Design and Analysis of Algorithms

Vandana M L
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Asymptotic Notations

Slides courtesy of **Anany Levitin**

Vandana M L

Department of Computer Science & Engineering

Orders of growth of an algorithm's basic operation count is important

How do we compare order of growth??

Using Asymptotic Notations

O(g(n)): class of functions f(n) that grow *no faster* than g(n)
Ω(g(n)): class of functions f(n) that grow *at least as fast* as g(n)
Θ (g(n)): class of functions f(n) that grow *at same rate* as g(n)
o(g(n)): class of functions f(n) that grow *at slower rate* than g(n)
ω(g(n)): class of functions f(n) that grow *at faster rate* than g(n)

O-notation

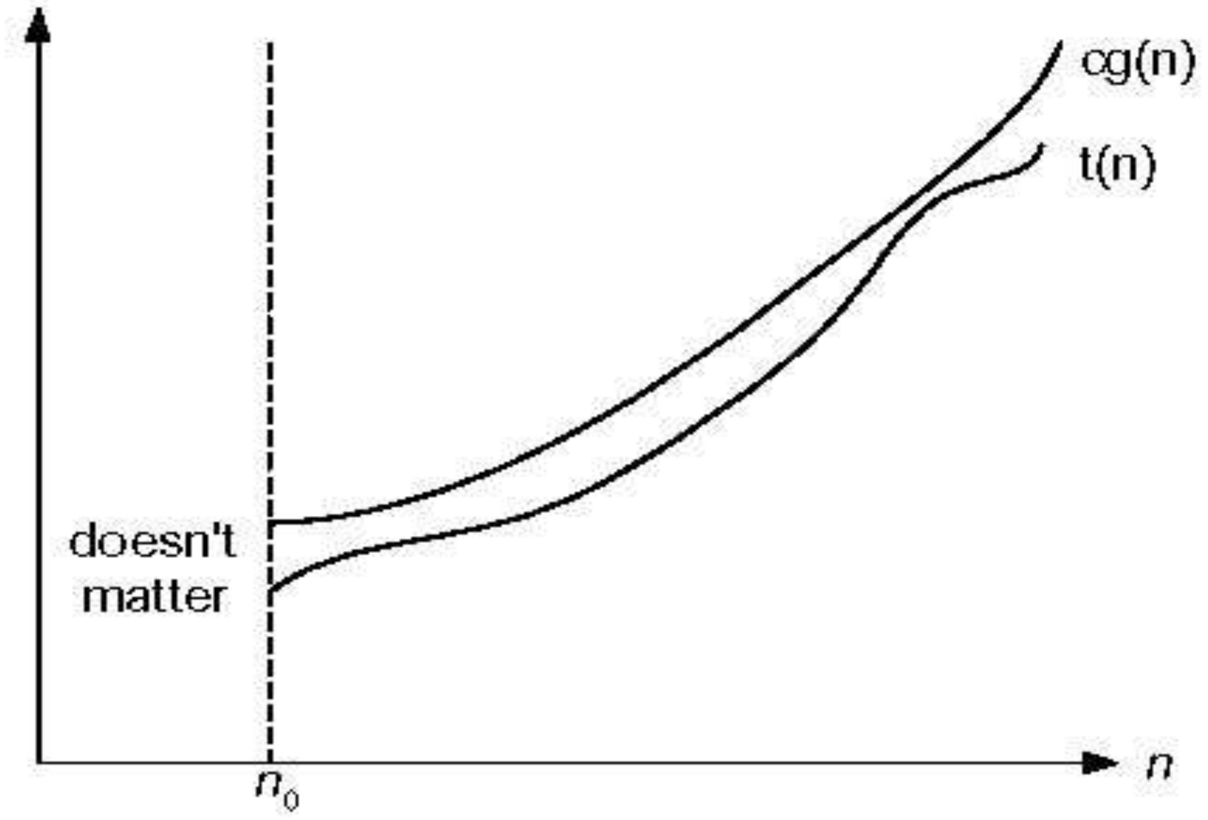


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

O-notation

Formal definition

A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n,

i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \leq cg(n) \text{ for all } n \geq n_0$$

Exercises: prove the following using the above definition

$10n^2 \in O(n^2)$

$10n^2 + 2n \in O(n^2)$

$100n + 5 \in O(n^2)$

$5n+20 \in O(n)$

Ω-notation

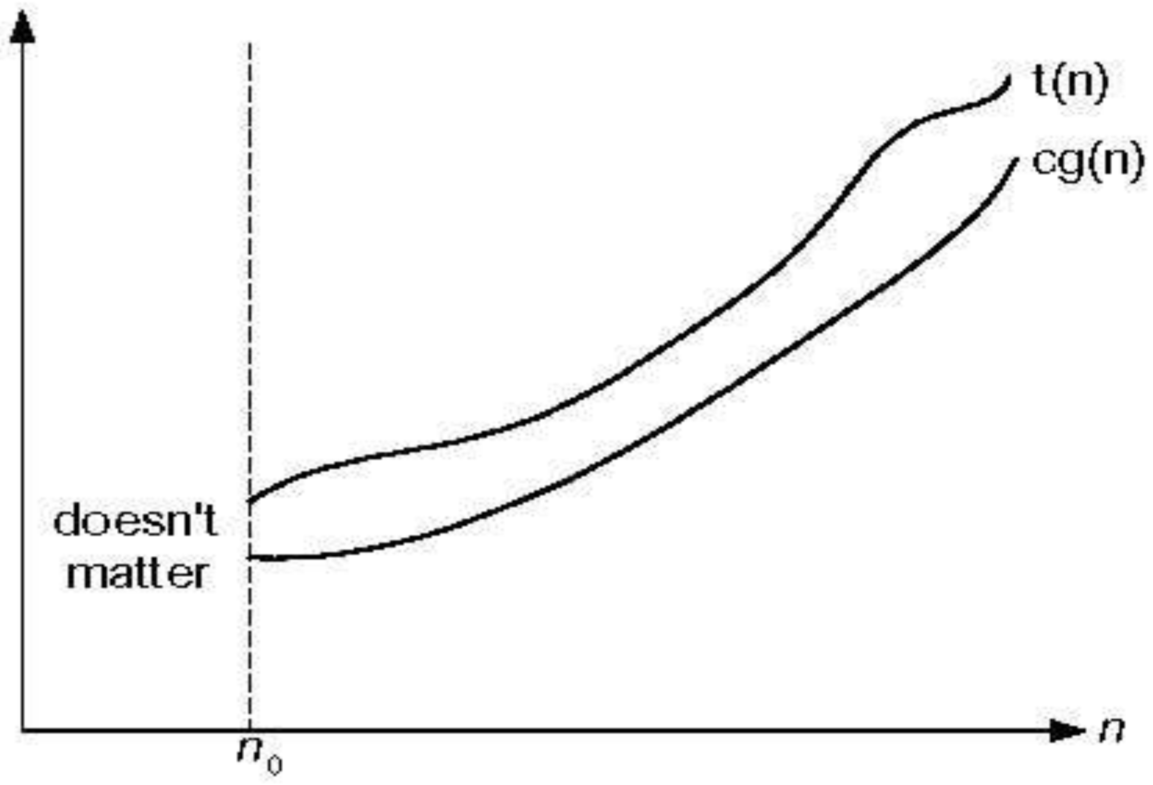


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

Ω-notation

Formal definition

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some constant multiple of $g(n)$ for all large n,

i.e., <u>if there exist some positive constant c and some nonnegative integer n_0 such that</u>

$$t(n) \geq cg(n) \text{ for all } n \geq n_0$$

Exercises: prove the following using the above definition

$10n^2 \in \Omega(n^2)$

$10n^2 + 2n \in \Omega(n^2)$

$10n^3 \in \Omega(n^2)$

Θ-notation

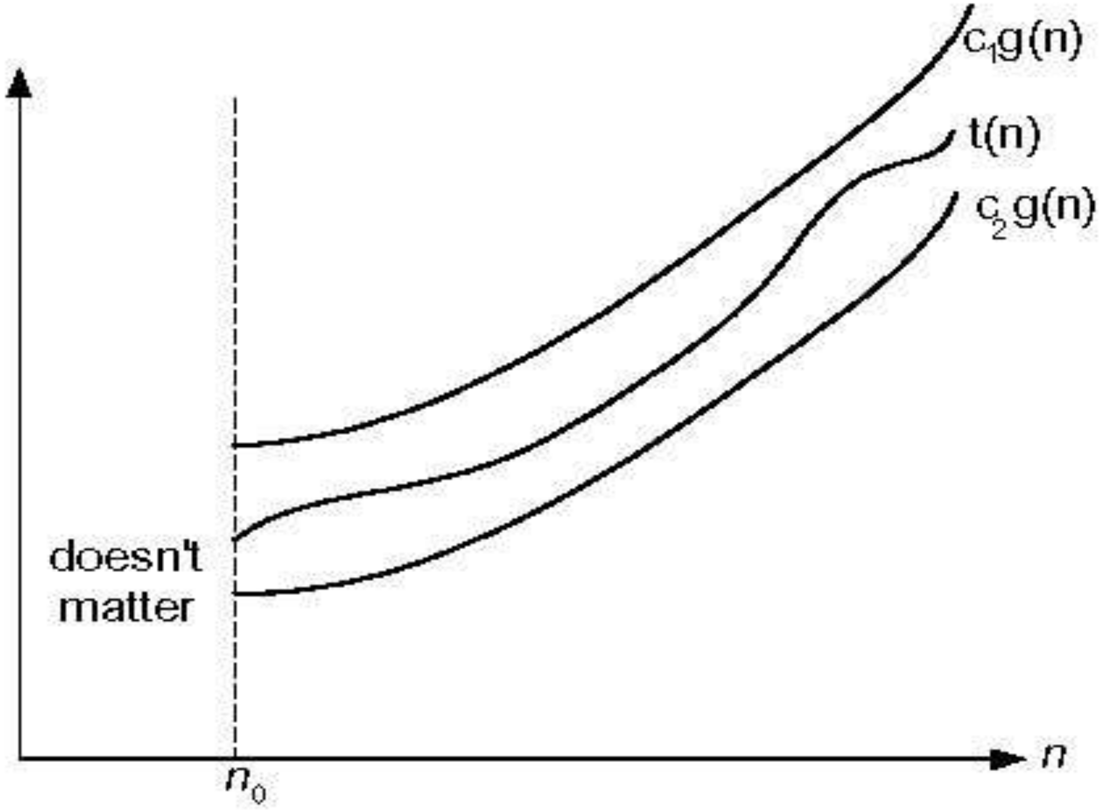


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Θ-notation

Formal definition

A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n,

i.e., <u>if there exist some positive constant c_1 and c_2 and some nonnegative integer n_0 such that</u>

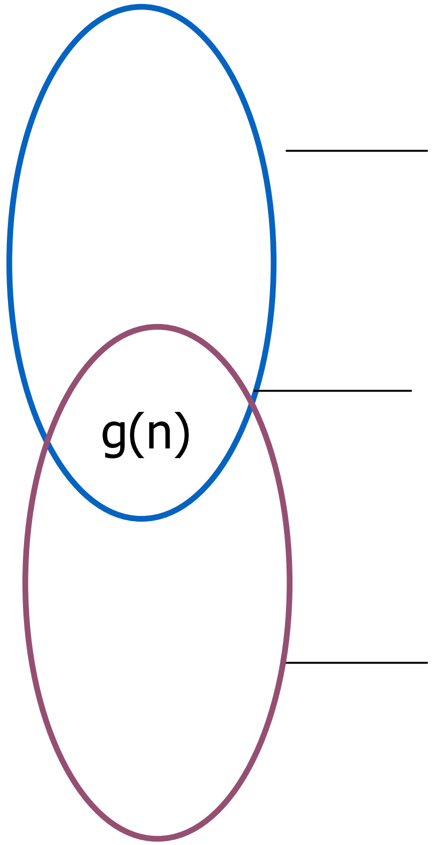
$$c_2\, g(n) \le t(n) \le c_1\, g(n) \text{ for all } n \ge n_0$$

Exercises: prove the following using the above definition

$10n^2 \in \Theta(n^2)$

$10n^2 + 2n \in \Theta(n^2)$

$(1/2)n(n-1) \in \Theta(n^2)$



>=

$\Omega(g(n))$, functions that grow at least as fast as $g(n)$

=

$\Theta(g(n))$, functions that grow at the same rate as $g(n)$

<=

$O(g(n))$, functions that grow no faster than $g(n)$

Little-o Notation

Formal Definition:

A function *t(n)* is said to be in Little-o*(g(n)),* denoted $t(n) \in o(g(n))$,
iif there exist some positive constant c and some nonnegative integer n_0 such that

$0 \leq t(n) < cg(n)$ for all $n \geq n_0$

Little-o Notation

Example: $f(n) = 2n^2$ and $g(n) = n^2$ and $c = 2$

$f(n) = O(g(n))$ - Big-O

$f(n) \neq o(g(n))$ - little-o

If $f(n) = 2n$ & $g(n) = n^2$, and $c = 3$,

then $f(n) = o(n^2)$

Note : For non-negative functions, f(n) and g(n),
f(n) is little o of g(n), if and only if,

$f(n) = O(g(n))$ and $f(n) \neq \theta(g(n))$

[strict upper bound, no lower bound]

Little Omega Notation

Formal Definition:

A function *t(n)* is said to be in Little- $\omega(g(n))$, denoted $t(n) \in \omega(g(n))$, if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) > cg(n) \geq 0 \text{ for all } n \geq n_0$$

Little Omega Notation

Example : If $f(n) = 3n + 2$, $g(n) = n$ and $c = 3$

$f(n) = \omega(n)$

Note : For non-negative functions, f(n) *and g*(n),

f(n) is little ω of g(n), if and only if

$f(n) = \Omega(g(n))$ and $f(n) \neq \theta(g(n))$

[strict lower bound, no upper bound]

Theorems

- If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

  For example,

  $$5n^2 + 3n\log n \in O(n^2)$$

- If $t1(n) \in \Theta(g1(n))$ and $t2(n) \in \Theta(g2(n))$, then $t1(n) + t2(n) \in \Theta(\max\{g1(n), g2(n)\})$

- $t1(n) \in \Omega(g1(n))$ and $t2(n) \in \Omega(g2(n))$, then $t1(n) + t2(n) \in \Omega(\max\{g1(n), g2(n)\})$

Implication: The algorithm's overall efficiency will be determined by the part with a larger order of growth, I.e., its least efficient part.

THANK YOU

Vandana M L
Department of Computer Science & Engineering

vandanamd@pes.edu