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:**

$H(n)$: n has a prime factor

Base: $H(2)$: 2 is a PF of 2 ✓

Ind: $H(K-1) \to H(K)$

Warm-Up

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

- **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, contradiction

Warm-Up

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

Base: Same as before, $H(2)$ is T

Ind: $H(2) \wedge H(3) \wedge \ldots \wedge H(K-1) \rightarrow H(K)$

case 1 – K is prime, K is a PF

Case 2 – not prime, $\exists a, b \in \mathbb{N}$ s.t. $1 < a, b < K$

$H(a)$ is T $\rightarrow y \cdot z = a$ (y is a PF)

$K = a \cdot b = y \cdot z \cdot b$

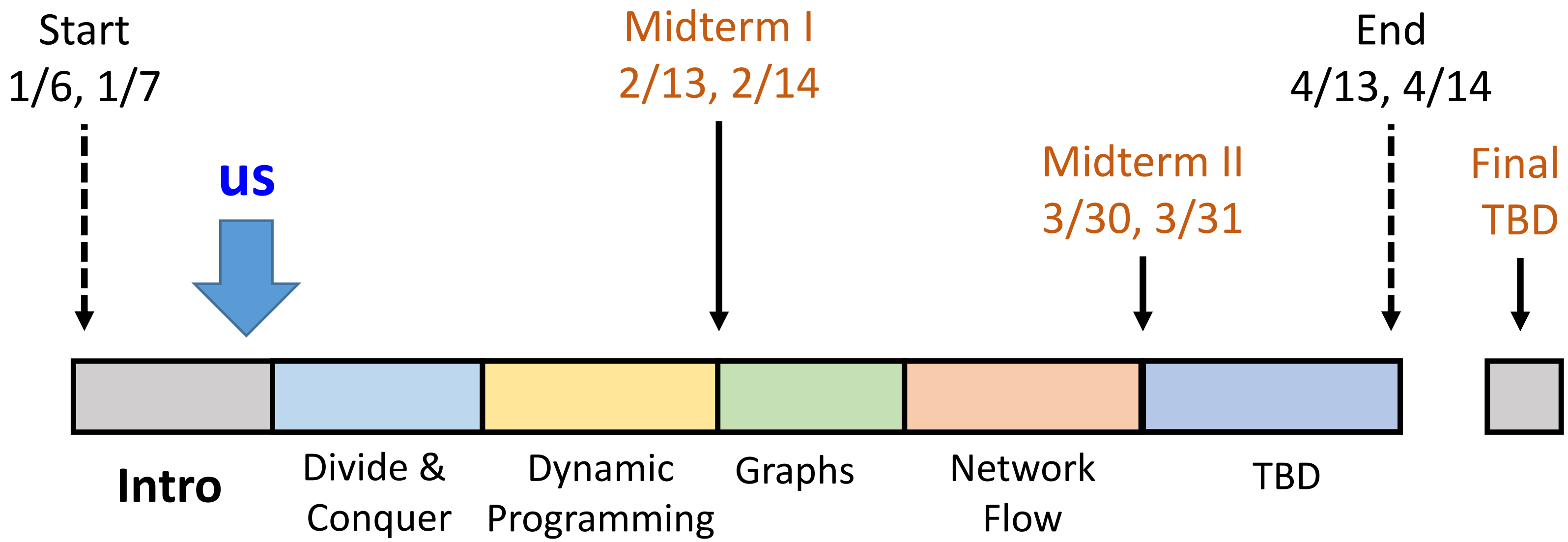
y is a PF of K ✓

CS3000: Algorithms & Data
Drew van der Poel

Lecture 3:
- Asymptotic Analysis

Jan 13/14, 2020

Outline



Last class: stable matching, proof by contradiction

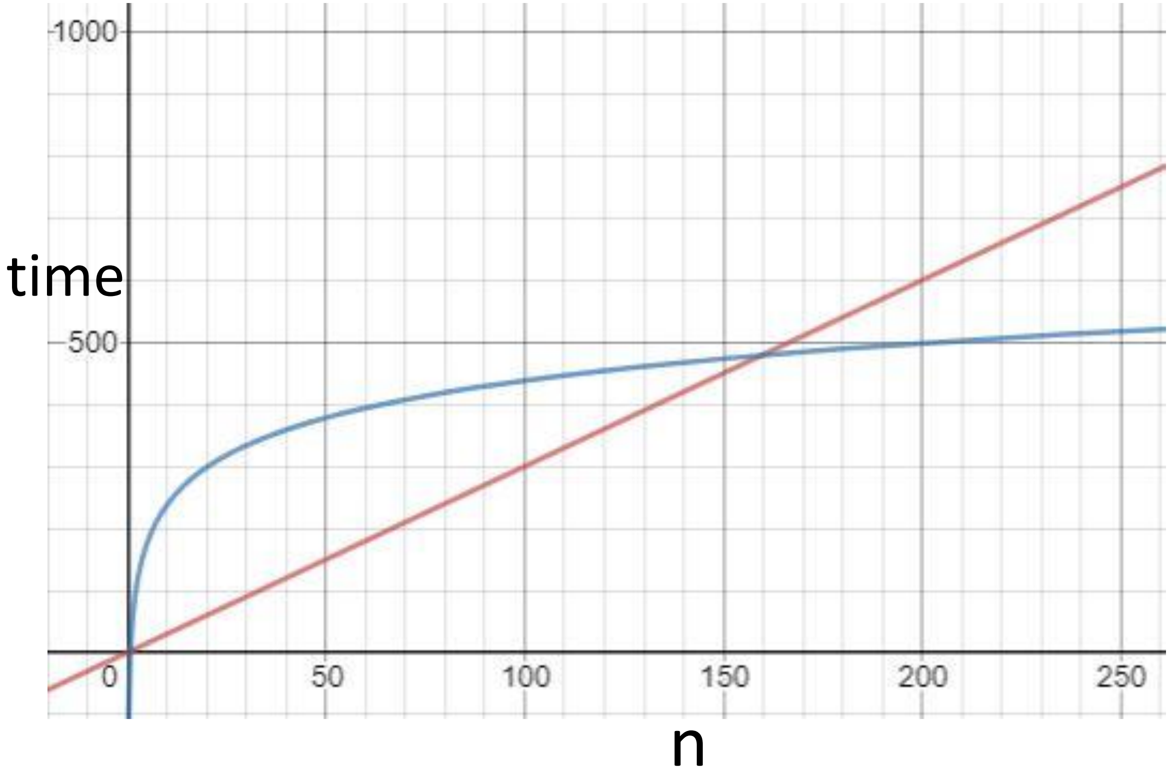
Next class: divide and conquer: Mergesort, divide and conquer: multiplying n-digit numbers (Karatsuba)

Asymptotic Analysis

- 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 Lecture 1...

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



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

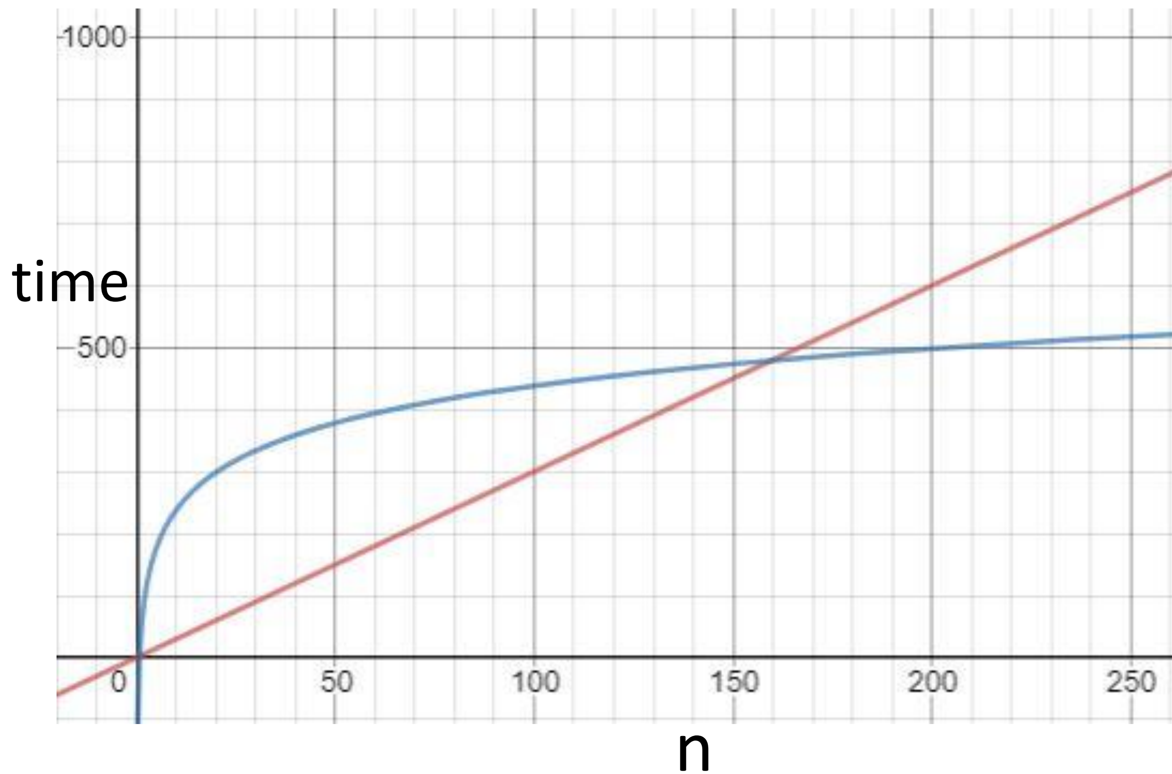
Asymptotic Order Of Growth

- 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

- Mostly we want to *compare* algorithms, so we can select the right one for the job
- Mostly 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, depends on machine
  - *g(n):* nice function, summarizes performance

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 $n \ge n_0$.
  - Asymptotic version of $f(n) \boxed{\le} g(n)$
  - 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 $f(n) \leq c \cdot g(n)$ for every $n \geq n_0$.

- Ex. $\overbrace{3n^2 + n}^{f(n)} = O\big(\overset{g(n)}{n^2}\big)$ $\quad c = 4,\ n_0 = 1$

$3n^2 + n \leq 4 \cdot n^2 \quad \forall\, n \geq 1$

$3n^2 + n \leq 3n^2 + n^2$

Ask the Audience

- **"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 $n \ge n_0$.

- Which of these statements are true?
  - $n^3 = O\big(n^2\big)$ F $\lim_{n \to \infty} \frac{n^3}{n^2} = \infty$
  - $10n^4 = O\big(n^5\big)$ T $c = 4,\ n_0 = 4$ $10n^4 \le 4n^5\ \forall n \ge 4$
  - $\log_2 n = O(\log_{16} n)$ T $c = 10,\ n_0 = 1$

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

$\frac{\log_2 n}{\log_2 16} = \frac{1}{4} \log_2 n$ $c = 6,\ n_0 = 1$

Asymptotic Analysis Rules

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

- **Smaller exponents are Big-Oh of larger exponents**
  - $\forall a > b \quad n^b = O(n^a)$ $\quad n^3 = O(n^{3.00001})$
- **Any logarithm is Big-Oh of any polynomial**
  - $\forall a, \varepsilon > 0 \quad \log_2^a n = O(n^\varepsilon)$ $\quad \log_2^{1000} n = O(n^{0.0001})$
- **Any polynomial is Big-Oh of any exponential**
  - $\forall\ a > 0, b > 1 \quad n^a = O(b^n)$ $\quad n^{1000} = O(1.0001^n)$

A Word of Caution

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

$n^2 = O(n^3)$

$n^2 = O(n^2)$

$n^2 = O(4^n)$

not really "="

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. $f(n) \boxed{\geq} c \cdot g(n)$ 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:** $f(n) = O(g(n))$ AND $f(n) = \Omega(g(n))$

Asymptotic Running Times

- **We *like* to write running time as a Big-Theta**
  - **More precise than Oh or Omega**
  - Note: Sometimes it is considerably easier to just use Oh

More Examples

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

O: $c = 75$, $n_0 = 2$

$30 \log_2 n + 45 \leq 75 \log_2 n \quad \forall n \geq 2$

Ω: $c = 1$, $n_0 = 2$

$30 \log_2 n + 45 \geq \log_2 n \quad \forall n \geq 2$

- $4n \log_2 2n = \Theta(n \log n)$ $\quad 4n(\log_2 2 + \log_2 n)$

$\log_x ab = \log_x a + \log_x b$

O: $c = 5$, $n_0 = 16$
$= 8$, $= 2$

$4n + 4n \log_2 n \leq 5 n \log_2 n \quad \forall n \geq 16$

Ω: $c = 3$, $n_0 = 2$

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

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

O: $c = 2$, $n_0 = 1$

Ω: $c = 1/2$, $n_0 = 1$

Asymptotic Order Of Growth

Strict! $n^2 = o(n^3)$, $n^2 \neq o(n^2)$

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

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
  - $100n$
  - $3^{\log_2 n}$ $= 2^{\log_2 3^{\log_2 n}} = 2^{\log_2 n^{\log_2 3}} = n^{\log_2 3} \approx n^{1.585}$

$a = b^{\log_b a}$

correct

$\boxed{1.\ 100n \quad 2.\ n \log n \quad 3.\ 3^{\log_2 n} \quad 4.\ n^2}$

$100n \quad n \log n \quad n^2 \quad 3^{\log_2 n}$

Motivation: How Big of a Difference?

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)
3. Large instances are when there is a difference (log still good!)

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