CPSC 320: Tutorial 3

1. Let A be an array of n distinct integers. In this problem, we are interested in finding the longest increasing subsequence of A. That is, we want to find elements $A[i1], A[i2], ...A[it]$ such that $i_1 < i_2 < ... < i_t$, $A[i_1] < A[i_2] < ... < A[i_t]$, and t is as big as possible.

   For instance, if $A = (1, 9, 17, 5, 8, 6, 4, 7, 12, 3)$, then both $(1, 9, 17)$ and $(1, 5, 6, 7, 12)$ are increasing subsequences. Note that the subsequence given by the greedy algorithm, that is the subsequence $(1, 9, 17)$, is not the longest one.

   In order to find the longest increasing subsequence in the array, you can compute for each position i the length L[i] of the longest subsequence that ends with element $A[i]$.

   - Give a recurrence relation that expresses $L[i]$ as a function of $L[j]$ for values of j that are smaller than i. Hints: you need to consider the position of the previous element of the longest increasing subsequence.
   - Write pseudo-code for an algorithm that finds the longest increasing subsequence of an array with n elements.
   - What are the space and time complexities of your algorithm? How do these compare to a brute-force approach?

2. Given an unlimited number of coins with denominations $x_1, x_2, ...x_n$, and an amount A, find the minimum number of coins needed to make the amount A or output "impossible" if amount A cannot be made. Your algorithm should run in time $O(nA)$. Why doesn't greedy work?

3. Consider the binary operator @ defined by the following "addition" table:

@	1	2	3
1	2	2	1
2	3	2	1
3	1	3	3

   Note that @ is not associative, and that it is not commutative either (for instance 1 @ 2 = 2, but 2 @ 1 = 3).

Design an efficient algorithm that takes in an expression $x_1@x_2@x_3@...@x_n$, where each xi is either 1, 2 or 3, and determines if it is possible to insert parentheses in that expression so it evaluates to 1. For instance, your algorithm should return YES for 2 @ 2 @ 2 @ 2 @ 1, since (2 @ (2 @ 2)) @ (2 @ 1) = 1.

Analyze the running time and space of your algorithm.