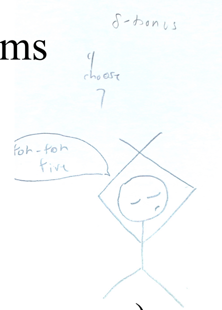
CS445 – Introduction to Algorithms

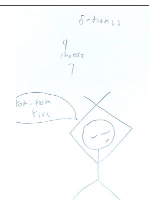


- **Webpages**
- Course webpage – google doc (reach via my homepage)
- Use D2L to reach recordings of lectures (Panopto), calendar
- Use Gradescope to submit his and view feedback
- Use Piazza for course communication, discussions and announcements.
- Use Overleaf to view assignments.

Homeworks workflow. Collaboration vs Cheating

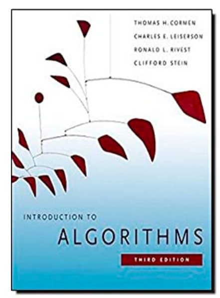
- Alg: Once a homeworks is published
  - ❑Read questions
    - ❑ If needed, re-watch lectures (Alon and Others) online,
  - ❑Thinks really hard. Discover what does **not** work and why
  - ❑Meet your peers and discuss and does/does not work and why?
  - ❑Write Solutions yourself.
  - Diverging from this algorithm might improve your hw grade but is likely to impact your exams grades (not to mention ethical issues, honor code etc).
- Homework's rules.
  - Collaborations ++. Brainstorming in **small** groups
  - Give credit. Specify your contribution to each solution (in %).
  - Sharing text is cheating.

CS445 - Regulation, Bureaucracy

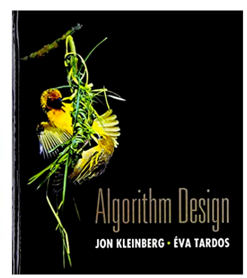


1. Grading Scheme (midterm vs. final)
2. Textbooks
3. Video recording
4. Web Resources
5. Prerequisites (course is mostly self contained, but harder if you did not pass cs345.
6. Piazza.
   I. Post are for clarifications.
   II. Be careful not to share any hints in your posts
   Eg. “*are we allowed to use Quicksort for the solution of hw3 Q7*” is a violation of code of conduct, considered cheating, and could get you blocked from piazza.
   I. If you have any doubts, send a private message.
7. Attendance - strongly recommended.
   1. Active learning - your webcam should be **on** during active learning (talk to me if there are any technical difficulties).

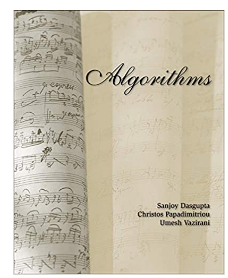
1. Textbook



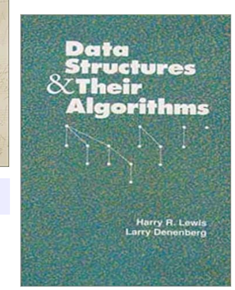
CLRS



Kleinberg & Tardos



Sanjoy Dasgupta



Lewis Denenberg

Course slides

Introduction to Algorithms

- In this course, we will discuss problems, and algorithms for solving these problems.
- There are so many algorithms – why focus on the ones in the syllabus ?

Why study algorithms and performance?

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
  - (e.g., by using big-O –notation.
  - Will see lots of `big-O's of quantities you might have not seen before:
    - (CPU, Space, I/O, parallel steps, GPU)
- In real life, many algorithms, though different from each other, fall into one of several ***paradigms*** (discussed shortly).
- These paradigms can be studied, and applied for new problems

Why these algorithms (cont.)

1. **Main paradigms:**
   a) **Greedy algorithms**
   b) **Divide-and-Conquers**
   c) **Dynamic programming**
   d) **Brach-and-Bound (mostly in AI)**
   e) **Etc etc.**

2. **Other reasons:**
   a) **Relevance to many areas:**
      - **E.g., networking, internet, search engines…**

   b) **Coolness**

Other goals of the course

- Knowing when running time counts, and what to do when it does
- Magic of randomness and sampling

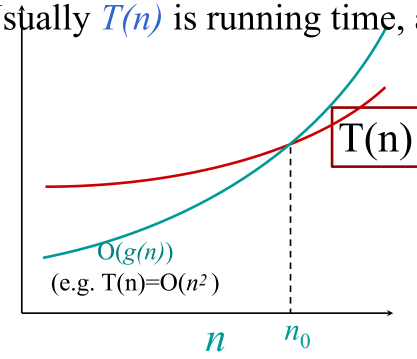
O-notation

we say that $T(n) = O(g(n))$ **iff**

there exists positive constants c_1, and n_0 such that

$0 \le T(n) \le c_1 g(n)$ for all $n \ge n_0$

Usually $T(n)$ is running time, and n is size of input



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- **Asymptotic analysis** is a useful tool to help to structure our thinking.

Ω-notation

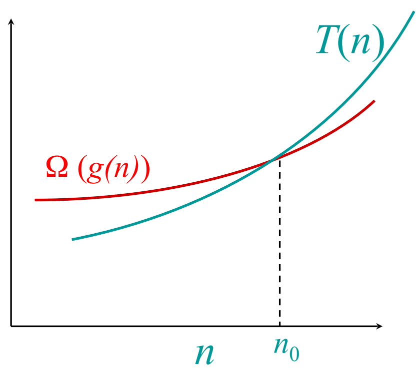
Math:

We say that $T(n) = \Omega(g(n))$ **iff**

there exists positive constants c_2, and n_0

such that

$0 \le c_1 g(n) \le T(n)$ for all $n \ge n_0$



Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Omega(n^3)$

Θ-notation

We say that $T(n) = \Theta(g(n))$ **iff**

there are positive constants

c_1, c_2, and n_0

such that

$0 \le c_1 g(n) \le T(n) \le c_1 g(n)$

for every n, provide that $n \ge n_0$

in other words, we could say that

$T(n) = \Theta(g(n))$

iff it is true that

$T(n) = O(g(n))$ and that $T(n) = \Omega(g(n))$.

For example, for every size n of an input array, bubble sort, insertion sort and swap-sort will never needs more than n^2 operations (up to a constant).

So their running time is $O(n^2)$.

On the other hand, we can find an input (one is enough) that causes their running time to be no less than n^2. So their running time is also $\Omega(n^2)$.

Putting it together, their running time is $\Theta(n^2)$

Notation - cont

So if $T(n)= O(n^2)$ then we are also sure that
$T(n)= O(n^3)$ and that
$T(n)= O(n^{3.5})$ and
$T(n)= O(2^n)$

But it might or might not be true that $T(n)= O(n^{1.5})$.

However, if $T(n)= \Omega(n^2)$ then it is **not** true that
$T(n)= O(n^{1.5})$

Big difference between O and Ω: we can talk about Ω of a **problem** (that is, any algorithm that solves this problem takes Ω(something)

Eg. Sorting takes $\Omega(n \log n)$

Examples 3

```
1. Read(n);
2. k=1 ;
3. while( k ≤ n )
4.          k=2k ;
```

What is the running time of this code (as a function of n) ?

- We know that each iteration takes $O(1)$ times. Need to find the number time line 3 is executed.
  - After the first iteration $k=2=2^1$
  - After the 2nd iteration $k=4=2^2$
  - After the 3rd iteration $k=8=2^3$
  - ….
  - After the i'th iteration $k=2^i$

Cheatsheet :

- $\log(ab)=\log(a)+\log(b)$
- $\log(a^b) = b \log a$
- $\log_a(x) = \log_b(x) / \log_b a$
- $x \le y$ implies $\log_2(x) \le \log_2(y)$

Lets count the number ***j*** *of times that the condition of line 3 was checked and yield true.*

- If the condition is true, then $k \le n$. But $k=2^j$. So $k = 2^j \le n$.
- Taking $\log_2$ from both sides, we have that

$\log_2 k = \log_2(2^j) \le \log_2(n)$ or..

$\log_2(2^j) = j \log_2 2 = j \le \log_2(n)$ or..

$j=O(\log_2 n)$. $T(n)=O(\log n)$

- Homework: Prove $T(n)= \Theta(\log n)$

Examples 4

```
read(n) ;
for(i=1 ; i < n ;  i++)
  for( j=i ; j <n ; j += i )
  print( “*” ) ;
```

- Time Complexity Analysis – first approach:
  - The outer loop (on i) runs exactly $n-1$ times
  - The inner loop (on j) runs $O(n)$ times.
  - Together $T(n)=O(n^2)$.

Examples 4

```
read(n) ;
for(i=1 ; i < n ;  i++)
  for( j=i ; j <n ; j += i )
  print( “*” ) ;
```

- Time Complexity Analysis – first approach:
  - The outer loop (on i) runs exactly $n-1$ times
  - The inner loop (on j) runs $O(n)$ times.
  - Together $T(n)=O(n^2)$.

Is it true that the running time is $\Omega(n^2)$?

Examples 4

```
read(n) ;
for(i=1 ; i < n ; i++)
  for( j=i ; j <n ; j += i )
  print( "*" ) ;
```

- Time Complexity Analysis – first approach:
  - The outer loop (on i) runs exactly $n-1$ times
  - The inner loop (on j) runs $O(n)$ times.
  - Together $T(n)=O(n^2)$.

Is it true that the running time is $\Omega(n^2)$?

- More "sensitive" analysis:
  - For $i=1$ we run through $j=1,2,3,4...n$, total n times.
  - For $i=2$ we run through $j=2,4,6,8,10...n$, total $n/2$ times.
  - For $i=3$ we run through $j=3,6,9,12...n$, total $n/3$ times .
  - For $i=4$ we run through $j=4,8,12,16...n$, total $n/4$ times.
  - For $i=n$ we run through $j=n$, total $n/n=1$ times.
- Summing up: $T(n)=n+n/2+n/3+n/4+...n/n =$
  $n(1+1/2+1/3+1/4+...1/n) \approx n \ln n$

Harmonic Sum

Example 5

Pay attention - very relevant to this course

```
read(n) ;    a=0.5
while( n>1)  {
     For( j=1; j<n ; j++ )  print("*")
     n=a*n ;
}
```

- The **first** time the outer loop is called, the "print" is called n times.
- The **2nd** time the outer loop is called, the "print" is called an times.
- The **3rd** time the outer loop is called, the "print" is called a^2n times…
- The **k'th** time the outer loop is called, the "print" is called $a^k n$ times

- Let t be the number of iterations of the outer loop. Then the total time

$$= n + an + a^2n + a^3n + \ldots a^t n = n(1 + a + a^2 + a^3 + \ldots a^t) <$$
$$n(1 + a + a^2 + a^3 + \ldots a^t + \ldots) = n / (1-a) = O(n).$$

- Same analysis holds for any $a<1$

Geometric sum

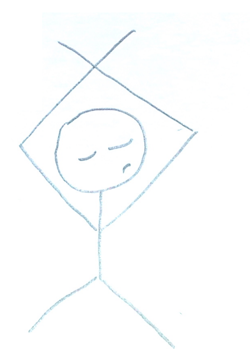
Recall: $1+a+a^2+\ldots+a^t = (1-a^{t+1})/(1-a)$.
If $a<1$ **then** $1+a+a^2+\ldots+a^t+\ldots = 1/(1-a)$

More about Ω( )

Sometimes we would talk about a lower bound on the running time of a **specific algorithms**
E.g. The insertion sort might take $\Omega(n^2)$ for some input

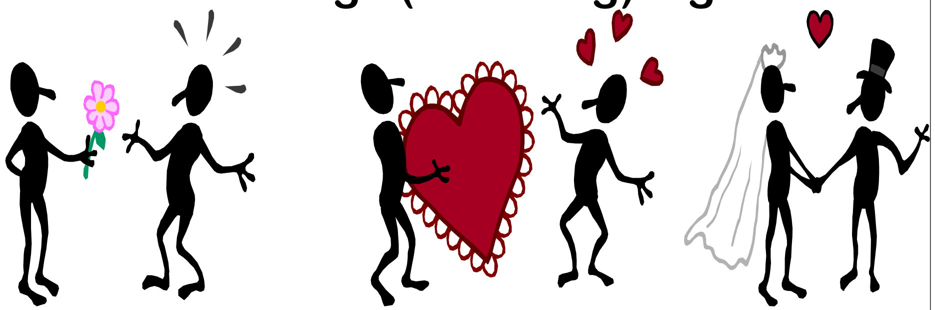
Sometimes we would talk about a lower bound on the running time of a **problem**
E.g.
1. Any algorithms that reads all the input (for any problem) requires $\Omega(n)$ time.
2. Any algorithm that stores all the data requires $\Omega(n)$ space.
3. Any algorithm that sort n keys requires $\Omega(n \log n)$
   (disclaimer – could be better if we make some assumptions about the keys or the model. Usually
- Sorting **sort integers** takes $\Omega(n)$ (how?)
- Sorting **floats** takes $\Omega(n \log n)$

CS445 – Salute



Credits:
Steven Rudich

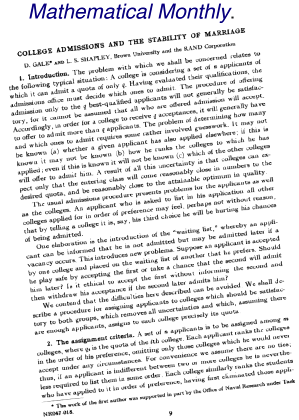
The Mathematics Of 1950's Dating: Who wins the battle of the sexes? Stable marriage (matching) algorithm.



Gale Shapley Stable Matching Algorithm

In 2012, Nobel Memorial Prize in Economic Sciences was awarded to Lloyd S. Shapley and Alvin E. Roth "for the theory of stable allocations and the practice of market design."[2]

Gale, D.; Shapley, L. S. (1962). "College Admissions and the Stability of Marriage". *American Mathematical Monthly*.



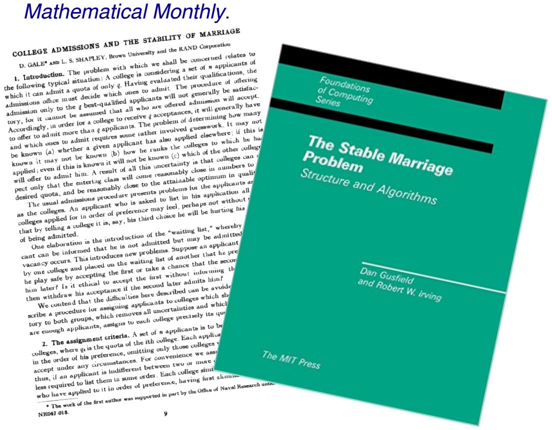
COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* and L. S. SHAPLEY, Brown University and the RAND Corporation

Gale Shapley Stable Matching Algorithm

In 2012, Nobel Memorial Prize in Economic Sciences was awarded to Lloyd S. Shapley and Alvin E. Roth "for the theory of stable allocations and the practice of market design."[2]

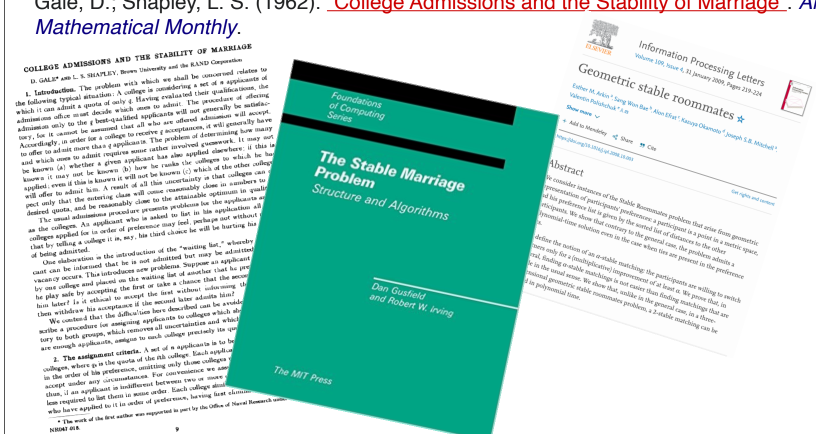
Gale, D.; Shapley, L. S. (1962). "College Admissions and the Stability of Marriage". *American Mathematical Monthly*.



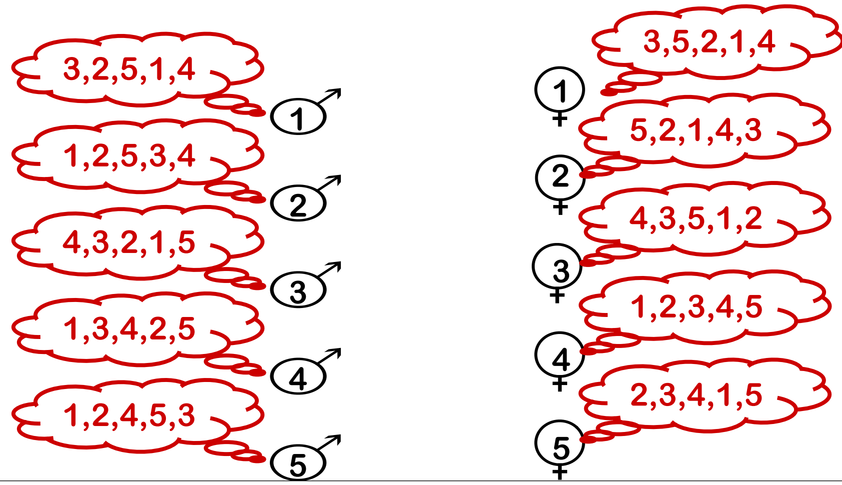
Gale Shapley Stable Matching Algorithm

In 2012, Nobel Memorial Prize in Economic Sciences was awarded to Lloyd S. Shapley and Alvin E. Roth "for the theory of stable allocations and the practice of market design."[2]

Gale, D.; Shapley, L. S. (1962). "College Admissions and the Stability of Marriage". *American Mathematical Monthly*.



- There are n males and n females
- Each female has her own ranked preference list of all the males
  - E.g., women #1 most prefers male #3 over any other male.
- Each male has his own ranked preference list of the females
- How should we match them (1-to-1)



Product Details
- **Series:** Foundations of Computing

- **Hardcover:** 258 pages
- **Publisher:** The MIT Press (August 22, 1989)
- **Language:** English
- **ISBN-10:** 0262071185
- **ISBN-13:** 978-0262071185
- **Product Dimensions:** 9.4 x 7.3 x 0.8 inches
- **Shipping Weight:** 1.4 pounds

Definition of a Matching in this lecture

- A **matching** in this context is a list of couples that according to the algorithm, should be matched to each other. Each male is married to a single female and vice versa.

$M = \{ (m_1, f_{13}), (m_2, f_7), \ldots\ldots (m_n, f_n) \}$

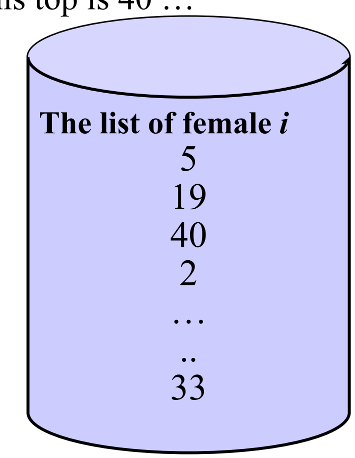
The algorithm aims to find a good matching (under some definition)

- Sometimes the term **pairing** is used

Definitions about the preference lists

In her list,
- male 5 is her **top choice.**
- If he is not interested, her top choice is male 19.
- If neither 5 nor 19 are interested, his top is 40 …

– This is a full ranking of all males.



Definition: Rogue Couples

- Consider a given matching M (that is, assume that matching is done) .

A rouge couple (in this matching) is a couple (female, male) who are **not** married to each other, but prefer each other over their spouses.

- In the example to the right
  - **Zod** is married to **Evanora (6)**, but prefers **Aradia (3)**
  - **Aradia** is married to **Syndrom (5)** ,but prefers **Zod** (2)

Zod's list
----
1. Allegra
2. Beatrix
3. **Aradia**
4. Cassandra
5. Cordelia
6. **Evanora**
7. Gullveig

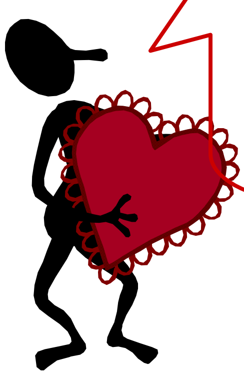
Aradia's list
----
1. Mr Burn
2. **Zod**
3. Hannibal
4. …
5. **Syndrom**
6. …
7. Gus Fring
8.

- They will be called a rogue couple.
- They both would gain from dumping their mates and marry each other.
- A source of confusion: A couple that is married to each other could not be rouge. The other couples are the ones we are concern about.
- A matching is called **stable** if it does not contain any rouge couples.
- The source of the 'instability': They would **both** benefit from changing the situation
- How could we obtain stability: Make sure that if one gains, the other loose

The study of stability will be the subject of the entire lecture.

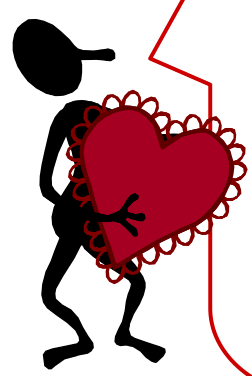
We will: Analyze various mathematical properties of an algorithm that looks a lot like 1950's dating.

Given a set of preference lists, how do we find a stable matching?



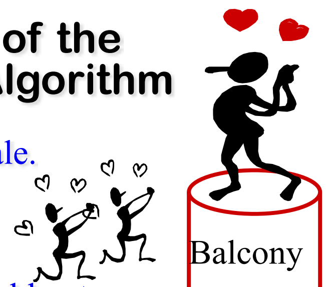
Wait! We don't even know that such a matching always exists!

Is there always a stable matching ?



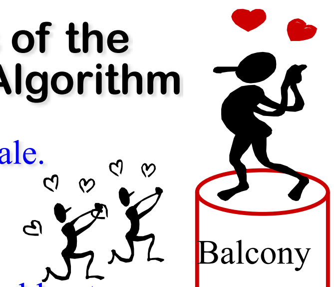
- Will show: every set of preference lists have a stable matching.
- Will prove it by presenting a fast algorithm that, given any set of input lists, will output a stable matching.
- Furthermore, we will discover the unfairness of the 1950 Traditional Matching Algorithm (TMA).

Terminology and principles of the 1950 Traditional Matching Algorithm



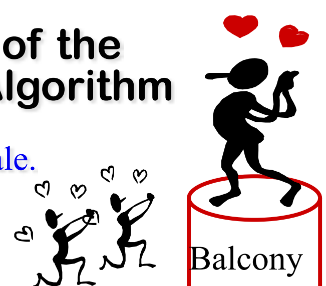
- A male can **propose** (marriage) to a female.
- A female can **reject** the proposal.
- During most of the process, a female would not accept a proposal, but would tell a proposing male "**maybe**".
- This is called "**putting the male on a string".**
- This male will come back the next day to propose again (cannot change his mind).
- Once a male is rejected, he **crosses** off from his list the rejecting female – he will not propose to her again

Terminology and principles of the 1950 Traditional Matching Algorithm



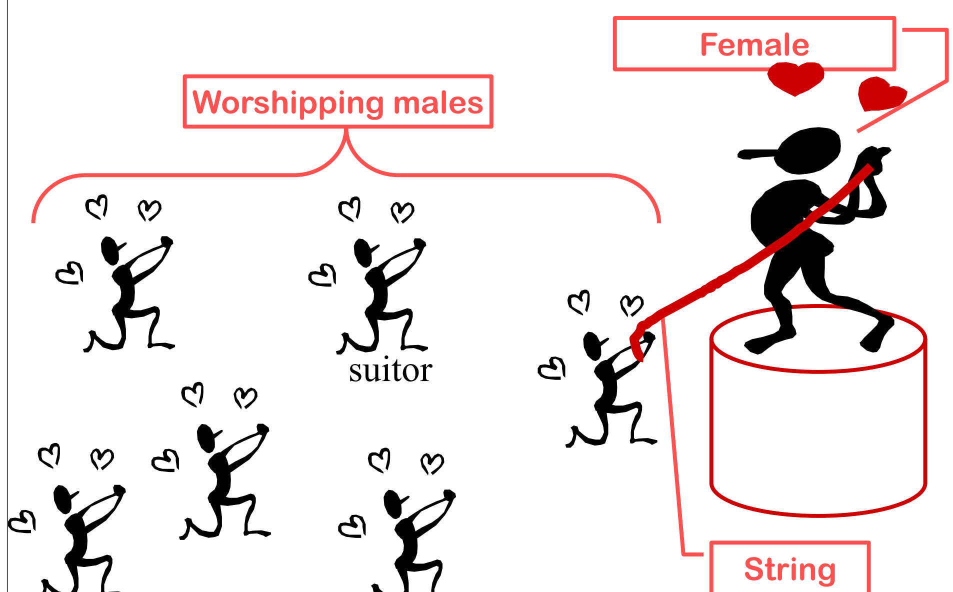
- A male can **propose** (marriage) to a female.
- A female can **reject** the proposal.
- During most of the process, a female would not accept a proposal, but would tell a proposing male "**maybe**".
- This is called "**putting the male on a string".**
- This male will come back the next day to propose again (cannot change his mind).
- Once a male is rejected, he **crosses** off from his list the rejecting female – he will not propose to her again

Terminology and principles of the 1950 Traditional Matching Algorithm



- A male can **propose** (marriage) to a female.
- A female can **reject** the proposal.
- During most of the process, a female would not accept a proposal, but would tell a proposing male "**maybe**".
- This is called "**putting the male on a string".**
- This male will come back the next day to propose again (cannot change his mind).
- Once a male is rejected, he **crosses** off from his list the rejecting female – he will not propose to her again

The Traditional Marriage Algorithm



Traditional Marriage Algorithm (TMA)

1) Repeat at each day {
- Morning
  - Each male proposes to the best female (according to his list) that has not rejected him.
- Afternoon (for each females with at least one proposal)
  - To today's best offer (according to her list): **"Maybe, come back tomorrow" (putting him on a string)**
  - All other proposals are rejected.
- Evening
  - Any rejected male crosses the rejecting female off his list.

}Until all males are on strings.

2) Each female marries the last male she just said "maybe"

Lemma (monotonically improving lemma):

If a female has a male b on a string, then she will either marry him, or marry someone she prefers over him.

Proof:

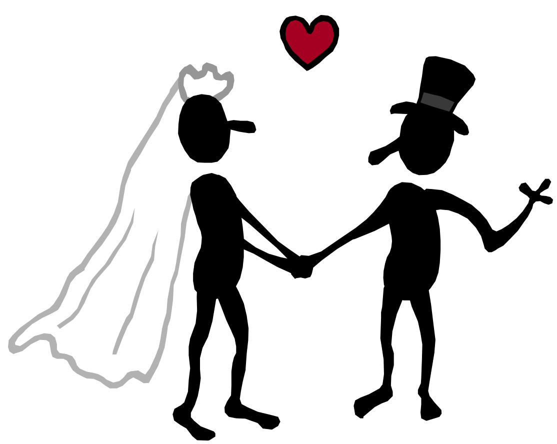
- She would only let go of *b* in order to "maybe" b' which she prefers over *b*
- She would only let go of b' for someone b" she prefers over b' etc.

When the process terminates, she is left with someone she prefers over b.

QED

Corollary:

Each female will marry her absolute favorite of the males who visit her during the Traditional Matching Algorithm (TMA)



Lemma: if the number of males are females are equal, then no male can be rejected by all the females

- Proof by contradiction.
- Suppose male b is rejected by all the females. At that point:
  - Each female must have a suitor other than b (By previous Lemma, once a female has a suitor she will always have at least one)
  - The n females have n suitors, b not among them. Thus, there are at least $n+1$ males.

  Contradiction

  QED

Theorem:
The TMA always terminates after at most n^2 days

Proof

– The total length of the lists of all males is
$n \times n = n^2$.

–Each day at least one male is rejected, so at least one female is deleted from one of the lists.

–Therefore, the number of days is bounded by the original size of the master list $= n^2$.

QED

Great! We know that TMA will terminate and produce a pairing.

But is it stable?

Theorem: TMA Produces a stable matching T.

Zod's list
----
1. Allegra
2. ….
3. **Beatrix**
4. ….
5. ….
6. **Evanora**
7. ….

Beatrix list
----
1. Mr Burn
2. **Zod**
3. Hannibal
4. …
5. **Syndrom**
6. …
7. Gus Fring
8.

- Zod and Beatrix are a rouge couple.
- Zod is married to **Evanora (6)**, but prefers **Beatrix (3)**
- **Beatrix** is married to **Syndrom (5)**, but prefers **Zod (2)**

Theorem: TMA Produces a stable matching T.

- Let m_2 and f_1 be any couple in T. *(Beatrix,Zod) in the example*
- Suppose m_2 prefers f_1 (*Beatrix*) over his wife f_2 *(Evanora)*.
- We will argue that f_1 prefers her husband over m_2 ***(Zod)***
- During TMA, male m_2 (*Zod*) proposed to f_1 ***(Beatrix)*** before he proposed to f_2.
- Hence, at some point f_1 rejected m_2 for someone she preferred.
- By the Monotonic Improvement lemma, the male (Zod) that f_2 (*Beatrix*) married was also preferable to m_2
- Thus, every male will be rejected by any female he prefers to his wife.
- T is stable. QED.

Zod's list
----
1. Allegra
2. ….
3. **Beatrix**
4. ….
5. ….
6. **Evanora**
7. ….

Beatrix list
----
1. Mr Burn
2. **Zod**
3. Hannibal
4. …
5. **Syndrom**
6. …
7. Gus Fring
8.

- Zod and Beatrix are a rouge couple.
- Zod is married to **Evanora (6)**, but prefers **Beatrix (3)**
- **Beatrix** is married to **Syndrom (5)**, but prefers **Zod (2)**