Question 1

You are going on a canoe trip downstream a river. Along the way there are \( n \) wilderness outfitters 1, \(...\), \( n \). Your trip starts at outfitter 1 and ends at outfitter \( n \). In each outfitter \( i \), \( 1 \leq i < n \), you can rent a canoe to go to any outfitter \( j > i \), at a cost of \( c_{ij} \). If you rent a canoe from \( i \) to \( j \), you must leave this canoe at \( j \) and rent another canoe to continue your trip.

The costs \( c_{ij} \) are completely arbitrary. E.g., it may be that \( c_{ij} < c_{ik} \) even though \( j > k \), and it may be that \( c_{ij} + c_{jk} \) is much smaller or much greater than \( c_{ik} \).
We want to solve the following problem: Given the costs $c_{i,j}, 1 \leq i < j \leq n$, find the minimum total cost of renting canoes to complete the trip from outfitter 1 to outfitter $n$.

**Question 1a.** Does the following greedy strategy work?

Rent the cheapest canoe from your present outfitter. Take the canoe to the destination of the rental, and continue recursively from there until you have reached outfitter $n$. 
Question 1 (cont’d)

**Question 1b.** Give a polynomial time dynamic programming algorithm to solve this problem.

- Define the subproblems of your DP algorithm.
- Give a recursive formula to compute the subproblems.
- Describe your DP algorithm in pseudocode.
- Analyze the running time of your algorithm.
In the optimal alignment problem we used three types of “edits” to transform one string to another: insert a character, delete a character, and change a character.

Introduce a new kind of edit: Transpose two adjacent characters. With a transposition we can transform the string \( xaby \) to the string \( xbay \), where \( a, b \) are characters and \( x, y \) are (possibly empty) strings in one “edit”.

Now define the edit distance between strings \( x \) and \( y \) to be the minimum number of single character insertions, deletions, and changes, or two-character transpositions to transform \( x \) to \( y \).
Question 2 (cont’d)

Question 2a. Modify the recursive formula to compute the edit distance $ED(i, j)$ between $x[1..i]$ and $y[1..j]$ under this new definition.

Question 2b. (Do at home.) Modify the algorithm we saw in class to compute the edit distance between two strings under this new definition.
Question 3

Recall that the product of a $p \times q$ matrix $A$ by a $q \times r$ matrix $B$ is a $p \times r$ matrix $C$.

To compute each entry of $C$, we take the dot-product of a row of $A$ and a column of $B$, each of which is a vector of $q$ entries. This requires $q$ multiplications of numbers. So, to compute $A \times B$ requires $p \times q \times r$ multiplications of numbers (for each of the $p \times r$ entries).
Since matrix multiplication is associative (but not commutative!), there are many different orders to evaluate the product of $n$ matrices

$$A_1 \times A_2 \times \ldots \times A_n$$

namely, as many as there are binary trees with $n - 1$ nodes (one per multiplication). The number of binary trees with $k$ nodes is the $k$-th Catalan number, and is $\text{Choose}(2k, k)/k + 1$, where $\text{Choose}(p, q)$ is the number of ways of choosing $q$ out of $p$ elements.

These evaluation orders may have different costs.
Question 3 (cont’d)

For example, suppose we want to evaluate $A_1 \times A_2 \times A_3$, where $A_1$ is a 10 x 100 matrix, $A_2$ is a 100 x 5 matrix, and $A_3$ is a 5 x 50 matrix.

How many multiplications (of numbers) are required to evaluate the matrix product in the order $(A_1 \times A_2) \times A_3$?

How many multiplications (of numbers) are required to evaluate the matrix product in the order $A_1 \times (A_2 \times A_3)$?
Each order of evaluating the product $A_1 \times A_2 \times \ldots \times A_n$ corresponds to a full binary tree, whose leaves contain $A_1, A_2, \ldots, A_n$ in left-to-right order, and the root corresponds to the last multiplication performed.

Each subtree has recursive structure: It corresponds to the evaluation of $A_i \times \ldots \times A_j$ in some order, for some $i \leq j$, with its root being the last of the $j - i$ matrix multiplications performed.
Question 3 (cont’d)

Give a dynamic programming algorithm to find the minimum number of multiplications to compute the product $A_1 \times \ldots \times A_n$, given the dimensions $m_1, \ldots, m_{n+1}$ of the matrices ($A_i$ is an $m_i \times m_{i+1}$ matrix).

- Define the subproblems of your DP algorithm.
- Give a recursive formula to compute the subproblems.
- Describe your DP algorithm in pseudocode.
- Analyze the running time of your algorithm.
- (Do at home.) Modify the algorithm to find the evaluation order that yields the minimum number of multiplications.