Job Sequencing Problem - GeeksforGeeks

Source: https://www.geeksforgeeks.org/job-sequencing-problem/

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 Job Sequencing Problem Last Updated : 8 Sep, 2025 Given two arrays, deadline[] and profit[] , where deadline[i] is the last time unit by which the i-th job must be completed, and profit[i] is the profit earned from completing it. Each job takes 1 unit time, and only one job can be scheduled at a time. A job earns profit only if finished within its deadline. Find the number of jobs completed and maximum profit. Examples: Input: deadline[] = [4, 1, 1, 1], profit[] = [20, 10, 40, 30] Output: [2, 60] Explanation: Job 1 (profit 20, deadline 4) can be scheduled. Among the three jobs with deadline 1, only one fits, so we pick the highest profit (40). Hence, 2 jobs with total profit = 60. Input: deadline[] = [2, 1, 2, 1, 1], profit[] = [100, 19, 27, 25, 15] Output: [2, 127] Explanation: Picking the job with profit 100 (deadline 2) and the job with profit 27 (deadline 2); they can occupy the two available slots before deadline 2. Thus 2 jobs are scheduled for a maximum total profit of 127. Try it on GfG Practice Table of Content [Naive Approach] Using Sorting - O(n ^ 2) Time and O(n) Space [Expected Approach] Using Sorting and MinHeap- O(n * log(n)) Time and O(n) Space [Alternate Approach] Using Disjoint Set - O(n * log(d)) Time and O(d) Space [Naive Approach] Using Sorting - O(n 2) Time and O(n) Space The idea is to sort the jobs in descending order of profit and for each job, try to place it in the latest available slot before its deadline. This ensures maximum profit while keeping earlier slots free for other jobs. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; vector < int > jobSequencing (vector < int > & deadline , vector < int > & profit) { int n = deadline . size (); int cnt = 0 ; int totProfit = 0 ; // pair the profit and deadline of // all the jos together vector < pair < int , int >> jobs ; for (int i = 0 ; i < n ; i ++) { jobs . push_back ({ profit [i], deadline [i]}); } // sort the jobs based on profit // in decreasing order sort (jobs . begin (), jobs . end (), greater < pair < int , int >> ()); vector < int > slot (n , 0); for (int i = 0 ; i < n ; i ++) { int start = min (n , jobs [i]. second) - 1 ; for (int j = start ; j >= 0 ; j --) { // if slot is empty if (slot [j] == 0) { slot [j] = 1 ; cnt ++ ; totProfit += jobs [i]. first ; break ; } } } return { cnt , totProfit }; } int main () { vector < int > deadline = { 2 , 1 , 2 , 1 , 1 }; vector < int > profit = { 100 , 19 , 27 , 25 , 15 }; vector < int > ans = jobSequencing (deadline , profit); cout << ans [0] << " " << ans [1]; return 0 ; } Java import java.util.ArrayList ; class GfG { public static ArrayList < Integer > jobSequencing (int [] deadline , int [] profit) { int n = deadline . length ; int cnt = 0 ; int totProfit = 0 ; // pair the profit and deadline of all the jobs together ArrayList < int []> jobs = new ArrayList <> (); for (int i = 0 ; i < n ; i ++) { jobs . add (new int [] { profit [i] , deadline [i] }); } // sort the jobs based on profit in decreasing order jobs . sort ((a , b) -> Integer . compare (b [0] , a [0])); int [] slot = new int [n] ; for (int i = 0 ; i < n ; i ++) { int start = Math . min (n , jobs . get (i) [1]) - 1 ; for (int j = start ; j >= 0 ; j --) { // if slot is empty if (slot [j] == 0) { slot [j] = 1 ; cnt ++ ; totProfit += jobs . get (i) [0] ; break ; } } } ArrayList < Integer > result = new ArrayList <> (); result . add (cnt); result . add (totProfit); return result ; } public static void main (String [] args) { int [] deadline = { 2 , 1 , 2 , 1 , 1 }; int [] profit = { 100 , 19 , 27 , 25 , 15 }; ArrayList < Integer > ans = jobSequencing (deadline , profit); System . out . println (ans . get (0) + " " + ans . get (1)); } } Python from typing import List , Tuple def jobSequencing (deadline : List [int], profit : List [int]) -> List [int]: n = len (deadline) cnt = 0 totProfit = 0 # pair the profit and deadline of # all the jobs together jobs = [(profit [i], deadline [i]) for i in range (n)] # sort the jobs based on profit # in decreasing order jobs . sort (key = lambda x : x [0], reverse = True) slot = [0] * n for i in range (n): start = min (n , jobs [i][1]) - 1 for j in range (start , - 1 , - 1): # if slot is empty if slot [j] == 0 : slot [j] = 1 cnt += 1 totProfit += jobs [i][0] break return [cnt , totProfit] if __name__ == "__main__" : deadline = [2 , 1 , 2 , 1 , 1] profit = [100 , 19 , 27 , 25 , 15] ans = jobSequencing (deadline , profit) print (ans [

0], ans [1]) C# using System ; using System.Collections.Generic ; class GfG { static List < int > jobSequencing (int [] deadline , int [] profit) { int n = deadline . Length ; int cnt = 0 ; int totProfit = 0 ; List < Tuple < int , int >> jobs = new List < Tuple < int , int >> (); // pair the profit and deadline of // all the jos together for (int i = 0 ; i < n ; i ++) { jobs . Add (new Tuple < int , int > (profit [i], deadline [i])); } // sort the jobs based on profit // in decreasing order jobs . Sort ((a , b) => b . Item1 . CompareTo (a . Item1)); int [] slot = new int [n]; for (int i = 0 ; i < n ; i ++) { int start = Math . Min (n , jobs [i]. Item2) - 1 ; for (int j = start ; j >= 0 ; j --) { // if slot is empty if (slot [j] == 0) { slot [j] = 1 ; cnt ++ ; totProfit += jobs [i]. Item1 ; break ; } } } return new List < int > { cnt , totProfit }; } static void Main () { int [] deadline = { 2 , 1 , 2 , 1 , 1 }; int [] profit = { 100 , 19 , 27 , 25 , 15 }; List < int > ans = jobSequencing (deadline , profit); Console . WriteLine (ans [0] + " " + ans [1]); } } JavaScript function jobSequencing (deadline , profit) { let n = deadline . length ; let cnt = 0 ; let totProfit = 0 ; // pair the profit and deadline of // all the jobs together let jobs = []; for (let i = 0 ; i < n ; i ++) { jobs . push ([profit [i], deadline [i]]); } // sort the jobs based on profit // in decreasing order jobs . sort ((a , b) => b [0] - a [0]); let slot = Array (n). fill (0); for (let i = 0 ; i < n ; i ++) { let start = Math . min (n , jobs [i][1]) - 1 ; for (let j = start ; j >= 0 ; j --) { // if slot is empty if (slot [j] === 0) { slot [j] = 1 ; cnt ++ ; totProfit += jobs [i][0]; break ; } } } return [cnt , totProfit]; } let deadline = [2 , 1 , 2 , 1 , 1]; let profit = [100 , 19 , 27 , 25 , 15]; let ans = jobSequencing (deadline , profit); console . log (ans [0] + ' ' + ans [1]); Output 2 127 [Expected Approach] Using Sorting and MinHeap - O(n * log(n)) Time and O(n) Space The idea is to sort the jobs based on their deadlines in ascending order. This ensures that jobs with earlier deadlines are processed first, preventing situations where a job with a short deadline remains unscheduled because a job with a later deadline was chosen instead. We use a min-heap to keep track of the selected jobs, allowing us to efficiently replace lower-profit jobs when a more profitable job becomes available. Step by Step implementation: Store jobs as pairs of (Deadline, Profit). Sort Jobs array in ascending order of deadline. For each job in the sorted list: If the job can be scheduled within its deadline, push its profit into the heap. If the heap is full (equal to deadline), replace the existing lowest profit job with the current job if it has a higher profit. This ensures that we always keep the most profitable jobs within the available slots. Traverse through the heap and store the total profit and the count of jobs. C++ #include <iostream> #include <vector> #include <algorithm> #include <queue> using namespace std ; vector < int > jobSequencing (vector < int > & deadline , vector < int > & profit) { int n = deadline . size (); vector < int > ans = { 0 , 0 }; vector < pair < int , int >> jobs ; for (int i = 0 ; i < n ; i ++) { jobs . push_back ({ deadline [i], profit [i]}); } // sort the jobs based on deadline // in ascending order sort (jobs . begin (), jobs . end ()); priority_queue < int , vector < int > , greater < int >> pq ; for (int i = 0 ; i < jobs . size (); i ++) { // if job can be scheduled within its deadline if (jobs [i]. first > pq . size ()) pq . push (jobs [i]. second); // replace the job with the lowest profit else if (! pq . empty () && pq . top () < jobs [i]. second) { pq . pop (); pq . push (jobs [i]. second); } } while (! pq . empty ()) { ans [1] += pq . top (); pq . pop (); ans [0] ++ ; } return ans ; } int main () { vector < int > deadline = { 2 , 1 , 2 , 1 , 1 }; vector < int > profit = { 100 , 19 , 27 , 25 , 15 }; vector < int > ans = jobSequencing (deadline , profit); cout << ans [0] << " " << ans [1]; return 0 ; } Java import java.util.ArrayList ; import java.util.Arrays ; import java.util.List ; import java.util.Comparator ; import java.util.PriorityQueue ; public class GfG { static ArrayList < Integer > jobSequencing (int [] deadline , int [] profit) { int n = deadline . length ; ArrayList < Integer > ans = new ArrayList <> (Arrays . asList (0 , 0)); List < int []> jobs = new ArrayList <> (); for (int i = 0 ; i < n ; i ++) { jobs . add (new int [] { deadline [i] , profit [i] }); } // sort jobs by deadline in ascending order jobs . sort (Comparator . comparingInt (a -> a [0])); PriorityQueue < Integer > pq = new PriorityQueue <> (); for (int i = 0 ; i < jobs . size (); i ++) { int [] job = jobs . get (i); // if job can be scheduled within its deadline if (job [0] > pq . size ()) { pq . add (job [1]); } // replace the job with the lowest profit else if (! pq . isEmpty () && pq . peek () < job [1]) { pq . poll (); pq . add (job [1]); } } while (! pq . isEmpty ()) { ans . set (1 , ans . get (1) + pq . poll ()); ans . set (0 , ans . get (0) + 1); } return ans ; } public static void main (String [] args) { int [] deadline = { 2 , 1 , 2 , 1 , 1 }; int [] profit = { 100 , 19 , 27 , 25 , 15 }; ArrayList < Integer > result = jobSequencing (deadline , profit); System . out . println (result . get (0) + " " + result . get (1)); } } Python import heapq def jobSequencing (deadline , profit): n = len (deadline) ans = [0 , 0] jobs = [(deadline [i], profit [i]) for i in range (n)] # sort the jobs based on deadline # in ascending order jobs . sort () pq = [] for job in jobs : # if job can be scheduled within its deadline if job [0] > len (pq): heapq . heappush (pq , job [1]) # Replace the job with the lowest profit elif pq and pq [0] < job [1]: heapq . heappop (pq) heapq . heappush (pq , job [1]) while pq : ans [1] += heapq . heappop (pq) ans [0] += 1 return ans if __name__ == "__main__" : deadline = [2 , 1 , 2 , 1 , 1] profit = [100 , 19 , 27 , 25 , 15] ans = jobSequencing (deadline , profit) print (ans [0], ans [1]) C# using System ; using System.Collections.Generic ; class MinHeap { private List < int > heap = new

List < int > (); public int Count => heap . Count ; public int Peek () { if (heap . Count == 0) throw new InvalidOperationException (); return heap [0]; } public void Push (int val) { heap . Add (val); HeapifyUp (heap . Count - 1); } public int Pop () { if (heap . Count == 0) throw new InvalidOperationException (); int top = heap [0]; heap [0] = heap [heap . Count - 1]; heap . RemoveAt (heap . Count - 1); if (heap . Count > 0) HeapifyDown (0); return top ; } private void HeapifyUp (int idx) { while (idx > 0) { int parent = (idx - 1) / 2 ; if (heap [parent] <= heap [idx]) break ; Swap (parent , idx); idx = parent ; } } private void HeapifyDown (int idx) { int n = heap . Count ; while (true) { int left = 2 * idx + 1 ; int right = 2 * idx + 2 ; int smallest = idx ; if (left < n && heap [left] < heap [smallest]) smallest = left ; if (right < n && heap [right] < heap [smallest]) smallest = right ; if (smallest == idx) break ; Swap (idx , smallest); idx = smallest ; } } private void Swap (int i , int j) { int temp = heap [i]; heap [i] = heap [j]; heap [j] = temp ; } } class GfG { static List < int > jobSequencing (int [] deadline , int [] profit) { int n = deadline . Length ; var jobs = new List < (int d , int p) > (); for (int i = 0 ; i < n ; i ++) jobs . Add ((deadline [i], profit [i])); // sort jobs by deadline jobs . Sort ((a , b) => a . d . CompareTo (b . d)); var minHeap = new MinHeap (); foreach (var job in jobs) { // if job can be scheduled within its deadline if (minHeap . Count < job . d) { minHeap . Push (job . p); } // replace the job with the lowest profit else if (minHeap . Count > 0 && minHeap . Peek () < job . p) { minHeap . Pop (); minHeap . Push (job . p); } } int totalJobs = minHeap . Count ; int totalProfit = 0 ; while (minHeap . Count > 0) { totalProfit += minHeap . Pop (); } return new List < int > { totalJobs , totalProfit }; } public static void Main () { int [] deadline = { 2 , 1 , 2 , 1 , 1 }; int [] profit = { 100 , 19 , 27 , 25 , 15 }; List < int > ans = jobSequencing (deadline , profit); Console . WriteLine (ans [0] + " " + ans [1]); } } JavaScript class MinHeap { constructor () { this . heap = []; } push (val) { this . heap . push (val); this . heapifyUp (); } pop () { if (this . heap . length === 1) return this . heap . pop (); let top = this . heap [0]; this . heap [0] = this . heap . pop (); this . heapifyDown (); return top ; } top () { return this . heap [0]; } size () { return this . heap . length ; } heapifyUp () { let idx = this . heap . length - 1 ; while (idx > 0) { let parent = Math . floor ((idx - 1) / 2); if (this . heap [parent] <= this . heap [idx]) break ; [this . heap [parent], this . heap [idx]] = [this . heap [idx], this . heap [parent]]; idx = parent ; } } heapifyDown () { let idx = 0 ; while (2 * idx + 1 < this . heap . length) { let left = 2 * idx + 1 , right = 2 * idx + 2 ; let smallest = left ; if (right < this . heap . length && this . heap [right] < this . heap [left]) smallest = right ; if (this . heap [idx] <= this . heap [smallest]) break ; [this . heap [idx], this . heap [smallest]] = [this . heap [smallest], this . heap [idx]]; idx = smallest ; } } } function jobSequencing (deadline , profit) { let n = deadline . length ; let ans = [0 , 0]; let jobs = []; for (let i = 0 ; i < n ; i ++) { jobs . push ([deadline [i], profit [i]]); } // sort the jobs based on deadline // in ascending order jobs . sort ((a , b) => a [0] - b [0]); let pq = new MinHeap (); for (let job of jobs) { // if job can be scheduled within its deadline if (job [0] > pq . size ()) { pq . push (job [1]); } // Replace the job with the lowest profit else if (pq . size () > 0 && pq . top () < job [1]) { pq . pop (); pq . push (job [1]); } } while (pq . size () > 0) { ans [1] += pq . pop (); ans [0] ++ ; } return ans ; } // Driver Code let deadline = [2 , 1 , 2 , 1 , 1]; let profit = [100 , 19 , 27 , 25 , 15]; let ans = jobSequencing (deadline , profit); console . log (ans [0] + " " + ans [1]); Output 2 127 [Alternate Approach] Using Disjoint Set - O(n * log(d)) Time and O(d) Space The job sequencing can also be done using disjoint set based on the maximum deadline of all the jobs. This approach is explained in article Job Sequencing - Using Disjoint Set. Comment Article Tags: Article Tags: Greedy Sorting Heap DSA Accolite + 1 More