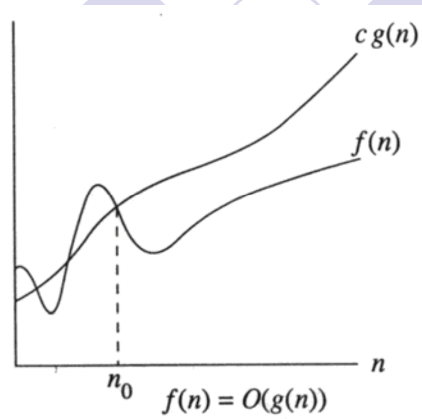
Algorithm Analysis (part 2)

CSE 2011

Fall 2009

Growth Rate



- The idea is to establish a relative order among functions for large n.
- $\exists$ c , $n_0 > 0$ such that $f(N) \le c\, g(N)$ when $N \ge n_0$
- f(N) grows no faster than g(N) for "large" N

Asymptotic Notation: Big-Oh

- $f(N) = O(g(N))$
- There are positive constants c and n_0 such that
  $f(N) \leq c.g(N)$ when $N \geq n_0$
- The growth rate of f(N) is *less than or equal to* the growth rate of g(N)
- g(N) is an upper bound on f(N)

3

Big-Oh: Examples

- Let $f(N) = 2N^2$. Then
  - $f(N) = O(N^4)$
  - $f(N) = O(N^3)$
  - $f(N) = O(N^2)$ (best answer, asymptotically tight)
- $O(N^2)$: reads "order N-squared" or "Big-Oh N-squared"

4

Big Oh: More Examples

- $N^2 / 2 – 3N = O(N^2)$
- $1 + 4N = O(N)$
- $7N^2 + 10N + 3 = O(N^2) = O(N^3)$
- $\log_{10} N = \log_2 N / \log_2 10 = O(\log_2 N) = O(\log N)$
- $\sin N = O(1)$; $10 = O(1)$, $10^{10} = O(1)$
- $\sum_{i=1}^{N} i \leq N \cdot N = O(N^2)$

  $\sum_{i=1}^{N} i^2 \leq N \cdot N^2 = O(N^3)$
- $\log N + N = O(N)$
- $\log^k N = O(N)$ for any constant k
- $N = O(2^N)$, but 2^N is not $O(N)$
- 2^{10N} is not $O(2^N)$

5

Math Review: Logarithmic Functions

$$x^a = b \quad iff \quad \log_x b = a$$

$$\log ab = \log a + \log b$$

$$\log_a b = \frac{\log_m b}{\log_m a}$$

$$\log a^b = b \log a$$

$$a^{\log n} = n^{\log a}$$

$$\log^b a = (\log a)^b \neq \log a^b$$

$$\frac{d \log_e x}{dx} = \frac{1}{x}$$

6

Some Rules

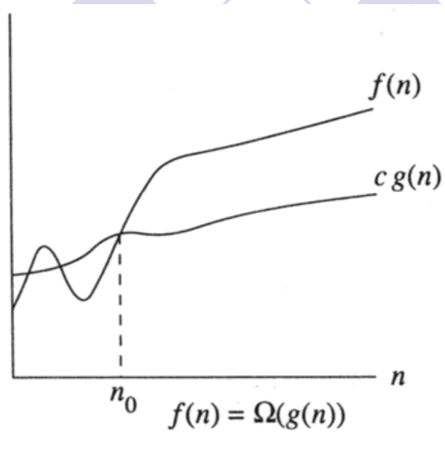
When considering the growth rate of a function using O()

- Ignore the lower order terms and the coefficients of the highest-order term
- No need to specify the base of logarithm
  - Changing the base from one constant to another changes the value of the logarithm by only a constant factor

- If $T_1(N) = O(f(N)$ and $T_2(N) = O(g(N))$, then
  - $T_1(N) + T_2(N) = O(f(N) + g(N))$
    (or less formally it is max (O(f(N)), O(g(N)))),
  - $T_1(N) * T_2(N) = O(f(N) * g(N))$

7

Big-Omega



- $\exists$ c , $n_0 > 0$ such that $f(N) \geq c\, g(N)$ when $N \geq n_0$
- f(N) grows no slower than g(N) for “large” N

8

Big-Omega

- $f(N) = \Omega(g(N))$

- There are positive constants c and n_0 such that

  $f(N) \geq c\, g(N)$ when $N \geq n_0$

- The growth rate of f(N) is *greater than or equal to* the growth rate of g(N).

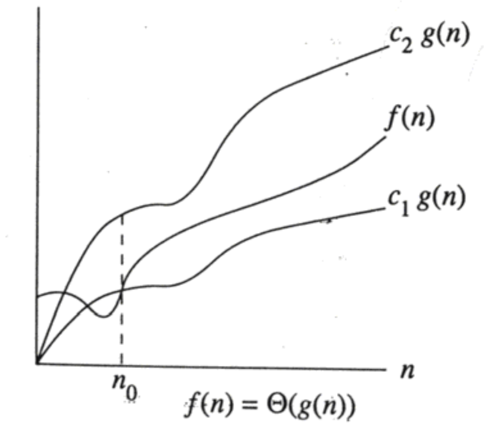
9

Big-Omega: Examples

- Let $f(N) = 2N^2$. Then
  - $f(N) = \Omega(N)$
  - $f(N) = \Omega(N^2)$ (best answer)

10

Big-Theta



- The growth rate of f(N) *is the same as* the growth rate of g(N)
- f(N) = Θ(g(N)) iff f(N) = O(g(N)) and f(N) = Ω(g(N))

11

Big-Theta: Example

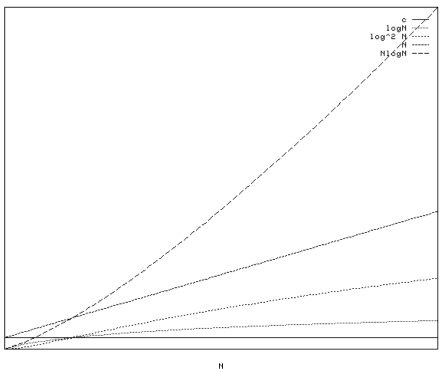
- Let f(N)=N^2 , g(N)=$2N^2$
  - Since f(N) = O(g(N)) and f(N) = Ω(g(N)), f(N) = Θ(g(N)).
- $c_1 = 1$, $n_1 = 0$
- $c_2 = ½$, $n_2 = 0$

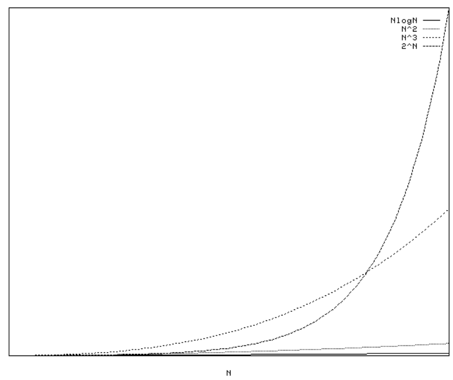
12

Typical Growth Rates

Function	Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Figure 2.1 Typical growth rates





Some More Rules

- If T(N) is a polynomial of degree k, then $T(N) = \Theta(N^k)$.
- For logarithmic functions, $T(\log_m N) = \Theta(\log N)$.
- $\log^k N = O(N)$ for any constant k (logarithms grow very slowly)

14

Small-oh

- $f(N) = \boldsymbol{o}(g(N))$

- $\forall c, \exists n_0$ such that $f(N) < c\, g(N)$ when $N > n_0$

- Less formally, $f(N) = \boldsymbol{o}(g(N))$
  if $f(N) = \boldsymbol{O}(g(N))$ and $f(N) \neq \Theta(g(N))$

- g(N) grows faster than f(N) for "large" N.

15

Small-oh: Example

- Let $f(N) = \frac{3}{4} N^2$ and $f(N) = \boldsymbol{o}(g(N))$
  - $g(N) = N^2$?
  - $g(N) = N^2 \log N$?
  - $g(N) = N^3$?

16

Determining Relative Growth Rates of Two Functions

- Using simple algebra (Figure 2.1)
- Example:
  - $f(N) = N \log N$
  - $g(N) = N^{1.5}$
- Using L'Hôpital's rule

17

Using L' Hôpital's Rule

- L' Hôpital's rule

If $\lim_{n\to\infty} f(N) = \infty$ and $\lim_{n\to\infty} g(N) = \infty$

then $\lim_{n\to\infty} \frac{f(N)}{g(N)}$ = $\lim_{n\to\infty} \frac{f'(N)}{g'(N)}$

- Determine the relative growth rates: compute $\lim_{n\to\infty} \frac{f(N)}{g(N)}$
  - if 0: f(N) = ***o***(g(N))
  - if constant ≠ 0: f(N) = Θ(g(N))
  - if ∞: g(N) = ***o***(f(N))
  - limit oscillates: no relation

18

Next time …

- Recursion

19