Recall that a dynamic programming algorithm to solve a given problem $P$ involves the following elements:

(a) A definition of the subproblems to be solved (and from whose solution we will compute the solution to $P$ — see (c) below).
(b) A recursive formula to compute the solution to each subproblem from the solutions to smaller subproblems. This induces a partial order on the subproblems defined in (a).
(c) A way to compute the solution to $P$ from the solutions to the subproblems computed in (b).

Proving the correctness of a dynamic programming algorithm amounts to justifying (i) why the recursive formula in step (b) correctly computes the subproblems defined in step (a), and (ii) why the computation in (c) yields a solution to the given problem. Part (ii) is often immediate from the definition of the subproblems.

**Question 1.** (20 marks) Let $A[1..n]$ be a non-empty array of integers (they may be positive, negative, or zero) and $k$ be a positive integer. We want to find the maximum sum of a (contiguous) subarray of $A$ of length at least $k$. That is, we want to find $\max\left\{ \sum_{i=1}^{j} A[i] : 1 \leq i \leq j \leq n \text{ and } j - i + 1 \geq k \right\}$. 
Use dynamic programming to give an $O(n)$ algorithm that solves this problem. Describe your algorithm in pseudocode, explain why it is correct, and analyze its running time.

**Hint:** You may find it useful to first solve the special case $k = 1$ (i.e., when there is no length constraint on the subarray), and then generalize. Do not submit your answer to the special case. For the general case you may find it useful to precompute the sum of every length $k$ subarray of $A$.

**Question 2.** (20 marks) Recall that a **substring** of a string $x$ is a (possibly empty) string $x'$ such that, for some (possibly empty) strings $y$ and $z$, $x = yx'z$. A string is a **palindrome** if it reads the same forward and backward — for example, the inscription on the fountain in the courtyard of Agia Sophia: $\nu\iota\pi\sigma\alpha\nu\alpha\omicron\nu\mu\alpha\omicron\tau\alpha\mu\omicron\alpha\nu\omicron\nu$, which (with inter-word spaces omitted) means “wash the sins, not just the face”, is a palindrome.

Give an efficient dynamic programming algorithm that takes a string $x[1..n]$ as input and returns the **number** of substrings of $x$ that are palindromes. Justify the correctness of your algorithm and analyze its running time.

**Question 3.** (25 marks) (DPV Exercise 6.2, slightly rephrased.) You are going on a long trip. Along the way, there are $n$ hotels, at distances $d_1 < d_2 < \ldots < d_n$ from your starting point. You are only allowed to stop at these hotels, but you can choose at which ones to stop — except that you **must** stop at the last hotel, your destination.

Ideally, you want to travel 300 km between successive stops, but this is not always possible because the spacing of the hotels may not allow it. If you travel a distance of $x$ km between successive stops, you pay a penalty of $(300 - x)^2$ for that interval of travel. For a given sequence of stops made during the trip, the **total penalty** for that sequence, is the sum of the penalties for the between-stop intervals. (We think of the starting point as a “stop”.)

You wish to determine an optimal sequence of stops, i.e., one that minimizes the total penalty.

**a.** Consider the following greedy strategy, applied after each stop. Among all hotels you have not already passed or stopped at, find one that is closest to 300 km from your present location; that hotel will be your next stop. Give a simple example that demonstrates that this greedy algorithm does not yield an optimal sequence of stops.

**b.** Give a polynomial-time dynamic programming algorithm that, given the sorted sequence $d_1, d_2, \ldots, d_n$, returns a pair $(S, p)$, where $S = s_1, \ldots, s_k$ is an optimal sequence of stops in increasing order, and $p$ is the associated total penalty. Justify the correctness of your algorithm and analyze its running time.