**DSA Coding Problems**

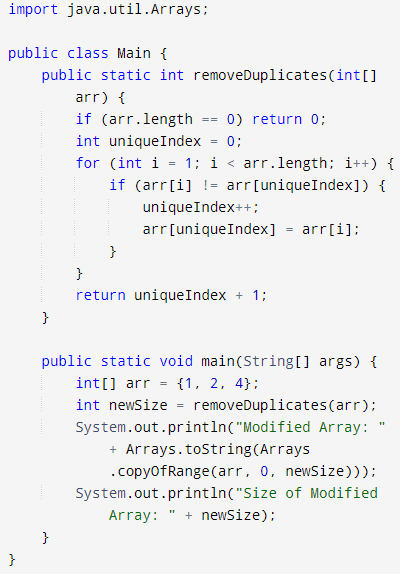
**1.Remove Duplicates Elements in the List:**

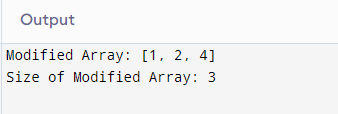
Given a sorted array arr. Return the size of the modified array which contains only distinct elements.  
Note:  
1. Don't use set or HashMap to solve the problem.  
2. You must return the modified array size only where distinct elements are present and modify the original array such that all the distinct elements come at the beginning of the original array.

Input: arr = [1, 2, 4]

Output: [1, 2, 4]  
Explation: As the array does not contain any duplicates so you should return 3.

Code:





**Time Complexity: O(n)**

**2.First Repeating Element:**

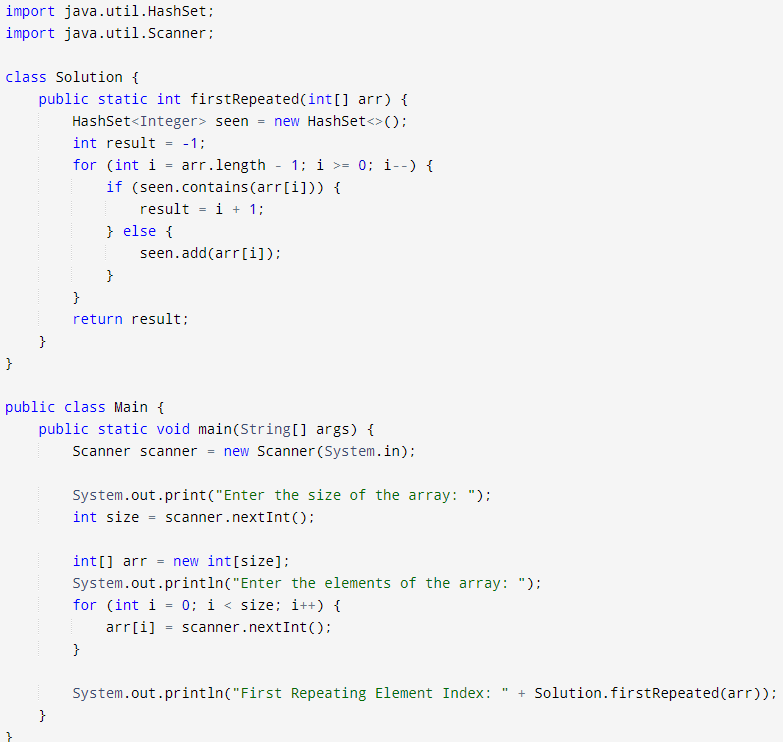
Given an array arr[], find the first repeating element. The element should occur ore than once and the index of its first occurrence should be the smallest.

Note:- The position you return should be according to 1-based indexing.

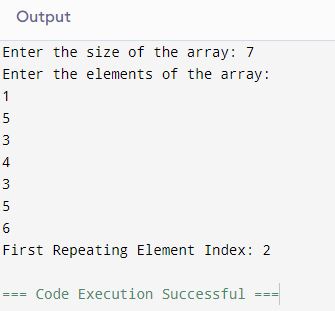
Input: arr[] = [1, 5, 3, 4, 3, 5, 6]

Output: 2

Explanation: 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.



**Time Complexity: O(n)**



**Time Complexity: O(n**

**3.Find Transition Point:**

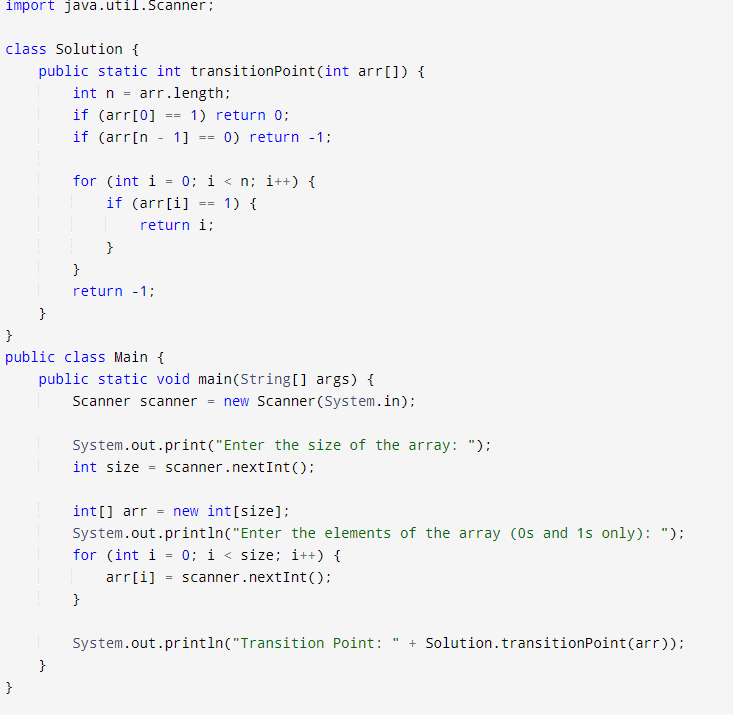
Given a sorted array, arr[] containing only 0s and 1s, find the transition point, i.e., the first index where 1 was observed, and before that, only 0 was observed.  If arr does not have any 1, return -1. If array does not have any 0, return 0.

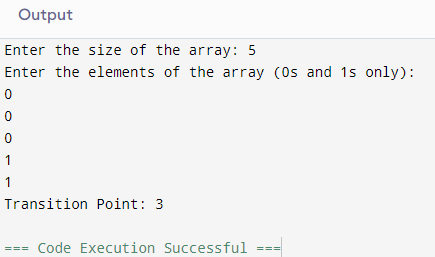
Input: arr[] = [0, 0, 0, 1, 1]

Output: 3

Explanation: index 3 is the transition point where 1 begins.

Code:





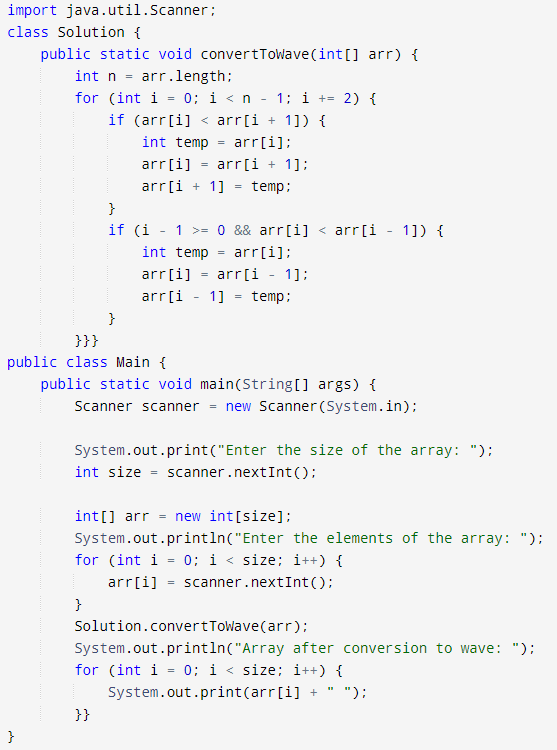
**Time Complexity: O(n)**

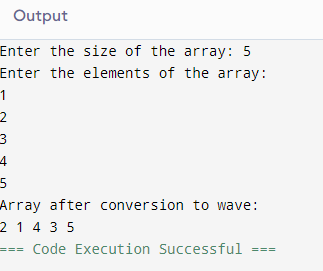
**4.Wave Array:**

Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....  
If there are multiple solutions, find the lexicographically smallest one.

Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array

Code:





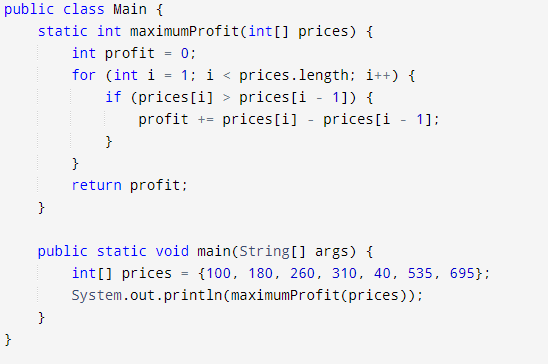
**Time Complexity: O(n**)

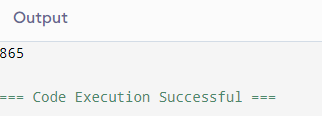
**5.Stock buy and Sell:**

Given an array prices[] of size n denoting the cost of stock on each day, the task is to find the maximum total profit if we can buy and sell the stocks any number of times.

Note: We can only sell a stock which we have bought earlier and we cannot hold multiple stocks on any day.

Input: prices[] = {100, 180, 260, 310, 40, 535, 695}  
Output:865  
Explanation: Buy the stock on day 0 and sell it on day 3 => 310 – 100 = 210  
                       Buy the stock on day 4 and sell it on day 6 => 695 – 40 = 655  
                       Maximum Profit  = 210 + 655 = 865





**Time Complexity: O(n)**

**6.Coin Change:**

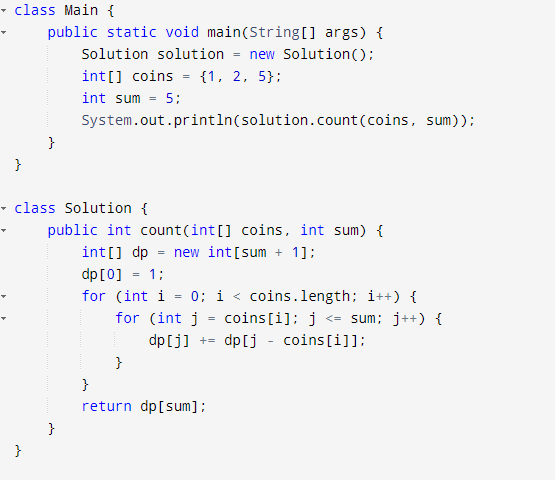
Given an integer array coins[ ] representing different denominations of currency and an integer sum, find the number of ways you can make sum by using different combinations from coins[ ].   
Note: Assume that you have an infinite supply of each type of coin. And you can use any coin as many times as you want.  
Answers are guaranteed to fit into a 32-bit integer.

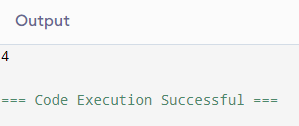
Input: coins[] = [1, 2, 3], sum = 4

Output: 4

Explanation: Four Possible ways are: [1, 1, 1, 1], [1, 1, 2], [2, 2], [1, 3].

Code:





**Time Complexity: O(n)**

**7.First and Last Occurences:**

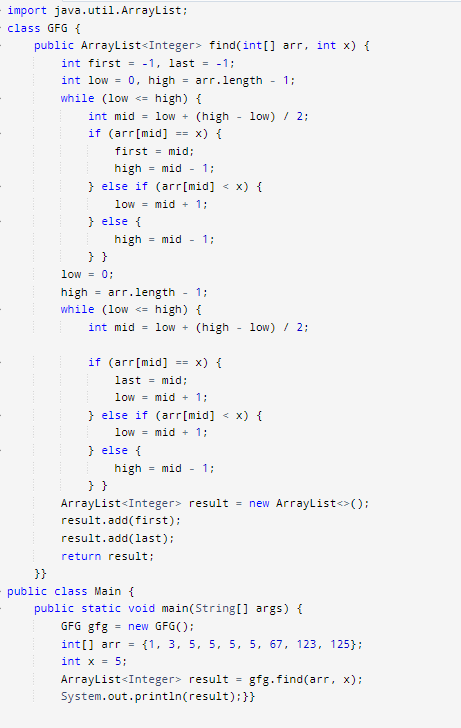
Given a sorted array arr with possibly some duplicates, the task is to find the first and last occurrences of an element x in the given array.  
Note: If the number x is not found in the array then return both the indices as -1.

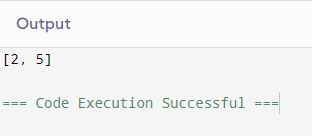
Input: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Output: [2, 5]

Explanation: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

Code:





**Time Complexity: O(n)**

**8.Maximum Index:**

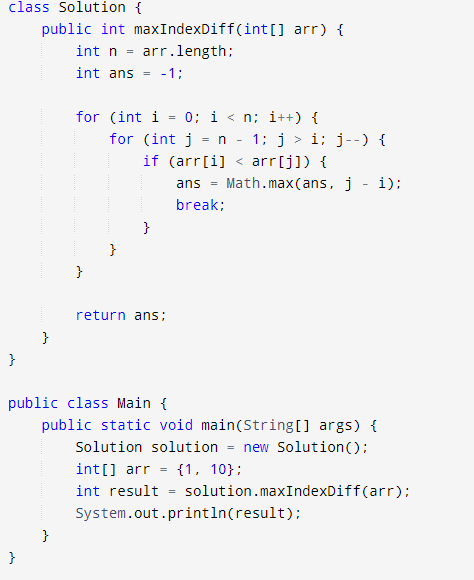
Given an array arr of positive integers. The task is to return the maximum of j - i subjected to the constraint of arr[i] < arr[j] and i < j.

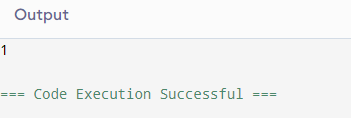
Input: arr[] = [1, 10]

Output: 1

Explanation: arr[0] < arr[1] so (j-i) is 1-0 = 1.

Code:





**Time Complexity: O(n)**

**9.Kth Smallest Number:**

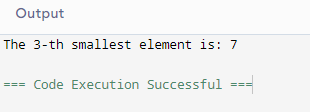
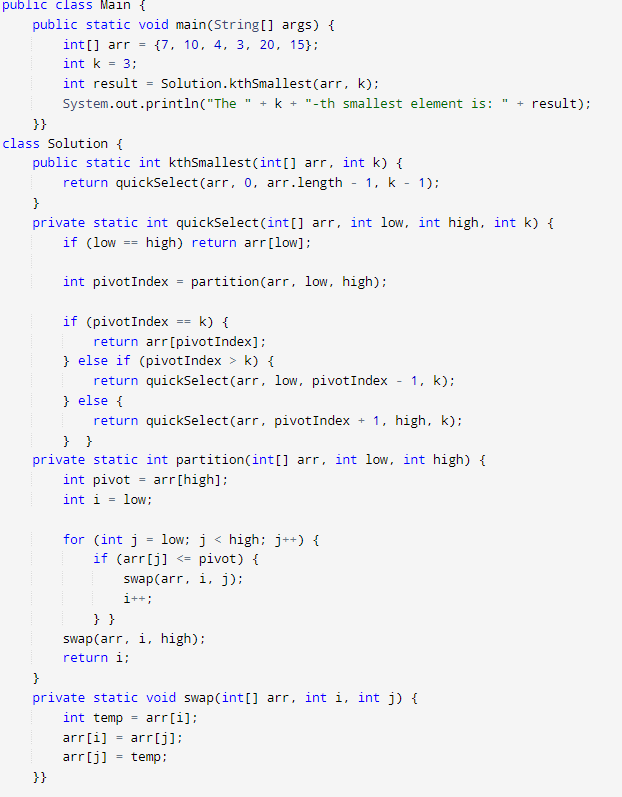
Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth smallest element in the given array.

Note: Don't solve it using the inbuilt sort function.

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3

Output: 7

Code:



**Time Complexity: O(n)**

**10.Minimize the Height 2’s**

Given an array arr[] denoting heights of N towers and a positive integer K.

For each tower, you must perform exactly one of the following operations exactly once.

Increase the height of the tower by K

Decrease the height of the tower by K

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.

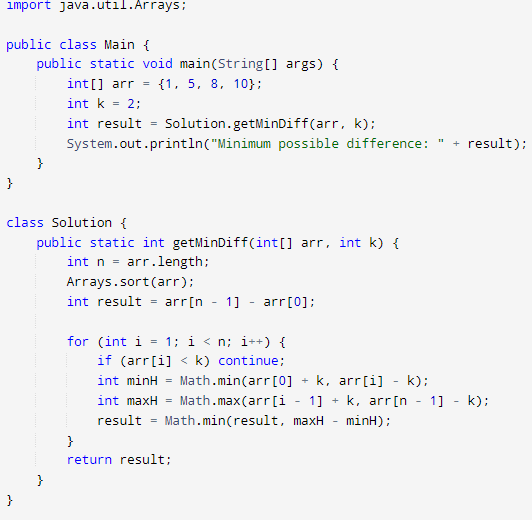
You can find a slight modification of the problem [here](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/).  
Note: It is compulsory to increase or decrease the height by K for each tower. After the operation, the resultant array should not contain any negative integers

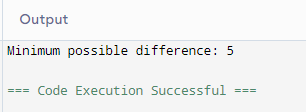
Input: k = 2, arr[] = {1, 5, 8, 10}

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

Code:





**Time Complexity: O(n)**

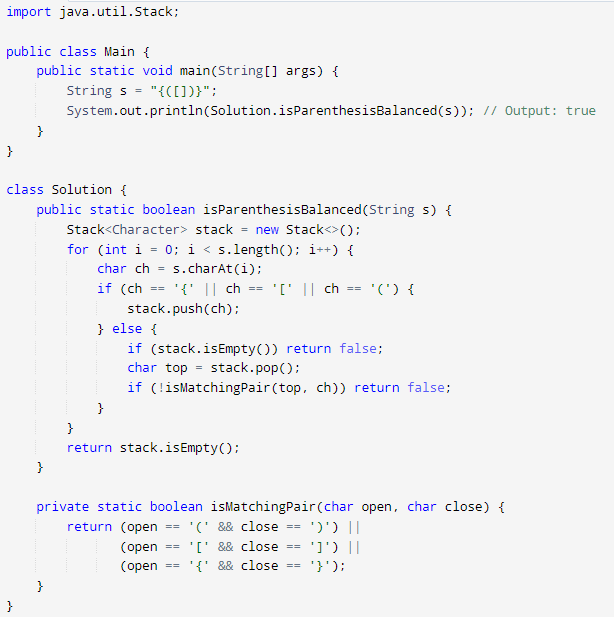
**11.Parenthesis Checker:**

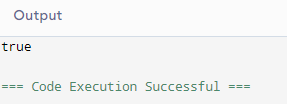
You are given a string s representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

Input: s = "{([])}"

Output: true

Code:





**Time Complexity: O(n)**

**12.Equilibrium Point:**

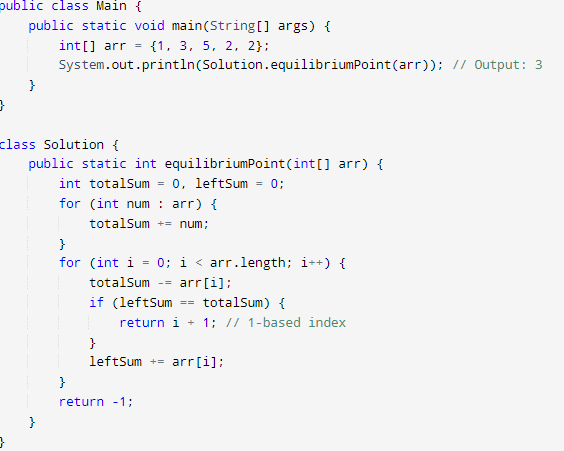
Given an array arr of non-negative numbers. The task is to find the first equilibrium point in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements before that index is the same as the sum of elements after it.

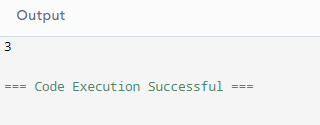
Note: Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

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

Explanation: The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

Code:

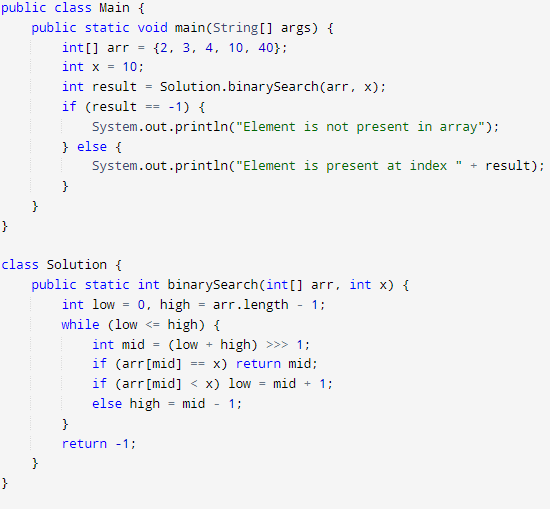


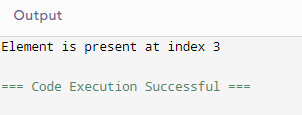


**Time Complexity: O(n)**

**13.Binary Search:**

Code:





**Time Complexity: O(n)**

**14. Next Greater Element:**

Given an array arr[ ] of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.  
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

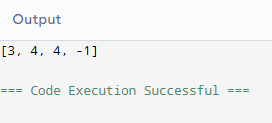
Input: arr[] = [1, 3, 2, 4]

Output: [3, 4, 4, -1]

Explanation: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

Code:





**Time Complexity: O(n)**

**15.Union Of Two Arrays:**

Given two arrays a[] and b[], the task is to find the number of elements in the union between these two arrays.

The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.

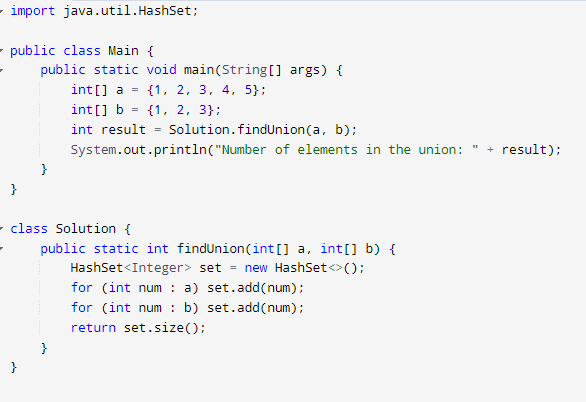
Note: Elements are not necessarily distinct.

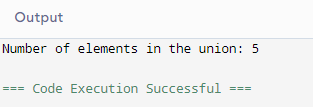
Input: a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]

Output: 5

Explanation: 1, 2, 3, 4 and 5 are the elements which comes in the union set of both arrays. So count is 5.

Code:





**Time Complexity: O(n)**

**16.Anagram:**

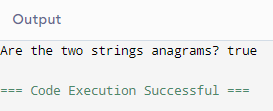
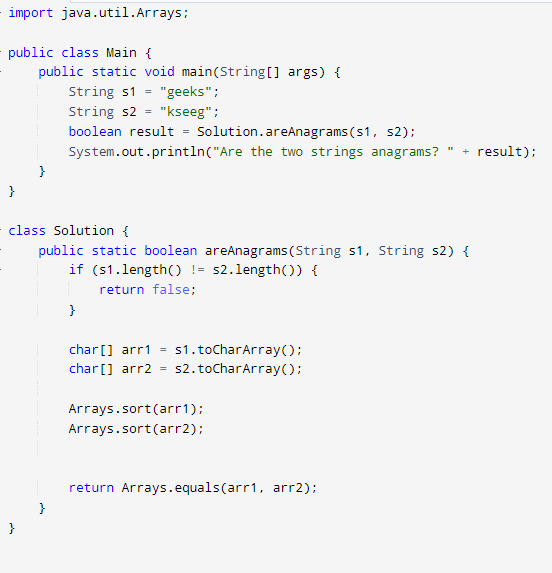
Given two strings s1 and s2 consisting of lowercase characters. The task is to check whether two given strings are an anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, act and tac are an anagram of each other. Strings s1 and s2 can only contain lowercase alphabets.

Note: You can assume both the strings s1 & s2 are non-empty.

Input: s1 = "geeks", s2 = "kseeg"

Output: true

Code:



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

**17.Row with Max 1’s:**

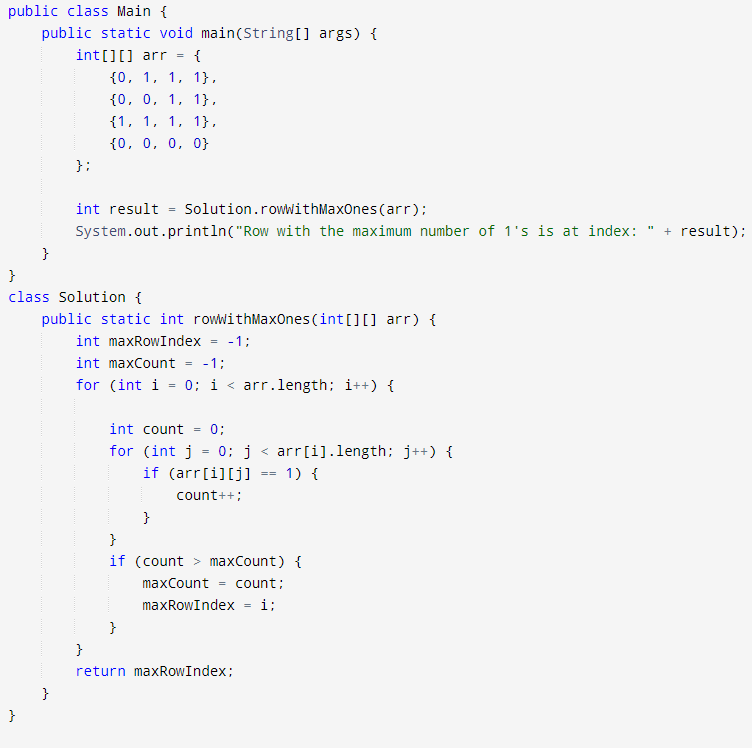
You are given a 2D array consisting of only 1's and 0's, where each row is sorted in non-decreasing order. You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return -1.  
Note: 0-based indexing is followed.

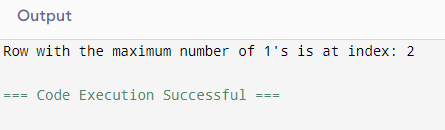
Input: arr[][] = [[0, 1, 1, 1], [0, 0, 1, 1], [1, 1, 1, 1], [0, 0, 0, 0]]

Output: 2

Explanation: Row 2 contains 4 1's.

Code:





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

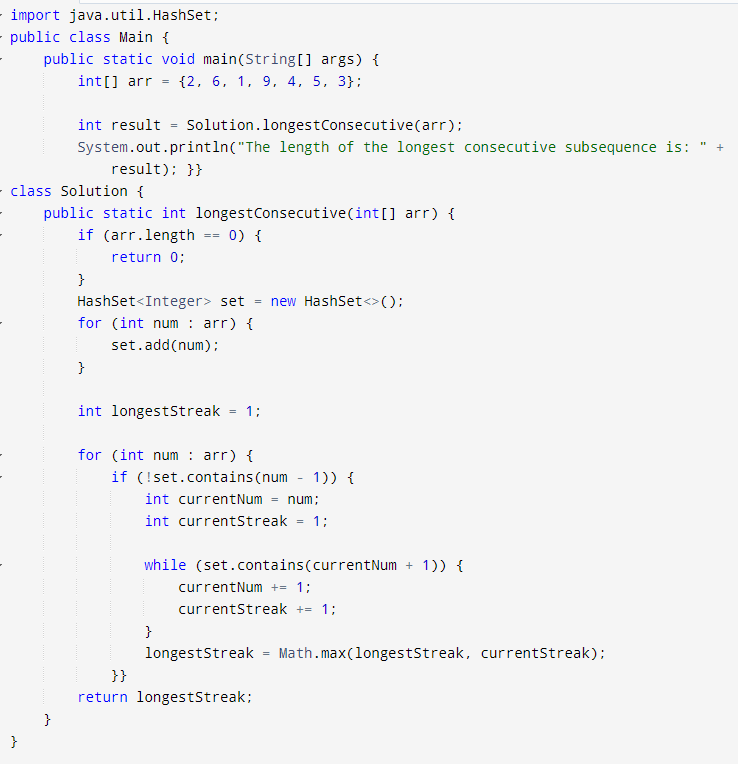
**18.Longest consecutive sequence:**

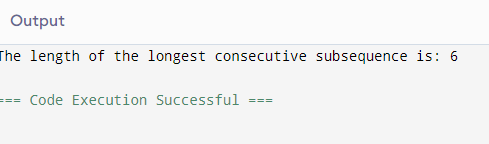
Given an array arr of non-negative integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Input: arr[] = [2, 6, 1, 9, 4, 5, 3]

Output: 6

**Code:**





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

**19.Longest palindrome in string**

Given a string s, your task is to find the longest palindromic substring within s. A substring is a contiguous sequence of characters within a string, defined as s[i...j] where 0 ≤ i ≤ j < len(s).

A palindrome is a string that reads the same forward and backward. More formally, s is a palindrome if reverse(s) == s.

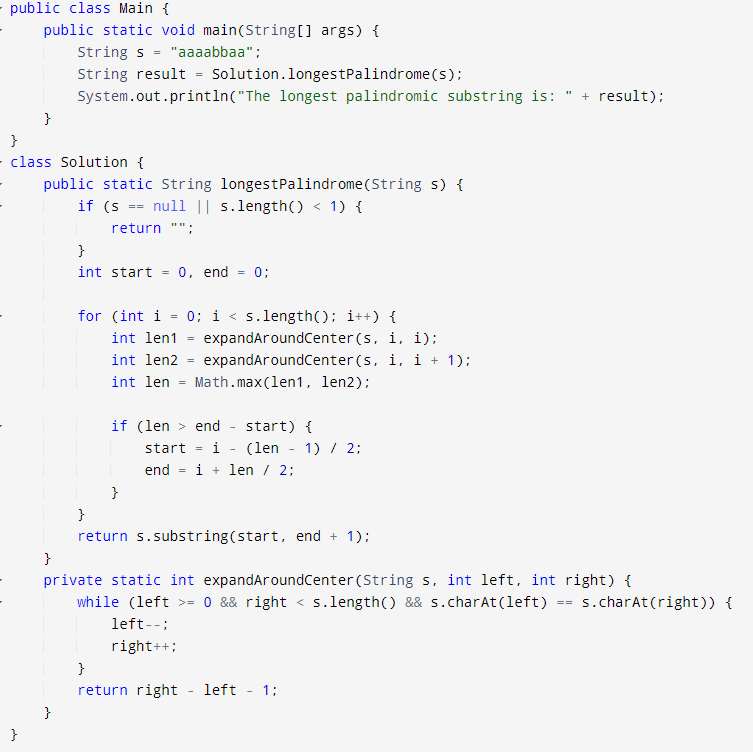
Note: If there are multiple palindromes with the same length, return the first occurrence of the longest palindromic substring from left to right.

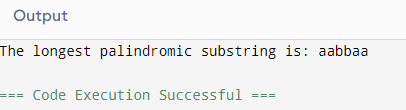
Input: s = "aaaabbaa"

Output: "aabbaa"

Explanation: The longest palindromic substring is "aabbaa".

Code:





**Time Complexity:** O(n^3)

**20.Rat in the maze**

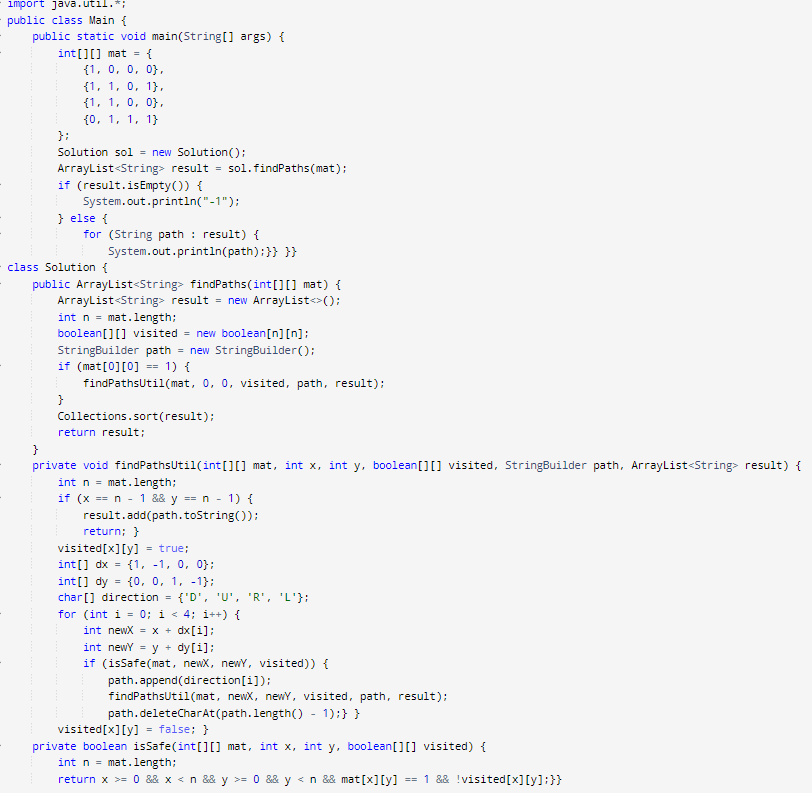
Consider a rat placed at (0, 0) in a square matrix mat of order n\* n. It has to reach the destination at (n - 1, n - 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.  
Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell. In case of no path, return an empty list. The driver will output "-1" automatically.

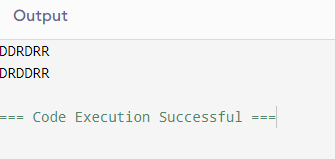
Input: mat[][] = [[1, 0, 0, 0], [1, 1, 0, 1], [1, 1, 0, 0], [0, 1, 1, 1]]

Output: DDRDRR DRDDRR

Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

Code:





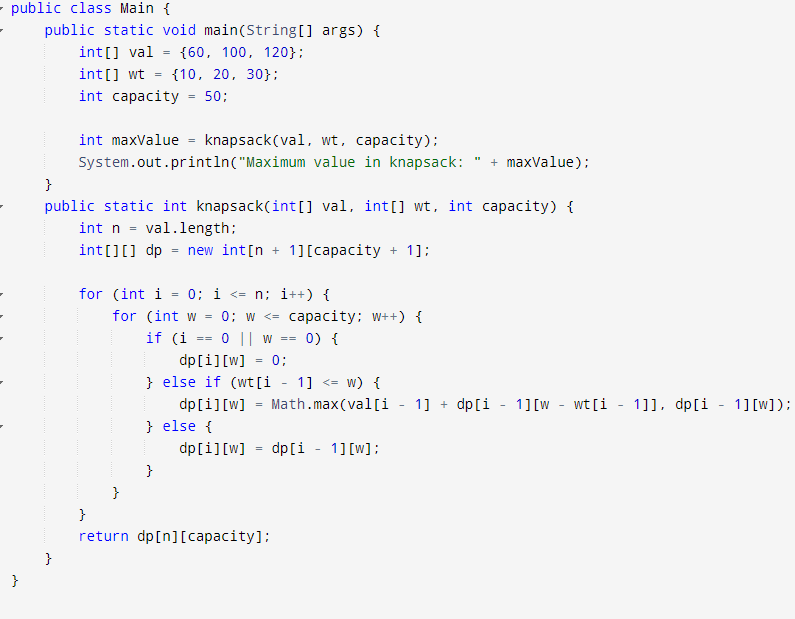
**Time Complexity:** O(3^(m\*n))

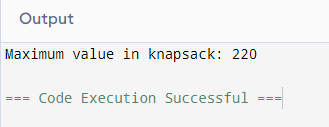
**21.Knapsack Problem:**

You are given the weights and values of items, and you need to put these items in a knapsack of capacity capacity to achieve the maximum total value in the knapsack. Each item is available in only one quantity. In other words, you are given two integer arrays val[] and wt[], which represent the values and weights associated with items, respectively. You are also given an integer capacity, which represents the knapsack capacity. Your task is to find the maximum sum of values of a subset of val[] such that the sum of the weights of the corresponding subset is less than or equal to capacity. You cannot break an item; you must either pick the entire item or leave it (0-1 property).

Time Complexity: O(n\*w)

Code:





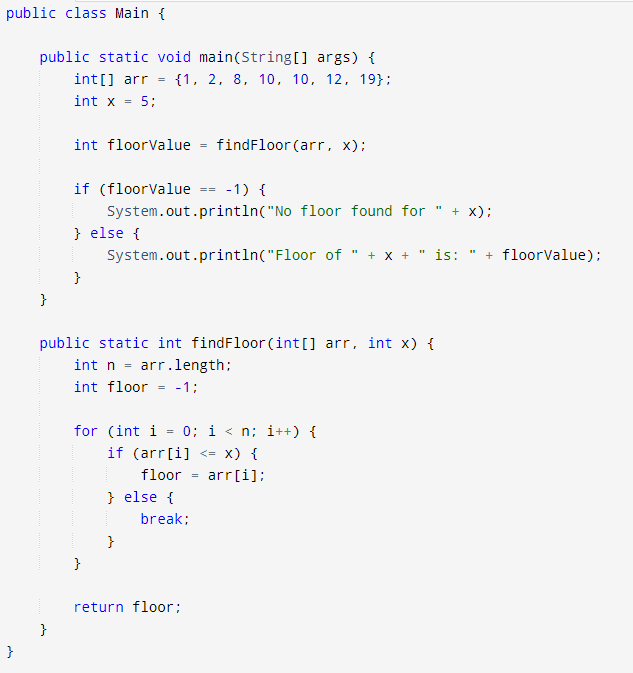
Time Complexity: O(N)

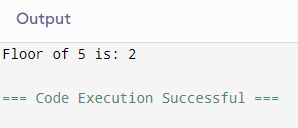
**22.Floor in sorted array:**

Given a sorted array and a value x, the floor of x is the largest element in the array smaller than or equal to x. Write efficient functions to find the floor of x .

Time Complexity: O(N)

Code:





Time Complexity: O(N)

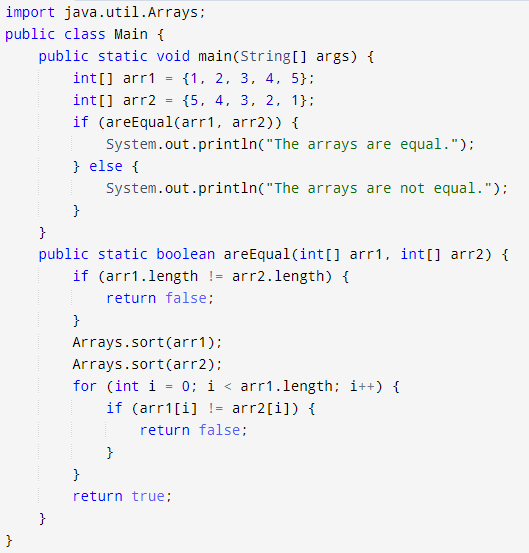
**23.Check equal arrays:**

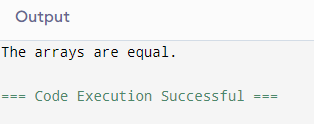
Given two arrays arr1 and arr2 of equal size, the task is to find whether the given arrays are equal. Two arrays are said to be equal if both contain the same set of elements, arrangements (or permutations) of elements may be different though.

Note: If there are repetitions, then counts of repeated elements must also be the same for two arrays to be equal.

Time Complexity: O(n)

Code:

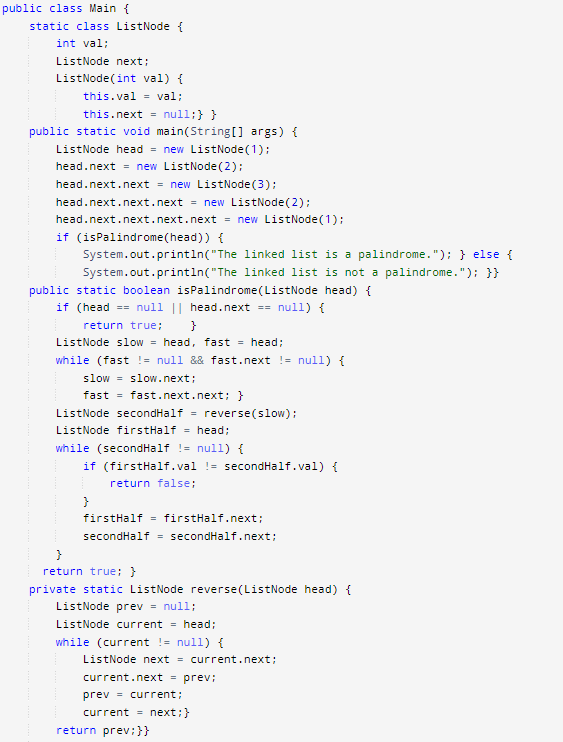


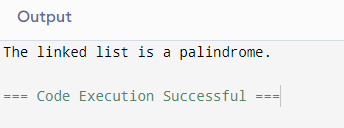


Time Complexity: O(N)

**24. Palindrome Linked List:**

Given a singly linked list of integers. The task is to check if the given linked list is palindrome or not.





Time Complexity: O(N)

**25.Balanced Tree Check:**

Given a binary tree, find if it is height balanced or not. A tree is height balanced if difference between heights of left and right subtrees is not more than one for all nodes of tree.

Time Complexity: O(n)

**Input:**

1

/

2

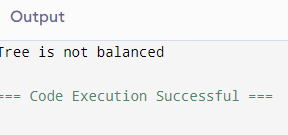
\

3

**Output:** 0

Code:



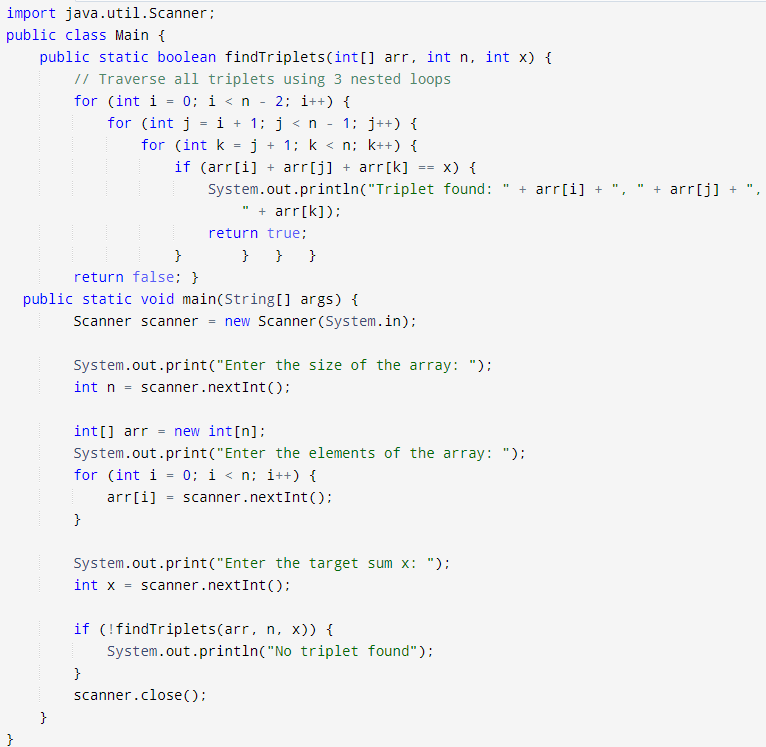


**26.Triplets Sum in Array:**

Given an array arr of size n and an integer x. Find if there's a triplet in the array which sums up to the given integer x. Examples Input:n = 6, x = 13, arr[] = [1,4,45,6,10,8] Output: 1 Explanation: The triplet {1, 4, 8} in the array sums up to 13.

**Input**:n = 6, x = 13, arr[] = [1,4,45,6,10,8]

**Output**: 1



Time Complexity: O(n^3)