渐近记号

Θ "同阶" 增长速度一样 (=)
O "低阶" 不比后者大 (≤)
Ω "高阶" 不比后者小 (≥)
o "低阶" 比后者小 (<)
ω "高阶" 比后者大 (>)

定义1.1 若存在两个正的常数 c 和 n_0，对于任意 $n \geq n_0$ 都有 $T(n) \leq c \times f(n)$，则称 $T(n) = O(f(n))$.

定义1.2 若存在两个正的常数 c 和 n_0，对于任意 $n \geq n_0$，都有 $T(n) \geq c \times g(n)$，则称 $T(n) = \Omega(g(n))$

定义1.3 若存在三个正的常数 c_1，c_2 和 n_0，对于任意 $n \geq n_0$，都有 $c_1 \times f(n) \geq T(n) \geq c_2 \times f(n)$，则称 $T(n) = \Theta(f(n))$.

① 大O符号用于描述增长率的上限，当输入规模为 n 时，算法消耗时间的最大值，这个上限的阶越低结果越有价值.

!应该注意的是，定义1.1给了很大的自由度来选择常量 c 和 n_0 的特定值，例如，下列的推导是合理的：

$100n+5 \leq 100n+n$ (当 $n \geq 5$) $= 101n = O(n)$ ($c=101, n_0=5$)

$100n+5 \leq 100n+5n$ (当 $n \geq 1$) $= 105n = O(n)$ ($c=105, n_0=1$)

KOKUYO

不爱学习的 Shirley
博客：https://www.cnblogs.com/shirleyya/

② 大Ω notation用来描述增长率的下限，当输入规模为n时，算法消耗的时间最小值。这个下限的阶越高，说明算法越有效率。

- 大Ω notation常用来分析某个问题或某类算法的时间下限。

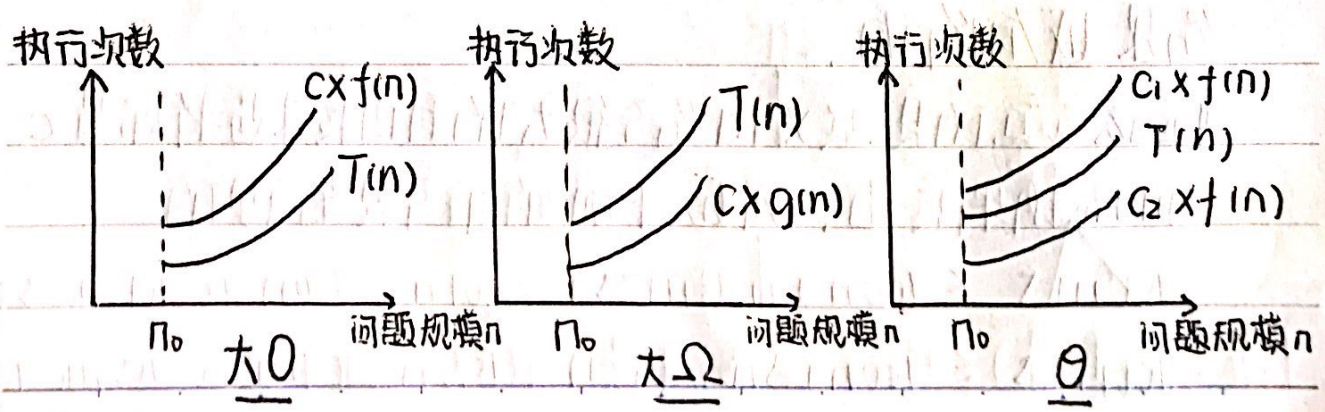
例：矩阵乘法问题的时间下界为 $\Omega(n^2)$。

基于比较的排序算法的时间下界为 $\Omega(n\log_2 n)$

- 大Ω notation常常大O notation配合以证明某一问题的一个特定算法是该问题的最优算法，或是该问题中的某算法类中的最优算法。

③ Θ notation意味着 $T(n)$ 与 $f(n)$ 同阶，用来表示算法的精确阶。

定理 1.1 若 $T(n)=a_m n^m + a_{m-1}n^{m-1} + \cdots + a_1 n + a_0$ $(a_m>0)$ 则有 $T(n)=O(n^m)$，且 $T(n)=\Omega(n^m)$，因此有 $T(n)=\Theta(n^m)$



不爱学习的 Shirley

博客：https://www.cnblogs.com/shirleyya/

定理：对任意两个函数 $f(n)$ 和 $g(n)$，我们有 $f(n)=\Theta(g(n))$ 当且仅当 $f(n)=O(g(n))$ 且 $f(n)=\Omega(g(n))$

但通常来说，都是用定义从渐近上界和下界来证明渐近确界。 $[O(g(n)) \wedge \Omega(g(n)) \Rightarrow \Theta(g(n))]$

英原版 P48 第二段（Since O-notation … $\Theta(n)$ time）

对于某一算法的最坏情况运行时间

- 界 $O(n^2)$ 对每一输入都适用 （n^2 为上界）
- 界 $\Theta(n^2)$ 并不是对每一输入都适用 （与 n^2 同界）
  （如排序，对已排好序的输入，应为 $\Theta(n)$）
- 界 $\Omega(n^2)$ 对每一输入都适用 （n^2 为下界）。

注 n^2 为示例，可为 $g(n)$ …