CS 4349.501 Midterm Exam—Problems and Instructions

October 4, 2017

Please read the following instructions carefully before you begin.

- **DON'T PANIC!**
- Write your name and Net ID on the *answer sheets* cover page and your Net ID on each additional page. Answer each of the five questions on the answer sheets provided. Two sheets were intentionally left blank to provide you with scratch paper.
- You're allowed to bring in one 8.5" by 11" piece of paper with notes written or printed on front and back.
- You have two hours to take the exam.
- Please turn in these problem sheets, your answer sheets, scratch paper, and notes by the end of the two hours.
- If asked to design and analyze an algorithm, you should describe your algorithm clearly and give its asymptotic running time using big-oh notation in terms of the input size. The only exception is Problem 2, where you are asked for an exact number of operations. ***You do not have to write proofs for this exam.***
- Feel free to ask for clarification on any of the problems.
- Greedy algorithms never work, at least not on this exam.

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

   (a) $T(n) = 9T(n/3) + n^2$

   (b) $T(n) = T(n/2) + \sqrt{n}$

   (c) $T(n) = 5T(n/2) + n^2$

   (d) $T(n) = T(n/4) + T(3n/4) + n$

   (e) $T(n) = T(n/6) + T(2n/3) + n$

2. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a *flip*—insert a spatula under the top of k pancakes, for some integer k between 1 and n, and flip them all over.

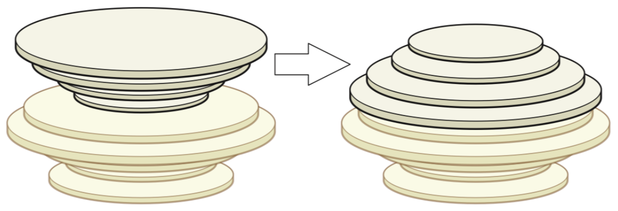

Figure 1. Flipping the top four pancakes.

   Describe an algorithm to sort an arbitrary stack of n pancakes using as few flips as possible. *Exactly* how many *flips* does your algorithm perform in the worst case?

3. Suppose you are given a sorted array of n distinct numbers that has been *rotated* k steps, for some ***unknown*** integer k between 1 and $n-1$. That is, you are given an array $A[1..n]$ such that the prefix $A[1..k]$ is sorted in increasing order, the suffix $A[k+1..n]$ is sorted in increasing order, and $A[n] < A[1]$.

   For example, you might be given the following 16-element array (where $k = 10$):

9	13	16	18	19	23	28	31	37	42	−4	0	2	5	7	8

   Describe and analyze an algorithm to compute the unknown integer k in $o(n)$ time. Yes, that is a little-oh.

4. It's almost time to show off your sweet dancing skills! Tomorrow is the big dance contest you've been training for your whole life. Fortunately, you've been given an advanced copy of the list of n songs the judges will play during the contest, in chronological order.

   You know the songs, 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'll be too tired to dance for the next $Wait[k]$ songs (that is, you have to skip songs $k+1$ through $k+Wait[k]$). Of course, you can skip other songs even if you are well-rested if you think its points are not worth its wait. Songs you skip are worth 0 points. Naturally, you want to maximize your total score.

   Describe and analyze an efficient algorithm to compute the maximum total score you can achieve. The input to your algorithm is the pair of arrays $Score[1..n]$ and $Wait[1..n]$.

5. Recall, a subsequence of a sequence/array X is any sequence obtained by extracting elements from X in order; the elements need not be contiguous in X.

   Let $A[1..m]$ and $B[1..n]$ be two arbitrary arrays. A ***common subsequence*** of A and B is another sequence that is a subsequence of both A and B. Describe and analyze an efficient algorithm to compute the length of the *longest* common subsequence of A and B.