COT5405: Homework 3 (Spring 2016)

Instructions: study Solved Exercises 1 and 2 on pages 307 – 311. Complete the following exercises by defining the recurrence formulae, designing DP algorithms, and analyzing their running time and space requirements. The first two's analyses have been done for you.

1. Exercise 8(b) on pages 319-320.

(b) Let $OPT(j)$ be the maximum number of robots that can be destroyed for the instance of the problem just on $x_1, \ldots, x_j$. Clearly if the input ends at x_j, there is no reason not to activate the EMP then (you're not saving it for anything), so the choice is just when to last activate it before step j. Thus $OPT(j)$ is the best of these choices over all i:

$$OPT(j) = \max_{0 \le i < j} [OPT(i) + \min(x_j, f(j-i))],$$

where $OPT(0) = 0$. The full algorithm is just

```
Set $OPT(0) = 0$
For $i = 1, 2, \ldots, n$
  Compute $OPT(j)$ using the recurrence
Endfor
Return $OPT(n)$.
```

The running time is $O(n)$ per iteration, for a total of $O(n^2)$.

An alternate solution would define $OPT'(j,k)$ to be the best solution for steps j through n, given that the EMP in step j has already been charging for k steps. The optimal way to solve this sub-problem would be to either activate the EMP in step j or not, and $OPT'(j,k)$ is just the better of these two choices:

$$OPT'(j,k) = \max(\min(x_j, f(k)) + OPT'(j+1, 1), OPT'(j+1, k+1)).$$

We initialize $OPT'(n,k) = \min(x_n, f(k))$ for all k, and the full algorithm is

```
Set $OPT'(n,k) = \min(x_n, f(k))$ for all $k$.
For $j = n-1, n-2, \ldots, 1$
  For $k = 1, 2, \ldots, j$
    Compute $OPT'(j,k)$ using the recurrence
  Endfor
Endfor
Return $OPT'(1,1)$.
```

The running time is $O(1)$ per entry of OPT', for a total of $O(n^2)$.

2. Exercise 27 on pages 333-334.

Hint: A solution is specified by the days on which orders of gas arrive, and the amount of gas that arrives on those days. In computing the total cost, we will take into account delivery and storage costs, but we can ignore the cost for buying the actual gas, since this is the same in all solutions. (At least all those where all the gas is exactly used up.)

Consider an optimal solution. It must have an order arrive on day 1, and it the next order is due to arrive on day i, then the amount ordered should be $\sum_{j=1}^{i-1} g_j$. Moreover, the capacity requirements on the storage tank say that i must be chosen so that $\sum_{j=1}^{i-1} g_j \leq L$.

What is the cost of the storing this first order of gas? We pay g_2 to store the g_2 gallons for one day until day 2, and $2g_3$ to store the g_3 gallons for two days until day 3, and so forth, for a total of $\sum_{j=1}^{i-1}(j-1)g_j$. More generally, the cost to store an order of gas that arrives on day a and lasts through day b is $\sum_{j=a}^{b}(j-a)g_j$. Let us denote this quantity by S(a,b).

Let OPT(a) denote the optimal solution for days a through n, assuming that the tank is empty at the start of day a, and an order is arriving. We choose the next day b on which an order arrives: we pay P for the delivery, S(a,b-1) for the storage, and then we can behave optimally from day b onward. Thus we know how to compute OPT(a):

$$OPT(a) = P + \min_{b>a:\sum_{j=a}^{b-1} g_j \leq L} S(a, b-1).$$

The values of OPT can be built up in order of decreasing a, in time $O(n-a)$ for iteration a, leading to a total running time of $O(n^2)$. The value we want is $OPT(1)$, and the best ordering strategy can be found by tracing back through the array of OPT values.

3. Exercise 2(a) & (b) on pages 313-314.

(a) This algorithm is too short-sighted; it might take a high-stress job too early and an even better one later.

	Week 1	Week 2	Week 3
ℓ	2	2	2
h	1	5	10

The algorithm in (a) would take a high-stress job in week 2, when the unique optimal solution would take a low-stress job in week 1, nothing in week 2, and then a high-stress job in week 3.

(b) Let $OPT(i)$ denote the maximum value revenue achievable in the input instance restricted to weeks 1 through i. The optimal solution for the input instance restricted to weeks 1 through i will select *some* job in week i, since it's not worth skipping all jobs — there are no future high-stress jobs to prepare for. If it selects a low-stress job, it can behave optimally up to week $i-1$, followed by this job, while if it selects a high-stress job, it can behave optimally up to week $i-2$, followed by this job. Thus we have justified the following recurrence.

$$OPT(i) = \max(\ell_i + OPT(i-1), h_i + OPT(i-2)).$$

We can compute all OPT values by invoking this recurrence for $i = 1, 2, \ldots, n$, with the initialization $OPT(1) = \max(\ell_1, h_1)$. This takes constant time for each value of i, for a total time of $O(n)$. As usual, the actual sequence of jobs can be reconstructed by tracing back through the set of OPT values.

An alternate, but essentially equivalent, solution is as follows. We define the following sub-problems. Let $L(i)$ be the maximum revenue achievable in weeks 1 through i, given that you select a low-stress job in week i, and let $H(i)$ be the maximum revenue achievable in weeks 1 through i, given that you select a high-stress job in week i.

Again, the optimal solution for the input instance restricted to weeks 1 through i will select some job in week i. Now, if it selects a low-stress job in week i, it can select anything it wants in week $i-1$; and if it selects a high-stress job in week i, it has to sit out week $i-1$ but can select anything it wants in week $i-2$. Thus we have

$$L(i) = \ell_i + \max(L(i-1), H(i-1)),$$

$$H(i) = h_i + \max(L(i-2), H(i-2)).$$

The L and H values can be built up by invoking these recurrences for $i = 1, 2, \ldots, n$, with the initializations $L(1) = \ell_1$ and $H_1 = h_1$.