Analysis of Algorithms and Asymptotics

CS 4231, Fall 2012

Mihalis Yannakakis

Analysis of Algorithms

- Correctness:
  The algorithm *terminates* with the *correct* answer
- Performance
  – Mainly Running time (Time complexity)
  – Use of other resources (space, …)
- Experimental vs. analytical evaluation of algorithms
- Other issues: simplicity, extensibility, …

Time Complexity

- Running time depends on the input
- Parameterize by the size *n* of the input, and express complexity as function *T(n)*

Worst Case: maximum time over all inputs of size *n*

Average Case: expected time, assuming a probability distribution over inputs of size *n*

Analysis

Cost of each operation depends on machine

Simplification 1: machine-independent analysis: assume all operations unit cost →

can add the costs of the different steps

Asymptotic Analysis

Simplification 2: Look at *growth* of T(n) as n goes to infinity; focus on dominant term

- Example: $3n^2 + 7n + 10$

  Dominant term: $3n^2$

- Simplification 3: Look at the *rate (order)* of growth: suppress the constant coefficient
  - Example: Quadratic complexity $\Theta(n^2)$

Benefits of asymptotic analysis

- Machine independence – intrinsic complexity of algorithms
- Abstraction from details, concentrate on dominant factors
- A linear-time algorithm becomes faster than a quadratic algorithm eventually (for large enough n)

But .. caution:

- Eventually may be too late, if the constant of the linear-time algorithm that we ignored is huge,
  eg. $10^9 n > n^2 \quad \text{for} \quad n < 10^9$

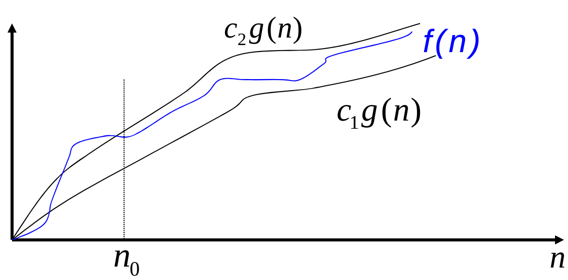
- Some operations may be much more costly than others, and we may want to count them separately (for example, comparisons in sorting of complex objects)

Asymptotic Notations: Theta, Big-Oh, Omega

Theta: $$\Theta(g(n)) = \{\ f(n) \mid \exists \text{ constants } c_1, c_2 > 0 \text{ and } n_0 \text{ s.t. } \forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)\ \}$$

Convention : We usually write $f(n) = \Theta(g(n))$

Caution: = here denotes membership, not equality

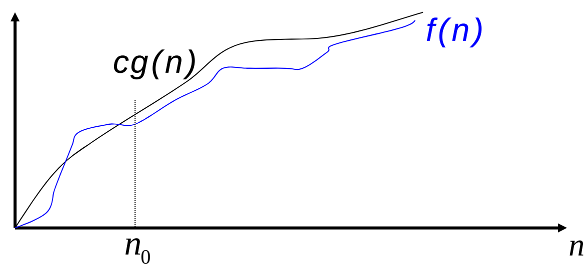


Asymptotic Notations: Theta, Big-Oh, Omega

Big-Oh: (Order)

$$O(g(n)) = \{\ f(n) \mid \exists \text{ constant } c > 0 \text{ and } n_0 \text{ s.t. } \forall n \geq n_0 : 0 \leq f(n) \leq c\,g(n)\ \}$$

Convention: We usually write $f(n) = O(g(n))$



Example:

$5n = O(n^2)$

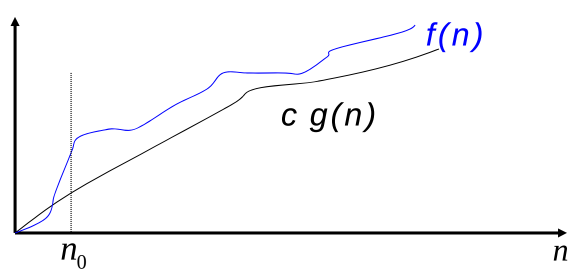
but not vice-versa

Asymptotic Notations: Theta, Big-Oh, Omega

Omega:

$$\Omega(g(n)) = \{\ f(n) \mid \exists \text{ constant } c > 0 \text{ and } n_0 \text{ s.t. } \forall n \geq n_0 : c\,g(n) \leq f(n)\ \}$$

Convention: We usually write $f(n) = \Omega(g(n))$



Example:

$5n^2 = \Omega(n)$

but not vice-versa

Asymptotic Notations: little-oh, little-omega

little-oh: $o(g(n)) = \{\ f(n) \mid \forall \text{ constant } c > 0\ \exists n_0 \text{ s.t. } \forall n \ge n_0 :$
$0 \le f(n) \le c\, g(n)\ \}$

little-omega: $\omega(g(n)) = \{\ f(n) \mid \forall \text{ constant } c > 0\ \exists n_0 \text{ s.t. } \forall n \ge n_0 :$
$0 \le c\, g(n) \le f(n)\ \}$

f(n)=o(g(n)) means that for large *n,* function *f* is smaller than any constant fraction of *g*

f(n)=ω*(g(n))* means that for large *n,* function *f* is larger than any constant multiple of *g,* i.e., *g=o(f(n))*

Example: $5n = o(n^2)$, $5n^2 = \omega(n)$

Asymptotic Notations Summary

Notation	Ratio *f(n)/g(n)* for large *n*
$f(n) = \omega(g(n))$	$f(n)/g(n) \to \infty$
$f(n) = \Omega(g(n))$	$c \le f(n)/g(n)$
$f(n) = \Theta(g(n))$	$c_1 \le f(n)/g(n) \le c_2$
$f(n) = O(g(n))$	$f(n)/g(n) \le c$
$f(n) = o(g(n))$	$f(n)/g(n) \to 0$

Example: Polynomials

- Polynomial: $a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0, \text{ where } a_d > 0$
  $= \Theta(n^d)$
  $\text{Ex}: 5n^3 + 4n^2 - 3n + 8 = \Theta(n^3)$

Proof: $\frac{f(n)}{n^d} = a_d + \frac{a_{d-1}}{n} + \cdots + \frac{a_0}{n^d} \to a_d + 0 + \cdots + 0 = a_d$

$(0 <) c < d \Leftrightarrow n^c = o(n^d)$
$\text{Ex}: n^{3.2} = o(n^{3.3})$

Proof: $\frac{n^c}{n^d} = \frac{1}{n^{d-c}} \to 0$

Example: logarithms

- $\log_{10} n = \Theta(\log_2 n)$
- Proof: $\log_{10} n = \log_2 n / \log_2 10 = \log_2 n / 3.32$
- Same for any change of logarithm from one constant base a to another base b: $\log_a n = \Theta(\log_b n)$
- Notation: logn for $\log_2 n$; ln n for $\log_e n$ (natural log)

Logs vs. powers/roots

- $\log n = o(n^c)$ for all $c>0$
- For example: $\log n = o(n^{0.4});\ \log n = o(\sqrt[20]{n})$
- Proof: Use L'Hopital's rule

$$\lim_{n\to\infty}\frac{\ln n}{n^c} = \lim_{n\to\infty}\frac{1/n}{cn^{c-1}} = \lim_{n\to\infty}\frac{1}{cn^c} = 0$$

Some common functions

$$\underbrace{n < n\log n < n^2 < n^3 < \cdots}_{\text{polynomial}} << \underbrace{2^n < 3^n < n!}_{\text{exponential}}$$

Properties

$$f(n) = o(g(n)) \;\Rightarrow\; f(n) = O(g(n))$$
$$f(n) = \omega(g(n)) \;\Rightarrow\; f(n) = \Omega(g(n))$$
$$f(n) = \Theta(g(n)) \;\Rightarrow\; f(n) = O(g(n)),\; f(n) = \Omega(g(n))$$
$$f(n) = \Theta(g(n)) \Leftarrow f(n) = O(g(n)),\; f(n) = \Omega(g(n))$$

Transitivity:

$f = O(g)$ and $g = O(h) \;\Rightarrow\; f = O(h)$

same for $o,\ \omega,\ \Omega,\ \Theta$

Sum: $f+g = \Theta(\max(f,g))$

Asymptotic notation in equations

$f(n) = 3n^2 + O(n)$ means

$f(n) = 3n^2 + h(n)$ for some function $h(n)$ that is $O(n)$

Can write equations like

$$3n^3 + O(n^2) + O(n) + O(1) = \Theta(n^3)$$

Caution: O(1)+O(1)+…+O(1) (n times) is not O(1)

O(n) + Ω(n) = ? It is not Θ(n)

…..