



华中科技大学

计算机科学与技术学院

School of Computer Science & Technology, HUST

算法设计与分析

刘渝

Liu_yu@hust.edu.cn

2025秋季-华科-计算机

24级CS

Anytime·Everywhere
Computing

计算·无限





算法分析与设计

第三章

函数的增长

time



目 录

- 01、限界函数的定义
- 02、限界函数的性质
- 03、复杂度分类

限界函数的定义

算法时间复杂度的限界函数

限界函数	上界函数	下界函数	渐近紧确界函数
渐进符号	O	Ω	Θ



定义

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

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。



上界函数

$O(g(n))$ 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

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

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。



Example: 渐近上界

$$T(n) = 4n^2 - 2n + 2$$

当 $n = 500$, $4n^2$ 项是 $2n$ 项的1000倍大, $O(T(n)) = O(n^2)$

$$T(n) = n^3 - 10000n^2 + 2$$

当 $n > 10000$, n^3 将占主导, $O(T(n)) = O(n^3)$



思考：



华中科技大学
计算机科学与技术学院
School of Computer Science & Technology, HUST

无穷小的算式以及上界情况要如何表达呢？

PiMe



上界函数

O 记号给出的是渐近上界，称为上界函数

上界函数代表了**算法最坏情况下的时间复杂度**

紧确上界

阶最小的 $g(n)$ 作为 $f(n)$ 的上界函数



紧确上界与松散上界

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。



下界函数

$\Omega(g(n))$ 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

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

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**不小于** $|g(n)|$ 的一个常数倍。



下界函数

Ω 记号给出的是渐近下界，称为下界函数

下界函数代表了**算法最好情况下的时间复杂度**

紧确下界

阶最大的 $g(n)$ 作为 $f(n)$ 的下界函数



紧确下界与松散下界

若： $3n^2+2 = \Omega(n)$ 则是**松散**的下界；

而： $3n^2+2 = \Omega(n^2)$ 就是**紧确**的下界。



渐近紧确界函数

$\Theta(g(n))$ 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，则记为 $f(n) \in \Theta(g(n))$ ，表示 $f(n)$ 是集合 $\Theta(g(n))$ 的成员。并通常记作

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

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，所用的时间既**不小于** $|g(n)|$ 的一个常数倍，也**不大于** $|g(n)|$ 的一个常数倍。 **g 既是 f 的下界，也是 f 的上界。**



渐近紧确界函数

$\Theta(g(n))$ 表示以下函数的集合：

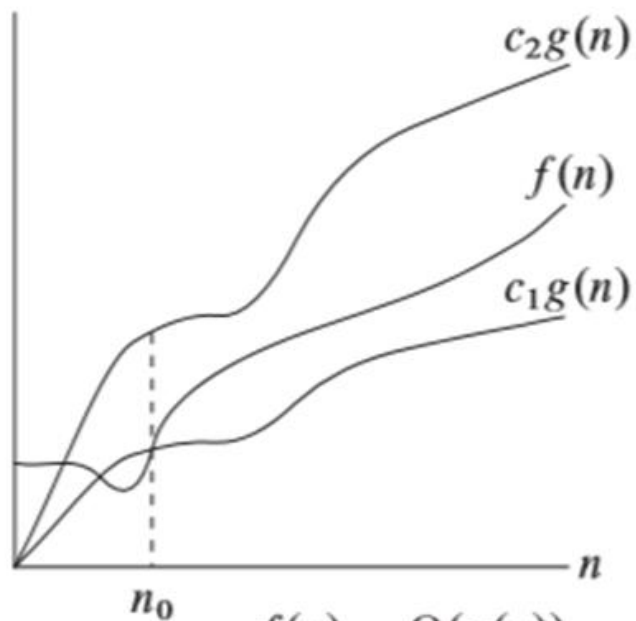
$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\} .^1$$

- ◆ 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:

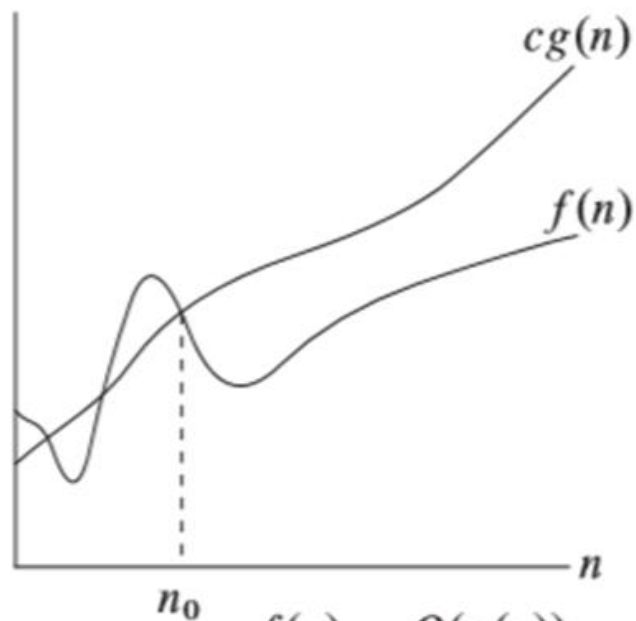
既有 $f(n) = \Omega(g(n))$, 又有 $f(n) = O(g(n))$



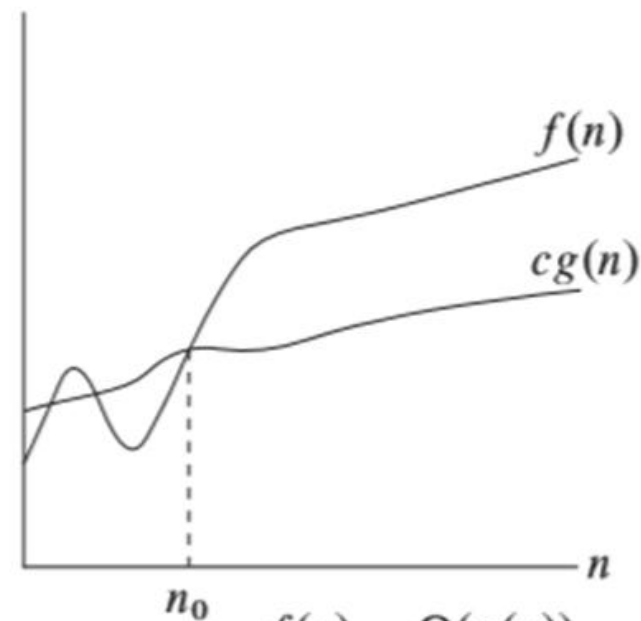
定义图例



(a)



(b)



(c)



总结：



华中科技大学
计算机科学与技术学院
School of Computer Science & Technology, HUST

- 算法分析的工作内容：求算法时间/空间复杂度的限界函数
 - 函数表达式的数量级：最高次项的次数
 - 限界函数：频率计数函数表达式中的最高次项
 - 统计算法中各类运算的执行次数，最终表示称为关于问题规模 n 的特征函数——**限界函数**





总结：

◆ 算法的五个重要特性：

- 确定性、可行性、输入、输出、有穷性
- 时效性



华中科技大学
计算机科学与技术学院
School of Computer Science & Technology, HUST





总结:

◆ 渐近记号 (Asymptotic notation) :

- 用于刻画算法的时间、空间复杂度限界函数
- 限界函数常用的有三个:

上界函数 O

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

下界函数 Ω

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

渐近紧确界 Θ

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}.$ ¹



华中科技大学
计算机科学与技术学院
School of Computer Science & Technology, HUST





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 , 和 n_0 , 以使得对所有
的 $n \geq n_0$, 有：

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

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： $c_1 = 1/14$, $c_2 = 1/2$, 和 $n_0 = 7$, 即得证：

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

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： $n \leq c_2/6$ ，

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。



渐近的进一步说明

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

➤ $f(n)$ 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。



渐近的进一步说明

(3) 等式和不等式中的渐近记号

类似以下的表达式：

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

当渐近记号出现在某个公式中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



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： $f(n) = o(g(n))$ 。

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，

$$\text{即 } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

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

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： $f(n) = \omega(g(n))$ 。

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，

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

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

O和o

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

$$o: f(n) = o(g(n)) \Leftrightarrow \forall c_0 : (\exists n_0 : n \geq n_0 \Rightarrow f(n) \leq c_0 g(n))$$

Ω 和 ω

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

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

限界函数的性质

传递性

$f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ imply $f(n) = \Theta(h(n))$,

$f(n) = O(g(n))$ and $g(n) = O(h(n))$ imply $f(n) = O(h(n))$,

$f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$,

$f(n) = o(g(n))$ and $g(n) = o(h(n))$ imply $f(n) = o(h(n))$,

$f(n) = \omega(g(n))$ and $g(n) = \omega(h(n))$ imply $f(n) = \omega(h(n))$.

自反性

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

$$f(n) = O(f(n)) ,$$

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

对称性

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

转置对称性

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

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

仅从函数的数学定义理解其含义



定理1.1 多项式定理

若 $A(n)=a_m n^m + \dots + a_1 n + a_0$ 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

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

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

证毕。



定理1.1 多项式定理

若 $A(n) = a_m n^m + \dots + a_1 n + a_0$ 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。

- 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。



Example: 二次函数

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

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 $p(n) = \Theta(n^d)$



定理1.2 复杂性估算定理

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

Example: 复杂性估算定理

$$n^3 + n^2 \log n = O(n^3) ;$$

$$n^4 + n^{2.5} \log^{20} n = O(n^4) ;$$

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



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**；

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

(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 $O(n^y)$;



Example

例: $2n^3 + 4n^2 \log n = O(n^3)$

证明: $\log n = \underline{O(n)}$

规则6

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

规则3

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

规则2

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

规则1

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

时间复杂度分类

根据上界函数的特性，可以将算法分为：

- 多项式时间算法
- 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

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

→ 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

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

→ 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

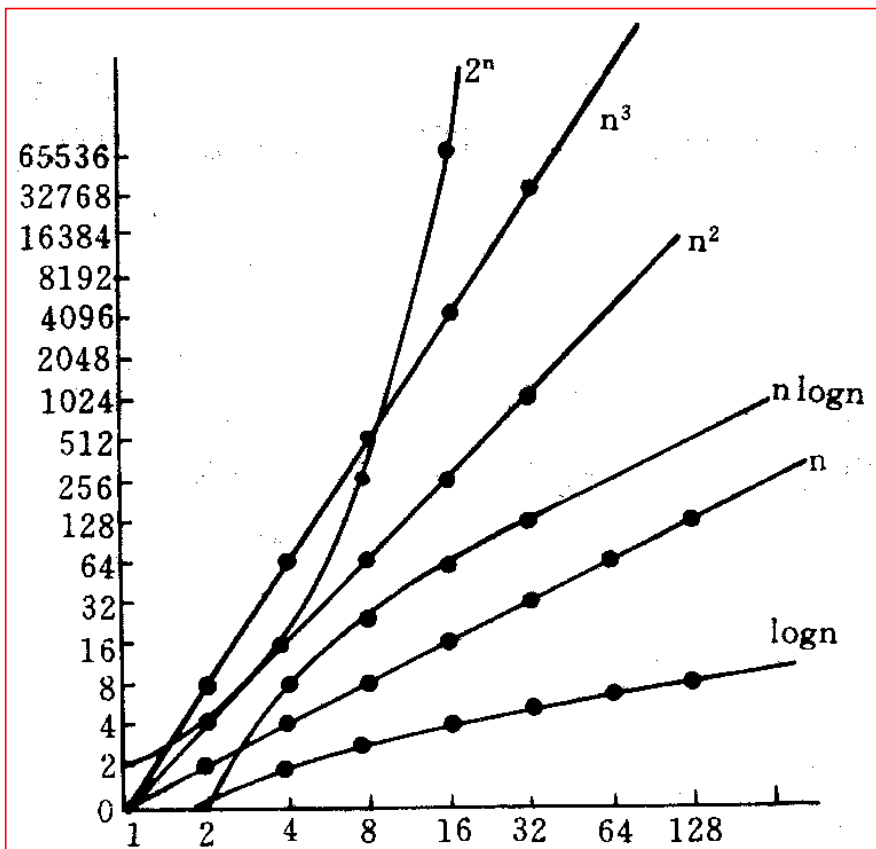


图 1.1 一般计算时间函数的曲线

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



应对复杂性的认识

- 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。



标准符号与常用函数



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 $f(m) > f(n)$.



向下取整和向上取整

- 对于所有的实数 x ,

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

- 对于任意的整数 n ,

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

- 对于任意的实数 $x \geq 0$ 和整数 $a, b > 0$,

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



模运算

如果 $(a \bmod n) = (b \bmod n)$, 我们写作 $a \equiv b(\bmod n)$ 称作 模 n 时 a 等价于 b , 或 a 、 b 同余。

多项式

对于一个非负的整数 d , 幂为 d 的多项式 n 是函数 $p(n)$

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

其中, 常数列 a_0, a_1, \dots, a_d 是多项式的系数, 并且 $a_d \neq 0$ 。



多项式

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

- 当且仅当 $a_d > 0$ 时，多项式渐近为正
- 对于d次幂的渐近正多项式 $p(n)$ ，我们有 $p(n) = \Theta(n^d)$.
- 当且仅当存在常数 k 满足 $f(n) = O(n^k)$ 时， $f(n)$ 多项式有界。



指数

- 对于所有的实数 $a > 0$, m , 和 n , 我们有以下特性:

$$a^0 = 1,$$

$$a^1 = a,$$

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

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

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

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



指数

- 对于所有的实数 a 和 b , 使得 $a > 1$ 有以下特性:

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

由此我们可以得出 $n^b = O(a^n)$.

任何底数严格大于1的指数函数都比任何多项式函数增长得快!

对数

➤ 对于所有的实数 $a > 0$, $b > 0$, $c > 0$, 和 n 有以下特性:

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

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

底数不能为1



阶乘

- 阶乘函数的弱上界是 $n! \leq n^n$
- 斯特林近似公式

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

由此可得：

$$n! = o(n^n),$$

$$n! = \omega(2^n),$$

$$\lg(n!) = \Theta(n \lg n)$$