CS 4349.003 Midterm 1—Questions and Instructions

October 2, 2019

Please read the following instructions carefully before you begin.

- Write your name and Net ID on the *answer sheets* cover page and your Net ID on each additional page. Answer each of the ***four questions*** on the answer sheets provided. One sheet was intentionally left blank to provide you with scratch paper.
- Questions are not necessary 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.
- You have one hour and 15 minutes to take the exam.
- Please turn in these question sheets, your answer sheets, and scratch paper at the end of the exam period.
- 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.
- Feel free to ask for clarification on any of the questions.
- Just breathe. You can do this.

1. Please answer each of the following questions. You do not have to justify your answers.

   (a) **(3 out of 10)** What is the worst-case running time of the following procedure in terms of m and n?

```
StrangeSum(A[1 .. m], B[1 .. n]):
  sum ← 42
  for i ← 1 to m
      for j ← n down to 1
          for k ← i down to 1
              sum ← sum + A[i] + A[k] + B[j]
  return sum
```

   (b) **(2 out of 10)** What is the worst-case running time of quicksort on an n-element array if the pivot is chosen arbitrarily?

   (c) **(2 out of 10)** What is the running time of quicksort on an n-element array if the rank of the pivot is guaranteed to be between $n/5$ and $4n/5$?

   (d) **(3 out of 10)** Consider the function $StrangeRec(i, j)$ defined recursively as follows:

$$StrangeRec(i,j) = \begin{cases} 9 & \text{if } i = 0 \\ 2 & \text{if } i > 0 \text{ and } j = 0 \\ StrangeRec(i-1,j) + StrangeRec(i-1,j-1) & \text{otherwise} \end{cases}$$

   In terms of m and n, how long does it take to compute $StrangeRec(m, n)$ using dynamic programming? ***You do not need to write any code to answer this question.***

2. **(10 points)** Use Θ-notation to provide asymptotically tight bounds in terms of n for the solution to each recurrence. Assume each recurrence has a non-trivial base case of $T(n) = \Theta(1)$ for all $n \le n_0$ where n_0 is a suitably large constant. For example, if asked to solve $T(n) = 2T(n/2) + n$, your answer should be $\Theta(n \log n)$. You do not need to justify your answers. Each part is worth 2 points out of 10.

   (a) $T(n) = 2T(n/3) + n$

   (b) $T(n) = 8T(n/2) + n$

   (c) $T(n) = 8T(n/2) + n^3$

   (d) $T(n) = 2T(n/2) + n^2$

   (e) $T(n) = T(5n/12) + T(7n/12) + n$

3. **(10 points)** Suppose you are given an integer k and an array $A[1 .. n]$ of n *distinct* integers (i.e., no two integers in A are equal), *sorted in increasing order*. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i + k$. For full credit, your algorithm should run in $O(\log n)$ time. Slower but correct algorithms are worth some partial credit. You do not need to justify correctness of your algorithm, but you should provide a brief explanation for its running time.

   [Hint: Check if $A[\lceil n/2 \rceil] = \lceil n/2 \rceil + k$. If not, is there a subset of $\lfloor n/2 \rfloor$ elements you can search recursively? Be careful with the value of k you pass into your recursive call.]

4. It's almost time to show off your flippin' sweet dancing skills! Tomorrow is the big dance contest you've been training for your entire life. You've obtained an advance copy of the list of n songs that the judges will play during the contest, in chronological order. Yesssssssss!

   You know all the songs, all the judges, and your own dancing ability extremely well. For each integer k, you know that if you dance to the kth song on the schedule, you will be awarded exactly $Score[k]$ points, but then you will be physically unable to dance for the next $Wait[k]$ songs (that is, you cannot dance to songs $k+1$ through $k+Wait[k]$). The dancer with the highest total score at the end of the night wins the contest, so you want your total score to be as high as possible.

   Let's work through designing and analyzing an algorithm to compute the maximum total score you can achieve. The input to this sweet algorithm will be a pair of arrays $Score[1..n]$ and $Wait[1..n]$. For simplicity, we'll assume for each integer k that $Wait[k] \leq n-k$.

   (a) **(4 out of 10)** Let $MaxScore(i)$ be the maximum total score you can obtain dancing to some subset of the songs i through n. We can design a recursive definition for $MaxScore(i)$ based on the following observation: we can either dance to song i for $Score[i]$ points and then be forced to skip the next couple songs, or we can skip song i.

   Complete the following recursive definition of $MaxScore(i)$ based on the above observation. There are **three** blanks to fill in. You do not need to justify your answer.

$$MaxScore(i) = \begin{cases} \_\_\_\_\_ & \text{if } i > n \\ \max\left\{ \begin{array}{c} Score[i] + MaxScore(\_\_\_\_\_), \\ MaxScore(\_\_\_\_\_) \end{array} \right\} & \text{otherwise} \end{cases}$$

   (b) **(6 out of 10)** Describe and analyze an efficient dynamic programming algorithm to compute the maximum total score you can achieve. 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. You may assume your answer to part (a) is correct when designing your algorithm.