算法设计与分析

主讲人：吴庭芳

Email: tfwu@suda.edu.cn

苏州大学

计算机科学与技术学院







第二讲 函数增长

内容提要：

- □ 渐进记号
  - ✓ 定义：O，Ω，Θ，o，ω
  - ✓ 证明例子
- □ 常用函数

限界函数

- 限界函数：取自频率计算函数表达式中的最高次项，并忽略常系数，记为 $g(n)$
  - $g(n)$ 是关于 n 的形式简单的单项式函数，如 $n\lg n$
  - $g(n)$ 是对算法中最复杂的计算部分分析而来的
- 算法时间复杂度的三种常用的限界函数：

  上界函数、下界函数、紧确界函数

  对应的渐进记号：　O　　　Ω　　　Θ

  用于分别界定算法的时间复杂度上界、下界和紧确界

频率计算函数表达式：

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n}t_j+c_6\sum_{j=2}^{n}(t_j-1)+c_7\sum_{j=2}^{n}(t_j-1)+c_8(n-1)$$

限界函数







- 算法的实际执行时间为 $f(n)$，分析所得到的限界函数为 $g(n)$
  - n：问题实例的输入规模
  - $f(n)$：算法的“实际”执行时间，**与机器及语言有关**
  - $g(n)$：是事前分析估算的结果，一个形式简单的函数，通过计算对运行时间有消耗的基本操作的执行次数得到的，**与机器及编译语言无关**

渐近上界-O记号

- **O($g(n)$) 描述一个函数 $f(n)$ 的渐近上界，表示以下函数集合：**

$$O(g(n)) = \{\ f(n)\text{: 存在正常量 } c \text{ 和 } n_0\text{，使得对所有 } n \ge n_0 \text{ 有 } 0 \le f(n) \le cg(n)\ \}$$

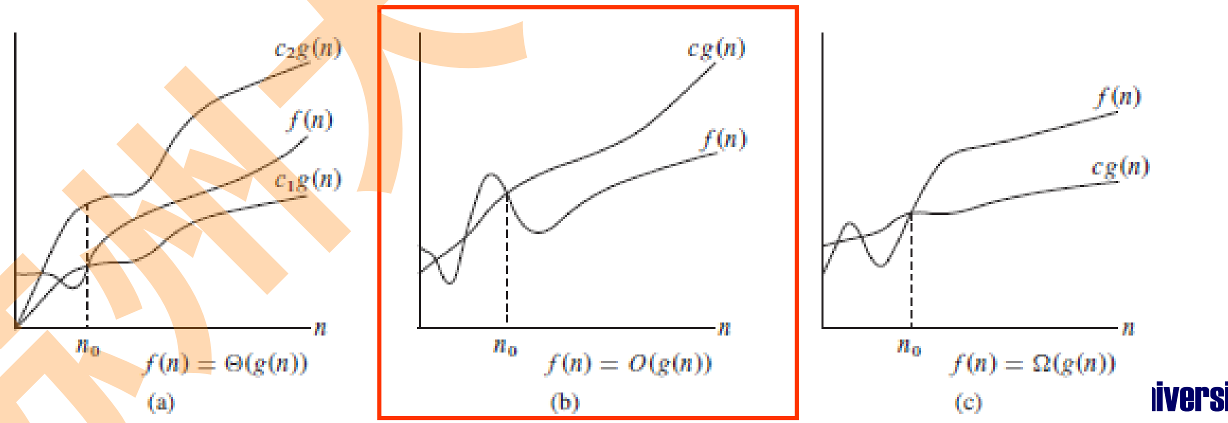
- 若 $f(n)$ 和 $g(n)$ 满足以上关系，记为 $f(n) \in O(g(n))$，表示 $f(n)$ 是集合 $O(g(n))$ 中的一员。通常记为：

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

- $f(n) = O(g(n))$ 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）在任意一台机器上运行，所用的时间总小于 $|g(n)|$ 的一个常数倍

渐近上界-O记号

- 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在**最坏情况下**的时间复杂度，隐含地给出了在任意输入下运行时间的上界
- 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)**
  - ✓ 例：若：$3n+2=O(n^2)$，则是松散的界限
  - ✓ 若：$3n+2=O(n)$，则是紧确的界限



渐近下界-Ω记号







- **$\Omega(g(n))$ 描述一个函数 $f(n)$ 的渐近下界，表示以下函数集合：**

  $\Omega(g(n))$ = { $f(n)$: 存在正常量 c 和 n_0，使得对所有 $n \geq n_0$,

  有$0 \leq cg(n) \leq f(n)$ }

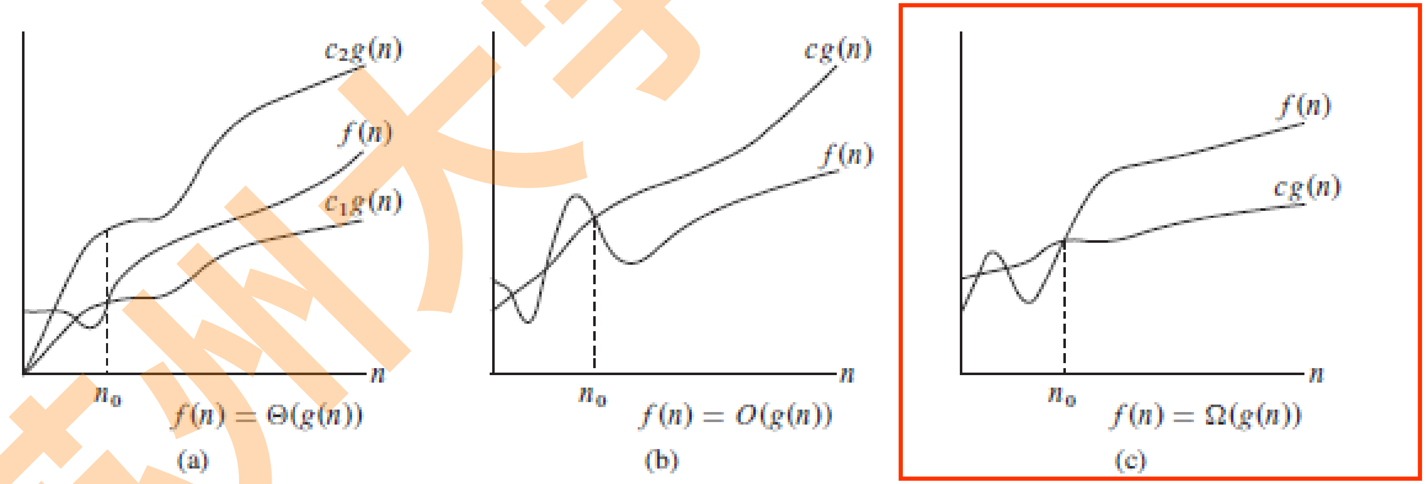
  - 若 $f(n)$ 和 $g(n)$ 满足以上关系，记为 $f(n) \in \Omega(g(n))$，表示 $f(n)$是 $\Omega(g(n)$ 中的一员。通常记为：

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

  - $f(n) = \Omega(g(n))$ 表示如果算法用 n 值不变的同一类数据在任意一台机器上运行，所用的时间总不小于 $|g(n)|$ 的一个常数倍

渐近下界-Ω记号

- Ω 记号给出一个**渐进下界**，称为**下界函数** (lower bound)
- 下界函数代表了算法在**最佳情况下**的时间复杂度，隐含地给出了在任意输入下运行时间的下界
- 在确定下界函数时，应试图找出**阶最大**的 $g(n)$ 作为 $f(n)$ 的下界函数——**紧确下界** (tight lower bound)



渐近紧确界-Θ记号







- **$\Theta(g(n))$ 描述一个函数 $f(n)$ 的渐近紧确界，表示以下函数集合：**

  $\Theta(g(n)) = \{ f(n)$: 存在正常量 c_1，c_2，和 n_0，使得对所有 $n \geq n_0$，有 $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$

  - 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 $\Theta(g(n))$ 中的一员。通常记为：

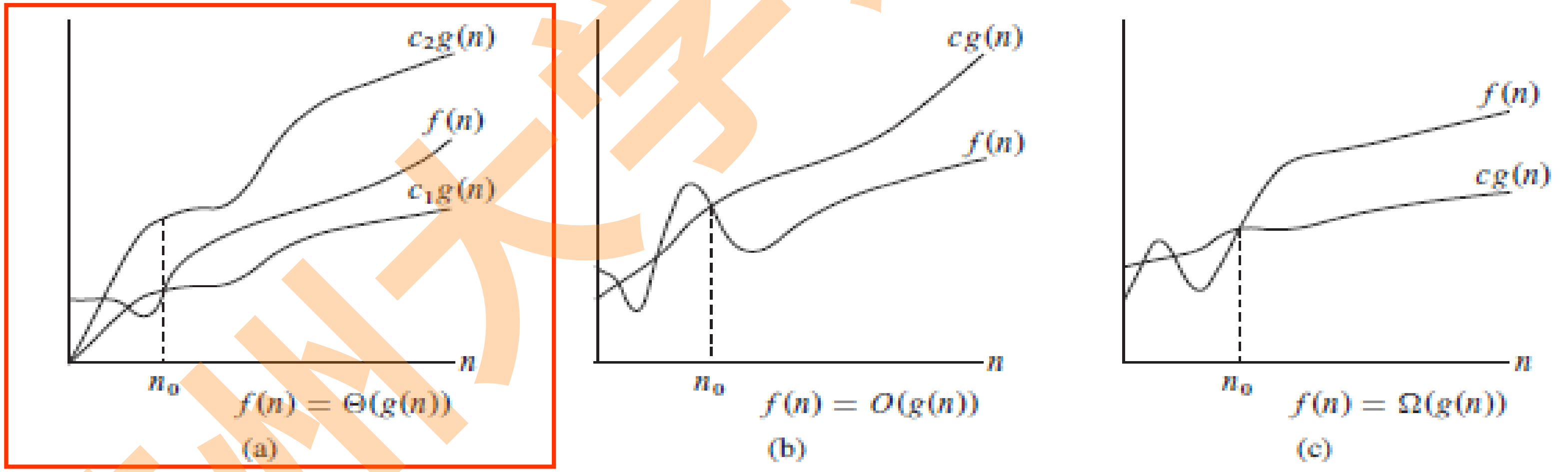
    $$\boldsymbol{f(n) = \Theta(g(n))}$$

  - $f(n) = \Theta(g(n))$ 表示如果算法用 n 值不变的同一类数据在任意一台机器上运行，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

渐近紧确界-Θ记号

- Θ 记号给出算法的**精确复杂度**，既给出上界也给出下界
- 从时间复杂度的角度看，$f(n) = \Theta(g(n))$ 表示算法在最佳和最坏情况下的运行时间就一个**常数因子范围内而言**是相同的，可看作：

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



渐近紧确界-Θ记号

例 1：证明 $1/2n^2-3n = \Theta(n^2)$

分析： 根据 Θ 的定义，存在正常量 c_1， c_2 和 n_0，使得对所有 $n \geq n_0$，有 $0 \leq c_1n^2 \leq 1/2n^2-3n \leq c_2n^2$

证明： 两边同时除以 n^2 得：$c_1 \leq 1/2-3/n \leq c_2$

选择任意常量 $c_2 \geq 1/2$，使得右边不等式对于任何 $n \geq 1$ 成立；

选择任意常量 $c_1 \leq 1/14$，使得左边不等式对于任何 $n \geq 7$ 成立。

因此，通过选择 $c_1 = 1/14$，$c_2 = 1/2$， $n_0= 7$，得证 $1/2n^2-3n = \Theta(n^2)$

Note： 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）即可得证

渐近紧确界-Θ记号

例 2：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 $n \geq n_0$，有

$6n^3 \leq c_2 n^2$

两边同时除以 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

渐近紧确界-Θ记号

- **关于 $\Theta(1)$ 的含义（$O(1)$、$\Omega(1)$ 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 没有关系

渐近紧确界-Θ记号







- **定理 3.1 对任意两个函数 $f(n)$ 和 $g(n)$，我们有 $f(n) = \Theta(g(n))$，当且仅当 $f(n) = O(g(n))$ 且 $f(n) = \Omega(g(n))$**

证明：

$$
\begin{aligned}
\Rightarrow:\ & f(n)=\Theta(g(n)),\ \text{then } \exists c_1>0, c_2>0, n_0>0,\\
& \text{s.t. } n \ge n_0\ ,\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n)\\
& \text{then } n \ge n_0,\ 0 \le f(n) \le c_2 g(n) \Rightarrow f(n)=O(g(n))\\
& \text{then } n \ge n_0,\ 0 \le c_1 g(n) \le f(n) \Rightarrow f(n)=\Omega(g(n))\\
\Leftarrow:\ & f(n)=O(g(n)),\ \text{then } \exists c_2>0, n_{20}>0,\\
& \text{s.t. } n \ge n_{20},\ 0 \le f(n) \le c_{20} g(n)\\
& f(n)=\Omega(g(n)),\ \text{then } \exists c_{10}>0, n_{10}>0,\\
& \text{s.t. } n \ge n_{10},\ 0 \le c_{10} g(n) \le f(n)\\
& \text{let } n_0=\max\{n_{10}, n_{20}\},\ \text{then } n \ge n_0,\\
& 0 \le c_{10} g(n) \le f(n) \le c_{20} g(n),\ \text{that is } f(n)=\Theta(g(n)).
\end{aligned}
$$

等式和不等式中的渐近记号

- **如 $n = O(n^2)$，$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 等，如何来解释这些公式呢?**
  - 当渐近记号出现在等式的右边时，则等号表示左边的函数属于右边函数集合中的元素，即等号表示集合的成员关系，即 $n \in O(n^2)$
  - 当渐近记号出现在某个公式中时，将其解释为某个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。例如公式 $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$，$\Theta(n)$ 表示一个与 n 成正比的项目（包括线性项和常数项）。这里，左边的 $3n+1$ 部分被归入 $\Theta(n)$，表达式中的主导项是 $2n^2$，而较低阶的项（线性项和常数项）被压缩在 $\Theta(n)$ 中。这种表示法常用于简化复杂度分析，突出主要增长趋势

等式和不等式中的渐近记号

- **如 $n = O(n^2)$，$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ 等，如何来解释这些公式呢?**
  - 当渐近记号出现在等式左边时，如 $2n^2 + \Theta(n) = \Theta(n^2)$，用如下规则来解释：无论等号左边的匿名函数如何选择，总有办法选取等号右边的匿名函数使等式成立。这样，对于任意函数 $f(n) \in \Theta(n)$，存在某个函数 $g(n) \in \Theta(n^2)$，使得对所有的 n，有 $2n^2 + f(n) = g(n)$ 成立。换而言之，等式右边提供了较左边更少的细节，忽略低阶项和高阶项常系数这些“噪音”，专注于由算法本质决定的增长率

非渐近紧确上界-o记号

- **大 O 记号所提供的渐近上界可能是、也可能不是渐近紧确的；例如 $2n^2 = O(n^2)$ 是渐近紧确的，但 $2n = O(n^2)$ 却不是**
- **利用小 o 记号来表示非渐近紧确的上界，其定义如下：**

  $o(g(n)) = \{ f(n)$：对任意正常量 $c > 0$，存在常量 $n_0 > 0$，使得对所有 $n \geq n_0$，有 $0 \leq f(n) < cg(n) \}$

- **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 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- **例如 $2n = o(n^2)$，但 $2n^2 \neq o(n^2)$**

非渐近紧确下界-ω记号







- **ω 记号与 Ω 记号类似于 o 记号与 O 记号的关系**
- **利用小 ω 记号来表示非渐近紧确的下界，其定义如下：**

  $\omega(g(n)) = \{ f(n)$：对任意正常量 $c > 0$，存在常量 $n_0 > 0$，使得对所有 $n \geq n_0$，有 $0 \leq cg(n) < f(n) \}$

- **含义：**在 ω 表示中，当 n 趋于无穷时，$f(n)$ 相对于 $g(n)$ 来说变得无穷大了，即 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$
- **例如 $n^2/2 = \omega(n)$，但 $n^2/2 \neq \omega(n^2)$**







非渐近紧确下界-ω记号

- **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:** $f(n) = o(g(n)) \Leftrightarrow \forall\ c: (\exists n_0: n \geq n_0 \Rightarrow f(n) < cg(n))$
- **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:** $f(n) = \omega(g(n)) \Leftrightarrow \forall\ c: (\exists\ n_0: n \geq n_0 \Rightarrow cg(n) < f(n))$

限界函数的性质







- **实数的许多关系性质也适用于渐近比较。下面假定 $f(n)$ 和 $g(n)$ 渐近为正**
- **传递性 (Transitivity)**

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

- **自反性 (Reflexivity)**

$$f(n)=\Theta(f(n)),$$
$$f(n)=O(f(n)),$$
$$f(n)=\Omega(f(n)).$$

限界函数的性质

- **对称性 (Symmetry)**

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

- **转置对称性 (Transpose Symmetry)**

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

限界函数的性质







□ **函数渐近性比较与实数比较的类比**

$$f(n)=o(g(n)) \ \approx \ a<b,$$

$$f(n)=O(g(n)) \ \approx \ a\le b,$$

$$f(n)=\Theta(g(n)) \ \approx \ a=b,$$

$$f(n)=\Omega(g(n)) \ \approx \ a\ge b,$$

$$f(n)=\omega(g(n)) \ \approx \ a>b.$$

第二讲 函数增长

内容提要：

- □ 渐进记号
- □ 常用函数







相关定理

- **定理 3.2 多项式定理：**若 $A(n)=a_m n^m+\cdots+a_1 n+a_0$ 是一个 n 的 m 次项式，其中 a_i 为常量，$i=0$，…，m，且 $a_m>0$，则有 $A(n)=\Theta(n^m)$

证明：需要证明 $c_1 n^m \leqslant a_m n^m+\cdots+a_1 n+a_0 \leqslant c_2 n^m$

取 $n_0=1$，当 $n\geq n_0$ 时，有

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

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

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

令 $c_2=|a_m|+|a_{m-1}|+\cdots+|a_0|$，

即有 $|A(n)|\leq c_2 n^m=O(n^m)$

相关定理







证明： 需要证明 $c_1 n^m \leqslant a_m n^m + \cdots + a_1 n + a_0$。

$$
\begin{aligned}
\mathrm{A}(n) &= a_m n^m + \cdots + a_1 n + a_0 \\
&= \frac{1}{2} a_m n^m + \frac{1}{2} a_m n^m + a_{m-1} n^{m-1} + \cdots + a_0 \\
&= \frac{1}{2} a_m n^m + \left(\frac{a_m}{2m} n^m + a_{m-1} n^{m-1} \right) \\
&\quad + \left(\frac{a_m}{2m} n^m + a_{m-2} n^{m-2} \right) \\
&\quad + \cdots + \left(\frac{a_m}{2m} n^m + a_0 \right)
\end{aligned}
$$







相关定理

证明： $\because\ a_m > 0$， 对于足够大的 n，有

$$\frac{a_m}{2m} n^m + a_{m-1} n^{m-1} \geq 0$$

$$\frac{a^m}{2m} n^m + a_{m-2} n^{m-2} \geq 0$$

$$\vdots$$

$$\frac{a^m}{2m} n^m + a_0 \geq 0$$

$\therefore$ 对于足够大的 n，有 $A(n) \geq \frac{1}{2} a_m n^m = \Omega(n^m)$，

取 $c_1 = 1/2a_m$

相关定理

- **应用：** 如果一个算法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

$$A(n)=a_m n^m+\cdots+a_1 n+a_0 \longrightarrow A(n)=\Theta(n^m)$$

- 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：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

相关定理

- **定理 3.3 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 $a>0$，$ad(n)=O(f(n))$
  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. 对于任意固定的实常量 b 和 $a > 1$，n^b 是 $o(a^n)$ —— 任意底大于 1 的指数函数比任意多项式函数增长得快
  5. 对于任意固定的常数 $b > 0$ 和 $a > 0$，$\lg^b n$ 是 $o(n^a)$ —— 任意正的多项式函数都比任意多对数函数增长得快

算法时间复杂度的分类

- 根据**限界函数**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：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：

$O(1) < O(\lg n) < O(n) < O(n\lg n) < O(n^2) < O(n^3)$ 复杂度越来越高

  - **指数时间算法：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

算法时间复杂度的分类

- 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

计算时间的典型函数曲线：

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

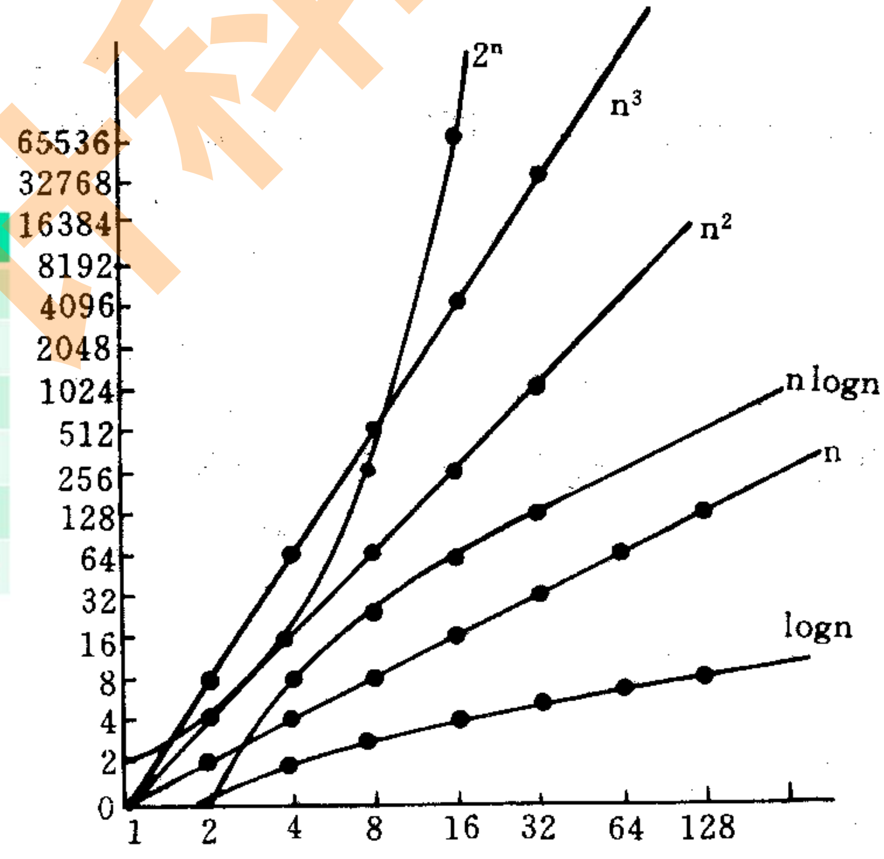


图 1.1 一般计算时间函数的曲线

谢 谢！