

算法设计与分析

Computer Algorithm Design & Analysis

赵峰

zhaof@hust.edu.cn



Chapter 3

Growth of Functions

函数的增长

本章研究算法的渐进效率

- 当输入规模 $\rightarrow \infty$ 时，算法的运行时间相对问题的规模如何变化呢？
- 当输入规模足够大，执行时间仅与增长量级有关。
- 本章引入渐进记号，并基于这些记号，给出算法的下界、上界和紧确界的定义。

渐进记号 (Asymptotic notation) 用于刻画算法的时间复杂度限界函数 (空间类似 , 这里仅以时间分析为例) 。

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、渐进紧确界函数。

记 : 算法的实际执行时间为 $f(n)$, 执行时间的限界函数为 $g(n)$ 。

1. Θ 记号

对于给定的 $g(n)$, $\Theta(g(n))$ 表示以下函数的集合：

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

- 如果存在正常量 c_1 和 c_2 , 使得对于足够大的 n , 函数 $f(n)$ 能 “夹入”
 $c_1 g(n)$ 和 $c_2 g(n)$ 之间 , 则称 $f(n)$ 属于集合 $\Theta(g(n))$, 记为 $f(n) \in \Theta(g(n))$
表示 $f(n)$ 是 $\Theta(g(n))$ 的一员。

- 通常也记为 $f(n) = \Theta(g(n))$

即：如果存在正常数 c_1 , c_2 和 n_0 , 对于所有的 $n \geq n_0$, 有

$$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)| ,$$

则记作 $f(n) = \Theta(g(n))$

含义：

- 对所有的 $n \geq n_0$ ，函数 $f(n)$ 在一个常数因子内等价于 $g(n)$ 。
- 称 $g(n)$ 是 $f(n)$ 的一个渐进紧确界(*asymptotically tight bound*)
 - 从算法时间复杂度的角度看，就是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 $f(n) = \Omega(g(n))$ ，又有 $f(n) = O(g(n))$

即， g 既是 f 的下界，又是 f 的上界。

例：证明 $n^2/2 - 3n = \Theta(n^2)$

思路：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：
$$c_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 n^2$$

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

可得：只要 $c_1 \leq 1/14, c_2 \geq 1/2$, 且 $n_0 \geq 7$, 不等式即成立。

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

$$n^2/2 - 3n = \Theta(n^2) . \quad \blacksquare$$

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：

假设存在 c_2 和 n_0 ，使得对所有的 $n \geq n_0$ ，有： $6n^3 \leq c_2 n^2$ 。

两边同除 n^2 得： $n \leq c_2/6$ ，

而 c_2 是常量，所以对任意大的 n ，该式不可能成立。■

进一步说明：

- 一个渐进正函数的低阶项在确定渐进确界时可以忽略

- 因为对于足够大的 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。

因此，在证明不等式成立时，只要将 c_1 置为稍小于最高阶项系数的值并将 c_2 置为稍大于最高阶项系数的值即能使 Θ 记号定义中的不等式得到满足。

同时最高阶项系数同样可以忽略，因为它仅仅根据一个等于该系数的常量因子来改变 c_1 和 c_2 。

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

根据上述思路，去掉低阶项并忽略常量系数后即得：

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

对比形式化证明：

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：

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

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 $a_d > 0$ ， $p(n) = \Theta(n^d)$

- $f(n)=O(g(n))$ 不能写成 $g(n)=O(f(n))$ 。
 - $f(n)$ 与 $g(n)$ 并不等价，这里的等号不是通常相等的含义。
 - 关于 $\Theta(1)$
 - 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 $\Theta(1)$ 。
- 注： $\Theta(1)$ 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）。
- 经常用 $\Theta(1)$ 表示具有常量计算时间的算法的时间复杂度—— 算法的执行计算时间与问题的规模 n 没关系。

2. O记号

- $O(g(n))$ 表示以下函数的集合：

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

- 也可表述如下：

如果存在两个正常数 c 和 n_0 ，对于所有的 $n \geq n_0$ ，有 $|f(n)| \leq c|g(n)|$ ，则记作 $f(n)=O(g(n))$ 。

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

含义：

- ◆ $f(n) = O(g(n))$ 表示 $f(n)$ 是集合 $O(g(n))$ 的成员；
- ◆ $f(n) = \Theta(g(n))$ 蕴涵 $f(n) = O(g(n))$ ；
- ◆ 从算法时间复杂度的角度，可以理解为：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记号给出一个**渐进上界**。
- 函数f 至多是函数g 的c倍，除非 $n < n_0$ 。即当n 充分大时，g 是f 的一个**上界函数** (upper bound) 。
- 可用O记号来表示**算法的最坏情况时间复杂度**。
 - ▣ 应试图找**阶最小的** $g(n)$ 作为 $f(n)$ 的上界函数。
即 $g(n)$ 应该尽量接近函数 $f(n)$ ——紧确(***tight bound***)。
如：若： $3n+2=O(n^2)$ 则是**松散**的界限；
而： $3n+2=O(n)$ 就是**紧确**的界限。
- **数量级越小，限界越精确**。
- 不要产生错误界限。
如： $n^2+100n+6$ ，当 $n < 3$ 时， $n^2+100n+6 < 106n$ ，但不代表 $n^2+100n+6 = O(n)$ ：
对任何正的常数c，只要 $n > c-100$ 就有 $n^2+100n+6 > cn$ 。
同理， $3n^2+4 \times 2n = O(n)$ 也是错误的。

3. Ω 记号

- $\Omega(g(n))$ 表示以下函数的集合：

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

- 也可表述如下：

如果存在两个正常数 c 和 n_0 ，对于所有的 $n \geq n_0$ ，有 $|f(n)| \geq c|g(n)|$ ，
则记作 $f(n) = \Omega(g(n))$ 。

含义：

- ◆ $f(n) = \Omega(g(n))$ 表示 $f(n)$ 是集合 $\Omega(g(n))$ 的成员；
- ◆ $f(n) = \Theta(g(n))$ 蕴涵 $f(n) = \Omega(g(n))$ ；
- ◆ 如果算法用 n 值不变的同的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 记号给出一个渐进下界。

- 函数 f 至少是函数 g 的 c 倍，除非 $n < n_0$ 。即当 n 充分大时， g 是 f 的一个下界函数 (lower bound)。
- 应试图找出数量级最大的 $g(n)$ 作为 $f(n)$ 的下界函数，即 $g(n)$ 应该尽量接近函数 $f(n)$ 。
- 类似于大 O 符号，可以参考定理1.1 所列的不等式，来估计复杂性函数的下界。

等式和不等式中的渐进记号

- 类似以下的表达式：

$$T(n) = 2T(n/2) + \Theta(n)$$

当**渐进记号**出现在**某个公式中**，该如何理解？

- 将其解释为代表我们不关注名称的**匿名函数**。
 - ▣ 该匿名函数代表表达式中无关紧要的细节 —— 存在但不被特别关注。
 - ▣ 我们只对 $T(n)$ 的渐进行为感兴趣， $T(n)$ 里面的渐进记号仅代表低阶项部分。
- 这样的表示帮助消除了表达式中一些无关紧要的细节。
 - ▣ 但在实际化简的过程中，还要根据需要给予“具体化”
 - ▣ 详见P28~29

4. o, ω 记号

O 、 Ω 给出的渐进上界或下界可能是也可能不是渐进紧确的。

这里引入 o, ω 记号专门用来表示一种非渐进紧确的上界或下界。

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
 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
 $g(n)$ 来说变得微不足道了，即

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

ω 记号：对任意正常数 c ，存在常数 $n_0 > 0$ ，使对所有的 $n \geq n_0$ ，有 $c|g(n)| \leq |f(n)|$ ，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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

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

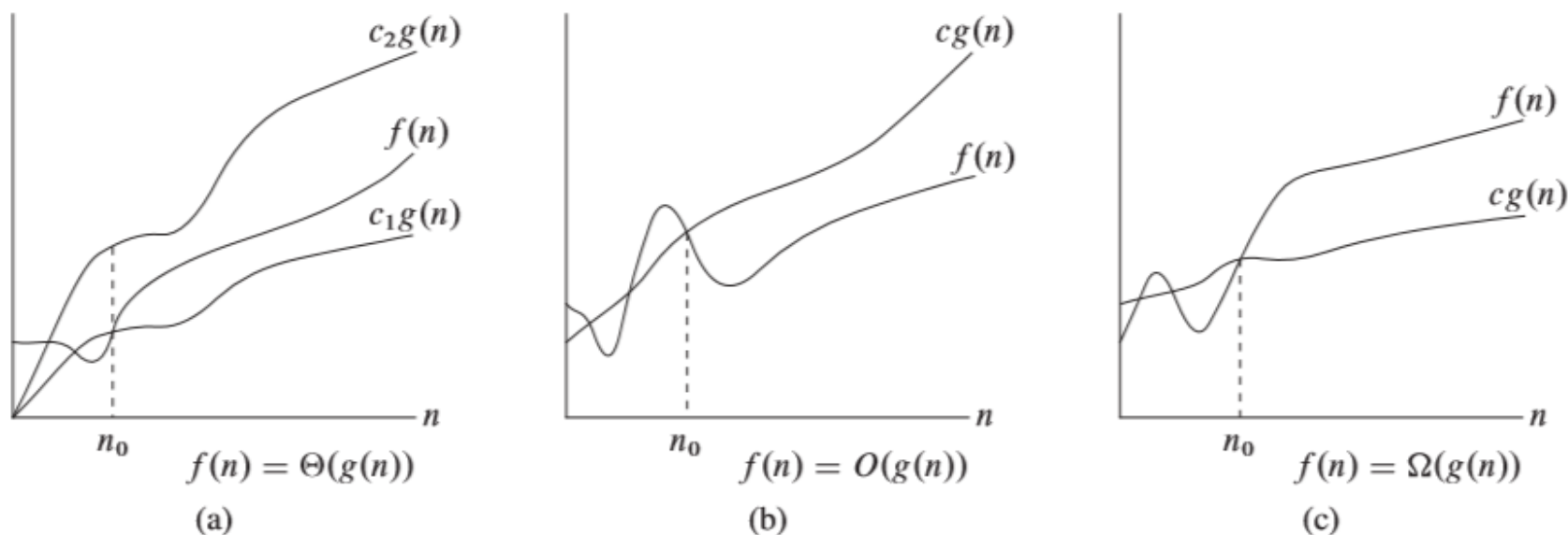


Figure 3.1 Graphic examples of the Θ , O , and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ -notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0 , c_1 , and c_2 such that at and to the right of n_0 , the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ 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 $cg(n)$.

5. 限界函数的性质

① 传递性 (Transitivity) :

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

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) :

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

算法时间复杂度的分类

根据上界函数（渐进紧确上界）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。

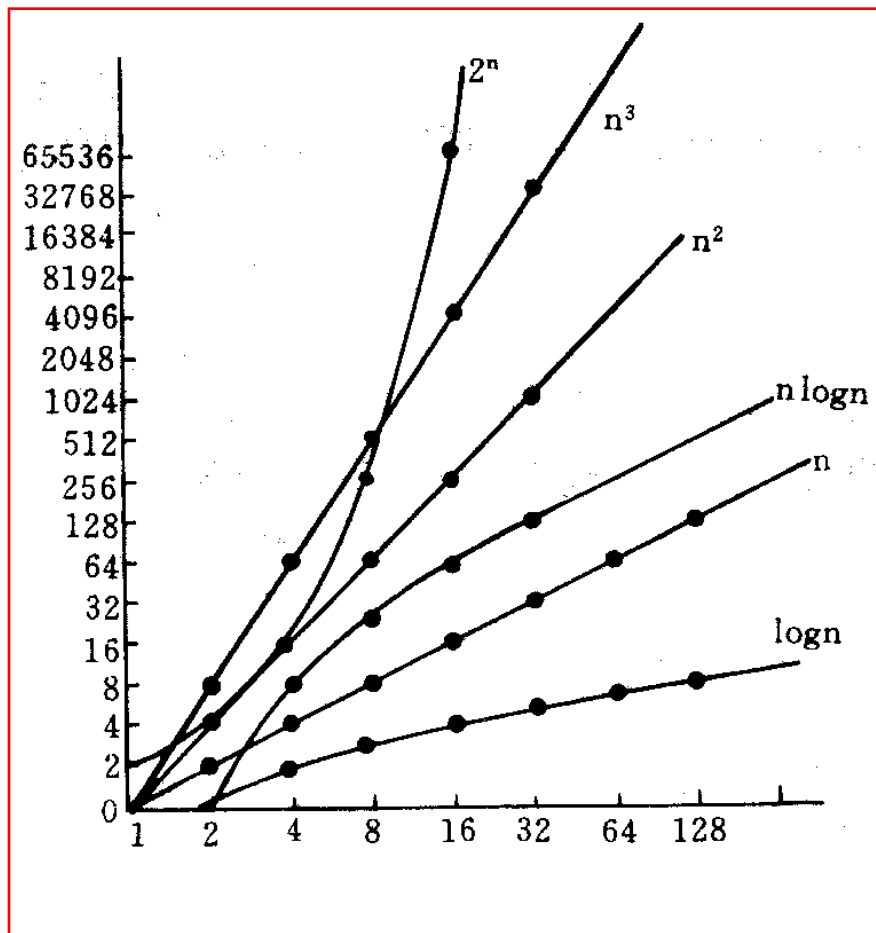
常见的指数时间限界函数：

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

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 $O(n \log n)$ 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。

3.2 标准记号与常用函数

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 $f(m) \geq f(n)$.
- 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 ,

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

- For any integer n ,

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

- for any real number $x \geq 0$ and integers $a, b > 0$,

$$\begin{aligned} \left\lceil \frac{\lfloor x/a \rfloor}{b} \right\rceil &= \left\lceil \frac{x}{ab} \right\rceil, & \left\lceil \frac{a}{b} \right\rceil &\leq \frac{a + (b - 1)}{b}, \\ \left\lfloor \frac{\lceil x/a \rceil}{b} \right\rfloor &= \left\lfloor \frac{x}{ab} \right\rfloor, & \left\lfloor \frac{a}{b} \right\rfloor &\geq \frac{a - (b - 1)}{b}. \end{aligned}$$

3. Modular arithmetic (模运算)


- If $(a \bmod n) = (b \bmod n)$, 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, \dots, 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(渐进为正) 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:

$$a^0 = 1 ,$$

$$a^1 = a ,$$

$$a^{-1} = 1/a ,$$

$$(a^m)^n = a^{mn} ,$$

$$(a^m)^n = (a^n)^m ,$$

$$a^m a^n = a^{m+n} .$$

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

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

- 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)

$$\lg n = \log_2 n \quad (\text{binary logarithm}) ,$$

$$\ln n = \log_e n \quad (\text{natural logarithm}) ,$$

$$\lg^k n = (\lg n)^k \quad (\text{exponentiation}) ,$$

$$\lg \lg n = \lg(\lg n) \quad (\text{composition}) .$$

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

$$a = b^{\log_b a} ,$$

$$\log_c(ab) = \log_c a + \log_c b ,$$

$$\log_b a^n = n \log_b a ,$$

$$\log_b a = \frac{\log_c a}{\log_c b} ,$$

$$\log_b(1/a) = -\log_b a ,$$

$$\log_b a = \frac{1}{\log_a b} ,$$

$$a^{\log_b c} = c^{\log_b a} ,$$

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 :

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