

# 算法设计与分析

Computer Algorithm Design & Analysis



吕志鹏

[zhipeng.lv@hust.edu.cn](mailto:zhipeng.lv@hust.edu.cn)

群名称：算法设计与分析2020

群 号：271069522

群名称：算法设计与分析2020

群 号：271069522




## Chapter 3

---

# Growth of Functions

函数的增长



本章研究算法的渐进效率，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，引入渐进记号。

## 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、渐进紧确界函数。

对应的渐进记号： $O$                        $\Omega$                        $\Theta$

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:

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

- ◆ 若 $f(n)$ 和 $g(n)$ 满足以上关系, 则记为 $f(n) \in O(g(n))$ , 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 $n$ 值不变的同的一类数据 (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记号给出的是渐进上界，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——紧确上界(*tight upper 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。

## 2. 下界, $\Omega$ 记号

$\Omega(g(n))$ 表示以下函数的集合:


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

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

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

含义:

$f(n) = \Omega(g(n))$ 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。

- 
- $\Omega$ 记号给出一个渐进下界，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——紧确下界。



### 3. 渐进紧确界， $\Theta$ 记号：

$\Theta(g(n))$  表示以下函数的集合：

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

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

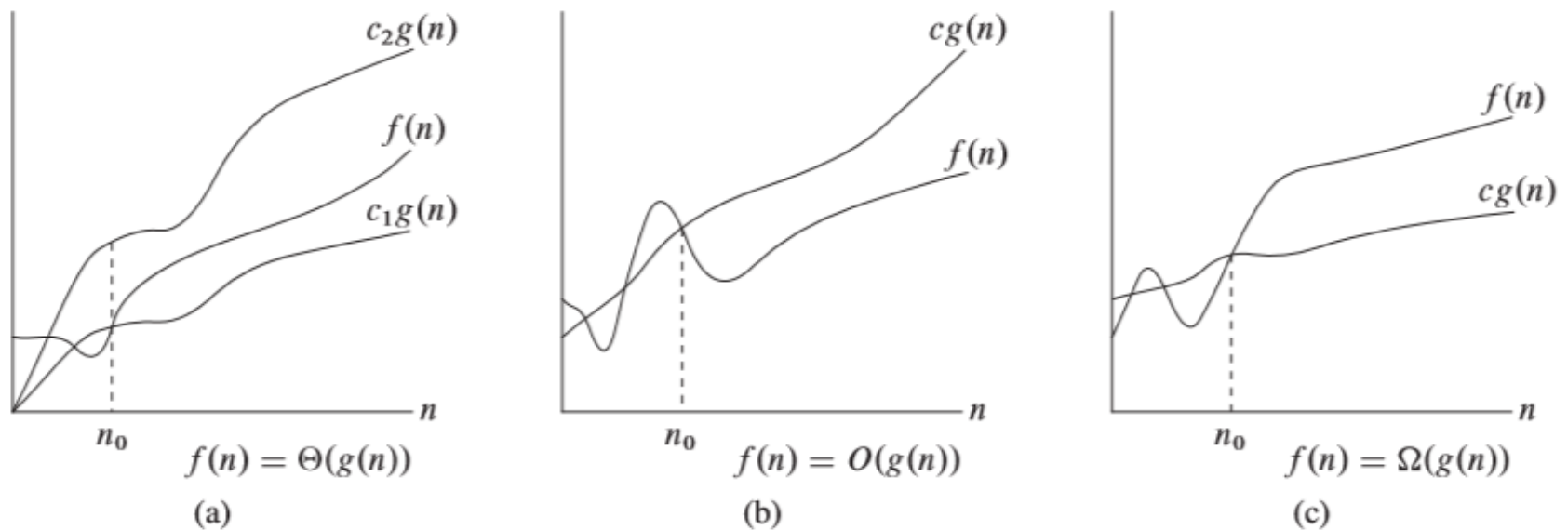
$$f(n) = \Theta(g(n))$$

含义：

$f(n) = \Theta(g(n))$ 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。

- ◆  $\Theta$ 记号给出的是渐进紧确界(*asymptotically tight bound*)
- ◆ 从时间复杂度的角度看,  $f(n) = \Theta(g(n))$  表示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  $f(n) = O(g(n))$



**Figure 3.1** Graphic examples of the  $\Theta$ ,  $O$ , and  $\Omega$  notations. In each part, the value of  $n_0$  shown is the minimum possible value; any greater value would also work. **(a)**  $\Theta$ -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)**  $\Omega$ -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 $\Theta$ 的定义，仅需确定正常数 $c_1$ ,  $c_2$ , and  $n_0$  以使得

$$\text{对所有的 } n \geq n_0, \text{ 有: } c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

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

只要  **$c_1 \leq 1/14$ ,  $c_2 \geq 1/2$ , 且  $n_0 \geq 7$** , 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：

$$n^2/2 - 3n = \Theta(n^2) \text{ 。}$$



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  $n \geq n_0$ ，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。

所以假设不成立。 

## 关于渐进记号的进一步说明：

(1)  $f(n)=O(g(n))$  不能写成  $g(n)=O(f(n))$ ， $\Omega$ 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 $\Theta(1)$  ( $O(1)$ 、 $\Omega(1)$ 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： $\Theta(1)$ 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

### (3) 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？如上式的  $\Theta(n)$

- 将其解释为代表我们不关注名称的匿名函数，用以消除表达式中一些无关紧要的细节。
  - ▣ 这些细节存在但不被特别关注，分析时还是只对  $T(n)$  的渐进行为感兴趣。渐进记号仅代表低阶项部分，在实际化简的过程中，根据需要进行“具体化”，然后再进行化简处理。

## 4. o, ω 记号

$O$ 、 $\Omega$ 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  $o$ 、 $\omega$  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  $g(n)$  来说变得

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

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



**$\omega$  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  $\omega$  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  $n/2 \neq \omega(n)$ 、 $n^2/2 \neq \omega(n^2)$

## 3.2 限界函数的性质

### ① 传递性 (Transitivity) :

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \quad \text{imply} \quad f(n) = \Theta(h(n)) ,$$

$$f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \quad \text{imply} \quad f(n) = O(h(n)) ,$$

$$f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \quad \text{imply} \quad f(n) = \Omega(h(n)) ,$$

$$f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \quad \text{imply} \quad f(n) = o(h(n)) ,$$

$$f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \quad \text{imply} \quad f(n) = \omega(h(n)) .$$

### 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) :

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

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

仅从函数的数学定义理解其含义

### 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, 与此多项式的最高阶 $n^m$ 同阶。

证明: 取 $n_0=1$ , 当 $n \geq n_0$ 时, 有

$$\begin{aligned} |A(n)| &\leq |a_m| n^m + \cdots + |a_1| n + |a_0| \\ &= (|a_m| + |a_{m-1}|/n + \cdots + |a_0|/n^m) n^m \\ &\leq (|a_m| + |a_{m-1}| + \cdots + |a_0|) n^m \end{aligned}$$

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

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 $a_d > 0$ ，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  $\varepsilon$ ， 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 $O(f(n)+g(n))$ ；

(3) 如果 $d(n)$ 是 $O(f(n))$ ， $e(n)$ 是 $O(g(n))$ ，那么 $d(n)e(n)$ 是 $O(f(n)g(n))$ ；

(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 $O(\log n)$ ；

(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：  $\log n = \underline{O(n)}$  规则6

$\underline{4n^2 \log n} = \underline{O(4n^3)}$  规则3

$\underline{2n^3 + 4n^2 \log n} = \underline{O(2n^3 + 4n^3)}$  规则2

$\underline{2n^3 + 4n^3} = \underline{O(n^3)}$  规则1

所以，  $\underline{2n^3 + 4n^2 \log n} = \underline{O(n^3)}$



# 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。

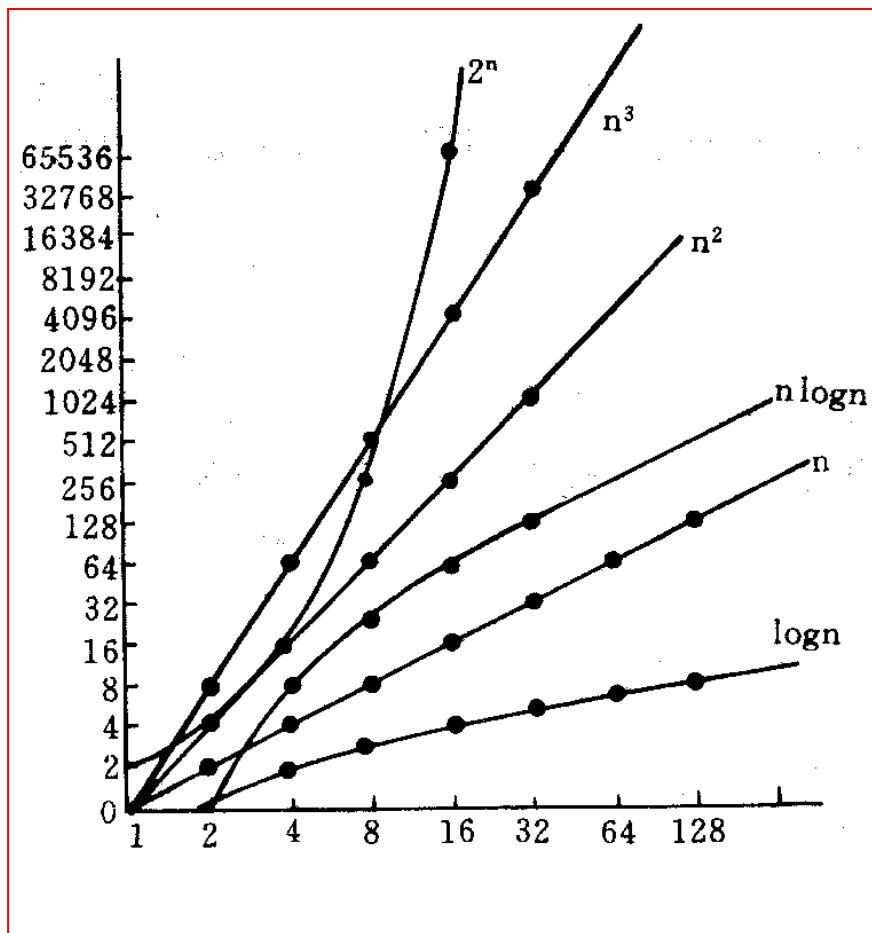
■ 常见的指数时间限界函数：

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

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：



# 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中运行具有比 $O(n\log n)$ 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。

## 3.3 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  $f(m) \geq f(n)$ .
- 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:  
$$\begin{aligned} n! &= o(n^n), \\ n! &= \omega(2^n), \\ \lg(n!) &= \Theta(n \lg n) \end{aligned}$$

课外阅读：

算法导论 (3<sup>rd</sup>) : 3.1 渐进记号

其他小节自行阅读

■ 作业：

**3.1-5** 证明定理 3.1。