Dynamic Programming

1. In a top-down approach to dynamic programming, the larger subproblems are solved before the smaller ones. True/False

2. Any Dynamic Programming algorithm with n subproblems will run in $O(n)$ time. True/False

3. Imagine starting with the given number n, and repeatedly chopping off a digit from one end or the other (your choice), until only one digit is left. The *square-depth* $S(n)$ of n is defined to be the maximum number of square numbers you can arrange to see along the way. For example, $S(32492) = 3$ via the sequence

   $$32492 \rightarrow 3249 \rightarrow 324 \rightarrow 24 \rightarrow 4$$

   since 3249, 324, and 4 are squares, and there is no better sequence of chops giving a larger score (although you can do as well other ways, since 49 and 9 are both squares).

   Describe an efficient algorithm to compute the *square-depth*, $S(n)$, of a given number n, written as a d-digit decimal number $a_1a_2 \ldots a_d$. Analyze your algorithm's running time.
   Your algorithm should run in time polynomial in d. You may assume a function IS-SQUARE(x) that returns, in constant time, 1 if x is square, and 0 if not.

4. The Borg want to find the ***largest square parking space*** in which to park the Borg Cube. That is, find the largest k such that there exists a $k \times k$ square in A containing all ones and no zeros.

	0	1	2	3	4
0	0	1	1	1	0
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	0	0	0
4	1	1	1	0	1

   In the example figure, the solution is 3, as illustrated by the 3×3 highlighted box.

   Describe an efficient algorithm that finds the size of the largest square parking space in A. Analyze the running time of your algorithm.

   Hint: Call $A[0][0]$ be the *top-left* of the parking lot, and call $A[n-1][n-1]$ the *bottom-right*. Use dynamic programming, with the subproblem $S[i, j]$ being the side length of the largest square parking space whose bottom-right corner is at (i, j).

5. In dynamic programming, we derive a recurrence relation for the solution to one subproblem in terms of solutions to other subproblems. To turn this relation into a bottom-up dynamic programming algorithm, we need an order to fill in the solution cells in a table, such that all *needed subproblems are solved before solving a subproblem*. For each of the following relations, give such a valid traversal order, or if no traversal order is possible for the given relation, briefly justify why.

i. $A(i,j) = \mathrm{F}(A(i,j-1), A(i-1,j-1), A(i-1,j+1))$

ii. $A(i,j) = \mathrm{F}(A(\min\{i,j\}-1,\min\{i,j\}-1), A(\max\{i,j\}-1,\max\{i,j\}-1))$

iii. $A(i,j) = \mathrm{F}(A(i-2,j-2), A(i+2,j+2))$

7. Obert Ratkins is having dinner at an upscale tapas bar, where he will order many small plates which will comprise his meal. There are n plates of food on the menu, with each plate p_i having integer volume v_i, integer calories c_i, and may or may not be sweet. Obert is on a diet: he wants to eat no more than C calories during his meal, but wants to fill his stomach as much as he can. He also wants to order exactly s sweet plates, for some $s \in [1, n]$, without purchasing the same dish twice. Describe a dynamic programming algorithm to find a set of plates that maximizes the volume of food Obert can eat given his constraints.

   Be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time.

Reference:

From “6.006: Introduction to Algorithms”, MIT.