Objectives

- Proving correctness of Stable Matching algorithm
- Analyzing algorithms
- Asymptotic running times

Wiki notes:

- Read after class; I am giving loose guidelines – the point is to review and synthesize
- Monday midnight deadline

If you're interested, join the
W&L Computer Science Facebook Group!

Review

- What is the stable matching problem?
  - What is given?
  - What is our goal?
- Provide a sketch of the algorithm
- What observations do you have about the algorithm and how it progresses?
  - What can we say about any woman's partner during the execution of the algorithm?
  - How does a woman's state change over the execution of the algorithm?
  - What can we say about a man's partner?

Propose-And-Reject Algorithm

[Gale-Shapley 1962]

```
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
```


Observations about the Algorithm

- What can we say about any woman's partner during the execution of the algorithm?
  - Observation 1. He gets "better" → she prefers him over her last partner
- How does a woman's state change over the execution of the algorithm?
  - Observation 2. Once a woman is matched, she never becomes unmatched; she only "trades up"
- What can we say about a man's partner?
  - Observation 3. She gets "worse"

Proving Correctness

- Need to show
  - Algorithm terminates
  - Result is a perfect matching
  - Result is a stable matching

1) Algorithm Termination

[Gale-Shapley 1962]

Does algorithm terminate?

```
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
```

Proof of Correctness: Termination

- Claim. Algorithm terminates after at most n^2 iterations of while loop.
  - Hint: How wouldn't the algorithm terminate?

Proof of Correctness: Termination

- 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.

Number of proposals is a good measure for termination
→ strictly increases; limited

Proof of Correctness: Termination

- 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.

Note: not yet discussing the cost *in the body* of the while loop

2) Algorithm Analysis: Perfect Matching

Prove that final matching is a *perfect* matching

- **Perfect matching**: everyone is matched monogamously
- Hint: in algorithm, we know if *m* is free at some point in the execution of the algorithm, then there is a woman to whom he has not yet proposed.

Proof of Correctness: Perfection

- Claim. All men and women get matched.
- Pf. (by contradiction)
  - Where should we start?

Suppose that some man *m* is not matched upon termination of algorithm

Proof of Correctness: Perfection

- Claim. All men and women get matched.
- Pf. (by contradiction)
  - Suppose that *m* is not matched upon termination of algorithm
  - Then some woman, say *w*, is not matched upon termination.
  - By Observation 2, *w* was never proposed to.
  - But, last man proposed to everyone, since he ends up unmatched
    - (by the while loop's condition)
  - Contradiction ▪

Proof of Correctness: Stability

- Claim. No unstable pairs.

S*
Amy-Yancey
Bertha-Zeus
...

What does it mean for a given matching S* to be unstable?

How do you think we should approach this proof?

Jan 11, 2019 Sprenkle - CSCI211 13

Proof of Correctness: Stability

S*
Amy-Yancey
Bertha-Zeus
...

- Claim. No unstable pairs.
- Pf. (by contradiction)
  - Suppose m-w is an unstable pair: m, w prefers each other to partner in Gale-Shapley matching S*.

What are the possibilities that lead to this?

Jan 11, 2019 Sprenkle - CSCI211 14

Proof of Correctness: Stability

S*
Amy-Yancey
Bertha-Zeus
. . .

- Claim. No unstable pairs.
- Pf. (by contradiction)
  - Suppose m-w is an unstable pair: m, w prefers each other to partner in Gale-Shapley matching S*.
  - **Case 1: m never proposed to w** ← men propose in decreasing order of preference
    - ⇒ m prefers his GS partner to w.
    - ⇒ m-w is stable.
  - **Case 2: m proposed to w**
    - ⇒ w rejected m (right away or later) ← women only trade up
    - ⇒ w prefers her GS partner to m.
    - ⇒ m-w is stable.
  - In either case m-w is stable, a contradiction. ▪

Summary So Far…

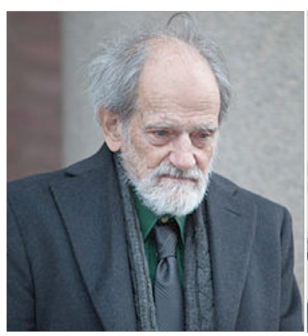
- **Stable matching problem.** Given *n* men and *n* women and their preferences, find a stable matching if one exists.
- **Gale-Shapley algorithm.** Guarantees to find a stable matching for *any* input

Remaining Questions:
- If there are multiple stable matchings, which one does GS find? *(see book)*
- How to implement GS algorithm efficiently? (next week)
  - What is our goal running time?

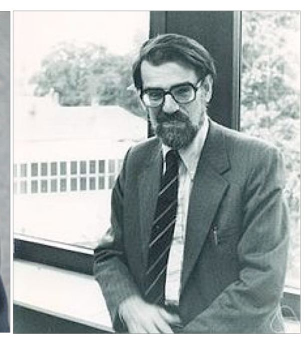
Review: Our Process

1. Understand/identify problem
   - Simplify as appropriate
2. Design a solution
3. Analyze
   - Correctness, efficiency
   - May need to go back to step 2 and try again
4. Implement
   - Within bounds shown in analysis

Lloyd Shapley



2012



1980

- 2012 Nobel Memorial Prize in Economic Sciences "for the theory of stable allocations and the practice of market design."

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.
  - Claim: can implement algorithm *efficiently*

Jan 11, 2019 Sprenkle - CSCI211 19

TODAY'S GOAL:
DEFINE ALGORITHM EFFICIENCY

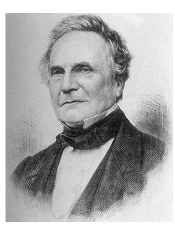
Jan 11, 2019 Sprenkle - CSCI211 20

Our Process

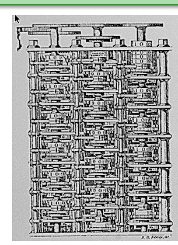
1. Understand/identify problem
   - Simplify as appropriate
2. Design a solution
3. Analyze
   - Correctness, efficiency
   - May need to go back to step 2 and try again
4. Implement (On Wednesday)
   - Within bounds shown in analysis

Computational Tractability

> As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?
>
> -- *Charles Babbage*



Charles Babbage (1864)



Analytic Engine (schematic)

http://plan28.org/

Brute Force

- For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution
  "Exponential"
  - Typically takes 2^N time or worse for inputs of size N
  - Unacceptable in practice

Example: How many possible solutions are there in the stable matching problem?
In other words, how many possible *perfect* matchings are there?
For each perfect match, we'll check if it's stable.

Brute Force

- For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution
  "Exponential"
  - Typically takes 2^N time or worse for inputs of size N
  - Unacceptable in practice
- Example: Stable matching: n! with n men and n women
  - If n increases by 1, what happens to the running time?

How Do We Measure Runtime?

Worst-Case Running Time

- Obtain bound on *largest possible* running time of algorithm on input of a given size N
  - Generally captures efficiency in practice
  - Draconian view but hard to find effective alternative

What are alternatives to worst-case analysis?

Average Case Running Time

- Obtain bound on running time of algorithm on *random* input as a function of input size N
  - Hard (or impossible) to accurately model real instances by random distributions
  - Algorithm tuned for a certain distribution may perform poorly on other inputs

Towards a Definition of Efficient…

- Desirable scaling property: When input size doubles, algorithm should only slow down by some constant factor C
  - Doesn't grow multiplicatively

Polynomial-Time

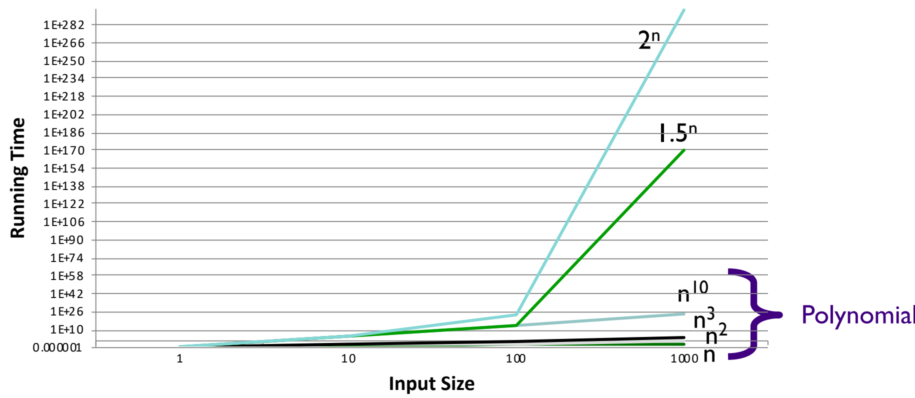
Defn. There exists constants c > 0 and d > 0 such that on every input of size N, its running time is bounded by $c N^d$ steps.

- ✓ Desirable scaling property: When input size doubles, algorithm should only slow down by some constant factor C
  - What happens if we double N?
- Defn. An algorithm is *polynomial time* (or *polytime)* if the above scaling property holds.

Algorithm Efficiency

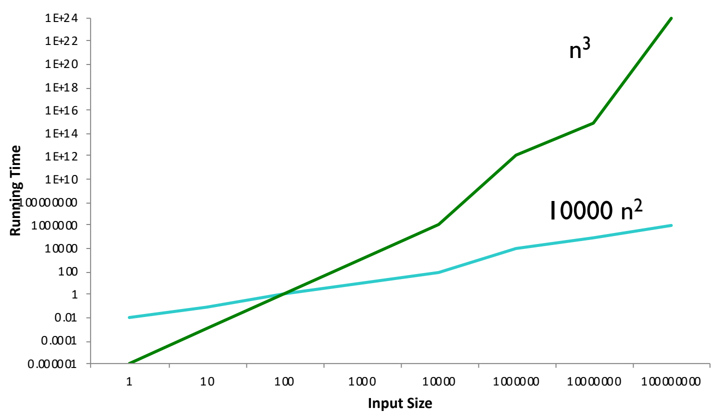
- **Defn.** An algorithm is ***efficient*** if its running time is *polynomial*
- Justification: It really works in practice!
  - In practice, poly-time algorithms that people develop almost always have low constants and low exponents
  - Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem
- Exceptions
  - Some poly-time algorithms do have high constants and/or exponents ($6.02 \times 10^{23} \times N^{20}$) and are useless in practice
  - Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare

Visualizing Running Times



- Huge difference from polynomial to not polynomial
- Differences in runtime matter more as input size increases

Comparing 10000 n^2 and n^3



As input size increases, n^3 dominates large constant * n^2

➔Care about running time as input size approaches infinity

➔Only care about highest-order term

Asymptotic Order of Growth: Upper Bounds

- **T(n)** is the worst case running time of an algorithm
- We say that 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

$T(n) \leq c \cdot f(n)$

"order f(n)"

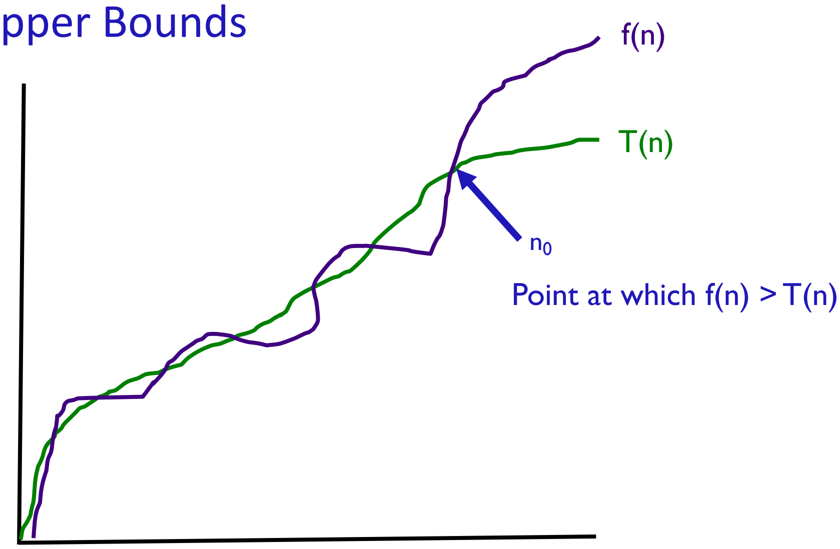
c cannot depend on n

sufficiently large n

T(n) is bounded above by a constant multiple of f(n)

T is ***asymptotically upperbounded*** by f

Asymptotic Order of Growth: Upper Bounds



Upper Bounds Example

- Find an upperbound for

  $T(n) = pn^2 + qn + r$

  - p, q, r are positive constants

Motivation: Why can we simplify to just the largest term?

Upper Bounds Example

- Find an upperbound for

  $T(n) = pn^2 + qn + r$

  - p, q, r are positive constants

Idea: Let's inflate the terms in the equation so that all terms are n^2

Upper Bounds Example

- $T(n) = pn^2 + qn + r$
  - p, q, r are positive constants
- For all $n \geq 1$,

$$\begin{aligned} T(n) &= pn^2 + qn + r \\ &\leq pn^2 + qn^2 + rn^2 \\ &= (p+q+r)\, n^2 \\ &= c\, n^2 \end{aligned}$$

➔ $T(n) \leq cn^2$, where $c = p+q+r$

➔ $T(n) = O(n^2)$

- Also correct to say that $T(n) = O(n^3)$

Notation

- $T(n) = O(f(n))$ is a slight abuse of notation
  - Asymmetric:
    - $f(n) = 5n^3$; $g(n) = 3n^2$
    - $f(n) = O(n^3) = g(n)$
    - 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
  - Use Ω for lower bounds

Asymptotic Order of Growth: Lower Bounds

- Complementary to upper bound

ε cannot depend on n

- T(n) is **Ω(f(n))** if there exist constants $\varepsilon > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have

sufficiently large n

$T(n) \geq \varepsilon \cdot f(n)$

T(n) is bounded below by a constant multiple of f(n)

T is ***asymptotically lowerbounded*** by f

Example: Lower Bound

- $T(n) = pn^2 + qn + r$
  - p, q, r are positive constants
- Idea: *Deflate* terms rather than inflate

Example: Lower Bound

- $T(n) = pn^2 + qn + r$
  - p, q, r are positive constants
- Idea: *Deflate* terms rather than inflate
- For all $n \geq 0$,

$$T(n) = pn^2 + qn + r \geq pn^2$$
$$\rightarrow T(n) \geq \varepsilon n^2, \text{ where } \varepsilon = p > 0$$
$$\rightarrow T(n) \in \Omega(n^2)$$

- Also correct to say that $T(n) \in \Omega(n)$

Tight bounds

T(n) is **Θ(f(n))** if T(n) is both O(f(n)) and Ω(f(n))

- The "right" bound

A Fashion Analogy

- O == Hammer pants
  - Loose and baggy with plenty of room for the pants to shrink or the body to grow
- Ω == The pants you plan to fit in this summer after working off the snacks from Christmas
- Θ == Katy Perry's skin tight jeans in a teenage dream
  - Can't make them any smaller, and no extra room to even fit a cell phone in the pocket

Courtesy Andy Danner, Swarthmore

Looking Ahead

- Continue reading Chapter 2
  - Covering later sections on Wednesday
- Journal for 2 pages of Preface, 1.1, Chapter 2, 2.1, 2.2 due Monday at midnight
  - No journal for Chapter 1.2
  - Wrapping up 2.2 in class on Monday; first part is helpful for problem set
- Problem Set 1 due next Friday before class
  - Proof, stable matching, asymptotic bound
  - Start early!
    - Read problems and let your brain start thinking about them
    - Solved exercises in book
  - Honor Code