華中科技大學
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
2023秋季-华科-计算机
22级CS

算法分析与设计
第三章
函数的增长

目　录

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：

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

- 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))

  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。

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：

華中科技大學
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？

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：

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

- 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

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

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**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：

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

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

  f(n)=Θ(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

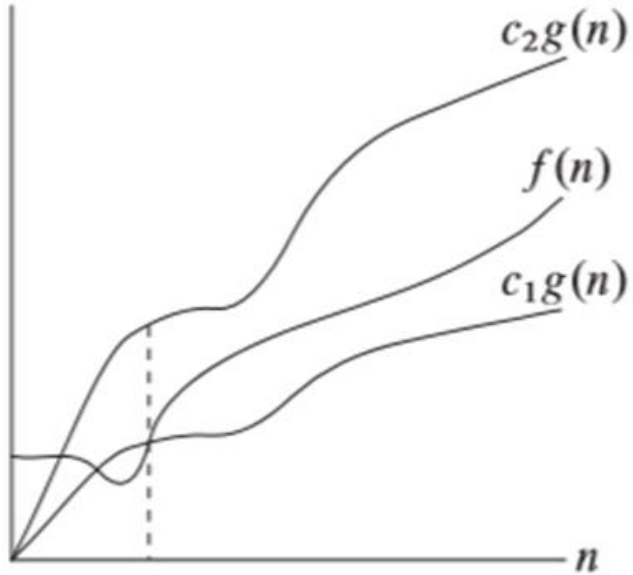
$\Theta(g(n))$表示以下函数的集合：

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

- 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：

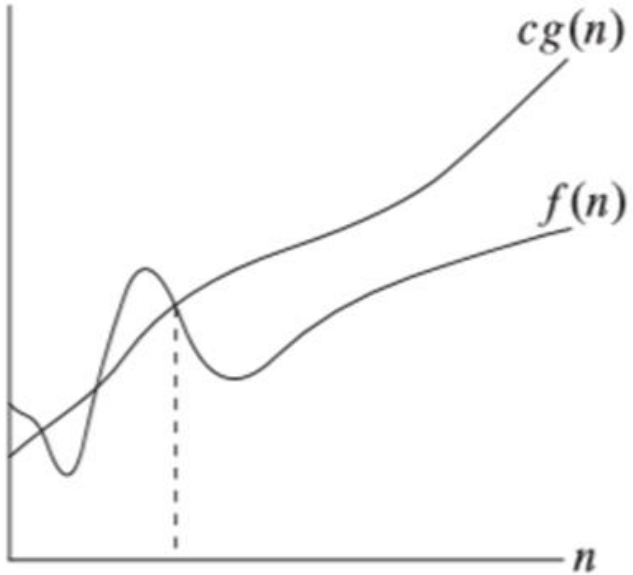
既有 f(n) = Ω(g(n))，又有f(n) = O(g(n))

定义图例



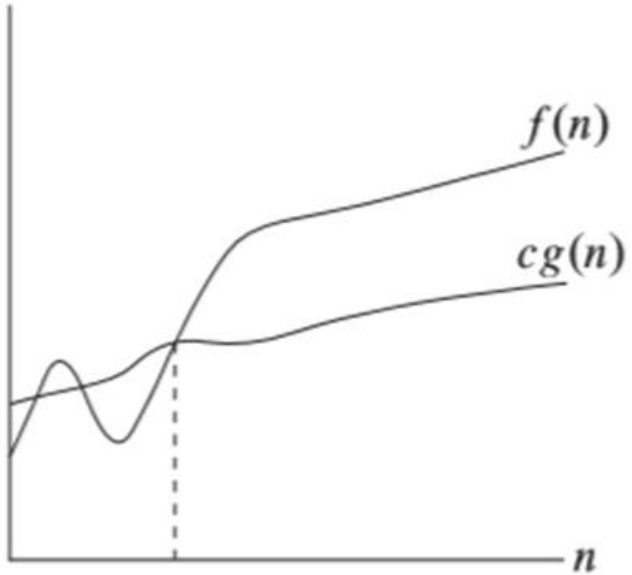
$f(n) = \Theta(g(n))$

(a)



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

(b)



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

(c)

总结：

華中科技大學
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：

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

- 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

总结：

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

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

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

渐近紧确界Θ $\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.[1]

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：$6n^3 \leq c_2n^2$。

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Θ(1)。
- 通常用Θ(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，

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$n \geq n_0$，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，

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

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

限界函数的定义

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

Ω和ω

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

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

限界函数的性质

传递性

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

限界函数的性质

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

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

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

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

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

证毕。

定理1.1 多项式定理

若$A(n)=a_m n^m+\ldots+a_1 n+a_0$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：$\log n = O(n)$ 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，$2n^3+4n^2\log n = O(n^3)$

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

时间复杂度分类

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

时间复杂度分类

- 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

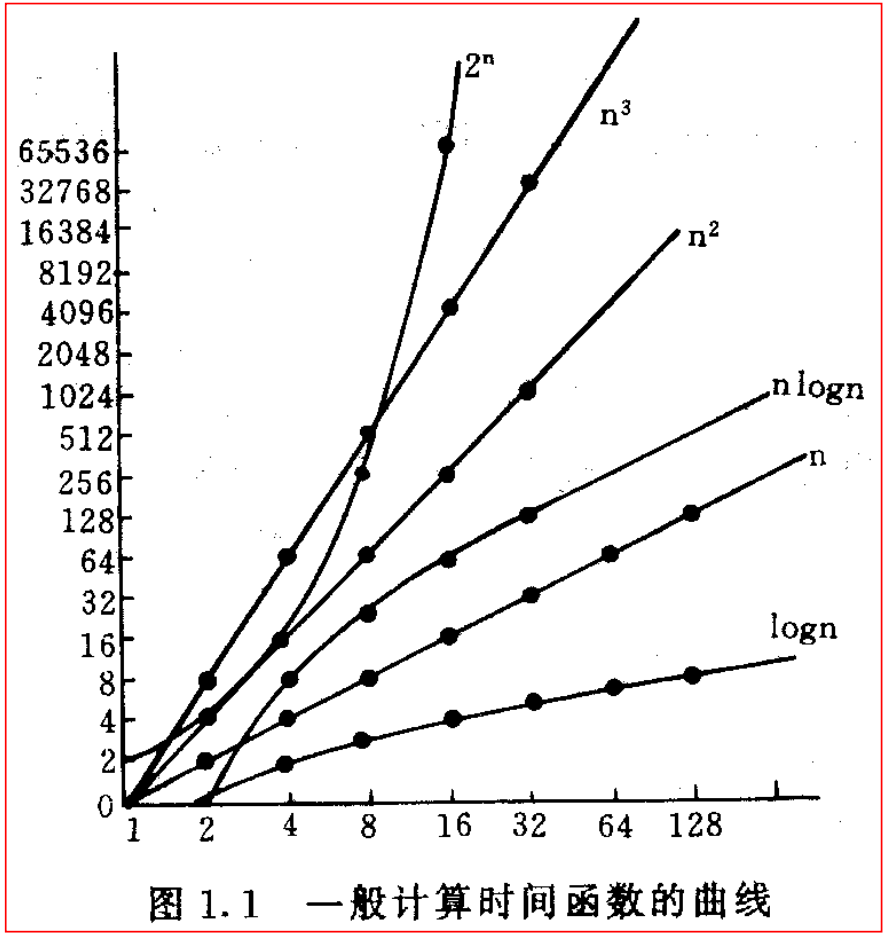


图 1.1 一般计算时间函数的曲线

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。


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

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 ≤ n implies f(m) ≤ f(n).
- A function f(n) is ***monotonically decreasing （单调递减）*** if m ≤ n implies f(m) ≥ 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≥0 和整数 a, b > 0,

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

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

模运算

如果 (a mod n) = (b mod n), 我们写作 a ≡ b(mod 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, ... ,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：

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

指数

- 对于所有的实数a和b，使得 a > 1有以下特性：

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

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：

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

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

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：

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