Back to asymptotics……

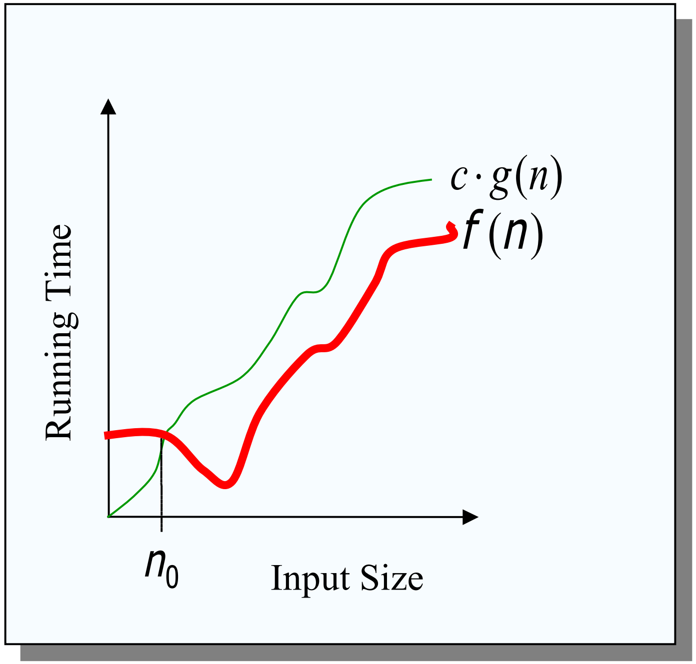
We will now look more formally at the process of simplifying running times and other measures of complexity.

Asymptotic analysis

- Goal: to simplify analysis of running time by getting rid of ”details”, which may be affected by specific implementation and hardware
  - like “rounding”: $1{,}000{,}001 \approx 1{,}000{,}000$
  - $3n^2 \approx n^2$
- Capturing the essence: how the running time of an algorithm increases with the size of the input *in the limit*.
  - Asymptotically more efficient algorithms are best for all but small inputs

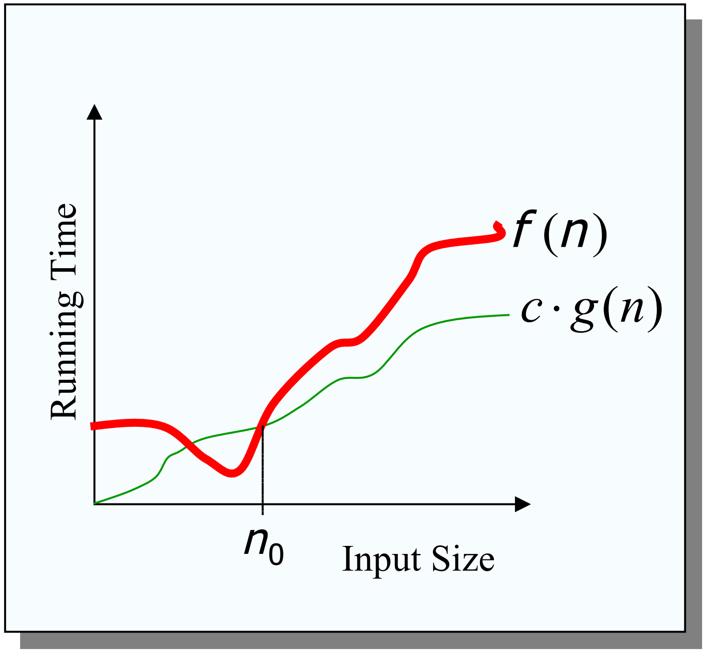
Asymptotic notation

- The "big-Oh" *O*-Notation
  - asymptotic upper bound
  - $f(n) \in O(g(n))$, if there exists constants c and n_0, *s.t.* $\mathbf{f(n) \le c\, g(n)}$ for $n \ge n_0$
  - $f(n)$ and $g(n)$ are functions over non-negative integers
- Used for *worst-case* analysis



Asymptotic notation – contd.

- The "big-Omega" Ω–Notation
  - asymptotic lower bound
  - $f(n) \in \Omega(g(n))$ if there exists constants c and n_0, *s.t.* $\mathbf{c\ g(n) \leq f(n)}$ for $n \geq n_0$
- Used to describe *best-case* running times or lower bounds of algorithmic problems
  - E.g., lower-bound of searching in an unsorted array is $\Omega(n)$.



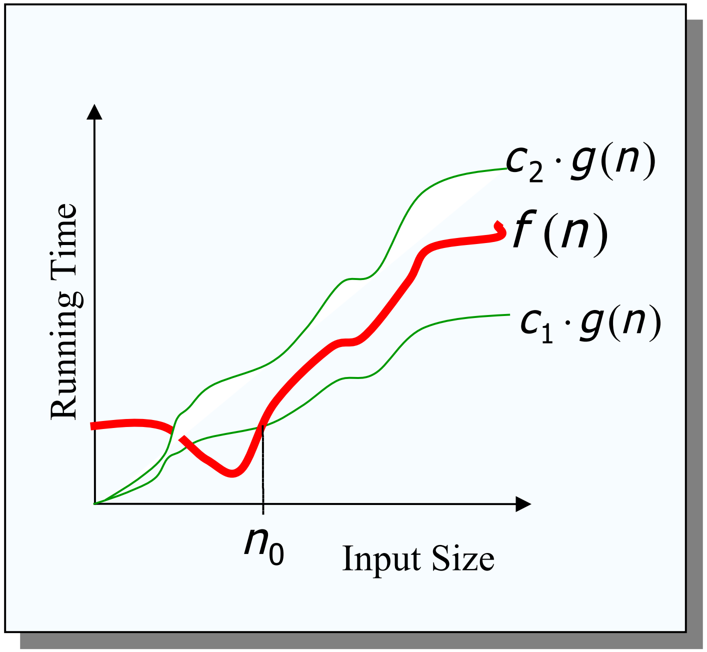
Asymptotic notation – contd.

- Simple Rule: Drop lower order terms and constant factors.
  - $50\, n \log n \in O(n \log n)$
  - $7n - 3 \in O(n)$
  - $8n^2 \log n + 5n^2 + n \in O(n^2 \log n)$

- Note: Even though $50\, n \log n \in O(n^5)$, we usually try to express a O() expression using as small an order as possible

Asymptotic notation – contd.

- The "big-Theta" Θ–Notation
  - asymptoticly tight bound
  - $f(n) \in \Theta(g(n))$ if there exists constants c_1, c_2, and n_0, $s.t.$ $\mathbf{c_1\, g(n) \le f(n) \le c_2\, g(n)}$ for $n \ge n_0$
- $f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
- $O(f(n))$ is often misused instead of $\Theta(f(n))$



Asymptotic notation – contd.

- Two more asymptotic notations
  - "Little-Oh" notation $f(n)=o(g(n))$
    non-tight analogue of Big-Oh
    - For every c, there should exist n_0, s.t. $\mathbf{f(n) \leq c\, g(n)}$ for $n \geq n_0$
    - Used for **comparisons** of running times. If $f(n) \in o(g(n))$, it is said that *g(n) dominates f(n)*.
    - More useful defn:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

  - "Little-omega" notation $f(n) \in \omega(g(n))$
    non-tight analogue of Big-Omega

Asymptotic notation – contd.

- (VERY CRUDE) Analogy with real numbers
  - $f(n) = O(g(n)) \quad \cong f \leq g$
  - $f(n) = \Omega(g(n)) \quad \cong f \geq g$
  - $f(n) = \Theta(g(n)) \quad \cong f = g$
  - $f(n) = o(g(n)) \quad \cong f < g$
  - $f(n) = \omega(g(n)) \quad \cong f > g$
- Abuse of notation: $f(n) = O(g(n))$ actually means $f(n) \in O(g(n))$.

Points to ponder and lessons

Common "colloquial" uses:

$\Theta(1)$ – *constant.*

$n^{\Theta(1)}$ – *polynomial*

$2^{\Theta(n)}$ – *exponential*

<u>Be careful!</u>

$n^{\Theta(1)} \neq \Theta(n^1)$

$2^{\Theta(n)} \neq \Theta(2^n)$

- When is asymptotic analysis useful?
- When is it NOT useful?

Many, many abuses of asymptotic notation in Computer Science literature.

Lesson: Always remember the implicit assumptions…

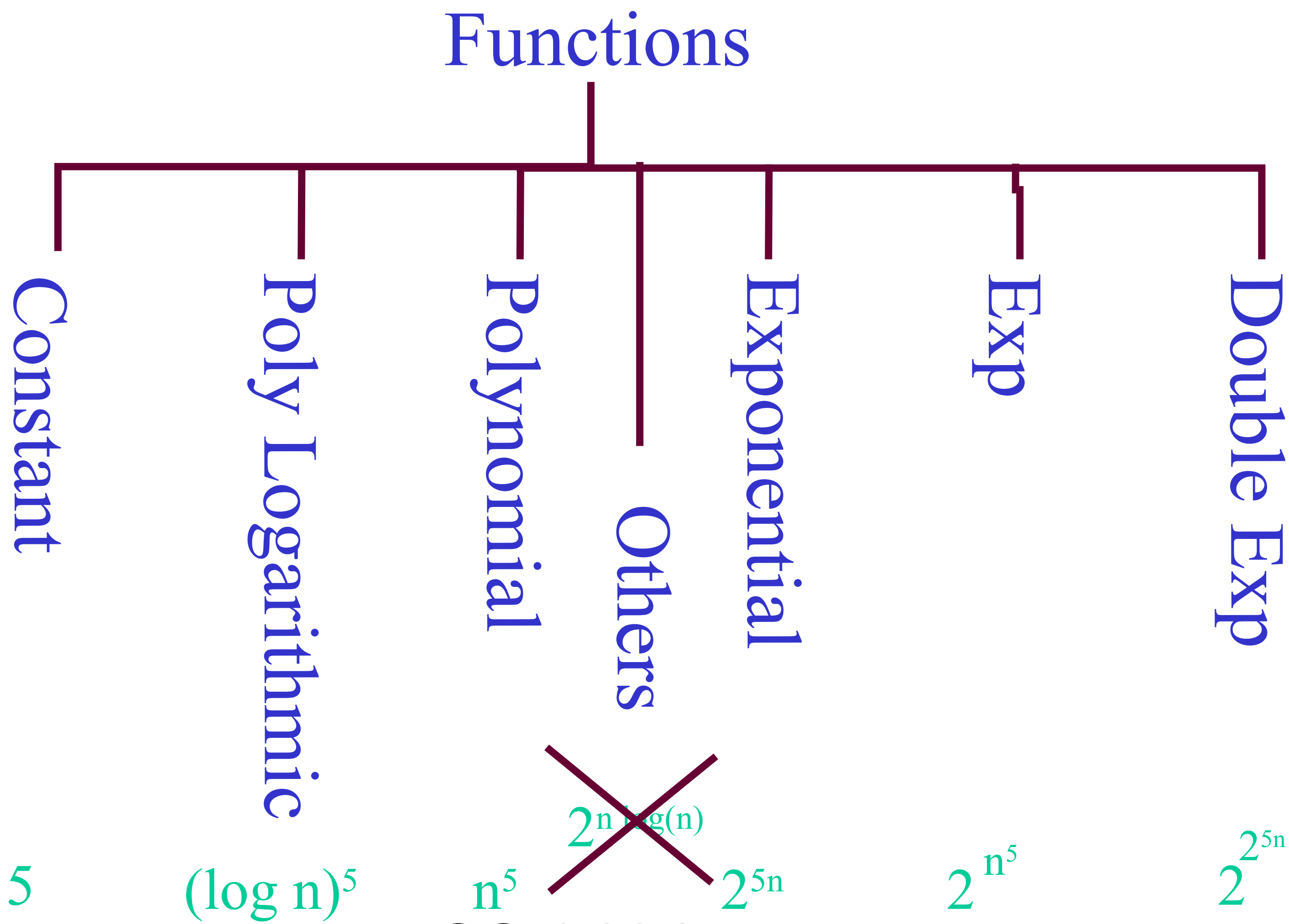
Comparison of Running Times

Running Time	Maximum problem size (n)		
	1 second	1 minute	1 hour
$400n$	2500	150000	9000000
$20n \log n$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Classifying functions

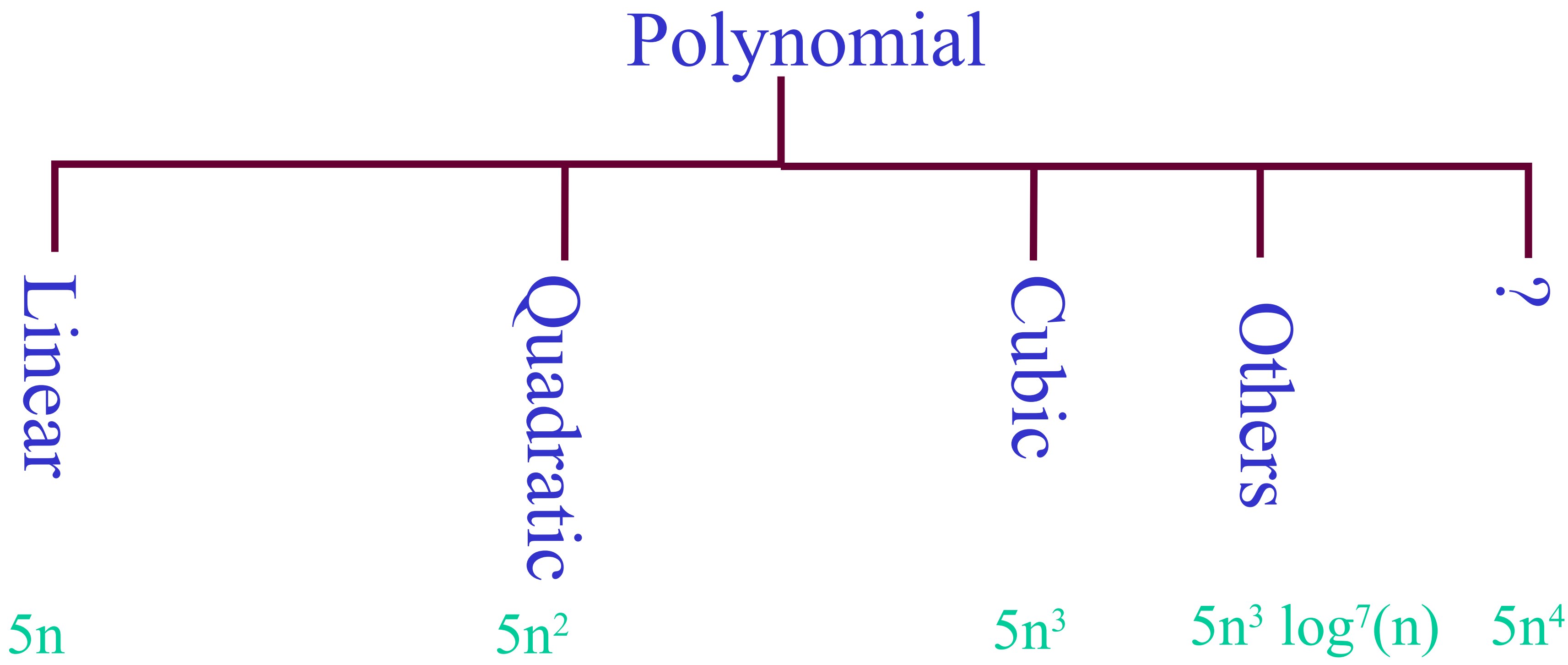
$T(n)$	10	100	1,000	10,000
$\log n$	3	6	9	13
$n^{1/2}$	3	10	31	100
n	10	100	1,000	10,000
$n \log n$	30	600	9,000	130,000
n^2	100	10,000	10^6	10^8
n^3	1,000	10^6	10^9	10^{12}
2^n	1,024	10^{30}	10^{300}	10^{3000}

Hierarchy of functions



Classifying Polynomials

Dominant term is of the form n^c



Logarithmic functions

- $\log_{10} n$ = # digits to write n
- $\log_2 n$ = # bits to write n
  $= 3.32 \log_{10} n$

Differ only by a multiplicative constant.

- $\log(n^{1000}) = 1000 \log(n)$

Poly Logarithmic (a.k.a. polylog)

$$(\log n)^5 = \log^5 n$$

Crucial asymptotic facts

Logarithmic << Polynomial

$\log^{1000} n \ll n^{0.001}$ For sufficiently large n

Linear << Quadratic

$10000\, n \ll 0.0001\, n^2$ For sufficiently large n

Polynomial << Exponential

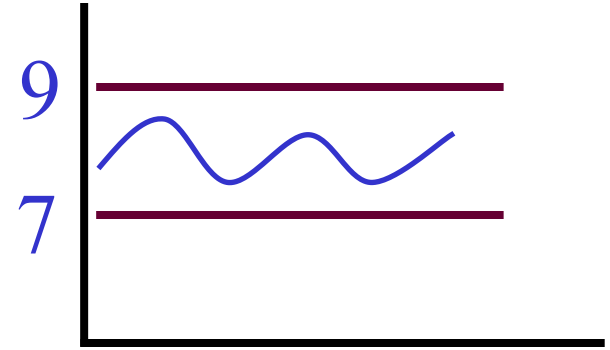
$n^{1000} \ll 2^{0.001\, n}$ For sufficiently large n

Are constant functions constant?

The running time of the algorithm is a "constant"
It does not depend **significantly**
on the size of the input.

Write $\theta(1)$.

- Yes • 5
- Yes • 1,000,000,000,000
- Yes • 0.000000000001
- No • -5
- No • 0
- Yes • 8 + sin(n)



Lie in between

Polynomial Functions

Quadratic

- n^2
- $0.001\, n^2$
- $1000\, n^2$
- $5n^2 + 3000n + 2\log n$

Lie in between

Polynomial

- n^c
- $n^{0.0001}$
- n^{10000}
- $5n^2 + 8n + 2\log n$
- $5n^2 \log n$
- $5n^{2.5}$

Lie in between

Exponential functions

- 2^n
- $2^{0.0001\,n}$
- $2^{10000\,n}$
- $8^n \quad = 2^{3n}$
- $2^n / n^{100} \quad > 2^{0.5n}$
- $2^n \cdot n^{100} \quad < 2^{2n}$

$$2^{0.5n} > n^{100}$$

$$2^n = 2^{0.5n} \cdot 2^{0.5n} > n^{100} \cdot 2^{0.5n}$$

$$2^n / n^{100} > 2^{0.5n}$$

Proving asymptotic expressions

Use definitions!

e.g. $f(n) = 3n^2 + 7n + 8 = \theta(n^2)$

$f(n) \in \Theta(g(n))$ if there exists constants c_1, c_2, and n_0, $s.t.$

$\mathbf{c_1\, g(n) \le f(n) \le c_2\, g(n)}$ for $n \ge n_0$

Here $g(n) = n^2$

One direction $(f(n) = \Omega(g(n))$ is easy

$c_1\, g(n) \le f(n)$ holds for $c_1 = 3$ and $n \ge 0$

The other direction $(f(n) = O(g(n))$ needs more care

$f(n) \le c_2\, g(n)$ holds for $c_2 = 18$ and $n \ge 1$ (CHECK!)

So $n_0 = 1$

Proving asymptotic expressions – contd.

Caveats!

1. constants c_1, c_2 MUST BE POSITIVE.
2. Could have chosen $c_2 = 3 + \varepsilon$ for any $\varepsilon > 0$. WHY?

-- because $7n + 8 \le \varepsilon n^2$ for $n \ge n_0$ for some sufficiently large n_0. Usually, the smaller the ε you choose, the harder it is to find n_0. So choosing a large ε is easier.

3. Order of quantifiers

$\exists c_1\ c_2\ \exists n_0\ \forall\ n \ge n_0,\ c_1 g(n) \le f(n) \le c_2 g(n)$

vs

$\exists n_0\ \forall\ n \ge n_0\ \exists c_1\ c_2,\ c_1 g(n) \le f(n) \le c_2 g(n)$

-- allows a different c_1 and c_2 for each n. Can choose $c_2 = 1/n$!! So we can "prove" $n^3 = \Theta(n^2)$.

Why polynomial vs exponential?

Philosophical/Mathematical reason – polynomials have different properties, grow much slower; mathematically natural distinction.

Practical reasons

1. almost every algorithm ever designed and every algorithm considered practical are very low degree polynomials with reasonable constants.
2. a large class of natural, practical problems seem to allow only exponential time algorithms. Most experts believe that there do not exist any polynomial time algorithms for any of these; i.e. P $\neq$ NP.