1. (6 + 6 points)
   (a) Use the bottom-up (i.e., iterative) algorithm \textsc{Matrix-Chain-Order}(p) seen in class to determine the minimum number of multiplications needed to compute the product of a sequence of six matrices, whose dimensions are \( p = \langle p_0, p_1, \ldots, p_6 \rangle = \langle 10, 5, 15, 30, 1, 100, 20 \rangle \). You must show your work, i.e., the filled-in lookup table, the optimal parenthesization, and its cost.
   (b) Ex. 15.2-5, p. 378.

2. (12 points) Give a top-down, memoized version of the algorithm \textsc{LCS-Length}(X, Y) to compute, in \( O(mn) \) time, the length of a longest common subsequence of strings \( X \) and \( Y \), where \( m = |X| \) and \( n = |Y| \). (You do not have to retrieve the LCS itself; just compute its length.) Give a careful analysis of the running time.

3. (10 points) Ex. 25.2-4, p. 699. Justify your answer carefully.

4. (12 points) Let \( A = a_1a_2\ldots a_n \) be a string of distinct integers, not necessarily in sorted order. A subsequence, \( B \), of \( A \) is an increasing subsequence iff the integers of \( B \) are in increasing order. (For instance, 4,7,10 is an increasing subsequence of 1,4,2,7,5,9,10,8; however, 4,9,8 is not.) The goal is to find a longest increasing subsequence (LIS) of \( A \). In the example above, one possible LIS is 1, 4, 7, 9, 10, which has length 5.
   Give a bottom-up dynamic programming algorithm to find an LIS of \( A \) in \( \Theta(n^2) \) time.
   Your answer should include (i) a brief description of the main ideas, including the dynamic programming recurrence and its justification, (ii) pseudocode, and (iii) an analysis of the running time.
   \textit{Note:} There is a simple approach that uses the LCS algorithm as a subroutine: Sort \( A \) into a new string \( A' \) and find the LCS of \( A \) and \( A' \). Convince yourself that this works but do not use this method; instead, solve the problem from first principles.
   \textit{Hint:} For \( 1 \leq i \leq n \), let \( \ell_i \) be the length of an LIS of \( A \) which terminates at \( a_i \).

5. (15 points) Consider the problem of transforming a string, \( A = a_1a_2\ldots a_m \), of characters into another string, \( B = b_1b_2\ldots b_n \), by a sequence of insert (I), delete (D), and substitute (S) operations. For instance, the string \textit{unix} can be transformed to \textit{mutiny}, as follows: (In what follows, \textit{M} denotes the non-operation of simply matching two identical characters.)

\begin{verbatim}
IMDIMIS <- operations
A: un i x
B: mu tiny
\end{verbatim}

That is, we insert an \( m \), match the two \( u \)'s, delete the \( n \), insert a \( t \), match the two \( i \)'s, insert an \( n \), and substitute \( x \) with \( y \).

Alternatively, we could have accomplished the transformation as follows:
SIIDMIID <- operations
A: u ni x
B: mut iny

Note that in the first case the number of I, D, and S operations is 5, while in the second case it is 7. Our goal is to come up with a transformation that minimizes the total number of I, D, and S operations. (M’s are not counted.)

Define the edit distance between two strings as the minimum total number of I, D, and S operations needed to transform the first string into the second. Design a bottom-up dynamic programming algorithm to compute the edit distance between A and B in time $\Theta(mn)$. The output is the edit distance and the corresponding operations.

Your answer should include (i) a brief description of the main ideas, including the recurrence equation and its justification, (ii) pseudocode, and (iii) an analysis of the running time.

Hint: Let $e(i, j)$ be the edit distance between $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$.

6. (15 points) Let $P$ be a convex polygon in the plane, with vertices $v_1, v_2, \ldots, v_n$ ($n \geq 3$) numbered clockwise around $P$. A triangulation of $P$ is a partition of its interior into non-overlapping triangles, where the vertices of the triangles are vertices of $P$. (See figure.) The weight of a triangulation of $P$ is the sum of lengths of the perimeters of its triangles. For example, the triangulation in the figure has weight $(|v_1v_2| + |v_2v_3| + |v_3v_1|) + (|v_1v_3| + |v_3v_5| + |v_5v_1|) + (|v_3v_4| + |v_4v_5| + |v_5v_3|)$. (Here $|v_1v_2|$ denotes the length of edge $v_1v_2$.)

There are exponentially-many different triangulations of $P$ and different ones can have different weights. Give a bottom-up dynamic programming algorithm to compute a minimum-weight triangulation of $P$ in $\Theta(n^3)$ time. The output should be the weight of the optimal triangulation and sufficient information to be able to construct it. (You do not have to write the routine to construct it though.) Assume the availability of a routine $\text{Peri}(i, j, k)$ which returns the perimeter of triangle $v_i v_j v_k$ in constant time.

Your answer should include (i) a brief description of the main ideas, including the recurrence equation and its justification, (ii) pseudocode, and (iii) an analysis of the running time.

Hint: In any triangulation of $P$, edge $v_1v_n$ belongs to some triangle $v_1v_nv_k$. Use this triangle to split the problem into subproblems. Let $c(i, j)$ be the weight of a minimum-weight triangulation of a convex polygon with vertices $v_i, \ldots, v_j$, $1 \leq i < j \leq n$. 