EECS 376: Foundations of Computer Science **Due 8:00pm, January 31**
University of Michigan, Winter 2024 **Homework 3** **(accepted until 9:59 pm, no credit after)**

This homework has 8 questions, for a total of 100 points and 9 extra-credit points.
Unless otherwise stated, each question requires *clear*, *logically correct*, and *sufficient* justification to convince the reader.
For bonus/extra-credit questions, we will provide very limited guidance in office hours and on Piazza, and we do not guarantee anything about the difficulty of these questions.
We strongly encourage you to typeset your solutions in LaTeX.
If you collaborated with someone, you must state their name(s). You must *write your own solution* for all problems and *may not use any other student's write-up*.

(0 pts) 0. **Before you start; before you submit.**

(a) Carefully review What to give when you "give an algorithm" before starting this assignment, and apply it to the solutions you submit.

(b) If applicable, state the name(s) and uniqname(s) of your collaborator(s).

Solution:

(10 pts) 1. **Self assessment.**

Carefully read and understand the posted solutions to the previous homework. Identify one part for which your own solution has the most room for improvement (e.g., has unsound reasoning, doesn't show what was required, could be significantly clearer or better organized, etc.). Copy or screenshot this solution, then in a few sentences, explain what was deficient and how it could be fixed.

(Alternatively, if you think one of your solutions is significantly *better* than the posted one, copy it here and explain why you think it is better.)

Solution:

(10 pts) 2. **Bodybuilders in the magic room.**

You are at a party with n bodybuilders having distinct weights, and want to identify the kth lightest person, for some particular k. You are allowed to ask questions of the form "Is person A lighter than person B?" An easy solution is to use a sorting algorithm, which takes $\Theta(n \log n)$ comparisons. However, you suspect you can do better because you have access to a "magic room."

The "magic room" works as follows: if you bring any S people into this room, it will identify the person whose weight is the $\lceil S/2 \rceil$th lightest among them. Give an algorithm for the above problem that uses only $O(n)$ comparisons and $O(\log n)$ accesses to the magic room.

Solution:

3. **Pikachu: I trade you!**

You are a savvy investor who is interested in an especially hot asset that is now on the market: a rare Pikachu Pokémon trading card. You want to purchase and sell this card *just once*, and time these events to maximize your rate of return, i.e., the percentage by which the card's value increases between purchase and sale.

Your top trading-desk researchers have prepared their best estimates of how much the card's value will change on each of the following n days. These estimates are given as nonnegative real *multiplicative* factors, i.e., a 10% gain is represented by 1.1, and a 20% loss is represented by 0.8. Given the array $A[1, \ldots, n]$ of these estimates, you want to find indices $1 \leq i \leq j \leq n$ that *maximize*

$$A[i] \cdot A[i+1] \cdots A[j] \ ,$$

i.e., the aggregate growth in value if the card is purchased at the start of day i and sold at the end of day j. (If the card would lose value every day, this can be indicated by taking $i > j$.)

For instance, if $A[1, \ldots, 6] = [0.8, 1.1, 0.95, 1.5, 0.8, 1.25]$, then the maximum aggregate growth is 1.5675, which is achieved by $i = 2$, $j = 6$. (Note that it is also achieved by $i = 2$, $j = 4$. So, an optimal choice of i and j is not necessarily unique, but the maximum aggregate growth is unique.)

In this question you will design a divide-and-conquer algorithm that solves this problem. You may assume that two real numbers can be multiplied in constant time, and for simplicity, you may assume that n is a power of two (if you wish).

(7 pts) (a) Suppose that for some index k of interest, you want to optimize the aggregate growth under the constraint that you own the card on day k and sell it strictly after day k (i.e., you purchase it at the start of some day $i \leq k$ and sell it at the end of some day $j > k$). Give an algorithm that solves this problem in $O(n)$ time. On input $A[1, \ldots, n]$ and k, it should output the maximum aggregate growth, and corresponding purchase and sale days i, j that achieve it.

Solution:

(8 pts) (b) Now, give a divide-and-conquer algorithm that solves the main problem in $O(n \log n)$ time. Given the array $A[1, \ldots, n]$, it should output the optimal aggregate growth.

Solution:

4. **Playing at Pinball Pete's.**

There is a game at Pinball Pete's in which you need *exactly* p points to win. You start at zero points, and can score one, two, or three points per turn. You are interested in the number of different ways a person can win. Different orderings of the same numbers of points count as distinct ways to win (for example, if $p = 3$, then scoring 1 point and then 2 points is different than scoring 2 points and then 1 point). Define $SP(p)$ to be the number of ways to score exactly p points.

(10 pts) (a) Derive, with justification, a recurrence relation for $SP(p)$. Remember to include the base case(s).

Solution:

(10 pts) (b) Give pseudocode for a (bottom-up) dynamic-programming algorithm that, on input p, outputs $SP(p)$, the number of ways to get exactly p points in $O(p)$ time. Is the algorithm "efficient," as defined by this course? (For simplicity, here you may assume that integer addition takes constant time.)

Solution:

(3 EC pts) (c) *Optional extra-credit.* There is another popular game with some strange scoring rules. In each "turn" it is possible to score two, three, or six points. Moreover, immediately after (and only after) scoring six points, one can attempt to score either one or two "extra" points, but this attempt might fail (resulting in zero extra points).

Define $FP(p)$ to be the number of ways to score exactly p points in this game (again, order matters). As above, derive (with justification and base case(s)) a recurrence relation for $FP(p)$, give pseudocode for a dynamic-programming algorithm that outputs $FP(p)$ given input p, and analyze its running time.

Solution:

5. **A pebble game.**

Many two-player strategic games, like the several variants of Nim, can be modeled as follows. There is a directed acyclic graph $G = (V, E)$ presented in topological order: the vertices are labeled as $V = \{1, \ldots, n\}$, and every edge $(u, v) \in E$ has $u < v$. Moreover, every vertex $i < n$ has at least one outgoing edge.

Two players, called maize and blue, take turns in the following game on this graph. They start with a pebble at vertex 1, and maize plays first. On each turn, the acting player must move the pebble from the current vertex u to a new vertex v along some edge $(u, v) \in E$ of the graph, of the acting player's choice. If a player moves the pebble to vertex n (the final one), then that player wins the game.

Give a dynamic-programming algorithm that, given the graph G as input, determines which player can be guaranteed a win by playing perfectly, no matter how the opponent plays.

(10 pts) (a) Define $W(i)$ to indicate whether the acting player can guarantee a win if the pebble is at vertex i. Derive, with justification, a recurrence relation for $W(i)$. Remember to include appropriate base case(s).

Solution:

(10 pts) (b) Give pseudocode for a (bottom-up) dynamic programming algorithm that solves the problem, and analyze its running time.

Solution:

(5 pts) (c) Extend your algorithm from the previous part to also output a "winning strategy" for whichever player has one. This should be represented as an array specifying, for each vertex u, where the player should move the pebble if it is at vertex u on that player's turn. Also, analyze the algorithm's running time.

Solution:

6. **Houston, we have a problem!**

Alina and Kaitlyn decide to make the drive from Ann Arbor to Houston to cheer on the Wolverines at the College Football National Championship Game! They start their drive from Ann Arbor and make their way to Houston. Along their journey, there are n hotels, located at positive distances $a_1 < a_2 < \cdots < a_n$ from Ann Arbor. They can stop only at these specified hotels, but may choose which one(s) to spend their night(s) at, and which ones to skip. They must end their journey at the last hotel, located at distance a_n.

The trip from Ann Arbor to Houston is 1300 miles. They don't want to drive too much or too little in a day, so they are aiming to go about 500 miles per day. However, due to the distance between the hotels, this may not always be possible. If they drive x miles on a given day, they will face a "penalty" of $(500 - x)^2$ for that day.

For example, if $a_1 = 450, a_2 = 670, a_3 = 1300$, then we have the following possible choices of hotels (by their indices) and the associated penalties:

$$\begin{aligned} 1, 2, 3 &\to (500-450)^2 + (500-220)^2 + (500-630)^2 && = 97800 \\ 2, 3 &\to (500-670)^2 + (500-630)^2 && = 45800 \\ 1, 3 &\to (500-450)^2 + (500-850)^2 && = 125000 \\ 3 &\to (500-1300)^2 && = 640000 \end{aligned}$$

Therefore, the minimum possible total penalty is 45,800. Alina and Kaitlyn should drive to the second hotel (at 670 miles) on the first day, then drive the remaining 630 miles to Houston on the second day.

The goal in this problem is to devise an efficient algorithm that, given an array $A[1 \ldots n]$ of the distances $a_1, \ldots, a_n$, determines which hotel(s) to stop at so as to minimize the total penalty incurred on the trip.

Hint: Consider a function $p(j)$ that represents the minimum possible total penalty when starting from Ann Arbor and ending at hotel j.

(10 pts) (a) Give, with justification, a recurrence relation for $p(j)$ that is suitable for a dynamic-programming solution to the problem. Be sure to identify the base case(s).

Solution:

(10 pts) (b) Give pseudocode for a (bottom-up) dynamic programming algorithm that solves the problem, and analyze its running time.

Solution:

(6 EC pts) 7. **Optional extra credit: flash mob!**

You've accepted a summer internship at a startup that's making a new flash mob app. Here's how it works: Given n people in the plane, the app will periodically convene a flash mob of size k, where k is some constant. The objective is to find a size-k subset of the n people so that the sum of the pairwise distances between those k people is as small as possible. In other words, given a set of n points, we wish to find a subset P of exactly k points that minimizes the sum $\sum_{p,q \in P} d(p, q)$, where $d(\cdot, \cdot)$ denotes the Euclidean distance. Give an $O(n \log n)$-time algorithm for this problem.

Solution: