1. Do the following for each problem:

   (i) Define the subproblems to be solved in English.

   (ii) Define an appropriate recurrence for the subproblems, and use an inductive argument to prove the recurrence correct.

   (iii) Implement the dynamic programming algorithm in pseudocode using iterative, bottom-up methods (not recursion or memoization).

   (iv) Analyze the worst-case time and memory performance of the algorithm in terms of n, where n is the *length* of the input. (Remember that integers are encoded in binary, so the encoded length of the integer x is $\Theta(\log x)$.)

   The problems start here:

   (a) Let $1 = x_1 < x_2 < \cdots < x_n$ be n coin denominations. Give an algorithm to efficiently compute the minimum number of coins needed to make change for an integer input amount $v \geq 1$. (Note that "efficient" here means "polynomial in terms of both n and $\log v$.")

   (b) Let $G = (V, E)$ be a weighted undirected graph with n vertices. Assume that the weights of the edges in E are distinct. For a path P in G, define the cost of P to be the maximum of the costs of the edges on P. Formulate a dynamic programming algorithm to compute a lowest-cost path between two given vertices in G.

   [Hint: Label the vertices 1 to n. Let $C[i][j][k]$ denote the cost of a lowest-cost path from vertex i to vertex j whose intermediate vertices are in the set $\{1, ..., k\}$; and define it to be ∞ if no such path exists.]

   (c) Consider an n-character binary string $x = x_0 \cdots x_{n-1}$. A *subsequence* of x is a string of the form $x_{i_1} x_{i_2} \cdots x_{i_k}$, where $0 \leq i_1 < \cdots < i_k \leq n-1$. A string is *palindromic* if it is equal to its own reverse (that is, the string is the same whether read backwards or forwards). Compute the length of the longest palindromic subsequence of x.

   (d) In an undirected graph, an *independent set* is a set of vertices no two of which are adjacent. Given a tree T, compute the size of a maximum independent set in T.

   [Hint: Assume that the tree is rooted at a vertex r. For each vertex v, define $I[v]$ to be the size of a maximum independent set in the subgraph consisting of v and its descendents (its children, children's children, etc.).]

2. We use the *optimal-substructure property* in deriving dynamic programming algorithms:

the optimal solution to the overall problem consists of optimal solutions to its subproblems.

For example, consider the shortest path problem: Given a directed graph $G = (V, E)$ with positive edge-weights and two nodes $s, t \in V$, find a shortest directed path from s to t. If P is a shortest path from node s to node t, then for every pair of nodes u, v on P, the subpath from u to v on P is also a shortest path from u to v.

(a) Show that the optimal-substructure property does not hold for the following problem: Given a directed graph $G = (V, E)$ and two nodes $s, t \in V$, find a longest simple directed path from s to t. (This problem is discussed in CLRS Chapter 15, pp. 381–384. Try working it yourself before looking up the solution.)

(b) (CLRS 15.3-3) Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the sequence of matrices so as to maximize rather than minimize the number of scalar multiplications. Does this problem exhibit optimal substructure?

(c) (CLRS 15.3-5) Suppose that in the rod-cutting problem of Section 15.1 we also had limit ℓ_i (for $i = 1, 2, ..., n$) on the number of pieces of length i that we are allowed to produce. Show that the optimal-substructure property described in Section 15.1 no longer holds.