4 Sum - All Distinct Quadruplets with given Sum in an Array - GeeksforGeeks

Source: https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/

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 4 Sum - All Distinct Quadruplets with given Sum in an Array Last Updated : 13 Aug, 2025 Given an array arr[] , and an integer target, find all possible unique quadruplets in an array whose sum is equal to the given target value. We can return quadruplets in any order, but all the quadruplets should be internally sorted , i.e., for any quadruplets [q1, q2, q3, q4] the following should follow: q1 <= q2 <= q3 <= q4. Examples: Input: arr[] = [10, 11, 10, 12, 11], target = 43 Output: [[10, 10, 11, 12]] Explanation: The quadruplets are: [10, 11, 10, 12], sum = 10 + 11 + 10 +12 = 43 [10, 11, 10, 11], sum = 10 + 11 + 10 + 11 = 42 [10, 11, 12, 11], sum = 10 + 11 + 12 + 11 = 44 [10, 10, 12, 11], sum = 10 + 10 + 12 + 11 = 43 [11, 10, 12, 11], sum = 11 + 10 + 12 + 11 = 44 When arranged in sorted order, there is only one distinct quadruplet with sum = 43, that is [10, 10, 11, 12] Input: arr[] = [10, 2, 3, 4, 5, 7, 8], target = 23 Output: [[2, 3, 8, 10], [2, 4, 7, 10], [3, 5, 7, 8]] Explanation: There are only three distinct quadruplets with sum = 23. Input: arr[] = [1, 1, 1, 1, 1, 1], target = 4 Output: [[1, 1, 1, 1]] Try it on GfG Practice We have discussed how to find if a quadruple with given sum exists or not in an array . We are going to extend the ideas here to find all distinct Quadruplets. Table of Content [Naive Approach] Generating all quadruplets - O(n^5) Time and O(1) Space [Better Approach] Using Hashing - O(n^3) Time and O(n) Space [Expected Approach] Sorting and Two Pointer - O(n^3) Time and O(1) Space [Naive Approach] Generating all quadruplets - O(n^5) Time and O(1) Space We run 4 nested loops to generate all quadruplets. For every quadruple, we check if its sum is equal to the given target. If yes, then we first sort it to match the question requirements, then we check if this is a duplicate or not. If it is a new quadruple, we add it to the result. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; vector < vector < int >> fourSum (vector < int >& arr , int target) { vector < vector < int >> res ; int n = arr . size (); // Generating all possible quadruplets for (int i = 0 ; i < n - 3 ; i ++) { for (int j = i + 1 ; j < n - 2 ; j ++) { for (int k = j + 1 ; k < n - 1 ; k ++) { for (int l = k + 1 ; l < n ; l ++) { if (arr [i] + arr [j] + arr [k] + arr [l] == target) { vector < int > curr = { arr [i], arr [j], arr [k], arr [l]}; // Sort as needed in the output sort (curr . begin (), curr . end ()); // Making sure that all quadruplets with // target sum are distinct if (find (res . begin (), res . end (), curr) == res . end ()) { res . push_back (curr); } } } } } } return res ; } int main () { vector < int > arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; vector < vector < int >> ans = fourSum (arr , target); for (const auto & v : ans) { for (int x : v) cout << x << " " ; cout << endl ; } return 0 ; } C #include <stdio.h> // A helper function to compare // two integers (used in qsort) int compare (const void * a , const void * b) { return (* (int *) a - * (int *) b); } // Function to check if the quadruplet // already exists in the result int isDuplicate (int res [][4], int resSize , int curr []) { for (int i = 0 ; i < resSize ; i ++) { if (res [i][0] == curr [0] && res [i][1] == curr [1] && res [i][2] == curr [2] && res [i][3] == curr [3]) { return 1 ; } } return 0 ; } void fourSum (int arr [], int n , int target) { // Array to store unique quadruplets int res [100][4]; int resSize = 0 ; // Generating all possible quadruplets for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { for (int k = j + 1 ; k < n ; k ++) { for (int l = k + 1 ; l < n ; l ++) { if (arr [i] + arr [j] + arr [k] + arr [l] == target) { int curr [] = { arr [i], arr [j], arr [k], arr [l]}; qsort (curr , 4 , sizeof (int), compare); // Making sure that all quadruplets with // target sum are distinct if (! isDuplicate (res , resSize , curr)) { for (int m = 0 ; m < 4 ; m ++) { res [resSize][m] = curr [m]; } resSize ++ ; // Print the unique quadruplet printf ("%d %d %d %d \n " , curr [0], curr [1], curr [2], curr [3]); } } } } } } } int main () { int arr [] = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; int n = sizeof (arr) /

sizeof (arr [0]); fourSum (arr , n , target); return 0 ; } Java import java.util.ArrayList ; import java.util.Arrays ; import java.util.Collections ; class GfG { static ArrayList < ArrayList < Integer >> fourSum (int [] arr , int target) { ArrayList < ArrayList < Integer >> res = new ArrayList <> (); int n = arr . length ; // Generating all possible quadruplets for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { for (int k = j + 1 ; k < n ; k ++) { for (int l = k + 1 ; l < n ; l ++) { if (arr [i] + arr [j] + arr [k] + arr [l] == target) { ArrayList < Integer > curr = new ArrayList <> (Arrays . asList (arr [i] , arr [j] , arr [k] , arr [l])); // Sort to avoid duplicates in different order Collections . sort (curr); // Check for uniqueness if (! res . contains (curr)) { res . add (curr); } } } } } } return res ; } public static void main (String [] args) { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; ArrayList < ArrayList < Integer >> ans = fourSum (arr , target); for (ArrayList < Integer > v : ans) { for (int x : v) { System . out . print (x + " "); } System . out . println (); } } } Python def fourSum (arr , target): res = [] n = len (arr) # Generating all possible quadruplets for i in range (n): for j in range (i + 1 , n): for k in range (j + 1 , n): for l in range (k + 1 , n): if arr [i] + arr [j] + arr [k] + arr [l] == target : curr = [arr [i], arr [j], arr [k], arr [l]] # Sort as needed in the output curr . sort () # Making sure that all quadruplets with target # sum are distinct if curr not in res : res . append (curr) return res if __name__ == "__main__" : arr = [10 , 2 , 3 , 4 , 5 , 7 , 8] target = 23 ans = fourSum (arr , target) for v in ans : print (" " . join (map (str , v))) C# using System ; using System.Collections.Generic ; class GfG { static List < List < int >> fourSum (int [] arr , int target) { List < List < int >> res = new List < List < int >> (); int n = arr . Length ; // Generating all possible quadruplets for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { for (int k = j + 1 ; k < n ; k ++) { for (int l = k + 1 ; l < n ; l ++) { if (arr [i] + arr [j] + arr [k] + arr [l] == target) { List < int > curr = new List < int > { arr [i], arr [j], arr [k], arr [l] }; // Sort the current quadruplet curr . Sort (); // Check for uniqueness bool isUnique = true ; foreach (var quad in res) { if (quad [0] == curr [0] && quad [1] == curr [1] && quad [2] == curr [2] && quad [3] == curr [3]) { isUnique = false ; break ; } } // Add to the result if unique if (isUnique) { res . Add (curr); } } } } } return res ; } static void Main () { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; List < List < int >> ans = fourSum (arr , target); foreach (List < int > v in ans) { Console . WriteLine (string . Join (" " , v)); } } } JavaScript function fourSum (arr , target) { let res = []; let n = arr . length ; // Generating all possible quadruplets for (let i = 0 ; i < n ; i ++) { for (let j = i + 1 ; j < n ; j ++) { for (let k = j + 1 ; k < n ; k ++) { for (let l = k + 1 ; l < n ; l ++) { if (arr [i] + arr [j] + arr [k] + arr [l] === target) { let curr = [arr [i], arr [j], arr [k], arr [l]]; // Sort as needed in the output curr . sort ((a , b) => a - b); // Making sure that all quadruplets with // target sum are distinct if (! res . some (x => x . join () === curr . join ())) { res . push (curr); } } } } } } return res ; } // Driver code let arr = [10 , 2 , 3 , 4 , 5 , 7 , 8]; let target = 23 ; let ans = fourSum (arr , target); ans . forEach (v => console . log (v . join (" "))); Output 2 3 8 10 2 4 7 10 3 5 7 8 [Better Approach] Using Hashing - O(n^3) Time and O(n) Space We mainly generate all pairs and for every pair, we use hashing to find the remaining two pairs. We mainly use hashing at two places. For finding the remaining two elements of the quadruplets. Making sure that all quadruplets are distinct. C++ #include <iostream> #include <vector> #include <set> #include <unordered_set> #include <algorithm> using namespace std ; vector < vector < int >> fourSum (vector < int >& arr , int target) { int n = arr . size (); // Set to store unique sorted quadruplets // to avoid duplicates set < vector < int >> resSet ; // Fix the first two elements // using two nested loops for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // Use an unordered_set to check if the // required fourth number exists unordered_set < int > s ; // Loop for the third element for (int k = j + 1 ; k < n ; k ++) { int sum = arr [i] + arr [j] + arr [k]; int last = target - sum ; // If the fourth number is found in the // set, we have a valid quadruplet if (s . find (last) != s . end ()) { vector < int > curr = { arr [i], arr [j], arr [k], last }; // Sort the quadruplet to ensure uniqueness // when storing in the set sort (curr . begin (), curr . end ()); resSet . insert (curr); } s . insert (arr [k]); } } } return vector < vector < int >> (resSet . begin (), resSet . end ()); } int main () { vector < int > arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; vector < vector < int >> ans = fourSum (arr , target); for (const auto & v : ans) { for (int x : v) { cout << x << " " ; } cout << endl ; } return 0 ; } Java import java.util.ArrayList ; import java.util.HashSet ; import java.util.Set ; import java.util.Collections ; import java.util.Arrays ; class GfG { static ArrayList < ArrayList < Integer >> fourSum (int [] arr , int target) { int n = arr . length ; // Use a set to avoid duplicate // quadruplets Set < ArrayList < Integer >> resSet = new HashSet <> (); // Generate all triplets and check // for the fourth element for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // Use a hash set to look up the // needed fourth element Set < Integer > s = new HashSet <> (); for (int k = j + 1 ; k < n ; k ++) { int sum = arr [i] + arr [j] + arr [k] ; int last = target - sum ; if (s . contains (last)) { ArrayList < Integer > curr = new ArrayList <> (Arrays . asList (arr [i] , arr [j] , arr [k] , last)); // Sort to ensure uniqueness Collections . sort (curr); // Set avoids duplicates resSet . add (curr); } s . add (arr [k]); } } } return new ArrayList <> (resSet); } public static void main (String [] args

) { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; ArrayList < ArrayList < Integer >> ans = fourSum (arr , target); for (ArrayList < Integer > v : ans) { for (int x : v) { System . out . print (x + " "); } System . out . println (); } } } Python def fourSum (arr , target): # Initialize a set to store unique # quadruplets as sorted tuples res_set = set () n = len (arr) # Iterate to fix the first two elements for i in range (n): for j in range (i + 1 , n): # Set to track elements seen # so far for the third loop s = set () # Loop to fix the third element and find the fourth for k in range (j + 1 , n): sum_val = arr [i] + arr [j] + arr [k] last = target - sum_val # If the fourth required element is already seen if last in s : curr = sorted ([arr [i], arr [j], arr [k], last]) res_set . add (tuple (curr)) # Add current number to the set for future lookup s . add (arr [k]) return [list (t) for t in res_set] if __name__ == "__main__" : arr = [10 , 2 , 3 , 4 , 5 , 7 , 8] target = 23 ans = fourSum (arr , target) for v in ans : print (" " . join (map (str , v))) C# using System ; using System.Collections.Generic ; using System.Linq ; class GfG { static List < List < int >> fourSum (int [] arr , int target) { // Initialize a set to store unique sorted // quadruplets using custom comparer var resSet = new HashSet < List < int >> (new ListComparer ()); int n = arr . Length ; // Iterate to fix the first two // elements of the quadruplet for (int i = 0 ; i < n ; i ++) { for (int j = i + 1 ; j < n ; j ++) { // Use a set to track seen elements // for the third loop var s = new HashSet < int > (); // Loop to fix the third element // and check for the fourth for (int k = j + 1 ; k < n ; k ++) { int sum = arr [i] + arr [j] + arr [k]; int last = target - sum ; // If the fourth required element // has already been seen if (s . Contains (last)) { var curr = new List < int > { arr [i], arr [j], arr [k], last }; // Sort to ensure a consistent // order for uniqueness curr . Sort (); // Add the sorted quadruplet to the set resSet . Add (curr); } // Add current element to the hash // set for future checks s . Add (arr [k]); } } } // Convert the set to a list and return return new List < List < int >> (resSet); } // Custom comparer class to check list // equality and generate hash codes class ListComparer : IEqualityComparer < List < int >> { public bool Equals (List < int > x , List < int > y) { if (x . Count != y . Count) return false ; return x . SequenceEqual (y); } public int GetHashCode (List < int > obj) { int hash = 17 ; // Compute hash code based on list content foreach (int item in obj) { hash = hash * 31 + item . GetHashCode (); } return hash ; } } public static void Main () { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; var ans = fourSum (arr , target); foreach (var v in ans) { Console . WriteLine (string . Join (" " , v)); } } } JavaScript function fourSum (arr , target) { // Set to store unique quadruplets as // comma-separated strings let resSet = new Set (); let n = arr . length ; // Fix the first two elements // using two nested loops for (let i = 0 ; i < n ; i ++) { for (let j = i + 1 ; j < n ; j ++) { // Set to track elements seen so // far for the third loop let s = new Set (); // Loop to fix the third element // and find the fourth for (let k = j + 1 ; k < n ; k ++) { let sum = arr [i] + arr [j] + arr [k]; let last = target - sum ; // If the required fourth element // is already in the set if (s . has (last)) { let curr = [arr [i], arr [j], arr [k], last]. sort ((a , b) => a - b); // Convert array to string for storage // in Set to ensure uniqueness resSet . add (curr . toString ()); } // Add current element to the set for future lookups s . add (arr [k]); } } } return Array . from (resSet). map (e => e . split (","). map (Number)); } // Driver code const arr = [10 , 2 , 3 , 4 , 5 , 7 , 8]; const target = 23 ; const ans = fourSum (arr , target); ans . forEach (v => console . log (v . join (" "))); Output 2 3 8 10 2 4 7 10 3 5 7 8 [Expected Approach] Sorting and Two Pointer - O(n^3) Time and O(1) Space Sort the array Generate all pairs. For every pair, find the remaining two elements using two pointer technique . How do we ensure that we get only distinct ? While generating pairs, we skip duplicates in both outer and inner loops by comparing with the previous element (note that the array is sorted first). In two Pointer technique, when we find a match, we skip all occurrences of that element in the array. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; vector < vector < int >> fourSum (vector < int >& arr , int target) { vector < vector < int >> res ; int n = arr . size (); // Sort the array sort (arr . begin (), arr . end ()); // Generate quadruplets for (int i = 0 ; i < n ; i ++) { // Skip duplicates for i if (i > 0 && arr [i] == arr [i - 1]) continue ; for (int j = i + 1 ; j < n ; j ++) { // Skip duplicates for j if (j > i + 1 && arr [j] == arr [j - 1]) continue ; int k = j + 1 , l = n - 1 ; // Two pointers approach while (k < l) { int sum = arr [i] + arr [j] + arr [k] + arr [l]; if (sum == target) { res . push_back ({ arr [i], arr [j], arr [k], arr [l]}); k ++ ; l -- ; // Skip duplicates for k and l while (k < l && arr [k] == arr [k - 1]) k ++ ; while (k < l && arr [l] == arr [l + 1]) l -- ; } else if (sum < target) { k ++ ; } else { l -- ; } } } } return res ; } int main () { vector < int > arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; vector < vector < int >> ans = fourSum (arr , target); for (const auto & v : ans) { for (int x : v) { cout << x << " " ; } cout << endl ; } return 0 ; } C #include <stdio.h> #include <stdlib.h> // A utility function to compare // two integers (used in qsort) int compare (const void * a , const void * b) { return (* (int *) a - * (int *) b); } // Function to find quadruplets // that sum to the target void fourSum (int arr [], int n , int target) { // Sort the array qsort (arr , n , sizeof (int), compare); // Generate quadruplets for (int i = 0 ; i < n ; i ++) { // Skip duplicates for i if (i > 0 && arr [i] == arr [i - 1]) continue ; for (int j = i + 1 ; j < n ; j ++) { // Skip duplicates for j if (j > i + 1 && arr [j] == arr [j - 1]) continue ; int k = j + 1 ; int l = n - 1 ; // Two pointers approach while (k < l) { int sum = arr [i] + arr [j] + arr [k] + arr [l]; if (sum == target) { printf ("%d %d %d %d \n " , arr [i], arr [j], arr [k], arr [l]); k ++ ; l -- ; // Skip duplicates for k and l while (k < l && arr [k] == arr [k - 1]) k ++ ; while (k < l && arr [l] == arr [l + 1]) l -- ; } else if (sum < target) { k ++ ; } else { l -- ; } } } } } int main () { int arr [] = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; int n = sizeof (arr) / sizeof (arr [0]); fourSum (arr , n , target); return 0 ; } Java import java.util.ArrayList ; import java.util.Arrays ; class GfG { static ArrayList < ArrayList < Integer >> fourSum (int [] arr , int target) { ArrayList < ArrayList < Integer >> res = new ArrayList <> (); int n = arr . length ; // Sort the array to apply two-pointer approach Arrays . sort (arr); // Fix the first two elements for (int i = 0 ; i < n ; i ++) { // Skip duplicates for i if (i > 0 && arr [i] == arr [i - 1]) continue ; for (int j = i + 1 ; j < n ; j ++) { // Skip duplicates for j if (j > i + 1 && arr [j] == arr [j - 1]) continue ; int k = j + 1 ; int l = n - 1 ; // Use two-pointer technique for remaining two elements while (k < l) { int sum = arr [i] + arr [j] + arr [k] + arr [l] ; if (sum == target) { ArrayList < Integer > quad = new ArrayList <> (); quad . add (arr [i]); quad . add (arr [j]); quad . add (arr [k]); quad . add (arr [l]); res . add (quad); k ++ ; l -- ; // Skip duplicates for k while (k < l && arr [k] == arr [k - 1]) k ++ ; // Skip duplicates for l while (k < l && arr [l] == arr [l + 1]) l -- ; } else if (sum < target) { k ++ ; } else { l -- ; } } } } return res ; } public static void main (String [] args) { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; ArrayList < ArrayList < Integer >> ans = fourSum (arr , target); for (ArrayList < Integer > v : ans) { for (int x : v) { System . out . print (x + " "); } System . out . println (); } } } Python def fourSum (arr , target): res = [] n = len (arr) # Sort the array arr . sort () # Generate quadruplets for i in range (n): # Skip duplicates for i if i > 0 and arr [i] == arr [i - 1]: continue for j in range (i + 1 , n): # Skip duplicates for j if j > i + 1 and arr [j] == arr [j - 1]: continue k , l = j + 1 , n - 1 # Two pointers approach while k < l : total = arr [i] + arr [j] + arr [k] + arr [l] if total == target : res . append ([arr [i], arr [j], arr [k], arr [l]]) k += 1 l -= 1 # Skip duplicates for k and l while k < l and arr [k] == arr [k - 1]: k += 1 while k < l and arr [l] == arr [l + 1]: l -= 1 elif total < target : k += 1 else : l -= 1 return res if __name__ == "__main__" : arr = [10 , 2 , 3 , 4 , 5 , 7 , 8] target = 23 ans = fourSum (arr , target) for v in ans : print (" " . join (map (str , v))) C# using System ; using System.Collections.Generic ; class GfG { // Function to find quadruplets // that sum to the target static List < List < int >> fourSum (int [] arr , int target) { List < List < int >> res = new List < List < int >> (); int n = arr . Length ; // Sort the array Array . Sort (arr); // Generate quadruplets for (int i = 0 ; i < n ; i ++) { // Skip duplicates for i if (i > 0 && arr [i] == arr [i - 1]) continue ; for (int j = i + 1 ; j < n ; j ++) { // Skip duplicates for j if (j > i + 1 && arr [j] == arr [j - 1]) continue ; int k = j + 1 , l = n - 1 ; // Two pointers approach while (k < l) { int sum = arr [i] + arr [j] + arr [k] + arr [l]; if (sum == target) { res . Add (new List < int > { arr [i], arr [j], arr [k], arr [l] }); k ++ ; l -- ; // Skip duplicates for k and l while (k < l && arr [k] == arr [k - 1]) k ++ ; while (k < l && arr [l] == arr [l + 1]) l -- ; } else if (sum < target) { k ++ ; } else { l -- ; } } } } return res ; } public static void Main () { int [] arr = { 10 , 2 , 3 , 4 , 5 , 7 , 8 }; int target = 23 ; List < List < int >> ans = fourSum (arr , target); foreach (var v in ans) { Console . WriteLine (string . Join (" " , v)); } } } JavaScript // JavaScript Program to find all Distinct Quadruplets with // given Sum in an Array using Two Pointer Technique // Function to find quadruplets that sum to the target function fourSum (arr , target) { let res = []; let n = arr . length ; // Sort the array arr . sort ((a , b) => a - b); // Generate quadruplets for (let i = 0 ; i < n ; i ++) { // Skip duplicates for i if (i > 0 && arr [i] === arr [i - 1]) continue ; for (let j = i + 1 ; j < n ; j ++) { // Skip duplicates for j if (j > i + 1 && arr [j] === arr [j - 1]) continue ; let k = j + 1 , l = n - 1 ; // Two pointers approach while (k < l) { let sum = arr [i] + arr [j] + arr [k] + arr [l]; if (sum === target) { res . push ([arr [i], arr [j], arr [k], arr [l]]); k ++ ; l -- ; // Skip duplicates for k and l while (k < l && arr [k] === arr [k - 1]) k ++ ; while (k < l && arr [l] === arr [l + 1]) l -- ; } else if (sum < target) { k ++ ; } else { l -- ; } } } } return res ; } // Driver code const arr = [10 , 2 , 3 , 4 , 5 , 7 , 8]; const target = 23 ; const ans = fourSum (arr , target); ans . forEach (v => console . log (v . join (" "))); Output 2 3 8 10 2 4 7 10 3 5 7 8 Further Optimizations: The outermost and second outermost loops can be optimized to run till i < n-3 and j < n-2 We can add some checks for early skipping the loop like if sum of first 4 elements is more than target inside the second loop or third loop, then we exit. Because there is no possibility of finding a target value. Comment Article Tags: Article Tags: Sorting Hash DSA Arrays Microsoft Amazon Google two-pointer-algorithm 2Sum 3Sum + 6 More