CSE 417, Winter 2012

Introduction, Examples, and Analysis

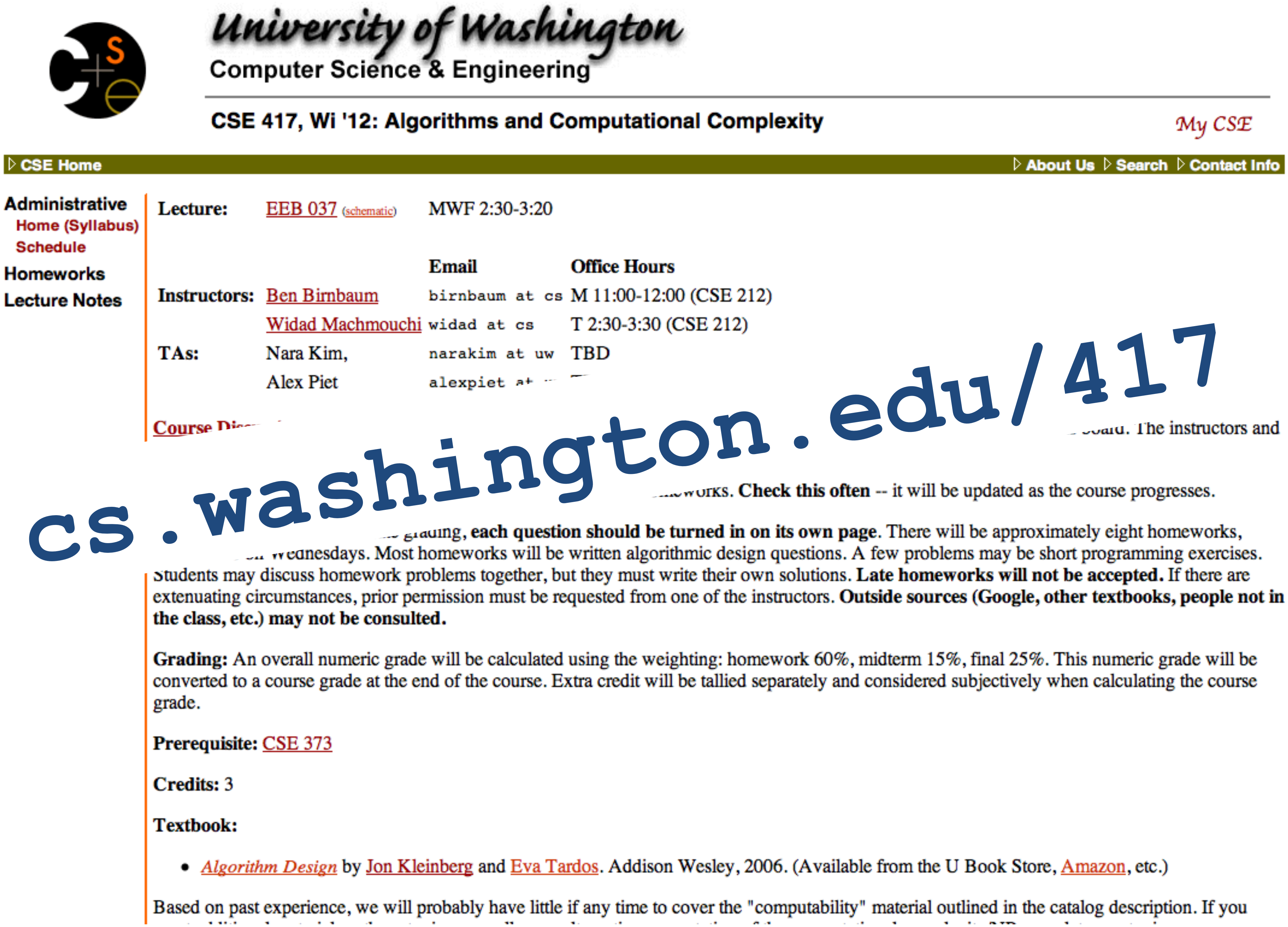
Ben Birnbaum

Widad Machmouchi

Slides adapted from Larry Ruzzo,
Steve Tanimoto, and Kevin Wayne

CSE 417: Algorithms and Computational Complexity

- Instructors:
  - Ben Birnbaum (Computer Science Ph.D.)
  - Widad Machmouchi (Computer Science Ph.D.)
  - (Mostly) team-teaching by unit
- TAs:
  - Nara Kim (Computer Science B.S.)
  - Alex Piet (Applied Math M.S.)



University of Washington
Computer Science & Engineering

CSE 417, Wi '12: Algorithms and Computational Complexity

My CSE

▷ CSE Home ▷ About Us ▷ Search ▷ Contact Info

Administrative
Home (Syllabus)
Schedule
Homeworks
Lecture Notes

Lecture: EEB 037 (schematic) MWF 2:30-3:20

		Email	Office Hours
Instructors:	Ben Birnbaum	birnbaum at cs	M 11:00-12:00 (CSE 212)
	Widad Machmouchi	widad at cs	T 2:30-3:30 (CSE 212)
TAs:	Nara Kim,	narakim at uw	TBD
	Alex Piet	alexpiet at	

Course Dis ...oard. The instructors and

...works. **Check this often** -- it will be updated as the course progresses.

...grading, **each question should be turned in on its own page**. There will be approximately eight homeworks, ...n Wednesdays. Most homeworks will be written algorithmic design questions. A few problems may be short programming exercises. Students may discuss homework problems together, but they must write their own solutions. **Late homeworks will not be accepted.** If there are extenuating circumstances, prior permission must be requested from one of the instructors. **Outside sources (Google, other textbooks, people not in the class, etc.) may not be consulted.**

Grading: An overall numeric grade will be calculated using the weighting: homework 60%, midterm 15%, final 25%. This numeric grade will be converted to a course grade at the end of the course. Extra credit will be tallied separately and considered subjectively when calculating the course grade.

Prerequisite: CSE 373

Credits: 3

Textbook:

- *Algorithm Design* by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006. (Available from the U Book Store, Amazon, etc.)

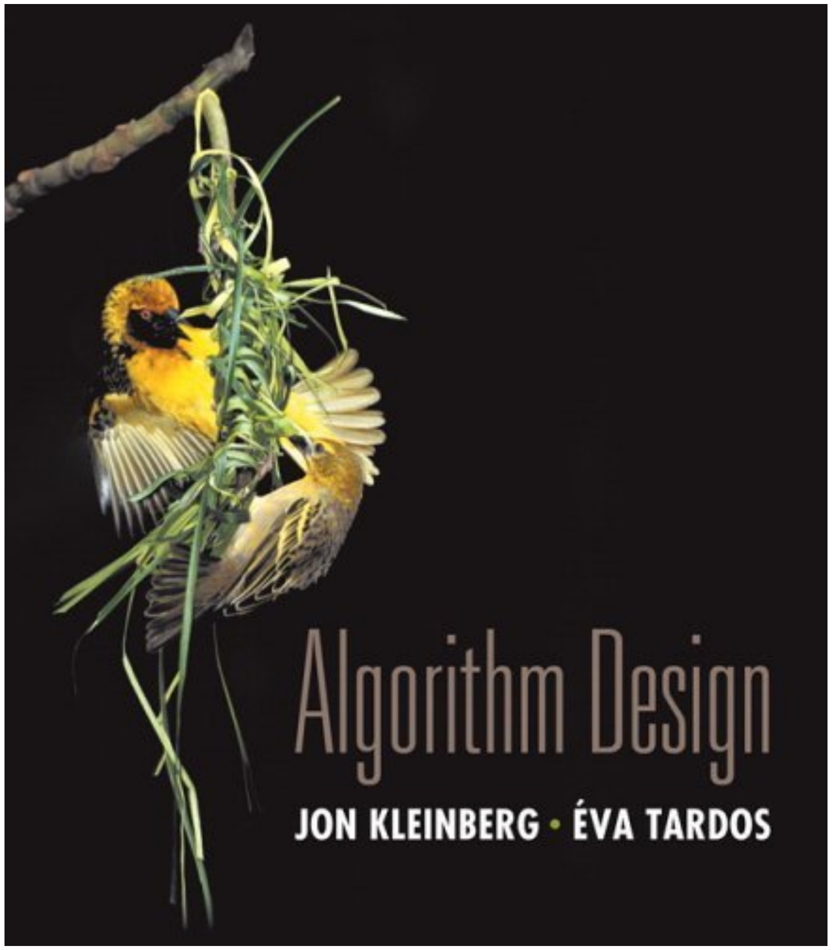
Based on past experience, we will probably have little if any time to cover the "computability" material outlined in the catalog description. If you

cs.washington.edu/417

Other resources
(all linked from website)

- Catalyst discussion board (use it!)
- Course email list
- Schedule
- Office hours
  - Ben: M 11-12
  - Widad: T 2:30-3:30
  - Nara and Alex (TBD)

Textbook



What you have to do

- Homework (60%)
  - Roughly 8 weekly assignments, due on Wednesday.
    - Mostly written design, analysis, and argument.
    - A couple of small programming assignments.
  - Late assignments not accepted
  - **Turn in each question on its own page**
  - Can discuss with classmates, writeups must be your own. Do not consult other textbooks, Google, etc.
  - Extra credit counted separately and considered subjectively.
  - Extra credit given for exceptional solutions.
- In-class midterm, Feb. 8 (15%)
- Final, Mar. 13 2:30-4:30 (25%)
- This class stresses **problem solving** and **proofs**. These are *hard*. We will curve generously.
- Ask questions!

Homework 0

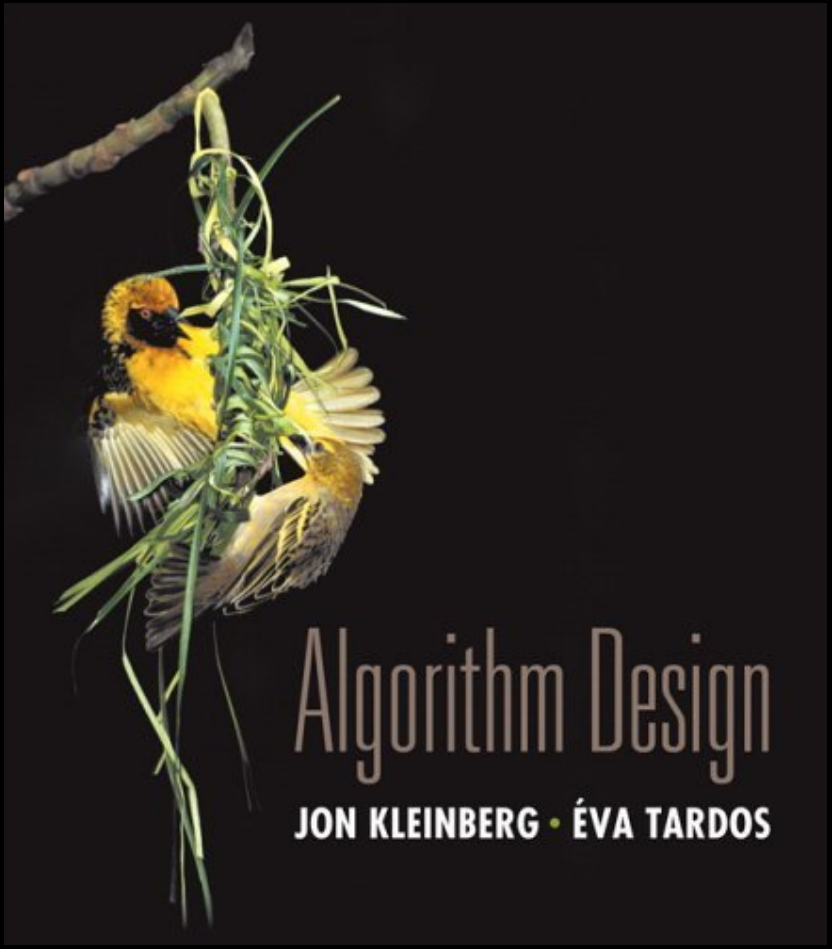
- Complete our online background survey by **this Friday, January 6**.
- Will count for 10 homework points (about ¼ of a typical homework).
- No wrong answers.
- Available on website.

What the course is about

- Algorithm design (first 7 weeks)
  - Design methods (greedy, divide & conquer, dynamic programming, etc.)
  - Analysis of algorithms, efficiency
  - Correctness proofs
- Intractability (last 3 weeks)
  - Important to know when problems *cannot* be solved efficiently.
  - NP-completeness theory captures many problems that (probably) cannot be solved efficiently.
- Schedule is available online

Reading

- KT, Chapter 1
- KT, Chapter 2.1 – 2.4



Chapter 1

Introduction: Some Representative Problems

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

1.1 A First Problem: Stable Matching

Motivation: a job application process.

Setting. College seniors applying for jobs. Each student has preferences on employers. Each employer has preferences on students.

Goal. Given a set of preferences, assign students to employers in a self-reinforcing way.

Unstable pair: applicant a and employer e are unstable if:

- a prefers e to her assigned employer.
- e prefers a to one of its accepted students.

Stable assignment. Assignment with no unstable pairs.

- Natural and desirable condition.
- Individual self-interest will prevent any applicant/employer deal from being made.

An abstraction: the Stable Matching Problem

Goal. Given n men and n women, find a "suitable" matching.

- Participants rate members of opposite sex.
- Each man lists women in order of preference from best to worst.
- Each woman lists men in order of preference from best to worst.

favorite ↓ … least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ … least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Zeus	Yancey	Xavier
Clare	Zeus	Yancey	Xavier

Women's Preference Profile

An abstraction: the Stable Matching Problem

Perfect matching: everyone is matched monogamously.

- Each man gets exactly one woman.
- Each woman gets exactly one man.

Stability: no incentive for some pair of participants to undermine assignment by joint action.

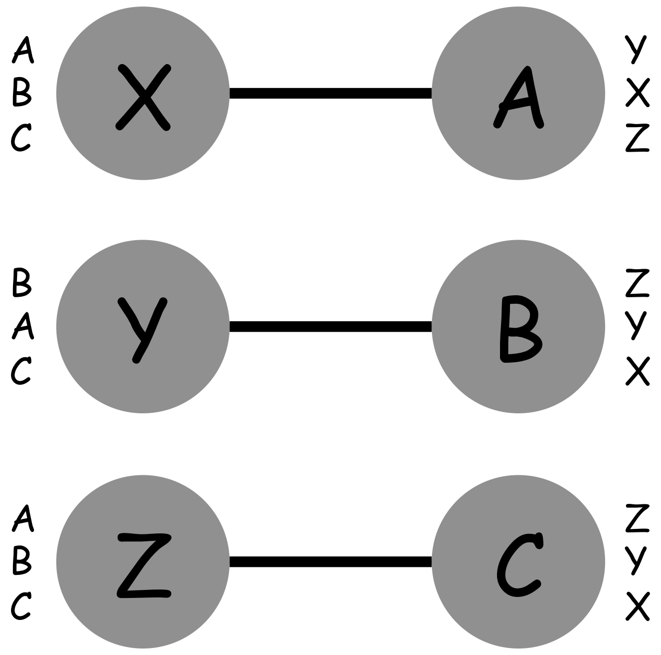
- In matching M, an unmatched pair m-w is unstable if man m and woman w prefer each other to current partners.
- Unstable pair m-w could each improve by eloping.

Stable matching: perfect matching with no unstable pairs.

Stable matching problem. Given the preference lists of n men and n women, find a stable matching if one exists.

Stable Matching Problem

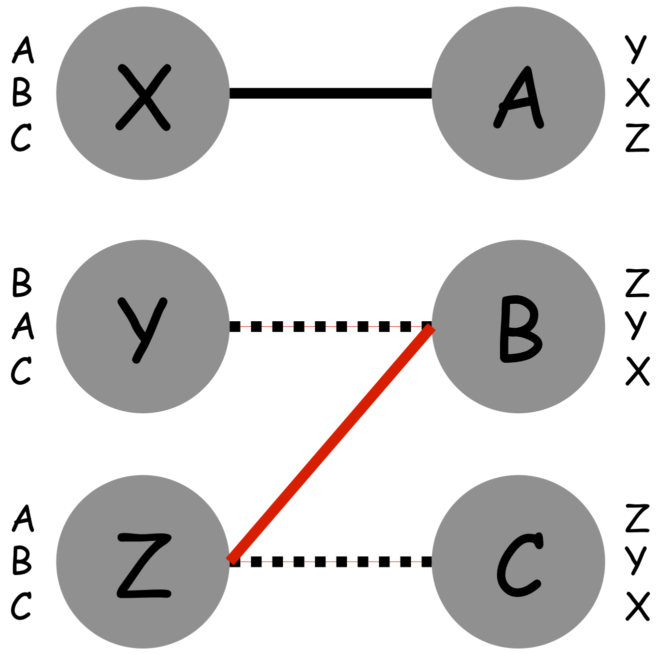
Q. Is assignment X-A, Y-B, Z-C stable?



Stable Matching Problem

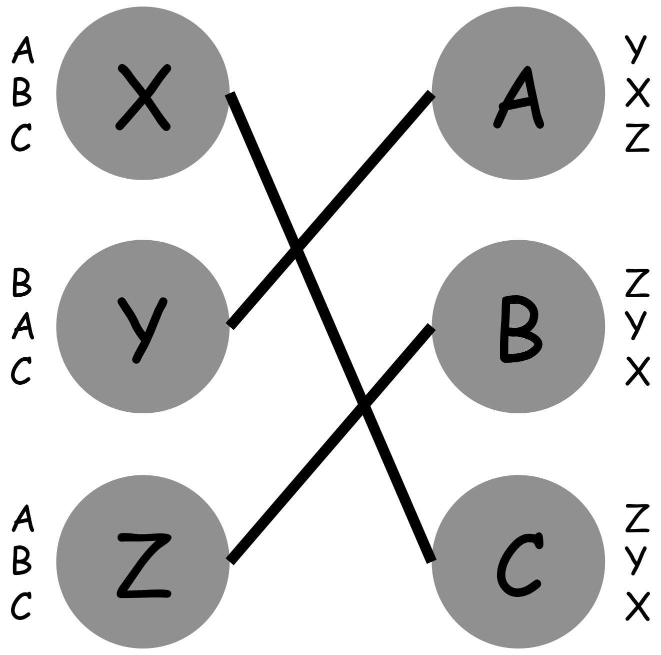
Q. Is assignment X-A, Y-B, Z-C stable?

A. No. Zeus and Berta will hook up. (They are an unstable pair.)



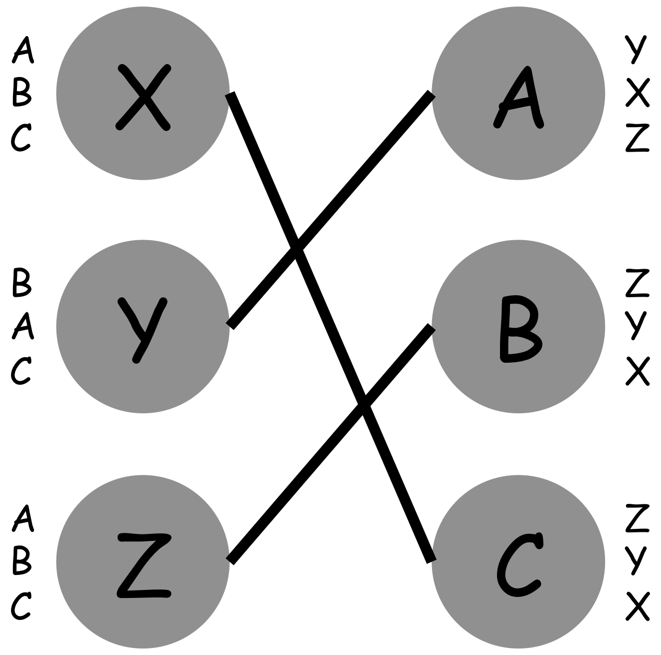
Stable Matching Problem

Q. Is assignment X-C, Y-A, Z-B stable?



Stable Matching Problem

Q. Is assignment X-C, Y-A, Z-B stable?
A. Yes. (No unstable pairs.)



Q. Do stable matchings always exist?
A. Not obvious.

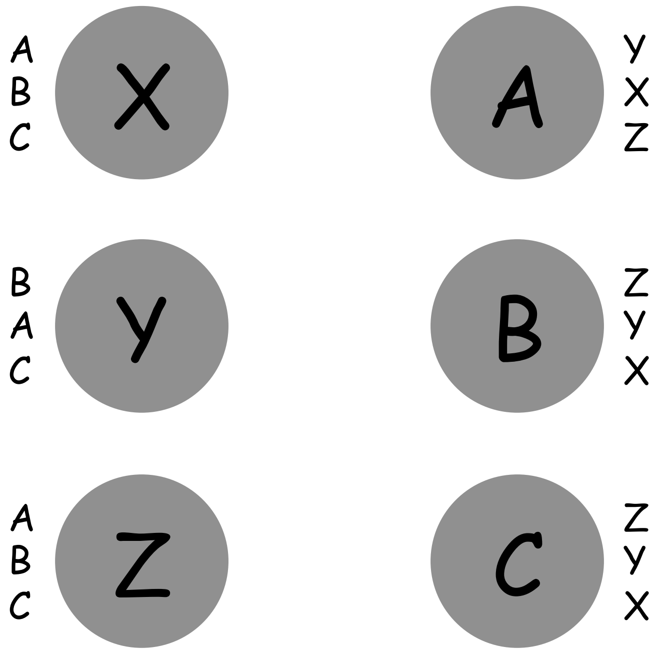
Propose-And-Reject Algorithm

Propose-and-reject algorithm. [Gale-Shapley 1962] Intuitive method that guarantees to find a stable matching.

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

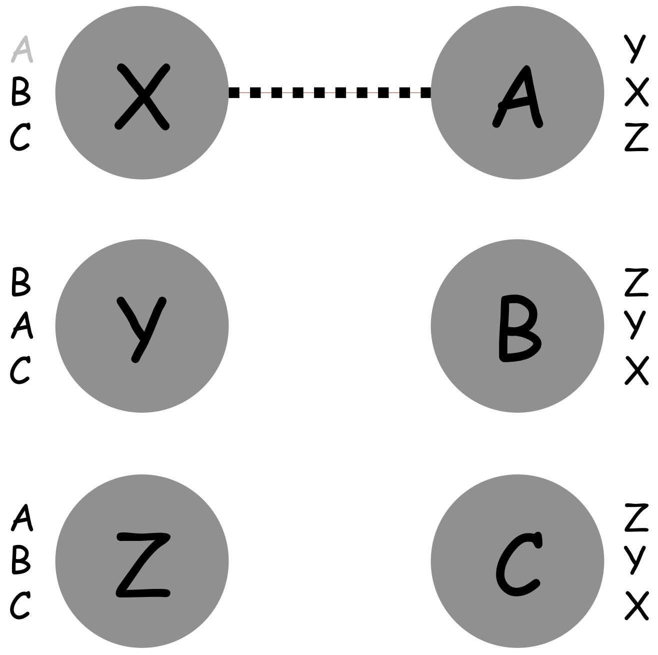
Propose-and-Reject Algorithm, Illustrated

```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



Propose-and-Reject Algorithm, Illustrated

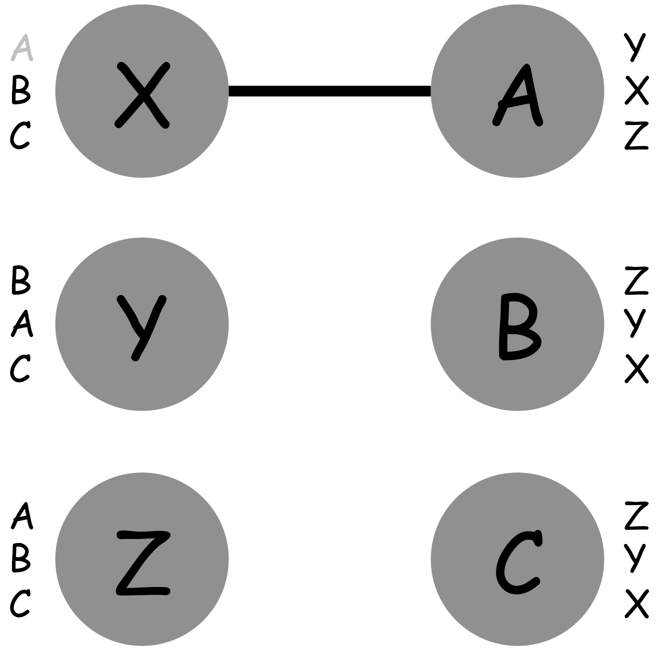
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



X proposes to A.

Propose-and-Reject Algorithm, Illustrated

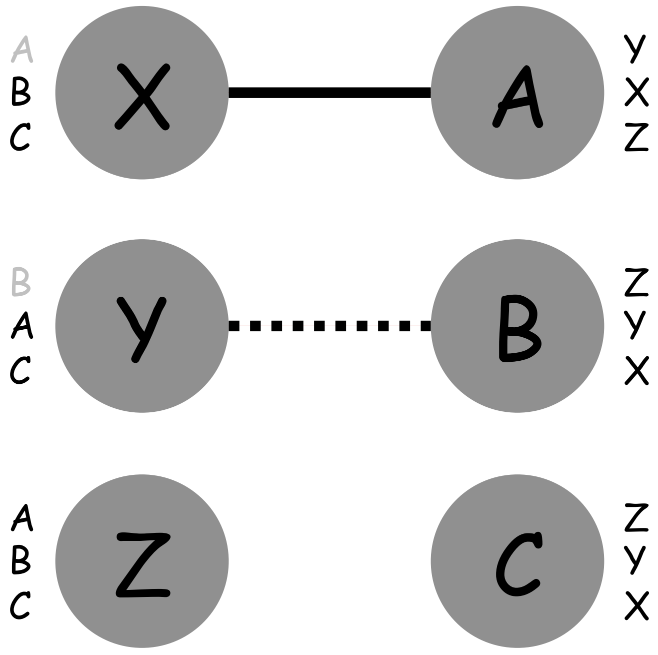
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



A accepts X's proposal.

Propose-and-Reject Algorithm, Illustrated

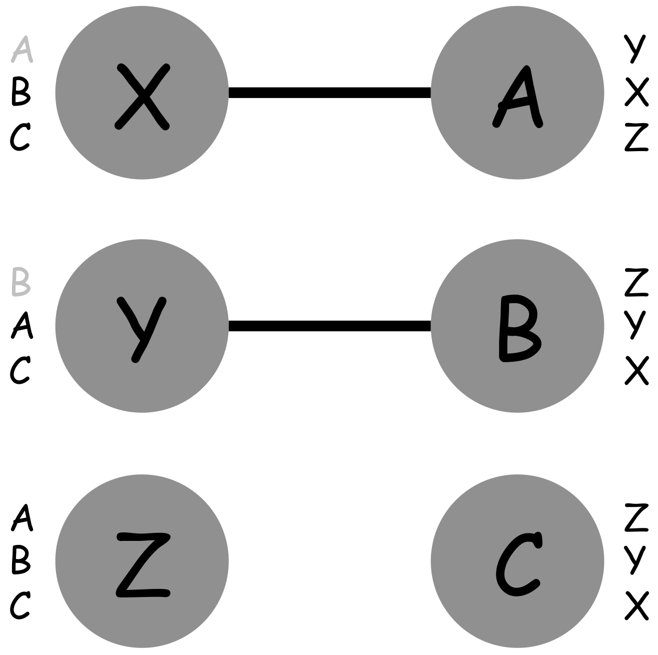
```
Initialize each person to be free.
while (some man is free and
           hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



Y proposes to B.

Propose-and-Reject Algorithm, Illustrated

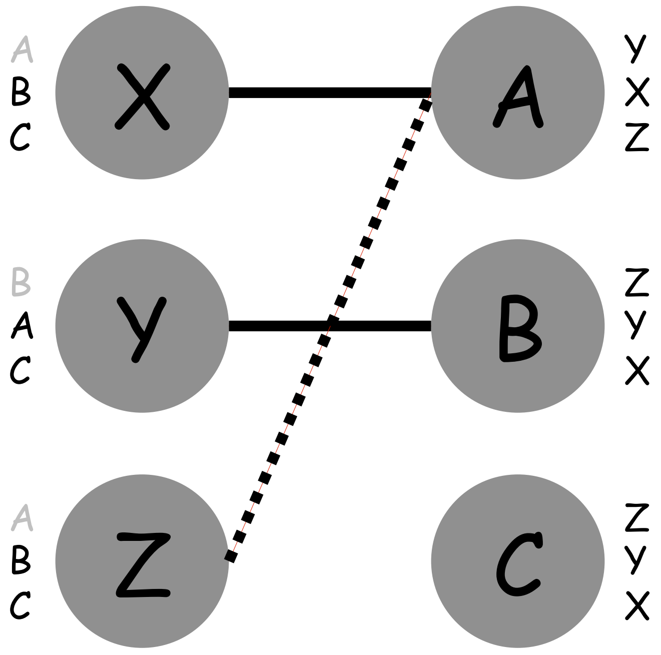
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



B accepts Y's proposal.

Propose-and-Reject Algorithm, Illustrated

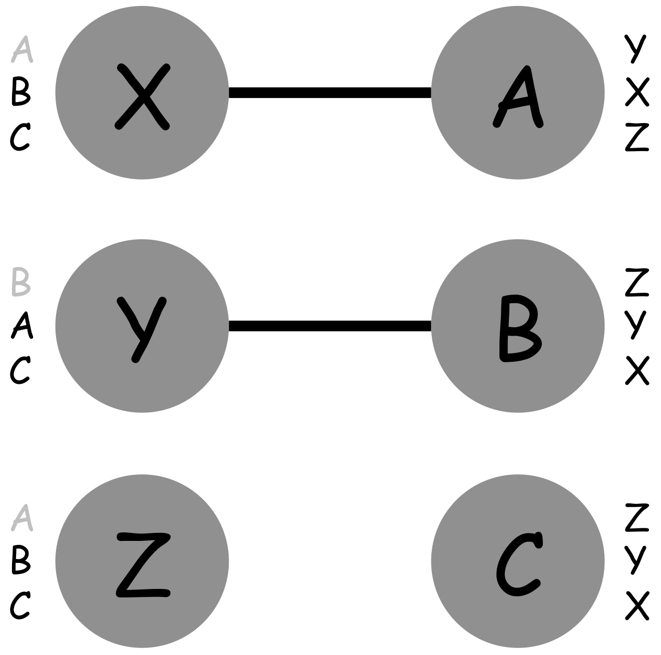
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



Z proposes to A.

Propose-and-Reject Algorithm, Illustrated

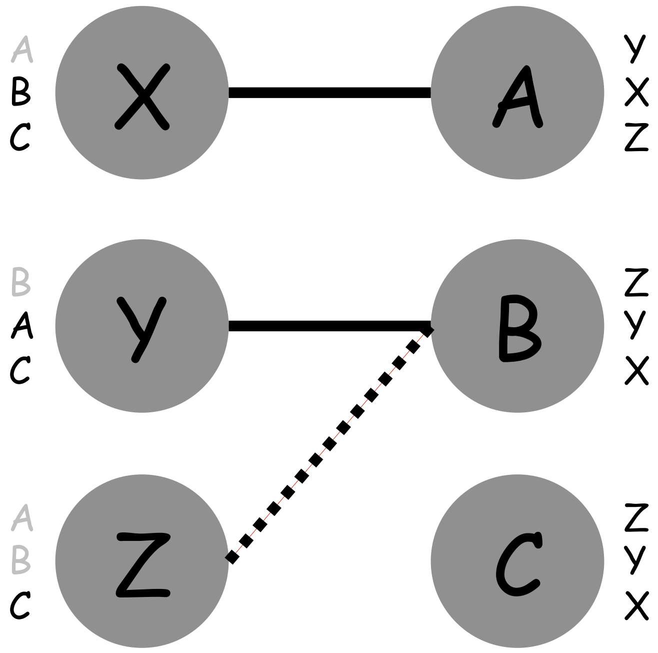
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



A rejects Z's proposal. (She prefers X.)

Propose-and-Reject Algorithm, Illustrated

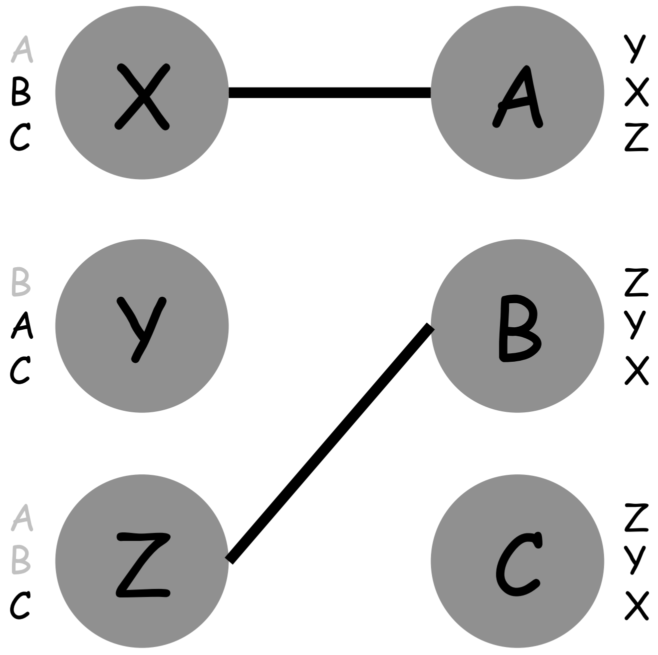
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



Z proposes to B.

Propose-and-Reject Algorithm, Illustrated

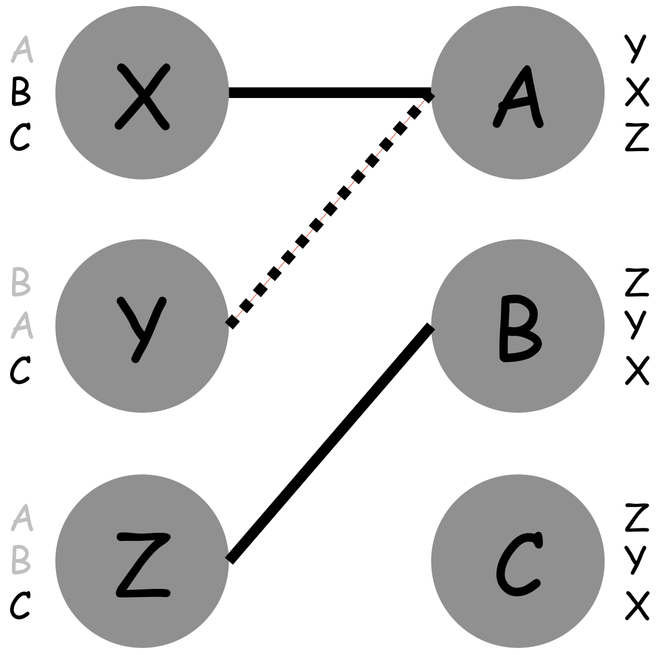
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



B accepts Z's proposal (and breaks engagement with Y).

Propose-and-Reject Algorithm, Illustrated

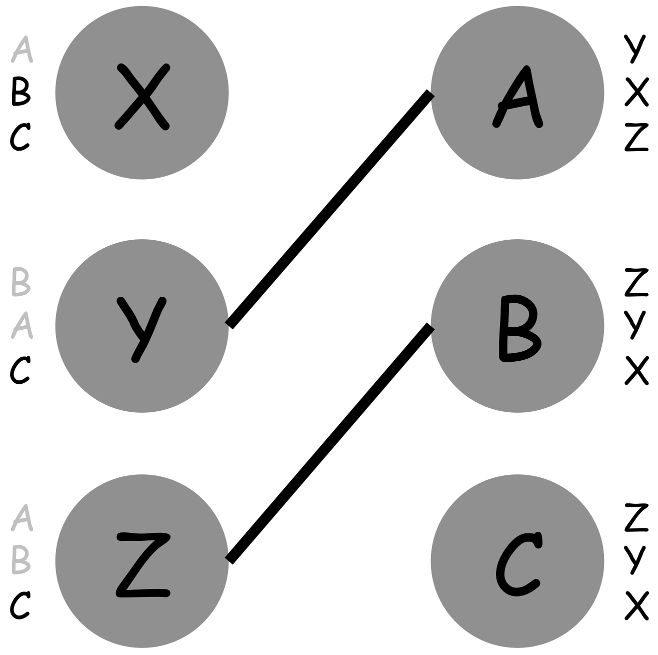
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



Y proposes to A.

Propose-and-Reject Algorithm, Illustrated

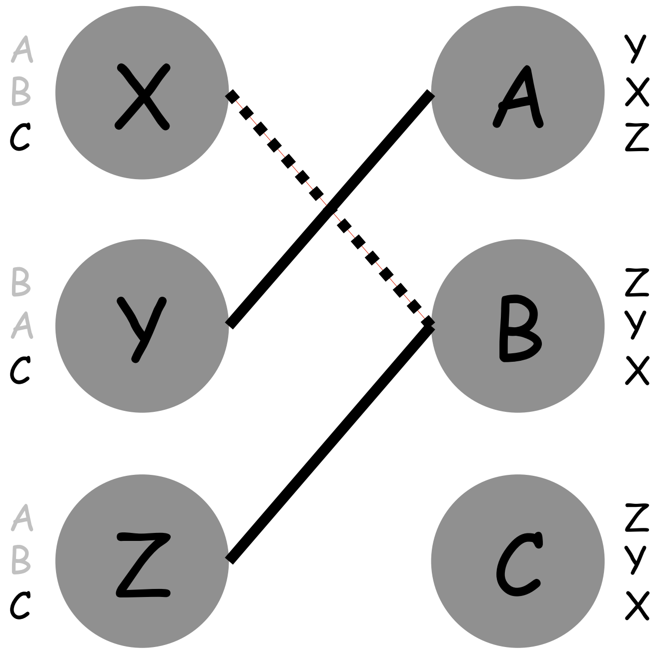
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



A accepts Y's proposal (and breaks engagement with X).

Propose-and-Reject Algorithm, Illustrated

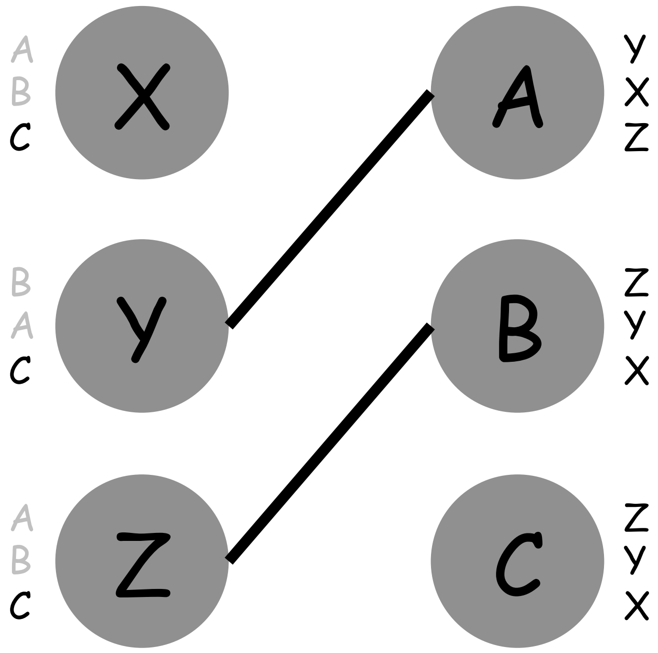
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



X proposes to B.

Propose-and-Reject Algorithm, Illustrated

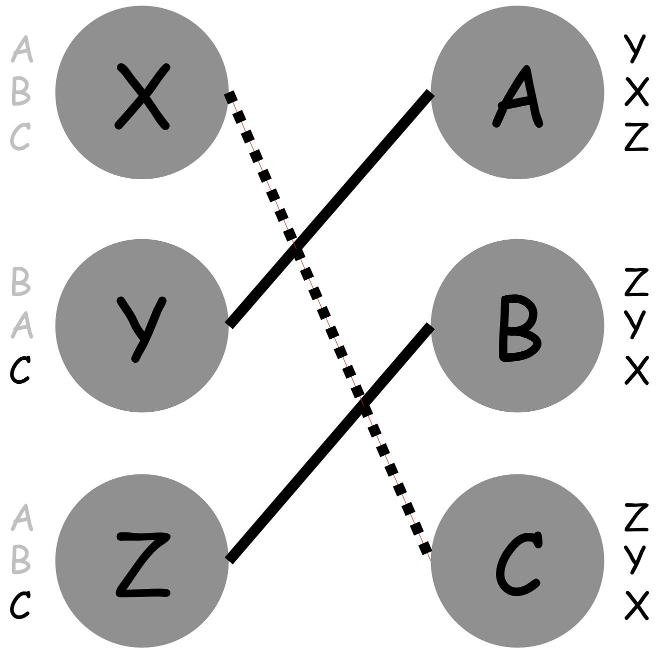
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



B rejects X's proposal. (She prefers Z.)

Propose-and-Reject Algorithm, Illustrated

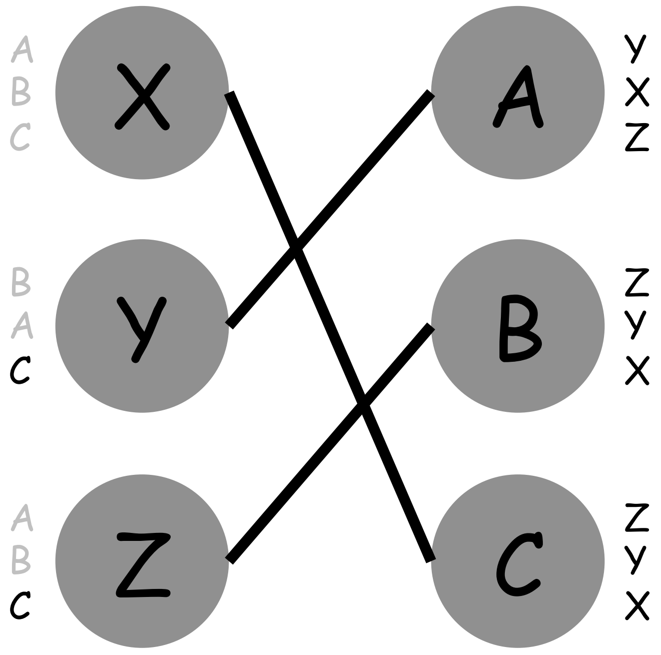
```
Initialize each person to be free.
while (some man is free and
       hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



X proposes to C.

Propose-and-Reject Algorithm, Illustrated

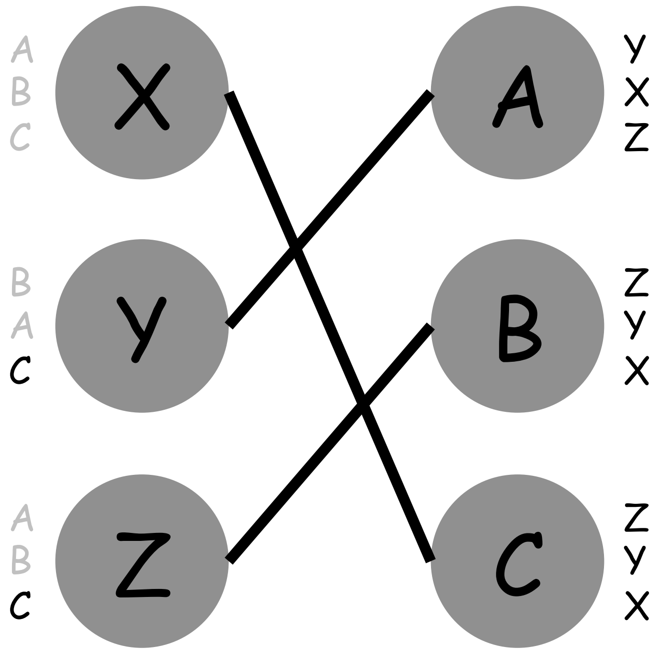
```
Initialize each person to be free.
while (some man is free and
           hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



C accepts X's proposal.

Propose-and-Reject Algorithm, Illustrated

```
Initialize each person to be free.
while (some man is free and
           hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has
        not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged
        assign m' to be free
    else
        w rejects m
}
```



A stable matching!

Q. Does this algorithm always work?

A. Yes! (We need to prove this.)

Proof of Correctness: Termination

Observation 1. Once a woman is matched, she never becomes unmatched; she only "trades up."

Observation 2. Men propose to women in decreasing order of preference.

Claim. Algorithm terminates after at most n^2 iterations of while loop.
Pf. Each time through the while loop a man proposes to a new woman. There are only n^2 possible proposals. ▪

Proof of Correctness: Perfection

Claim. All men and women get matched.

Pf. (by contradiction)

- Suppose, for the sake of contradiction, that there exists someone who is not matched.
- Then since the number of men matched is the same as the number of women, there must exist both a man and a woman that are not matched. Call them m and w.
- By Observation 1 (once a woman is matched, she stays matched), w was never proposed to.
- But, because m is unmatched at the end of the algorithm, the only way the while loop could have terminated is if he proposed to everyone, including w.
- This is a contradiction, so we conclude that all men, and hence all women, must be matched. ▪

Proof of Correctness: Stability

Claim. No unstable pairs.
Pf. (by contradiction)

- Suppose m-w is an unstable pair: each prefers each other to partner in Gale-Shapley matching.

- Case 1: m never proposed to w.
  $\Rightarrow$ m prefers his GS partner to w. $\swarrow$ Obs. 2: men propose in decreasing order of preference
  $\Rightarrow$ m-w is stable.

- Case 2: m proposed to w.
  $\Rightarrow$ w rejected m (right away or later)
  $\Rightarrow$ w prefers her GS partner to m. $\longleftarrow$ Obs. 1: women only trade up
  $\Rightarrow$ m-w is stable.

- In either case m-w is stable, a contradiction. ▪

Summary

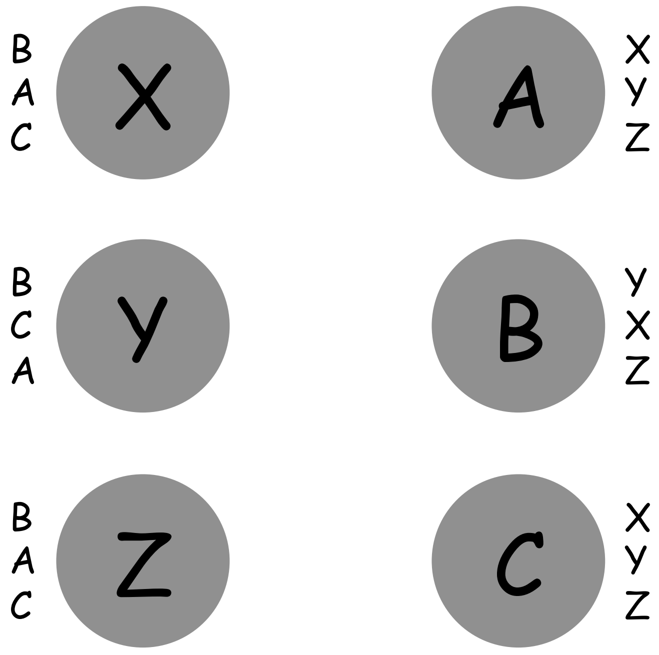
Stable matching problem. Given n men and n women, and their preferences, find a stable matching if one exists.

Remember, it's not even clear if a stable matching always exists!

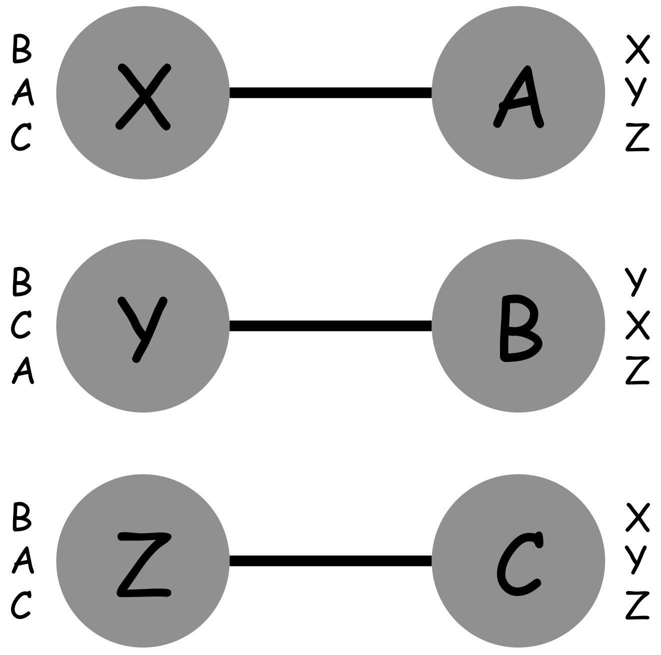
Gale-Shapley algorithm. Shows that a stable matching always exists by giving an algorithm guaranteed to find one for any problem instance.

That's pretty cool.

Warm up



Warm up



- This is stable even though Z and C hate each other.
- Why did everyone get the same answer? (Theorem 1.7 in book).

Who cares? Matching Residents to Hospitals

Before 1952:

“In general, hospitals benefited from filling their positions as early as possible, and applicants benefited from delaying acceptance of positions. The combination of these factors lead to offers being made for positions up to two years in advance. While efforts made to delay the start of the application process were somewhat effective, they ultimately resulted in very short deadlines for responses by applicants, and the opportunities for dissatisfaction on the part of both applicants and hospitals remained.” (Gusfield and Irving 1989, via Wikipedia).

After 1952:

The National Resident Matching Program (NRMP)

Who cares? Matching Residents to Hospitals

Men $\approx$ hospitals, Women $\approx$ med school residents.

Variant 1. Some participants declare others as unacceptable.

resident A unwilling to work in Cleveland

Variant 2. Unequal number of men and women.

Variant 3. Limited polygamy.

hospital X wants to hire 3 residents

Def. Matching S unstable if there is a hospital h and resident r such that:

- h and r are acceptable to each other; and
- either r is unmatched, or r prefers h to her assigned hospital; and
- either h does not have all its places filled, or h prefers r to at least one of its assigned residents.

A variant of the GS algorithm works, and is used!

Lessons Learned

Powerful ideas learned in course.

- Isolate underlying structure of problem.
- Create useful and efficient algorithms that are provably correct.

Basics of Algorithm Analysis

"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." - *Francis Sullivan*

(Preliminary) Survey Results

With 43 respondents,

- 9% are not at all comfortable with asymptotic analysis (Big “Oh” notation)
- 47% are somewhat comfortable
- 44% are very comfortable

What does it mean to bound the running time of an algorithm?

Depends on how you measure it.

- Which computer?
- Which programming language?
- Clock time, or something else?

Even if we fix a model, it still depends on the input.

What does it mean to bound the running time of an algorithm?

Any bound depends on the size of the input, e.g. T(n) = 3n^2 + 5n – 2.

But there are many different inputs of the same size.

How should one bound apply to all of them?

Complexity analysis

Problem size n

- *Best-case* complexity:
  - fastest time on any input of size n
- *Average-case* complexity:
  - average time on inputs of size n
- *Worst-case* complexity:
  - slowest time on any input of size n

Pros and cons:

Best-case

- unrealistic oversell

Average-case

- over what probability distribution? (different people may have different “average” problems)
- analysis often hard

Worst-case?

- a fast algorithm has a comforting guarantee
- maybe too pessimistic

Why Worst-Case Analysis?

Comforting guarantee.

Appropriate for time-critical applications, e.g. avionics.

Unlike Average-Case, no debate about what the right definition is.

Analysis often easier.

Result is often representative of "typical" problem instances.

Of course there are exceptions…

What about the model?

Let's say we have bounded the worst-case running time on a particular computer as

$$T(n) = 3n^2 + 5n - 2.$$

What about a computer that's twice as fast?

What if the compiler changes?

The running time could change.

We need a way to describe running times that is independent of this.

That's where asymptotic analysis comes in.

That’s where asymptotic analysis comes in.

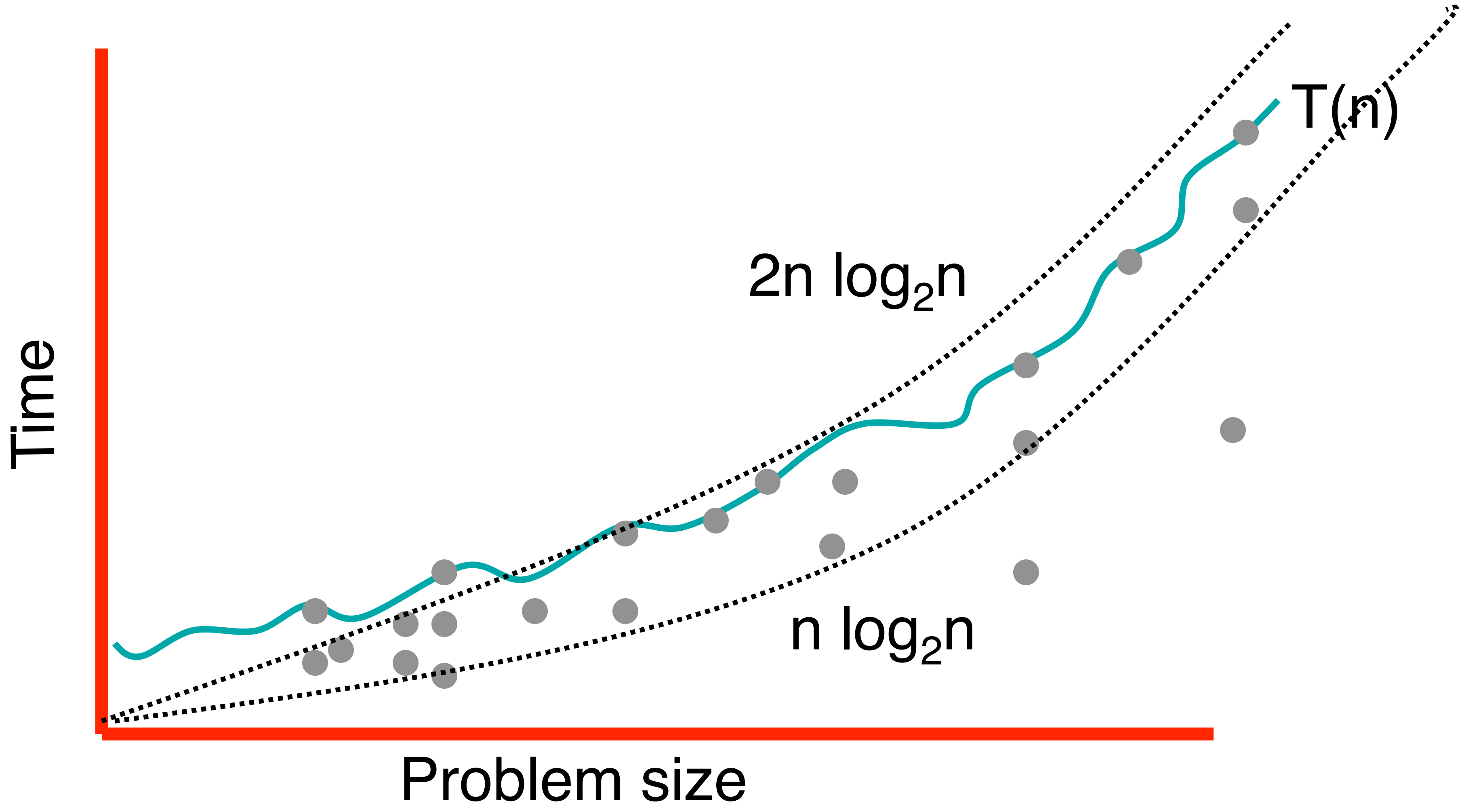
Given two functions f and g: $N \to R$

f(n) is $O(g(n))$ iff there is a constant $c>0$ so that
f(n) is eventually always $\leq c\, g(n)$

f(n) is $\Omega(g(n))$ iff there is a constant $c>0$ so that
f(n) is eventually always $\geq c\, g(n)$

f(n) is $\Theta(g(n))$ iff f(n) is $O(g(n))$ and f(n) is $\Omega(g(n))$.

Complexity



Working with O-Ω-Θ notation

Claim: For any a, and any $b>0$, $(n+a)^b$ is $\Theta(n^b)$

$(n+a)^b \leq (2n)^b$ for $n \geq |a|$

$= 2^b n^b$

$= cn^b$ for $c = 2^b$

so $(n+a)^b$ is $O(n^b)$

$(n+a)^b \geq (n/2)^b$ for $n \geq 2|a|$ (even if $a < 0$)

$= 2^{-b} n^b$

$= c'n$ for $c' = 2^{-b}$

so $(n+a)^b$ is $\Omega(n^b)$

Working with O-Ω-Θ notation

Claim: For any a, b>1 $\log_a n$ is Θ ($\log_b n$)

$\log_a b = x$ means $a^x = b$

$a^{\log_a b} = b$

$(a^{\log_a b})^{\log_b n} = b^{\log_b n} = n$

$(\log_a b)(\log_b n) = \log_a n$

$c \log_b n = \log_a n$ for the constant $c = \log_a b$

So :

$\log_b n = \Theta(\log_a n) = \Theta(\log n)$

Asymptotic Bounds for Some Common Functions

Polynomials:

$a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$

Logarithms:

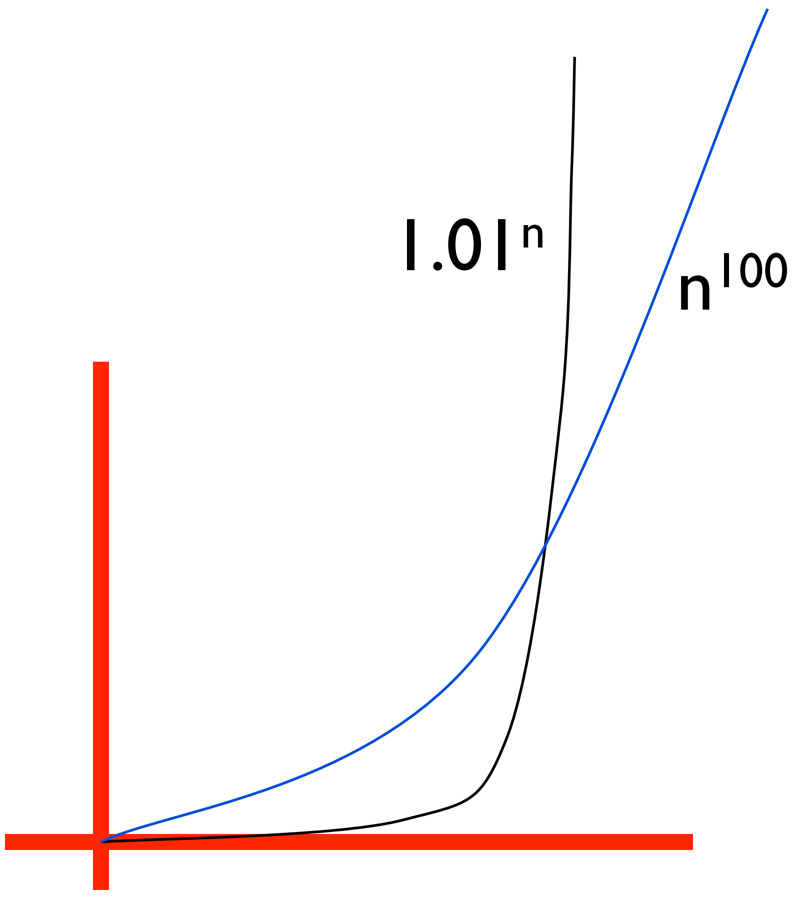
For all $x > 0$, $\log n = O(n^x)$

log grows slower than every polynomial

Asymptotic Bounds for Some Common Functions

Exponentials. For all $r > 1$ and all $d > 0$, $n^d = O(r^n)$.

every exponential
grows faster than
every polynomial



"One-Way Equalities"

What's ok to write?

$2n^2 + 5\ n$ is $O(n^3)$

$2n^2 + 5\ n = O(n^3)$

~~$O(n^3) = 2n^2 + 5\ n$~~

Bottom line:

OK to put big-O in R.H.S. of equality, but not left.

[Better, but uncommon, notation: $T(n) \in O(f(n))$.]

Just the right level of precision

- It’s not realistic to be more precise than up to a constant factor.
- On the other hand, order of growth really matters…

Here's why order of growth matters

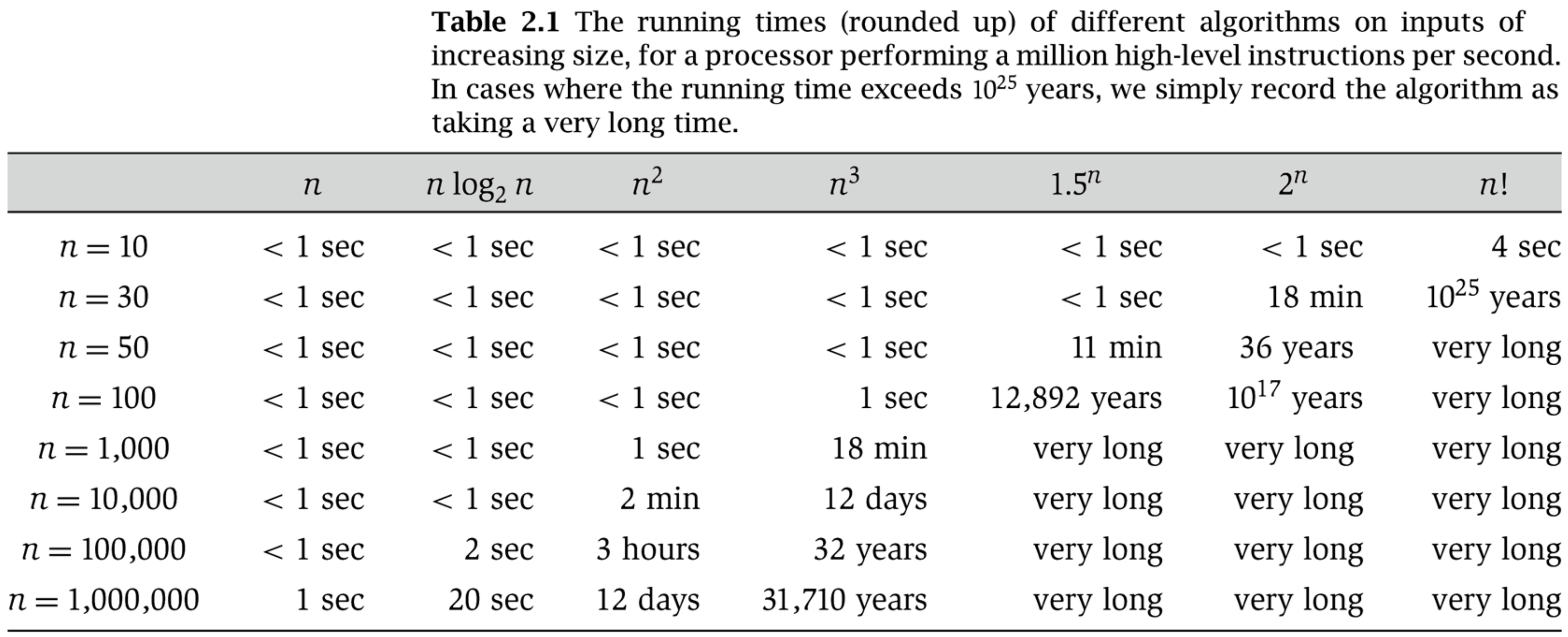


Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	4 sec
$n = 30$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	18 min	10^{25} years
$n = 50$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	11 min	36 years	very long
$n = 100$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	$<$ 1 sec	$<$ 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	$<$ 1 sec	$<$ 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	$<$ 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

All of these functions have different *orders of growth.* That is, for no two functions f and g is it the case that f = Θ(g).

Now back to the model

With asymptotic notation, we don't worry too much about the model of computation.

We just need something reasonable.

Time ≈ # of instructions executed in a simple programming language

- only simple operations (+,*,-,=,if,call,…)
- each operation takes one time step
- each memory access takes one time step
- no fancy stuff (add these two matrices, copy this long string, …) built in; write it/charge for it as above

Is this reasonable?

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

It's good pseudo-code, but not clear if every step can be implemented in constant time.

So what is efficient?

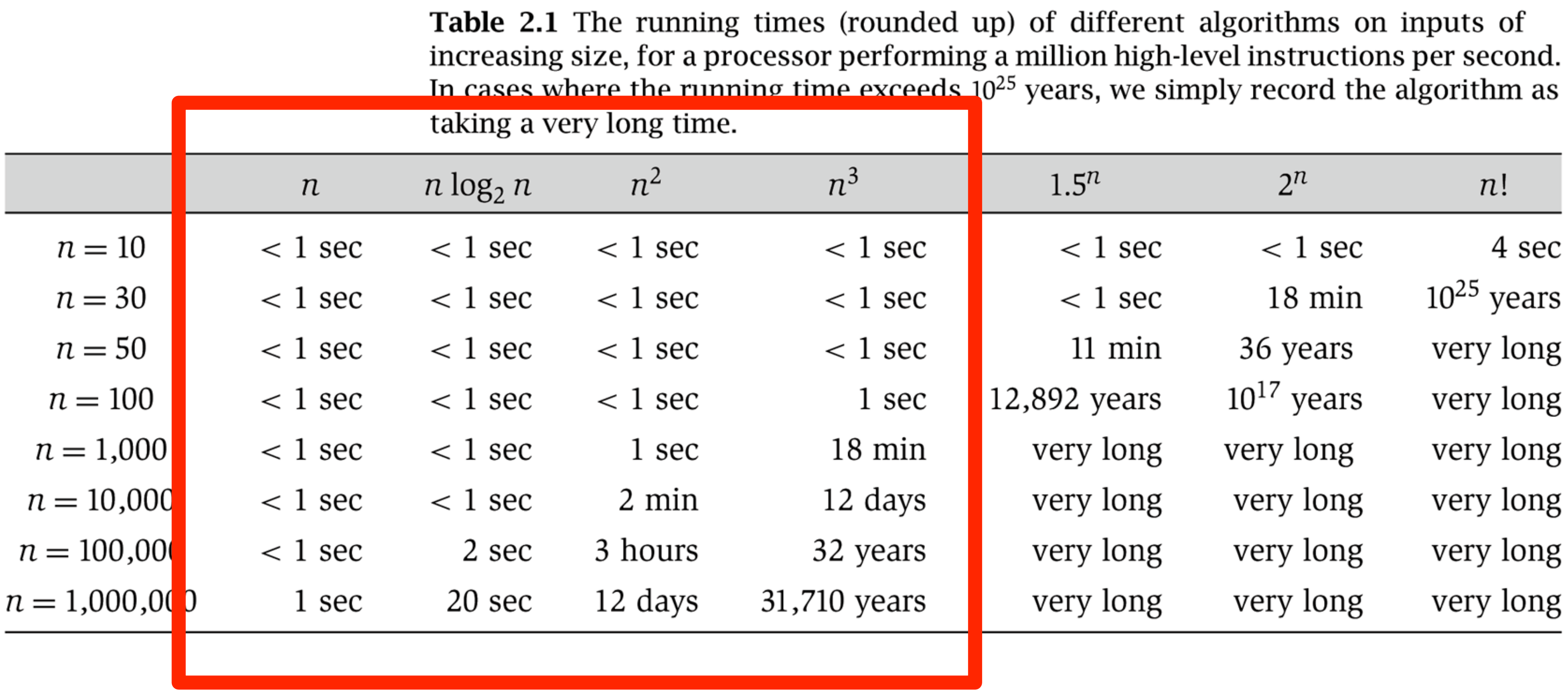


Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	4 sec
$n = 30$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	18 min	10^{25} years
$n = 50$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	11 min	36 years	very long
$n = 100$	$<$ 1 sec	$<$ 1 sec	$<$ 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	$<$ 1 sec	$<$ 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	$<$ 1 sec	$<$ 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	$<$ 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Polynomial time: running time is $O(n^d)$ for some constant d independent of the input size n

Why Polynomial Time?

Not a perfect definition:

n^{100} vs. $n^{1+.02(\log n)}$

But it generally works in practice.

Usually, polynomial is faster than the “brute force” solution, so such a solution signifies insight.

Negatable.

Back to Stable Matching

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

We showed that the algorithm takes $O(n^2)$ iterations. But can we also show it takes $O(n^2)$ time?
Yes!

Efficient Implementation

We need to perform the following operations in constant time:

- Identify a free man.
- For a man m, identify the highest-ranked woman to whom he has not yet proposed.
- Decide if a woman w is currently engaged, and if she is, identify her current partner.
- For a woman w and two men m and m', be able to decide which of m or m' is preferred by w.

Represent men and women as numbers 1 through n.

Maintain the following data structures

- FreeList
- ManPref[m, i], WomanPref[w, i]
- Next[m]
- Current[w]
- Ranking[w, m]

Efficient Implementation

Women rejecting/accepting.

- Does woman `w` prefer man `m` to man `m'`?
- For each woman, create inverse of preference list of men.
- Constant time access for each query after O(n) preprocessing.

Amy	1st	2nd	3rd	4th	5th	6th	7th	8th
WomanPref	8	3	7	1	4	5	6	2

Amy	1	2	3	4	5	6	7	8
Ranking	4th	8th	2nd	5th	6th	7th	3rd	1st

```
for i = 1 to n
   ranking[pref[i]] = i
```

Amy prefers man 3 to 6
since `ranking[3]` < `ranking[6]`

2 7