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

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

How do we compare order of growth??

Using Asymptotic Notations

A way of comparing functions that ignores constant factors and small input sizes

- $O(g(n))$: class of functions $f(n)$ that grow *no faster* than $g(n)$
- $\Omega(g(n))$: class of functions $f(n)$ that grow *at least as fast* as $g(n)$
- $\Theta(g(n))$: 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 $g(n)$
- $\omega(g(n))$: class of functions $f(n)$ that grow *at faster rate* than $g(n)$

O-notation

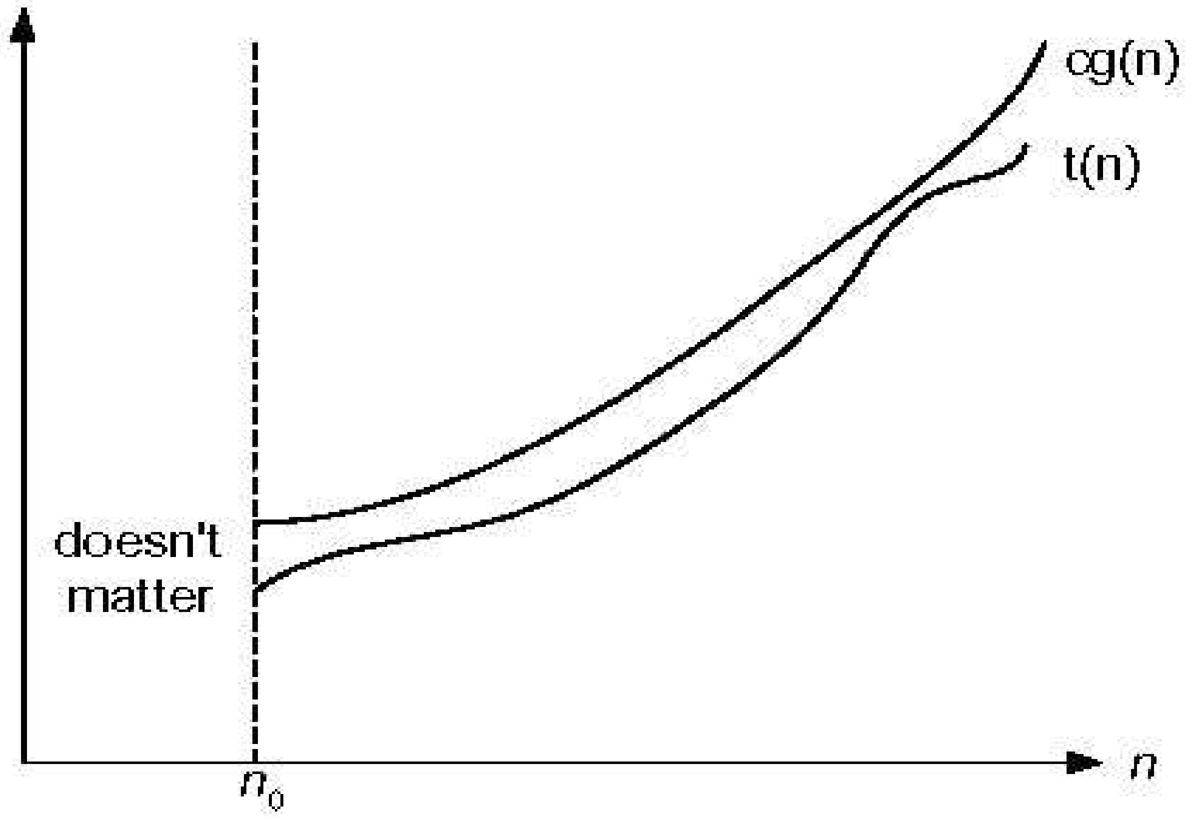


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

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

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

Example: $100n+5 \in O(n)$

Ω-notation

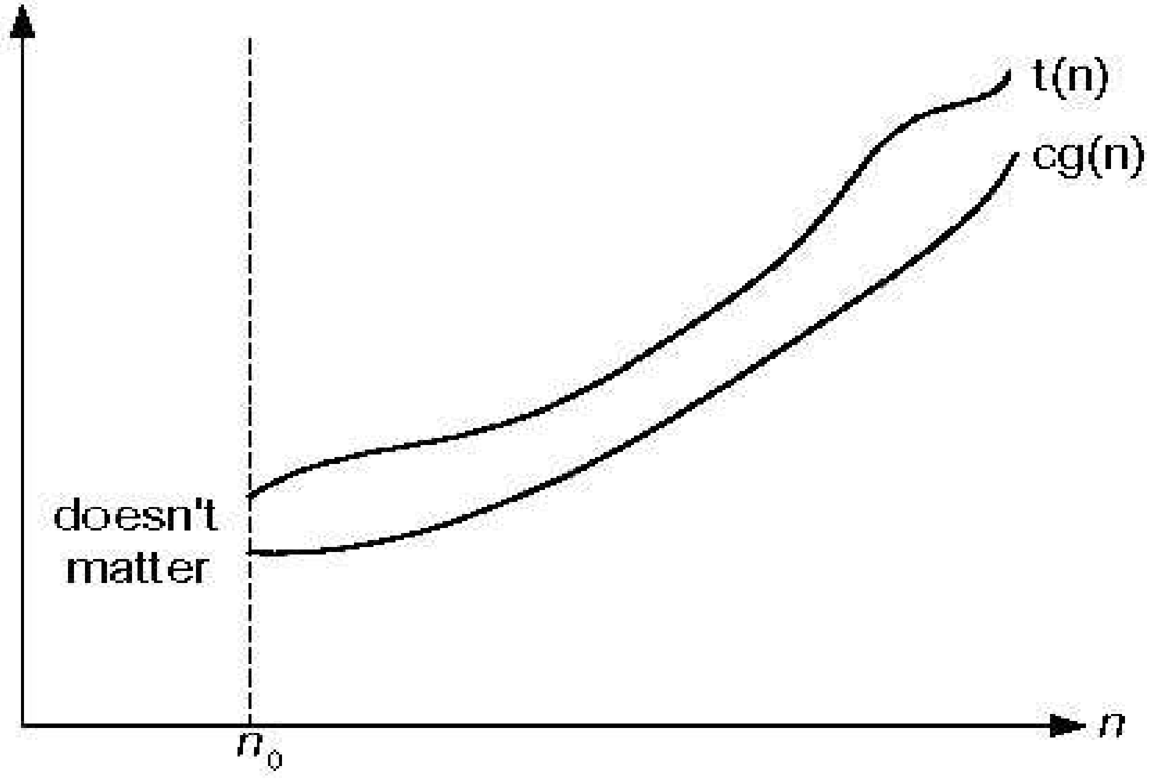


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

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

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

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

Θ-notation

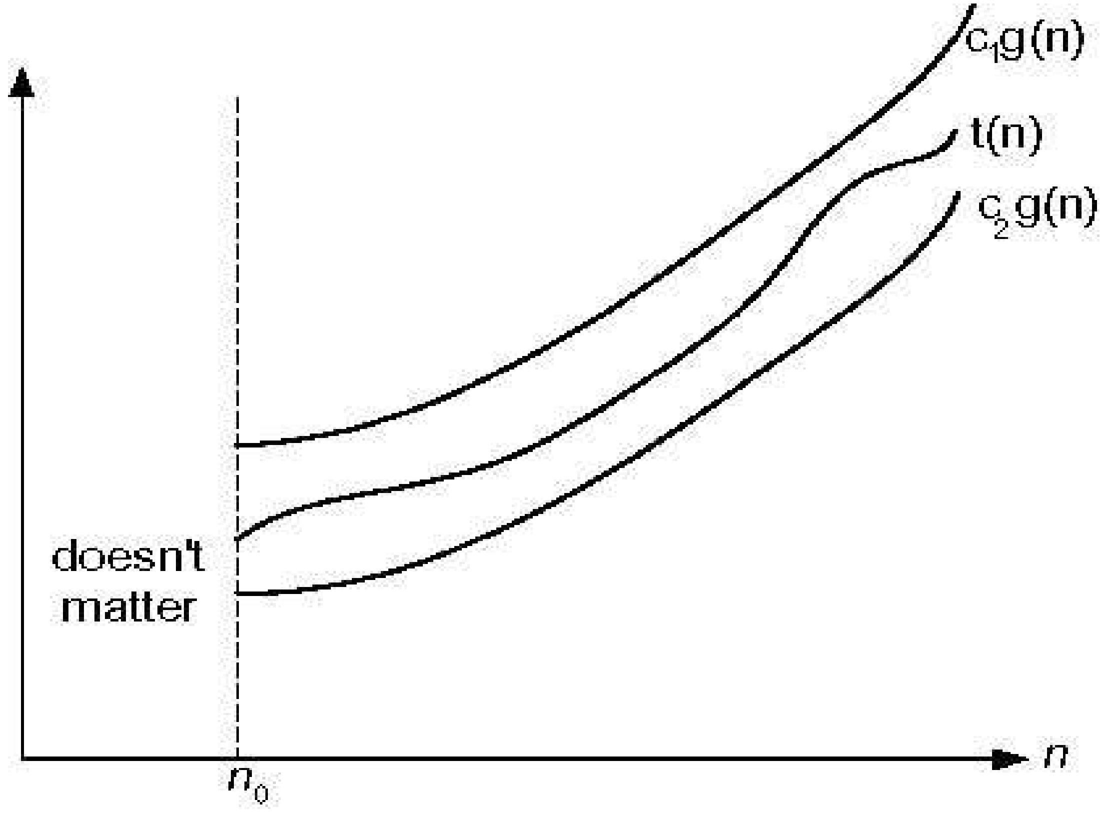


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Θ-notation

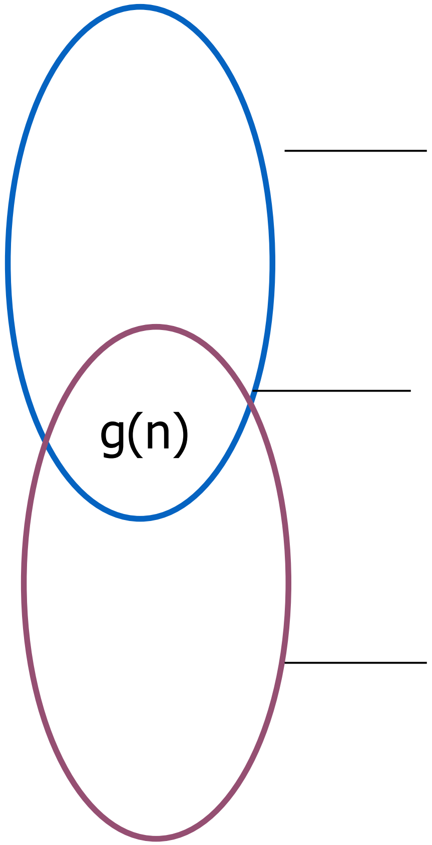
Formal definition

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., if there exist some positive constants c_1 and c_2 and some nonnegative integer n_0 such that

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

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



$>=$

$\Omega(g(n))$, 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 $t(n) \in o(g(n))$, if for any positive constant c and some nonnegative integer n_0

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

Example: $n \in o(n^2)$

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 for any positive constant c and some nonnegative integer n_0

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

Example: $3n^2 + 2 \in \omega(n)$

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