(Record)   *Induction Problems on Canvas*

CS3000: Algorithms & Data
Drew van der Poel

Lecture 3
- Finish Gale-Shapley
- Asymptotic Analysis

May 12, 2021

Warm-Up

- **Claim:** Every natural number ≥ 2 has at least one prime factor.
  - *Prime* – positive integer greater than 1 that cannot be formed by multiplying two smaller natural numbers
  - *Factor* – a number that divides another number evenly

- **Proof by Induction:** $(2 \cdot 1 = 2)$

Base: $H(2)$ – 2 is a PF of 2

Ind.: $\cancel{H(k-1) \rightarrow H(k)}$ not going to work

Warm-Up

- **Strong Induction:** In inductive step, to prove *H(i)*, we may assume *H(1), H(2), …, H(i-1)*

$H(1) \wedge H(2) \wedge \ldots \wedge H(i-1) \rightarrow H(i)$

- **Note:** in "weak" induction, we only assume *H(i-1)* because it is all we need to prove *H(i)*

- Problems: counting students, stable matching
- Alg. techniques:
- Analysis:
- Proof techniques: **(strong)** induction

Warm-Up

- **Claim:** Every natural number ≥ 2 has at least one prime factor.
- **Proof by Induction:**

Base. $H(2)$ - same as before

"and" "and"

$H(2) \land \dots \land H(k-1) \to H(k)$

Ind. Case 1: k is prime - k is a PF of k

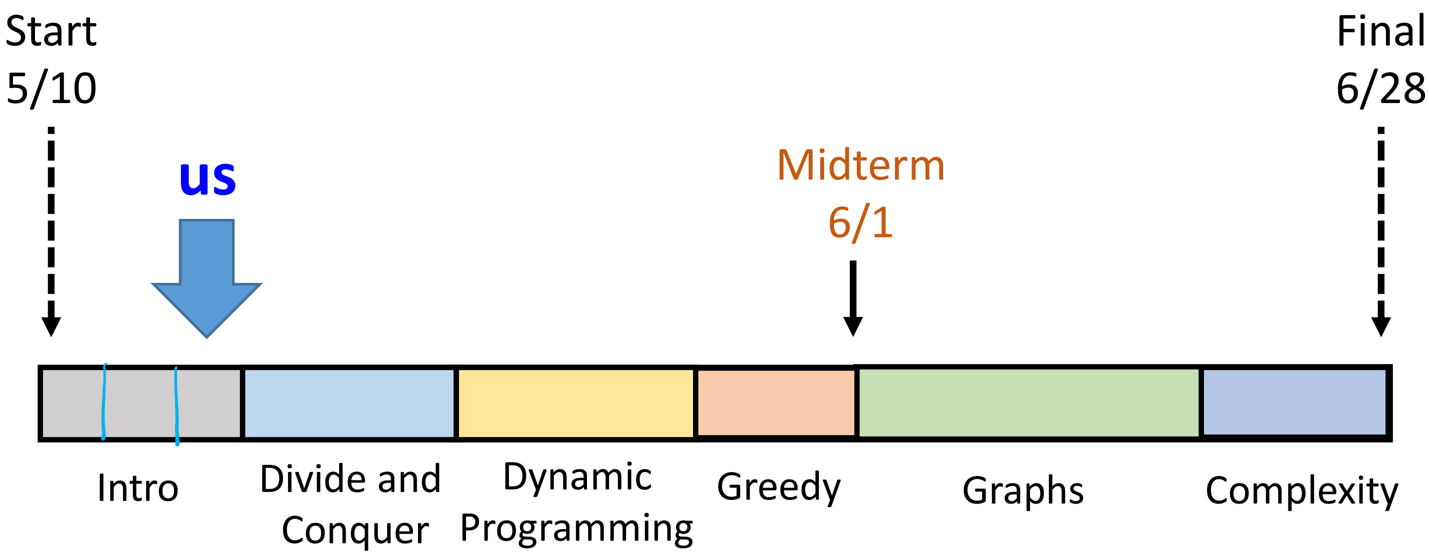
Case 2: k is not prime $\to$ $k = a \cdot b$ where $a, b \in \mathbb{N}$ and $1 < a, b < k$

$a < k \to$ IH holds for a ($H(a)$ is True)

$a = x \cdot y$ where x is prime

$k = a \cdot b = x \cdot y \cdot b$ $\therefore$ x is a PF of k $\square$

Outline



Last class: stable matching

Next class: divide and conquer: Merge sort

Gale-Shapley Algorithm

“job offer”

```
• Let M be empty
• While (some hospital h is unmatched):
    • If (h has offered a job to everyone): break
    • Else: let d be the highest-ranked doctor to
      which h has not yet offered a job
    • h makes an offer to d:
        • If (d is unmatched):
            • d accepts, add (d,h) to M
        • ElseIf (d is matched to h′ & d: h′ > h):
            • d rejects, do nothing
        • ElseIf (d is matched to h′ & d: h > h′):
            • d accepts, remove (d,h′) from M and
              add (d,h) to M
• Output M
```

Observations (for proofs later on)

1. Hospitals make offers in descending order

   If $(d, h) \in M$, h will never be w/ a doctor it prefers to d

2. Doctors that get a job never become unemployed

   - If d is matched at some point, it is matched forever

3. Doctors accept offers in ascending order

   If $(d, h) \in M$, d only leaves for a hospital it prefers to h

   (doctors get happier)

Gale-Shapley Algorithm

- Questions about the Gale-Shapley Algorithm:
  - Will this algorithm terminate? And how long will it take?
  - Does it output a perfect matching?
  - Does it output a stable matching?
  - How do we implement this algorithm efficiently?

GS Algorithm: Termination

n = # of doctors/hospitals

- **Claim:** The GS algorithm terminates after at most n^2 iterations of the main loop (n^2 job offers).

- each hospital offers ≤ 1 time to each doctor

# of offers $\leq$ n hospitals $\times$ n doctors

$= n^2$

Gale-Shapley Algorithm

- Questions about the Gale-Shapley Algorithm:
  - Will this algorithm terminate? Yes!
  - Does it output a perfect matching?
  - Does it output a stable matching?
  - How do we implement this algorithm efficiently?

GS Algorithm: Perfect Matching

- **Claim:** The GS algorithm returns a perfect matching (all doctors/hospitals are matched)

Proof by Contradiction

C = Claim WTS is True

- **Important:** No claim/proposition can be both true and false
- Assume the claim *C* that you want to prove true is *false* (~*C* is true) Assume C is False
- Then show the claim being false implies **contradictory** assertions (that both an assertion *Q* and not-*Q* are true)
- Since *Q* and not-*Q* cannot both be true, *C* must be true

"one of a mathematician's finest weapons" – G. H. Hardy

- Problems: counting students, stable matching

- Alg. techniques:

- Analysis:

- Proof techniques: (strong) induction, **contradiction**

GS Algorithm: Perfect Matching

(c)

- **Claim:** The GS algorithm returns a perfect matching (all doctors/hospitals are matched)

Assume Claim is False → GS does not return a PM

→ $\underbrace{\text{some doctor is unmatched}}_{d}$ and $\underbrace{\text{some hospital is unmatched}}_{h}$

During the alg., h offers job to d

① d accepts → by Obs. 2, doctors stay "employed" → d is matched →←

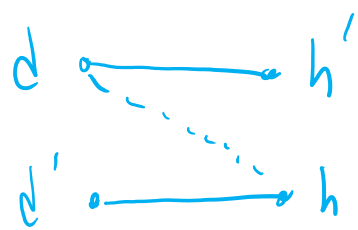
② d rejects → d was matched w/ h' → by Obs. 2, d is matched →←

Gale-Shapley Algorithm

- Questions about the Gale-Shapley Algorithm:
  - Will this algorithm terminate? Yes!
  - Does it output a perfect matching? Yes!
  - Does it output a stable matching?
  - How do we implement this algorithm efficiently?

GS Algorithm: Stable Matching

- **Stability:** GS algorithm outputs a stable matching
- Proof by contradiction:
  - Suppose there is an instability *(d, h'), (d', h)*
  - That is, given a matching which includes *(d, h'), (d', h), d* prefers *h* to *h'* and *h* prefers *d* to *d'*



h: $d > d'$

GS Algorithm: Stable Matching

- **Stability:** GS algorithm outputs a stable matching
- Proof by contradiction:
  - Suppose there is an instability *(d, h'), (d', h)*
    - *h: d > d'*
    - d: h > h'
- We know *h* made an offer to *d* before *d'* (by obs. 1)
  - Case 1 – d accepts
  - Case 2 – d rejects

GS Algorithm: Stable Matching

- **Stability:** GS algorithm outputs a stable matching
- Proof by contradiction:
  - Suppose there is an instability *(d, h'), (d', h)*
    - *h: d > d'*
    - d: h > h'

  We know *h* made an offer to *d* before *d'*
  - Case 1 – *d* rejected the offer

d was matched w/ h^* (d: $h^* > h$)

↳ by obs. 3, doctors get happier

$d: h' \geq h^* > h$ →←

GS Algorithm: Stable Matching

- **Stability:** GS algorithm outputs a stable matching
- Proof by contradiction:
  - Suppose there is an instability *(d, h'), (d', h)*
    - *h: d > d'*
    - d: h > h'

  We know *h* made an offer to *d* before *d'*
  - Case 2 – *d* accepted the offer

$(d, h) \in M$ at some point

$\hookrightarrow$ by obs. 3, doctors get happier

$d: h' > h \implies \Leftarrow$

$\square$

Gale-Shapley Algorithm

- Questions about the Gale-Shapley Algorithm:
  - Will this algorithm terminate? Yes!
  - Does it output a perfect matching? Yes!
  - Does it output a stable matching? Yes!
  - How do we implement this algorithm efficiently?

GS Algorithm: Running Time

- **Running Time:**
  - A straightforward implementation requires $\approx n^3$ operations in the worst case, $\approx n^2$ space
  - ($\approx$ -> dropping constants & lower-order terms)

GS Algorithm: Running Time

```
• Let M be empty
• While (some hospital h is unmatched):
  • If (h has offered a job to everyone): break
  • Else: let d be the highest-ranked doctor to
    which h has not yet offered a job
  • h makes an offer to d:
    • If (d is unmatched):
      • d accepts, add (d,h) to M
    • ElseIf (d is matched to h′ & d: h′ > h):
      • d rejects, do nothing
    • ElseIf (d is matched to h′ & d: h > h′):
      • d accepts, remove (d,h′) from M and
        add (d,h) to M
• Output M
```

"job offer"

GS Algorithm: Running Time

```
• Let M be empty
• While (some hospital h is unmatched):
  • If (h has offered a job to everyone): break
  • Else: let d be the highest-ranked doctor to
    which h has not yet offered a job
  • h makes an offer to d:
    • If (d is unmatched):
      • d accepts, add (d,h) to M
    • ElseIf (d is matched to h′ & d: h′ > h):
      • d rejects, do nothing
    • ElseIf (d is matched to h′ & d: h > h′):
      • d accepts, remove (d,h′) from M and
        add (d,h) to M
• Output M
```

"job offer"

- Loop runs ≤ n^2 times; ≤ *n* operations to <u>find *h, h'* in *d*'s preferences</u>
- n^2 offers * *n* operations = n^3 total operations

GS Algorithm: Running Time

- **Running Time:**
  - A careful implementation requires just $\approx n^2$ operations in the worst case and $\approx n^2$ space

GS Algorithm: Running Time

- **Running Time:**
  - A careful implementation requires just $\approx n^2$ operations in the worst case and $\approx n^2$ space
  - Create an array of doctor x hospital in n^2 steps

	1st	2nd	3rd	4th	5th
Alice	CH	MGH	BW	MTA	BID
Bob	BID	BW	MTA	MGH	CH
Clara	BW	BID	MTA	CH	MGH
Dorit	MGH	CH	MTA	BID	BW
Ernie	MTA	BW	CH	BID	MGH

	MGH	BW	BID	MTA	CH
Alice					
Bob					
Clara					
Dorit					
Ernie					

GS Algorithm: Running Time

- **Running Time:**
  - A careful implementation requires just $\approx n^2$ operations in the worst case and $\approx n^2$ space
  - Create an array of doctor x hospital in n^2 steps

	1st	2nd	3rd	4th	5th
Alice	CH	MGH	BW	MTA	BID
Bob	BID	BW	MTA	MGH	CH
Clara	BW	BID	MTA	CH	MGH
Dorit	MGH	CH	MTA	BID	BW
Ernie	MTA	BW	CH	BID	MGH

→

	MGH	BW	BID	MTA	CH
Alice	2^{nd}	3^{rd}	5^{th}	4^{th}	1^{st}
Bob	4^{th}	2^{nd}	1^{st}	3^{rd}	5^{th}
Clara	5^{th}	1^{st}	2^{nd}	3^{rd}	4^{th}
Dorit	1^{st}	5^{th}	4^{th}	3^{rd}	2^{nd}
Ernie	5^{th}	2^{nd}	4^{th}	1^{st}	3^{rd}

GS Algorithm: Running Time

- **Running Time:**
  - A careful implementation requires just $\approx n^2$ operations in the worst case and $\approx n^2$ space
  - n^2 operations to convert doctor x rank -> doctor x hospital
  - Loop runs $\leq n^2$ times; *2* operations to find *h* & *h'* in *d*'s preferences
  - $\approx n^2$ total operations

Real World Impact

TABLE I

STABLE AND UNSTABLE (CENTRALIZED) MECHANISMS

Market	Stable	Still in use (halted unraveling)
American medical markets		
NRMP	yes	yes (new design in '98)
Medical Specialties	yes	yes (about 30 markets)
British Regional Medical Markets		
Edinburgh ('69)	yes	yes
Cardiff	yes	yes
Birmingham	no	no
Edinburgh ('67)	no	no
Newcastle	no	no
Sheffield	no	no
Cambridge	no	yes
London Hospital	no	yes
Other healthcare markets		
Dental Residencies	yes	yes
Osteopaths (<'94)	no	no
Osteopaths (≥'94)	yes	yes
Pharmacists	yes	yes
Other markets and matching processes		
Canadian Lawyers	yes	yes (except in British Columbia since 1996)
Sororities	yes (at equilibrium)	yes

Table 1. Reproduced from Roth (2002, Table 1).

Real World Impact

- **Doctors ↔ Hospitals**
  - Have to deal with two-body problems
  - Have to make sure doctors do not game the system
- **Kidneys ↔ Patients**
  - Not all matches are feasible (blood types)
  - Certain pairs must be matched
- **Students ↔ Public Schools**
  - Siblings, walking zones, diversity

2012 Nobel Prize to Lloyd Shapley and Al Roth!

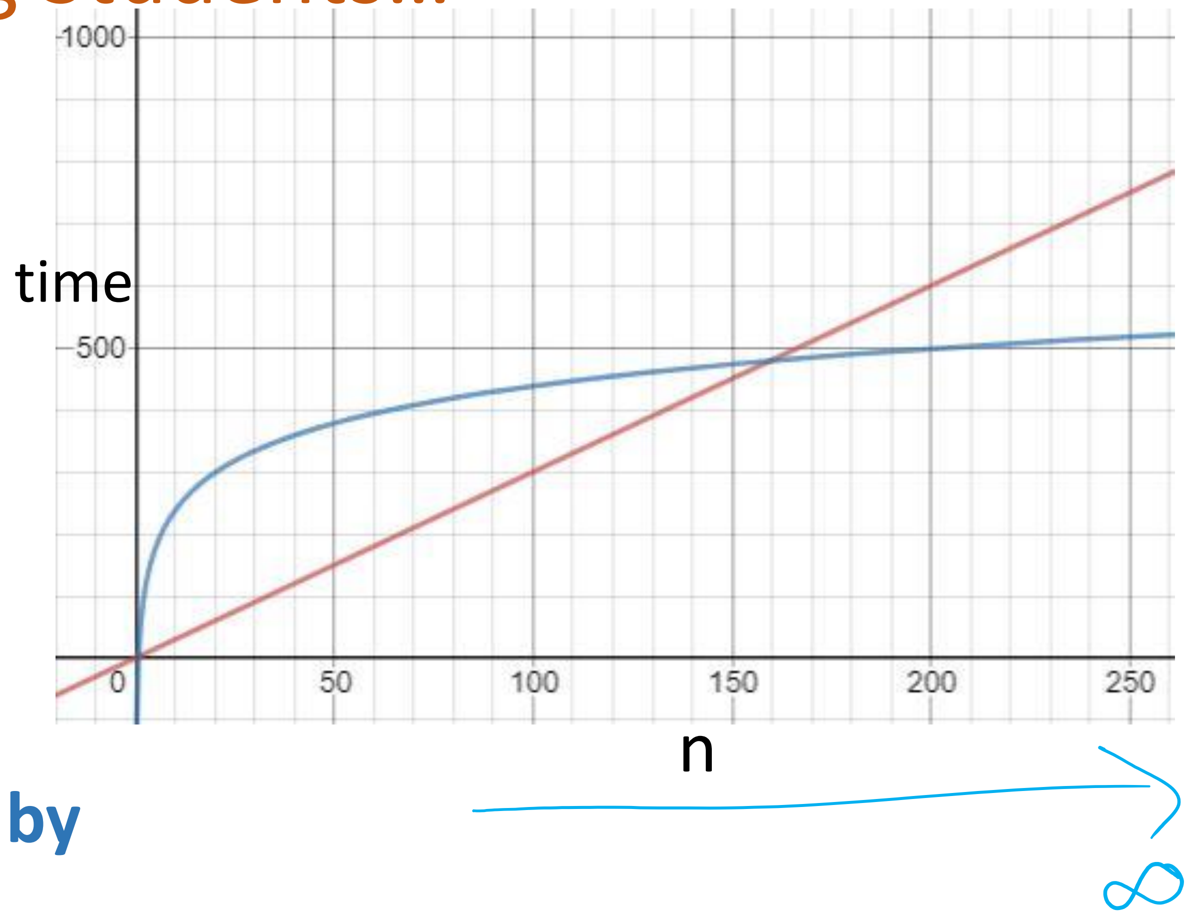
Asymptotic Analysis

of operations → run times of algs. → functions

- Tool used to compare algorithms (functions)
- Describes performance based on input size (*n*)
- Measures speed/size as input grows (gets really big)
  - Focuses on dominant (largest) term of function

From Counting Students…

- **Simple counting:**
  $3n$ time
- **Recursive counting:**
  $60 \log_2 n + 40$ time



- **Compare algorithms by asymptotics!**
  - Log-time beats linear-time as $n \to \infty$

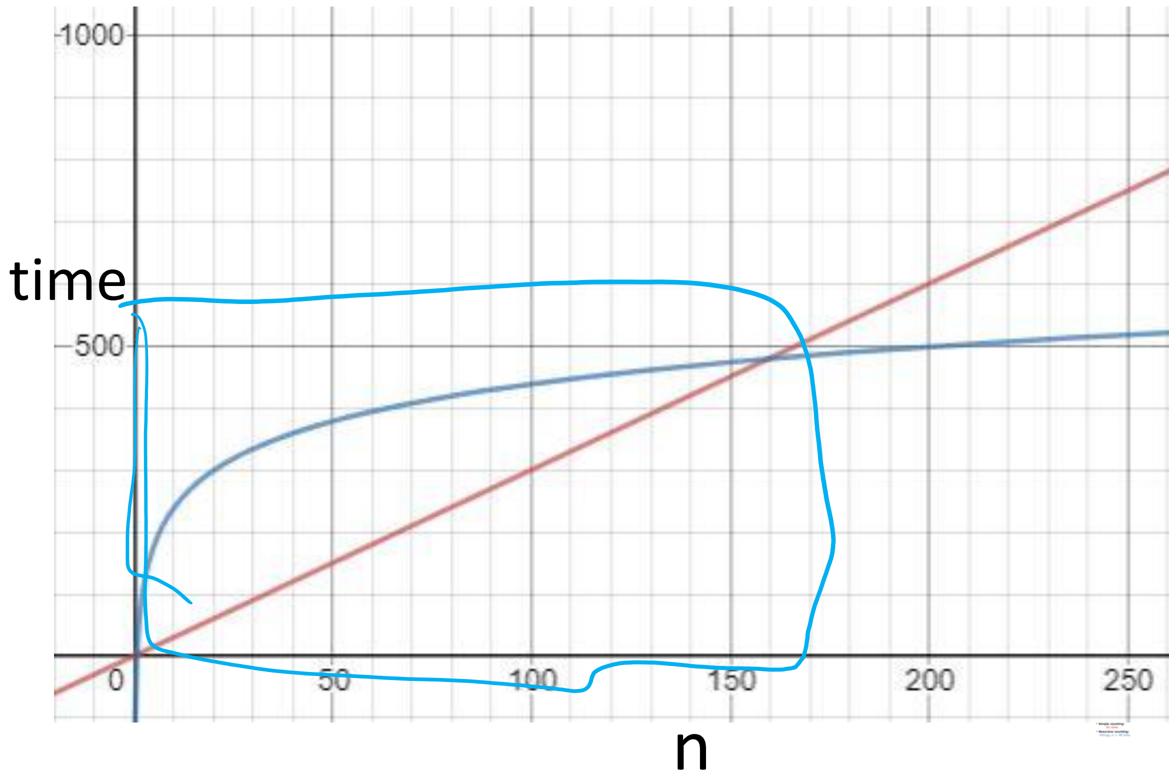
How long will this take?

- Predicting the wall-clock time of an algorithm is nigh impossible.
  - What machine will actually run the algorithm?
  - Impossible to exactly time operations

Asymptotic Order Of Growth

- Typically, we want to *compare* algorithms, so we can select the right one for the job
- Typically, we don't care about small inputs, we care about how the algorithm will *scale*

- **Simple counting:** $3n$ time
- **Recursive counting:** $60 \log_2 n + 40$ time



Asymptotic Order Of Growth

- **Asymptotic Analysis:** How does the running time grow as the size of the input grows?

- *Exact running time (# of operations)* *vs.* *order of growth*
  - *f(n):* messy function
  - *g(n):* nice function, summarizes performance

Asymptotic Order Of Growth

(worst-case) messy neat

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \le c \cdot g(n)$ for every $n \ge n_0$.
  - Asymptotic version of $f(n) \le g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

Asymptotic Order Of Growth

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \le c \cdot g(n)$ for every $n \ge n_0$.

- Ex. $\underbrace{3n^2 + 1n}_{f(n)} = O\big(\underbrace{n^2}_{g(n)}\big)$

$C = 4$

$n_0 = 1$

$3n^2 + n \le 4 \cdot n^2 \quad \forall n \ge 1$

$\frac{n}{n} \le \frac{n^2}{n} \quad \forall n \ge 1$

$1 \le n \quad \forall n \ge 1$

Ask the Audience

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

- **"Big-Oh" Notation:** $f(n) = O\big(g(n)\big)$ if there exists $c \in (0, \infty)$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.

- Which of these statements are true?
  - $n^3 = O(n^2)$ NO $\lim_{n \to \infty} \left(n^3 / n^2 = n \right) = \infty$
  - $10n^4 = O(n^5)$ → YES $c = 10$, $n_0 = 1$
    $10n^4 \leq 10n^5 \ \forall n \geq 1$
    $1 \leq n \ \forall n \geq 1$ ✓
  - $\log_2 n = O(\log_{16} n)$

* all log funcs. are asymptotically equiv.! *

$\log_2 n = \frac{\log_{16} n}{\boxed{\log_{16} 2}} = \frac{\log_{16} n}{1/4} = 4 \log_{16} n$

YES $c = 4$, $n_0 = 16$ $\underbrace{\log_2 n = 4 \log_{16} n}_{f(n)} \leq \underbrace{4 \log_{16} n}_{g(n)} \ \forall n \geq 16$

General Proof Example

- Prove that if $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then $f(n) + g(n) = O(h(n))$

$f(n) \leq c_1 h(n) \quad \forall n \geq n_0'$

$g(n) \leq c_2 h(n) \quad \forall n \geq n_0''$

A Word of Caution

- The notation $f(n) = O\big(g(n)\big)$ is weird—do not take it too literally

Asymptotic Analysis Rules

- **Constant factors can be ignored**
  - $\forall C > 0 \quad Cn = O(n)$
- **Lower order terms can be dropped**
  - E.g. $n^2 + n^{3/2} + n = O\left(n^2\right)$

- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \ \ n^a = O(b^n)$

Ask the Audience!

- Rank the following functions in increasing order of growth (i.e. f_1, f_2, f_3, f_4 so that $f_i = O(f_{i+1})$)
  - $n \log_2 n$
  - n^2
  - $100n$
  - $3^{\log_2 n}$

Asymptotic Order Of Growth

- **"Big-Omega" Notation:** $f(n) = \Omega\big(g(n)\big)$ if there exists $c \in (0,\infty)$ and $n_0 \in \mathbb{N}$ s.t. $f(n) \geq c \cdot g(n)$ for every $n \geq n_0$.
    - Asymptotic version of $f(n) \geq g(n)$
    - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$
- **"Big-Theta" Notation:** $f(n) = \Theta\big(g(n)\big)$ if there exists $c_1 \leq c_2 \in (0,\infty)$ and $n_0 \in \mathbb{N}$ such that $c_2 \cdot g(n) \geq f(n) \geq c_1 \cdot g(n)$ for every $n \geq n_0$.
    - Asymptotic version of $f(n) = g(n)$
    - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0,\infty)$
    - **Equivalent to:**

Asymptotic Running Times

- **It is *nice* to write running time as a Big-Theta**
  - **More precise than Oh or Omega**
  - Note: Sometimes it is simpler to just use Oh

More Examples

- $30 \log_2 n + 45 = \Theta(\log_2 n)$

- $4n \log_2 2n = \Theta(n \log_2 n)$

- $\sum_{i=1}^{n} i = \Theta\left(n^2\right)$

- Problems: counting students, stable matching
- Alg. techniques:
- Analysis: **asymptotic analysis**
- Proof techniques: (strong) induction, contradiction

Asymptotic Order Of Growth

- **"Little-Oh" Notation:** $f(n) = o\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ s.t. $f(n) < c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) < g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- **"Little-Omega" Notation:** $f(n) = \omega\big(g(n)\big)$ if for every $c > 0$ there exists $n_0 \in \mathbb{N}$ such that $f(n) > c \cdot g(n)$ for every $n \geq n_0$.
  - Asymptotic version of $f(n) > g(n)$
  - Roughly equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

Motivation: Why look at order of growth?

of operations

	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

Instance size

Running time when 1 op. takes ~1 microsecond

Observations:

1. Different polynomials make a BIG difference! (e.g. n^2 vs. n^3 when n ≥ 1000)
2. Things go bad quickly (look down any polynomial or exponential column)! We want **scalability**!
3. Large instances are when there is a difference (log still good!)