Analyzing Running Time (Chapter 2)

- What is efficiency?
- Tools: asymptotic growth of functions
- Practice finding asymptotic running time of algorithms

Is My Algorithm Efficient?

- Idea: Implement it, time how long it takes.
- Problems?
  - Effects of the programming language?
  - Effects of the processor?
  - Effects of the amount of memory?
  - Effects of other things running on the computer?
  - Effects of the input values?
  - Effects of the input size?

Worst-Case Running Time

- Worst-case running time: bound the largest possible running time on any input of size N
- Seems pessimistic, but:
  - Effective in practice
  - Hard to find good alternative (e.g., average case analysis)

Brute Force

- Good starting point for thinking about efficiency: can we do better than a "brute force" solution?
- What is the brute force solution for the stable matching problem?

Worst-case: Gale-Shapley vs. Brute Force

# Colleges N	4	8	16
G-S N^2	16	64	256
Brute Force N!	24	40,320	20,922,789,888,000

Working Definition of Efficient

- Efficient = better than brute force
- Desired property: if input size increases by constant factor algorithm slows down by a constant factor
  - E.g. size N -> 2N, time T -> 4T

Polynomial Time

- Definition: an algorithm runs in polynomial time if
  - Number of steps is at most $c * N^d$, where N is input size, and c and d are constants
- Does this satisfy desired property?

If there is no polynomial time solution, we say there is no efficient solution.

Running Times as Functions of Input Size

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	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

Asymptotic Growth: Big O(), Ω(), Θ()

- Goal: build tools to help coarsely classify algorithm's running times
  - Running time = number of primitive "steps" (e.g., line of code or assembly instruction)
  - Coarse: $1.62n^2 + 3.5n + 8$ is too detailed

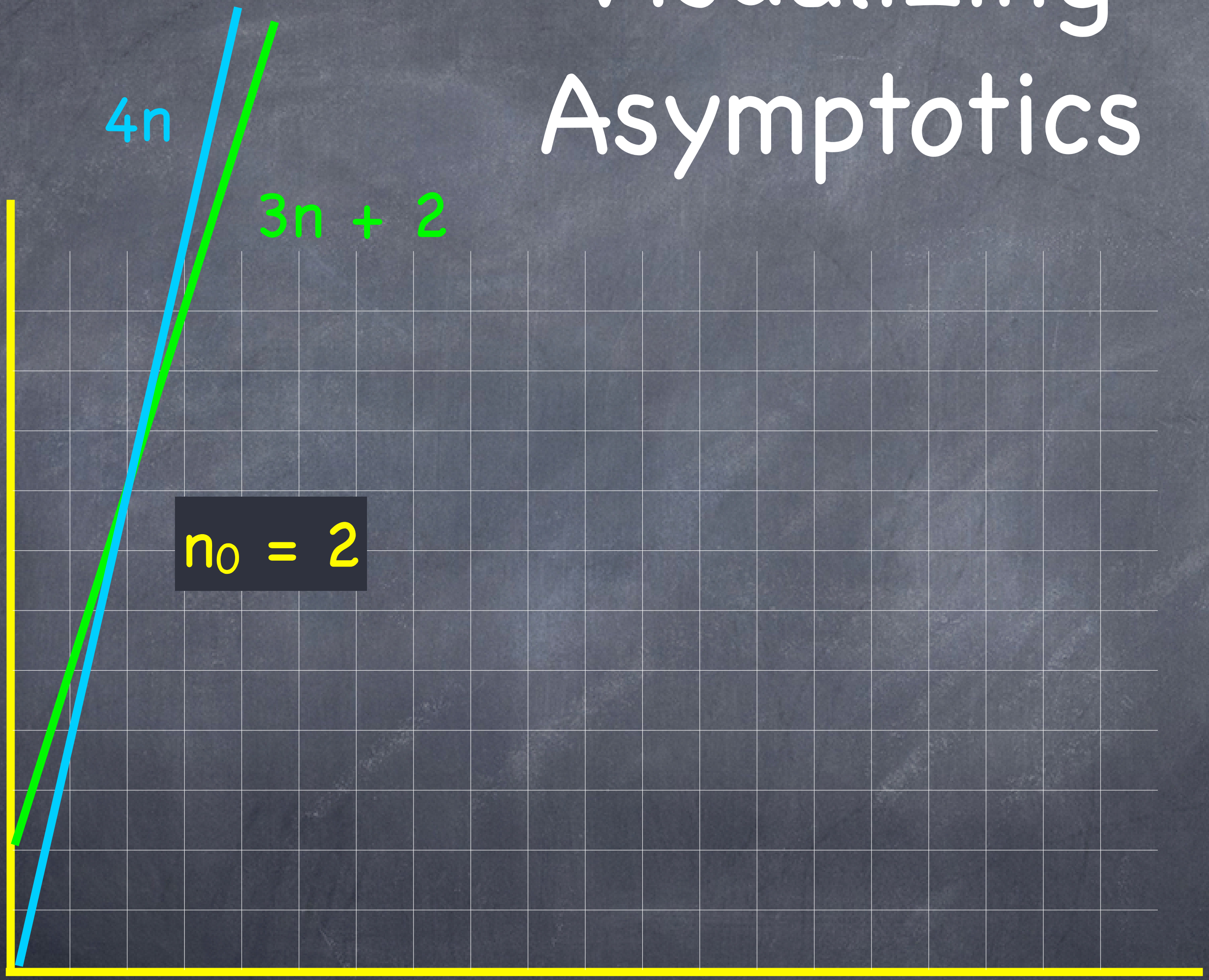
Notation

- Let $T(n)$ be a function that defines the worst-case running time of an algorithm.
- For the remainder of the lecture, we assume that all functions $T(n)$, $f(n)$, $g(n)$, etc. are nonnegative

Big O Notation

- $T(n)$ is $O(f(n))$ if
  $T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$
- (Example on board)
- $O(n)$ is the asymptotic upper bound of $T(n)$.

Visualizing Asymptotics

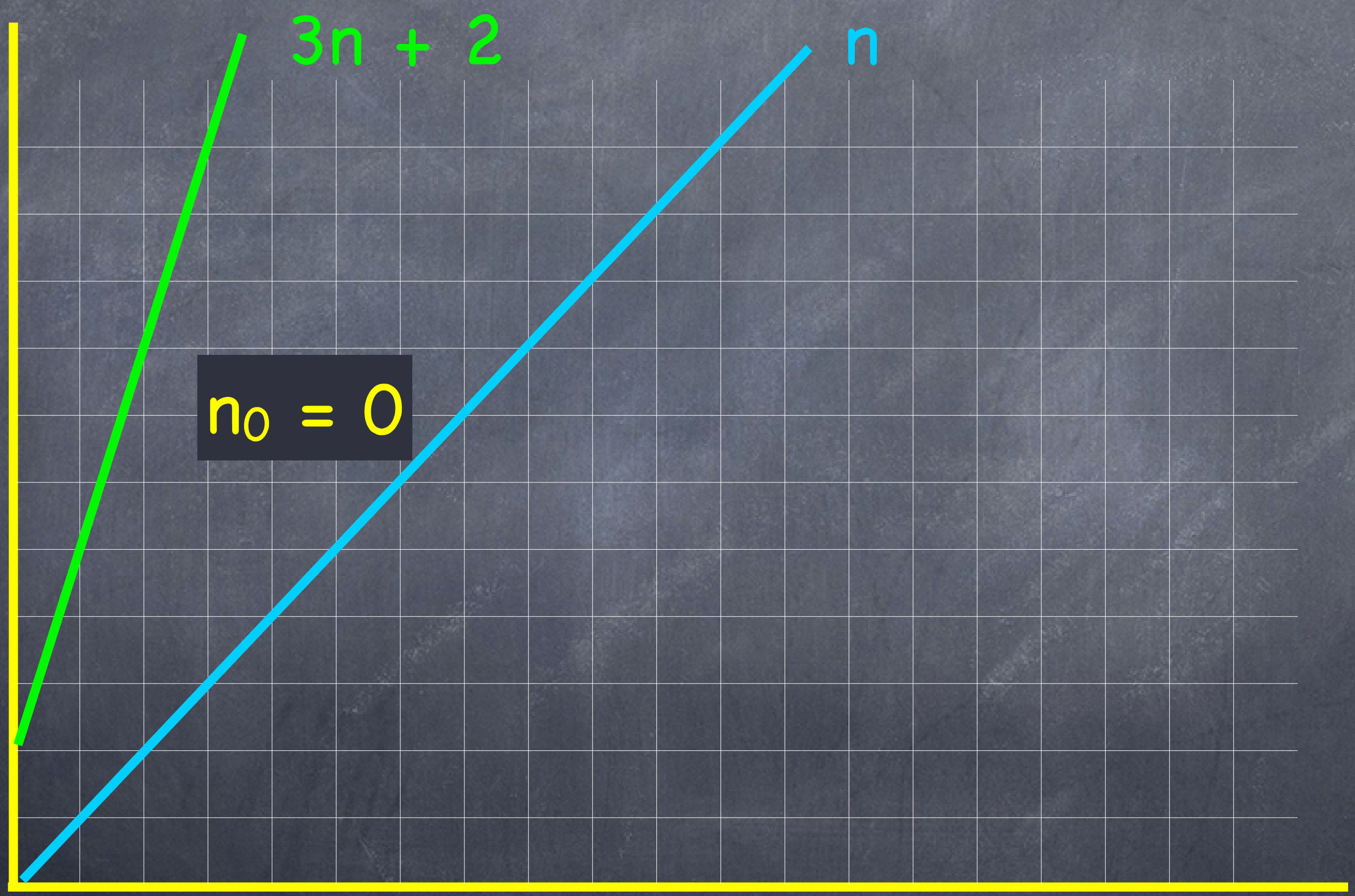


T(n) = 3n + 2 is O(n)

Ω Notation

- T(n) is Ω(f(n)) if
  $T(n) \geq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$
- (Example on board)
- Ω(n) is the asymptotic lower bound of T(n).

Visualizing Asymptotics

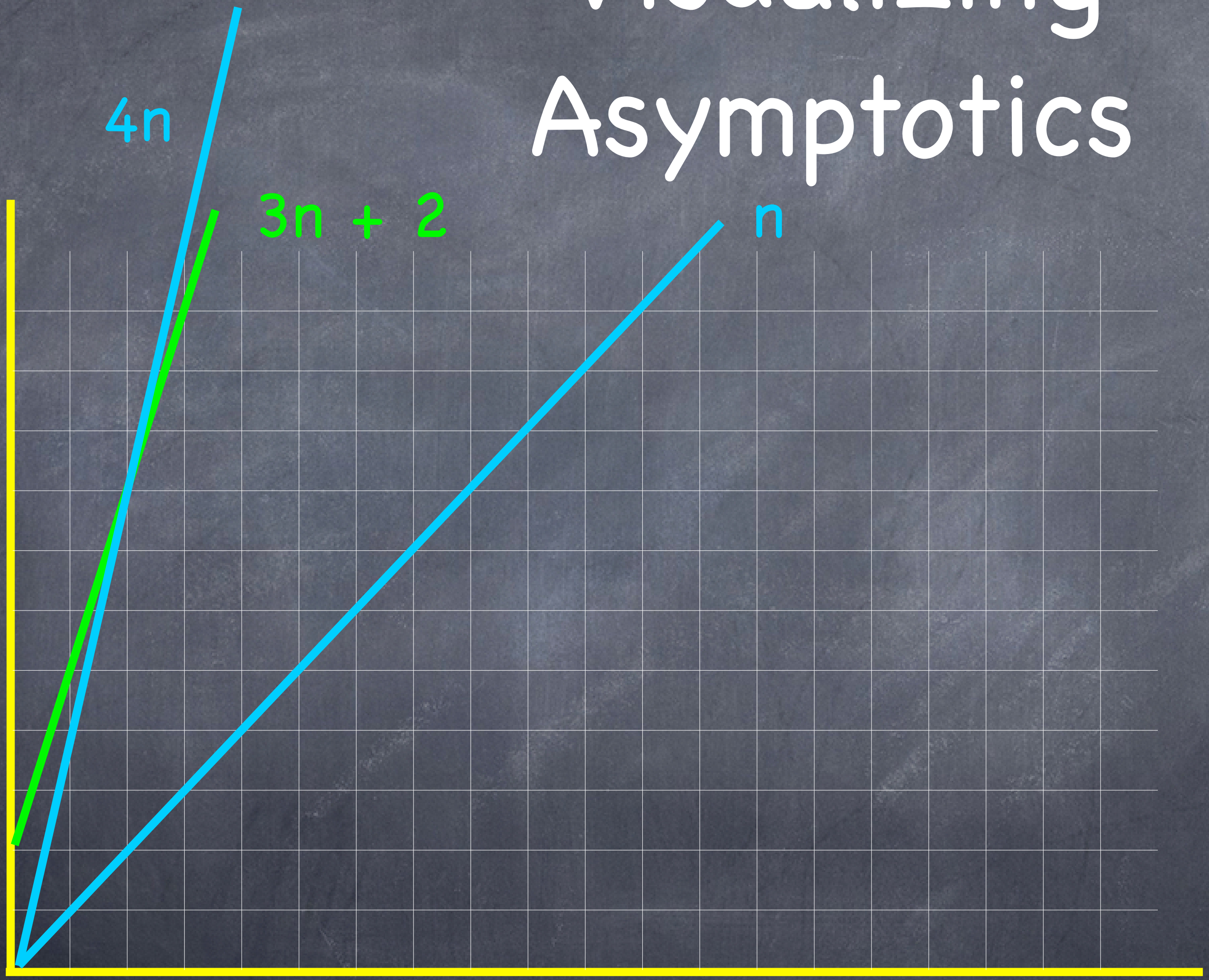


$T(n) = 3n + 2$ is $\Omega(n)$

Θ Notation

- $T(n)$ is $\Theta(f(n))$ if
  $T(n)$ is $O(n)$ and $T(n)$ is $\Omega(n)$
- (Example on board)
- $\Theta(n)$ is the asymptotic tight bound of $T(n)$.

Visualizing Asymptotics



$T(n) = 3n + 2$ is $\Theta(n)$

$O(), \Omega(), \Theta()$

- $O()$ - upper bound
  $T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

- $\Omega()$ - lower bound
  $T(n) \geq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$

- $\Theta()$ - tight bound
  Both $O()$ and $\Omega()$

Properties of $O()$, $\Omega()$, $\Theta()$

Transitivity

Claim:

(a) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$

(b) If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

(b) If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$

Prove (a) on board

Additivity

Claim:

(a) If $f = O(h)$ and $g = O(h)$, then $f+g = O(h)$

(b) If $f_1, f_2, \ldots, f_k$ are each $O(h)$, then $f_1 + f_2 + \ldots + f_k$ is $O(h)$

(c) If $f = O(g)$, then $f+g = \Theta(g)$

Prove (a) on board; discuss (c)

Asymptotic Bounds For Common Functions

Polynomial Time

Claim: Let $T(n) = c_0 + c_1 n + c_2 n^2 + \ldots + c_d n^d$, where c_d is positive. Then $T(n)$ is $\Theta(n^d)$.

Proof: repeated application of the additivity rule (c)

New definition of polynomial-time algorithm: running time $T(n)$ is $O(n^d)$

Logarithm Review

Defn: $\log_b(a)$ is the unique number c s.t. $b^c = a$

"log base b of a": Informally, the number of times you can divide a into b parts before each has size one

Facts:

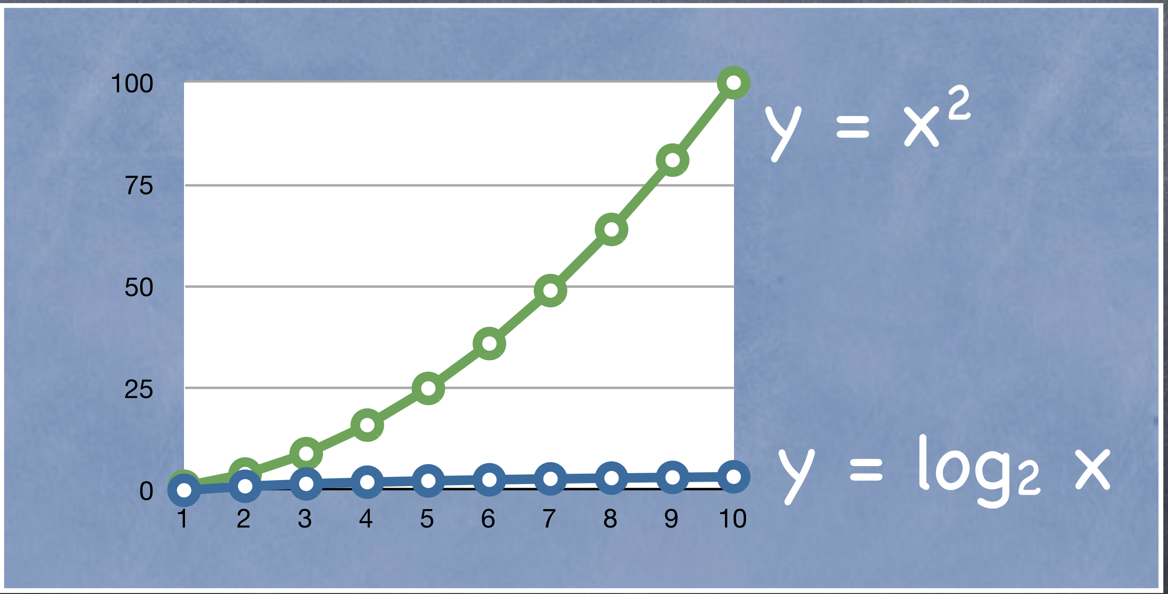
$$\log_b(b^n) = n$$

$$b^{log_b(n)} = n$$

$$\log_a(n) = \frac{log_b(n)}{log_b(a)} \qquad \log_a(n) = \Theta(log_b(n))$$

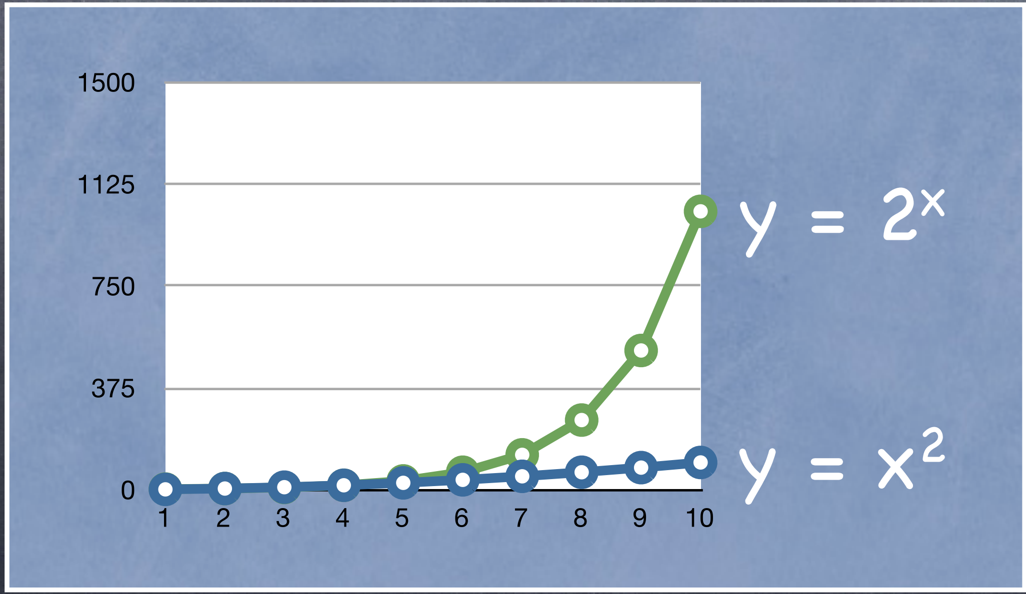
Other Asymptotic Orderings

- Logarithms:
  $\log_a n$ is $O(n^d)$, for all bases a and all degrees d
  → All logarithms grow slower than all polynomials



Other Asymptotic Orderings

- Exponential functions:
  n^d is $O(r^n)$ when $r > 1$
  ➔ Polynomials grow no more quickly than exponential functions.



A Harder Example

- Which of these grows faster?
  $n^{4/3}$
  $n(\log n)^3$

Recap

What you should know

- Polynomial time = efficient
- Definitions of O(), Ω(), Θ()
- Transitivity and additivity. Basic proof techniques
- How to "sort" functions: log(n), polynomials, n*log(n), exponentials