Due date: Friday, February 10 at 10:30 AM. You will need to submit this via GradeScope. Late problem sets are not accepted.

In CompSci161, your response to each numbered question must be contained within a single piece of paper. Each numbered question must be responded to on a separate page. When you submit to GradeScope, you will need to inform the system which page of your scanned PDF contains the response you want graded. On occasion, we might request you to tag two (or more) parts, even though we know the two parts will be on the same page; when that happens, it usually means we are going to grade the parts separately. Failure to follow these directions will be treated as non-submission for the question(s) affected.

Please review the syllabus and course reference for the expectations of assignments in this class. Remember that problem sets are not online treasure hunts. You are welcome to discuss matters with classmates, but remember the Kenny Loggins rule. Remember that you may not seek help from any source where not all respondents are subject to UC Irvine's academic honesty policy.

Questions 2 and 3 here, as well as the questions on problem set 5, and also one question on your topic exam on Friday of week 6, will require a dynamic programming solution. For any such problem, you must do the following; missing one of these can cost you significant credit.

- Give a clear and precise English definition that describes the function you are implementing. Not *how* it works (yet), but rather *what* it solves.
- Give that function a meaningful variable name. This is not [just] me being pedantic; I have found it helps students with this topic if they do this. "OPT" is not a meaningful variable name, nor is "table." Single letters are not meaningful variable names. Yes, I know you're going to read question one first, and yes, I read it too.
- Give a clear recursive formula or algorithm in terms of smaller instances of *exactly* the same problem. If you aren't solving exactly the same problem, you might need to go back to the previous step.
- Describe the iterative running time correctly. This can either be by writing the iterative algorithm (in which case, you can point out where the previous part is within the solution), or by taking your recursive solution, counting the cases, describing the order in which the table would be filled in, and analyzing the time accordingly.

1. Suppose I am going to choose an integer between 1 and $n$, inclusive, according to some probability distribution. For each integer $i$, I have written $p_i$, the probability that I select $i$ as the chosen integer. You may assume that $\sum_{i=1}^{n} p_i = 1$.
   (a) Give an $\mathcal{O}(n^3)$ time algorithm to compute a $2D$-array $X$, where $X[i, j]$ is the probability that some integer in the range $[i, j]$ (inclusive) is chosen. You may assume that arithmetic operations take $\mathcal{O}(1)$ time each.
   (b) Give an $\mathcal{O}(n^2)$ time algorithm to solve the problem in part (a). If you are confident that your answer to this question is $\mathcal{O}(n^2)$, you may elect to skip the previous part and count this as your answer to both.

2. Professor Shindler gives lots of homework assignments, each of which have an easy version and a hard version[1]. Each student is allowed, for each homework, to submit either their answer to the easy version (and get $e_i > 0$ points) or the hard version (and get $h_i > 0$ points, which is also guaranteed to always be more than $e_i$) or to submit neither (and get 0 points for the assignment). Note that $e_i$ might have different values for each $i$, as might $h_i$. The values for all $n$ assignments are known at the start of the quarter.

   The catch is that the hard version is, perhaps not surprisingly, more difficult than the easy version. In order for you to do the hard version, you must have not done the immediate previous assignment at all: neither the easy nor the hard version (and thus are more relaxed, de-stressed, etc). You are allowed to do the hard version of assignment one if you want. Your goal is to maximize the number of points you get from homework assignments over the course of the quarter. Give an efficient dynamic programming algorithm to determine the largest number of points possible for a given quarter's homework choices.

3. *Skittles* are small candies; each piece of candy is one of five flavors (grape, lemon, orange, strawberry, or green). An exciting new party game, *Skittle Trader*, is sweeping the nation, and the skills you are learning in CompSci 161 will help you to achieve a high score!

   Here are the rules of the game. You are at a party with $n$ of your friends. Each of you has one piece of *Skittles* candy. Your friends form a line. You walk down the line, starting at friend 1 and finishing at friend $n$. At each friend, you have a choice: do you want to trade the candy you are holding for the one they are holding?

   - If you do not trade candy, your score remains the same.
   - If you and the friend have the same flavor candy, you earn one point.
   - If you and the friend trade, but you had different flavor types, you lose one point. Your score *can* go negative.

   You begin the game with zero points. Your goal is to finish the game with the most possible points.

   For purposes of the input to this problem, the Skittles flavors are numbers $0, 1, 2, 3, 4$. Your input to this problem is an array $C[1 \dots n]$, where each $C[i] \in [0, 4]$ is the type of candy the $i$th friend in line is holding. You also have a parameter for the type of candy you begin the game with.

   If you decide to actually play this game with friends, perhaps to test out your algorithm before submitting, please use individually wrapped candies instead, such as Starburst. This is nearly the most disgusting question I have given in a class, second only to the Chicken McNuggets question from ICS 46.

[1]This is not the actual policy.