From last time...

For a given problem instance, there may be several stable matchings. Do all executions of Gale-Shapley yield the same stable matching? If so, which one?

Definition: College c is a valid partner of student s if there exists some stable matching in which they are matched.

College-optimal assignment: Each college receives best valid student.

Claim: All executions of GS yield college-optimal assignment, which is a stable matching!

Proof by contradiction

We want to prove that the GS algorithm produces a mapping that is optimal for the choosing group.

Definition: best(c) is the valid partner with the highest ranking of all of c’s valid partners.

First, we assume that what we want to prove is not true: Assume that there is a stable matching M that contains (c, s) where best(c) ≠ s.

For this to be the case, one of 2 things must be true:

Case 1: c never made an offer to best(c)
Case 2: c made an offer to best(c) but it was rejected.
Proof of Case 1

Case 1: c never made an offer to best(c)
Since c makes offers from most preferred to least preferred, and c made an offer to s before best(c), c must prefer s to best(c). This contradicts the definition of best(c), so this case does not hold.

Proof of Case 2 (where I got stuck)

Case 2: c made an offer to best(c) but it was rejected.
Let’s consider the first rejection of a valid partner that occurs when constructing M.
Assume the college being rejected is c. The student doing the rejecting must be best(c) since this is the first rejection by a valid partner.

Proof of Case 2 (continued)

Given: c preferences: ... best(c) ...
best(c) rejects c
Assume that best(c) accepted an offer from c’. This means best(c) prefers c’ to c. Since we are considering the first rejection, it must be the case that best(c) is also best(c’).

Given: c’ preferences: ... best(c’) ...
Conclude: best(c) prefers c’ to c
best(c) and best(c’) are the same student
Proof of Case 2 (continued)

Given: c preferences: ..., best(c) ...
best(c) rejects c
C’ preferences: ..., best(c') ...
Conclude: best(c) prefers c’ to c
best(c) and best(c’) are the same student

There must be another stable matching M’ that contains
(c, best(c)) (based on the definition of best).
In M’, c’ must be paired with a different student s’.
However, the pair (c’, best(c)) is an instability in M’ since:
   1. best(c) preferred c’ to c
   2. c’ prefers best(c) to any other valid partner

Contradiction!

What have we accomplished?
- Gone from an informal problem description to a precise problem description
- Designed an algorithm to solve the problem
- Proved the algorithm is correct and has some other interesting properties

Time to switch topics!
Goal of course: Find efficient solutions to problems.

What is efficiency?
- Time
- Memory

Definition: An efficient algorithm is one that runs quickly on real inputs.
Efficiency

Goal of course: Find efficient solutions to problems

What is efficiency?
- Time
- Memory

Definition: An efficient algorithm is one that runs quickly on real inputs.

Measuring Time

Idea: Implement a program. Time how long it takes.

Problems?

Effects of the programming language?
- Effects of the processor?
- Effects of the amount of memory?
- Effects of other things running on the computer?
- Effects of the input values?
- Effects of the input size?
Measuring Time

Idea: Implement a program. Time how long it takes.

Problems?
- Effects of the programming language?
- Effects of the processor?
- Effects of the amount of memory?
- Effects of other things running on the computer?
- Effects of the input values?
- Effects of the input size?

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Search</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
</tbody>
</table>
What are the maximum number of elements we need to look at to find a value with these algorithms?

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Binary</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Quicksort vs. Brute Force Sort

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quicksort</td>
<td>8</td>
<td>24</td>
<td>64</td>
</tr>
<tr>
<td>N log N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brute</td>
<td>24</td>
<td>40,320</td>
<td>20,922,789,888,000</td>
</tr>
<tr>
<td>Force Sort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N!</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Polynomial Time

- An algorithm requires polynomial time if it slows down by a constant factor as the input size increases.
- Number of steps in a polynomial algorithm: $c \cdot N^d$, where $c$ and $d$ are constants $> 0$ and $N$ is the input size.
- Suppose input size increases from $N$ to $2N$. Time increases by a constant multiplicative factor of $2^d$: $c \cdot (2N)^d = c \cdot 2^d \cdot N^d$
Polynomial Time

An algorithm requires polynomial time if it slows down by a constant factor as the input size increases.

Number of steps in a polynomial algorithm:
$c \cdot N^d$, where $c$ and $d$ are constants > 0 and $N$ is the input size.

Suppose input size increases from $N$ to $2N$. Time increases by a constant multiplicative factor of $2^d$:
$c \cdot (2N)^d = c \cdot 2^d \cdot N^d$

If there is no polynomial time solution, we say there is no efficient solution.

Running Times as Functions of Input Size

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size for a processor performing a million high-level instructions per second. In cases where the running time exceeds $10^9$ years, we simply record the time as taking a very long time.

<table>
<thead>
<tr>
<th>$n$</th>
<th>n log $n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$n^p$</th>
<th>$2^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 sec</td>
</tr>
<tr>
<td>50</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>10^9 years</td>
</tr>
<tr>
<td>100</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>11 min</td>
<td>36 years</td>
<td>very long</td>
</tr>
<tr>
<td>500</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>23 days</td>
<td>12,892 years</td>
<td>very long</td>
</tr>
<tr>
<td>1000</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
<td>18 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>10^3</td>
<td>&lt; 1 sec</td>
<td>1 sec</td>
<td>2 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>10^6</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
<td>4 min</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
<tr>
<td>10^9</td>
<td>&lt; 1 sec</td>
<td>20 sec</td>
<td>12 days</td>
<td>very long</td>
<td>very long</td>
<td>very long</td>
</tr>
</tbody>
</table>

Big O Notation

Let $T(n)$ be a function that defines the worst-case running time of an algorithm.

$T(n)$ is $O(f(n))$ if
$T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

Example:
Let $T(n) = 3n + 2$
$T(n)$ is $O(n)$ because
$T(n) \leq 4n$ for all $n \geq 2$

$O(n)$ is the *asymptotic upper bound* of $T(n)$. 
Let $T(n)$ be a function that defines the worst-case running time of an algorithm.

$T(n)$ is $\Omega(f(n))$ if

$T(n) \geq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

Example:

Let $T(n) = 3n + 2$

$T(n)$ is $\Omega(n)$ because

$T(n) \geq n$ for all $n \geq 0$

$\Omega(n)$ is the asymptotic lower bound of $T(n)$. 
**Notation**

- Let $T(n)$ be a function that defines the worst-case running time of an algorithm.

  - $T(n)$ is $\Theta(f(n))$ if $T(n)$ is $O(n)$ and $T(n)$ is $\Omega(n)$

**Example:**

- Let $T(n) = 3n + 2$
  - $T(n)$ is $\Theta(n)$ because $T(n)$ is $O(n)$ and $T(n)$ is $\Omega(n)$

  - $\Theta(n)$ is the asymptotic tight bound of $T(n)$.

**Visualizing Asymptotics**

- $3n + 2$ is $\Theta(n)$

**Best Case vs. Worst Case**

```plaintext
sorted = false
while (!sorted) {
    sorted = true
    for each i in 0 to length(A) - 2{
        if A[i] > A[i+1] {
            swap( A[i], A[i+1] )
            sorted = false
        }
    }
}
```
Best Case vs. Worst Case

```java
sorted = false
while (!sorted) {
    sorted = true
    for each i in 0 to length(A) - 2{
        if A[i] > A[i+1] {
            swap( A[i], A[i+1] )
            sorted = false
        }
    }
}
```

We care about worst case when we discuss asymptotic complexity.

---

**$O()$, $\Omega()$, $\Theta()$**

- $O()$ - upper bound
  
  $T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

- $\Omega()$ - lower bound
  
  $T(n) \geq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

- $\Theta()$ - tight bound
  
  Both $O()$ and $\Omega()$

---

**Transitivity**

Claim: If $f$ is $O(g)$ and $g$ is $O(h)$, then $f$ is $O(h)$

How would you prove that?
Transitivity
Claim: If $f$ is $O(g)$ and $g$ is $O(h)$, then $f$ is $O(h)$
How would you prove that?
Significance: It means we can order the $O()$ functions.

Additivity (1)
Claim: If $f$ is $O(h)$ and $g$ is $O(h)$, then $f+g$ is $O(h)$
How would you prove that?

Additivity (2)
Claim: If $f$ is $O(g)$, then $f+g$ is $Θ(g)$
How would you prove that?
Additivity (2)

Claim: If \( f \) is \( O(g) \), then \( f+g \) is \( \Theta(g) \).

How would you prove that?

Example: If \( f \) is \( n \) and \( g \) is \( n^2 \), then \( n^2 + n \) is also \( O(n^2) \). It means we can ignore all but the highest order term when discussing \( O() \).

Other Asymptotic Orderings

Logarithms:
- \( \log_a n \) is \( O(n^d) \), for all bases \( a \) and all degrees \( d \)
- All logarithms grow slower than all polynomials

Exponential functions:
- \( n^d \) is \( O(r^n) \) when \( r > 1 \)
- Polynomials grow no more quickly than exponential functions.
A Harder Example

Which of these grows faster?
$n^{4/3}$
$n \log n^3$