

1. Textbook problem 6.2

Suppose you're managing a consulting team of expert computer hackers, and each week you have to choose a job for them to undertake. As you can well imagine, the set of possible jobs is divided into those that are *low-stress* (e.g. setting up a Web site for a class at the local elementary school) and those that are *high-stress* (e.g., protecting the nation's most valuable secrets.) The basic question each week is whether to take on a low-stress job or a high-stress job.

If you select a low-stress job for your team in week i , then you get a revenue of $l_i > 0$ dollars; if you select a high-stress job, you get a revenue of $h_i > 0$ dollars. High-stress jobs typically pay more. The catch, however, is that in order for the team to take on a high-stress job in week i , it's required that they do no job (of either type) in week $i - 1$; they need a full week of prep time to get ready for the crushing stress level. On the other hand, it's okay for them to take a low-stress job in week i even if they have done a job (of either type) in week $i - 1$.

So, given a sequence of n weeks, a *plan* is specified by a choice of "low-stress", "high-stress" or "none" for each of the n weeks, with the property that if "high-stress" is chosen for week $i > 1$, then "none" has to be chosen for week $i - 1$. (It's okay to choose a high-stress job in week 1.) The *value* of the plan is determined in the natural way; for each i , you add l_i to the value if you choose "low-stress" in week i , and you add h_i to the value if you choose "high-stress" in week i . (You add 0 if you choose "none" in week i .)

The problem. Given sets of values l_1, l_2, \dots, l_n and h_1, h_2, \dots, h_n , find a plan of maximum value (such a plan is called *optimal*.) Give an efficient algorithm to take the input values and return the value of the optimal plan.

2. Textbook problem 6.8

The residents of an underground city defend themselves through a combination of kung fu, heavy artillery, and efficient algorithms. Recently they have become interested in automated methods that can help fend off attacks by swarms of robots.

Here's what one of those robot attacks looks like:

- A swarm of robots arrives over the course of n seconds; in the i^{th} second, x_i robots arrive. Based on remote sensing data, you know this sequence x_1, x_2, \dots, x_n in advance.
- You have at your disposal an *electromagnetic pulse* (EMP) which can destroy some of the robots as they arrive; the EMP's power depends on how long it's been allowed to charge up. To make this precise, there is a function $f(\cdot)$ so that if j seconds have passed since the EMP was last used, then it is capable of destroying up to $f(j)$ robots.
- So specifically, if it is used in the k^{th} second, and it has been j seconds since it was previously used, then it will destroy $\min(x_k, f(j))$ robots. After this use, it will be com-

pletely drained, and will have to recharge for some time before you can use it again.

- Assume that the EMP starts off completely drained, so if it is used for the first time in the j^{th} second, then it is capable of destroying $f(j)$ robots.

Given the data on robot arrivals x_1, x_2, \dots, x_n , and given the recharging function $f(\cdot)$, choose the points in time at which you're going to activate the EMP so as to destroy as many robots as possible (equivalently, leaving as few robots as possible to be destroyed by other means).

a) Show that the following algorithm does not correctly solve this problem.

```

Schedule-EMP( $x_1, x_2, \dots, x_n$ )
  Let  $j$  be the smallest number for which  $f(j) \geq x_n$ 
  (If no such  $j$  exists, then set  $j = n$ )
  Activate the EMP in the  $n^{\text{th}}$  second
  If  $n - j \geq 1$ , then
    Recursively call Schedule-EMP( $x_1, x_2, \dots, x_{n-j}$ )
    
```

b) Give an efficient dynamic programming algorithm that takes the data on robot arrivals and the recharging function, and returns the maximum number of robots that can be destroyed by a sequence of EMP activations.

3. Prove by using an adversary argument that any algorithm that finds both the minimum and maximum of a set of distinct numbers requires $\lceil 3n/2 \rceil - 2$ comparisons.

Hint: Assume that initially, every element is marked with a '+' and '-', indicating that it could be either the maximum or the minimum. The adversary answers queries until at most one element has a '+' and one element has a '-', indicating that these elements are the maximum and minimum respectively.

4. Given a sequence of n elements of n/k blocks (k elements per block), where it is inter-block sorted (that is, if $i < j$, then for any element x in block i and any element y in block j , we have $x < y$). Show that, based on comparisons only, you can not have the whole sequence sorted using less than $\Omega(n \log k)$ comparisons.

Note that combining the lower bounds for each block is not adequate.

5. You have n objects that you wish to put in order using the relations " $<$ " and " $=$ ". For example, with three objects, 13 different orderings are possible. $a = b = c, a = b < c, a < b = c, a < b < c, a < c < b, a = c < b, b < a = c, b < a < c, b < c < a, b = c < a, c < a = b, c < a < b, c < b < a$. Give a dynamic programming algorithm that can calculate, as a function of n , the number of different possible orderings.

Hint: consider how many distinct values do you have in your ordering. For example $a = d < b = c$ has only two distinct values. Now consider how this number of distinct values changes when you insert e .