Count all increasing subsequences - GeeksforGeeks

Source: https://www.geeksforgeeks.org/count-all-increasing-subsequences/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Count all increasing subsequences Last Updated : 23 Jul, 2025 Given an array arr[] of integers (values lie in range from 0 to 9). The task is to count the number of strictly increasing subsequences that can be formed from this array. Note: A strictly increasing subsequence is a sequence where each element is greater than the previous one, and the elements are taken in the same order as they appear in the original array. Examples: Input: arr[] = [1, 2, 3] Output: 7 Explanation: All strictly increasing subsequences are: [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3] Input: arr[] = [3, 1, 2] Output: 5 Explanation: Strictly increasing subsequences are: [3], [1], [2], [1,2], [3] (counted individually, no [3,1] or [3,2] as they are not increasing) Input: arr[] = [4, 4, 4] Output: 3 Explanation: Only single-element subsequences [4], [4], [4] are increasing. No multi-element increasing subsequences possible. Try it on GfG Practice Table of Content [Approach 1] Using Dynamic Programming - O(n^2) Time and O(n) Space [Approach 2] Using Optimized Dynamic Programming - O(n) Time and O(1) Space [Approach 1] Using Dynamic Programming - O(n^2) Time and O(n) Space Recursive Structure in the problem : // We count all increasing subsequences ending at every index i subCount(i) = Count of increasing subsequences ending at arr[i]. // Like LIS , this value can be recursively computed subCount(i) = 1 + SUM(subCount(j)) where j is index of all elements such that arr[j] < arr[i] and j < i. 1 is added as every element itself is a subsequence of size 1. // Finally we add all counts to get the result. Result = SUM(subCount(i)) where i varies from 0 to n-1. The idea is use Dynamic Programming . The thought process is similar to the LIS (Longest Increasing Subsequence) problem, but instead of max length, we store counts in the dp array. For every i , we check all previous j < i and if arr[j] < arr[i] , we add dp[j] to dp[i] . Finally, the sum of all values in dp gives the total number of increasing subsequences. C++ // C++ Code to count strictly increasing subsequences // Using Dynamic Programming #include <bits/stdc++.h> using namespace std ; int countSub (vector < int >& arr) { int n = arr . size (); // dp[i] will store count of strictly // increasing subsequences ending at index i vector < int > dp (n , 1); // Calculate dp[i] for each i using // previous values for (int i = 1 ; i < n ; i ++) { for (int j = 0 ; j < i ; j ++) { // If a valid strictly // increasing condition if (arr [j] < arr [i]) { dp [i] += dp [j]; } } } int total = 0 ; // Sum of all dp[i] will give total // increasing subsequences for (int i = 0 ; i < n ; i ++) { total += dp [i]; } return total ; } // Driver Code int main () { vector < int > arr = { 1 , 2 , 3 }; cout << countSub (arr) << endl ; return 0 ; } Java // Java Code to count strictly increasing subsequences // Using Dynamic Programming import java.util.* ; class GfG { static int countSub (int [] arr) { int n = arr . length ; // dp[i] will store count of strictly // increasing subsequences ending at index i int [] dp = new int [n] ; Arrays . fill (dp , 1); // Calculate dp[i] for each i using // previous values for (int i = 1 ; i < n ; i ++) { for (int j = 0 ; j < i ; j ++) { // If a valid strictly // increasing condition if (arr [j] < arr [i]) { dp [i] += dp [j] ; } } } int total = 0 ; // Sum of all dp[i] will give total // increasing subsequences for (int i = 0 ; i < n ; i ++) { total += dp [i] ; } return total ; } public static void main (String [] args) { int [] arr = { 1 , 2 , 3 }; System . out . println (countSub (arr)); } } Python # Python Code to count strictly increasing subsequences # Using Dynamic Programming def countSub (arr): n = len (arr) # dp[i] will store count of strictly # increasing subsequences ending at index i dp = [1] * n # Calculate dp[i] for each i using # previous values for i in range (1 , n): for j in range (i): # If a valid strictly # increasing condition if arr [j] < arr [i]: dp [i] += dp [j] total = 0 # Sum of all dp[i] will give total # increasing subsequences for i in range (n): total += dp [i] return total if __name__ == "__main__" : arr = [1 , 2 , 3] print (countSub (arr)) C# // C# Code to count strictly increasing subsequences // Using Dynamic Programming using System ; class GfG { static int countSub (int [] arr

) { int n = arr . Length ; // dp[i] will store count of strictly // increasing subsequences ending at index i int [] dp = new int [n]; for (int i = 0 ; i < n ; i ++) { dp [i] = 1 ; } // Calculate dp[i] for each i using // previous values for (int i = 1 ; i < n ; i ++) { for (int j = 0 ; j < i ; j ++) { // If a valid strictly // increasing condition if (arr [j] < arr [i]) { dp [i] += dp [j]; } } } int total = 0 ; // Sum of all dp[i] will give total // increasing subsequences for (int i = 0 ; i < n ; i ++) { total += dp [i]; } return total ; } static void Main (string [] args) { int [] arr = { 1 , 2 , 3 }; Console . WriteLine (countSub (arr)); } } JavaScript // JavaScript Code to count strictly increasing subsequences // Using Dynamic Programming function countSub (arr) { let n = arr . length ; // dp[i] will store count of strictly // increasing subsequences ending at index i let dp = new Array (n). fill (1); // Calculate dp[i] for each i using // previous values for (let i = 1 ; i < n ; i ++) { for (let j = 0 ; j < i ; j ++) { // If a valid strictly // increasing condition if (arr [j] < arr [i]) { dp [i] += dp [j]; } } } let total = 0 ; // Sum of all dp[i] will give total // increasing subsequences for (let i = 0 ; i < n ; i ++) { total += dp [i]; } return total ; } // Driver Code let arr = [1 , 2 , 3]; console . log (countSub (arr)); Output 7 [Approach 2] Using Optimized Dynamic Programming - O(n) Time and O(1) Space The idea is to optimize the previous approach based on the observation that the values in the array are only digits from 0 to 9 . The thought process is to maintain a count[] array where count[d] stores the number of increasing subsequences ending with digit d . For each element, we add 1 + sum of count[d] for all digits less than it. For example, arr[] = {3, 2, 4, 5, 4} // We create a count array and initialize it as 0. count[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} // Note that here value is used as index to store counts count[3] += 1 // i = 0, arr[0] = 3 = 1 count[2] += 1 // i = 1, arr[1] = 2 = 1 // Let us compute count for arr[2] which is 4 count[4] += 1 + count[3] + count[2] += 1 + 1 + 1 = 3 // Let us compute count for arr[3] which is 5 count[5] += 1 + count[3] + count[2] + count[4] += 1 + 1 + 1 + 3 = 6 // Let us compute count for arr[4] which is 4 count[4] += 1 + count[0] + count[1] += 1 + 1 + 1 += 3 = 3 + 3 = 6 Note that count[] = {0, 0, 1, 1, 6, 6, 0, 0, 0, 0} Result = count[0] + count[1] + ... + count[9] = 1 + 1 + 6 + 6 {count[2] = 1, count[3] = 1 count[4] = 6, count[5] = 6} = 14 C++ // C++ Code to count strictly increasing subsequences // Using Optimized Dynamic Programming #include <bits/stdc++.h> using namespace std ; int countSub (vector < int >& arr) { // There are only digits 0-9 vector < int > count (10 , 0); for (int i = 0 ; i < arr . size (); i ++) { int curr = arr [i]; // Count all increasing subsequences // ending before current digit int sum = 0 ; for (int d = 0 ; d < curr ; d ++) { sum += count [d]; } // Current digit contributes one (itself) // + sum of previous smaller count [curr] += (1 + sum); } int total = 0 ; // Sum of all counts gives the total // strictly increasing subsequences for (int i = 0 ; i < 10 ; i ++) { total += count [i]; } return total ; } // Driver Code int main () { vector < int > arr = { 1 , 2 , 3 }; cout << countSub (arr) << endl ; return 0 ; } Java // Java Code to count strictly increasing subsequences // Using Optimized Dynamic Programming class GfG { public static int countSub (int [] arr) { // There are only digits 0-9 int [] count = new int [10] ; for (int i = 0 ; i < arr . length ; i ++) { int curr = arr [i] ; // Count all increasing subsequences // ending before current digit int sum = 0 ; for (int d = 0 ; d < curr ; d ++) { sum += count [d] ; } // Current digit contributes one (itself) // + sum of previous smaller count [curr] += (1 + sum); } int total = 0 ; // Sum of all counts gives the total // strictly increasing subsequences for (int i = 0 ; i < 10 ; i ++) { total += count [i] ; } return total ; } public static void main (String [] args) { int [] arr = { 1 , 2 , 3 }; System . out . println (countSub (arr)); } } Python # Python Code to count strictly increasing subsequences # Using Optimized Dynamic Programming def countSub (arr): # There are only digits 0-9 count = [0] * 10 for i in range (len (arr)): curr = arr [i] # Count all increasing subsequences # ending before current digit sum = 0 for d in range (curr): sum += count [d] # Current digit contributes one (itself) # + sum of previous smaller count [curr] += (1 + sum) total = 0 # Sum of all counts gives the total # strictly increasing subsequences for i in range (10): total += count [i] return total if __name__ == "__main__" : arr = [1 , 2 , 3] print (countSub (arr)) C# // C# Code to count strictly increasing subsequences // Using Optimized Dynamic Programming using System ; class GfG { public static int countSub (int [] arr) { // There are only digits 0-9 int [] count = new int [10]; for (int i = 0 ; i < arr . Length ; i ++) { int curr = arr [i]; // Count all increasing subsequences // ending before current digit int sum = 0 ; for (int d = 0 ; d < curr ; d ++) { sum += count [d]; } // Current digit contributes one (itself) // + sum of previous smaller count [curr] += (1 + sum); } int total = 0 ; // Sum of all counts gives the total // strictly increasing subsequences for (int i = 0 ; i < 10 ; i ++) { total += count [i]; } return total ; } public static void Main () { int [] arr = { 1 , 2 , 3 }; Console . WriteLine (countSub (arr)); } } Javascript // JavaScript Code to count strictly increasing subsequences // Using Optimized Dynamic Programming function countSub (arr) { // There are only digits 0-9 let count = new Array (10). fill (0); for (let i = 0 ; i < arr . length ; i ++) { let curr = arr [i]; // Count all increasing subsequences // ending before current digit let sum = 0 ; for (let d = 0 ; d < curr ; d ++) { sum += count [d]; } // Current digit contributes one (itself) // + sum of previous smaller count [curr] += (1 + sum); } let total = 0 ; // Sum of all counts gives the total // strictly

increasing subsequences for (let i = 0 ; i < 10 ; i ++) { total += count [i]; } return total ; } // Driver Code let arr = [1 , 2 , 3]; console . log (countSub (arr)); Output 7 Comment Article Tags: Article Tags: Dynamic Programming DSA Arrays subsequence