

Ruolin Li 31764160

Step 5: Design a better algorithm.

1. Brainstorm some ideas, then sketch out an algorithm.

Try out your algorithm on some examples.

You may have lots of ideas. For example, you might have noticed that if a employer and a applicant both most-prefer each other, we must match them; that might form the kernel of some kind of algorithm. For our algorithm, we'll work with the Gale-Shapley algorithm, also in textbook and on part b of this worksheet.

```
1: function Gale-Shapley( $n, P_E, P_A$ )
2:    $t \triangleright n \geq 1$  is the number of employers and also the number of applicants
3:    $t \triangleright P_E$  is the collection of complete preference lists ( $>_e$ ) of the employers
4:    $t \triangleright P_A$  is the collection of complete preference lists ( $>_a$ ) of the applicants
5:    $t \triangleright$  return a stable matching  $M$  for the stable matching instance  $(n, P_E, P_A)$ 
6:
7:    $M \leftarrow \emptyset$   $t \triangleright$  matching  $M$  is initially empty
8:   while some employer is unmatched and has not considered every applicant do
9:     choose any such employer  $e$ 
10:    let  $a$  be the highest-ranked in  $e$ 's list that  $e$  has not yet considered
11:     $t \triangleright e$  now considers  $a$  (i.e., "makes an offer" to  $a$ ) as follows:
12:    if  $a$  is unmatched then
13:      add match  $(e, a)$  to  $M$   $t \triangleright a$  accepts  $e$ 's offer
14:    else  $t \triangleright a$  is matched
15:      let  $a'$  be currently matched to  $a$ 
16:      if  $e >_a a'$  then  $t \triangleright a$  prefers  $e$  to  $a'$ 
17:        remove match  $(a', a)$  from  $M$   $t \triangleright a$  rejects  $a'$ 's offer
18:        add match  $(e, a)$  to  $M$   $t \triangleright a$  accepts  $e$ 's offer
19:      else  $t \triangleright a$  prefers  $a'$  to  $e$ , in which case  $M$  does not change
20:  return  $M$ 
```

2. Analyze the running time of your algorithm.

Using big- O notation, can you bound the number of iterations of the while loop? How much time does each iteration take? Is your bound tight in the worst case?

Step 5.
 $O(n^2)$ since every employers will send an ^{offer up to} n times in the worst case # of possible offers
 $O(1)$ for each iteration from employers to applicants
 $\Theta(n^2)$ worst case run time.

Comparing Orders of Growth for Functions

We'll take a break from Gale-Shapley, and turn to comparing algorithm runtimes. Imagine that each of the functions below represents the runtime of some algorithm on instances of size n . Give a good Θ bound for each, and then arrange these functions by increasing order of growth (fastest algorithm to slowest). Notation that we use here and throughout the course:

$$\lg n = \log_2 n, \ln n = \log_e n, \text{ and } \log n = \log_{10} n.$$

$$\begin{array}{ll} n + n^2 & 2^n \\ 55n + 4 & 1.5n \lg n \\ n! & \frac{\ln n}{n} \\ 2n \log(n^2) & \log n \\ (n \lg n)(n+1) & (n+1)! \end{array}$$

$$1.6^{2n}$$

tricky, but doable!

Compare Orders of Growth for Functions. log n grows slower than n.

$n + n^2$	$\Theta(n^2)$	2^n	$\Theta(2^n)$
$55n + 4$	$\Theta(n)$	$1.5n \lg n$	$\Theta(n \lg n) / \Theta(n \lg n)$
$n!$	$\Theta(n!)$	$\frac{n}{\log n}$	$\Theta(\frac{n}{\log n}) / \Theta(\frac{n}{\log n})$
$2n \log(n^2) = 4n \log n$	$\Theta(n \lg n) / \Theta(n \lg n)$	$(n+1)!$	$\Theta(n \cdot n!)$
$(n \lg n)(n+1)$	$\Theta(n^2 \lg n) / \Theta(n^2 \lg n)$	$\ln n$	$\Theta(\log n)$
1.6^{2n}	$\Theta(2.56^n)$		

$\ln n < \frac{n}{\log n} < 55n + 4 < 2n \log(n^2) = 1.5n \lg n < n + n^2 < (n \lg n)(n+1) < 2^n < 1.6^{2n} < n! < (n+1)!$

Now, back to analysis of the Gale-Shapley algorithm

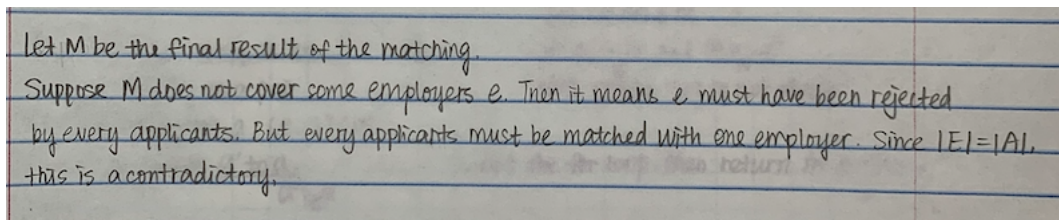
3. Show that your algorithm is correct.

It's always good to start by writing down what it means for the algorithm to be correct. For the SMP problem, it means that the algorithm outputs a perfect matching with no instabilities.

- **Show that the output is a perfect matching.**

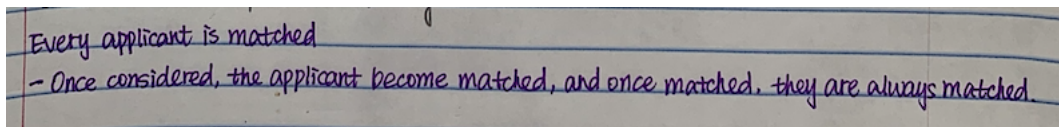
We'll break this down further:

- Every applicant has been considered at least once upon termination.



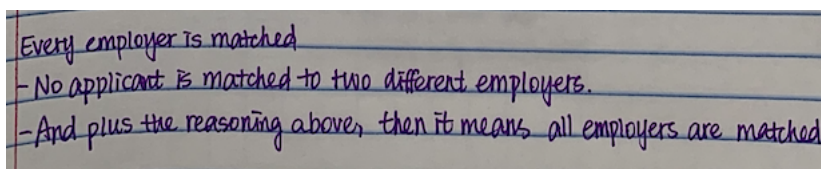
Let M be the final result of the matching.
Suppose M does not cover some employers e . Then it means e must have been rejected by every applicants. But every applicants must be matched with one employer. Since $|E| = |A|$, this is a contradictory.

- Every applicant is matched upon termination.



Every applicant is matched
– Once considered, the applicant become matched, and once matched, they are always matched.

- Every employer is matched upon termination.



Every employer is matched
– No applicant is matched to two different employers.
– And plus the reasoning above, then it means all employers are matched

- **Show that the output has no instabilities.**

Recall that a pair (e, a') is an instability if (e, a) and (e', a') are distinct pairs in M , and also e prefers a' to a , and a' prefers e to e' . That is, e and a' would prefer to be matched with each other than with their matches in M .

One natural approach is to show that the (partial) matching constructed after each iteration of the While loop avoids instabilities. Let M_k be the matching at the end of iteration $k \geq 0$ of the While loop, and let E_k and A_k be the set of employers and the set of applicants, respectively, that are matched in M_k . Show by induction that there is no instability in $E_k \times A_k$ with respect to M_k .

Claim: There is no instability in $E_k \times A_k$ with respect to M_k .

Base case:

Inductive step:

Challenge problems

These are just for fun, some are easier than others.

1. Design an algorithm to generate each possible perfect matching between n employers and n applicants. (As always, it will help tremendously to start by giving your algorithm and its parameters names! Your algorithm will almost certainly be recursive.)
2. A "local search" algorithm might pick a matching and then "repair" instabilities one at a time by matching the pair causing the instability and also matching their partners. Use the smallest possible instance to show how bad this algorithm can get.
3. Design a scalable SMP instance that forces the Gale-Shapley algorithm to take its maximum possible number of iterations. How many is that? (A "scalable instance" is really an algorithm that takes a size and produces an instance of that size, just like the "input" in worst case analysis is scalable to any n .)