Stickler Thief - GeeksforGeeks

Source: https://www.geeksforgeeks.org/maximum-sum-such-that-no-two-elements-are-adjacent/

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 Stickler Thief Last Updated : 23 Jul, 2025 Stickler the thief wants to loot money from a society having n houses in a single line. He is a weird person and follows a certain rule when looting the houses. According to the rule, he will never loot two consecutive houses . At the same time, he wants to maximize the amount he loots. The thief knows which house has what amount of money but is unable to come up with an optimal looting strategy. He asks for your help to find the maximum money he can get if he strictly follows the rule. ith house has arr [i] amount of money present in it. Examples: Input: arr[] = [6, 5, 5, 7, 4] Output: 15 Explanation: Maximum amount he can get by looting 1st, 3rd and 5th house. Which is 6+5+4=15. Input: arr[] = [1, 5, 3] Output: 5 Explanation: Loot only 2nd house and get maximum amount of 5. Input: arr[] = [4, 4, 4, 4] Output: 8 Explanation: The optimal choice is to loot every alternate house. Looting the 1st and 3rd houses, or the 2nd and 4th, both give a maximum total of 4 + 4 = 8. Try it on GfG Practice Table of Content [Naive Approach] Using Recursion- O(2^n) Time and O(n) Space [Better Approach 1] Using Memoization - O(n) Time and O(n) Space [Better Approach 2] Using Tabulation - O(n) Time and O(n) Space [Expected Approach] Using Space-Optimized DP - O(n) Time and O(1) Space [Naive Approach] Using Recursion- O(2^n) Time and O(n) Space The idea is to explore all the possibilities for each house using Recursion . We can start from the last house and for each house, we have two choices: Rob the current house and skip the house just before it. Skip the current house and move to the next house. So, the recurrence relation will be: maxLootRec(n) = max(arr[n - 1] + maxLootRec(n - 2), maxLootRec(n - 1)) , where maxLootRec(n) returns the maximum amount of money which can be stolen if n houses are left. C++ #include <bits/stdc++.h> using namespace std ; // Calculate the maximum stolen value recursively int maxLootRec (vector < int >& arr , int n) { // If no houses are left, return 0. if (n <= 0) return 0 ; // If only 1 house is left, rob it. if (n == 1) return arr [0]; // Two Choices: Rob the nth house and do not rob the nth house int pick = arr [n - 1] + maxLootRec (arr , n - 2); int notPick = maxLootRec (arr , n - 1); // Return the max of two choices return max (pick , notPick); } // Function to calculate the maximum stolen value int findMaxSum (vector < int >& arr) { int n = arr . size (); // Call the recursive function for n houses return maxLootRec (arr , n); } int main () { vector < int > arr = { 6 , 5 , 5 , 7 , 4 }; cout << findMaxSum (arr); return 0 ; } Java class GfG { // Calculate the maximum stolen value recursively static int maxLootRec (int [] arr , int n) { // If no houses are left, return 0. if (n <= 0) return 0 ; // If only 1 house is left, rob it. if (n == 1) return arr [0] ; // Two Choices: Rob the nth house and do not rob the nth house int pick = arr [n - 1] + maxLootRec (arr , n - 2); int notPick = maxLootRec (arr , n - 1); // Return the max of two choices return Math . max (pick , notPick); } // Function to calculate the maximum stolen value static int findMaxSum (int [] arr) { int n = arr . length ; // Call the recursive function for n houses return maxLootRec (arr , n); } public static void main (String [] args) { int [] arr = { 6 , 5 , 5 , 7 , 4 }; System . out . println (findMaxSum (arr)); } } Python def maxLootRec (arr , n): # If no houses are left, return 0. if n <= 0 : return 0 # If only 1 house is left, rob it. if n == 1 : return arr [0] # Two Choices: Rob the nth house and do not rob the nth house pick = arr [n - 1] + maxLootRec (arr , n - 2) notPick = maxLootRec (arr , n - 1) # Return the max of two choices return max (pick , notPick) # Function to calculate the maximum stolen value def findMaxSum (arr): n = len (arr) # Call the recursive function for n houses return maxLootRec (arr , n) if __name__ == "__main__" : arr = [6 , 5 , 5 , 7 , 4] print (findMaxSum (arr)) C# using System ; class GfG { // Calculate the maximum stolen value recursively static int maxLootRec (int [] arr , int n) { // If no houses are left, return 0. if (n <= 0) return 0 ; // If only 1 house is left, rob it. if (n == 1) return arr [0]; // Two Choices: Rob the nth house and do not rob the nth house int pick = arr [n - 1] + maxLootRec (arr , n - 2); int notPick = maxLootRec (arr , n - 1); // Return the max of two choices return Math . Max (pick , notPick); } // Function to calculate the maximum stolen value static int findMaxSum (int [] arr) { int n = arr . Length ; // Call the recursive function for n houses return maxLootRec (arr , n); } static void Main () { int [] arr = { 6 , 5 , 5 , 7 , 4 }; Console . WriteLine (findMaxSum (arr)); } } JavaScript function maxLootRec (arr , n) { // If no houses are left, return 0. if (n <= 0) return 0 ; // If only 1 house is left, rob it. if (n === 1) return arr [0]; // Two Choices: Rob the nth house and do not rob the nth house let pick = arr [n - 1] + maxLootRec (arr , n - 2); let notPick = maxLootRec (arr , n - 1); // Return the max of two choices return Math . max (pick , notPick); } // Function to calculate the maximum stolen value function findMaxSum (arr) { let n = arr . length ; // Call the recursive function for n houses return maxLootRec (arr , n); } let arr = [6 , 5 , 5 , 7 , 4]; console . log (findMaxSum (arr)); Output 15 [Better Approach 1] Using Memoization - O(n) Time and O(n) Space The above solution has optimal substructure and overlapping subproblems . See the below recursion tree, maxLootRec(2) is being evaluated twice. Recursion Tree for House Robber We can optimize this solution using a memo array of size (n + 1), such that memo[i] represents the maximum value that can be collected from first i houses. Please note that there is only one parameter that changes in recursion and the range of this parameter is from 0 to n. C++ #include <bits/stdc++.h> using namespace std ; int maxLootRec (const vector < int >& arr , int n , vector < int >& memo) { if (n <= 0) return 0 ; if (n == 1) return arr [0]; // Check if the result is already computed if (memo [n] != -1) return memo [n]; int pick = arr [n - 1] + maxLootRec (arr , n - 2 , memo); int notPick = maxLootRec (arr , n - 1 , memo); // Store the max of two choices in the memo array and return it memo [n] = max (pick , notPick); return memo [n]; } int findMaxSum (vector < int >& arr) { int n = arr . size (); // Initialize memo array with -1 vector < int > memo (n + 1 , -1); return maxLootRec (arr , n , memo); } int main () { vector < int > arr = { 6 , 5 , 5 , 7 , 4 }; cout << findMaxSum (arr); return 0 ; } Java import java.util.Arrays ; class GfG { static int maxLootRec (int [] arr , int n , int [] memo) { if (n <= 0) return 0 ; if (n == 1) return arr [0] ; // Check if the result is already computed if (memo [n] != - 1) return memo [n] ; int pick = arr [n - 1] + maxLootRec (arr , n - 2 , memo); int notPick = maxLootRec (arr , n - 1 , memo); // Store the max of two choices in the memo array and return it memo [n] = Math . max (pick , notPick); return memo [n] ; } // Function to calculate the maximum stolen value static int findMaxSum (int [] arr) { int n = arr . length ; // Initialize memo array with -1 int [] memo = new int [n + 1] ; Arrays . fill (memo , - 1); return maxLootRec (arr , n , memo); } public static void main (String [] args) { int [] arr = { 6 , 5 , 5 , 7 , 4 }; System . out . println (findMaxSum (arr)); } } Python def maxLootRec (arr , n , memo): if n <= 0 : return 0 if n == 1 : return arr [0] # Check if the result is already computed if memo [n] != - 1 : return memo [n] pick = arr [n - 1] + maxLootRec (arr , n - 2 , memo) notPick = maxLootRec (arr , n - 1 , memo) # Store the max of two choices in the memo array and return it memo [n] = max (pick , notPick) return memo [n] def findMaxSum (arr): n = len (arr) # Initialize memo array with -1 memo = [- 1] * (n + 1) return maxLootRec (arr , n , memo) if __name__ == "__main__" : arr = [6 , 5 , 5 , 7 , 4] print (findMaxSum (arr)) C# // C# Program to solve House Robber Problem using // Memoization using System ; class GfG { static int MaxLootRec (int [] arr , int n , int [] memo) { if (n <= 0) return 0 ; if (n == 1) return arr [0]; // Check if the result is already computed if (memo [n] != - 1) return memo [n]; int pick = arr [n - 1] + MaxLootRec (arr , n - 2 , memo); int notPick = MaxLootRec (arr , n - 1 , memo); // Store the max of two choices in the memo array // and return it memo [n] = Math . Max (pick , notPick); return memo [n]; } static int findMaxSum (int [] arr) { // Initialize memo array with -1 int n = arr . Length ; int [] memo = new int [n + 1]; for (int i = 0 ; i <= n ; ++ i) { memo [i] = - 1 ; } int result = MaxLootRec (arr , n , memo); return result ; } static void Main () { int [] arr = { 6 , 5 , 5 , 7 , 4 }; Console . WriteLine (findMaxSum (arr)); } } JavaScript // JavaScript Program to solve House Robber Problem using Memoization function maxLootRec (arr , n , memo) { if (n <= 0) return 0 ; if (n === 1) return arr [0]; // Check if the result is already computed if (memo [n] !== - 1) return memo [n]; const pick = arr [n - 1] + maxLootRec (arr , n - 2 , memo); const notPick = maxLootRec (arr , n - 1 , memo); // Store the max of two choices in the memo array and return it memo [n] = Math . max (pick , notPick); return memo [n]; } // Function to calculate the maximum stolen value function findMaxSum (arr) { const n = arr . length ; // Initialize memo array with -1 const memo = new Array (n + 1). fill (- 1); return maxLootRec (arr , n , memo); } const arr = [6 , 5 , 5 , 7 , 4]; console . log (findMaxSum (arr)); Output 15 [Better Approach 2] Using Tabulation - O(n) Time and O(n) Space The idea is to build the solution in bottom-up manner. We create a dp[] array of size n+1 where dp[i] represents the maximum value that can be collected with first i houses. We first fill the known values, dp[0] and dp[1] and then fill the remaining values using the formula: dp[i] = max(arr[i] +

dp[i - 2], dp[i - 1]) . The final result will be stored at dp[n]. C++ #include <bits/stdc++.h> using namespace std ; // Function to calculate the maximum stolen value using bottom-up DP int findMaxSum (vector < int >& arr) { int n = arr . size (); // Create a dp array to store the maximum loot at each house vector < int > dp (n + 1 , 0); // Base cases dp [0] = 0 ; dp [1] = arr [0]; // Fill the dp array using the bottom-up approach for (int i = 2 ; i <= n ; i ++) dp [i] = max (arr [i - 1] + dp [i - 2], dp [i - 1]); return dp [n]; } int main () { vector < int > arr = { 6 , 5 , 5 , 7 , 4 }; cout << findMaxSum (arr) << endl ; return 0 ; } Java class GfG { // Function to calculate the maximum stolen value using bottom-up DP static int findMaxSum (int [] arr) { int n = arr . length ; // Create a dp array to store the maximum loot at each house int [] dp = new int [n + 1] ; // Base cases dp [0] = 0 ; dp [1] = arr [0] ; // Fill the dp array using the bottom-up approach for (int i = 2 ; i <= n ; i ++) { dp [i] = Math . max (arr [i - 1] + dp [i - 2] , dp [i - 1]); } return dp [n] ; } public static void main (String [] args) { int [] arr = { 6 , 5 , 5 , 7 , 4 }; System . out . println (findMaxSum (arr)); } } Python def findMaxSum (arr): n = len (arr) # Create a dp array to store the maximum loot at each house dp = [0] * (n + 1) # Base cases dp [0] = 0 dp [1] = arr [0] # Fill the dp array using the bottom-up approach for i in range (2 , n + 1): dp [i] = max (arr [i - 1] + dp [i - 2], dp [i - 1]) return dp [n] arr = [6 , 5 , 5 , 7 , 4] print (findMaxSum (arr)) C# using System ; class GfG { // Function to calculate the maximum stolen value using bottom-up DP static int findMaxSum (int [] arr) { int n = arr . Length ; // Create a dp array to store the maximum loot at each house int [] dp = new int [n + 1]; // Base cases dp [0] = 0 ; dp [1] = arr [0]; // Fill the dp array using the bottom-up approach for (int i = 2 ; i <= n ; i ++) { dp [i] = Math . Max (arr [i - 1] + dp [i - 2], dp [i - 1]); } return dp [n]; } static void Main () { int [] arr = { 6 , 5 , 5 , 7 , 4 }; Console . WriteLine (findMaxSum (arr)); } } JavaScript function findMaxSum (arr) { const n = arr . length ; // Create a dp array to store the maximum loot at each house const dp = new Array (n + 1). fill (0); // Base cases dp [0] = 0 ; dp [1] = arr [0]; // Fill the dp array using the bottom-up approach for (let i = 2 ; i <= n ; i ++) dp [i] = Math . max (arr [i - 1] + dp [i - 2], dp [i - 1]); return dp [n]; } const arr = [6 , 5 , 5 , 7 , 4]; console . log (findMaxSum (arr)); Output 15 [Expected Approach] Using Space-Optimized DP - O(n) Time and O(1) Space On observing the dp[] array in the previous approach, it can be seen that the answer at the current index depends only on the last two values . In other words, dp[i] depends only on dp[i - 1] and dp[i - 2] . So, instead of storing the result in an array, we can simply use two variables to store the last and second last result. C++ #include <bits/stdc++.h> using namespace std ; // Function to calculate the maximum stolen value int findMaxSum (vector < int > & arr) { int n = arr . size (); if (n == 0) return 0 ; if (n == 1) return arr [0]; // Set previous 2 values int secondLast = 0 , last = arr [0]; // Compute current value using previous two values // The final current value would be our result int res ; for (int i = 1 ; i < n ; i ++) { res = max (arr [i] + secondLast , last); secondLast = last ; last = res ; } return res ; } int main () { vector < int > arr = { 6 , 5 , 5 , 7 , 4 }; cout << findMaxSum (arr) << endl ; return 0 ; } Java import java.util.Arrays ; class GfG { // Function to calculate the maximum stolen value static int findMaxSum (int [] arr) { int n = arr . length ; if (n == 0) return 0 ; if (n == 1) return arr [0] ; // Set previous 2 values int secondLast = 0 , last = arr [0] ; // Compute current value using previous // two values. The final current value // would be our result int res = 0 ; for (int i = 1 ; i < n ; i ++) { res = Math . max (arr [i] + secondLast , last); secondLast = last ; last = res ; } return res ; } public static void main (String [] args) { int [] arr = { 6 , 5 , 5 , 7 , 4 }; System . out . println (findMaxSum (arr)); } } Python def findMaxSum (arr): n = len (arr) if n == 0 : return 0 if n == 1 : return arr [0] # Set previous 2 values secondLast = 0 last = arr [0] # Compute current value using previous two values # The final current value would be our result res = 0 for i in range (1 , n): res = max (arr [i] + secondLast , last) secondLast = last last = res return res arr = [6 , 5 , 5 , 7 , 4] print (findMaxSum (arr)) C# using System ; class GfG { // Function to calculate the maximum stolen value static int findMaxSum (int [] arr) { int n = arr . Length ; if (n == 0) return 0 ; if (n == 1) return arr [0]; // Set previous 2 values int secondLast = 0 , last = arr [0]; // Compute current value using previous two values // The final current value would be our result int res = 0 ; for (int i = 1 ; i < n ; i ++) { res = Math . Max (arr [i] + secondLast , last); secondLast = last ; last = res ; } return res ; } static void Main () { int [] arr = { 6 , 5 , 5 , 7 , 4 }; Console . WriteLine (findMaxSum (arr)); } } JavaScript function findMaxSum (arr) { const n = arr . length ; if (n === 0) return 0 ; if (n === 1) return arr [0]; // Set previous 2 values let secondLast = 0 , last = arr [0]; // Compute current value using previous two values // The final current value would be our result let res ; for (let i = 1 ; i < n ; i ++) { res = Math . max (arr [i] + secondLast , last); secondLast = last ; last = res ; } return res ; } const arr = [6 , 5 , 5 , 7 , 4]; console . log (findMaxSum (arr)); Output 15 Comment Article Tags: Article Tags: Dynamic Programming DSA Arrays Amazon Yahoo Walmart Paytm Accolite Oxigen Wallet OYO + 6 More