1. A five step process to design and analyze an algorithm.

Context

- You encounter a new problem
  - In a course, on a job interview, or while working.
  - The problem statement might be vague.
    - Assign coop students to employers in a way that will not end up in a student getting fired so the employer can hire another student.
  - You are asked to design an efficient algorithm to solve it.

Step 1: building intuition

- Write down some problem instances
  - trivial ones where the answer can be found without any computations.
  - small ones which are not completely trivial but whose answers you can compute manually.
    - e1: a1, a2   a1: e2, e1
      e2: a1, a2   a2: e1, e2
- Write down solutions for these instances
  - you will use these instances to test ideas.
    - (e1, a2), (e2, a1)

Step 2: Specify the problem formally

- Develop notation to describe
  - An instance of the problem
    - A list $E = \{ e_1, ..., e_n \}$ of employers.
    - A list $A = \{ a_1, ..., a_n \}$ of n applicants.
    - For each employer, a permutation $P[e_i]$ of A.
    - For each applicant, a permutation $P[a_j]$ of E.
  - A valid solution
    - A list of of n pairs $(e_{i(k)}, a_{j(k)})$ where each employer and applicant appears exactly once.

o Nothing ridiculous like 1 employer : many employees.
or vice versa

Step 2: Specify the problem formally

- Explain what makes a solution good
  - There do not exist pairs $(e_{i(k)}, a_{j(k)})$ and $(e_{i(k')}, a_{j(k')})$ where
    - $e_{i(k)}$ comes before $e_{i(k')}$ in $P[a_{j(k')}]$.
    - $a_{j(k')}$ comes before $a_{j(k)}$ in $P[e_{i(k)}]$.
  - Why?
    - $e_{i(k)}$ would fire $a_{j(k)}$ and make an offer to $a_{j(k')}$.
    - $a_{j(k')}$ would leave $e_{i(k')}$ to go work for $e_{i(k)}$.

→ no applicants poached

instability

→ No employers poaching

Step 3: Identify similar problems

- Try to find problems that
  - look similar to the new problem you are given
  - you already know how to solve.
- You might be able to adapt their solution to get a solution to your new problem.

Step 4: Evaluate simple solutions

→ Don't over complicate.

- Look at really simple solutions, such as brute force. → check every single sol^n (brute force) → eventually find sol^n
  - Most likely they are too slow.
    - There are n! ways to pair n employers and applicants.
  - But you never know.
  - And sometimes you can modify them to get a more efficient solution. → can tweak

- Not efficient enough → or design better

Step 5: Design a better algorithm

- Depends on problem → cannot solve all..

- Try possible ideas on your trivial and small examples from step 1. → if yes, then there is hope
  ↳ otherwise try again.
- Once you have one that appears to work,
  - Prove its correctness (make sure it doesn't work only on your examples from step 1!)
  - Analyze its time and space requirements.

- hard, but we'll learn