算法设计与分析

Computer Algorithm Design & Analysis

2019.10

王多强

dqwang@mail.hust.edu.cn

群名称：2019-算法

群　号：835135560

群名称：2019-算法

群　号：835135560

Chapter 3

Growth of Functions

函数的增长

本章研究算法的渐近效率，给出算法运行时间随问题规模的变化关系，给出时间/空间复杂度限界函数的定义，引入渐近记号。

3.1 限界函数的定义

算法（时间）复杂度的限界函数常用的有三个：

上界函数、下界函数、渐近紧确界函数。

对应的渐近记号： O Ω Θ

定义如下：

记：算法的实际执行时间为***f(n)***，分析所得的限界函数为***g(n)***

其中，n：问题规模的某种测度。

$f(n)$：是与机器及语言有关的量。

$g(n)$：是事前分析的结果，一个形式简单的函数，与频率计数有关、而与机器及语言无关。

1. 上界，O记号

O(g(n))表示以下函数的集合：

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

- 若f(n)和g(n)满足以上关系，则记为f(n)∈O(g(n))，表示f(n)是集合O(g(n))的成员。并通常记作

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

含义：

f(n)=O(g(n))表示如果算法用n值不变的同一类数据（规模相等，性质相同）在某台机器上运行，所用的时间总小于|g(n)|的一个常数倍。

- O记号给出的是渐近上界，称为上界函数（upper bound）
- 上界函数代表了**算法最坏情况下的时间复杂度**。
- 在确定上界函数时，应试图找阶最小的g(n)作为f(n)的上界函数——紧确的上界(***tight upper bound***)。

如：若：$3n+2=O(n^2)$ 则是松散的上界；

而：$3n+2=O(n)$ 就是紧确的上界。

2. 下界，Ω记号

Ω(g(n))表示以下函数的集合：

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

- 若f(n)和g(n)满足以上关系，则记为f(n)∈Ω(g(n))，表示f(n)是集合Ω(g(n))的成员。并通常记作

f(n)=Ω(g(n))。

含义：

f(n)=Ω(g(n))表示如果算法用n值不变的同一类数据在某台机器上运行，所用的时间总不小于|g(n)|的一个常数倍。

- Ω记号给出一个渐近下界，称为下界函数（lower bound）。
- 在确定下界函数时，应试图找出数量级最大的g(n)作为f(n)的下界函数——紧确的下界(***tight lower bound***)。

如：若：$3n^2+2=\Omega(n)$ 则是松散的下界；

而：$3n^2+2=O(n^2)$ 就是紧确的下界。

3. 渐近紧确界，Θ记号：

Θ(g(n))表示以下函数的集合：

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}.^1$$

- 若f(n)和g(n)满足以上关系，记为：$\mathrm{f}(n) \in \Theta(g(n))$，表示f(n)是Θ(g(n))的一员。通常记为：

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

含义：

f(n)=Θ(g(n))表示如果算法用n值不变的同一类数据在某台机器上运行，所用的时间既不小于|g(n)|的一个常数倍，也不大于|g(n)|的一个常数倍，亦即g 既是f 的下界，也是f 的上界。

- Θ记号给出的是渐近紧确界(*asymptotically tight bound*)

- 从时间复杂度的角度看, $f(n)=\Theta(g(n))$ 表示算法在最好和最坏情况下的计算时间就一个常数因子范围内而言是相同的，可看作：

既有 f(n) = Ω(g(n))，又有f(n) = O(g(n))

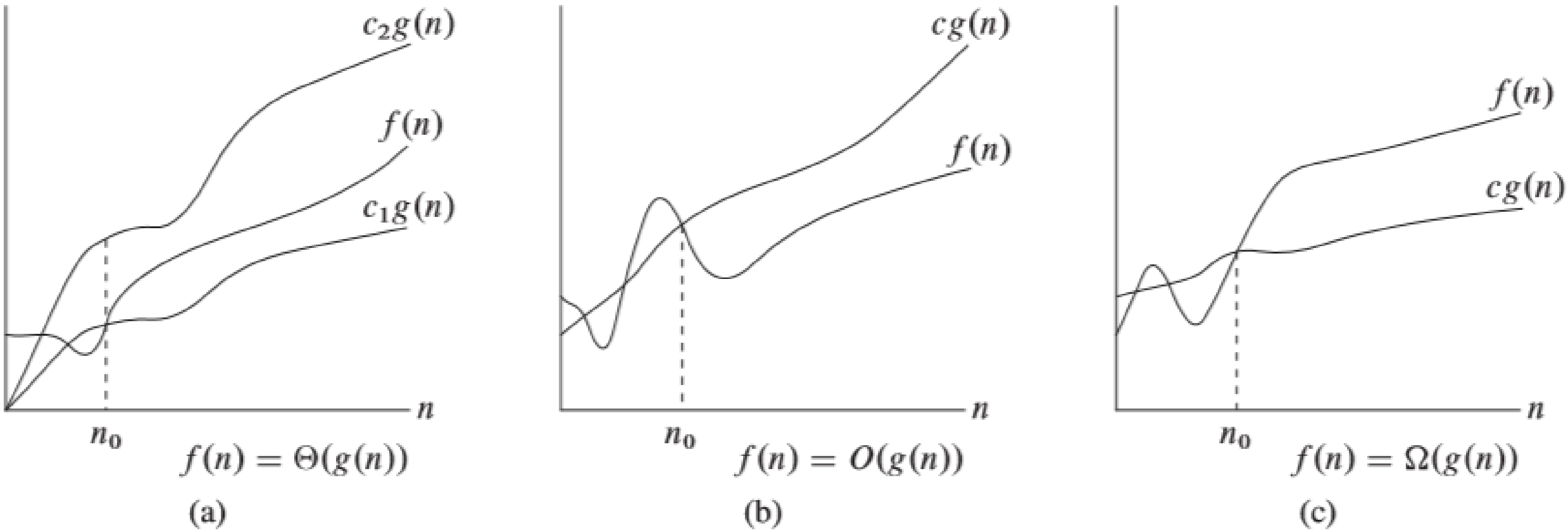


Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that at and to the right of n_0, the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

例：证明 $n^2/2-3n=\Theta(n^2)$

分析： 根据Θ的定义，仅需确定正常数c_1，c_2，and n_0 以使得对所有的$n \geq n_0$，有：$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$

解： 两边同除n^2得：$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$.

只要**$c_1 \le 1/14$, $c_2 \ge 1/2$, 且$n_0 \ge 7$**, 不等式即成立。

所以，这里取：$c_1=1/14$，$c_2=1/2$，and $n_0=7$， 即得证：

$n^2/2-3n=\Theta(n^2)$ 。 ■

注：还有其它常量可选，但根据定义，只要存在一组选择（如上述的c_1, c_2, 和 n_0）即得证。

再如：证明 $6n^3 \neq \Theta(n^2)$

采用反证法：假设$6n^3 = \Theta(n^2)$

则存在c_2和n_0，使得对所有的$n \geqslant n_0$，有：$6n^3 \leqslant c_2 n^2$。

两边同除n^2得： $n \leqslant c_2/6$，

由于n是变量，c_2是常量，所以对任意大的n，该式不可能成立。

所以假设不成立。 ■

关于渐近记号的进一步说明：

（1）$f(n)=O(g(n))$ 不能写成 $g(n)=O(f(n))$，Ω相同。

- $f(n)$与$g(n)$并不等价，这里的等号不是通常相等的含义。

（2）关于Θ(1)（O(1)、Ω(1)有类似的含义）

- 因为任意常量都可看做是一个0阶多项式，所以可以把任意常量函数表示成$\Theta(n^0)$或$\Theta(1)$。
- 通常用$\Theta(1)$表示具有**常量计算时间**的复杂度，即算法的执行时间为一个固定量，与问题的规模n没关系。

注：Θ(1)有“轻微活用”的意思，因为该表达式没有指出是什么量趋于无穷（见P27的说明）

4. o,ω记号

O、Ω给出的渐近上界或下界可能是也可能不是**渐近紧确**的。这里引入o, ω记号专门用来表示一种**非渐近紧确**的上界或下界。

o记号：对**任意**正常数c，存在常数$n_0 > 0$，使对**所有的**$n \geq n_0$，有$|f(n)| \leq c|g(n)|$，则记作：$f(n) = o(g(n))$。

含义：在o表示中，当n趋于无穷时，f(n)相对于g(n)来说变得微不足道了，即 $\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$

例：$2n = o(n^2)$，但$2n \neq o(n)$、 $2n^2 \neq o(n^2)$

ω记号：对**任意**正常数c，存在常数$n_0 > 0$，使对所有的$n \ge n_0$，有$c|g(n)| \le |f(n)|$，则记作：$f(n) = \omega(g(n))$。

含义：在ω表示中，当n趋于无穷时，f(n)相对于g(n)来说变得任意大了，即 $\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$

例：$n^2/2 = \omega(n)$，但$n/2 \neq \omega(n)$、$n^2/2 \neq \omega(n^2)$

O和o的区别

O： $f(n)=O(g(n)) \Leftrightarrow \exists c, n_0 : (n \geq n_0 \Rightarrow f(n) \leq cg(n))$

o： $f(n)=o(g(n)) \Leftrightarrow \forall c : (\exists n_0 : n \geq n_0 \Rightarrow f(n) \leq cg(n))$

Ω和ω的区别

Ω： $f(n)=\Omega(g(n)) \Leftrightarrow \exists c, n_0 : (n \geq n_0 \Rightarrow cg(n) \leq f(n))$

ω： $f(n)=\omega(g(n)) \Leftrightarrow \forall c : (\exists n_0 : n \geq n_0 \Rightarrow cg(n) \leq f(n))$

3.2 限界函数的性质

① 传递性（Transitivity）：

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

② 自反性（Reflexivity）：

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

③ 对称性（Symmetry）：

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

④ 转置对称性（Transpose Symmetry）：

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

仅从函数的数学定义理解这些表达式的含义

3.3 相关定理

定理1.1（多项式定理）若$A(n)=a_m n^m+\cdots+a_1 n+a_0$是一个n的m次多项式，则有 $A(n) = O(n^m)$

即：变量n的固定阶数为m的多项式，与此多项式的最高次项n^m同阶。

证明：取$n_0=1$，当$n \geq n_0$时，有

$$|A(n)| \leq |a_m| n^m+\cdots+|a_1| n+|a_0|$$

$$=(|a_m|+|a_{m-1}|/n+\cdots+|a_0|/n^m)\ n^m$$

$$\leq (|a_m|+|a_{m-1}|+\cdots+|a_0|)\ n^m$$

令$c= |a_m|+|a_{m-1}|+\cdots+|a_0|$，即有$|A(n)| \leq cn^m$。 证毕。

应用：

- 如果一个算法的时间复杂度函数是多项式形式，则其阶函数(复杂度函数表示)就可取该多项式的最高次项。

$$A(n)=a_m n^m+\cdots+a_1 n+a_0 \Rightarrow A(n) = O(n^m)$$

- 事实上，根据渐近关系，对于足够大的n，低阶项（包括常数项）是无足轻重的，即当n较大时，即使最高阶项的一个很小部分都足以“支配”所有的低阶项。所以用阶函数表示限界函数时，低阶项和常数项均被忽略。

例：考虑二次函数$f(n)=an^2+bn+c$，其中a、b、c为常量且$a>0$。

根据上述思路，去掉低阶项并忽略常系数后即得：

$$f(n)=\Theta(n^2)$$

对比下面形式化证明：

取常量：$c_1=a/4$，$c_2=7a/4$，$n_0 = 2\cdot\max(|b|/a, \sqrt{|c|/a})$

可以证明对所有的$n\geq n_0$，有:

$$0 \leq c_1 n^2 \leq an^2+bn+c \leq c_2 n^2 \qquad \blacksquare$$

一般而言，对任意多项式 $p(n)=\sum_{i=0}^{d} a_i n^i$，其中$a_i$为常数且$a_d>0$，都有 $p(n)=\Theta(n^d)$

用于估算复杂性（函数阶的大小）的定理

[定理1.2] 对于任意正实数x 和 ε ，有下面的不等式：

1）存在某个n_0，使得对于任何$n \geq n_0$，有$(\log n)^x < (\log n)^{x+\varepsilon}$

2）存在某个n_0，使得对于任何$n \geq n_0$，有$n^x < n^{x+\varepsilon}$。

3）存在某个n_0，使得对于任何$n \geq n_0$，有$(\log n)^x < n$。

4）存在某个n_0，使得对于任何$n \geq n_0$，有$n^x < 2^n$。

5）对任意实数y，存在某个n_0，使得对于任何$n \geq n_0$，有

$$n^x (\log n)^y < n^{x+\varepsilon}$$

例:

根据定理1.2，很容易得出：

$n^3 + n^2\log n = O(n^3)$；

$n^4 + n^{2.5}\log^{20} n = O(n^4)$；

$2^n n^4 \log^3 n + 2^n n^5/\log^3 n = O(2^n n^5)$。

定理1.3： 设d(n)、e(n)、f(n)和g(n)是将非负整数映射到非负实数的函数，则

(1) 如果d(n)是O(f(n))，那么对于**任何常数**a>0，ad(n)是O(f(n))；

(2) 如果d(n)是O(f(n))， e(n)是O(g(n))，那么d(n)+e(n)是O(f(n)+g(n)) —— 加法法则；

(3) 如果d(n)是O(f(n))， e(n)是O(g(n))，那么d(n)e(n)是O(f(n)g(n)) —— 乘法法则；

(4)对于**任意固定**的x>0和a>1，n^x是$O(a^n)$；

(5)对于任意固定的x>0，$\log n^x$是O(logn)；

(6)对于任意固定的常数x>0和y>0，$\log^x n$是$O(n^y)$；

例：$2n^3+4n^2\log n=O(n^3)$

证明：$\log n = O(n)$ 规则6

$4n^2\log n = O(4n^3)$ 规则3

$2n^3+4n^2\log n = O(2n^3+4n^3)$ 规则2

$2n^3+4n^3 = O(n^3)$ 规则1

所以，$2n^3+4n^2\log n = O(n^3)$

算法时间复杂度的分类

根据上界函数的特性，可以将算法分为：多项式时间算法和指数时间算法。

- 多项式时间算法：可用多项式函数对计算时间限界的算法
  - 常见的多项式限界函数有：

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3)$$

复杂度越来越高

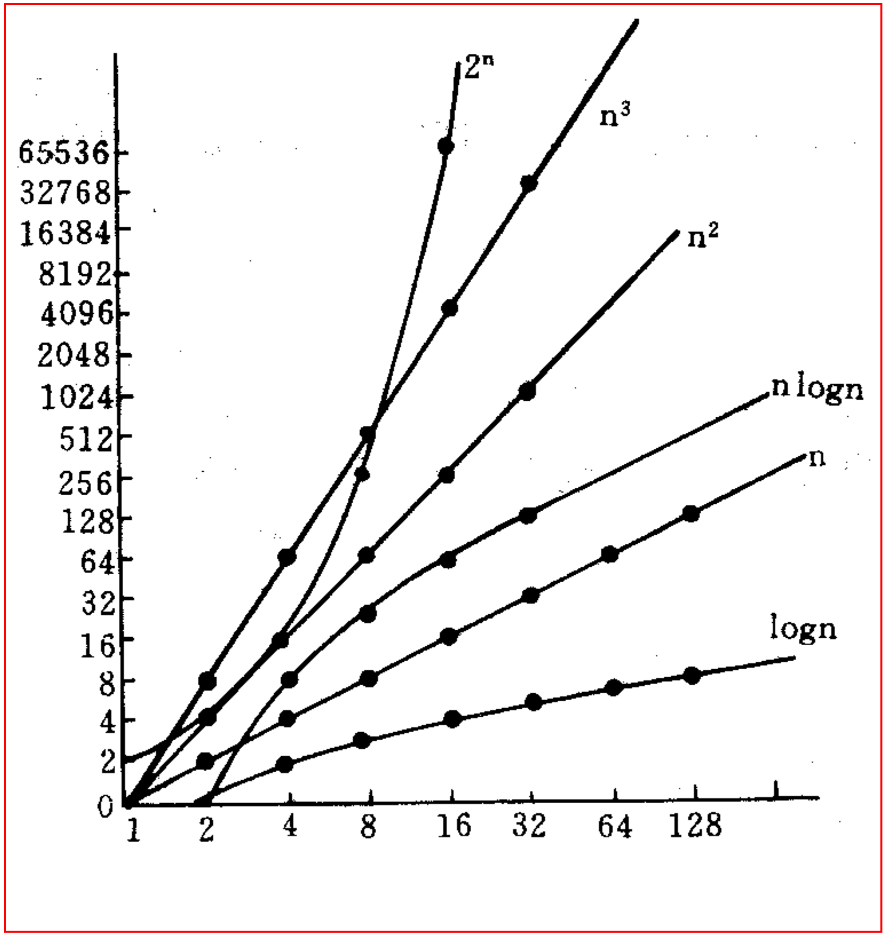
- 指数时间算法：计算时间用指数函数限界的算法。
  - 常见的指数限界函数：

$$O(2^n) < O(n!) < O(n^n)$$

复杂度越来越高

- 当n取值较大时，指数时间算法和多项式时间算法在计算时间上非常悬殊。

计算时间的典型函数曲线：



计算时间函数值的比较

表1. 1 典型函数的值

logn	n	nlogn	n_2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

对算法复杂性的一般认识

- 当数据集的规模很大时，要在现有的计算机系统上运行具有比O(nlogn)复杂度还高的算法是比较困难的。
- 指数时间算法只有在n取值非常小时才实用。
- 要想在顺序处理机上扩大所处理问题的规模，有效的途径是降低算法的计算复杂度，而不是（仅仅依靠）提高计算机的速度。

3.3 标准记号与常用函数（自学）

需要熟悉一些常用的数学函数和记号

1. Monotonicity（单调性）

- A function f(n) is ***monotonically increasing*** ***（单调递增）***

  if $m \leq n$ implies $f(m) \leq f(n)$.

- A function f(n) is ***monotonically decreasing*** ***（单调递减）***

  if $m \leq n$ implies $f(m) \geqslant f(n)$.

- A function f(n) is ***strictly increasing*** ***（严格递增）***

  if $m < n$ implies $f(m) < f(n)$

- A function f(n) is ***strictly decreasing*** ***（严格递减）***

  if $m < n$ implies $f(m) > f(n)$.

2. Floors and ceilings（向下取整和向上取整）

- For all real x,

$$x-1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x+1$$

- For any integer n,

$$\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$$

- for any real number x⩾0 and integers a, b > 0,

$$\left\lceil \frac{\lceil x/a \rceil}{b} \right\rceil = \left\lceil \frac{x}{ab} \right\rceil , \qquad \left\lceil \frac{a}{b} \right\rceil \le \frac{a+(b-1)}{b} ,$$

$$\left\lfloor \frac{\lfloor x/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor , \qquad \left\lfloor \frac{a}{b} \right\rfloor \ge \frac{a-(b-1)}{b} .$$

3. Modular arithmetic（模运算）

- If (a mod n) = (b mod n), we write $a \equiv b \pmod{n}$ and say that a is ***equivalent*** to b, modulo n(模n时a等价于b，a、b同余).

4. **Polynomials**（多项式）

- Given a nonnegative integer d, a ***polynomial in*** n ***of degree*** d is a function p(n) of the form

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

- where the constants $a_0, a_1, \ldots, a_d$ are the ***coefficients*** of the polynomial and $a_d \neq 0$.

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

- A polynomial is asymptotically positive(渐近为正) if and only if $a_d > 0$.
- For an asymptotically positive polynomial p(n) of degree d, we have $p(n)=\Theta(n^d)$.
- f(n) is ***polynomially bounded (多项式有界)*** if $f(n) = O(n^k)$ for some constant k.

5. Exponentials（指数）

- For all real a > 0, m, and n, we have the following identities:

$$\begin{aligned} a^0 &= 1 , \\ a^1 &= a , \\ a^{-1} &= 1/a , \\ (a^m)^n &= a^{mn} , \\ (a^m)^n &= (a^n)^m , \\ a^m a^n &= a^{m+n} . \end{aligned}$$

- For all real constants a and b such that a > 1,

$$\lim_{n \to \infty} \frac{n^b}{a^n} = 0 ,$$

  - from which we can conclude that $n^b = o(a^n)$.
  - That is any exponential function with a base strictly greater than 1 grows faster than any polynomial function.

6. Logarithms（对数）

$$
\begin{aligned}
\lg n &= \log_2 n && \text{(binary logarithm)} \ , \\
\ln n &= \log_e n && \text{(natural logarithm)} \ , \\
\lg^k n &= (\lg n)^k && \text{(exponentiation)} \ , \\
\lg \lg n &= \lg(\lg n) && \text{(composition)} \ .
\end{aligned}
$$

For all real a > 0, b > 0, c > 0, and n,

$$
\begin{aligned}
a &= b^{\log_b a} \ , \\
\log_c(ab) &= \log_c a + \log_c b \ , \\
\log_b a^n &= n \log_b a \ , \\
\log_b a &= \frac{\log_c a}{\log_c b} \ , \\
\log_b(1/a) &= -\log_b a \ , \\
\log_b a &= \frac{1}{\log_a b} \ , \\
a^{\log_b c} &= c^{\log_b a} \ ,
\end{aligned}
$$

where, in each equation above, logarithm bases are not 1.

7. Factorials（阶乘）

The notation $n!$ (read "n factorial") is defined for integers $n \geq 0$ as

$$n! = \begin{cases} 1 & \text{if } n = 0 , \\ n \cdot (n-1)! & \text{if } n > 0 . \end{cases}$$

Thus, $n! = 1 \cdot 2 \cdot 3 \cdots n$.

- A weak upper bound on the factorial function is $n! \leq n^n$.
- ***Stirling′s approximation(斯特林近似公式)***

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

- Can prove more:

$$\begin{aligned} n! &= o(n^n) , \\ n! &= \omega(2^n) , \\ \lg(n!) &= \Theta(n \lg n) \end{aligned}$$

课外阅读：

算法导论（3rd）：3.1渐近记号

其他小节自行阅读

- 作业：

3.1-5 证明定理 3.1。