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

(a) A definition of a polynomial number of subproblems that will be solved (and from whose solution we will compute the solution to \( P \) — see (1) 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 (1).

(c) A way to compute the solution to \( P \) from the solutions to the subproblems computed in (1).

Proving the correctness of a dynamic programming algorithm amounts to justifying (i) why the recursive formula in step (1) correctly computes the subproblems defined in step (a), and (ii) why the computation in (2) yields a solution to the given problem. Part (ii) is often immediate from the definition of the subproblems.

**Question 1.** (10 marks) Let \( S[1..n] \) be a string that may have been produced when all punctuation marks and spaces disappeared from an English text, and all letters became lower-case. (So, it might be
“many years later as he faced the firing squad . . . ”) Your programming language has access to a Boolean function \( \text{Dict}(w) \) which, given a string \( w \) returns true if and only if \( w \) is a valid English word.

a. (7 marks) Give a dynamic programming algorithm that determines whether \( S[1..n] \) can be reconstructed as a sequence of valid English words. Briefly explain why your algorithm is correct. Your algorithm’s complexity should be \( O(n^2) \), assuming each call to \( \text{Dict} \) takes \( O(1) \) time.

b. (3 marks) If \( S \) can be reconstructed as a sequence of valid English words, make your algorithm print that sequence of words, one per line.

**Question 2.** (10 marks) A certain string-processing language offers a primitive operation \( \text{Split}(S, i, S_1, S_2) \) which, given a string \( S \) of length \( n \) and a positive integer \( i \leq n \), copies \( S[1..i] \) into \( S_1 \) and \( S[i+1..n] \) into \( S_2 \). Because \( \text{Split} \) involves copying, it takes \( n \) units of time to apply on a string of length \( n \) regardless of the position \( i \) of the cut.

If a string is to be cut into several pieces, the order of the cuts matters. For example, if we want to cut a 20-character string at positions 3 and 10, then making the first cut at position 3 incurs a total cost of 37, while making the first cut at position 10 incurs a total cost of 30.

a. Give a polynomial-time dynamic programming algorithm which, given the positions of \( m \) cuts \( c_1 < c_2 < \ldots < c_m \), in a string of length \( n \), finds the minimum cost of cutting the string into the specified \( m+1 \) pieces. (The \( c_j \)'s, being positions of a string of length \( n \), are positive integers between 1 and \( n \).) Explain why your algorithm is correct, and analyze its running time.

b. Make your algorithm also print the order in which the cuts should be applied so as to incur the minimum cost.

**Question 3.** (10 marks) Consider the following problem. The input is a set \( A \) of \( n \) non-negative integers (given in an array \( A[1..n] \)) and a non-negative integer \( k \). The output is \textbf{true} if we can partition \( A \) into two subsets whose sums differ by \( k \), and \textbf{false} otherwise. More precisely, the output is \textbf{true} if there is some \( B \subseteq A \) such that

\[
\sum_{x \in B} x - \sum_{y \in A-B} y = k
\]

i.e., the difference between the sum of the integers in \( B \) and the sum of the integers in \( A - B \) is exactly \( k \); and the output is \textbf{false} if no such subset of \( A \) exists. (The sum of “the integers” in the empty set is, by definition, zero.)

Give a pseudopolynomial dynamic programming algorithm for this problem. Explain why your algorithm is correct, and analyze its running time. Why is your algorithm \textit{pseudo}-polynomial time and not polynomial time?