Two Pointers Technique - GeeksforGeeks

Source: https://www.geeksforgeeks.org/two-pointers-technique/

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 Two Pointers Technique Last Updated : 13 Feb, 2026 The Two-Pointers Technique is a simple yet powerful strategy where you use two indices (pointers) that traverse a data structure - such as an array, list, or string - either toward each other or in the same direction to solve problems more efficiently Two pointers is really an easy and effective technique that is typically used for Two Sum in Sorted Arrays , Closest Two Sum , Three Sum , Four Sum , Trapping Rain Water and many other popular interview questions. When to Use Two Pointers: Sorted Input : If the array or list is already sorted (or can be sorted), two pointers can efficiently find pairs or ranges. Example: Find two numbers in a sorted array that add up to a target. Pairs or Subarrays : When the problem asks about two elements, subarrays, or ranges instead of working with single elements. Example: Longest substring without repeating characters, maximum consecutive ones, checking if a string is palindrome. Sliding Window Problems : When you need to maintain a window of elements that grows/shrinks based on conditions. Example: Find smallest subarray with sum ≥ K, move all zeros to end while maintaining order. Linked Lists (Slow–Fast pointers) : Detecting cycles, finding the middle node, or checking palindrome property. Example: Floyd’s Cycle Detection Algorithm (Tortoise and Hare). Example Problem - Sum of Pair Equal to Target Given a sorted array arr (sorted in ascending order) and a target , find if there exists any pair of elements (arr[i], arr[j]) such that their sum is equal to the target. Illustration : Input : arr[] = [10, 20, 35, 50], target =70 Output : true Explanation : There is a pair (20, 50) with given target. Input : arr[] = [10, 20, 30], target =70 Output : false Explanation : There is no pair with sum 70 Input : arr[] = [-8, 1, 4, 6, 10, 45], target = 16 Output : true Explanation : There is a pair (6, 10) with given target. Table of Content Naive Method - O(n^2) Time and O(1) Space Two-Pointer Technique - O(n) time and O(1) space How does this work? More problems based on two pointer technique. Naive Method - O(n^2) Time and O(1) Space The very basic approach is to generate all the possible pairs and check if any of them add up to the target value. To generate all pairs, we simply run two nested loops. C++ #include <iostream> #include <vector> using namespace std ; bool twoSum (vector < int > & arr , int target) { int n = arr . size (); // Consider all pairs (arr[i], arr[j]) for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // Check if the sum of the current pair // equals the target if (arr [i] + arr [j] == target) { return true ; } } } // If no pair is found after checking all return false ; } int main () { vector < int > arr = { 0 , -1 , 2 , -3 , 1 }; int target = -2 ; cout << ((twoSum (arr , target)) ? "true" : "false"); return 0 ; } C #include <stdbool.h> #include <stdio.h> // Function to check whether any pair exists // whose sum is equal to the given target value bool twoSum (int arr [], int n , int target){ // Iterate through each element in the array for (int i = 0 ; i < n ; i ++){ // For each element arr[i], check every // other element arr[j] that comes after it for (int j = i + 1 ; j < n ; j ++){ // Check if the sum of the current pair // equals the target if (arr [i] + arr [j] == target) return true ; } } // If no pair is found after checking // all possibilities return false ; } int main (){ int arr [] = { 0 , -1 , 2 , -3 , 1 }; int target = -2 ; int n = sizeof (arr) / sizeof (arr [0]); // Call the twoSum function and print the result if (twoSum (arr , n , target)) printf ("true \n "); else printf ("false \n "); return 0 ; } Java class GfG { // Function to check whether any pair exists // whose sum is equal to the given target value static boolean twoSum (int [] arr , int target){ int n = arr . length ; // Iterate through each element in the array for (int i = 0 ; i < n ; i ++) { // For each element arr[i], check every // other element arr[j] that comes after it for (int j = i + 1 ; j < n ; j ++) { // Check if the sum of the current pair // equals the target if (arr [i] + arr [j] == target) { return true ; } } } // If no pair is found after checking // all possibilities return false ; } public static void main (String [] args){ int [] arr = { 0 , - 1 , 2 , - 3 , 1 }; int target = - 2 ; // Call the twoSum function and print the result if (twoSum (arr , target)) System . out . println ("true"); else System . out . println ("false"); } } Python # Function to check whether any pair exists # whose sum is equal to the given target value def two_sum (arr , target): n = len (arr) # Iterate through each element in the array for i in range (n): # For each element arr[i], check every # other element arr[j] that comes after it for j in range (i + 1 , n): # Check if the sum of the current pair # equals the target if arr [i] + arr [j] == target : return True # If no pair is found after checking # all possibilities return False arr = [0 , - 1 , 2 , - 3 , 1] target = - 2 # Call the two_sum function and print the result if two_sum (arr , target): print ("true") else : print ("false") C# using System ; class GfG { // Function to check whether any pair exists // whose sum is equal to the given target value static bool TwoSum (int [] arr , int target) { int n = arr . Length ; // Iterate through each element in the array for (int i = 0 ; i < n ; i ++) { // For each element arr[i], check every // other element arr[j] that comes after it for (int j = i + 1 ; j < n ; j ++) { // Check if the sum of the current pair // equals the target if (arr [i] + arr [j] == target) { return true ; } } } // If no pair is found after checking // all possibilities return false ; } static void Main () { int [] arr = { 0 , - 1 , 2 , - 3 , 1 }; int target = - 2 ; // Call the TwoSum function and print the result if (TwoSum (arr , target)) Console . WriteLine ("true"); else Console . WriteLine ("false"); } } JavaScript // Function to check whether any pair exists // whose sum is equal to the given target value function twoSum (arr , target) { let n = arr . length ; // Iterate through each element in the array for (let i = 0 ; i < n ; i ++) { // For each element arr[i], check every // other element arr[j] that comes after it for (let j = i + 1 ; j < n ; j ++) { // Check if the sum of the current pair // equals the target if (arr [i] + arr [j] === target) { return true ; } } } // If no pair is found after checking // all possibilities return false ; } let arr = [0 , - 1 , 2 , - 3 , 1]; let target = - 2 ; // Call the twoSum function and print the result if (twoSum (arr , target)) console . log ("true"); else console . log ("false"); Output true Two-Pointer Technique - O(n) time and O(1) space The idea of this technique is to begin with two corners of the given array. We use two index variables left and right to traverse from both corners. Initialize: left = 0, right = n - 1 Run a loop while left < right, do the following inside the loop Compute current sum, sum = arr[left] + arr[right] If the sum equals the target , we've found the pair. If the sum is less than the target , move the left pointer to the right to increase the sum . If the sum is greater than the target , move the right pointer to the left to decrease the sum . Illustration: C++ #include <iostream> #include <vector> using namespace std ; bool twoSum (vector < int > & arr , int target){ int left = 0 , right = arr . size () - 1 ; while (left < right){ int sum = arr [left] + arr [right]; if (sum == target) return true ; // Move toward a higher sum else if (sum < target) left ++ ; // Move toward a lower sum else right -- ; } // If no pair found return false ; } int main (){ vector < int > arr = { -3 , -1 , 0 , 1 , 2 }; int target = -2 ; if (twoSum (arr , target)) cout << "true" ; else cout << "false" ; return 0 ; } C #include <stdbool.h> #include <stdio.h> #include <stdlib.h> // Comparison function for qsort int compare (const void * a , const void * b){ return (* (int *) a - * (int *) b); } // Function to check whether any pair exists // whose sum is equal to the given target value bool twoSum (int arr [], int n , int target){ // Sort the array int left = 0 , right = n - 1 ; // Iterate while left pointer is less than right while (left < right){ int sum = arr [left] + arr [right]; // Check if the sum matches the target if (sum == target) return true ; else if (sum < target) left ++ ; // Move left pointer to the right else right -- ; // Move right pointer to the left } // If no pair found return false ; } int main (){ int arr [] = { -3 , -1 , 0 , 1 , 2 }; int target = -2 ; int n = sizeof (arr) / sizeof (arr [0]); // Call the twoSum function and print the result if (twoSum (arr , n , target)) printf ("true \n "); else printf ("false \n "); return 0 ; } Java import java.util.Arrays ; class GfG { // Function to check whether any pair exists // whose sum is equal to the given target value static boolean twoSum (int [] arr , int target){ // Sort the array int left = 0 , right = arr . length - 1 ; // Iterate while left pointer is less than right while (left < right) { int sum = arr [left] + arr [right] ; // Check if the sum matches the target if (sum == target) return true ; else if (sum < target) left ++ ; // Move left pointer to the right else right -- ; // Move right pointer to the left } // If no pair is found return false ; } public static void main (String [] args){ int [] arr = { - 3 , - 1 , 0 , 1 , 2 }; int target = - 2 ; // Call the twoSum function and print the result if (twoSum (arr , target)) { System . out . println ("true"); } else { System . out . println ("false"); } } } Python # Function to check whether any pair exists # whose sum is equal to the given target value def two_sum (arr , target): # Sort the array left , right = 0 , len (arr) - 1 # Iterate while left pointer is less than right while left < right : sum = arr [left] + arr [right] # Check if the sum matches the target if sum == target : return True elif sum < target : left += 1 # Move left pointer to the right else : right -= 1 # Move right pointer to the left # If no pair is found return False arr = [- 3 , - 1 , 0 , 1 , 2] target = - 2 # Call the two_sum function and print the result if two_sum (arr , target): print ("true") else : print ("false") C# using System ; using System.Linq ; class GfG { // Function to check whether any pair exists // whose sum is equal to the given target value static bool TwoSum (int [] arr , int target){ // Sort the array int left = 0 , right = arr . Length - 1 ; // Iterate while left pointer is less than right while (left < right) { int sum = arr [left] + arr [right]; // Check if the sum matches the target if (sum == target) return true ; else if (sum < target) left ++ ; // Move left pointer to the right else right -- ; // Move right pointer to the left } // If no pair is found return false ; } static void Main (){ int [] arr = { - 3 , - 1 , 0 , 1 , 2 }; int target = - 2 ; // Call the TwoSum function and print the result if (TwoSum (arr , target)) Console . WriteLine ("true"); else Console . WriteLine ("false"); } } JavaScript // Function to check whether any pair exists // whose sum is equal to the given target value function twoSum (arr , target) { // Sort the array let left = 0 , right = arr . length - 1 ; // Iterate while left pointer is less than right while (left < right) { let sum = arr [left] + arr [right]; // Check if the sum matches the target if (sum === target) return true ; else if (sum < target) left ++ ; // Move left pointer to the right else right -- ; // Move right pointer to the left } // If no pair is found return false ; } let arr = [- 3 , - 1 , 0 , 1 , 2]; let target = - 2 ; // Call the twoSum function and print the result if (twoSum (arr , target)) { console . log ("true"); } else { console . log ("false"); } Output true Time Complexity: O(n) as the loops runs at most n times. We either increase left, or decrease right or stop the loop. Auxiliary Space: O(1) How does this work? We need to prove that we never miss a valid pair. Case 1 (When we increment left) In this case we simply ignore current arr[left] and move to the next element by doing left++. We do this when arr[left] + arr[right] is smaller than the target. The reason this step is safe is, if arr[left] is giving a smaller value than sum, then it will given even much less values for the elements before arr[right]. Now how about the elements after arr[right]? Note that we moved right when we were sure that no pair can be formed with the current right (arr[right] was too high), so arr[left] can not form a pair with those values also. Case 2 (When we decrement right) We can use the same reasoning (as we discussed for left) to prove that we never miss out a valid pair. More problems based on two pointer technique. Top Problems for Two Pointers Coding Practice on Two Pointer Algorithms Comment Article Tags: Article Tags: Misc Searching Technical Scripter DSA Arrays two-pointer-algorithm + 2 More