Consider the problem of storing $n$ books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearranged. The $i$-th book $b_i$, where $1 \leq i \leq n$ has a thickness $t_i$ and height $h_i$ stored in arrays $t[1 \ldots n]$ and $h[1 \ldots n]$. The length of each bookshelf at this library is $L$. We want to minimize the sum of heights of the shelves needed to arrange these books.

(a) (5 points) Suppose all the books have the same height $h$ (i.e., $h = h_i$ for all $i$) and the shelves are each of height $h$, so any book fits on any shelf. The greedy algorithm would fill the first shelf with as many books as we can until we get the smallest $i$ such that $b_i$ does not fit, and then repeat with subsequent shelves. Show that the greedy algorithm always finds the shelf placement with the smallest total height of shelves, and analyze its time complexity.

Solution: ***************** INSERT YOUR SOLUTION HERE ************

(b) (6 points) Now assume that the books are not of the same height, and hence the height of any shelf is set to be the height of the largest book placed on that shelf. Show that the greedy algorithm in part (a) doesn’t work for this problem. Give an alternative dynamic programming algorithm to solve this problem. What is the running time of your algorithm?

Solution: ***************** INSERT YOUR SOLUTION HERE ************
You live in a country with very high medical costs (such as the US). You also anticipate undergoing a super-expensive medical treatment for the time period \([S, F]\). Luckily, as long as you have at least one job, you will have a cheap health insurance which will fully cover your expensive treatment. You are also very smart, so you get \(n\) job offers, where the \(i\)-th job is for the period \([s_i, f_i]\), so that the union of \([s_i, f_i]\) fully covers the desired interval \([S, F]\). For simplicity, assume the jobs are already sorted by their starting times, so that \(s_1 \leq s_2 \leq \ldots \leq s_n\).

Of course, one possible solution is to take all the jobs (it’s OK to have multiple jobs at the same time). However, being a naturally lazy person, you would like to take the smallest number of jobs which would cover the desired interval \([S, T]\).

(a) Design an efficient \(O(n)\) greedy algorithm for this problem. For partial credit, give \(O(n^2)\) algorithm. Do not forget to carefully argue the correctness of your algorithm.

**Solution:** ******************** INSERT YOUR SOLUTION HERE ******************  

(b) (Extra Credit). Assume you can hold at most two jobs at the same time (e.g., each job only has a day shift and a night shift, and you have to be at the job). Does your solution from part (a) still work in this case? If yes, prove it. Else, give a counter-example.

**Solution:** ******************** INSERT YOUR SOLUTION HERE ******************
Assume that you are given as input two arrays \( R[1 \ldots n] \) and \( C[1 \ldots n] \) containing integers from \( \{0, 1, \ldots, n\} \). You need to construct an \( n \times n \) matrix \( A \) that contains entries 0 or 1 such that for all \( i \), \( R[i] \) is the number of 1s in \( i \)-th row of \( A \), and \( C[i] \) is the number of 1s in \( i \)-th column of \( A \). If no such construction of \( A \) is possible, then your algorithm should output “Failure”. For example, if \( R[i] = C[i] = 1 \) for all \( i \), then one valid matrix \( A \) would be the identity matrix (but any so-called “permutation matrix” will also work). On the other hand, if \( \sum_i R[i] \neq \sum_j C[j] \), then no such matrix \( A \) can exist (as we must have \( \sum_i R[i] = \sum_{i,j} A[i, j] = \sum_j C[j] \)).

Consider the following greedy algorithm that does the following. It first checks whether \( \sum_i R[i] = \sum_j C[j] \), and if not, then it outputs “Failure”. Otherwise, it constructs \( A \) one row at a time. Assume inductively that the first \( i - 1 \) rows of \( A \) have been constructed. Let \( \text{Curr}[j] \) be the number of 1s in the \( j \)-th column in the first \( i - 1 \) rows of \( A \). Now, sort the values \( B[j] = C[j] - \text{Curr}[j] \), and consider \( R[i] \) columns \( j \) with \( R[i] \) largest values \( B[j] \). If \( B[j] = 0 \) for any of these \( R[i] \) columns, the algorithm outputs “Failure”. Otherwise, these \( R[i] \) “largest columns” are assigned 1s in row \( i \) of \( A \), and the rest of the columns are assigned 0s. That is, the columns that still need the most 1s are given 1s.

For example, if \( R[i] = C[i] = i \) for all \( i \), then the greedy algorithm will output the unique matrix \( A \) where \( A[i, j] = 1 \) iff \( i + j \geq n + 1 \) (when, as you can check, is the only feasible solution).

Formally prove that this algorithm is correct using the Local Swap argument.

Solution: ****************** INSERT YOUR SOLUTION HERE ******************
You operate a cable company and get \( n \) requests to install cable from customers, whom we simply call 1, \ldots, \( n \). Each customer ordered a different package, so that customer \( i \) pays price \( p[i] \) per each day of service. Unfortunately, you can only do one installation per day, so you must choose some order according to which you will do the \( n \) installations. Namely, you must choose a permutation \( \pi \) from \( \{1 \ldots n\} \) to \( \{1 \ldots n\} \), so that the first day you go to customer \( \pi(1) \), the second — to \( \pi(2) \), etc., until you finish at customer \( \pi(n) \) at day number \( n \). Since after day number \( n \) you get consistent revenue \( p_1 + \ldots + p_n \) per day irrespective of the order of installations, your objective is to find the order \( \pi \) maximizing the revenue during the first \( n \) days, which equals to

\[
Revenue(\pi) = p[\pi(1)] \cdot n + p[\pi(2)] \cdot (n - 1) + \ldots + p[\pi(n)] \cdot 1
\]

Design a greedy algorithm for this problem. Use the Greedy Stays Ahead principle to argue that your solution is correct. Namely, starting with \( i = 1 \) and using induction, show that that for every \( 1 \leq i \leq n \), the revenue of greedy (that you designed) for the first \( i \) days is at least as high as a revenue of any other algorithm for the first \( i \) days.

(Hint: Split the revenue for the first \( i \) days as the revenue for day 1 plus that for day 2, \ldots plus that for day \( i \). Argue that greedy gives a “pretty good” revenue for every specific day.)

Solution: **************** INSERT YOUR SOLUTION HERE **************