Subset Sum Problem - GeeksforGeeks

Source: https://www.geeksforgeeks.org/subset-sum-problem-dp-25/

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 Subset Sum Problem Last Updated : 23 Jul, 2025 Given an array arr[] of non-negative integers and a value sum , the task is to check if there is a subset of the given array whose sum is equal to the given sum . Examples: Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 9 Output: True Explanation: There is a subset (4, 5) with sum 9. Input: arr[] = [3, 34, 4, 12, 5, 2], sum = 30 Output: False Explanation: There is no subset that add up to 30. 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 For the recursive approach, there will be two cases (In both cases, the number of available elements decreases by 1) Consider the 'last' element to be a part of the subset. Now the new required sum = required sum - value of 'last' element. Don't include the 'last' element in the subset. Then the new required sum = old required sum . Mathematically the recurrence relation will look like the following: isSubsetSum(arr, n, sum) = isSubsetSum(arr, n-1, sum) OR isSubsetSum(arr, n-1, sum - arr[n-1]) Base Cases: isSubsetSum(arr, n, sum) = false, if sum > 0 and n = 0 isSubsetSum(arr, n, sum) = true, if sum = 0 Follow the below steps to implement the recursion: Build a recursive function and pass the index to be considered (here gradually moving from the last end) and the remaining sum amount. For each index check the base cases. If the answer is true for any recursion call, then there exists such a subset. Otherwise, no such subset exists. C++ //C++ implementation for subset sum // problem using recursion #include <bits/stdc++.h> using namespace std ; // Function to check if there is a subset // with the given sum using recursion bool isSubsetSumRec (vector < int >& arr , int n , int sum) { // Base Cases if (sum == 0) return true ; if (n == 0) return false ; // If last element is greater than sum, // then ignore it if (arr [n - 1] > sum) return isSubsetSumRec (arr , n - 1 , sum); // Check if sum can be obtained by including // or excluding the last element return isSubsetSumRec (arr , n - 1 , sum) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1]); } bool isSubsetSum (vector < int >& arr , int sum) { return isSubsetSumRec (arr , arr . size (), sum); } int main () { vector < int > arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) cout << "True" << endl ; else cout << "False" << endl ; return 0 ; } C //C implementation for subset sum // problem using recursion #include <stdio.h> // Function to check if there is a subset // with the given sum using recursion int isSubsetSumRec (int arr [], int n , int sum) { // Base Cases if (sum == 0) { return 1 ; } if (n == 0) { return 0 ; } // If last element is greater than sum, ignore it if (arr [n - 1] > sum) { return isSubsetSumRec (arr , n - 1 , sum); } // Check if sum can be obtained by including // or excluding the last element return isSubsetSumRec (arr , n - 1 , sum) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1]); } int isSubsetSum (int arr [], int n , int sum) { return isSubsetSumRec (arr , n , sum); } int main () { int arr [] = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; int n = sizeof (arr) / sizeof (arr [0]); if (isSubsetSum (arr , n , sum)) { printf ("True \n "); } else { printf ("False \n "); } return 0 ; } Java //Java implementation for subset sum // problem using recursion import java.util.* ; class GfG { // Function to check if there is a subset // with the given sum using recursion static boolean isSubsetSumRec (int [] arr , int n , int sum) { // Base Cases if (sum == 0) { return true ; } if (n == 0) { return false ; } // If last element is greater than // sum, ignore it if (arr [n - 1] > sum) { return isSubsetSumRec (arr , n - 1 , sum); } // Check if sum can be obtained by including // or excluding the last element return isSubsetSumRec (arr , n - 1 , sum) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1]); } static boolean isSubsetSum (int [] arr , int sum) { return isSubsetSumRec (arr , arr . length , sum); } public static void main (String [] args) { int [] arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python # Python implementation for subset sum # problem using recursion def isSubsetSumRec (arr , n , sum): # Base Cases if sum == 0 : return True if n == 0 : return False # If the last element is greater # than the sum, ignore it if arr [n - 1] > sum : return isSubsetSumRec (arr , n - 1 , sum) # Check if sum can be obtained by including # or excluding the last element return (isSubsetSumRec (arr , n - 1 , sum) or isSubsetSumRec (arr , n - 1 , sum - arr [n - 1])) def isSubsetSum (arr , sum): return isSubsetSumRec (arr , len (arr), sum) if __name__ == "__main__" : arr = [3 , 34 , 4 , 12 , 5 , 2] sum = 9 if isSubsetSum (arr , sum): print ("True") else : print ("False") C# // C# implementation for subset sum // problem using recursion using System ; class GfG { // Function to check if there is a subset // with the given sum using recursion static bool isSubsetSumRec (int [] arr , int n , int sum) { // Base Cases if (sum == 0) return true ; if (n == 0) return false ; // If the last element is greater than the sum, // ignore it if (arr [n - 1] > sum) return isSubsetSumRec (arr , n - 1 , sum); // Check if sum can be obtained by including // or excluding the last element return isSubsetSumRec (arr , n - 1 , sum) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1]); } static bool isSubsetSum (int [] arr , int sum) { return isSubsetSumRec (arr , arr . Length , sum); } static void Main (string [] args) { int [] arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) Console . WriteLine ("True"); else Console . WriteLine ("False"); } } JavaScript // Javascript implementation for subset sum // problem using recursion function isSubsetSumRec (arr , n , sum) { // Base Cases if (sum === 0) return true ; if (n === 0) return false ; // If the last element is greater than // the sum, ignore it if (arr [n - 1] > sum) { return isSubsetSumRec (arr , n - 1 , sum); } // Check if sum can be obtained by including // or excluding the last element return isSubsetSumRec (arr , n - 1 , sum) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1]); } function isSubsetSum (arr , sum) { return isSubsetSumRec (arr , arr . length , sum); } // Driver code const arr = [3 , 34 , 4 , 12 , 5 , 2]; const sum = 9 ; if (isSubsetSum (arr , sum)) { 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 If we notice carefully, we can observe that the above recursive solution holds the following two properties of Dynamic Programming . 1. Optimal Substructure : The solution to the subset sum problem can be derived from the optimal solutions of smaller subproblems. Specifically, for any given n (the number of elements considered) and a target sum , we can express the recursive relation as follows: If the last element (arr[n-1]) is greater than sum, we cannot include it in our subset isSubsetSum(arr,n,sum) = isSubsetSum(arr,n-1,sum) If the last element is less than or equal to sum, we have two choices: Include the last element in the subset, isSubsetSum(arr,n,sum) = isSubsetSum(arr,n-1,sum-arr[n–1]) Exclude the last element, isSubsetSum(arr,n,sum) = isSubsetSum(arr,n-1,sum) 2. Overlapping Subproblems : When implementing a recursive approach to solve the subset sum problem, we observe that many subproblems are computed multiple times. For instance, when computing isSubsetSum(arr, sum) , where arr[] = {2,3,1,1} and sum = 4 we might need to compute isSubsetSum(1,3) multiple times. Overlapping subproblems The recursive solution involves changing two parameters: the current index in the array (n) and the current target sum (sum). We need to track both parameters, so we create a 2D array of size (n+1) x (sum+1) because the value of n will be in the range [0, n] and sum will be in the range [0, sum]. We initialize the 2D array with -1 to indicate that no subproblems have been computed yet. We check if the value at memo[n][sum] is -1. If it is, we proceed to compute the result. otherwise, we return the stored result. C++ //C++ implementation for subset sum // problem using memoization #include <bits/stdc++.h> using namespace std ; // Recursive function to check if a subset // with the given sum exists bool isSubsetSumRec (vector < int >& arr , int n , int sum , vector < vector < int >> & memo) { // If the sum is zero, we found a subset if (sum == 0) return 1 ; // If no elements are left if (n <= 0) return 0 ; // If the value is already // computed, return it if (memo [n][sum] != -1) return memo [n][sum]; // If the last element is greater than // the sum, ignore it if (arr [n - 1] > sum) return memo [n][sum] = isSubsetSumRec (arr , n - 1 , sum , memo); else { // Include or exclude the last element return memo [n][sum] = isSubsetSumRec (arr , n - 1 , sum , memo) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1], memo); } } // Function to initiate the subset sum check bool isSubsetSum (vector < int >& arr , int sum) { int n = arr . size (); vector < vector < int >> memo (n + 1 , vector < int > (sum + 1 , -1)); return isSubsetSumRec (arr , n , sum , memo); } int main () { vector < int > arr = { 1 , 5 , 3 , 7 , 4 }; int sum = 12 ; if (isSubsetSum (arr , sum)) { cout << "True" << endl ; } else { cout << "False" << endl ; } return 0 ; } Java //Java implementation for subset sum // problem using memoization import java.util.Arrays ; class GfG { // Recursive function to check if a subset // with the given sum exists static boolean isSubsetSumRec (int [] arr , int n , int sum , int [][] memo) { // If the sum is zero, we found a subset if (sum == 0) { return true ; } // If no elements are left if (n <= 0) { return false ; } // If the value is already computed, return it if (memo [n][sum] != - 1) { return memo [n][sum] == 1 ; } // If the last element is greater than the sum, // ignore it if (arr [n - 1] > sum) { memo [n][sum] = isSubsetSumRec (arr , n - 1 , sum , memo) ? 1 : 0 ; } else { // Include or exclude the last element directly memo [n][sum] = (isSubsetSumRec (arr , n - 1 , sum , memo) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1] , memo)) ? 1 : 0 ; } return memo [n][sum] == 1 ; } // Function to initiate the subset sum check static boolean isSubsetSum (int [] arr , int sum) { int n = arr . length ; int [][] memo = new int [n + 1][sum + 1] ; for (int [] row : memo) { Arrays . fill (row , - 1); } return isSubsetSumRec (arr , n , sum , memo); } public static void main (String [] args) { int [] arr = { 1 , 5 , 3 , 7 , 4 }; int sum = 12 ; if (isSubsetSum (arr , sum)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python # Python implementation for subset sum # problem using memoization def isSubsetSumRec (arr , n , sum , memo): # If the sum is zero, we found # a subset if sum == 0 : return True # If no elements are left if n <= 0 : return False # If the value is already # computed, return it if memo [n][sum] != - 1 : return memo [n][sum] # If the last element is greater # than the sum, ignore it if arr [n - 1] > sum : memo [n][sum] = isSubsetSumRec (arr , n - 1 , sum , memo) else : # Include or exclude the last element # directly memo [n][sum] = (isSubsetSumRec (arr , n - 1 , sum , memo) or isSubsetSumRec (arr , n - 1 , sum - arr [n - 1], memo)) return memo [n][sum] def isSubsetSum (arr , sum): n = len (arr) memo = [[- 1 for _ in range (sum + 1)] for _ in range (n + 1)] return isSubsetSumRec (arr , n , sum , memo) if __name__ == "__main__" : arr = [1 , 5 , 3 , 7 , 4] sum = 12 if isSubsetSum (arr , sum): print ("True") else : print ("False") C# //C# implementation for subset sum // problem using memoization using System ; class GfG { // Recursive function to check if a subset with // the given sum exists static bool isSubsetSumRec (int [] arr , int n , int sum , int [,] memo) { // If the sum is zero, we found a subset if (sum == 0) return true ; // If no elements are left if (n <= 0) return false ; // If the value is already computed, // return it if (memo [n , sum] != - 1) return memo [n , sum] == 1 ; // If the last element is greater // than the sum, ignore it if (arr [n - 1] > sum) memo [n , sum] = isSubsetSumRec (arr , n - 1 , sum , memo) ? 1 : 0 ; else { // Include or exclude the last element directly memo [n , sum] = (isSubsetSumRec (arr , n - 1 , sum , memo) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1], memo)) ? 1 : 0 ; } return memo [n , sum] == 1 ; } // Function to initiate the subset sum check static bool isSubsetSum (int [] arr , int sum) { int n = arr . Length ; int [,] memo = new int [n + 1 , sum + 1]; for (int i = 0 ; i <= n ; i ++) for (int j = 0 ; j <= sum ; j ++) memo [i , j] = - 1 ; return isSubsetSumRec (arr , n , sum , memo); } static void Main () { int [] arr = { 1 , 5 , 3 , 7 , 4 }; int sum = 12 ; if (isSubsetSum (arr , sum)) Console . WriteLine ("True"); else Console . WriteLine ("False"); } } JavaScript //Javascript implementation for subset sum // problem using memoization function isSubsetSumRec (arr , n , sum , memo) { // If the sum is zero, we found a subset if (sum === 0) return true ; // If no elements are left if (n <= 0) return false ; // If the value is already computed, // return it if (memo [n][sum] !== - 1) return memo [n][sum] === 1 ; // If the last element is greater than // the sum, ignore it if (arr [n - 1] > sum) { memo [n][sum] = isSubsetSumRec (arr , n - 1 , sum , memo) ? 1 : 0 ; } else { // Include or exclude the last element directly memo [n][sum] = (isSubsetSumRec (arr , n - 1 , sum , memo) || isSubsetSumRec (arr , n - 1 , sum - arr [n - 1], memo)) ? 1 : 0 ; } return memo [n][sum] === 1 ; } // Function to initiate the subset sum check function isSubsetSum (arr , sum) { const n = arr . length ; const memo = Array . from (Array (n + 1), () => Array (sum + 1). fill (- 1)); return isSubsetSumRec (arr , n , sum , memo); } const arr = [1 , 5 , 3 , 7 , 4]; const sum = 12 ; if (isSubsetSum (arr , sum)) { 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 The approach is similar to the previous one. just instead of breaking down the problem recursively, we iteratively build up the solution by calculating in bottom-up manner. So we will create a 2D array of size (n + 1) * (sum + 1) of type boolean. The state dp[i][j] will be true if there exists a subset of elements from arr[0 . . . i] with sum = 'j'. The dynamic programming relation is as follows: if (arr[i-1] > j) dp[i][j] = dp[i-1][j] else dp[i][j] = dp[i-1][j] OR dp[i-1][j-arr[i-1]] This means that if the current element has a value greater than the 'current sum value' we will copy the answer for previous cases and if the current sum value is greater than the 'ith' element we will see if any of the previous states have already computed the sum= j OR any previous states computed a value 'j - arr[i]' which will solve our purpose. C++ //C++ implementation for subset sum // problem using tabulation #include <bits/stdc++.h> using namespace std ; // Function to check if there is a subset of arr[] // with sum equal to the given sum using tabulation with vectors bool isSubsetSum (vector < int > & arr , int sum) { int n = arr . size (); // 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 = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) cout << "True" << endl ; else cout << "False" << endl ; return 0 ; } Java //Java implementation for subset sum // problem using tabulation import java.util.* ; class GfG { // Function to check if there is a subset of arr[] // with sum equal to the given sum using tabulation static boolean isSubsetSum (int [] arr , int sum) { int n = arr . length ; // 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 = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python # Python implementation for subset sum # problem using tabulation def isSubsetSum (arr , sum): n = len (arr) # Create a 2D list for storing # results of subproblems dp = [[False] * (sum + 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 , sum + 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][sum] if __name__ == "__main__" : arr = [3 , 34 , 4 , 12 , 5 , 2] sum_value = 9 if isSubsetSum (arr , sum_value): print ("True") else : print ("False") C# //C# implementation for subset sum // problem using tabulation using System ; class GfG { // Function to check if there is a subset of arr[] // with sum equal to the given sum using tabulation static bool isSubsetSum (int [] arr , int sum) { int n = arr . Length ; // 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 (string [] args) { int [] arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) Console . WriteLine ("True"); else Console . WriteLine ("False"); } } JavaScript //Javascript implementation for subset sum // problem using tabulation function isSubsetSum (arr , sum) { const n = arr . length ; // Create a 2D array for storing results // of subproblems const dp = Array . from (Array (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 = [3 , 34 , 4 , 12 , 5 , 2]; const sum = 9 ; if (isSubsetSum (arr , sum)) { console . log ("True"); } else { console . log ("False"); } Output True [Expected Approach] Using Space Optimized DP – O(sum*n) Time and O(sum) Space In previous approach of dynamic programming we have derive the relation between states as given below: if (arr[i-1] > j) dp[i][j] = dp[i-1][j] else dp[i][j] = dp[i-1][j] OR dp[i-1][j-arr[i-1]] If we observe that for calculating current dp[i][j] state we only need previous row dp[i-1][j] or dp[i-1][j-arr[i-1]] . There is no need to store all the previous states just one previous state is used to compute result. Approach: Define two arrays prev and curr of size sum+1 to store the just previous row result and current row result respectively. Once curr array is calculated then curr becomes our prev for the next row. When all rows are processed the answer is stored in prev array. C++ // C++ Program for Space Optimized Dynamic Programming // Solution to Subset Sum Problem #include <bits/stdc++.h> using namespace std ; // Returns true if there is a subset of arr[] // with sum equal to given sum bool isSubsetSum (vector < int > arr , int sum) { 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 = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum) == true) cout << "True" ; else cout << "False" ; return 0 ; } Java // Java Program for Space Optimized Dynamic Programming // Solution to Subset Sum Problem import java.util.Arrays ; class GfG { // Returns true if there is a subset of arr[] // with sum equal to given sum static boolean isSubsetSum (int [] arr , int sum) { 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]] ; } } // Update prev to be the current row System . arraycopy (curr , 0 , prev , 0 , sum + 1); } return prev [sum] ; } public static void main (String [] args) { int [] arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) { System . out . println ("True"); } else { System . out . println ("False"); } } } Python # Python Program for Space Optimized Dynamic Programming # Solution to Subset Sum Problem def isSubsetSum (arr , sum): n = len (arr) prev = [False] * (sum + 1) curr = [False] * (sum + 1) # Base case: sum 0 can always # be achieved prev [0] = True # Fill the dp table in a # bottom-up manner for i in range (1 , n + 1): for j in range (sum + 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 [sum] if __name__ == "__main__" : arr = [3 , 34 , 4 , 12 , 5 , 2] sum_value = 9 if isSubsetSum (arr , sum_value): print ("True") else : print ("False") C# // C# Program for Space Optimized Dynamic Programming // Solution to Subset Sum Problem using System ; class GfG { static bool isSubsetSum (int [] arr , int sum) { int n = arr . Length ; bool [] prev = new bool [sum + 1]; bool [] curr = new bool [sum + 1]; // Base case: sum 0 can // always be achieved prev [0] = true ; // Fill the dp table in a // 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]]; } Array . Copy (curr , prev , sum + 1); } return prev [sum]; } static void Main () { int [] arr = { 3 , 34 , 4 , 12 , 5 , 2 }; int sum = 9 ; if (isSubsetSum (arr , sum)) Console . WriteLine ("True"); else Console . WriteLine ("False"); } } JavaScript // Javascript Program for Space Optimized Dynamic Programming // Solution to Subset Sum Problem function isSubsetSum (arr , sum) { const n = arr . length ; const prev = new Array (sum + 1). fill (false); const curr = new Array (sum + 1). fill (false); // Base case: sum 0 can always // be achieved prev [0] = true ; // Fill the dp table in a // 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]]; } } // Update prev to be the current row for (let j = 0 ; j <= sum ; j ++) { prev [j] = curr [j]; } } return prev [sum]; } // Driver code const arr = [3 , 34 , 4 , 12 , 5 , 2]; const sum = 9 ; if (isSubsetSum (arr , sum)) { console . log ("True"); } else { console . log ("False"); } Output True Related articles: Subset Sum Problem in O(sum) space Perfect Sum Problem (Print all subsets with given sum) Comment Article Tags: Article Tags: Dynamic Programming DSA Arrays Amazon Adobe Drishti-Soft subset Adobe-Question + 4 More