CSE 417 25au Homework 2: Gale–Shapley/Running time

Released: Friday, October 3, 2025 @ 11:30am

First due by: Friday, October 10, 2025 @ 11:59pm

Last resubmissions by: Wednesday, October 22, 2025 @ 11:59pm

Instructions

For Problems 3, 3X.1, and 4, you have four options for submission:

- **Film a video in which you explain your solution.** See the Homework Guide for more details.
- **Use LaTeX to type your solutions.** A template is provided in the "Tasks" page of the course website, if you like.
- **Use Google Docs or Microsoft Word to type your solutions.** If doing so, please use the Equation Editor to ensure that any equations are legible and easy to read.
- **Handwrite your solutions on paper or digitally.** Please write neatly and if on paper, scan in black/white mode, not grayscale.

We prefer either video or LaTeX, but accept any of the 4 options.

A few more reminders:

- **Submit all problems on Canvas.** Each problem should have its own submission. *Do not* submit one large file containing answers to several problems.
- **Suggested word counts are rough guidelines.** We won't actually count, but if your writing is verbose to the point of obscuring your main argument, we may ask you resubmit more concisely.
- **Review the collaboration policy in the syllabus**. Collaboration is encouraged but strict rules apply, and remember to cite your collaborators.
- If you don't finish in time, we encourage you to be honest and just upload what you have so far. Resubmission won't cost you anything, and we can give you timely feedback on your partial progress by submitting on time.

Happy problem solving!

Problem 3: Medical residency matching

The purpose of this problem is to apply ideas from the Gale–Shapley algorithm to new situations and practice analyzing running time.

In class, we mentioned that a version of the Gale–Shapley algorithm is actually currently in use by the National Resident Matching Program, which matches newly graduated medical students (residents) to hospitals where they can work. This setting is different from the original Gale–Shapley setup in a few ways.

- The number of residents n is different from the number of hospitals m.
- Each hospital may take multiple residents (and at least one).
- Residents and hospitals do not rank the entire other list. In fact, the number of entities ranked in practice does not really depend on how many of the other entity there are. (Would you have applied to more colleges if the US had twice as many colleges? Probably not.) Thus, consider the length of each preference list, as well as the capacity of each hospital, to be a constant independent of n and m.

A resident-hospital pair (r, h) is called *unstable* if the following three things are all true:

- r and h rank each other somewhere on their preference lists,
- r is either unmatched or prefers h to their current match, and
- h either has extra capacity or prefers r to at least one of its current matches.

The goal is to output a list of pairs (r, h) so that no resident r is matched with more than one hospital, no hospital h is matched with more residents than it has capacity for, and there are no unstable pairs. The modified algorithm to compute this is as follows:

```
1:  while there is a free resident r who has not yet applied to their whole list do
2:      Have r apply to the next hospital on their preference list, call it h.
3:      if h ranked r somewhere and is not at capacity then
4:          Have h admit r.
5:      else if h is at capacity but prefers r over one of its current admits then
6:          Let r′ be the current admit to h that h likes the least.
7:          Have h admit r and reject r′.
8:      else
9:          Have h reject r.
10: return list of all matches
```

Much like the pseudocode above, the original Gale–Shapley pseudocode discussed in class was also intentionally vague about how certain operations are performed, in order to focus on understanding the big picture of the algorithm. We discussed how to design data structures that flesh out precisely how the computer can do each step efficiently, achieving a $O(n^2)$ running time, when there were n proposers and n receivers.

Describe the data structures of a reasonable input format for this setting, as well as what additional data structures you would use in your algorithm to implement the pseudocode efficiently. Analyze the running time per iteration and the overall running time in terms of n and m. An "E" score requires the best overall running time possible, an "S" score does not. Do not use hash tables/sets. (suggested 250–500 words or 4–8 minutes video)

Problem 3X: Medical residency matching (Extensions)

This is an extension problem that builds on the ideas of Problem 3. **Pick one** of the following to complete:

1. Prove a full proof of correctness of this modified Gale–Shapley algorithm. (But for brevity, feel free to state simple loop invariants that you use without proof.) Note that your proof should be based on the pseudocode provided, *not* your implementation in Problem 3, which is too detailed and will make the proof too complicated. (suggested 200–400 words or 4–8 minutes video)

2. Read the following report on the results of the 2025 Main Residency Match by the National Resident Matching Program. Then synthesize your thoughts on this report with properties that you know about the Gale–Shapley algorithm and join the discussion on Canvas by responding to the following prompt, raising other questions of your own choosing, or replying directly to other students' responses. (suggested 150–300 words)

   > What are the benefits and drawbacks of using this algorithm? What are some considerations that the algorithm fails to accommodate? In what aspects is this algorithm better or worse than other systems used to match people to institutions, such as college admissions, fraternity/-sorority rush, job applications, or other systems that you may be familiar with?
   >
   > Alternatively, discuss this algorithm with a friend who is planning to become (or already is) a medical doctor, and report back on their thoughts. Does the algorithm feel fair to them? Do they wish it was different in any way?

   You are *not* required to cite the article or any other sources, but be sure to use some theoretical features and guarantees of the Gale–Shapley algorithm as supporting evidence. If you discuss any features or theorems regarding Gale–Shapley that we did not cover in class, please cite your sources. **Resubmissions will not be available for this part.**

Problem 4: Long division

The purpose of this problem is to practice effective communication of more complicated algorithms, and reinforce concepts about arbitrary precision arithmetic.

In elementary school, you likely learned about the long division algorithm, which takes two positive integers a and b, and computes both their integer quotient q and the remainder r. More concretely, the goal is to find positive integers q and r that satisfy $a = bq + r$ and $0 \leq r < b$.

1. Think of all the positive integers in this problem (a, b, q, and r) as lists of digits 0–9. Concisely describe in pseudocode the long division algorithm. Assume that we already know how to perform other operations on positive integers such as addition, subtraction, multiplication, comparison, etc. (suggested 6–12 lines)

   Be sure to describe an algorithm mimicking the *long division* algorithm, typically taught in US schools using notation that looks something like $37\overline{)2749}$. Do *not* describe division via simple repeated subtraction.

2. Suppose a has n digits and b has m digits. Analyze the running time of your algorithm in terms of n and m. Analyze the running time as tightly as possible, but you may assume that n and m are somewhat different in size.

   Depending on your pseudocode, this may require you to discuss the number of digits in intermediate variables compared to n and m, which may be justified using loop invariants. If that is the case, state (but no need to prove) the loop invariants. Additionally, if your pseudocode contained any sentences whose implementation was not fully spelled out, be sure to describe which implementation is being used in your analysis. (suggested 50–150 words)

3. Find the main loop invariant(s) that would allow you to prove your algorithm correct. State the invariant(s). No need to write a proof for this part (but you should think through it, so that you are confident that your invariant(s) is/are true and sufficient to prove correctness).

4. (Optional, not for credit) When $n \approx m$, for instance $n = m + 1$ (think 781942 divided by 12458), the number of iterations of long division can be greatly reduced. In particular, this example should only take 2 iterations. If your pseudocode in part 1 is written to take a long time for this case, briefly describe updated pseudocode that handles this case efficiently. Then analyze the running time in terms of n and m tightly in a single expression (meaning no cases), such that when $n = m + c$ for a constant c, the running time will simplify to $O(n)$.

(Video submissions are suggested to take 3–6 minutes total.)