CS221: Algorithms and Data Structures Big-O

Alan J. Hu

(Borrowing some slides from Steve Wolfman)

Learning Goals

- Define big-O, big-Omega, and big-Theta: $O(\bullet)$, $\Omega(\bullet)$, $\Theta(\bullet)$
- Explain intuition behind their definitions.
- Prove one function is big-O/Omega/Theta of another function.
- Simplify algebraic expressions using the rules of asymptotic analysis.
- List common asymptotic complexity orders, and how they compare.
- Work some examples.

Asymptotic Analysis of Algorithms

From last time, some key points:

- We will measure runtime, or memory usage, or whatever we are comparing, as a **function in terms of the input size n.**
- Because **we are comparing algorithms**, we only count “basic operations”, and since we don’t know how long each basic operation will really take, **we ignore constant factors.**
- We **focus only on when n gets big**.

Asymptotic Analysis of Algorithms

From last time, some key points:

- We will measure runtime, or memory usage, or whatever we are comparing, as a **function in terms of the input size n.**
- Because **we are comparing algorithms**, we only count "basic operations", and since we don't know how long each basic operation will really take, **we ignore constant factors.**
- We **focus only on when n gets big**.

Runtime Smackdown!

Alan’s Old Thinkpad x40

- Older Laptop
- Pentium M 32bit CPU at 1.4Ghz
- 1.5 GB of RAM

Pademelon

- 2011 Desktop PC
- Core i7-870 64bit CPU at 3Ghz w/ TurboBoost
- 16GB of RAM

Which computer is faster? By how much?

Runtime Smackdown II!

Tandy 200

- 1984 Laptop
- Intel 8085 8bit CPU at 2.4Mhz
- 24KB of RAM
- Interpreted BASIC

Pademelon

- 2011 Desktop PC
- Core i7-870 64bit CPU at 3Ghz w/ TurboBoost
- 16GB of RAM
- Compiled C++

Which computer is faster? By how much?

Runtime Smackdown III!

Tandy 200

- 1984 Laptop
- Intel 8085 8bit CPU at 2.4Mhz
- 24KB of RAM
- Interpreted BASIC

Pademelon

- 2011 Desktop PC
- Core i7-870 64bit CPU at 3Ghz w/ TurboBoost
- 16GB of RAM
- Compiled C++

Which computer is faster? By how much?

But what if we run asymptotically different algorithms?

Asymptotic Analysis of Algorithms

From last time, some key points:

- We will measure runtime, or memory usage, or whatever we are comparing, as a **function in terms of n.**
- Because **we are comparing algorithms**, we only count "basic operations", and since we don't know how long each basic operation will really take, **we ignore constant factors.**
- We **focus only on when n gets big**.

Silicon Downs

Post #1	Post #2
$n^3 + 2n^2$	$100n^2 + 1000$
$n^{0.1}$	$\log n$
$n + 100n^{0.1}$	$2n + 10 \log n$
$5n^5$	$n!$
$n^{-15}2^n/100$	$1000n^{15}$
$8^{2\lg n}$	$3n^7 + 7n$
mn^3	$2^m n$

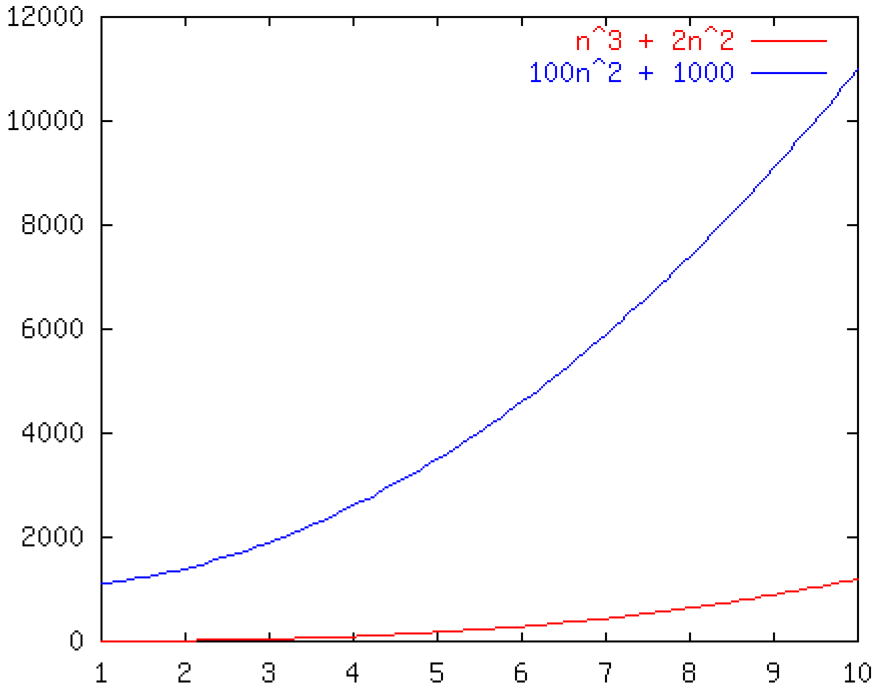
For each race, which “horse” grows bigger as n goes to infinity? (Note that in practice, smaller is better.)

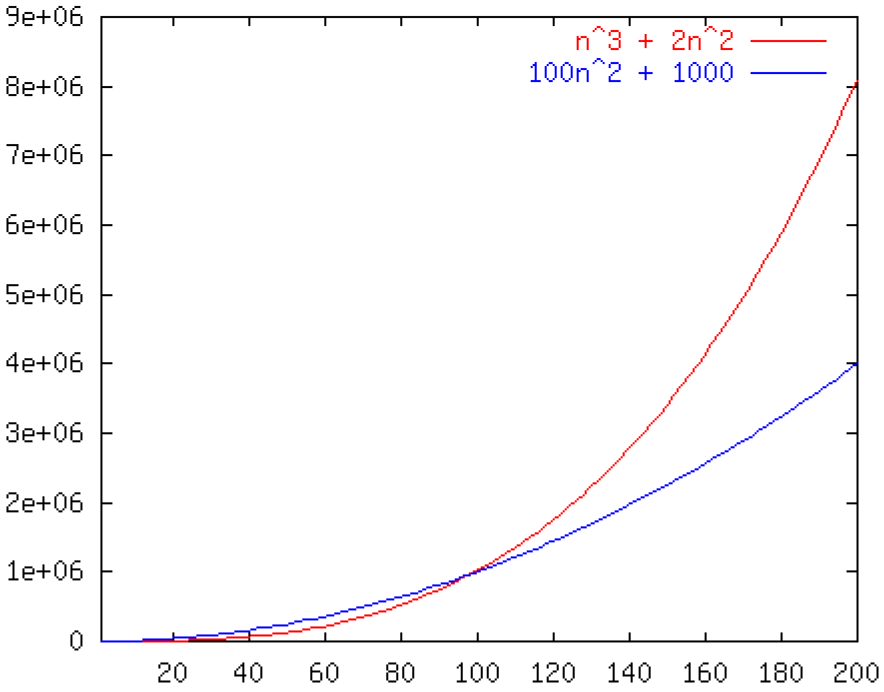
a. Left
b. Right
c. Tied
d. It depends
e. I am opposed to algorithm racing.

Race I

a. Left
b. Right
c. Tied
d. It depends

$n^3 + 2n^2$ vs. $100n^2 + 1000$

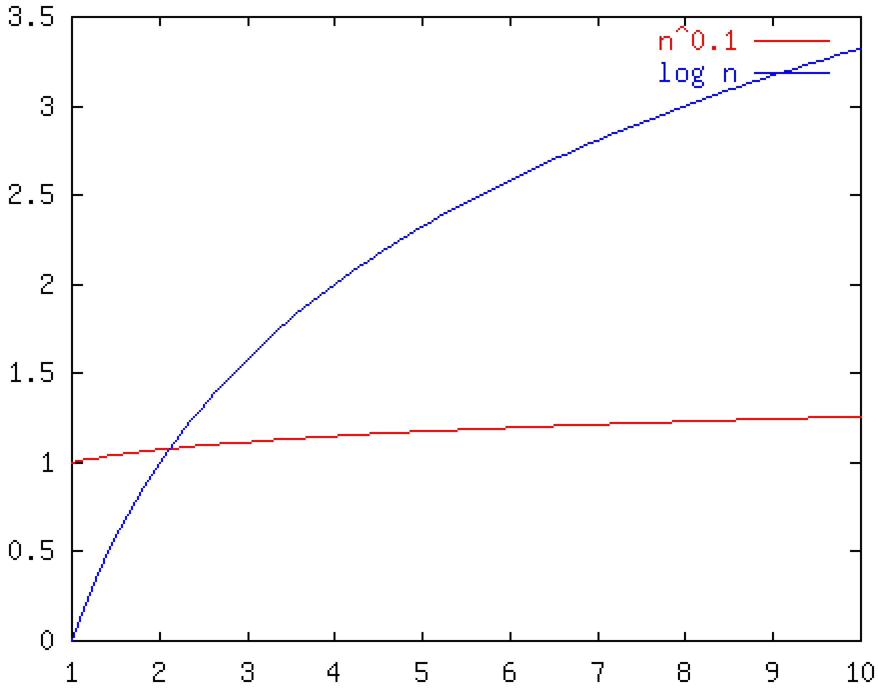


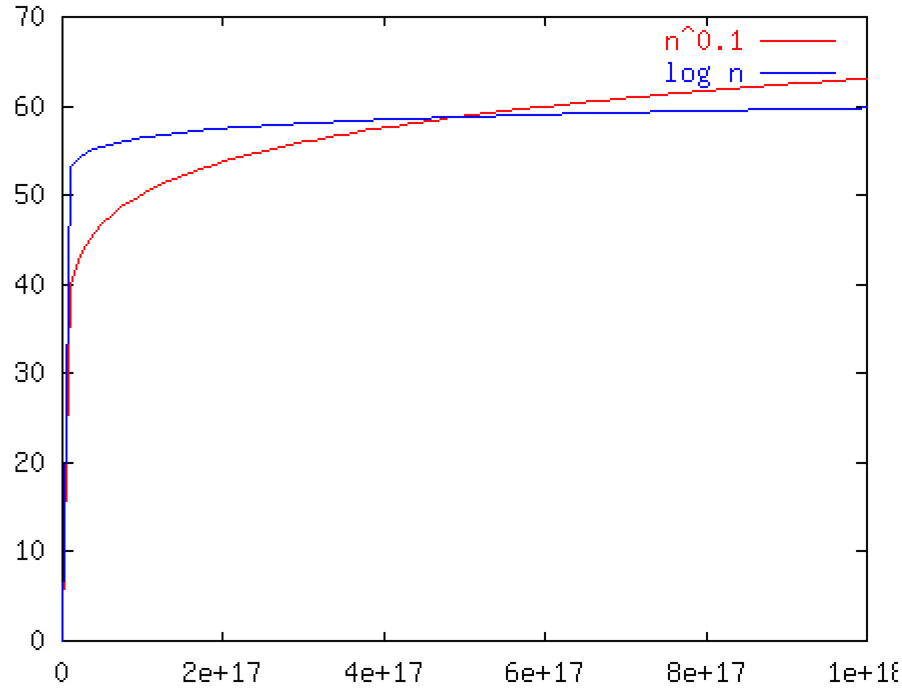


Race II

a. Left
b. Right
c. Tied
d. It depends

$n^{0.1}$ vs. **log n**

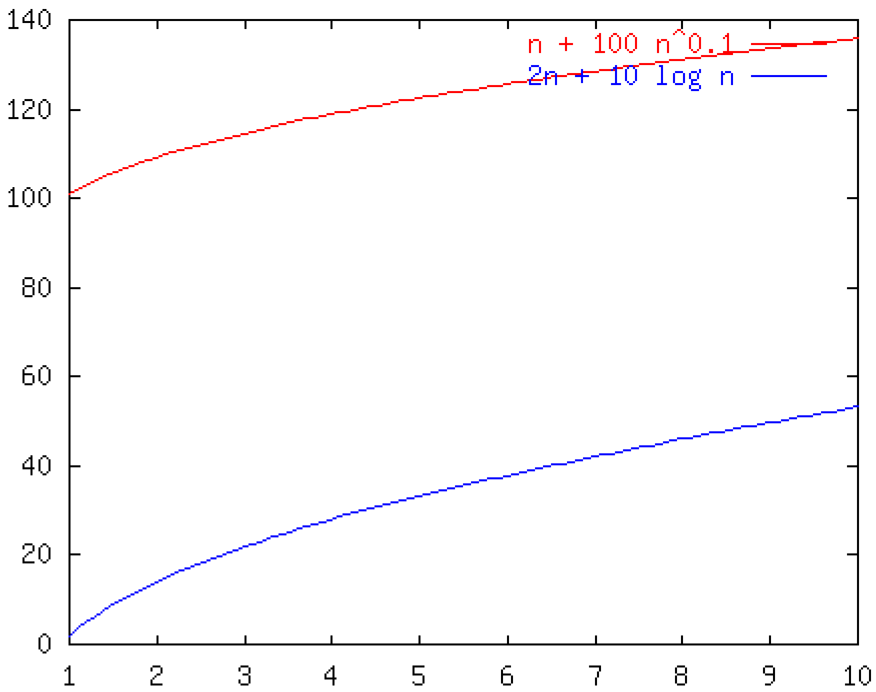


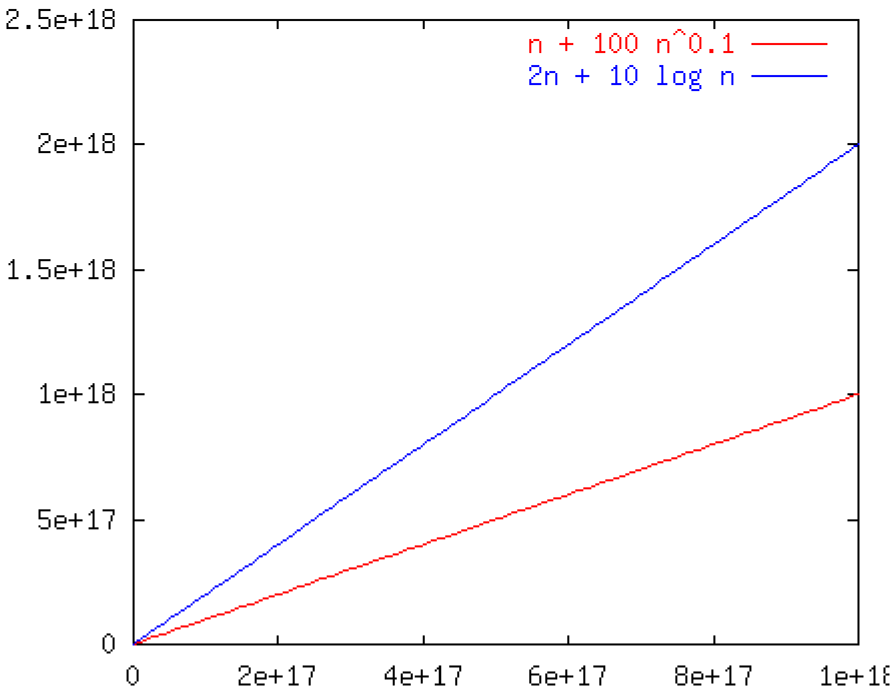


Race III

a. Left
b. Right
c. Tied
d. It depends

$\mathbf{n + 100n^{0.1}}$ vs. $\mathbf{2n + 10 \log n}$

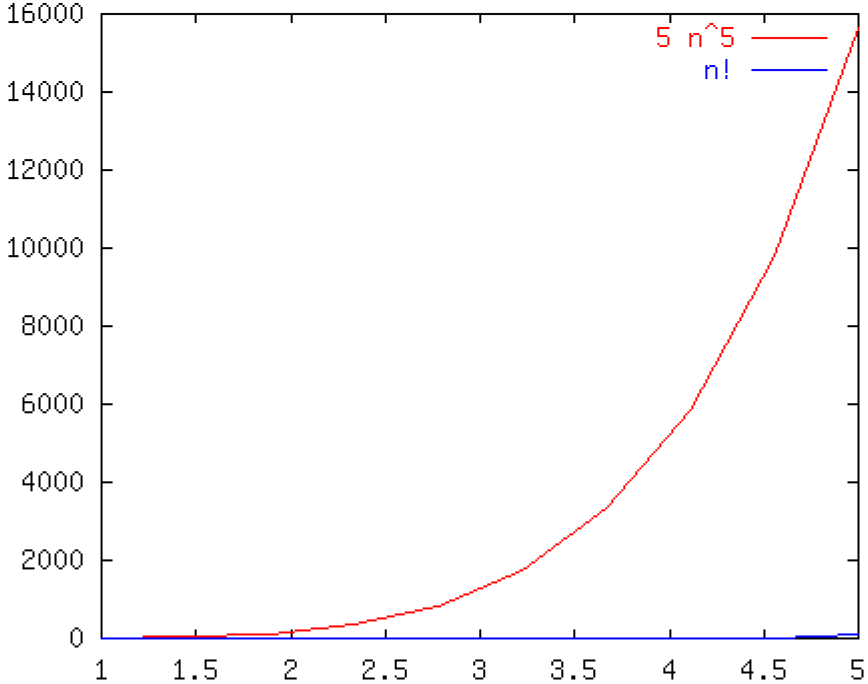


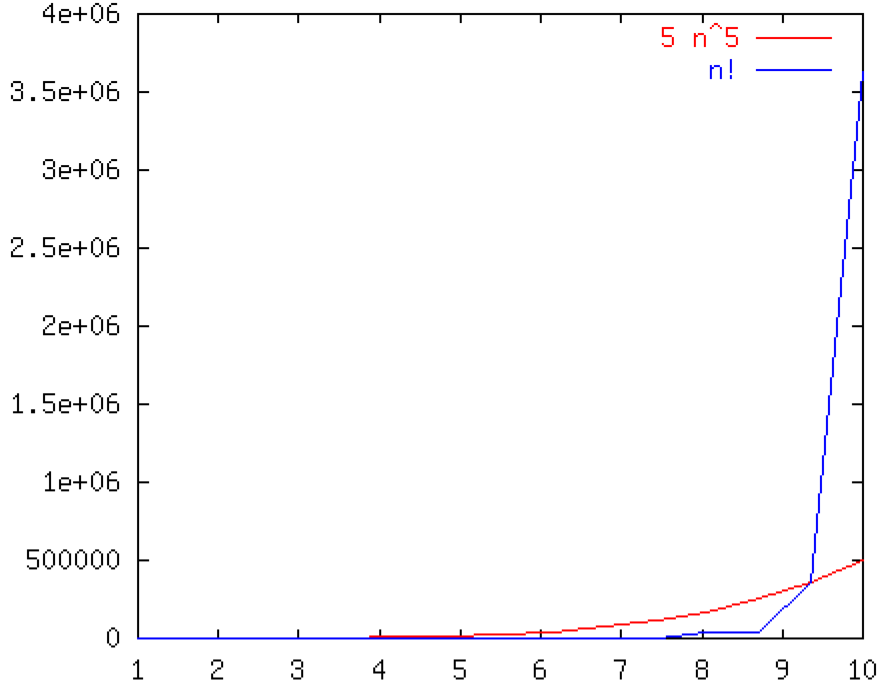


Race IV

a. Left
b. Right
c. Tied
d. It depends

$5n^5$ vs. $n!$

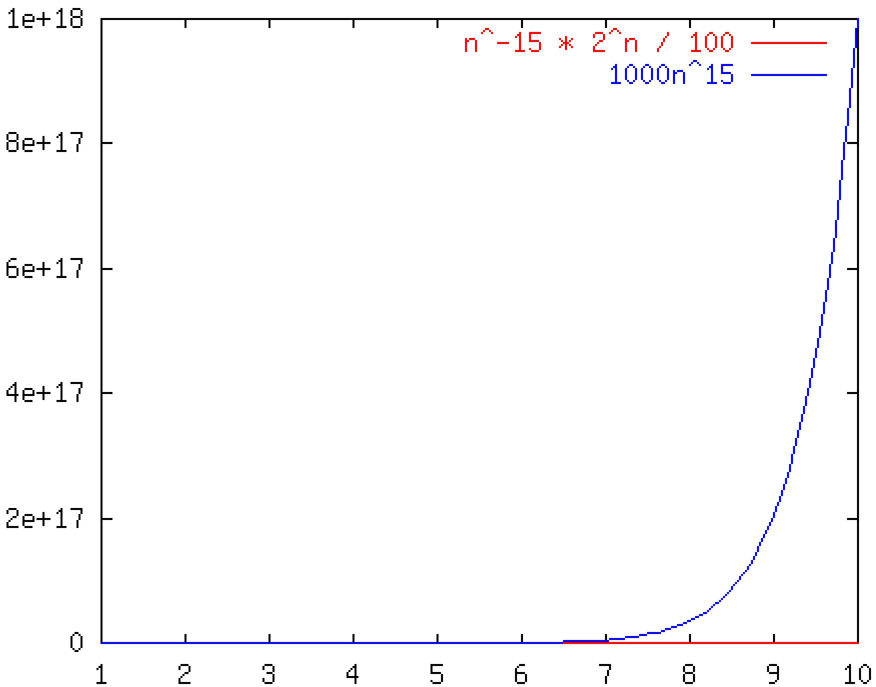


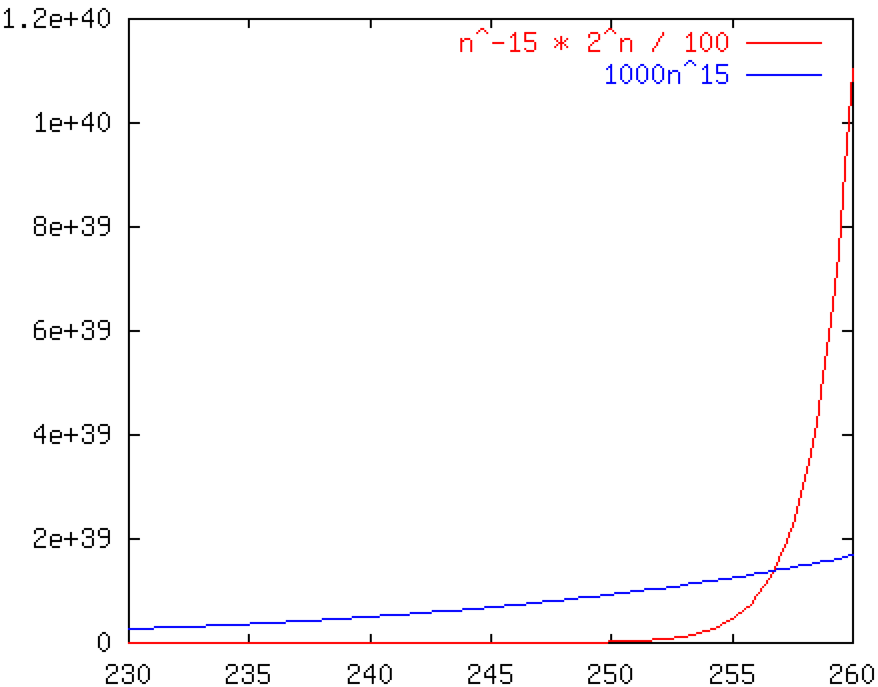


Race V

a. Left
b. Right
c. Tied
d. It depends

$n^{-15}2^n/100$ vs. $1000n^{15}$

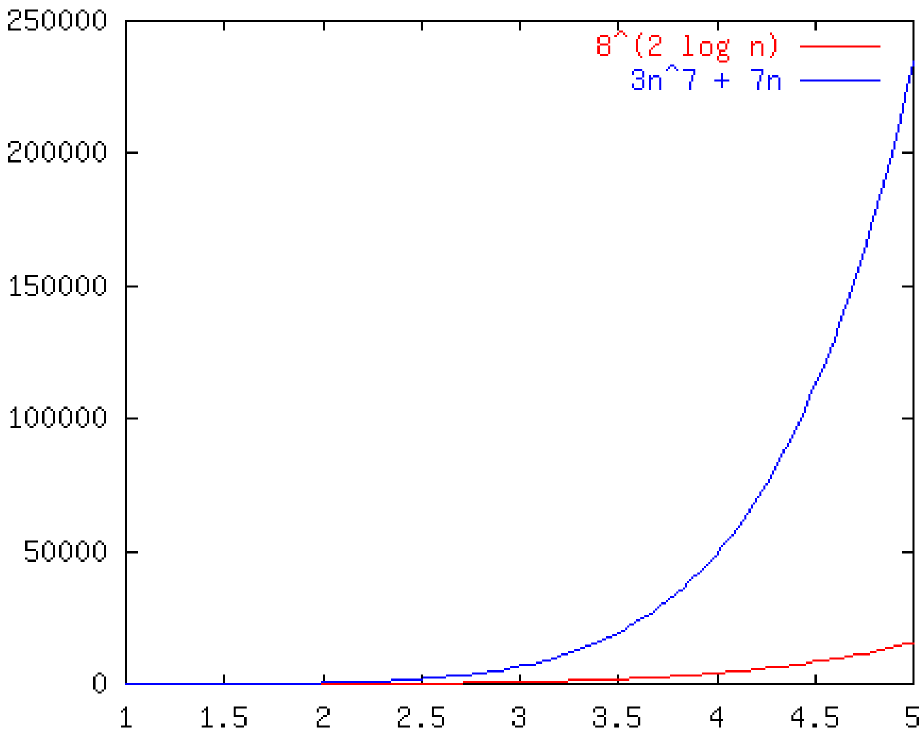




Race VI

a. Left
b. Right
c. Tied
d. It depends

$8^{2\lg(n)}$ vs. $3n^7 + 7n$



a. Left
b. Right
c. Tied
d. It depends

Race VII

mn^3 vs. $2^m n$

Silicon Downs

Post #1	Post #2	Grows Bigger
$n^3 + 2n^2$	$100n^2 + 1000$	$n^3 + 2n^2$
$n^{0.1}$	$\log n$	$n^{0.1}$
$n + 100n^{0.1}$	$2n + 10 \log n$	$2n + 10 \log n$ (tied)
$5n^5$	$n!$	$n!$
$n^{-15}2^n/100$	$1000n^{15}$	$n^{-15}2^n/100$
$8^{2\lg n}$	$3n^7 + 7n$	$3n^7 + 7n$
mn^3	$2^m n$	**IT DEPENDS**

Order Notation

- We’ve seen why we focus on the big inputs.
- We modeled that formally as the asymptotic behavior, as input size goes to infinity.
- We looked at a bunch of Steve’s “races”, to see which function “wins” or “loses”.
- How do we formalize the notion of winning? How do we formalize that one function “eventually catches up and grows faster”?

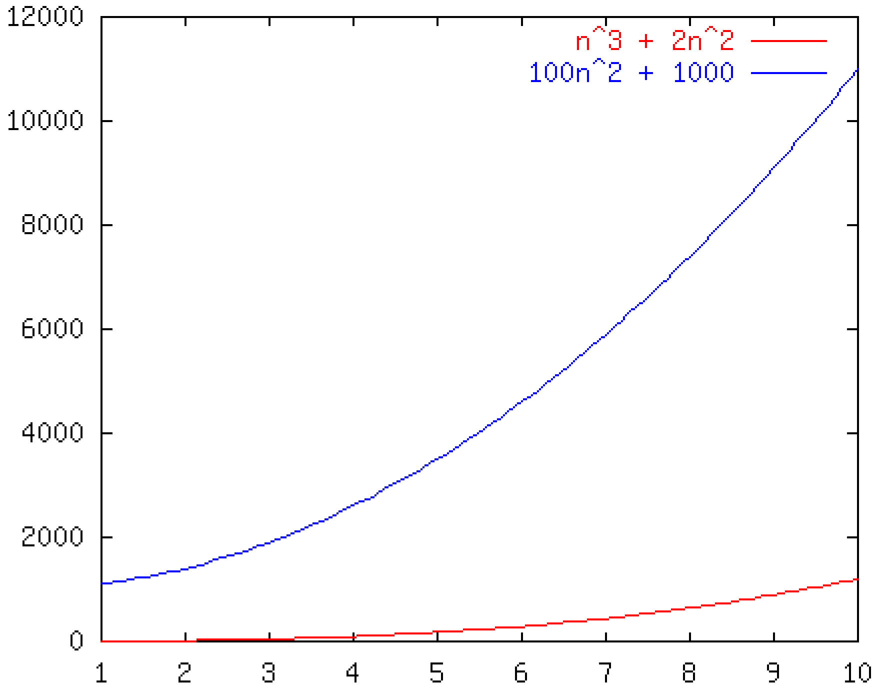
Order Notation

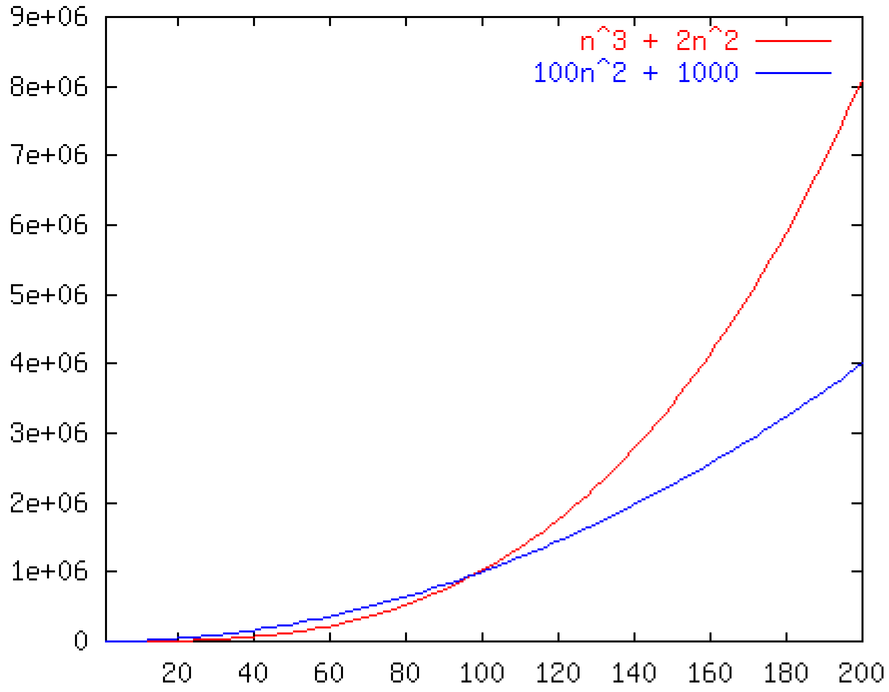
- We’ve seen why we focus on the big inputs.
- We modeled that formally as the asymptotic behavior, as input size goes to infinity.
- We looked at a bunch of Steve’s “races”, to see which function “wins” or “loses”.
- How do we formalize the notion of winning? How do we formalize that one function “eventually catches up and grows faster”?

Race I

a. Left
b. Right
c. Tied
d. It depends

$n^3 + 2n^2$ vs. $100n^2 + 1000$

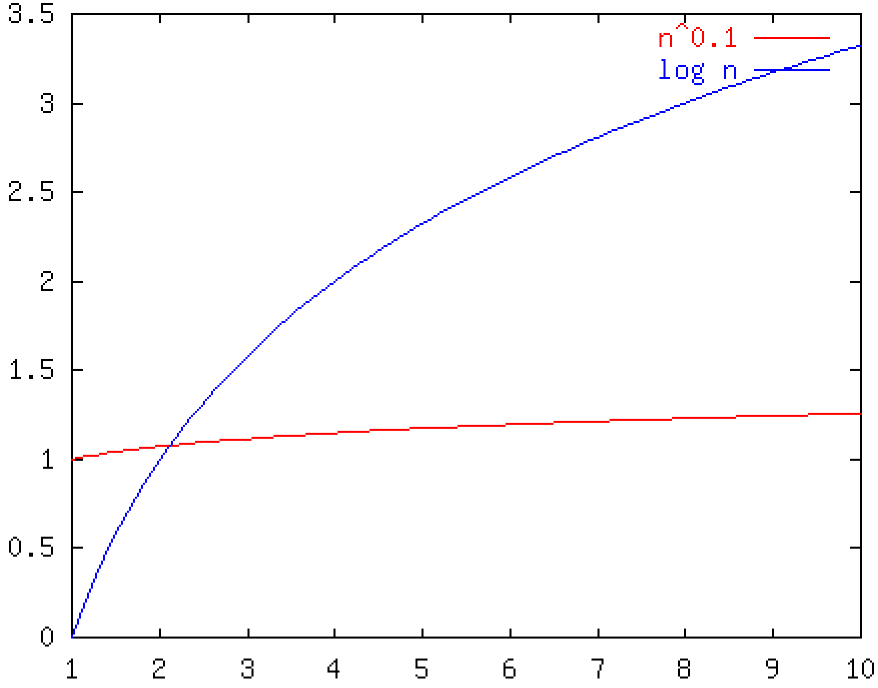


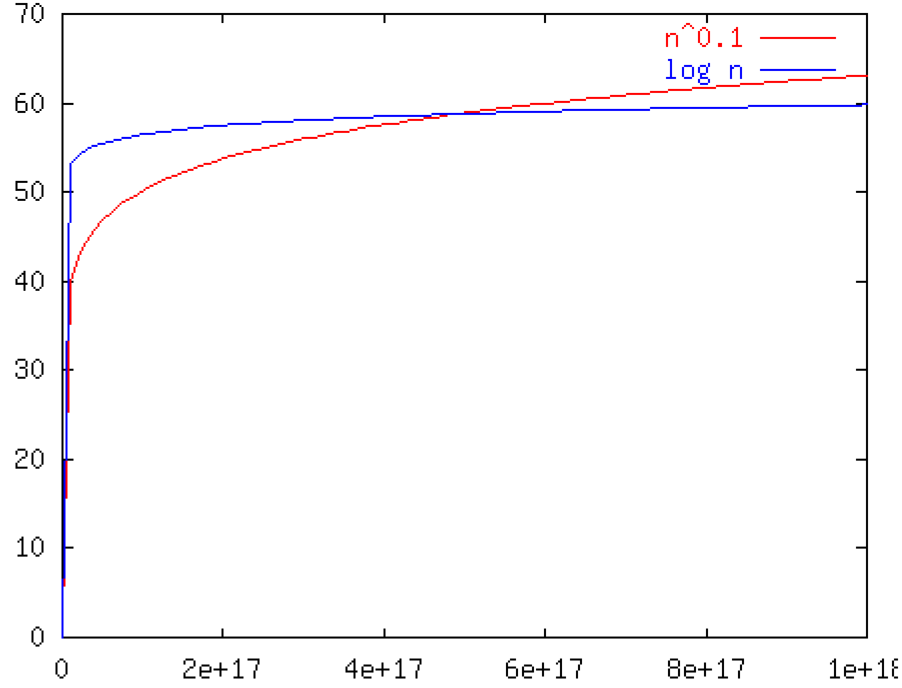


a. Left
b. Right
c. Tied
d. It depends

Race II

$n^{0.1}$ vs. $\log n$

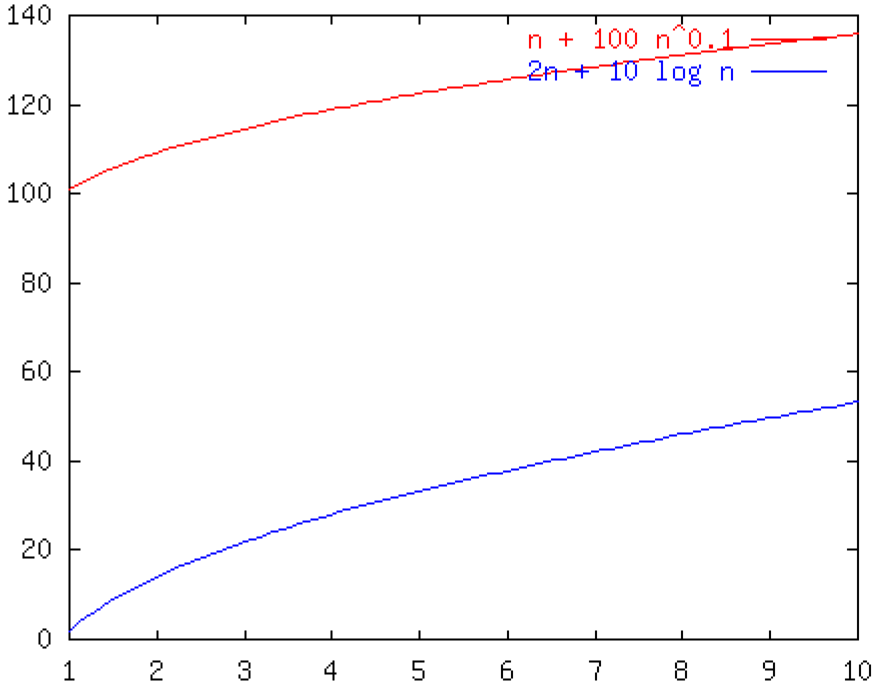


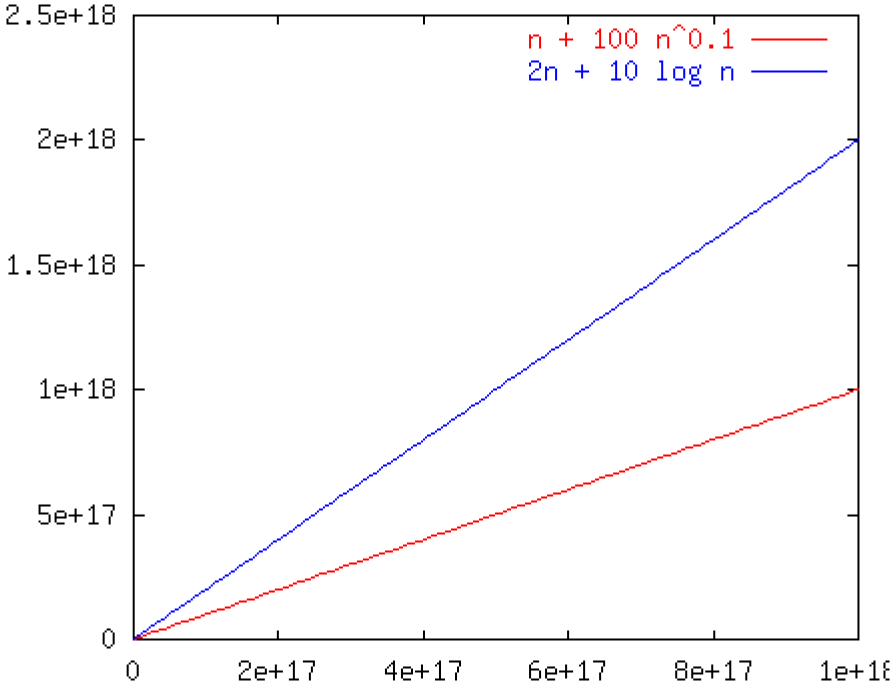


Race III

a. Left
b. Right
c. Tied
d. It depends

n + 100n$^{0.1}$ vs. **2n + 10 log n**





How to formalize winning?

- How to formally say that there’s some crossover point, after which one function is bigger than the other?
- How to formally say that you don’t care about a constant factor between the two functions?

Order Notation – Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$

Order Notation – Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$

- Why the n_0 ?
- Why the c ?

Order Notation – Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\ f(n)$ for all $n \geq n_0$

- Why the $\in$?

  (Many people write $T(n)=O(f(n))$,

  but this is sloppy. The $\in$ shows you why

  you should never write $O(f(n))=T(n)$,

  with the big-O on the left-hand side.)

Order Notation – Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$
- Intuitively, what does this all mean?

Order Notation – Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$

- Intuitively, what does this all mean?

The function f(n) is sort of, asymptotically "greater than or equal to" the function T(n).

In the "long run", f(n) (multiplied by a suitable constant) will upper-bound T(n).

Order Notation – Big-Theta and Big-Omega

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \le c\, f(n)$ for all $n \ge n_0$
- $T(n) \in \Omega(f(n))$ if $f(n) \in O(T(n))$
- $T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$

Examples

$10{,}000\ n^2 + 25\ n \in \Theta(n^2)$

$10^{-10}\ n^2 \in \Theta(n^2)$

$n \log n \in O(n^2)$

$n \log n \in \Omega(n)$

$n^3 + 4 \in O(n^4)$ but not $\Theta(n^4)$

$n^3 + 4 \in \Omega(n^2)$ but not $\Theta(n^2)$

Proofs?

$10{,}000\, n^2 + 25\, n \in \Theta(n^2)$

$10^{-10}\, n^2 \in \Theta(n^2)$

$n \log n \in O(n^2)$

$n \log n \in \Omega(n)$

$n^3 + 4 \in O(n^4)$ but not $\Theta(n^4)$

$n^3 + 4 \in \Omega(n^2)$ but not $\Theta(n^2)$

How do you prove a big-O? a big-Ω ? a big-Θ ?

Proving a Big-O

- $T(n) \in O(f(n))$ if there are constants $c > 0$ and n_0 such that $T(n) \leq c\, f(n)$ for all $n \geq n_0$

- Formally, to prove $T(n) \in O(f(n))$, you must show:

$$\exists c > 0, n_0 \forall n > n_0 \left[T(n) \leq cf(n) \right]$$

- How do you prove a “there exists” property?

Proving a “There exists” Property

How do you prove “There exists a good restaurant in Vancouver”?

How do you prove a property like

$$\exists c\left[c = 3c + 1\right]$$

Proving a $\exists \ldots \forall \ldots$ Property

How do you prove “There exists a restaurant in Vancouver, where all items on the menu are less than $10”?

How do you prove a property like

$$\exists c \forall x \left[c \leq x^2 - 10 \right]$$

Proving a Big-O

Formally, to prove $T(n) \in O(f(n))$, you must show:

$$\exists c > 0, n_0 \forall n > n_0 \left[T(n) \leq cf(n) \right]$$

So, we have to come up with specific values of c and n_0 that "work", where "work" means that for any $n > n_0$ that someone picks, the formula holds:

$$\left[T(n) \leq cf(n) \right]$$

Proving Big-O -- Example

$10{,}000\, n^2 + 25\, n \in \Theta(n^2)$

$10^{-10}\, n^2 \in \Theta(n^2)$

$\boxed{n \log n \in O(n^2)}$

$n \log n \in \Omega(n)$

$n^3 + 4 \in O(n^4)$ but not $\Theta(n^4)$

$n^3 + 4 \in \Omega(n^2)$ but not $\Theta(n^2)$

Prove $n \log n \in O(n^2)$

- Guess or figure out values of c and n_0 that will work.

  (Let's assume base-10 logarithms.)

Prove $n \log n \in O(n^2)$

- Guess or figure out values of c and n_0 that will work.

  (Let's assume base-10 logarithms.)
- Turns out c=1 and $n_0 = 1$ works!

  (What happens if you guess wrong?)

Prove n log n $\in O(n^2)$

- Guess or figure out values of c and n_0 that will work.

  (Let's assume base-10 logarithms.)
- Turns out c=1 and $n_0 = 1$ works!
- Now, show that n log n <= n^2 , for all n>1

Prove $n \log n \in O(n^2)$

- Guess or figure out values of c and n_0 that will work.

  (Let's assume base-10 logarithms.)
- Turns out c=1 and n_0 = 1 works!
- Now, show that n log n <= n^2 , for all n>1
- This is fairly trivial: log n <= n (for n>1)

  Multiply both sides by n (OK, since n>1>0)

Aside: Writing Proofs

- In lecture, my goal is to give you intuition.
  - I will just sketch the main points, but not fill in all details.
- When you ***write*** a proof (homework, exam, reports, papers), be sure to write it out formally!
  - Standard format makes it much easier to write!
    - Class website has links to notes with standard tricks, examples
    - Textbook has good examples of proofs, too.
    - Copy the style, structure, and format of these proofs.
  - On exams and homeworks, you’ll get more credit.
  - In real life, people will believe you more.

To Prove $n \log n \in O(n^2)$

Proof:

By the definition of big-O, we must find values of c and n_0 such that for all $n \geq n_0$, $n \log n \leq cn^2$.

Consider $c=1$ and $n_0 = 1$.

For all $n \geq 1$, $\log n \leq n$.

Therefore, $\log n \leq cn$, since $c=1$.

Multiplying both sides by n (and since $n \geq n_0 = 1$), we have $n \log n \leq cn^2$.

Therefore, $n \log n \in O(n^2)$.

QED

(This is more detail than you'll use in the future, but until you learn what you can skip, fill in the details.)

Proving Big-Ω

- Just like proving Big-O, but backwards…

Proving Big-Θ

- Just prove Big-O and Big-Ω

Proving Big-Θ -- Example

$\boxed{10{,}000\, n^2 + 25\, n \in \Theta(n^2)}$

$10^{-10}\, n^2 \in \Theta(n^2)$

$n \log n \in O(n^2)$

$n \log n \in \Omega(n)$

$n^3 + 4 \in O(n^4)$ but not $\Theta(n^4)$

$n^3 + 4 \in \Omega(n^2)$ but not $\Theta(n^2)$

Prove $10{,}000\, n^2 + 25\, n \in O(n^2)$

- What values of c and n_0 work?

(Lots of answers will work…)

Prove $10{,}000\ n^2 + 25\ n \in O(n^2)$

- What values of c and n_0 work?

  I'll use c=10025 and $n_0 = 1$.

$$10{,}000\ n^2 + 25\ n \le 10{,}000\ n^2 + 25\ n^2$$
$$\le 10{,}025\ n^2$$

Prove $10{,}000\, n^2 + 25\, n \in \Omega(n^2)$

- What is this in terms of Big-O?

Prove $n^2 \in O(10{,}000\, n^2 + 25\, n)$

- What values of c and n_0 work?

Prove $n^2 \in O(10{,}000\ n^2 + 25\ n)$

- What values of c and n_0 work?

  I'll use c=1 and $n_0 = 1$.

$$n^2 <= 10{,}000\ n^2$$
$$<= 10{,}000\ n^2 + 25\ n$$

Therefore, $10{,}000\ n^2 + 25\ n \in \Theta(n^2)$

Mounties Find Silicon Downs Fixed

- The fix sheet (typical growth rates in order)
  - constant: $O(1)$
  - logarithmic: $O(\log n)$ ($\log_k n$, $\log n^2 \in O(\log n)$)
  - poly-log: $O(\log^k n)$ (k is a constant >1)
  - linear: $O(n)$
  - (log-linear): $O(n \log n)$ (usually called "n log n")
  - (superlinear): $O(n^{1+c})$ (c is a constant, $0 < c < 1$)
  - quadratic: $O(n^2)$
  - cubic: $O(n^3)$
  - polynomial: $O(n^k)$ (k is a constant) "tractable"
  - exponential: $O(c^n)$ (c is a constant > 1)

"intractable"

Asymptotic Analysis Hacks

- These are quick tricks to get big-Θ category.
- Eliminate low order terms
  - $4n + 5 \Rightarrow 4n$
  - $0.5\ n \log n - 2n + 7 \Rightarrow 0.5\ n \log n$
  - $2^n + n^3 + 3n \Rightarrow 2^n$
- Eliminate coefficients
  - $4n \Rightarrow n$
  - $0.5\ n \log n \Rightarrow n \log n$
  - $n \log (n^2) = 2\ n \log n \Rightarrow n \log n$

Log Aside

$\log_a b$ means "the exponent that turns **a** into **b**"

lg x means "**$\log_2 x$**" (our usual log in CS)

log x means "**$\log_{10} x$**" (the common log)

ln x means "**$\log_e x$**" (the natural log)

But… **O(lg n) = O(log n) = O(ln n)** because:

$\log_a b = \log_c b / \log_c a$ (for **c > 1**)

so, there's just a constant factor between log bases

USE those cheat sheets!

- Which is faster, n^3 or $n^3 \log n$?

- Which is faster, n^3 or $n^{3.01}/\log n$?

(Split it up and use the “dominance” relationships.)

Rates of Growth

- Suppose a computer executes 10^{12} ops per second:

n =	10	100	1,000	10,000	10^{12}
n	10^{-11}s	10^{-10}s	10^{-9}s	10^{-8}s	1s
n log n	10^{-11}s	10^{-9}s	10^{-8}s	10^{-7}s	40s
n^2	10^{-10}s	10^{-8}s	10^{-6}s	10^{-4}s	10^{12}s
n^3	10^{-9}s	10^{-6}s	10^{-3}s	1s	10^{24}s
2^n	10^{-9}s	10^{18}s	10^{289}s		

10^4s = 2.8 hrs

10^{18}s = 30 billion years