## Problem 1 (Rod Cutting without Repetition, 40 pts)

Recall that in the (vanilla) *rod cutting problem* we are given a rod of length $n$ and a list of $n$ prices (assume all prices are positive), where $p_i$ is the price fetched by a piece of length $i$. The goal is to calculate the maximum revenue you can generate from cutting the rod.

In the *rod cutting without repetition* problem, your discerning customers desire to be unique, and hence refuse to buy a length of rod that has been bought before (weird assumption, I know. But let's say this customer is an enthusiastic rod collector, and aims to collect rods of different lengths ¯_(ツ)_/¯). Namely, even if you cut two pieces of length $i$, the total contribution to revenue from these pieces is just $p_i$ (not $2p_i$ as in the vanilla version).

In this problem, you will design a dynamic programming algorithm for solving the rod cutting without repetition in polynomial time.

(a) Start by writing a recurrence relation for this problem (make sure to state clearly what the base cases are) and justifying its correctness (i.e. why does this problem exhibit optimal substructure). In class, our recurrence relation had just one parameter, the length of the rod. Here, you can introduce the function $T(n, i)$ with two parameters $n$ and $i$, that is equal to the optimal revenue for the rod of length $n$ when we cut it into pieces of length at most $i$. You can then write the recurrence for $T(n, i)$.

(b) Next, write pseudo-code for your algorithm (either memoized recursion or bottom-up) and (very briefly) justify its run-time.

## Problem 2 (Longest Palindrome Subsequence, 20 pts)

A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. Justify the correctness of your algorithm and analyze its running time. *(Hint: There is an easy way to do this using results from class, but you are welcome to design a new dynamic programming algorithm from scratch as well (make sure to state the recurrence relation as well as the base cases clearly).)*

## Problem 3 (The Coin Game, 40 points)

Consider a row of $n$ coins of values $v_1, v_2, \ldots, v_n$, where $n$ is even. A turn-based game is being played between 2 players where they alternate turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. We want an algorithm that determines the maximum possible amount of money you can definitely win if you move first (assume your opponent will also play to maximize the amount they get). To this end, we will use dynamic programming. Define the subproblems $\mathsf{MaxGain}(i, j)$ (for $i \leq j$) to be the maximum guaranteed payoff for the first player if given the subarray $[v_i, \ldots, v_j]$.

(a) Suppose the row of coins is $A = [1, 1, 1, 2, 2, 2]$. If you move first and try to maximize your gains, how much money can you definitely win regardless of the opponent's moves? Describe in words your optimal play strategy for this particular instance.

(b) State the base cases for $\mathsf{MaxGain}(i, j)$ and their values.

(c) Give the overall DP algorithm for $\mathsf{MaxGain}$ (just stating the recurrence is sufficient). Justify the correctness and runtime of your proposed algorithm.