CS 6363.003 Midterm 1

March 11, 2021

Please read the following instructions carefully before you begin.

- The exam has 4 problems, some of which have multiple parts. Please start the solution to each problem on a new page.
- The exam is meant to take around 1 hour and 15 minutes, but you have some extra time to prepare the solutions for submission to eLearning and work through any related technical issues. You must finish uploading your solutions within 2 hours of beginning the exam or by 9:59am CST on Friday, March 12, *whichever comes first*. You may upload as many versions of your solutions as you would like, but you can only submit them once. Your solutions will be submitted automatically 2 hours after you begin the exam to prevent you from uploading but forgetting to actually submit your solutions.
- If you cannot successfully upload your solutions to eLearning before the end of the 2 hour time limit, email a copy to Kyle as soon as you possibly can.
- It is highly recommended that you take the exam between 9am CST and 4pm CST on Thursday, March 11th. Kyle will be actively watching for questions via email during this time. He will also try to answer emailed questions at other times during the exam period, but he may be slower to respond, especially if he is asleep! Feel free to ask for clarification on any of the problems.
- Questions are not necessarily given in order of difficulty, so read through them all before you begin writing.
- This exam is closed book. No notes or calculators are permitted.
- If asked to describe an algorithm, you should state your algorithm clearly (preferably with pseudocode) and briefly explain its asymptotic running time in big-O notation in terms of the input size. Each individual problem will specify whether or not you must justify correctness.
- Just breathe. You can do this.

1. Answer each of the following questions. There is no need to justify your answers for these questions.

   (a) **(2.5 out of 10)** Using Θ-notation in terms of n, what is the solution to the recurrence $T(n) = 9T(n/3) + n^2$?

   (b) **(2.5 out of 10)** Using Θ-notation in terms of n, what is the solution to the recurrence $T(n) = T(4n/7) + T(2n/7) + n$?

   (c) **(2.5 out of 10)** Using Θ-notation in terms of n, what is the solution to the recurrence $T(n) = 5T(n/3) + n$?

   (d) **(2.5 out of 10)** Consider the following recursive function which is defined in terms of a fixed array $X[1 \mathinner{..} n]$.

$$LaLa(i,j) = \begin{cases} 0 & \text{if } i \le 0 \text{ or } j \le 0 \\ \max_{1 \le k \le i}(X[i] \cdot LaLa(i-k, j-1)) & \text{otherwise} \end{cases}$$

   Using Θ-notation in terms of n, how long does it take to compute $LaLa(n,n)$ using dynamic programming?

2. (a) **(5 out of 10)** Suppose we're given two strings $A[1 \mathinner{..} m]$ and $B[1 \mathinner{..} n]$. A third string C is a ***common supersequence*** of A and B if both A and B are subsequences of C. Finding the length of a *shortest* common supersequence of A and B can be accomplished recursively/using dynamic programming.

   Specifically, let $SCS(i,j)$ be the length of the shortest common supersequence of $A[1 \mathinner{..} i]$ and $B[1 \mathinner{..} j]$. Rewrite the following recursive definition of $SCS(i,j)$ with all ***five*** blanks filled in correctly. There is no need to justify your answer.

$$SCS(i,j) = \begin{cases} i & \text{if } j = 0 \\ \underline{\qquad} & \text{if } i = 0 \\ \min \begin{cases} SCS(i, j-1) + \underline{\qquad} \\ SCS(i-1, j) + \underline{\qquad} \end{cases} & \text{if } A[i] \ne B[j] \\ SCS(\underline{\qquad}, \underline{\qquad}) + 1 & \text{otherwise} \end{cases}$$

(b) **(5 out of 10)** Your kindly woodcrafting neighbor has just moved out of the country but left you with n pieces of lumber. These pieces of wood have no particular value to you, but you hope to sell them some day to a enterprising carpenter. To prepare for your eventual payoff, you need to sort these pieces of lumber in increasing order of quality. You know nothing about the lumber industry, so you decide to ask an eccentric lumber enthusiast to compare pieces of lumber for you. The enthusiast does not want to feel like he's doing all the work for you, however, so he will only sort a collection of lumber if it has already been divided up into three groups of roughly equal size, and each individual group is already sorted.

More formally, you're given an array $L[1 \,..\, n]$ of *lumbers* and a procedure Merge3($W[1 \,..\, k]$) that sorts a subarray of lumbers $W[1 \,..\, k]$ assuming the subarrays $W[1 \,..\, \lfloor k/3 \rfloor]$, $W[\lfloor k/3 \rfloor + 1 \,..\, \lfloor 2k/3 \rfloor]$, and $W[\lfloor 2k/3 \rfloor + 1 \,..\, k]$ are already sorted. Design *and analyze* an algorithm to sort the array L. Accessing elements of and making change to L must occur only within calls to Merge3. In particular, you ***may not*** compare lumbers directly.

You do not need to justify correctness of your algorithm for this problem.

3. **(10 points)** Suppose you are given an array $A[1 \,..\, n]$ with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a ***local minimum*** if it is less than or equal to both of its neighbors, or more formally, if $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are six local minima in the following array:

9	**7**	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9
	▲			▲				▲		▲	▲			▲	

We can easily find a local minimum in $O(n)$ time by scanning through the whole array. Describe and analyze a faster algorithm that finds a local minimum in $O(\log n)$ time. You should *briefly* justify the correctness and running time of your algorithm. *[Hint: Recursively search a subarray with the same boundary conditions.]*

4. Everybody is falling in love with Flower Monsters (also known as Florámon), the exciting new hobby where gardeners pit beautiful and dangerous plants against one another in head-to-head competitions to see which ones can soak up the most nutriants from league approved battle pots. Hoping to cash in on this craze, you decide to open up a Flower Monster Nursery where you'll grow shiny new Flower Monsters to sell to enthuiastic gardeners.

   Unfortunately, growing Flower Monsters is hard work, and you only have the time and resources to take care of one Flower Monster at a time. It also costs you $10 for seeds and material each time you plant a brand new Flower Monster, and thanks to the way their abilities change throughout their strange life cycles, the amount you can sell your Flower Monster for directly depends upon how many days it has been since you first planted it.

   Suppose you are given an array $P[1 \,..\, n]$ where $P[i]$ denotes the amount of revenue you get from selling a Flower Monster i days after first planting it. (You may not sell a Flower Monster on the same day you first plant it.) You want to design an algorithm that will tell you the maximum total profit (total revenue minus the total cost of planting new Flower Monsters) you can obtain over the course of n days. For simplicity, you may assume the following: a) you plant a new Flower Monster on day 0; b) if you sell on any day 1 through $n-1$, you immediately plant a new Flower Monster on *the same day*; and c) you sell on day n without planting another Flower Monster. For example, if you decide to sell on days 3, 10, and n, then your total profit is $P[3-0]+P[10-3]+P[n-10]-3\cdot 10$.

   (a) **(5 out of 10)** For all $0 \le i \le n-1$, let $MaxProfit(i)$ be the maximum total profit you can obtain if you plant a new Flower Monster on day i and proceed to sell and plant Flower Monsters over the course of days $i+1$ through n. Our ultimate goal is to compute $MaxProfit(0)$.

   Give a recursive definition of $MaxProfit(i)$. If you wish, you may propose a convinient definition for $MaxProfit(n)$ to use as a base case. You should *briefly* justify the correctness of your recursive definition.

   (b) **(5 out of 10)** Describe and analyze an efficient dynamic programming algorithm to compute the maximum total profit you can obtain over the course of n days. You do not need to justify correctness of your algorithm, but you should go through the standard memoization steps anyway to make sure your algorithm is correct. Don't forget to explain your running time. Unless you gave a recursive definition for which there is no efficient memoization method, you may assume your answer to part (a) is correct when designing your algorithm.