CS41 Homework 1

This homework is due at 11:59PM on Wednesday, September 13. Write your solution using \LaTeX. Submit this homework using github as a file called hw1.tex. (You can edit this very file!) 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 someone else while in lab. In this case, note who you’ve worked with and what parts were solved during lab.

1. **Algorithm analysis.** Consider the following algorithm for the hiking problem.

   \begin{verbatim}
   UnusualHike()
   1   k = 2.
   2   while you haven’t met your friend yet:
   3       walk k miles east
   4       return to start
   5       walk k miles west
   6       return to start
   7       k = k^2.
   \end{verbatim}

   Describe the distance traveled in UnusualHike as a function of the initial distance from your friend in the worst case. Express your answer in big-O notation. How does this algorithm compare to the algorithms we saw in class?

2. **Stable Matching Runtime.** We showed in class (at least after Friday’s class) that the Gale-Shapely Algorithm for stable matching terminates after at most \( n^2 \) iterations of the while loop.

   (a) For two sets \( A \) and \( B \) of size \( n \), can a particular list of rankings actually result in a quadratic number of iterations? If so, describe what the rankings would look like. If not, argue why no set of rankings would ever result in a quadratic number of iterations.

   \textbf{Note:} the algorithm need not take exactly \( n^2 \) iterations, but \textit{asymptotically} \( n^2 \) iterations, meaning \( 0.1n^2 \) would be sufficient to show your claim.

   (b) Can a particular set of rankings result in strictly less than a quadratic number of iterations? Can you design an input that requires \( O(n) \) iterations? If so, describe the structure of this input. If not, argue why this is not possible.

   (c) Finally, can you design an input that takes fewer than \( n \) iterations? Why or why not?

   Aim for clarity and conciseness in your write up of this problem. You should have all the necessary tools to express your solutions. You do not need formal proofs or pseudocode, but you should be able to clearly articulate your ideas in plain English.

3. **Hipster Coffee Tours.** A group of \( n \) Portland hipsters \( H = \{h_1, \ldots, h_n\} \) are touring a set of local coffee shops \( C = \{c_1, \ldots, c_n\} \) over the course of \( m \geq n \) days. Each hipster \( h_j \) has an
itinerary where he/she decides to visit one coffee shop per day (or maybe take a day off if \( m > n \)). However, hipsters are fiercely independent and prefer not to share coffee shops with other hipsters. Furthermore, each hipster is looking for a favorite coffee shop to call his or her own. Each hipster \( h \) would like to choose a particular day \( d_h \) and stay at his/her current coffee shop \( c_h \) for the remaining \( m - d_h \) days of the tour. Of course, this means that no other hipsters can visit \( c_h \) after day \( d_h \), since hipsters don’t like sharing coffee shops.

Show that no matter what the hipsters’ itineraries are, it is possible to assign each hipster \( h \) a unique coffee shop \( c_h \), such that when \( h \) arrives at \( c_h \) according to the itinerary for \( h \), all other hipsters \( h' \) have either stopped touring coffee shops themselves, or \( h' \) will not visit \( c_h \) after \( h \) arrives at \( c_h \). 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 the hipster problem.

**Additional information:** It may be necessary to break ties; i.e., two hipsters might choose to visit the same coffee shop on the same day. You may assume that the tie can be broken by having hipsters arrive at different times of the day such that if \( h \) and \( h' \) 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 coffee shop, there is a well-defined ordering to the planned arrival time of the hipsters.

4. **Assigning homework partners.** (Extra credit.) In class, we discussed a version of the stable matching problem where we want to match \( n \) doctors to \( n \) hospitals. In this problem, we discuss the homogeneous version. The input is a set of students \( A = \{s_1, \ldots, s_{2n}\} \) of size \( 2n \). Each student ranks the others in order of preference. A homework partner assignment of students into partners \( M = \{(i, j)\} \) is a matching; it is unstable if there exists \((i, j), (i', j') \in M\) such that \( i \) prefers \( j' \) to \( j \) and that \( j' \) prefers \( i \) to \( i' \). It is stable if it is a perfect matching and there are no instabilities.

(a) Does a homework partner assignment always exist? Prove that such an assignment must always exist, or give an example where no stable assignment occurs. (Remember, you must have \( 2n \) students.)

(b) Design and analyze an efficient algorithm that either returns a stable matching for homework partners or outputs that no such matching exists.