Partition a Set into Two Subsets of Equal Sum - GeeksforGeeks

Source: https://www.geeksforgeeks.org/partition-problem-dp-18/

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 Partition a Set into Two Subsets of Equal Sum Last Updated : 27 Jan, 2026 Given an array arr[], check if it can be partitioned into two parts such that the sum of elements in both parts is the same. Note: Each element is present in either the first subset or the second subset, but not in both. Examples: Input: arr[] = [1, 5, 11, 5] Output: true Explanation: The array can be partitioned as [1, 5, 5] and [11] and the sum of both the subsets are equal. Input: arr[] = [1, 5, 3] Output: false Explanation: The array cannot be partitioned into equal sum sets. Try it on GfG Practice Table of Content [Naive Approach] Using Recursion - O(2^n) Time and O(n) Space [Better Approach 1] Using Top-Down DP (Memoization) - O(sum*n) Time and O(sum*n) Space [Better Approach 2] Using Bottom-Up DP (Tabulation) - O(sum*n) Time and O(sum*n) Space [Expected Approach] Using Space Optimized DP - O(sum*n) Time and O(sum) Space [Naive Approach] Using Recursion - O(2^n) Time and O(n) Space The idea of this approach is to try all possible ways of dividing the array into two subsets using recursion. For each element, we have two choices: either include it in the current subset or exclude it. The goal is to check whether there exists a subset whose sum is equal to half of the total array sum. If such a subset exists, the remaining elements will automatically form the second subset with equal sum. C++ #include <iostream> using namespace std ; bool isSubsetSum (int n , vector < int >& arr , int sum) { // base cases if (sum == 0) return true ; if (n == 0) return false ; // If element is greater than sum, then ignore it if (arr [n -1] > sum) return isSubsetSum (n -1 , arr , sum); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element return isSubsetSum (n -1 , arr , sum) || isSubsetSum (n -1 , arr , sum - arr [n -1]); } bool equalPartition (vector < int >& arr) { // Calculate sum of the elements in array int sum = accumulate (arr . begin (), arr . end (), 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; return isSubsetSum (arr . size (), arr , sum / 2); } int main () { vector < int > arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { cout << "True" << endl ; } else { cout << "False" << endl ; } return 0 ; } Java class GfG { static boolean isSubsetSum (int n , int [] arr , int sum) { // base cases if (sum == 0) return true ; if (n == 0) return false ; // If element is greater than sum, then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element return isSubsetSum (n - 1 , arr , sum) || isSubsetSum (n - 1 , arr , sum - arr [n - 1]); } static boolean equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; for (int i = 0 ; i < arr . length ; i ++) { sum += arr [i] ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; return isSubsetSum (arr . length , arr , sum / 2); } public static void main (String [] args) { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python def isSubsetSum (n , arr , sum): # base cases if sum == 0 : return True if n == 0 : return False # If element is greater than sum, then ignore it if arr [n - 1] > sum : return isSubsetSum (n - 1 , arr , sum) # Check if sum can be obtained by any of the following # either including the current element # or excluding the current element return isSubsetSum (n - 1 , arr , sum) or \ isSubsetSum (n - 1 , arr , sum - arr [n - 1]) def equalPartition (arr): # Calculate sum of the elements in array arrSum = sum (arr) # If sum is odd, there cannot be two # subsets with equal sum if arrSum % 2 != 0 : return False return isSubsetSum (len (arr), arr , arrSum // 2) if __name__ == "__main__" : arr = [1 , 5 , 11 , 5] if

equalPartition (arr): print ("True") else : print ("False") C# using System ; class GfG { static bool isSubsetSum (int n , int [] arr , int sum) { // base cases if (sum == 0) return true ; if (n == 0) return false ; // If element is greater than sum, // then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element return isSubsetSum (n - 1 , arr , sum) || isSubsetSum (n - 1 , arr , sum - arr [n - 1]); } static bool equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; foreach (int num in arr) { sum += num ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; return isSubsetSum (arr . Length , arr , sum / 2); } static void Main () { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { Console . WriteLine ("True"); } else { Console . WriteLine ("False"); } } } JavaScript function isSubsetSum (n , arr , sum) { // base cases if (sum == 0) return true ; if (n == 0) return false ; // If element is greater than sum, then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element return isSubsetSum (n - 1 , arr , sum) || isSubsetSum (n - 1 , arr , sum - arr [n - 1]); } function equalPartition (arr) { // Calculate sum of the elements in array let sum = arr . reduce ((a , b) => a + b , 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 !== 0) return false ; return isSubsetSum (arr . length , arr , sum / 2); } //Driver Code const arr = [1 , 5 , 11 , 5]; if (equalPartition (arr)) { console . log ("True"); } else { console . log ("False"); } Output True [Better Approach 1] Using Top-Down DP (Memoization) - O(sum*n) Time and O(sum*n) Space This approach optimizes the recursive solution by storing the results of overlapping subproblems using memoization. Instead of recomputing the same states repeatedly, a 2D memo table is used to remember whether a given sum can be achieved using the first n elements. This significantly reduces the exponential time complexity of the naive approach. C++ #include <iostream> using namespace std ; bool isSubsetSum (int n , vector < int >& arr , int sum , vector < vector < int >> & memo) { // base cases if (sum == 0) return true ; if (n == 0) return false ; if (memo [n -1][sum] != -1) return memo [n -1][sum]; // If element is greater than sum, then ignore it if (arr [n -1] > sum) return isSubsetSum (n -1 , arr , sum , memo); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element return memo [n -1][sum] = isSubsetSum (n -1 , arr , sum , memo) || isSubsetSum (n -1 , arr , sum - arr [n -1], memo); } bool equalPartition (vector < int >& arr) { // Calculate sum of the elements in array int sum = accumulate (arr . begin (), arr . end (), 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; vector < vector < int >> memo (arr . size (), vector < int > (sum + 1 , -1)); return isSubsetSum (arr . size (), arr , sum / 2 , memo); } int main () { vector < int > arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { cout << "True" << endl ; } else { cout << "False" << endl ; } return 0 ; } Java import java.util.Arrays ; class GfG { static boolean isSubsetSum (int n , int [] arr , int sum , int [][] memo) { // base cases if (sum == 0) return true ; if (n == 0) return false ; if (memo [n - 1][sum] != - 1) return memo [n - 1][sum] == 1 ; // If element is greater than sum, then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum , memo); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element memo [n - 1][sum] = (isSubsetSum (n - 1 , arr , sum , memo) || isSubsetSum (n - 1 , arr , sum - arr [n - 1] , memo)) ? 1 : 0 ; return memo [n - 1][sum] == 1 ; } static boolean equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; for (int num : arr) { sum += num ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; int [][] memo = new int [arr . length][sum + 1] ; for (int [] row : memo) { Arrays . fill (row , - 1); } return isSubsetSum (arr . length , arr , sum / 2 , memo); } public static void main (String [] args) { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python def isSubsetSum (n , arr , sum , memo): # base cases if sum == 0 : return True if n == 0 : return False if memo [n - 1][sum] != - 1 : return memo [n - 1][sum] # If element is greater than sum, then ignore it if arr [n - 1] > sum : return isSubsetSum (n - 1 , arr , sum , memo) # Check if sum can be obtained by any of the following # either including the current element # or excluding the current element memo [n - 1][sum] = isSubsetSum (n - 1 , arr , sum , memo) or \ isSubsetSum (n - 1 , arr , sum - arr [n - 1], memo) return memo [n - 1][sum] def equalPartition (arr): # Calculate sum of the elements in array arrSum = sum (arr) # If sum is odd, there cannot be two # subsets with equal sum if arrSum % 2 != 0 : return False memo = [[- 1 for _ in range (arrSum + 1)] for _ in range (len (arr))] return isSubsetSum (len (arr), arr , arrSum // 2 , memo) if __name__ == "__main__" : arr = [1 , 5 , 11 , 5] if equalPartition (arr): print ("True") else : print ("False") C# using System ; class GfG { static bool isSubsetSum (int n , int [] arr , int sum , int [,] memo) { // base cases if (sum == 0) return true ; if (n == 0) return false ; if (memo [n - 1 , sum] != - 1) return memo [n - 1 , sum] == 1 ; // If element is

greater than sum, then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum , memo); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element memo [n - 1 , sum] = (isSubsetSum (n - 1 , arr , sum , memo) || isSubsetSum (n - 1 , arr , sum - arr [n - 1], memo)) ? 1 : 0 ; return memo [n - 1 , sum] == 1 ; } static bool equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; foreach (int num in arr) { sum += num ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; int [,] memo = new int [arr . Length , sum + 1]; for (int i = 0 ; i < arr . Length ; i ++) { for (int j = 0 ; j <= sum ; j ++) { memo [i , j] = - 1 ; } } return isSubsetSum (arr . Length , arr , sum / 2 , memo); } static void Main () { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { Console . WriteLine ("True"); } else { Console . WriteLine ("False"); } } } JavaScript function isSubsetSum (n , arr , sum , memo) { // base cases if (sum == 0) return true ; if (n == 0) return false ; if (memo [n - 1][sum] !== - 1) return memo [n - 1][sum]; // If element is greater than sum, then ignore it if (arr [n - 1] > sum) return isSubsetSum (n - 1 , arr , sum , memo); // Check if sum can be obtained by any of the following // either including the current element // or excluding the current element memo [n - 1][sum] = isSubsetSum (n - 1 , arr , sum , memo) || isSubsetSum (n - 1 , arr , sum - arr [n - 1], memo); return memo [n - 1][sum]; } function equalPartition (arr) { // Calculate sum of the elements in array let sum = arr . reduce ((a , b) => a + b , 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 !== 0) return false ; let memo = Array . from (Array (arr . length), () => Array (sum + 1). fill (- 1)); return isSubsetSum (arr . length , arr , sum / 2 , memo); } //Driver Code const arr = [1 , 5 , 11 , 5]; if (equalPartition (arr)) { console . log ("True"); } else { console . log ("False"); } Output True [Better Approach 2] Using Bottom-Up DP (Tabulation) - O(sum*n) Time and O(sum*n) Space In this approach, a dynamic programming table is built iteratively in a bottom-up manner. The table dp[i][j] represents whether a subset with sum j can be formed using the first i elements. By filling the table row by row, we determine whether the target sum (half of total sum) is achievable using all elements. C++ #include <iostream> using namespace std ; bool equalPartition (vector < int >& arr) { // Calculate sum of the elements in array int sum = accumulate (arr . begin (), arr . end (), 0); int n = arr . size (); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; // Create a 2D vector for storing results // of subproblems vector < vector < bool >> dp (n + 1 , vector < bool > (sum + 1 , false)); // If sum is 0, then answer is true (empty subset) for (int i = 0 ; i <= n ; i ++) dp [i][0] = true ; // Fill the dp table in bottom-up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= sum ; j ++) { if (j < arr [i - 1]) { // Exclude the current element dp [i][j] = dp [i - 1][j]; } else { // Include or exclude dp [i][j] = dp [i - 1][j] || dp [i - 1][j - arr [i - 1]]; } } } return dp [n][sum]; } int main () { vector < int > arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { cout << "True" << endl ; } else { cout << "False" << endl ; } return 0 ; } Java class GfG { static boolean equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; for (int num : arr) { sum += num ; } int n = arr . length ; // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; // Create a 2D array for storing results // of subproblems boolean [][] dp = new boolean [n + 1][sum + 1] ; // If sum is 0, then answer is true (empty subset) for (int i = 0 ; i <= n ; i ++) { dp [i][0] = true ; } // Fill the dp table in bottom-up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= sum ; j ++) { if (j < arr [i - 1]) { // Exclude the current element dp [i][j] = dp [i - 1][j] ; } else { // Include or exclude dp [i][j] = dp [i - 1][j] || dp [i - 1][j - arr [i - 1]] ; } } } return dp [n][sum] ; } public static void main (String [] args) { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python def equalPartition (arr): # Calculate sum of the elements in array arrSum = sum (arr) n = len (arr) # If sum is odd, there cannot be two # subsets with equal sum if arrSum % 2 != 0 : return False arrSum = arrSum // 2 # Create a 2D array for storing results # of subproblems dp = [[False] * (arrSum + 1) for _ in range (n + 1)] # If sum is 0, then answer is true (empty subset) for i in range (n + 1): dp [i][0] = True # Fill the dp table in bottom-up manner for i in range (1 , n + 1): for j in range (1 , arrSum + 1): if j < arr [i - 1]: # Exclude the current element dp [i][j] = dp [i - 1][j] else : # Include or exclude dp [i][j] = dp [i - 1][j] or dp [i - 1][j - arr [i - 1]] return dp [n][arrSum] if __name__ == "__main__" : arr = [1 , 5 , 11 , 5] if equalPartition (arr): print ("True") else : print ("False") C# using System ; class GfG { static bool equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; foreach (int num in arr) { sum += num ; } int n = arr . Length ; // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; // Create a 2D array for storing results // of subproblems bool [,] dp = new bool [n + 1 , sum + 1]; // If sum is 0, then answer is true (empty subset) for (int i = 0 ; i <= n ; i ++) { dp [i , 0] = true ; } // Fill the dp table in bottom-up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= sum ; j ++) { if (j < arr [i - 1]) { // Exclude the current element dp [i , j] = dp [i - 1 , j]; }

else { // Include or exclude dp [i , j] = dp [i - 1 , j] || dp [i - 1 , j - arr [i - 1]]; } } } return dp [n , sum]; } static void Main () { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { Console . WriteLine ("True"); } else { Console . WriteLine ("False"); } } } JavaScript function equalPartition (arr) { // Calculate sum of the elements in array let sum = arr . reduce ((a , b) => a + b , 0); let n = arr . length ; // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 !== 0) return false ; sum = sum / 2 ; // Create a 2D array for storing results // of subproblems let dp = Array . from ({ length : n + 1 }, () => Array (sum + 1). fill (false)); // If sum is 0, then answer is true (empty subset) for (let i = 0 ; i <= n ; i ++) { dp [i][0] = true ; } // Fill the dp table in bottom-up manner for (let i = 1 ; i <= n ; i ++) { for (let j = 1 ; j <= sum ; j ++) { if (j < arr [i - 1]) { // Exclude the current element dp [i][j] = dp [i - 1][j]; } else { // Include or exclude dp [i][j] = dp [i - 1][j] || dp [i - 1][j - arr [i - 1]]; } } } return dp [n][sum]; } //Driver Code const arr = [1 , 5 , 11 , 5]; if (equalPartition (arr)) { console . log ("True"); } else { console . log ("False"); } Output True [Expected Approach] Using Space Optimized DP - O(sum*n) Time and O(sum) Space This approach further optimizes the tabulation method by reducing space usage. Instead of maintaining a full 2D table, it uses two 1D arrays to represent the previous and current states. Since each row depends only on the previous row, this optimization preserves correctness while significantly reducing space complexity. C++ #include <iostream> using namespace std ; bool equalPartition (vector < int >& arr) { // Calculate sum of the elements in array int sum = accumulate (arr . begin (), arr . end (), 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; int n = arr . size (); vector < bool > prev (sum + 1 , false), curr (sum + 1); // Mark prev[0] = true as it is true // to make sum = 0 using 0 elements prev [0] = true ; // Fill the subset table in // bottom up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 0 ; j <= sum ; j ++) { if (j < arr [i - 1]) curr [j] = prev [j]; else curr [j] = (prev [j] || prev [j - arr [i - 1]]); } prev = curr ; } return prev [sum]; } int main () { vector < int > arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { cout << "True" << endl ; } else { cout << "False" << endl ; } return 0 ; } Java class GfG { static boolean equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; for (int num : arr) { sum += num ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; int n = arr . length ; boolean [] prev = new boolean [sum + 1] ; boolean [] curr = new boolean [sum + 1] ; // Mark prev[0] = true as it is true // to make sum = 0 using 0 elements prev [0] = true ; // Fill the subset table in // bottom-up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 0 ; j <= sum ; j ++) { if (j < arr [i - 1]) curr [j] = prev [j] ; else curr [j] = (prev [j] || prev [j - arr [i - 1]]); } // copy curr into prev for (int j = 0 ; j <= sum ; j ++) { prev [j] = curr [j] ; } } return prev [sum] ; } public static void main (String [] args) { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python def equalPartition (arr): # Calculate sum of the elements in array arrSum = sum (arr) # If sum is odd, there cannot be two # subsets with equal sum if arrSum % 2 != 0 : return False arrSum = arrSum // 2 n = len (arr) prev = [False] * (arrSum + 1) curr = [False] * (arrSum + 1) # Mark prev[0] = true as it is true # to make sum = 0 using 0 elements prev [0] = True # Fill the subset table in # bottom-up manner for i in range (1 , n + 1): for j in range (arrSum + 1): if j < arr [i - 1]: curr [j] = prev [j] else : curr [j] = (prev [j] or prev [j - arr [i - 1]]) prev = curr . copy () return prev [arrSum] if __name__ == "__main__" : arr = [1 , 5 , 11 , 5] if equalPartition (arr): print ("True") else : print ("False") C# using System ; class GfG { static bool equalPartition (int [] arr) { // Calculate sum of the elements in array int sum = 0 ; foreach (int num in arr) { sum += num ; } // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 != 0) return false ; sum = sum / 2 ; int n = arr . Length ; bool [] prev = new bool [sum + 1]; bool [] curr = new bool [sum + 1]; // Mark prev[0] = true as it is true // to make sum = 0 using 0 elements prev [0] = true ; // Fill the subset table in // bottom-up manner for (int i = 1 ; i <= n ; i ++) { for (int j = 0 ; j <= sum ; j ++) { if (j < arr [i - 1]) curr [j] = prev [j]; else curr [j] = (prev [j] || prev [j - arr [i - 1]]); } prev = (bool []) curr . Clone (); } return prev [sum]; } static void Main () { int [] arr = { 1 , 5 , 11 , 5 }; if (equalPartition (arr)) { Console . WriteLine ("True"); } else { Console . WriteLine ("False"); } } } JavaScript function equalPartition (arr) { // Calculate sum of the elements in array let sum = arr . reduce ((a , b) => a + b , 0); // If sum is odd, there cannot be two // subsets with equal sum if (sum % 2 !== 0) return false ; sum = sum / 2 ; let n = arr . length ; let prev = Array (sum + 1). fill (false); let curr = Array (sum + 1). fill (false); // Mark prev[0] = true as it is true // to make sum = 0 using 0 elements prev [0] = true ; // Fill the subset table in // bottom-up manner for (let i = 1 ; i <= n ; i ++) { for (let j = 0 ; j <= sum ; j ++) { if (j < arr [i - 1]) { curr [j] = prev [j]; } else { curr [j] = (prev [j] || prev [j - arr [i - 1]]); } } prev = [... curr]; } return prev [sum]; } // Driver code const arr = [1 , 5 , 11 , 5]; if (equalPartition (arr)) { console . log ("True"); } else { console . log ("False"); } Output True Related Articles: Subset Sum Problem Perfect Sum Problem (Print all subsets with given sum) Comment Article Tags: Article Tags: Dynamic

Programming DSA Amazon Adobe Drishti-Soft Accolite subset + 3 More