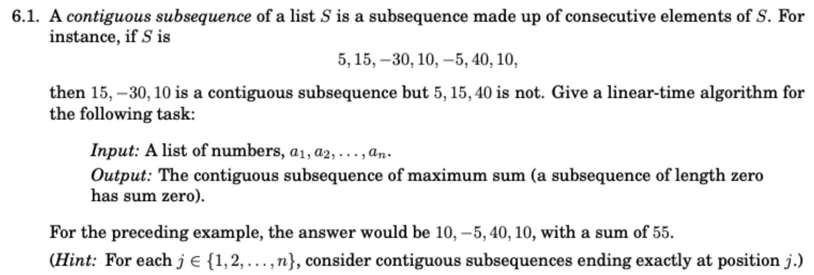


6.1. A *contiguous subsequence* of a list S is a subsequence made up of consecutive elements of S. For instance, if S is

$$5, 15, -30, 10, -5, 40, 10,$$

then $15, -30, 10$ is a contiguous subsequence but $5, 15, 40$ is not. Give a linear-time algorithm for the following task:

Input: A list of numbers, $a_1, a_2, \ldots, a_n$.
Output: The contiguous subsequence of maximum sum (a subsequence of length zero has sum zero).

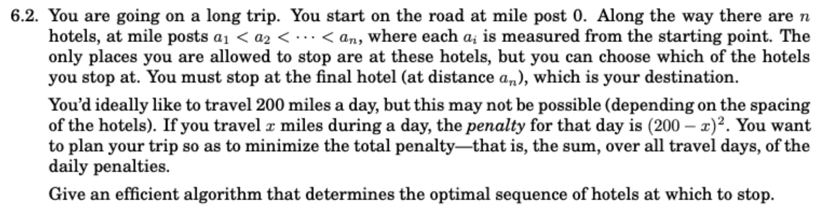
For the preceding example, the answer would be $10, -5, 40, 10$, with a sum of 55.
(*Hint:* For each $j \in \{1, 2, \ldots, n\}$, consider contiguous subsequences ending exactly at position j.)

Solution:

Let s[j] be the sum of the maximum contiguous sequence that ends at position j. Then for s[j+1] we can either start a new contiguous sequence with score a_{j+1} or continue the contiguous sequence with score s[j] + a_{j+1}. The recurrence is:

$$s[j] = \begin{cases} a_j, & \text{if } j = 0 \\ \max\{s[j-1] + a_j, a_j\} & \text{if } j > 0 \end{cases}$$

We can implement this recurrence in linear time by computing s[0],s[1],...,s[n]. To recover the sequence, we first scan through s[] to find the location s[j] which is maximum. We scan backwards from j to find the negative element s[i] at the beginning of the maximum contiguous sequence, or have i = −1. Then the maximum contiguous sequence is a_{i+1}, a_{i+2}, . . . A_j with score s[j]. The backtracking takes linear time as well.



6.2. You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

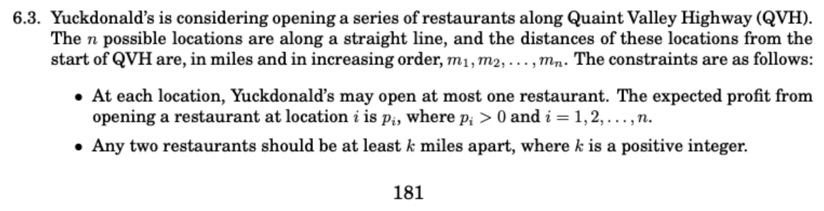
You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties.

Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

Solution:
Let p[j] be the minimum penalty required to travel to hotel j. To get the minimum penalty required to travel to hotel j + 1, we could drive from any of the previous hotels i to j and have a penalty of p[i] + (200 − (a_j − a_i))^2. The recurrence is:

$$p[j] = \begin{cases} a_j, & \text{if } j = 1 \\ \min_{1 \leq i < j}\{p[i] + (200 - (a_j - a_i))^2\}, & \text{if } j > 0 \end{cases}$$

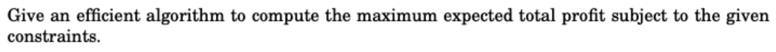
We can implement this recurrence in O(n2) time by computing p[0], p[1], . . . , p[n]. To determine the hotels that should be stopped at, we store the hotel i that we should travel from to reach each j that got the penalty p[j]. Starting with n, we can follow these hotel links backwards to get the sequence of hotels. The backtracking takes linear time, so this takes O(n^2) time in total.



6.3. Yuckdonald's is considering opening a series of restaurants along Quaint Valley Highway (QVH). The n possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order, $m_1, m_2, \ldots, m_n$. The constraints are as follows:

- At each location, Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
- Any two restaurants should be at least k miles apart, where k is a positive integer.

181



Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints.

```
procedure expected_profit(N,P)
Input: N locations; P[1,..,N] where P[i] denotes profit at location i
Output: Maximum expected profit $P_{max}$
Declare an array of maximum Expected profit Profit[1..N]:
    Profit[i] denotes maximum expected profit at location i
for i = 1 to N:
    Profit[i] = 0
for i = 2 to N
    for j = 1 to i-1
        temp = Profit[j] + $\alpha(m_i,m_j) \cdot$ P[i]
        if temp > Profit[i]:
            temp = Profit[i]
        if Profit[i] < P[i]:
            Profit[i] = P[i]
```

Time complexity: There are two for loops which gives a O(n2) complexity to this algorithm.