;  
  
class MainClass {  
 public int trapRainWater(int[] arr){  
 int left = 0;  
 int right = arr.length -1;  
 int leftMax = arr[0];  
 int rightMax = arr[arr.length -1];  
 int water = 0;  
 while(left<right){  
 if(arr[left]<arr[right]){  
 left++;  
 leftMax = Math.*max*(leftMax,arr[left]);  
 water += leftMax-arr[left];  
 }  
 else{  
 right--;  
 rightMax = Math.*max*(rightMax, arr[right]);  
 water += rightMax-arr[right];  
 }  
 }  
 return water;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter the length of the array : ");  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 MainClass c = new MainClass();  
 int res = c.trapRainWater(arr);  
 System.*out*.println(res);  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(logN)**

1. **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

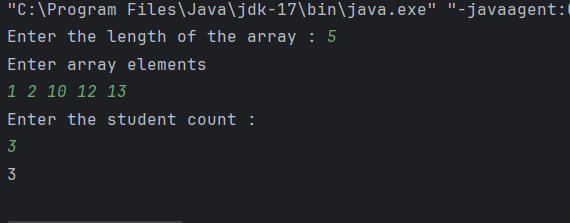
Output: 7

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

**SOURCECODE:**

import java.net.SocketOption;  
import java.util.\*;  
  
class MainClass {  
 public int chocolateDistribution(int[] arr, int m){  
 Arrays.*sort*(arr);  
 int mini = Integer.*MAX\_VALUE*;  
 for(int i=0;i<arr.length-m+1; i++){  
 mini = Math.*min*(mini,arr[i+m-1]-arr[i]);  
 }  
 return mini;  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter the length of the array : ");  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 System.*out*.println("Enter array elements");  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 System.*out*.println("Enter the student count : ");  
 int m = sc.nextInt();  
 MainClass c = new MainClass();  
 int res = c.chocolateDistribution(arr,m);  
 System.*out*.println(res);  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(NlogN)**

**8. A 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]]

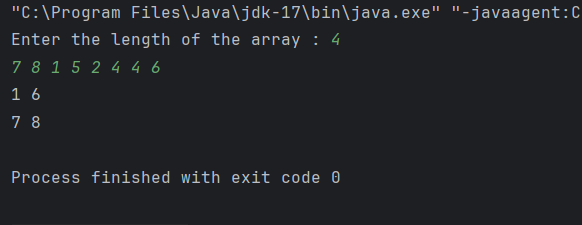
Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval

[1, 6].

**SOURCECODE:**

import java.net.SocketOption;  
import java.util.\*;  
import java.lang.\*;  
  
class MainClass {  
 public int[][] mergeOverlapping(int[][] arr){  
 Arrays.*sort*(arr, (a,b)-> Integer.*compare*(a[0],b[0]));  
 int[][] res = new int[arr.length-1][2];  
 int x =0;  
 res[x] = arr[x];  
 for(int i=1; i<= arr.length-1; i++){  
 if(res[x][1]>arr[i][0] && res[x][0]<arr[i][0]){  
 res[x][1] = Math.*max*(arr[i][1],res[x][1]);  
 }  
 else{  
 x++;  
 res[x] = arr[i];  
 }  
 }  
 return res;  
 }  
 public static void main(String[] args) {  
 MainClass c = new MainClass();  
 Scanner sc = new Scanner(System.*in*);  
 System.*out*.print("Enter the length of the array : ");  
 int n = sc.nextInt();  
 int[][] arr = new int[n][2];  
 int x,y;  
 for(int i=0; i<n; i++) {  
 x = sc.nextInt();  
 y = sc.nextInt();  
 int[] temp = {x, y};  
 arr[i] = temp;  
 }  
 int[][] res = c.mergeOverlapping(arr);  
 for(int i=0; i<=res.length-1; i++){  
 if(res[i][0]==0){  
 break;  
 }  
 System.*out*.print(res[i][0]+" ");  
 System.*out*.println(res[i][1]);  
 }  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(NlogN)**

**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},

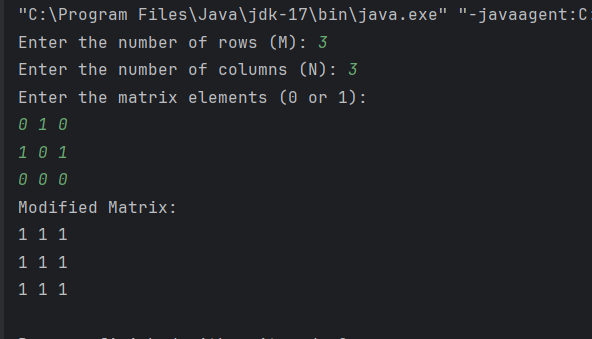
{1, 1, 1, 1},

{1, 0, 1, 1}}

**SOURCECODE:**

import java.util.Scanner;  
import java.util.HashSet;  
  
public class MainClass {  
 public static void modifyMatrix(int[][] mat) {  
 int m = mat.length;  
 int n = mat[0].length;  
 HashSet<Integer> rowsToUpdate = new HashSet<>();  
 HashSet<Integer> colsToUpdate = new HashSet<>();  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 if (mat[i][j] == 1) {  
 rowsToUpdate.add(i);  
 colsToUpdate.add(j);  
 }  
 }  
 }  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 if (rowsToUpdate.contains(i) || colsToUpdate.contains(j)) {  
 mat[i][j] = 1;  
 }  
 }  
 }  
 }  
 public static void printMatrix(int[][] mat) {  
 for (int i = 0; i < mat.length; i++) {  
 for (int j = 0; j < mat[0].length; j++) {  
 System.*out*.print(mat[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 }  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter the number of rows (M): ");  
 int m = scanner.nextInt();  
 System.*out*.print("Enter the number of columns (N): ");  
 int n = scanner.nextInt();  
 int[][] mat = new int[m][n];  
 System.*out*.println("Enter the matrix elements (0 or 1):");  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 mat[i][j] = scanner.nextInt();  
 }  
 }  
 System.*out*.println("Modified Matrix:");  
 *modifyMatrix*(mat);  
 *printMatrix*(mat);  
 }  
}

**OUTPUT:**



**Time Complexity: O(M\*N)**

**10.** **Print a given matrix in spiral form**

Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

Input: matrix = {{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16 }}

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

Input: matrix = { {1, 2, 3, 4, 5, 6},

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

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

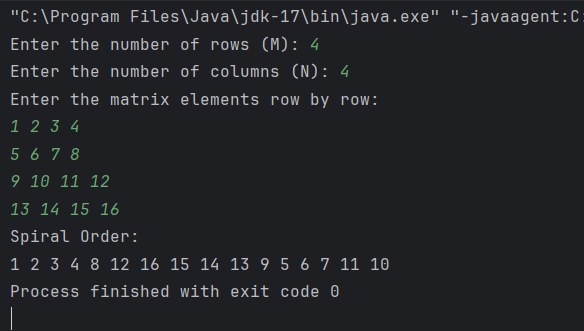
Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

Explanation: The output is matrix in spiral format.

**SOURCECODE:**

import java.util.Scanner;  
import java.util.HashSet;  
  
public class MainClass {  
 public static void printSpiral(int[][] matrix) {  
 if (matrix == null || matrix.length == 0) {  
 return;  
 }  
 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++;  
 }  
 }  
 }  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter the number of rows (M): ");  
 int m = scanner.nextInt();  
 System.*out*.print("Enter the number of columns (N): ");  
 int n = scanner.nextInt();  
 int[][] matrix = new int[m][n];  
 System.*out*.println("Enter the matrix elements row by row:");  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 matrix[i][j] = scanner.nextInt();  
 }  
 }  
 System.*out*.println("Spiral Order:");  
 *printSpiral*(matrix);  
 }  
}

**OUTPUT:**



**Time Complexity : O(M\*N)**

**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

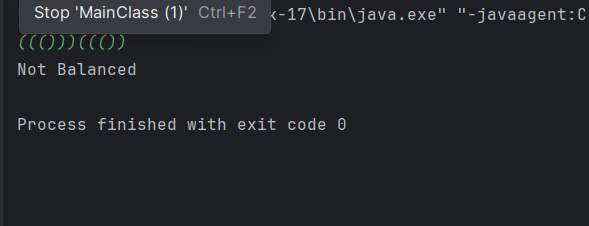
Input: str = “())((())”

Output: Not Balanced

**SOURCECODE:**

import java.util.\*;  
import java.lang.\*;  
  
class MainClass {  
 public String balancing(String s){  
 Stack<Character> st = new Stack<>();  
 for(int i=0; i<s.length(); i++){  
 char c = s.charAt(i);  
 if(c!=')') {  
 st.push(c);  
 }  
 else{  
 if(!st.isEmpty()) {  
 st.pop();  
 }  
 else{  
 return "Not Balanced";  
 }  
 }  
 }  
 if(st.isEmpty()){  
 return "Balanced";  
 }  
 return "Not Balanced";  
 }  
 public static void main(String[] args) {  
 MainClass c = new MainClass();  
 Scanner sc = new Scanner(System.*in*);  
 String s = sc.nextLine();  
 String res = c.balancing(s);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**

****

**Time Complexity : O(N)**

**14. Check if two Strings are Anagrams of each other**

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the

two given strings are anagrams of each other or not. An anagram of a string is another string that

contains the same characters, only the order of characters can be different.

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic”

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has

extra characters „i‟ and „c‟, so they are not anagrams.

Input: s1 = “g”, s2 = “g”

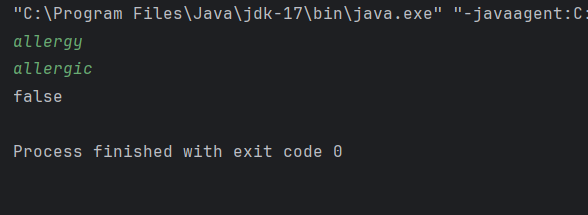
Output: true

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

**SOURCECODE:**

import java.util.\*;  
import java.lang.\*;  
  
class MainClass {  
 public static String sortString(String s) {  
 char tempArr[] = s.toCharArray();  
 Arrays.*sort*(tempArr);  
 return new String(tempArr);  
 }  
 public Boolean is\_anagram(String s1, String s2){  
 String sortedS1 = *sortString*(s1);  
 String sortedS2 = *sortString*(s2);  
 return sortedS1.equalsIgnoreCase(sortedS2);  
 }  
 public static void main(String[] args) {  
 MainClass c = new MainClass();  
 Scanner sc = new Scanner(System.*in*);  
 String s1 = sc.nextLine();  
 String s2 = sc.nextLine();  
 Boolean res = c.is\_anagram(s1,s2);  
 System.*out*.println(res);  
  
 }  
}

**OUTPUT:**



**Time Complexity : O(NlogN)**

**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: “”

**SOURCECODE:**

import java.util.Scanner;

class MainClass {

private static String expandAroundCenter(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 String longestPalindrome(String s) {

if (s == null || s.length() < 2) {

return s;

}

String longest = "";

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

String oddPalindrome = expandAroundCenter(s, i, i);

if (oddPalindrome.length() > longest.length()) {

longest = oddPalindrome;

}

String evenPalindrome = expandAroundCenter(s, i, i + 1);

if (evenPalindrome.length() > longest.length()) {

longest = evenPalindrome;

}

}

return longest;

}

public static void main(String[] args) {

MainClass c = new MainClass();

Scanner sc = new Scanner(System.in);

String s1 = sc.nextLine();

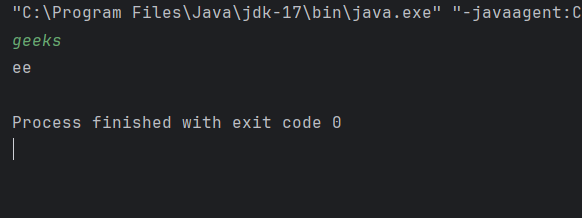
String res = c.longestPalindrome(s1);

System.out.println(res);

}

}

**OUTPUT:**



**Time Complexity : O(N)**

**16. Longest Common Prefix using Sorting**

Given an array of strings arr[]. The task is to retu. 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.

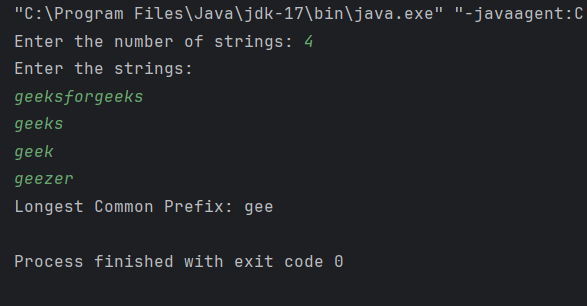
Input: arr[] = [“hello”, “world”]

Output: -1

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

import java.util.Scanner;  
import java.util.Arrays;  
  
public class MainClass{  
 public static String longestCommonPrefix(String[] arr) {  
 if (arr == null || arr.length == 0) {  
 return "-1";  
 }  
 Arrays.*sort*(arr);  
 String first = arr[0];  
 String last = arr[arr.length - 1];  
 int i = 0;  
 while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {  
 i++;  
 }  
 if (i > 0) {  
 return first.substring(0, i);  
 }  
 return "-1";  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter the number of strings: ");  
 int n = scanner.nextInt();  
 scanner.nextLine();  
 String[] arr = new String[n];  
 System.*out*.println("Enter the strings:");  
 for (int i = 0; i < n; i++) {  
 arr[i] = scanner.nextLine();  
 }  
 System.*out*.println("Longest Common Prefix: " + *longestCommonPrefix*(arr));  
  
 }  
}

**OUTPUT:**



**Time Complexity : O(NlogN)**

**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]

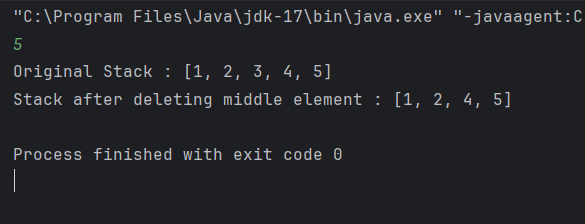
Input : Stack[] = [1, 2, 3, 4, 5, 6]

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

**SOURCECODE:**

import java.util.\*;  
  
class MainClass {  
 private static void deleteMiddle(Stack<Integer> stack, int currInd, int midInd) {  
 if (currInd == midInd) {  
 stack.pop();  
 return;  
 }  
 int top = stack.pop();  
 *deleteMiddle*(stack, currInd + 1, midInd);  
 stack.push(top);  
 }  
 public static void deleteMiddle(Stack<Integer> stack) {  
 int middleIndex = stack.size() / 2;  
 *deleteMiddle*(stack, 0, middleIndex);  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 Stack<Integer> st = new Stack<>();  
 for(int i=1; i<=n; i++){  
 st.push(i);  
 }  
 System.*out*.println("Original Stack : "+ st);  
 *deleteMiddle*(st);  
 System.*out*.println("Stack after deleting middle element : "+ st);  
 }  
}

**OUTPUT:**



**Time Complexity : O(N)**

**18.** **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

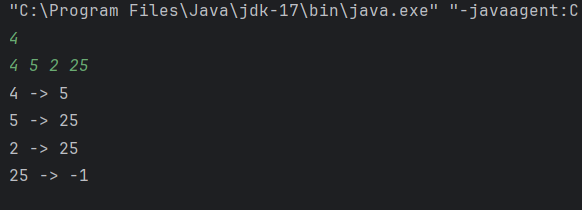
12 –> -1

Explanation: 13 and 12 don‟t have any element greater than them present on the right side

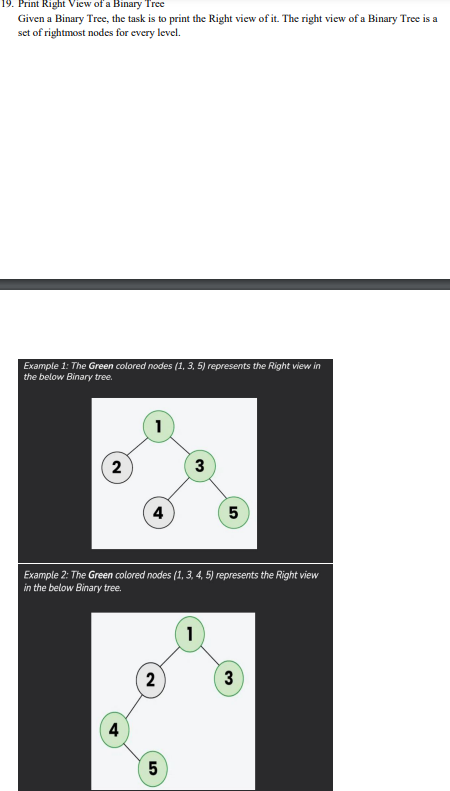
**SOURCECODE:**

import java.util.\*;  
class MainClass {  
 public static void findNextGreaterElements(int[] arr) {  
 int n = arr.length;  
 int[] nge = new int[n];  
 Stack<Integer> stack = new Stack<>();  
 for (int i = n - 1; i >= 0; i--) {  
 while (!stack.isEmpty() && stack.peek() <= arr[i]) {  
 stack.pop();  
 }  
 nge[i] = stack.isEmpty() ? -1 : stack.peek();  
 stack.push(arr[i]);  
 }  
  
 for (int i = 0; i < n; i++) {  
 System.*out*.println(arr[i] + " -> " + nge[i]);  
 }  
 }  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for(int i=0; i<n; i++){  
 arr[i] = sc.nextInt();  
 }  
 *findNextGreaterElements*(arr);  
 }  
}

**OUTPUT:**



**Time Complexity : O(NlogN)**



**SOURCECODE:**

import java.util.\*;

class TreeNode {

int data;

TreeNode left, right;

public TreeNode(int data) {

this.data = data;

left = right = null;

}

}

public class BinaryTreeRightView {

public static List<Integer> rightView(TreeNode root) {

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

if (root == null) {

return result;

}

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

TreeNode currentNode = queue.poll();

if (i == levelSize - 1) {

result.add(currentNode.data);

}

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

}

}

return result;

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

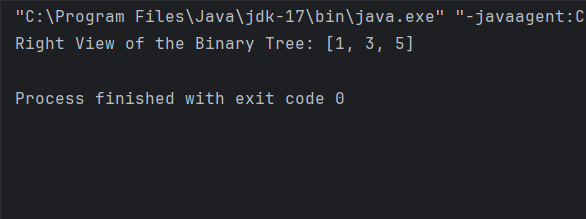
List<Integer> rightView = rightView(root);

System.out.println("Right View of the Binary Tree: " + rightView);

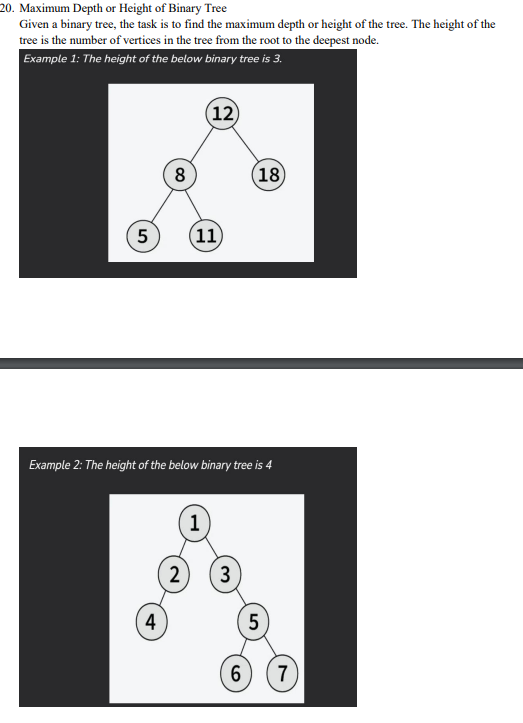
}

}

OUTPUT:



Time Complexity : O(N)

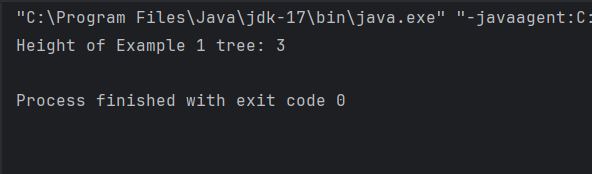


SOURCE CODE:

class TreeNode {  
 int val;  
 TreeNode left;  
 TreeNode right;  
  
 TreeNode(int x) {  
 val = x;  
 left = null;  
 right = null;  
 }  
}

public class BinaryTree {  
 public 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;  
 }  
  
 public static void main(String[] args) {  
 TreeNode root1 = new TreeNode(12);  
 root1.left = new TreeNode(8);  
 root1.right = new TreeNode(18);  
 root1.left.left = new TreeNode(5);  
 root1.left.right = new TreeNode(11);  
  
 BinaryTree tree1 = new BinaryTree();  
 System.*out*.println("Height of Example 1 tree: " + tree1.maxDepth(root1));  
  
  
 }  
}

**OUTPUT:**



**Time Complexity : O(N)**