CS180: Summer 2021

Introduction to Algorithms and Complexity

The Gale-Shapely Algorithm for Stable Matching

Prof. Mark Burgin

TAs: Yunqi Guo guoyunqi@gmail.com

Ling Ding lingding@cs.ucla.edu

Logistics

- Homework 1 due Jul 2, 11:59 PM
- Submit to CCLE-Gradescope

- No homework this week

Stable Matching Summary

Stable matching problem. Given preference profiles of n men and n women, find a stable matching.

no man and woman prefer to be with each other than assigned partner

Gale-Shapley algorithm. Finds a stable matching in $O(n^2)$ time.

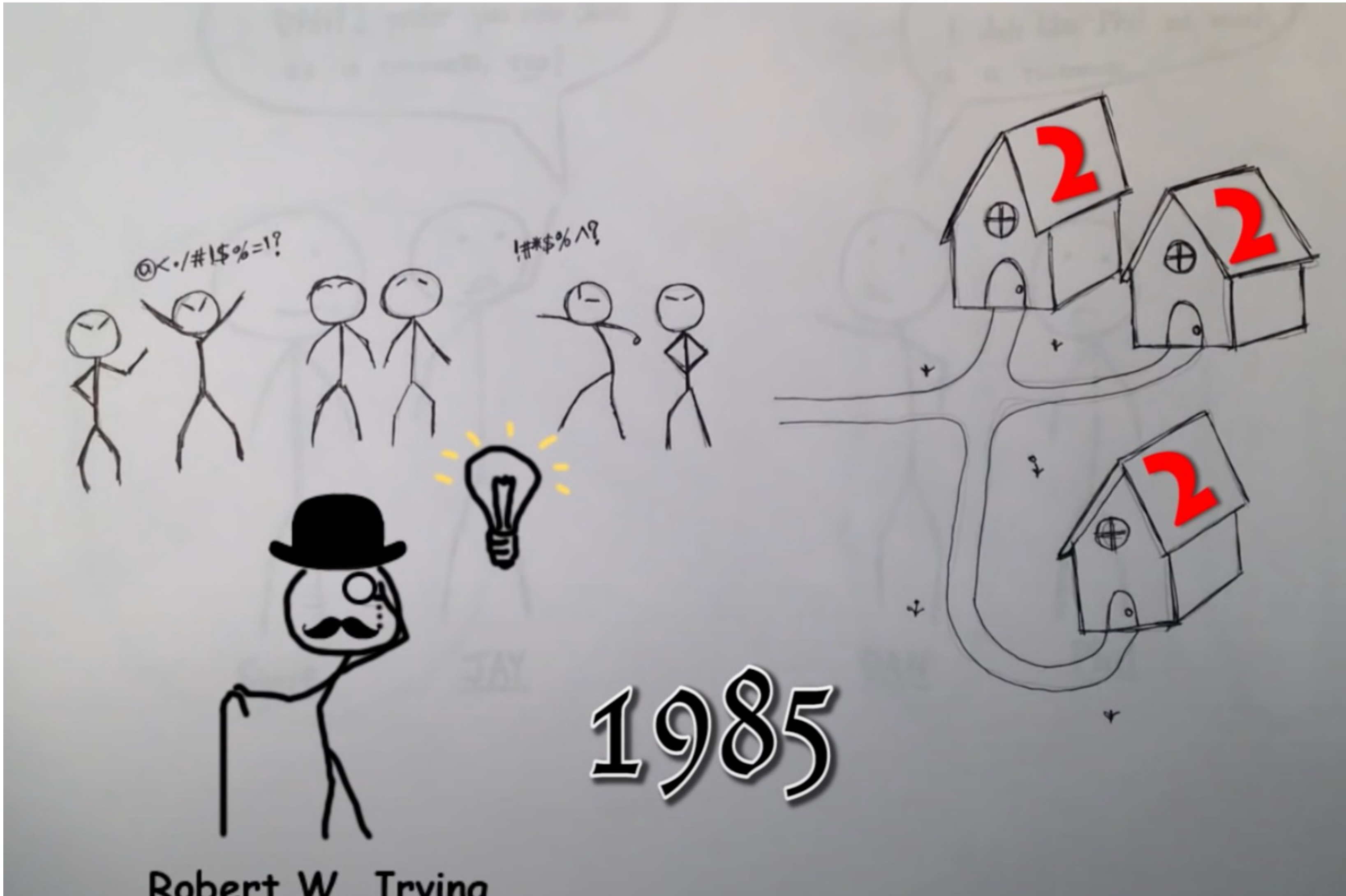
Man-optimality. In version of GS where men propose, each man receives best valid partner.

w is a valid partner of m if there exist some stable matching where m and w are paired

Roommate Matching and Uniqueness

- https://www.youtube.com/watch?v=5QLxAp8mRKo
- Unique matching: https://bit.ly/3gUfwvZ

Stable Roommate Matching



Stage 1:

Everybody proposes to their favourite. Order does not matter.

Proposal recipients then pick their most best proposer and reject the rest.

Those who got rejected keep proposing until accepted.

If someone gets rejected by everyone else then no stable matching exists.

Stage 2:

Everyone rejects those potential partners less desirable than their current accepted one.

Stage 3:

Find a participant who has more than one choice.

Write their second preference X, then the last preference, Y, of X.

Repeat previous step until the starting player appears again.

Every second preference and last preference then reject symmetrically.

Do this until everybody has only one option.

Conditions with no stable matching

A	B	C	M
B	C	A	M
C	A	B	M
M	A	B	C

2.2 Asymptotic Order of Growth

Asymptotic Order of Growth

Upper bounds. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.

Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.

Tight bounds. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Ex: $T(n) = 32n^2 + 17n + 32$.

- $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
- $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

Notation

Slight abuse of notation. $T(n) = O(f(n))$.

- Not transitive:
  - $f(n) = 5n^3$; $g(n) = 3n^2$
  - $f(n) = O(n^3) = g(n)$
  - but $f(n) \neq g(n)$.
- Better notation: $T(n) \in O(f(n))$.

Meaningless statement. Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.

- Statement doesn't "type-check."
- Use Ω for lower bounds.

Properties

Transitivity.

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$.

Additivity.

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$.

Asymptotic Bounds for Some Common Functions

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

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

Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$.

can avoid specifying the base

Logarithms. For every $x > 0$, $\log n = O(n^x)$.

log grows slower than every polynomial

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

every exponential grows faster than every polynomial

Special functions

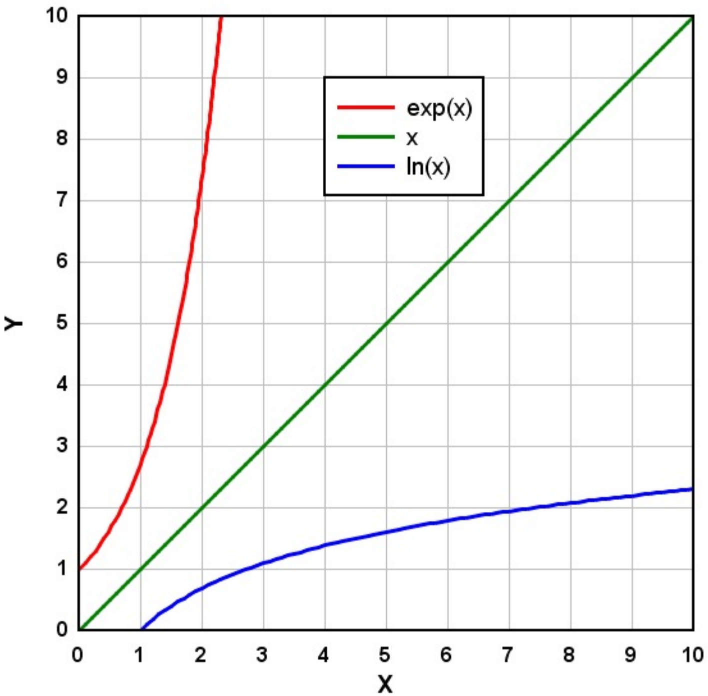
Polynomial $O(n^a)$, a independent of n

- $O(n)$ linear
- $O(n^2)$ quadratic
- $O(n^3)$ cubic
- $O(\log n)$ logarithms
  - $O(\log n) = O(n^{\varepsilon})$

    logarithms grow more slowly than polynomial

Nonpolynomial

- $O(n!), O(3^n)$



Practice

$2n$	
$2n + 3$	
$2n + 10{,}000{,}000{,}000$	
$2n - 10{,}000{,}000{,}000$	
$3n^2 + 2n + 3$	
$3n^2 + 10000000000n + 3$	
$2\log(n)$	
$2\log_{100}(n)$	
$2n + \log(n)$	
$3^n + n^{100}$	

$$\log_a b = \frac{\log_c a}{\log_c b}$$

Practice

$2n$	$O(n)$
$2n + 3$	$O(n)$
$2n + 10{,}000{,}000{,}000$	$O(n)$
$2n - 10{,}000{,}000{,}000$	$O(n)$
$3n^2 + 2n + 3$	$O(n^2)$
$3n^2 + 10000000000n + 3$	$O(n^2)$
$2\log(n)$	$O(\log(n))$
$2\log_{100}(n)$	$O(\log(n))$
$2n + \log(n)$	$O(n)$
$3^n + n^{100}$	$O(3^n)$

$$\log_a b = \frac{\log_c a}{\log_c b}$$

Practice

2. Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

   (a) n^2

   (b) n^3

   (c) $100n^2$

   (d) $n \log n$

   (e) 2^n

   (f) 2^{2^n}

Practice

One hour: $3.6 * 10^{13}$ operations

n^2	6,000,000
n^3	33019
$100n^2$	600,000
$n \log n$	$1.29 * 10^{12}$ (different results for different base)
2^n	45
n^{2^n}	5