Dynamic Programming - Advanced: Written Problems

1. Suppose you are given a row of n doors, each of which can either be locked or unlocked. Your goal is to have exactly S secured doors. The issue is that not all locked doors are secure. A door is secure if and only if:

   1. This door is locked AND
   2. This door is the first door (i.e., the left most door) OR The door to this one's left is also locked

   Develop an algorithm that, given the number of doors n and a value S, counts the total number of unique ways the doors can be locked to achieve EXACTLY S secure doors.

2. Suppose you are given two strings, each of length n. Let's call them A and B. Let's define the *similarity quotient (or SQ for short)* of two substrings of A and B to be the number of characters in those substrings that are not the same. To be more precise, let $SQ(i, j, k) = |\{0 \leq x < k : A[i + x] \neq B[j + x]\}|$. In other words, $SQ(i, j, k)$ is the number of characters that are not the same between the substrings, each of length k, beginning at index i in string A and beginning at index j in string B.

   For example, if A is *hello* and B is *there*, let's examine $SQ(1, 2, 4)$. Because $k = 4$, substrings are always length 4. Because $i = 1$ we start at the first character in A (notice we index from 1 here) and take $k = 4$ characters to get *hell*. In B, we start at index $j = 2$ and get *here*. These two substrings differ in their last two characters, so the total $SQ(1, 2, 4) = 2$.

   So, given all of this, solve the following problem: Given the strings A, B, and an integer I, find the maximum length L such that there exists some i and j such that $SQ(i, j, L) \leq I$. First, produce a brute-force algorithm for this problem and state the runtime.

3. Now, solve the problem more quickly by using dynamic programming. State the runtime of your new solution.

4. It's your sister's birthday and you decide to pull a big prank on her. You plan to wrap several decreasingly sized presents inside of each other so that she receives a large present, but many boxes later, realizes the gift is actually quite small in size. You silly prankster you!! You have a bunch of boxes and want to figure out how many boxes deep the prank can possibly go!

   Develop an algorithm that accepts as input the set of boxes $B = \{b_1, b_2, ..., b_n\}$ and returns the maximum number of boxes that can be placed inside of one another. Each b_i contains fields for that boxes width, height, and depth respectively. Remember that boxes can be rotated in any way. Your algorithm should return the actual set of boxes that should be used for the prank. State the runtime of your algorithm.

5. You are going skiing this weekend and wish to ski once down every unique run the mountain has to offer. The mountain has n runs given to you in a list called $R = \{r_1, r_2, ..., r_n\}$ where each r_i is a positive integer denoting the number of minutes it will take to ski that run and get back up the lift to the top of the mountain. You want to ski each run in the minimum number of days possible. Each day has L minutes of skiing available. Obviously, you must fit each run into a single day (you can't split a run over two days) and because of your obsessive compulsive disorder, you wish to ski the runs in order.

   In addition to this, you have preferences regarding the amount of unused time at the end of each day. If, at the end of a day, you have m minutes (e.g., only 10 mins) left in the day or less, than you are happy to stop skiing (you have a little time to get home, change, etc.). If, however, you have more than m minutes left in the day and not enough time for another run, you'll feel like you wasted valuable time. We will model your time wasted dissatisfaction (twd) as follows: '

$$twd(t) = \begin{cases} 0 & t = 0 \\ -C & 1 \leq t \leq m \\ (t-m)^2 & \text{otherwise} \end{cases}$$

   Where C is some constant. Develop an algorithm that minimizes the number of days needed to ski. If multiple optimal schedules exist, then pick the one that minimizes the sum of the $twd(t)$ values for all days.

6. Suppose you are given a rectangular board of size 2 by $w \geq 1$ (the units here don't matter). You are also given an endless bag of dominoes, each of size 2 by 1. Write a program that accepts the integer w as parameter, and returns the total number of unique ways the 2 by w board can be fully tiled. When we say *fully tiled* here, we mean that every square on the board is covered by dominoes. State the runtime of your algorithm. **See the next problem for some additional details.*

7. Now solve the exact same problem as above, except with a board of size 4 by w. State the runtime of your algorithm. For example, if the input given is $w = 2$, then the solution is 5, the following image shows all of the combinations of fully tiled boards of width $w = 2$: