CS41 Homework 3

This homework is due at 11:59PM on Sunday, September 24. Write your solution using LaTeX. Submit this homework in a file named `hw3.tex` using **github**. This is an individual homework. It's ok to discuss approaches at a high level. In fact, I encourage you to discuss general strategies. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab partner/group *while in lab*. In this case, note (in your post-homework survey) who you've worked with and what parts were solved during lab.

The main **learning goals** of this homework assignment are to (1) practice algorithmic design and analysis skills, and (2) get comfortable using the formal definition of big-O.

1. **College Choice.** A group of n high school seniors $S = \{s_1, s_2, s_3, \ldots, s_n\}$ are touring a set of colleges $C = \{c_1, \ldots, c_n\}$ over the course of $m \geq n$ days this autumn. Each student s_j has an itinerary where they individually decide to visit one college per day (or maybe take a day off if $m > n$). However, these students are fiercely independent and prefer not to share college campus tours with other students. Furthermore, each student is looking for one college to call their own. Each student s would like to choose a particular day d_s and stay at their current college admissions office for the remaining $m - d_s$ days of the autumn (laying claim to 100% of the admissions officer's time). Of course, this means that no other students can visit c_s after day d_s, since students don't like sharing.

   Show that no matter what the students' itineraries are, it is possible to assign each student s a unique college c_s, such that when s arrives at c_s according to the itinerary for s, all other students s' have either stopped touring colleges themselves, or s' will not visit c_s after s arrives at c_s. Describe an algorithm to find this *matching*.

   Hint: The input is somewhat like the input to stable matching, but at least one piece is missing. Find a clever way to construct the missing piece(s), run stable matching, and show that the final result solves this problem. (This hint means: **build your solution as a reduction**, using the Gale-Shapley algorithm as a black-box subroutine!)

   F.A.Q. What about ties? It may be necessary to break ties; i.e., two students might choose to visit the same college on the same day. You may assume that the tie can be broken by having students arrive at different times of the day such that if s and s' both want to visit c on the same day, that there is some timestamp on their visits such that it is easy to determine who arrived at c first. Thus, for any given day, at any given college, there is a well-defined ordering to the planned arrival time of the students.

2. **Asymptotic analysis.** Arrange the following functions in ascending order of growth rate. That is, if g follows f in your list, then it should be the case that $f(n)$ is $O(g)$.

   - $f_1(n) = n^{2.5}$
   - $f_2(n) = \sqrt{2n}$
   - $f_3(n) = n + 10$
   - $f_4(n) = 10^n$
   - $f_5(n) = 100^n$
   - $f_6(n) = \log_{1.1}(n)\sqrt[3]{n}$
   - $f_7(n) = n^n$
   - $f_8(n) = n^2 \log_2(n)$
   - $f_9(n) = n^{\log_2(n)}$

   No proofs are necessary, just give an ordering.

3. **Analysis.** Let $f(n) = 4n^{5/3}$ and $g(n) = n^{5/4}(\log n)^7$. Prove that $g(n) = O(f(n))$. You may use techniques and facts from class and the textbook; your proof should be formal and complete.

4. **Close to sorted.** Say that a list of numbers is "k-close-to-sorted" if each number in the list is less than k positions from its actual place in the sorted order. (So a 1-close-to-sorted list is *actually* sorted.) Give an $O(n \log k)$ algorithm for sorting a list of numbers that is k-close-to-sorted.

   In your algorithm, you may use any data structure or algorithm from CS35 by name, without describing how it works.

   In your analysis, you may use the CS35 analysis of any CS35 data structure or algorithm that you cited (and you do not need to recreate how it works).

   However, you *should* make sure that you analyze your algorithm and write a convincing proof that it halts, is correct, and runs in $O(n \log k)$ time. "Give an algorithm" means "... and a rigorous analysis!"

5. **Extra challenge.** Define a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ such that f is $O(g)$ for all exponential functions g, but f is *not* $O(h)$ for any polynomial function h.

6. **Extra challenge.** Is it possible to have two functions f and g such that f is not $O(g)$ and g is not $O(f)$? If so, give two example functions with this behavior; if not, prove that it is impossible for this to happen.