Growth of Functions

謝仁偉 副教授
jenwei@mail.ntust.edu.tw
國立台灣科技大學 資訊工程系
2017 Spring

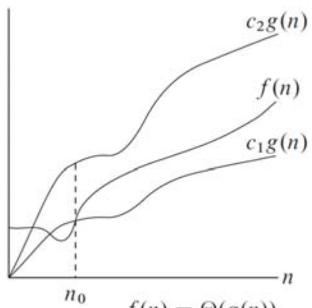
Introduction

- The order of growth of the running time of an algorithm gives a simple characterization of the algorithm's efficiency and also allows us to compare the relative performance of alternative algorithms.
- We are concerned with how the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.
  - Usually, an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs.

Θ-notation (1/2)

- For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}.$$



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

Θ-notation (2/2)

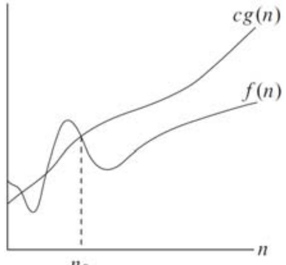
- Because $\Theta(g(n))$ is a set, we could write "$f(n) \in \Theta(g(n))$" to indicate that $f(n)$ is a member of $\Theta(g(n))$.
- We will usually write "$f(n) = \Theta(g(n))$" to express the same notion.
- For all $n \ge n_0$, the function $f(n)$ is equal to $g(n)$ to within a constant factor.

➡ We say that $g(n)$ is an asymptotically tight bound for $f(n)$.

O-notation (1/2)

- When we have only an asymptotic upper bound, we use O-notation.
- For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

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



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

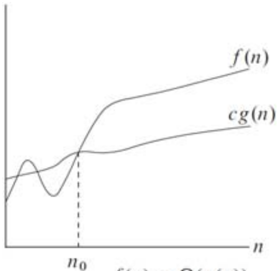
O-notation (2/2)

- Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$, since Θ-notation is a stronger notion than O-notation.

Ω-notation (1/2)

- Ω-notation provides an asymptotic lower bound on a function.
- For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \} .$$



$f(n) = \Omega(g(n))$

Ω-notation (2/2)

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. ■

- When we say that the running time of an algorithm is $\Omega(g(n))$, we mean that no matter what particular input of size n is chosen for each value of n, the running time on that input is at least a constant times $g(n)$, for sufficiently large n.

o-notation (1/2)

- The asymptotic upper bound provided by O-notation may or may not be asymptotically tight.
  - The bound $2n^2 = O(n^2)$ is asymptotically tight, but the bound $2n = O(n^2)$ is not.
- We use o-notation to denote an upper bound that is not asymptotically tight.

$$o(g(n)) = \{ f(n) : \text{ for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le f(n) < cg(n) \text{ for all } n \ge n_0 \} .$$

  - For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$

o-notation (2/2)

- In $f(n) = O(g(n))$, the bound $0 \le f(n) \le cg(n)$ holds for some constant $c > 0$, but in $f(n) = o(g(n))$, the bound $0 \le f(n) < cg(n)$ holds for all constants $c > 0$.
- In o-notation, the function $f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity; that is $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 .$

ω-notation

- We use ω-notation to denote a lower bound that is not asymptotically tight.
- One way to define it is by $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$.
- Formally, we define $\omega(g(n))$ as the set

$$\omega(g(n)) = \{ f(n) : \text{ for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le cg(n) < f(n) \text{ for all } n \ge n_0 \} .$$

- The relation $f(n) = \omega(g(n))$ implies that $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, if the limit exists.

Comparing Functions (1/2)

- Transitivity:

$$\begin{aligned}
f(n) &= \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \text{ imply } f(n) = \Theta(h(n)) , \\
f(n) &= O(g(n)) \text{ and } g(n) = O(h(n)) \text{ imply } f(n) = O(h(n)) , \\
f(n) &= \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \text{ imply } f(n) = \Omega(h(n)) , \\
f(n) &= o(g(n)) \text{ and } g(n) = o(h(n)) \text{ imply } f(n) = o(h(n)) , \\
f(n) &= \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \text{ imply } f(n) = \omega(h(n)) .
\end{aligned}$$

- Reflexivity:

$$\begin{aligned}
f(n) &= \Theta(f(n)) , \\
f(n) &= O(f(n)) , \\
f(n) &= \Omega(f(n)) .
\end{aligned}$$

Comparing Functions (2/2)

- Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$

- Transpose symmetry:

$$f(n) = O(g(n)) \quad \text{if and only if} \quad g(n) = \Omega(f(n)) ,$$
$$f(n) = o(g(n)) \quad \text{if and only if} \quad g(n) = \omega(f(n)) .$$