CS 4349.400 Midterm Exam—Problems and Instructions

October 3, 2018

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!
- 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 one hour and 15 minutes to take the exam.
- Please turn in these problem sheets, your answer sheets, scratch paper, and notes at the end of the exam period.
- Writing "I don't know" ***and nothing else*** for any question or lettered part of a question is worth 25% credit. If you leave the solution blank or write anything else, we will grade exactly what is written.
- 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. ***You do not have to justify (prove) correctness of algorithms for this exam.***
- Feel free to ask for clarification on any of the problems.
- ***This exam does not cover greedy algorithms, so do not use them.***

1. For parts (a) through (c), use Θ-notation to provide asymptotically tight bounds in terms of n for the solution to the 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 $T(n) = 2T(n/2) + n$, then your answer should be $\Theta(n \log n)$. ***You do not need to explain your answers.***

   (a) **(2 out of 10)** $T(n) = 5T(n/2) + n$

   (b) **(2 out of 10)** $T(n) = 8T(n/4) + n\sqrt{n}$

   (c) **(2 out of 10)** $T(n) = T(n/4) + T(2n/3) + n$

   (d) **(4 out of 10)** Give an asymptotically tight bound for the running time of the algorithm StoogeSort($A[1..n]$) given below.

```
StoogeSort($A[1..n]$):
  if $n = 2$ and $A[1] > A[2]$
      swap $A[1] \longleftrightarrow A[2]$
  else if $n > 2$
      $m \leftarrow \lceil 2n/3 \rceil$
      StoogeSort($A[1..m]$)
      StoogeSort($A[n-m+1..n]$)
      StoogeSort($A[1..m]$)
```

   [Hint: Write a recurrence for the running time, and be careful with the size of the recursive calls.]

2. Recall that the Tower of Hanoi puzzle consists of three pegs and n disks of different sizes. Initially, all the disks are on one peg, stacked in order by size, with the largest disk on the bottom and the smallest disk on top. In a single move, you can transfer the highest disk on any peg to a different peg, except that you may never place a larger disk on top of a smaller one. The goal is to move all the disks onto one other peg.

   Now suppose the pegs are arranged in a row, and you are forbidden to transfer a disk directly between the left and right pegs in a single move; every move must involve the middle peg.

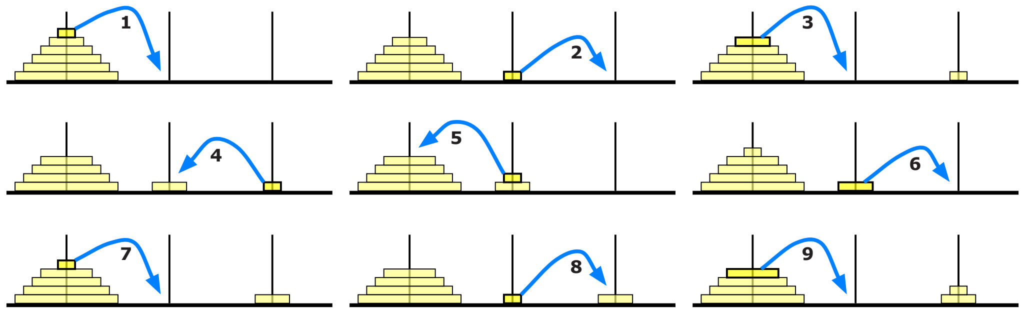

Figure 1. The first nine moves in a restricted Tower of Hanoi solution.

(a) **(7 out of 10)** Describe a recursive algorithm to transfer n disks from the ***left peg*** to the ***right peg*** under the above restriction. ***Do not analyze your algorithm for this part.*** Simply writing down the algorithm from lecture is worth *no credit*, because it moves disks directly between the left and right pegs.

(b) **(3 out of 10)** It is likely that your algorithm from part (a) uses exactly $3^n - 2$ moves. Give a short proof that this is the case. You will not receive credit for this part unless your algorithm actually uses $3^n - 2$ moves or your write "I don't know". *[Hint: Use induction.]*

3. Suppose we 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 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 find a local minimum in $O(n)$ time by scanning through the array. Describe an algorithm that finds a local minimum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array* **must** *have at least one local minimum. Why?]*

4. A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

$$\text{BANANA}_{\text{ANANAS}} \qquad \text{BAN}_{\text{ANA}}\text{ANA}_{\text{NAS}} \qquad \text{B}_{\text{AN}}\text{AN}_{\text{A}}\text{A}_{\text{NA}}\text{NA}_{\text{S}}$$

Similarly, the string PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

$$\text{PRO}^{\text{D}}\text{G}^{\text{Y}}\text{R}^{\text{NAM}}\text{AMMI}^{\text{I}}\text{N}^{\text{C}}\text{G} \qquad {}^{\text{DY}}\text{PRO}^{\text{N}}\text{G}^{\text{A}}\text{R}^{\text{M}}\text{AMM}^{\text{IC}}\text{ING}$$

Our goal for this problem is to design an algorithm that given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, determines whether C is a shuffle of A and B.

(a) **(5 out of 10)** Let $IsShuffle(i, j)$ be a function that, for any $0 \leq i \leq m$ and $0 \leq j \leq n$, returns TRUE if $C[1..i+j]$ is a shuffle of $A[1..i]$ and $B[1..j]$ and FALSE otherwise. Give a recursive definition of $IsShuffle(i, j)$ (i.e., a recurrence relation). Don't forget the base case(s)!

(b) **(5 out of 10)** Describe an efficient algorithm for determining if $C[1..m+n]$ is a shuffle of $A[1..m]$ and $B[1..n]$. Using big-O notation, state the running time ***and*** space usage of your algorithm. Feel free to use any details from your solution to part (a) you think are useful.