Analysis of Stable Matching

CS/MCS 401: Computer Algorithms I
Lecture 3
Professor: Jonathan Liu

Sit next to someone! If you don't do it now, you'll have to move later.

Introduce yourself!
Icebreaker: What's the last movie you watched? Would you recommend it?

Lesson Goals

Today we'll continue our analysis of the stable matching algorithm. We'll do so two ways:

- First, we'll talk about **runtime analysis** generally, and **analyze the runtime** of the algorithm.
- Then, we'll prove some **interesting properties of outcomes** with this algorithm, and use that to begin a discussion about algorithmic bias.

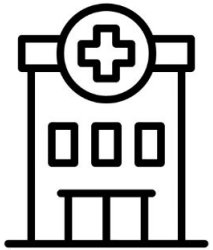
Stable Matching Recap

National Resident Matching Program (NRMP)

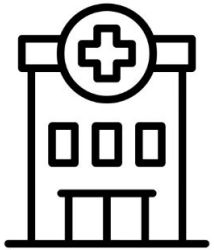
Newly-graduated doctors need to be matched to residency programs.

Doctor 0

Doctor 1



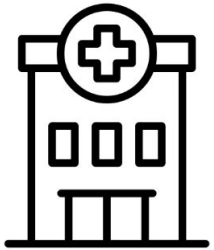
Hospital A



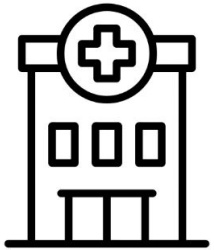
Hospital B

Doctor 2

Doctor 3



Hospital C



Hospital D

National Resident Matching Program (NRMP)

A > B > C > D

Doctor 0

A > C > B > D

Doctor 1

A > C > D > B

Doctor 2

B > A > C > D

Doctor 3

A **matching** is a set of (doctor, hospital) pairs where each entity is in exactly one pair.

The goal is, of course, to find a "good" matching.

1 > 3 > 0 > 2

Hospital A

1 > 2 > 0 > 3

Hospital B

3 > 1 > 2 > 0

Hospital C

2 > 3 > 0 > 1

Hospital D

Gale-Shapley Algorithm
(David Gale and Lloyd Shapley, 1962)

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

A matching is **stable** if there is no pair that prefers each other over their current assignments.

Proof that the returned matching is stable (worksheet)

Assume, for contradiction, that the returned matching is not stable. Then there exists some pair *(d, h)* that is blocking. For this pair to not be in the final matching, it either was never requested by *h*, or **it was rejected by *d***.

If it was never requested, then by line **4**, *h* has only made requests to doctors it prefers over *d*, so *h* must prefer the doctor it is matched with, so this is not a blocking pair.

If *(d, h)* was requested but is not in the final matching, then by line **8**, **the doctor has an offer it prefers**, so it is not possible for this to be a blocking pair.

Thus, it is impossible for there to exist a blocking *(d, h)*.

Runtime Analysis

Our Language for Algorithm Speed: Big-O Notation

When discussing runtime in this course, we follow a strict set of rules.

- Time complexity describes how the algorithm's runtime scales with input size.
- We consider polynomial time and better to be "efficient".
- We are (mostly) interested in worst-case time complexity.
- We ignore constant factors and lower-order terms.

Polynomial Time Complexity

Time Complexity: How does **the number of steps to run the algorithm** scale with the size of the input n?

An algorithm is **poly-time** if there are some constants b and c such that, for an input of size n, the algorithm uses

$\leq c \cdot n^b$ steps.

For this class, **poly-time algorithms are considered "efficient."**

In practice, this is a really good rule, but it's of course not perfect. After all, what if an algorithm requires $n^{9999999}$ steps to run?

Big-O Notation

Big-O: We say that f(n) is *O(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

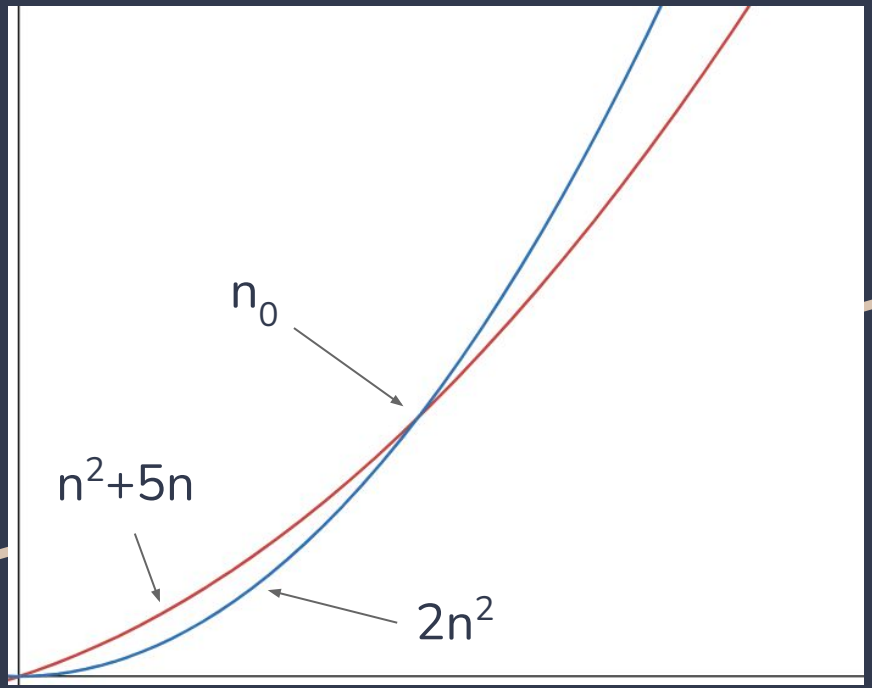
$$\mathbf{f(n) \leq c \cdot g(n).}$$

Big-O represents a **long-term upper bound** on f(n).

For example, if $f(n)=8n^3$, we can say that f(n) is $O(n^3)$.

An algorithm is *poly-time* if, for some constant d, its runtime is $O(n^d)$.

Big-O Notation



Big-O: We say that f(n) is *O(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

$f(n) \leq c \cdot g(n)$.

Consider the function $f(n) = n^2+5n$.

For small n, $n^2+5n > 2n^2$. But, once $n > 5$, $2n^2$ is larger, and remains larger.

So, $f(n) = n^2+5n$ is in $O(n^2)$.

Takeaway: Big-O notation allows you to ignore the lower-order terms! In the long-run, only the highest-degree term matters.

Big-O Notation Practice (Worksheet)

Big-O: We say that f(n) is *O(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

$f(n) \leq c \cdot g(n)$.

Find the Big-O bound for each of these runtimes. Then, sort them from fastest (1) to slowest (5).

Runtime	Big-O	Rank
$9n^3+7n^2+n^8$		
$14n+29$		
$n \log n + 3n$		
$3^n + 7n^9$		
123456789		

Big-O Notation Practice (Worksheet)

Big-O: We say that f(n) is *O(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

$f(n) \leq c \cdot g(n)$.

Find the Big-O bound for each of these runtimes. Then, sort them from fastest (1) to slowest (5).

Runtime	Big-O	Rank
$9n^3+7n^2+n^8$	$O(n^8)$	
$14n+29$	$O(n)$	
$n \log n + 3n$	$O(n \log n)$	
$3^n + 7n^9$	$O(3^n)$	
123456789	$O(1)$	

Big-O Notation Practice (Worksheet)

Big-O: We say that f(n) is *O(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

$f(n) \leq c \cdot g(n)$.

Find the Big-O bound for each of these runtimes. Then, sort them from fastest (1) to slowest (5).

Runtime	Big-O	Rank
$9n^3+7n^2+n^8$	$O(n^8)$	4
$14n+29$	$O(n)$	2
$n \log n + 3n$	$O(n \log n)$	3
$3^n + 7n^9$	$O(3^n)$	5
123456789	$O(1)$	1

Big-O Notation and its Cousins

Big-O: We say that f(n) is O*(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, **for all $n > n_0$,**

$$\mathbf{f(n) \leq c \cdot g(n).}$$

Note that Big-O is an upper bound: an algorithm that runs in n^2 steps is technically $O(n^{200})$.

Big-Ω: We say that f(n) is Ω*(g(n))* if there exist constants $c > 0$ and $n_0 \geq 0$ such that, for all $n > n_0$,

$$f(n) \geq c \cdot g(n).$$

Big-Θ: We say that f(n) is Θ*(g(n))* if **f(n) is O*(g(n))* *and* f(n) is Ω*(g(n))***.

In this course, we'll ask for the lowest possible Big-O bound, which is often the same as Big-Θ.

Common Runtimes and Real-Life Implications

If an computer performs a million operations per second, how large of an n can an algorithm with this runtime handle?

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	$2^{6 \cdot 10^7}$	$2^{36 \cdot 10^8}$	$2^{864 \cdot 10^8}$	$2^{25920 \cdot 10^8}$	$2^{315360 \cdot 10^8}$	$2^{31556736 \cdot 10^8}$
$\sqrt{n}$	10^{12}	$36 \cdot 10^{14}$	$1296 \cdot 10^{16}$	$746496 \cdot 10^{16}$	$6718464 \cdot 10^{18}$	$994519296 \cdot 10^{18}$	$995827586973696 \cdot 10^{16}$
n	10^6	$6 \cdot 10^7$	$36 \cdot 10^8$	$864 \cdot 10^8$	$2592 \cdot 10^9$	$31536 \cdot 10^9$	$31556736 \cdot 10^8$
$n \lg n$	62746	2801417	133378058	2755147513	71870856404	797633893349	68654697441062
n^2	1000	7745	60000	293938	1609968	5615692	56175382
n^3	100	391	1532	4420	13736	31593	146677
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h′, replace (d, h′) in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

????

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h′, replace (d, h′) in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h′, replace (d, h′) in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

?????

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h′, replace (d, h′) in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

n^2 times

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

n^2 times

O(1)

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

O(n^2)

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

O(n^2)

O(1)

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

O(n^2)

O(1)

Total = O(1) + O(n^2) + O(1) = **O(n^2)**

Runtime Analysis in Practice

Algorithm 1 Gale-Shapley (1962)

```
1:  procedure StableMatching(Doctor Preferences, Hospital Preferences)
2:      final_pairs ← empty array
3:      while there is an unmatched hospital h do
4:          h makes an offer to their favorite doctor d (that they haven't requested before)
5:          if d has no prior offer then
6:              Add (d, h) to final_pairs
7:          else
8:              If d prefers h to its current pairing h', replace (d, h') in final_pairs with (d, h).
9:          end if
10:     end while
11:     return final_pairs
12: end procedure
```

O(1)

O(n²)

O(1)

Total = O(1) + O(n²) + O(1) = **O(n²)**

Brute Force: O(n!)

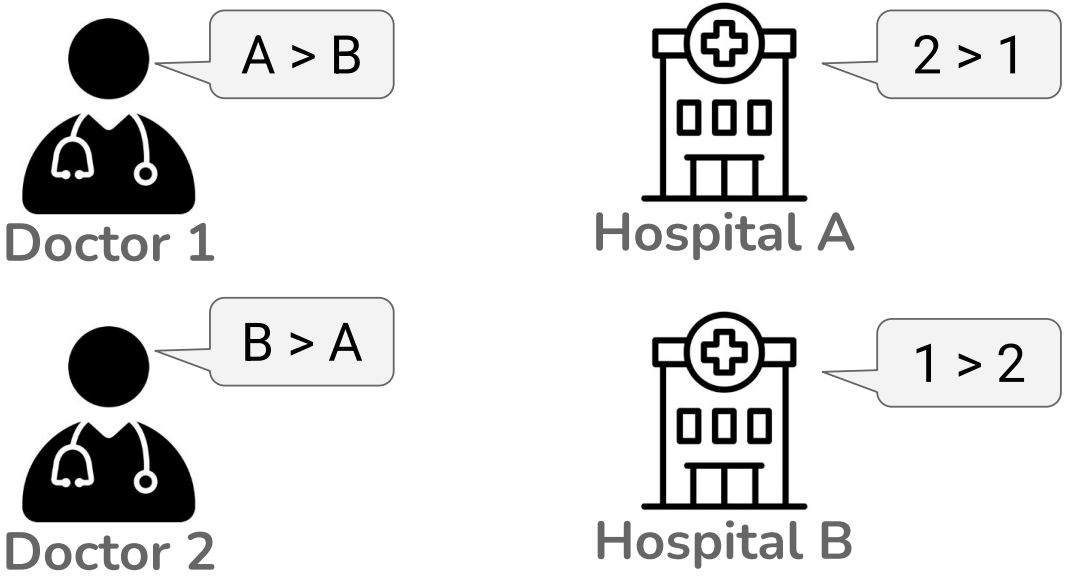
Runtime Analysis Recap

When discussing runtime in this course, we follow a strict set of rules.

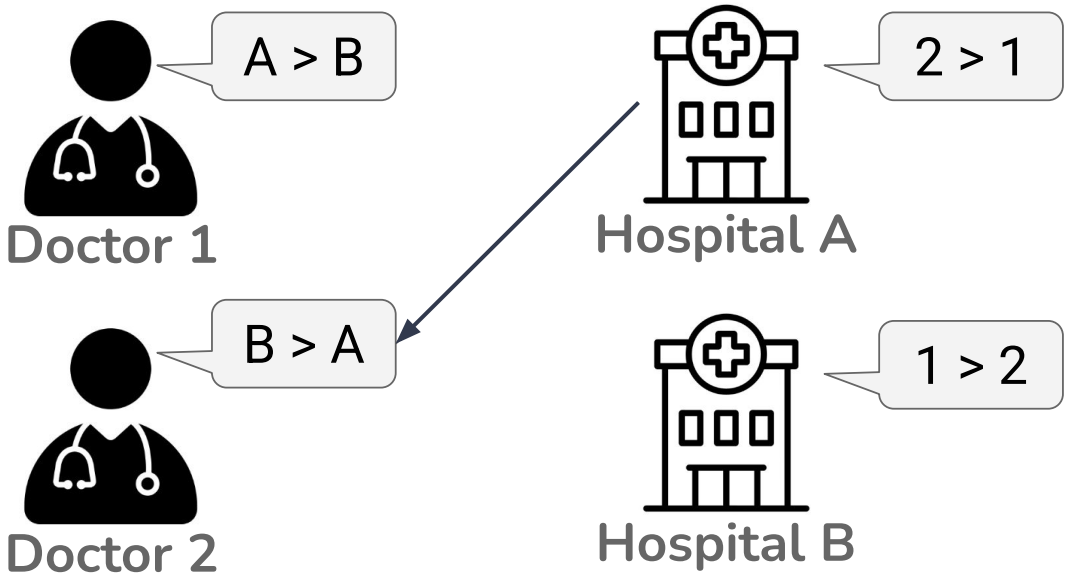
- Time complexity describes how the algorithm's runtime scales with input size.
- We consider polynomial time and better to be "efficient".
- We are (mostly) interested in worst-case time complexity.
- We ignore constant factors and lower-order terms.

Stable Matching: Who’s Really Winning?

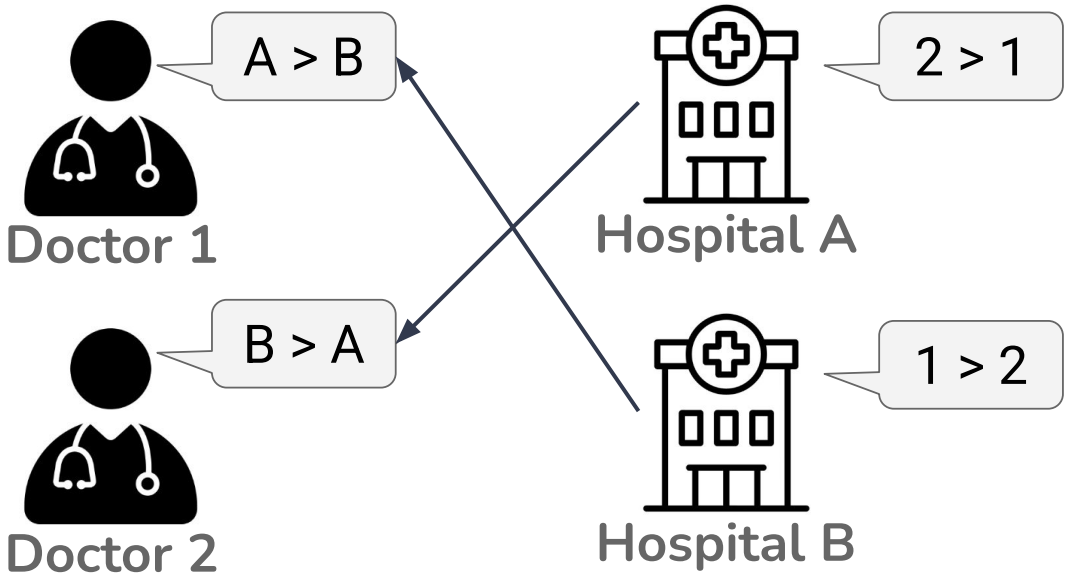
A very simple case.



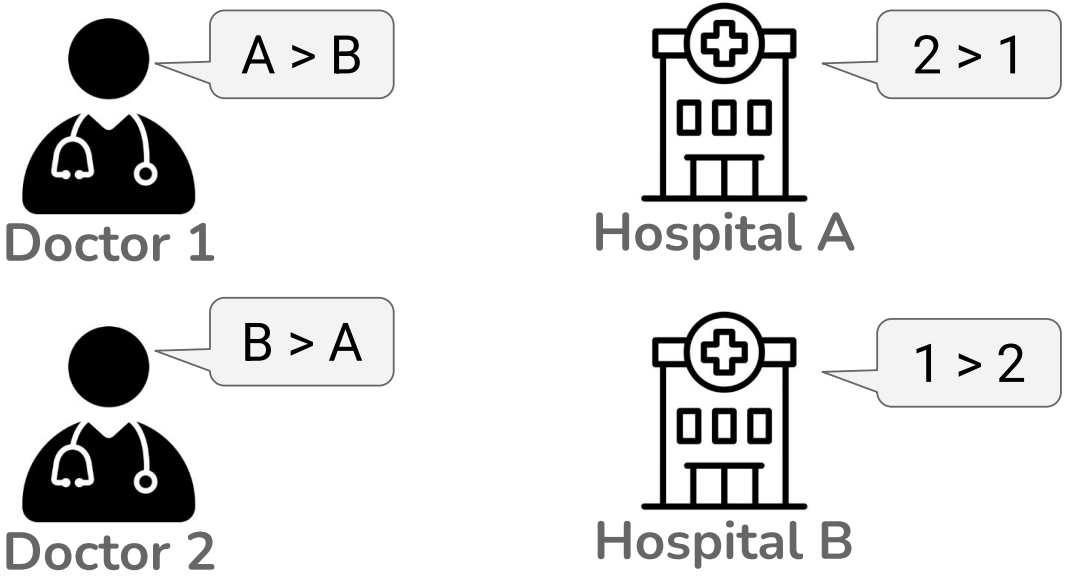
Running Gale-Shapley:



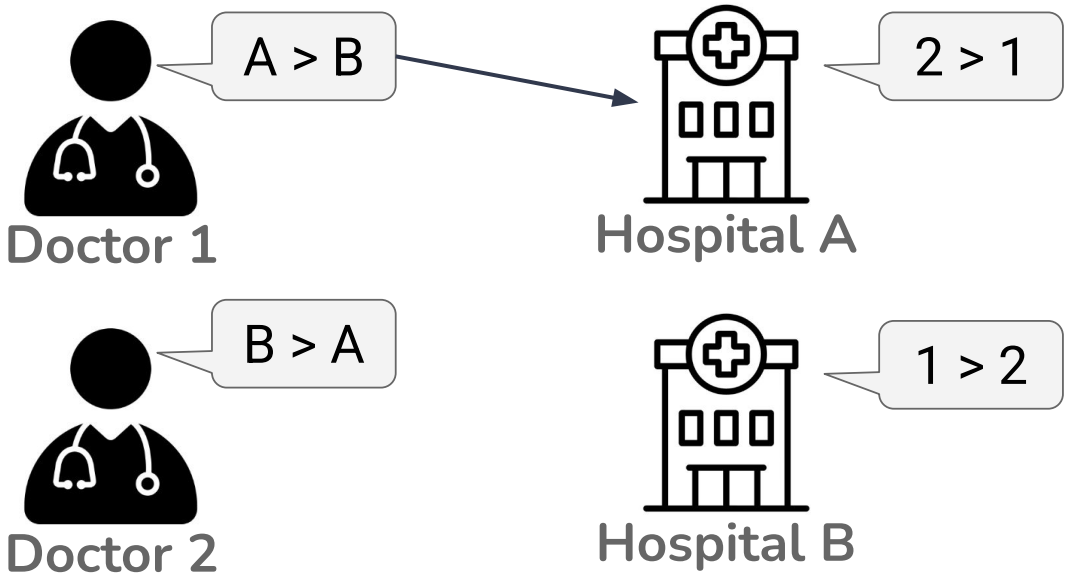
Running Gale-Shapley:



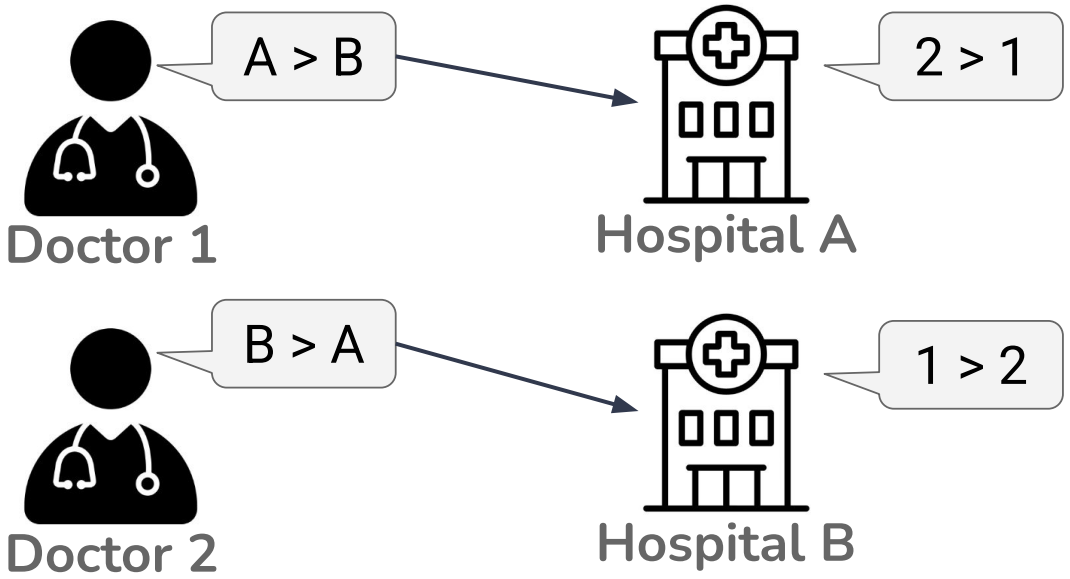
What if the Doctors proposed instead?



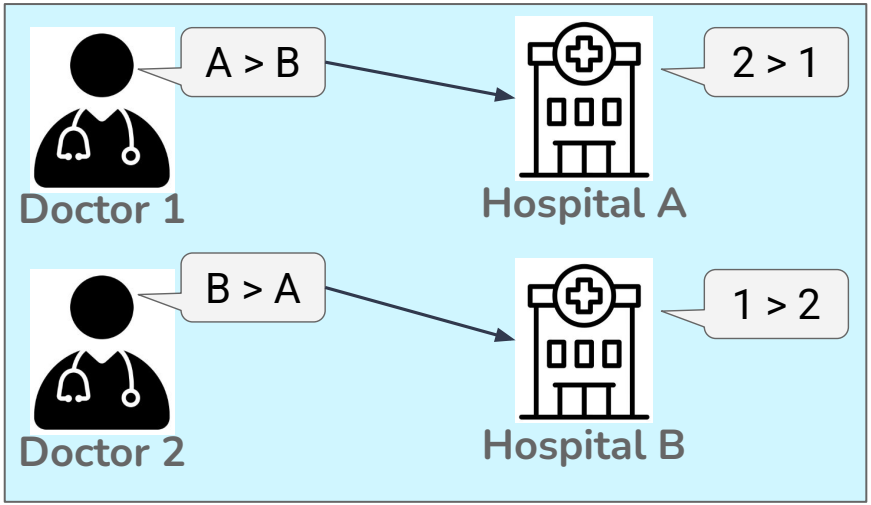
What if the Doctors proposed instead?

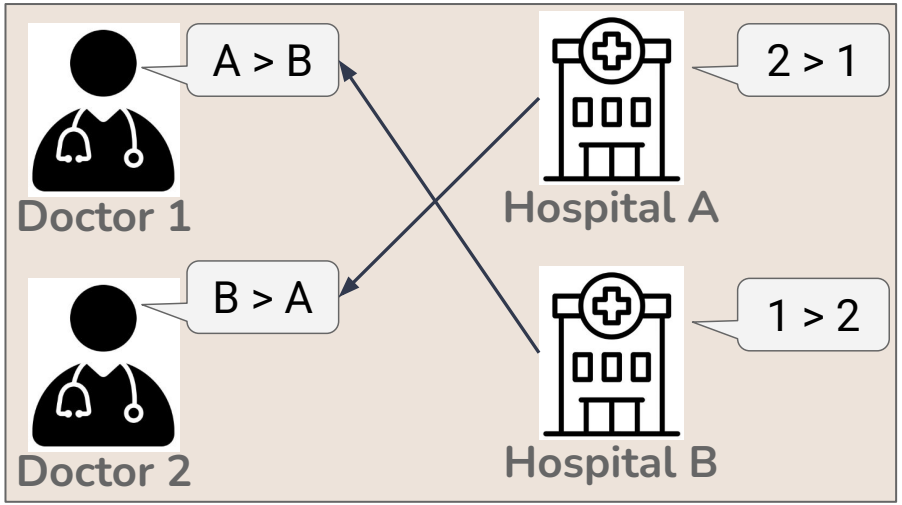


What if the Doctors proposed instead?

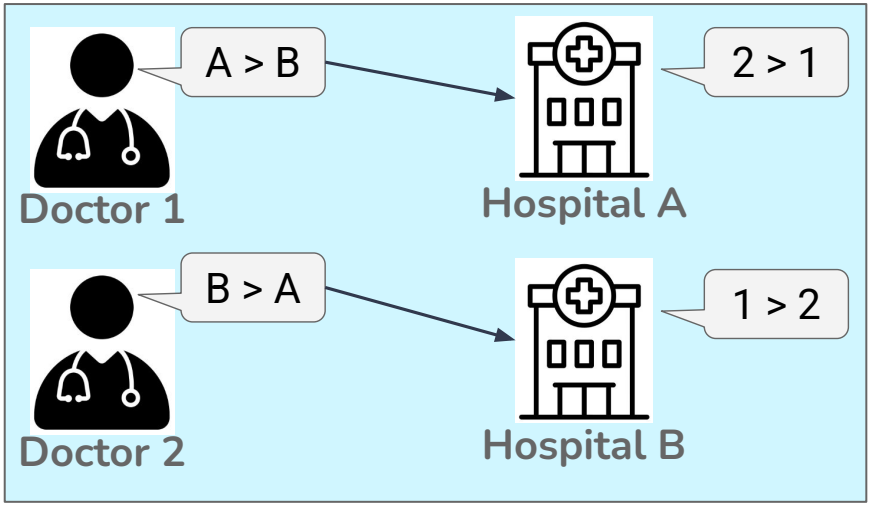


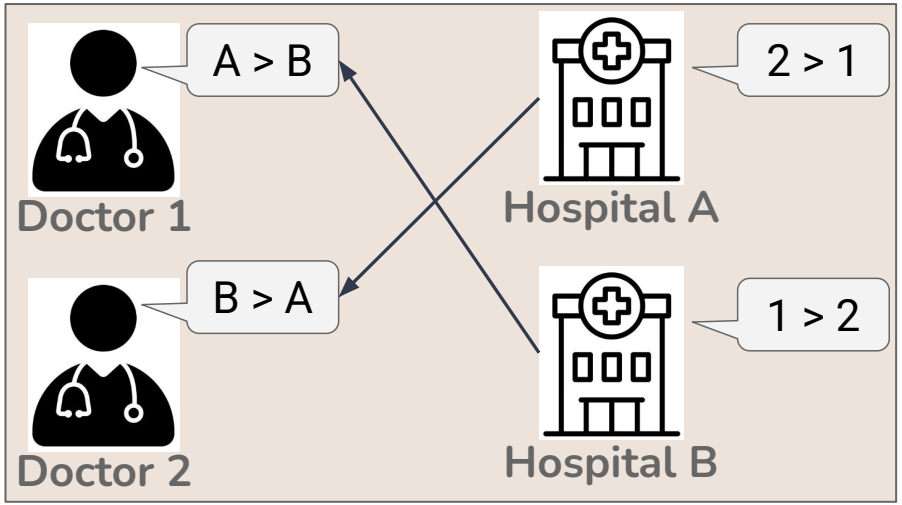
Two Possible Matchings!





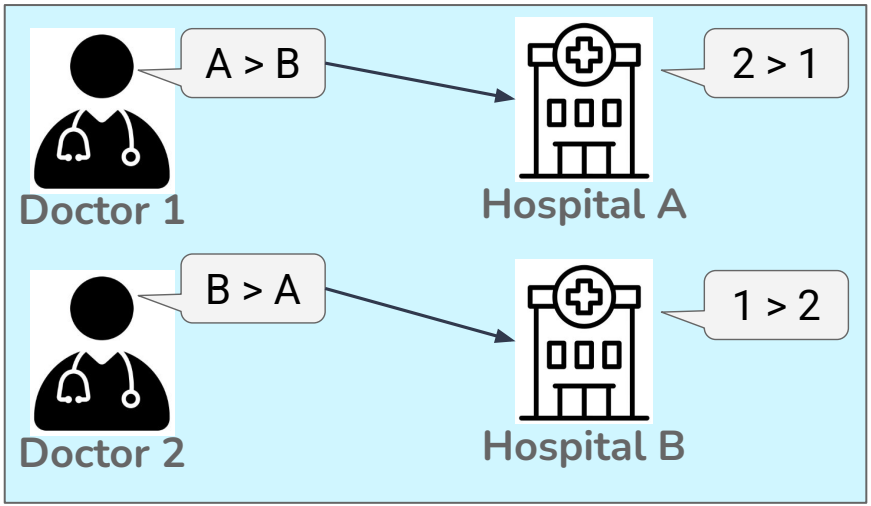
Two Possible Matchings!

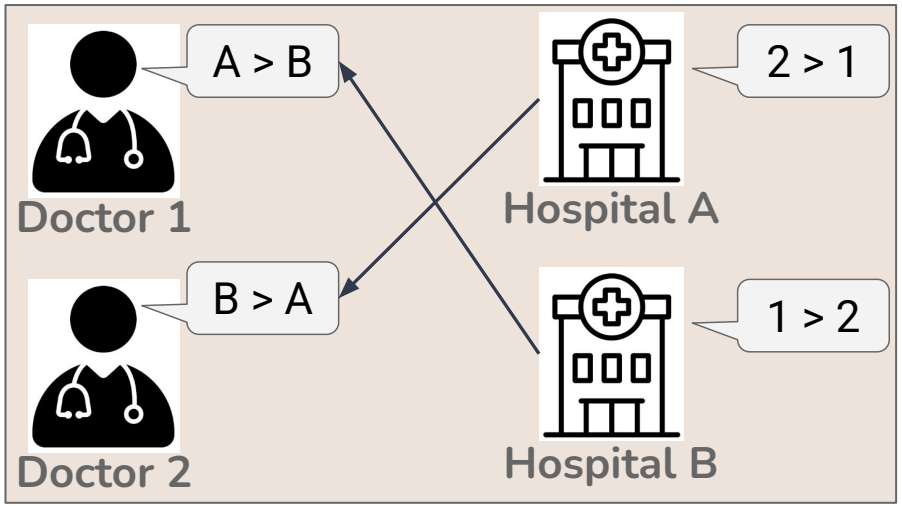




Under Gale-Shapley, the proposers seem to be happier. But why?

Two Possible Matchings!





Under Gale-Shapley, the proposers seem to be happier. But why?

Intuition: Proposers pick their favorite from everyone,
Receivers pick their favorite from their offers.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).

Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then...

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).

Claim: Every hospital gets its **favorite** valid partner.

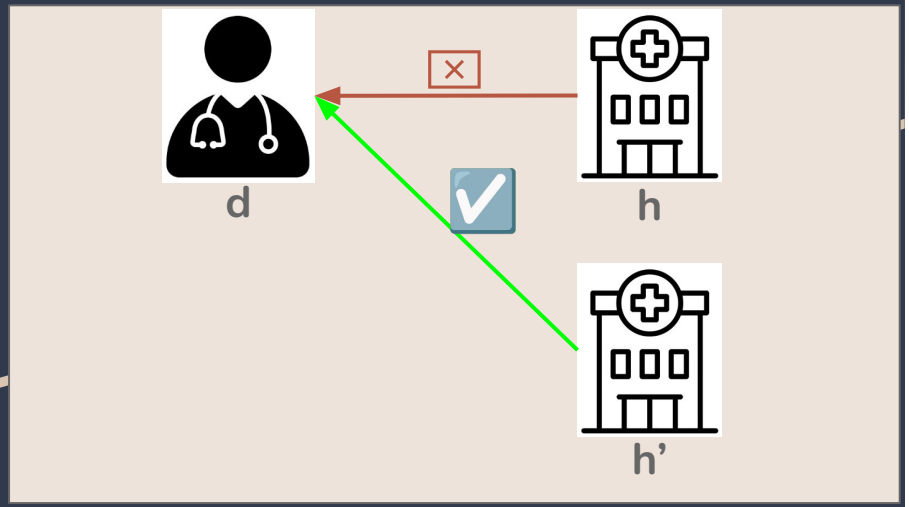
Proof by Contradiction:

Assume not. Then, in some run of Gale-Shapley, some hospital is rejected by a valid partner.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

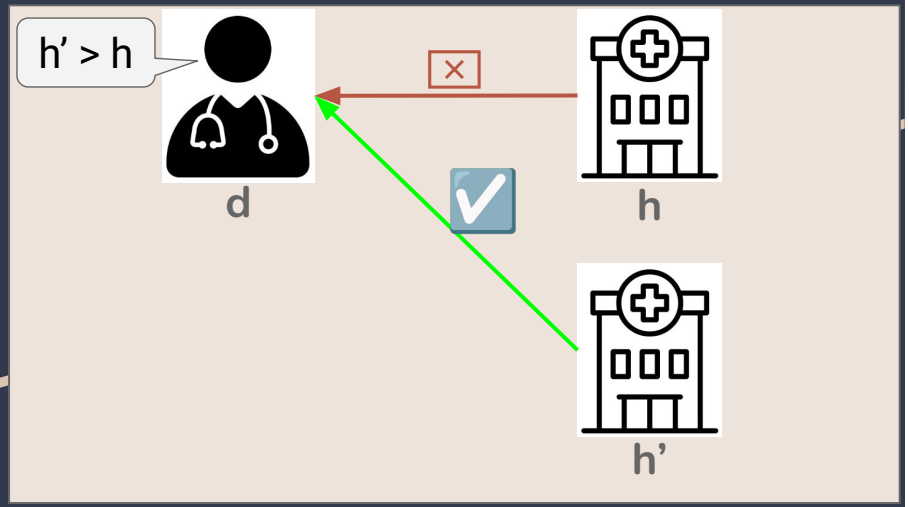
Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

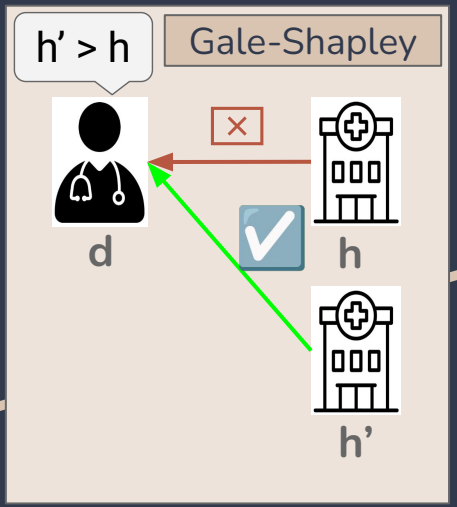
Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

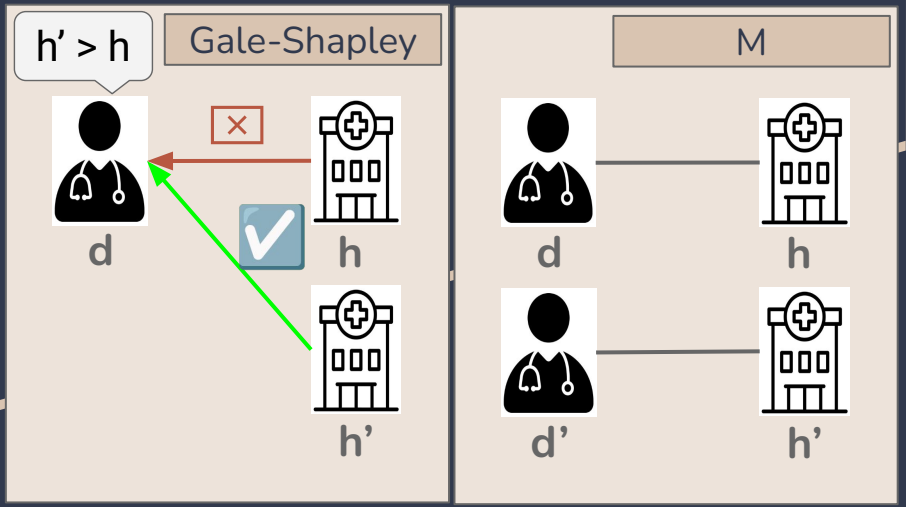
Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

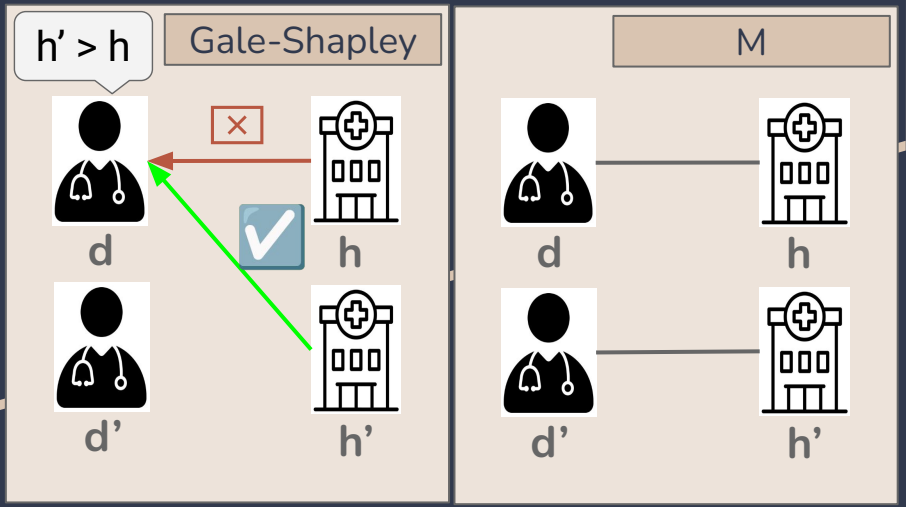
Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

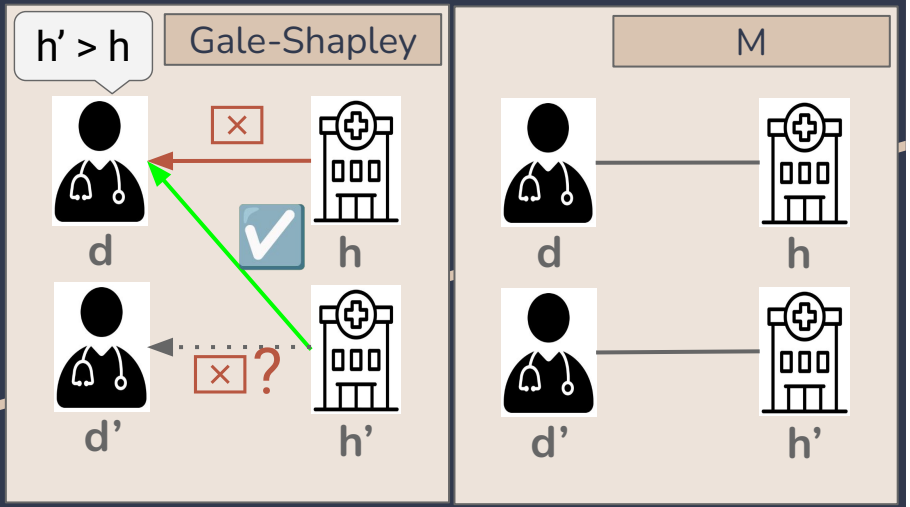
Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

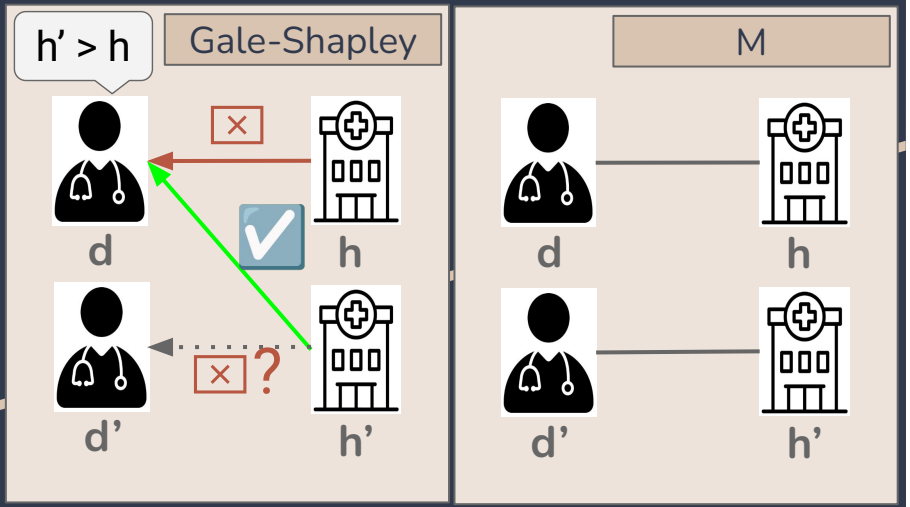
Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'. If h' preferred d' > d, then it must have already proposed to d' in our run of Gale-Shapley and been rejected.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

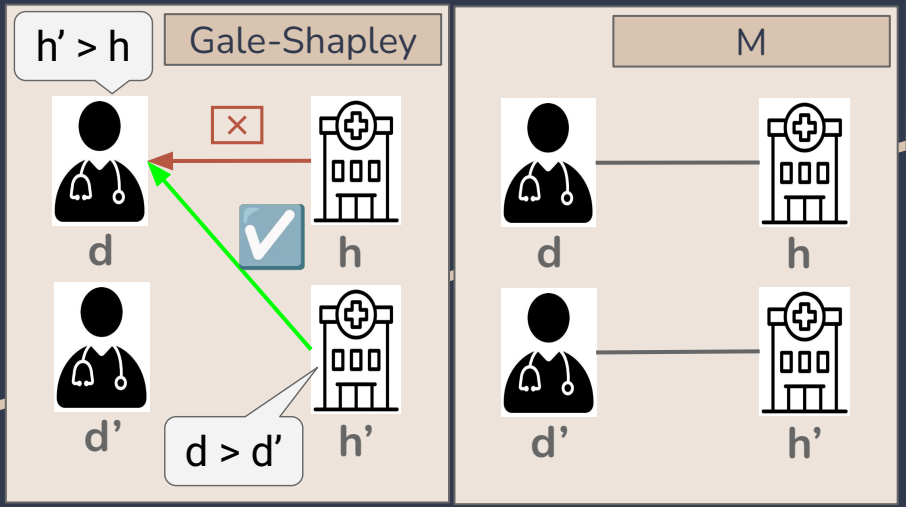
Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'. If h' preferred d' > d, then it must have already proposed to d' in our run of Gale-Shapley and been rejected. This is impossible because (d, h) is the first rejection by a valid partner.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

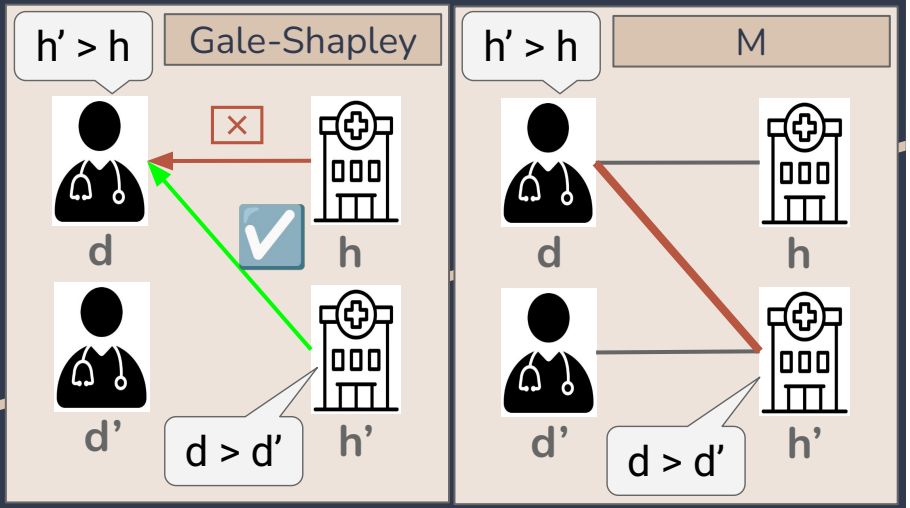
Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'. If h' preferred d' > d, then it must have already proposed to d' in our run of Gale-Shapley and been rejected. This is impossible because (d, h) is the first rejection by a valid partner. Thus, h' prefers d > d'.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

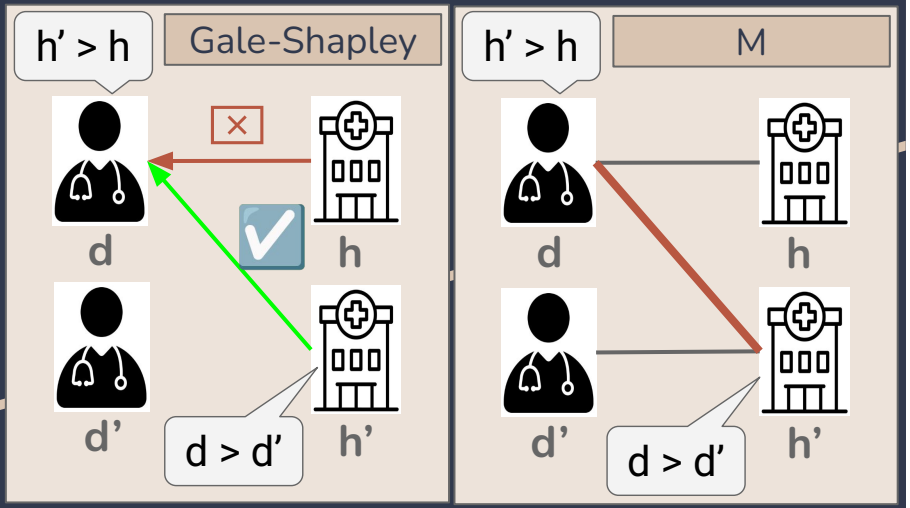
By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'. If h' preferred d' > d, then it must have already proposed to d' in our run of Gale-Shapley and been rejected. This is impossible because (d, h) is the first rejection by a valid partner. Thus, h' prefers d > d'.

Therefore, in M, (d, h') is a blocking pair.

A Formal Proof

Terminology

A doctor d is a **valid partner** for a hospital h if there exists a stable matching with (d, h).



Claim: Every hospital gets its **favorite** valid partner.

Proof by Contradiction:

Assume not. Then, in the run of Gale-Shapley, some hospital is rejected by a valid partner.

Take the *first* time during Gale-Shapley that a hospital is rejected by a valid partner. Denote the doctor as d, the rejected hospital h, and the proposing hospital h'. So d prefers h' > h.

By definition, there exists some stable matching M where (d, h) are paired. Denote by (d', h') the pair in M that contains h'. If h' preferred d' > d, then it must have already proposed to d' in our run of Gale-Shapley and been rejected. This is impossible because (d, h) is the first rejection by a valid partner. Thus, h' prefers d > d'.

Therefore, in M, (d, h') is a blocking pair. This contradicts the claim that M is stable, so M can not exist. Thus, no hospital is rejected by a valid partner.

Consequences

Every hospital gets its favorite valid partner.

By a very similar argument, we can find that **every doctor gets its least favorite valid partner.**

Parting Questions:

How can we design a more “fair” system?

Say you know everyone’s preferences. If you’re a hospital, can you get a better doctor than usual by lying about your preferences? If you’re a doctor, can you get a better hospital than usual by lying about your preferences?

Summary

These slides were strongly influenced by the official slides for Algorithm Design by Kleinberg and Tardos, and by slides from Professor Omar Ibrahim.

Today we used two new lenses to look at the Gale-Shapley algorithm for stable matching.

First, we developed the vocabulary to talk about algorithm runtimes, and used that to determine that the algorithm is “efficient”.

Next, we used a bias lens, analyzing the output of the algorithm to find that it heavily favors the proposers.