*Please enter your name and uID below.*

Name:

uID:

Collaborators, if any, and how you collaborated:

**Submission notes**

- Due at 11:59 pm on Thursday, September 22.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

**Sample of how to write a recursive formula; do not submit on Gradescope.**

**Problem:** (Antimicrobial Coins BT)
To combat the pandemic, your country has introduced a new currency, whose physical representation consists entirely of antimicrobial coins with values $v_1 = 1, v_2, \ldots, v_n \in \mathbb{Z}^+$. Since these are expensive to produce, the bank always wants to dispense the fewest possible coins for a given withdrawal request $W \in \mathbb{Z}^+$. The bank has access to an unlimited number of each denomination of coin, but wants to always make change using the minimum total number of coins.

Give a recursive backtracking algorithm *(specified by a recursive formula with 1 or more equations, along with necessary base cases)* for computing the minimum number of coins needed to satisfy a withdrawal $W$ in this currency.

**Solution:**

Let $coins(W)$ denote the minimum number of coins needed to satisfy a withdrawal $W$.

$$coins(W) = \begin{cases} \infty & \text{if } W < 0 \\ 0 & \text{if } W = 0 \\ 1 + \min_i coins(W - v_i) & \text{otherwise} \end{cases}$$

1. (Wall backtracking, 25pts) This problem asks you to write a backtracking algorithm to solve PS2 Programming – read the programming problem first if you haven't already.

   Let $D[1..n]$ be a global array containing the $n$ distances you'd like to climb. Let MinWall$(i, h)$ be the minimum wall height necessary to complete all exercises from $i$ onward, starting at height $h$ and ending at the ground (height 0).[1][2] Your task is to determine what MinWall should return in different cases, and then write its full recursive formula. MinWall should return $\infty$ if it's impossible to complete the desired exercises.

   (a) **Base cases**

      i. What should MinWall$(n+1, 0)$ return? That is, what is the minimum wall height needed to complete *no* exercises, starting at height 0?

      ii. What should MinWall$(n+1, 10)$ return?

   (b) **Recursive cases for MinWall**$(i, h)$. For the parts below, you need to complete exercises from $i$ onward and you're currently at height $h$.

      i. What subproblem (i.e. recursive call to MinWall) results if you choose to climb *down*?

      ii. What subproblem results if you choose to climb *up*?

      iii. If $h < D[i]$ (you're close to the ground), what single recursive call should MinWall$(i, h)$ return?

      iv. What expression gives the return value for MinWall$(i, h)$ in the general case, when the previous case and base cases don't apply? Use your expressions from (i) and (ii) as part, but not necessarily all, of your return value.

   (c) Your answers to the above should capture *all* cases that are necessary to consider for this problem. You do not, for example, need a separate case for when $i = n$. Write a full recursive formula for MinWall$(i, h)$.

   (d) What single call to MinWall will give the answer to your overall problem?

[1] subject to the rules listed in the programming problem

[2] MinWall defines a *subproblem*, and its two parameters contain the only information necessary to answer this subproblem (along with the global array $D$).

2. (Wall dynamic programming, 20pts) This problem asks you to describe the pieces of a dynamic programming algorithm for your backtracking algorithm in problem #1.

   (a) You can efficiently store all subproblems for MinWall in a 2D array. Given some array $D[1..n]$, let $H$ be the sum of all its distances, $H = \sum_i D[i]$. How big does your memoization data structure need to be? Give an exact number of rows and columns in terms of $n$ and/or $H$, including space for base case values.

   (b) Must you fill your array starting with small values of $i$, large values, or is either OK?

   (c) Must you fill your array starting with small values of $h$, large values, or is either OK?

   (d) You should fill your array with two nested loops. Must your outer loop iterate over $i$, $h$, or is either OK?

   (e) After you have filled your array, which location in the array contains the final answer to the problem?

   (f) What is the running time of this algorithm?

3. (Card game backtracking, 35pts) Consider the following card game. You start by laying $n$ cards in a line, and write on each card an integer (which could be positive, negative, or zero). We will refer to these card values with the array $C[1..n]$. The game has the following rules:

   - You start with a score of 0. You go through the cards one at a time, starting at 1 and ending at card $n$.
   - At each card, you must say either HI or LO.
   - If you say HI at card $i$, you add $C[i]$ points to your score (if $C[i] < 0$, this will result in a decrease in your score).
   - If you say LO at card $i$, you subtract $C[i]$ points from your score (if $C[i] < 0$, this will result in an increase in your score).
   - You cannot say the same word more than TEN times in a row. So if you choose HI for cards #5 through #14, then you must choose LO for card #15.

   Give a recursive backtracking algorithm (specified by a recursive formula with 1 or more equations, along with necessary base cases) for computing the maximum possible score in this game. What call to your recursive formula would you make in order to solve this game?