Algoritma & Pemrograman Saintifik

Gatot F. Hertono, Ph.D

Departemen Matematika

SCMA601401

Asymptotic Notation: Order of Growth

Background

Suppose, in worst case, a problem can be solved by using two different algorithms, with time complexity:

Algorithm A: $f(n) = 400n + 23$

Algorithm B: $g(n) = 2n^2 - 1$

Which one is better?

n	$3n^2$	14n+17
1	3	31
10	300	157
100	30,000	1,417
1000	3,000,000	14,017
10000	300,000,000	140,017

$3n^2 > 14n+17$

$\forall$ "large enough" n

Solution: Ignore the constants & low-order terms.

→ we use Asymptotic Notation (Ω, Θ dan O)

Order of Growth

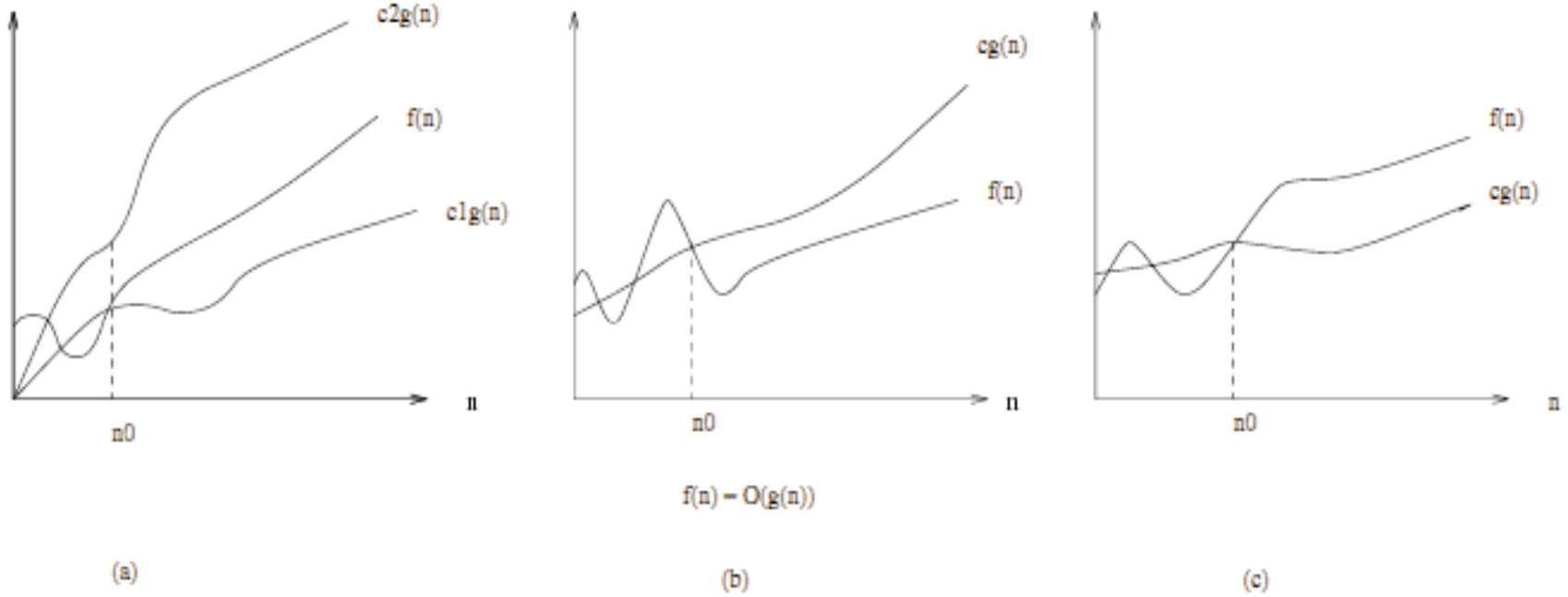
n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Values (some approximate) of several functions important for analysis of algorithms

Ω, Θ and O notations

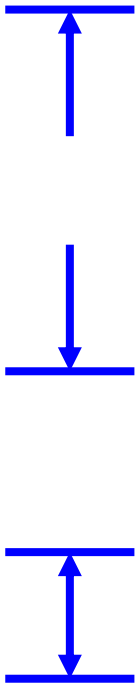
What does it mean?

O, Ω, and Θ

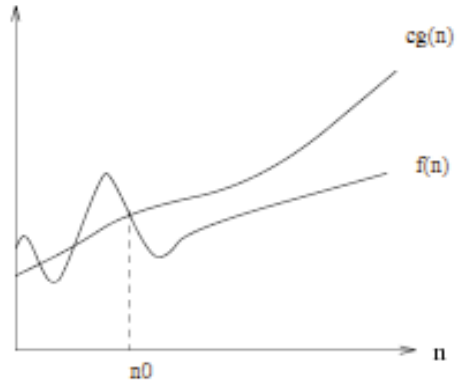


Names of Bounding Functions

- $f(n) = O(g(n))$ means C x $g(n)$ is an upper bound on $f(n)$;
- $f(n) = \Omega(g(n))$ means C x $g(n)$ is a lower bound on $f(n)$;
- $f(n) = \Theta(g(n))$ means C_1 x $g(n)$ is an upper bound on $f(n)$ and C_2 x $g(n)$ is a lower bound on $f(n)$.



O (big *Oh*) Notation



$f(n) = O(g(n))$: $g(n)$ is an *asymptotically upper bound* for $f(n)$. *i.e. f* does not grow faster than *g*.

Formal definition:

$$O(g(n)) = \{f(n) : \exists\ c > 0 \text{ and } n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}.$$

Remark: "$f(n) = O(g(n))$" means that

$$f(n) \in O(g(n)).$$

Examples:

(1) $n^2 + 2n = O(n^2)$.
(2) $200n^2 - 100n = O(n^2)$.
(3) $n \log_2 n = O(n^2)$.
(4) $n^2 \log_2 n \neq O(n^2)$.
(5) $\forall a, b > 1,\ \log_a n = O(\log_b n)$.

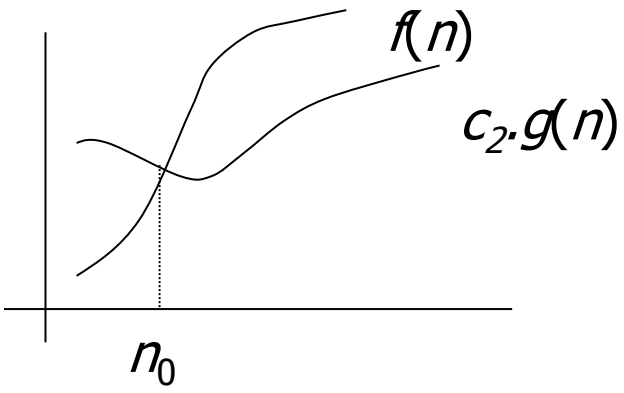
Example: $f(n) = n^3 + 20\,n^2 + 100.n$, then $f(n) = O(n^3)$.

Proof:

$\forall\ n \ge 0,\quad n^3 + 20\,n^2 + 100.n \le n^3 + 20\,n^3 + 100.n^3 = 121.n^3$.

Choose $c = 121$ and $n_0 = 0$, then it completes the definition.

Ω (big Omega) Notation



$f(n) = \Omega(g(n))$: $g(n)$ is an *asymptotically lower bound* for $f(n)$.

Formal definition:

$$\Omega(g(n)) = \{f(n) : \exists\ c > 0 \text{ and } n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$

Examples:

(1) $200n^2 - 100n = \Omega(n^2) = \Omega(n) = \Omega(1)$.

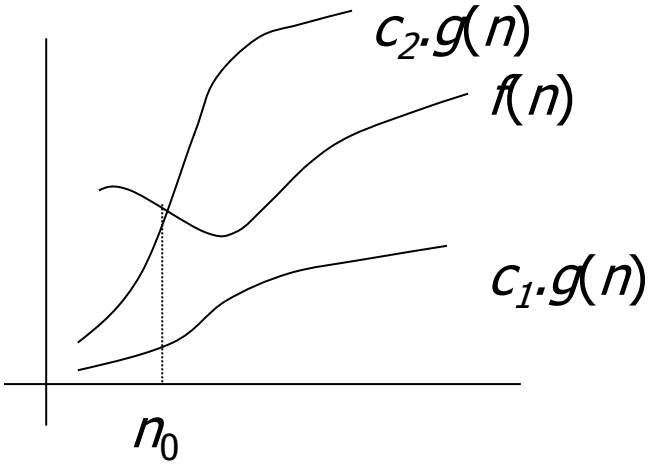
(2) $n^2 = \Omega(n)$.

(3) $n^2 \neq \Omega(n^2 \log n)$.

(3) Does $f(n) = O(g(n))$ imply $g(n) = \Omega(f(n))$?

(4) Does $f(n) = \Omega(g(n))$ imply $g(n) = O(f(n))$?

Θ (Big Theta) Notation



$f(n) = \Theta(g(n))$: $g(n)$ is an *asymptotically tight bound* for $f(n)$.

Formal definition:

$\Theta(g(n)) = \{f(n) : \exists\ n_0 > 0,\ c_1 > 0$ and $c_2 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.

Examples:

(1) $5n^2 - 2n + 5 = \Theta(n^2)$

(2) $5n^2 - 2n + 5 = \Theta(n^2 + \log n)$

Examples of Θ

$3n^2 + 7n + 8 = \Theta(n^2)$?

True

$$\exists c_1, c_2, \exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$n \geq 8$

$3 \quad 4 \quad 8$

$3 \cdot n^2 \leq 3n^2 + 7n + 8 \leq 4 \cdot n^2$

$n^2 = \Theta(n^3)$?

$$\exists c_1, c_2, \exists n_0, \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

0

$0 \cdot n^3 \leq n^2 \leq c_2 \cdot n^3$

False, since C1,C2 must be >0

Examples

$$3n^2 - 100n + 6 = O(n^2) \text{ because } 3n^2 > 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 = O(n^3) \text{ because } .01n^3 > 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 \neq O(n) \text{ because } c \cdot n < 3n^2 \text{ when } n > c$$

$$3n^2 - 100n + 6 = \Omega(n^2) \text{ because } 2.99n^2 < 3n^2 - 100n + 6$$
$$3n^2 - 100n + 6 \neq \Omega(n^3) \text{ because } 3n^2 - 100n + 6 < n^3$$
$$3n^2 - 100n + 6 = \Omega(n) \text{ because } 10^{10^{10}} n < 3n^2 - 100 + 6$$

$$3n^2 - 100n + 6 = \Theta(n^2) \text{ because } O \text{ and } \Omega$$
$$3n^2 - 100n + 6 \neq \Theta(n^3) \text{ because } O \text{ only}$$
$$3n^2 - 100n + 6 \neq \Theta(n) \text{ because } \Omega \text{ only}$$

Think of the equality as meaning *in the set of functions.*

Properties

Note that if

$$f(n) = \Theta(g(n)),$$

then

$$f(n) = \Omega(g(n)) \quad \text{and} \quad f(n) = O(g(n)).$$

In the other direction, if

$$f(n) = \Omega(g(n)) \quad \text{and} \quad f(n) = O(g(n)),$$

then

$$f(n) = \Theta(g(n)).$$

Transitivity.

- If f = O(g) and g = O(h) then f = O(h).
- If f = Ω(g) and g = Ω(h) then f = Ω(h).
- If f = Θ(g) and g = Θ(h) then f = Θ(h).

Additivity.

- If f = O(h) and g = O(h) then f + g = O(h).
- If f = Ω(h) and g = Ω(h) then f + g = Ω(h).
- If f = Θ(h) and g = O(h) then f + g = Θ(h).

Assignment : Based on the definition of Ω, Θ and O, prove that

$$f(n) = \Theta(g(n)) \quad \Leftrightarrow \quad f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

What does asymptotic property imply for an algorithm?

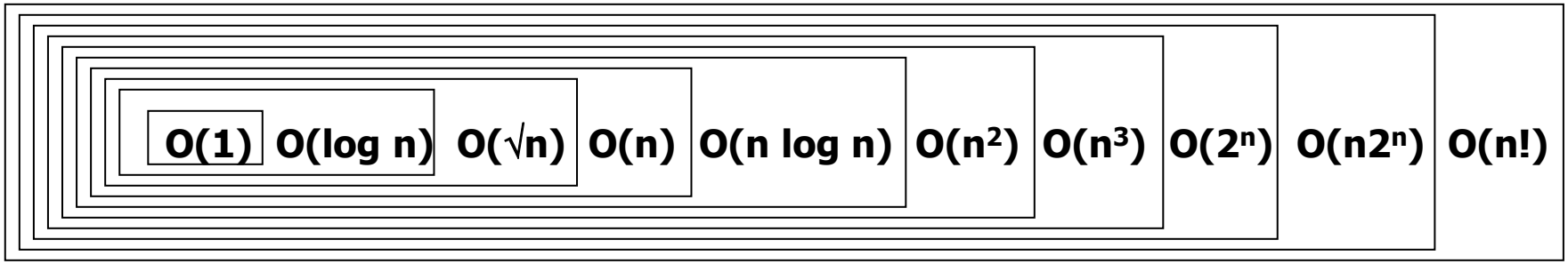
$t_1(n) \in O(g_1(n))$

$t_2(n) \in O(g_2(n))$

How is the overall efficiency of this algorithm?

Basic asymptotic efficiency classes

1	constant
log n	logarithmic
n	linear
n log n	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
n!	factorial



Time efficiency of nonrecursive algorithms

General Plan for Analysis

- Decide on parameter *n* indicating *input size*
- Identify algorithm's *basic operation*
- Determine *worst*, *average*, and *best* cases for input of size *n*
- Set up a sum for the number of times the basic operation is executed
- Simplify the sum using standard formulas and rules

Example 1: Maximum element

ALGORITHM *MaxElement*$(A[0..n-1])$

```
//Determines the value of the largest element in a given array
//Input: An array A[0..n − 1] of real numbers
//Output: The value of the largest element in A
maxval ← A[0]
for i ← 1 to n − 1 do
    if A[i] > maxval
        maxval ← A[i]
return maxval
```

$T(n) = \boldsymbol{O(?)}$

Example 2: Element uniqueness problem

ALGORITHM *UniqueElements*$(A[0..n-1])$

```
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n − 1]
//Output: Returns "true" if all the elements in A are distinct
//          and "false" otherwise
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

$T(n) = O(?)$

Example 3: Matrix multiplication

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

```
//Multiplies two n-by-n matrices by the definition-based algorithm
//Input: Two n-by-n matrices A and B
//Output: Matrix C = AB
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        C[i, j] ← 0.0
        for k ← 0 to n − 1 do
            C[i, j] ← C[i, j] + A[i, k] * B[k, j]
return C
```

$T(n) = O(?)$

Example 5: Counting binary digits

ALGORITHM *Binary*(n)

```
//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
count ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

It cannot be investigated the way the previous examples are.