Problem 1 (20 pts)

Prove or disprove:

a) (10 pts) For any instance of the stable matching problem, the following holds: In **every** stable matching, there is an **agent** (a company or an applicant) who gets their first (most favorite) choice.

Hint: If correct, you need to give a proof (be sure to show the claim for every instance and for every stable matching!). If incorrect, give a counter-example, i.e., an instance and a stable matching (be sure to justify that the matching is stable!). In this case, only one counterexample is enough.

Answer: This statement is *false*.

Counterexample. Consider a stable matching problem with three companies and three applicants, with the following preferences:

- Companies (C) and their preferences over Applicants (A):

$$C_1 : A_1 > A_2 > A_3 \quad C_2 : A_2 > A_3 > A_1 \quad C_3 : A_3 > A_1 > A_2.$$

- Applicants (A) and their preferences over Companies (C):

$$A_1 : C_2 > C_3 > C_1 \quad A_2 : C_3 > C_1 > C_2 \quad A_3 : C_1 > C_2 > C_3.$$

Consider the stable matching

$$M = \big\{(C_1, A_2), (C_2, A_3), (C_3, A_1)\big\}.$$

One can verify:

- No company or applicant gets their top choice in this matching.
- To check stability, make sure that there is no pair (C_i, A_j) that prefers each other over their current partners. We can confirm that there are no unstable pairs.

Therefore, M is stable, yet no one receives their first choice. Therefore, the statement "in the *every* stable matching, someone must get their first choice" is contradicted.

b) (10 pts) For every instance of the stable matching problem, the following holds: There **exists** a stable matching in which a company gets their first (most favorite) choice.

Answer: This statement is *true*.

Sketch of proof. Consider an instance where the company proposes. We know that it ends in a stable match. We claim that in *that* stable matching, at least one company is matched with its top choice. Intuitively:

- If a company never ends up with its top choice, that means that each time it proposes to that top-choice applicant, the applicant "rejects" it in favor of some other company.
- However, by carefully tracing who can "win" the top choice applicant (when every applicant has a strict total order), we see that some top-choice applicant must remain matched to a company that this applicant prefers not to reject.
- The details show that at least one company does indeed pair with its top choice in that final stable matching.

Therefore, while (a) is false for "every stable matching," (b) is true for "there exists a stable matching."

Problem 2 (10 pts)

Prove that in every instance of the stable matching problem with n companies and n applicants, companies make at most $n(n-1)+1$ proposals in the Gale-Shapley algorithm.

Hint: Prove that in the Gale-Shapley algorithm (when companies propose), at most one company gets its last choice.

Proof: We claim that the total number of proposals in the Gale-Shapley algorithm (when companies propose) is at most $n(n-1)+1$. We break the argument into two parts:

1. **No company can make more than n proposals.**
   Each company ranks the n applicants in a strict total order and never proposes to the same applicant twice. So, each company has at most n distinct applicants to propose to.

2. **At most one company can use all n of its proposals.**
   Suppose that two different companies, C and C', each end up proposing to all n applicants. By the time C is forced to propose to its $(n-1)$-th and then n-th choice, it must have been repeatedly rejected on its top $(n-1)$ applicants. But each rejection of C means that the applicant in question is already "holding" a proposal from a *more preferred* company.

   Every time C is rejected, there is a chain of companies more preferred by that rejecting applicant that are still engaged at that moment. If C' is also exhausting its preference list, it too would be repeatedly rejected by $(n-1)$ different applicants in favor of other companies (each time implying that those applicants are matched to strictly preferred partners). Eventually, one would reach a contradiction in the sense that two different companies are each being outbid for *all* of their first $(n-1)$ choices, creating an impossible cycle.

   Therefore, it is impossible for two or more companies to propose to all n applicants. At most only one company can run through its entire list.

3. **Conclusion:**
   Combining these two pieces, every other company (besides at most one) makes at most $(n-1)$ proposals. Therefore, the total number of proposals is at most

$$(n-1)\cdot(n-1) \;+\; n \;=\; n(n-1) \;+\; 1.$$

This proves that the company-proposing Gale-Shapley algorithm makes at most $n(n-1)+1$ proposals in total.

Problem 3 (10 pts)

Use induction to solve this problem: Given 2^k real numbers $x_1, \ldots, x_{2^k}$ such that $\sum_{i=1}^{2^k} x_i = 1$, show that:

$$\sum_i x_i^2 \geq \frac{\left(\sum_i x_i\right)^2}{2^k} = \frac{1}{2^k}.$$

Hint: You can use the following inequality: For any two real numbers a, b,

$$(a+b)^2 \leq 2a^2 + 2b^2.$$

Proof:

Base case: $k = 1$

Then we have $2^1 = 2$ numbers x_1, x_2 such that $x_1 + x_2 = 1$. We need to show:

$$x_1^2 + x_2^2 \;\geq\; \frac{1}{2}.$$

Set $x_2 = 1 - x_1$. Then

$$x_1^2 + (1-x_1)^2 = x_1^2 + 1 - 2x_1 + x_1^2 = 2x_1^2 - 2x_1 + 1.$$

This is a quadratic in x_1 with a minimum at $x_1 = \frac{1}{2}$, yielding $x_1^2 + x_2^2 = \frac{1}{2}$. Thus

$$x_1^2 + x_2^2 \;\geq\; \frac{1}{2}.$$

So the base case holds.

Induction

Assume the statement is true for 2^k numbers. Now take 2^{k+1} numbers $x_1, \ldots, x_{2^{k+1}}$ summing to 1. Pair them up in (a_i, b_i) for $i = 1, \ldots, 2^k$. Let $y_i = a_i + b_i$. Then $\sum_i y_i = 1$ and there are 2^k of the y_i's.

By the inductive hypothesis,

$$\sum_{i=1}^{2^k} y_i^2 \;\;\geq\;\; \frac{1}{2^k}.$$

Using the hint $(a+b)^2 \leq 2(a^2+b^2)$, we get

$$y_i^2 = (a_i + b_i)^2 \;\leq\; 2(a_i^2 + b_i^2).$$

Thus,

$$\sum_{i=1}^{2^k} y_i^2 \;\leq\; 2\sum_{i=1}^{2^k}(a_i^2 + b_i^2) \;=\; 2\sum_{j=1}^{2^{k+1}} x_j^2.$$

Therefore,

$$\sum_{j=1}^{2^{k+1}} x_j^2 \;\geq\; \frac{1}{2}\sum_{i=1}^{2^k} y_i^2 \;\geq\; \frac{1}{2}\cdot\frac{1}{2^k} = \frac{1}{2^{k+1}}.$$

Problem 4 (10 pts)

Let $I = (P, R)$ be an instance of the stable matching problem. Suppose that the preference lists of all $p \in P$ are identical, so without loss of generality, p_i has the preference list $[r_1, r_2, \ldots, r_n]$. Prove that there is a unique solution to this instance. Describe what the solution looks like, and why it is the only stable solution.

Note: Showing that the solution is the one found by the Gale-Shapely algorithm is not sufficient, as there could be other solutions.

Proof Let the preference list (common to all proposers) be $[\, r_1, r_2, \ldots, r_n \,]$. We claim:

1. In any stable matching, r_1 (the universally top receiver) must be matched to the proposer that r_1 prefers most, or else that proposer and r_1 would form a blocking pair.

2. Then remove r_1 and that chosen proposer from the instance, and repeat the same argument for r_2 among the *remaining* proposers.

3. This process continues deterministically. There is essentially no freedom for any other pairing, because any "deviation" would create a blocking pair (the unmatched pair would always prefer each other over their current matches).

Therefore the stable matching is forced and *unique*. It specifically pairs r_1 with whomever r_1 ranks highest among all p_i, then r_2 with her top choice among the *remaining* proposers, etc. Because all proposers rank r_1 highest, no one can "safely" deviate without creating a blocking pair. This one forced chain of matches is the only stable outcome.

Problem 5 (15 bonus pts)

Implement the Gale-Shapley algorithm to solve the Stable Matching Problem using $O(n^2)$ time complexity. You are free to write in any programming language you like. Make sure that you test your algorithm on small instance sizes, where you are able to check results by hand. Run your algorithm on the following testing instance of size $n = 4$ (define the algorithm as a function and give preference matrices as function parameters). Give the resulting matching that is found, along with the list of proposals performed by the algorithm. Submit your source code file.

- **Proposers (P):** $P = \{1, 2, \ldots, n\}$
- **Receivers (R):** $R = \{1', 2', \ldots, n'\}$
- **Preference profiles:**
  - Each proposer has a ranked list of receivers.
  - Each receiver has a ranked list of proposers.

Function definition: `gale_shapley(P_pref, R_pref)`

Input:

- Two preference matrices:
  - `P_pref[p][i]`: i-th preferred receiver of proposer p.
  - `R_pref[r][i]`: i-th preferred proposer of receiver r.

Output:

- `match_P[p]`: The receiver matched to proposer p.
- `match_R[r]`: The proposer matched to receiver r.

Testing Input:

$$\texttt{P\_pref} = \begin{bmatrix} 3 & 2 & 4 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\texttt{R\_pref} = \begin{bmatrix} 1 & 3 & 2 & 4 \\ 3 & 1 & 4 & 2 \\ 4 & 3 & 2 & 1 \\ 3 & 4 & 2 & 1 \end{bmatrix}$$

Note: Refer to the lecture slides of week 2, pages 10-11, on the details of efficient implementation. You may switch to 0-based indices if necessary.