Due Dates: One numbered question of your choice is due by **11:59PM Friday, 4 November** and will be graded on completion only (1 point). Your full answers to **any two** questions are due by **11:59PM Tuesday, 8 November**. Each solution will be graded for correctness and clarity (4 points per question).

At the top of your write-up for each problem, estimate the amount of time you spent on the problem, list your collaborators, and briefly describe the nature of your collaboration.

Each of the problems below is a well-known dynamic programming problem, so you may be able to find substantial guidance online. **Cite any resource you consult** to answer these problems.

I have provided detailed subquestions in the problems below to encourage you to develop a set of habits when looking at a new problem (specifically, in this case, one that is amenable to dynamic programming solutions).

1. **Maximum stock profit**

   Take a look at problem 7 on pages 318-319 of Kleinberg & Tardos. They start you off with a price p_i per share of a given stock, for days numbered $i = 1, ..., n$, and set you the task of developing an $O(n)$ dynamic programming algorithm for finding the largest profit per share you could have made had you bought the stock on day i and sold it on day $j > i$.

   More specifically, please do the following.

   (a) Show an example for which there is no available profit.

   (b) What constitutes "brute force" for this problem? Give and justify a big-oh estimate of the running time of a brute force algorithm.

   (c) Let $M(k)$ represent the maximum profit available among all pairs of days (i, j) where $1 \leq i < j \leq k$. (We will refer to $M(k)$ as a "parameterized subproblem" with one parameter. Note that $M(n)$ will be the solution to the problem we are trying to solve.)

   (d) Devise a recursive formula, including base case, for computing $M(k)$.

   (e) Give a big-oh estimate of the running time if you run your recursion as-is with no preprocessing or dynamic programming or other cleverness.

   (f) Show the values of $M(2)$ and $M(3)$ computed directly from your base case and your recurrence for $M(k)$.

   (g) Generalize the previous item to show an iterative algorithm for computing $M(1), M(2), ..., M(n)$ from your base case and recurrence.

   (h) Show the $M(1), ..., M(n)$ generated by your algorithm for $\mathbf{p} = [4, 5, 10, 7, 1, 6, 9, 8]$.

   (i) Give a big-oh estimate of the running time of your algorithm.

   (j) Show an enhanced algorithm that will generate not just the maximum profit $M(n)$, but also values of i and j that produce that profit.

   (k) Show that your enhanced algorithm has the same big-oh performance as your un-enhanced version.

2. **Wire-cutting with cost**

   In class (10/31 and 11/2) we discussed a wire-cutting problem (often referred to as a *rod-cutting problem*). We are given an array $p[1...m]$ where each $p[k]$ represents the price we can charge for a piece of wire of length k. The problem asks how to cut up a wire of length $n \leq m$ so as to maximize our total revenue from the resulting pieces of wire.

   For this problem, we will add a small twist–each cut you make will cost c units, thus cutting (sorry) into your profit. (For example, if you cut your original wire into three pieces, that will reduce your profit by $2c$ for the cost of making two cuts.)

   (a) What constitutes brute force for this problem? Give and justify a big-oh estimate of the running time of a brute force algorithm.

   (b) Define a parameterized subproblem (analogous to the $M(k)$ in the previous problem) suitable for solving this problem recursively.

   (c) Show how the maximum profit for a length-n piece of wire can be expressed in terms of one of your subproblems.

   (d) Show a recursive decomposition, including base case, of your parameterized subproblem.

   (e) Give a big-oh estimate of the running time if you run your recursion as-is with no preprocessing or dynamic programming or other cleverness.

   (f) Show an iterative algorithm for solving enough of your subproblems to enable you to solve the top-level problem for the length-n wire.

   (g) Give and justify a big-oh estimate of the running time of your algorithm.

   (h) Show an enhanced algorithm that will generate not just the maximum profit for the length-n wire, but also the lengths of the pieces (or the locations of the cuts) that produce that profit.

   (i) Give and justify a big-oh estimate of the running time of your enhanced algorithm.

3. **Longest common substring**

   Consider two character strings $A = A_1A_2...A_m$ and $B = B_1B_2...B_n$. We would like to find the length of the longest substring contained in both A and B. For example, if

   ```
   A = CATDOGEMUFOXGOAT
   ```

   and

   ```
   B = STRATEGEMS
   ```

   then the longest common substring (GEM) has length 3.

   (a) What constitutes brute force for this problem? Give and justify a big-oh estimate of the running time of a brute force algorithm.

   (b) Define a parameterized subproblem (analogous to the $M(k)$ in the previous problem) suitable for solving this problem recursively. Note that for this problem, your subproblem will have two parameters. (Hint: let the parameters i and j be indexes into the strings A and B.)

(c) Show how the length of the longest common substring can be expressed in terms of one of your subproblems.

(d) Show a recursive decomposition, including base case, of your parameterized subproblem.

(e) Give a big-oh estimate of the running time if you run your recursion as-is with no preprocessing or dynamic programming or other cleverness.

(f) Show an iterative algorithm for solving enough of your subproblems to enable you to solve the top-level problem for the full strings A and B.

(g) Give and justify a big-oh estimate of the running time of your algorithm.

(h) Show an enhanced algorithm that will generate not just the length of the longest common substring, but an actual common substring that has that length.

(i) Give and justify a big-oh estimate of the running time of your enhanced algorithm.