COMPSCI 311: Introduction to Algorithms **Fall 2019**

Homework 1

Released 9/9/2019 Due 9/23/2019 11:59pm in Gradescope

Instructions. You may work in groups, but you must write solutions yourself. List collaborators on your submission. Also list any sources of help (including online sources) other than the textbook and course staff.

If you are asked to design an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

1. **(15 points) Stable Assignments with Multiple Students**

   A more realistic scenario is when a college accepts multiple students. We have n colleges, each with some number of study places, and m students, exceeding the total number of places (thus, some students will not not get a place). Like before, each college and each student has a preference list.

   (a) Given an assigment of students to colleges, define what an instability is in this problem case.

   (b) Show that there always is an stable assignment of student to colleges, where every college fills all its slots, and give an algorithm to find one.
   You may reuse or adapt given proofs but make sure your arguments are clear and complete.

2. **(20+5 points) Latin Squares and Stable Matching.** A **Latin square** is an $n \times n$ array of numbers, each in the set $\{1, \ldots, n\}$, such that no number appears twice in any row or any column. (Thus a completed Sudoku puzzle is a 9×9 Latin square that also obeys some additional rules.)

   If we have a preference list where each college c_i ranks the n students, we can represent this list by an $n \times n$ array where the i'th row represents the preferences of college c_i. Another $n \times n$ array can represent the preferences of the n students.

   (a) Suppose that the preference list of the colleges is a Latin square. What is the largest and smallest number of proposals in a run of the Gale-Shapley algorithm (with colleges proposing) under this condition? (We assume nothing about the student preferences.) If you can't get the exact number of proposals as a function of n, find a function f such that you can show the number to be $\Theta(f)$.

   (b) Now suppose that the preference list of the students is a Latin square. Find the exact minimum possible number of proposals in any run of the Gale-Shapley algorithm (with colleges proposing) under this condition? (We assume nothing about the college preferences.) This means finding a function f such that for any n, there is a run with exactly $f(n)$ proposals and no run with fewer than $f(n)$ proposals.

   (c) Find a function g such that the *maximum* number of proposals on a run of Gale-Shapley with the student preference lists a Latin square and the colleges proposing is $\Theta(g)$.

   (d) (**extra credit**) Find the *exact* maximum number of proposals possible under this condition, with the colleges proposing. This means finding a function g so that there is a run with $g(n)$ for every n, and no run ever has more than $g(n)$.

3. **(15 points) Big-O.** For each function $f(n)$ below, find (1) the smallest integer constant H such that $f(n) = O(n^H)$, and (2) the largest positive real constant L such that $f(n) = \Omega(n^L)$.
   Otherwise, indicate that H or L do not exist. All logarithms are with base 2. Your answer should consist of: (1) the correct value of H, (2) a proof that $f(n)$ is $O(n^H)$, (3) the correct value of L, (4) a proof that $f(n)$ is $\Omega(n^L)$.

   (a) $f(n) = \frac{n(n+1)(2n+1))}{6}$.
   (b) $f(n) = \frac{n \log n}{\log(\log n)}$
   (c) $f(n) = \sum_{i=0}^{\lceil \log n \rceil} n(0.99)^i$.
   (d) $f(n) = \sum_{i=1}^{n} i^4$.
   (e) $f(n) = 2^{\log(n!)}$

4. **(15 points) Asymptotics.** A Latin square is defined in Question 1 above. In this question we look at determining whether a given $n \times n$ array A of numbers from $\{1, \ldots, n\}$ is a Latin square. Here is a pseudocode algorithm that solves this problem:

```
for (i = 1 to n)
     for (j = 1 to n)
          for (k = j+1 to n)
              if (A(i, j) = A(i, k)) return false;
for (k = 1 to n)
     for (i = 1 to n)
          for (j = i+1 to n)
              if (A(i, k) = A(j, k)) return false;
return true;
```

   (a) Prove, using the definition, that this algorithm returns true if and only if A is a Latin square.
   (b) Find a function f such that the running time of this algorithm is $O(f(n))$, and clearly explain why.
   (c) For the same function f argue that the running time of the algorithm is also $\Omega(f(n))$. (This establishes an asymptotically tight bound $\Theta(f(n))$.)
   (d) Design and analyze a faster algorithm for this problem. You should give an algorithm with running $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

5. **(20 points) Weak connections. K&T Ch 3, Ex 9.** There's a natural intuition that two nodes that are far apart in a graph—separated by many hops—have a more tenuous connection than two nodes that are close together. A number of algorithmic results make this notion precise in different ways. Here's one that involves the susceptibility of paths to the deletion of nodes.

   Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all $s - t$ paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running time $O(m + n)$ to find such a node v.

6. **(15 points) BFS and DFS Trees** Consider an undirected connected graph $G = (V, E)$. Find a necessary and sufficient condition for G so that it is possible to order the adjacency lists (neighbors) of every node so that for every node v, the BFS tree and DFS tree from v are different. Design an algorithm that checks this condition.

7. **(0 points).** How long did it take you to complete this assignment?