Rod Cutting - GeeksforGeeks

Source: https://www.geeksforgeeks.org/cutting-a-rod-dp-13/

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 Rod Cutting Last Updated : 23 Jul, 2025 Given a rod of length n inches and an array price[] . price[i] denotes the value of a piece of length i. The task is to determine the maximum value obtainable by cutting up the rod and selling the pieces. Note: price[] is 1-indexed array. Input: price[] = [1, 5, 8, 9, 10, 17, 17, 20] Output: 22 Explanation : The maximum obtainable value is 22 by cutting in two pieces of lengths 2 and 6, i.e., 5 + 17 = 22. Input : price[] = [3, 5, 8, 9, 10, 17, 17, 20] Output : 24 Explanation : The maximum obtainable value is 24 by cutting the rod into 8 pieces of length 1, i.e, 8*price[1]= 8*3 = 24. Input : price[] = [3] Output : 3 Explanation: There is only 1 way to pick a piece of length 1. Try it on GfG Practice Table of Content Using Recursion – O(n^n) Time and O(n) Space Using Top-Down DP (Memoization) - O(n^2) Time and O(n) Space Using Bottom-Up DP (Tabulation) - O(n^2) Time and O(n) Space Using the idea of Unbounded Knapsack - O(n^2) time and O(n^2) space Using Recursion - O(n^n) Time and O(n) Space The recursive approach involves solving the problem by considering all possible ways to cut the rod into two pieces at every l ength j (where 1<=j<=i), calculating the profit for each cut, and finding the maximum profit among these options. At each step, the rod of length i is divided into two parts: j and i - j . The profit from this cut is the sum of the price of the piece of length j (given as price[j-1]) and the maximum profit obtainable from the remaining rod of length i-j (computed recursively). The base case for this recursion is when i equals 0, where the maximum profit is simply 0. The recurrence relation will be: cutRod(i) = max(price[j-1] + cutRod(i-j)) for (1<=j<=i) Base Case: If i == 0, return 0. C++ // C++ program to find maximum // profit from rod of size n #include <bits/stdc++.h> using namespace std ; int cutRodRecur (int i , vector < int > & price) { // Base case if (i == 0) return 0 ; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = max (ans , price [j -1] + cutRodRecur (i - j , price)); } return ans ; } int cutRod (vector < int > & price) { int n = price . size (); return cutRodRecur (n , price); } int main () { vector < int > price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; cout << cutRod (price); return 0 ; } Java // Java program to find maximum // profit from rod of size n import java.util.* ; class GfG { static int cutRodRecur (int i , int [] price) { // Base case if (i == 0) return 0 ; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = Math . max (ans , price [j - 1] + cutRodRecur (i - j , price)); } return ans ; } static int cutRod (int [] price) { int n = price . length ; return cutRodRecur (n , price); } public static void main (String [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; System . out . println (cutRod (price)); } } Python # Python program to find maximum # profit from rod of size n def cutRodRecur (i , price): # Base case if i == 0 : return 0 ans = 0 # Find maximum value for each cut. # Take value of rod of length j, and # recursively find value of rod of # length (i-j). for j in range (1 , i + 1): ans = max (ans , price [j - 1] + cutRodRecur (i - j , price)) return ans def cutRod (price): n = len (price) return cutRodRecur (n , price) if __name__ == "__main__" : price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20] print (cutRod (price)) C# // C# program to find maximum // profit from rod of size n using System ; class GfG { static int cutRodRecur (int i , int [] price) { // Base case if (i == 0) return 0 ; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = Math . Max (ans , price [j - 1] + cutRodRecur (i - j , price)); } return ans ; } static int cutRod (int [] price) { int n = price . Length ; return cutRodRecur (n , price); } static void Main (string [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; Console . WriteLine (cutRod (price)); } } JavaScript // JavaScript program to find maximum // profit from rod of

size n function cutRodRecur (i , price) { // Base case if (i === 0) return 0 ; let ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (let j = 1 ; j <= i ; j ++) { ans = Math . max (ans , price [j - 1] + cutRodRecur (i - j , price)); } return ans ; } function cutRod (price) { const n = price . length ; return cutRodRecur (n , price); } const price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20]; console . log (cutRod (price)); Output 22 Using Top-Down DP (Memoization) - O(n^2) Time and O(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 : Maximum profit at index i , i.e., cutRod(i) , depends on the optimal solutions of the subproblems cutRod(i-1) , cutRod(i-2), ..., cutRod(0) . By comparing these optimal substructures, we can efficiently calculate the maximum profit at index i. 2. Overlapping Subproblems : While applying a recursive approach in this problem, we notice that certain subproblems are computed multiple times. There is only 1 parameter: i that changes in the recursive solution. So we create a 1D array of size n for memoization . We initialize this array as -1 to indicate nothing is computed initially. Now we modify our recursive solution to first check if the value is -1, then only make recursive calls. This way, we avoid re-computations of the same subproblems. C++ // C++ program to find maximum // profit from rod of size n #include <bits/stdc++.h> using namespace std ; int cutRodRecur (int i , vector < int > & price , vector < int > & memo) { // Base case if (i == 0) return 0 ; // If value is memoized if (memo [i -1] != -1) return memo [i -1]; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = max (ans , price [j -1] + cutRodRecur (i - j , price , memo)); } return memo [i -1] = ans ; } int cutRod (vector < int > & price) { int n = price . size (); vector < int > memo (price . size (), -1); return cutRodRecur (n , price , memo); } int main () { vector < int > price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; cout << cutRod (price); return 0 ; } Java // Java program to find maximum // profit from rod of size n import java.util.* ; class GfG { static int cutRodRecur (int i , int [] price , int [] memo) { // Base case if (i == 0) return 0 ; // If value is memoized if (memo [i - 1] != - 1) return memo [i - 1] ; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = Math . max (ans , price [j - 1] + cutRodRecur (i - j , price , memo)); } return memo [i - 1] = ans ; } static int cutRod (int [] price) { int n = price . length ; int [] memo = new int [n] ; Arrays . fill (memo , - 1); return cutRodRecur (n , price , memo); } public static void main (String [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; System . out . println (cutRod (price)); } } Python # Python program to find maximum # profit from rod of size n def cutRodRecur (i , price , memo): # Base case if i == 0 : return 0 # If value is memoized if memo [i - 1] != - 1 : return memo [i - 1] ans = 0 # Find maximum value for each cut. # Take value of rod of length j, and # recursively find value of rod of # length (i-j). for j in range (1 , i + 1): ans = max (ans , price [j - 1] + cutRodRecur (i - j , price , memo)) memo [i - 1] = ans return ans def cutRod (price): n = len (price) memo = [- 1] * n return cutRodRecur (n , price , memo) if __name__ == "__main__" : price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20] print (cutRod (price)) C# // C# program to find maximum // profit from rod of size n using System ; class GfG { static int cutRodRecur (int i , int [] price , int [] memo) { // Base case if (i == 0) return 0 ; // If value is memoized if (memo [i - 1] != - 1) return memo [i - 1]; int ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (int j = 1 ; j <= i ; j ++) { ans = Math . Max (ans , price [j - 1] + cutRodRecur (i - j , price , memo)); } return memo [i - 1] = ans ; } static int cutRod (int [] price) { int n = price . Length ; int [] memo = new int [n]; Array . Fill (memo , - 1); return cutRodRecur (n , price , memo); } static void Main (string [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; Console . WriteLine (cutRod (price)); } } JavaScript // JavaScript program to find maximum // profit from rod of size n function cutRodRecur (i , price , memo) { // Base case if (i === 0) return 0 ; // If value is memoized if (memo [i - 1] !== - 1) return memo [i - 1]; let ans = 0 ; // Find maximum value for each cut. // Take value of rod of length j, and // recursively find value of rod of // length (i-j). for (let j = 1 ; j <= i ; j ++) { ans = Math . max (ans , price [j - 1] + cutRodRecur (i - j , price , memo)); } memo [i - 1] = ans ; return ans ; } function cutRod (price) { const n = price . length ; const memo = Array (n). fill (- 1); return cutRodRecur (n , price , memo); } const price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20]; console . log (cutRod (price)); Output 22 Using Bottom-Up DP (Tabulation) - O(n^2) Time and O(n) Space The idea is to fill the dp table from bottom to up. For each rod of length i, starting from i = 1 to i = n, find the maximum value that can obtained by cutting it into two piece s of length j and (i-j) . C++ // C++ program to find maximum // profit from rod of size n #include <bits/stdc++.h> using namespace std ; int cutRod (vector < int > & price) { int n = price . size (); vector < int > dp (price . size () + 1 , 0); // Find maximum value for all // rod of length i. for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= i ; j ++) { dp [i] = max (dp [i], price [j -1] + dp [i - j]); } } return

dp [n]; } int main () { vector < int > price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; cout << cutRod (price); return 0 ; } Java // Java program to find maximum // profit from rod of size n import java.util.* ; class GfG { static int cutRod (int [] price) { int n = price . length ; int [] dp = new int [n + 1] ; // Find maximum value for all // rod of length i. for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= i ; j ++) { dp [i] = Math . max (dp [i] , price [j - 1] + dp [i - j]); } } return dp [n] ; } public static void main (String [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; System . out . println (cutRod (price)); } } Python # Python program to find maximum # profit from rod of size n def cutRod (price): n = len (price) dp = [0] * (n + 1) # Find maximum value for all # rod of length i. for i in range (1 , n + 1): for j in range (1 , i + 1): dp [i] = max (dp [i], price [j - 1] + dp [i - j]) return dp [n] if __name__ == "__main__" : price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20] print (cutRod (price)) C# // C# program to find maximum // profit from rod of size n using System ; class GfG { static int cutRod (int [] price) { int n = price . Length ; int [] dp = new int [n + 1]; // Find maximum value for all // rod of length i. for (int i = 1 ; i <= n ; i ++) { for (int j = 1 ; j <= i ; j ++) { dp [i] = Math . Max (dp [i], price [j - 1] + dp [i - j]); } } return dp [n]; } static void Main (string [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; Console . WriteLine (cutRod (price)); } } JavaScript // JavaScript program to find maximum // profit from rod of size n function cutRod (price) { const n = price . length ; const dp = Array (n + 1). fill (0); // Find maximum value for all // rod of length i. for (let i = 1 ; i <= n ; i ++) { for (let j = 1 ; j <= i ; j ++) { dp [i] = Math . max (dp [i], price [j - 1] + dp [i - j]); } } return dp [n]; } const price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20]; console . log (cutRod (price)); Output 22 Using the idea of Unbounded Knapsack - O(n^2) time and O(n^2) space This problem is very similar to the Unbounded Knapsack Problem , where there are multiple occurrences of the same item. Here we consider length of rod as capacity of knapsack. All lengths from 1 to n are considered as weights of items and prices as values of items. C++ // C++ program to find maximum // profit from rod of size n #include <bits/stdc++.h> using namespace std ; // Find the maximum value obtainable from rod // of length j. int cutRodRecur (int i , int j , vector < int > & price , vector < vector < int >> & memo) { // base case if (i == 0 || j == 0) return 0 ; // If value if memoized if (memo [i][j] != -1) return memo [i][j]; // There are two options: // 1. Break it into (i) and (i-j) rods and // take value of ith rod. int take = 0 ; if (i <= j) { take = price [i -1] + cutRodRecur (i , j - i , price , memo); } // 2. Skip i'th length rod. int noTake = cutRodRecur (i -1 , j , price , memo); return memo [i][j] = max (take , noTake); } int cutRod (vector < int > & price) { int n = price . size (); vector < vector < int >> memo (n + 1 , vector < int > (n + 1 , -1)); return cutRodRecur (n , n , price , memo); } int main () { vector < int > price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; cout << cutRod (price); return 0 ; } Java // Java program to find maximum // profit from rod of size n import java.util.Arrays ; class GfG { // Find the maximum value obtainable from rod // of length j. static int cutRodRecur (int i , int j , int [] price , int [][] memo) { // base case if (i == 0 || j == 0) return 0 ; // If value is memoized if (memo [i][j] != - 1) return memo [i][j] ; // There are two options: // 1. Break it into (i) and (i-j) rods and // take value of ith rod. int take = 0 ; if (i <= j) { take = price [i - 1] + cutRodRecur (i , j - i , price , memo); } // 2. Skip i'th length rod. int noTake = cutRodRecur (i - 1 , j , price , memo); return memo [i][j] = Math . max (take , noTake); } static int cutRod (int [] price) { int n = price . length ; int [][] memo = new int [n + 1][n + 1] ; for (int [] row : memo) { Arrays . fill (row , - 1); } return cutRodRecur (n , n , price , memo); } public static void main (String [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; System . out . println (cutRod (price)); } } Python # Python program to find maximum # profit from rod of size n def cutRodRecur (i , j , price , memo): # base case if i == 0 or j == 0 : return 0 # If value is memoized if memo [i][j] != - 1 : return memo [i][j] # There are two options: # 1. Break it into (i) and (i-j) rods and # take value of ith rod. take = 0 if i <= j : take = price [i - 1] + cutRodRecur (i , j - i , price , memo) # 2. Skip i'th length rod. noTake = cutRodRecur (i - 1 , j , price , memo) memo [i][j] = max (take , noTake) return memo [i][j] def cutRod (price): n = len (price) memo = [[- 1 for _ in range (n + 1)] for _ in range (n + 1)] return cutRodRecur (n , n , price , memo) if __name__ == "__main__" : price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20] print (cutRod (price)) C# // C# program to find maximum // profit from rod of size n using System ; class GfG { // Find the maximum value obtainable from rod // of length j. static int cutRodRecur (int i , int j , int [] price , int [,] memo) { // base case if (i == 0 || j == 0) return 0 ; // If value is memoized if (memo [i , j] != - 1) return memo [i , j]; // There are two options: // 1. Break it into (i) and (i-j) rods and // take value of ith rod. int take = 0 ; if (i <= j) { take = price [i - 1] + cutRodRecur (i , j - i , price , memo); } // 2. Skip i'th length rod. int noTake = cutRodRecur (i - 1 , j , price , memo); memo [i , j] = Math . Max (take , noTake); return memo [i , j]; } static int cutRod (int [] price) { int n = price . Length ; int [,] memo = new int [n + 1 , n + 1]; for (int i = 0 ; i <= n ; i ++) { for (int j = 0 ; j <= n ; j ++) { memo [i , j] = - 1 ; } } return cutRodRecur (n , n , price , memo); } static void Main (string [] args) { int [] price = { 1 , 5 , 8 , 9 , 10 , 17 , 17 , 20 }; Console . WriteLine (cutRod (price)); } } JavaScript // JavaScript program to find maximum // profit from rod of

size n function cutRodRecur (i , j , price , memo) { // base case if (i === 0 || j === 0) return 0 ; // If value is memoized if (memo [i][j] !== - 1) return memo [i][j]; // There are two options: // 1. Break it into (i) and (i-j) rods and // take value of ith rod. let take = 0 ; if (i <= j) { take = price [i - 1] + cutRodRecur (i , j - i , price , memo); } // 2. Skip i'th length rod. const noTake = cutRodRecur (i - 1 , j , price , memo); memo [i][j] = Math . max (take , noTake); return memo [i][j]; } function cutRod (price) { const n = price . length ; const memo = Array . from ({ length : n + 1 }, () => Array (n + 1). fill (- 1)); return cutRodRecur (n , n , price , memo); } const price = [1 , 5 , 8 , 9 , 10 , 17 , 17 , 20]; console . log (cutRod (price)); Output 22 Comment Article Tags: Article Tags: Dynamic Programming DSA