数据结构与算法设计

周　可

Mail：zhke@hust.edu.cn

华中科技大学，武汉光电国家研究中心

Preface

n次多项式求解——秦九韶算法

$$
\begin{aligned}
f(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \\
&= (a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_1)x + a_0 \\
&= ((a_n x^{n-2} + a_{n-1} x^{n-3} + \cdots + a_2)x + a_1)x + a_0 \\
&= \cdots\cdots \\
&= (\cdots(a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0
\end{aligned}
$$

秦九韶算法求解n次多项式只需**n次乘法**和**n次加法**

算法时间复杂度从$O(n^2)$降低到$O(n)$

Analysis of algorithms

The theoretical study of computer-program performance and resource usage.

- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.

Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Kinds of analyses

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on *some* input.

Machine-independent time

What is insertion sort's worst-case time?

- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

BIG IDEA:

- Ignore machine-dependent constants.
- Look at ***growth*** of $T(n)$ as $n \to \infty$.

"Asymptotic Analysis"

Asymptotic notation(Θ,O,Ω)

Asymptotic notation can be used to characterize the running times of algorithms.

Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

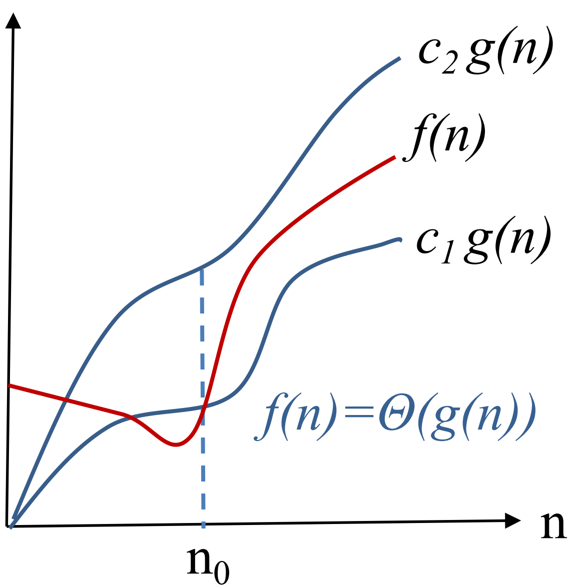
Θ-notation

Math:

$\Theta(g(n)) = \{ f(n) :$ 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 $n \ge n_0 \}$

- A function f (n) belongs to the set Θ(g(n)) if there exist positive constants c_1 and c_2 such that it can be "sandwiched" between $c_1g(n)$ and $c_2g(n)$, for sufficiently large n.
- f (n)∈Θg(n)) : indicate that f(n) is a member of Θ(g(n))
- Instead, we use f(n) = Θ(g(n)) to express the same notion.

- That is , for all $n \geq n_0$, the function f(n) is equal to Θ(g(n)) within a constant factor.
- Here, we say that g(n)) is an **asymptotically tight bound** (渐进紧确界) for f(n).



(a)

We write **$f(n) = \Theta(g(n))$** if there exists 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 $c_1 g(n)$ and $c_2 g(n)$ inclusive**.

- *Example* *Prove* $\frac{1}{2}n^2-3n=\Theta(n^2)$
- To do so, we must determine positive constants c_1, c_2, and n_0 such that

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2 \quad \text{for all } n \ge n_0 .$$

- Dividing by n^2 yields

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2 .$$

- Here, choosing $c_1 = 1/14$, $c_2 = 1/2$, and $n_0 = 7$, we can prove it.

- *How from an* asymptotically positive function(渐进正函数) to asymptotically **tight bounds**（渐进紧确界）
  - Ignore the lower-order terms and the constant, just leave the **highest-order term**, and ignore the coefficient of the highest-order term, then we get the asymptotically tight bound formula:
  - Example: $f(n) = an^2 + bn + c = \Theta(n^2)$
- In general, for any polynomial $p(n) = \sum_{i=0}^{d} a_i n^i$ where the a_i are constants and $a_d > 0$, we have

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

[定理4.1 大Θ比率定理]对于函数$f(n)$和$g(n)$，若$\lim_{n\to\infty}\frac{g(n)}{f(n)}$和$\lim_{n\to\infty}\frac{f(n)}{g(n)}$都存在，则$f(n) = \Theta(g(n))$，当且仅当存在确定的常数$c_1$和$c_2$，有$\lim_{n\to\infty}\frac{g(n)}{f(n)} \le c_1$ 和$\lim_{n\to\infty}\frac{f(n)}{g(n)} \le c_2$。

O-notation

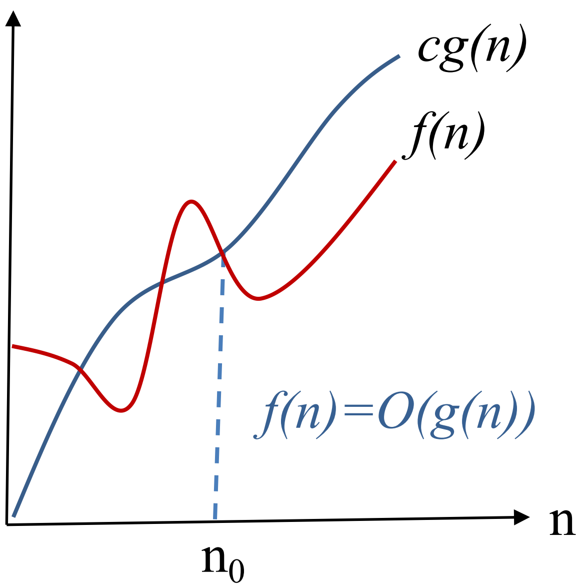
Math:

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

- O-notation describes an **upper bound**. We use it to bound the **worst case** running time of an algorithm,
- If f(n) = O(g(n)), O(g(n)) is an asymptotic upper bound on f(n).
- f(n) = Θ(g(n)) implies f(n) = O(g(n))

Math:

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



(b)

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)).

[定理4.2 大O比率定理]对于函数*f(n)*和*g(n)*，若 $\lim_{n\to\infty}\frac{f(n)}{g(n)}$ 存在，则$f(n)=O(g(n))$，当且仅当存在确定的常数*c*，有 $\lim_{n\to\infty}\frac{f(n)}{g(n)} \le c$。

证明：

1）*如果*$f(n)=O(g(n))$，则存在$c>0$及某个n_0，使得对于所有的$n\ge n_0$，有$f(n)/g(n)\le c$，因此 $\lim_{n\to\infty}\frac{f(n)}{g(n)} \le c$。

2）假定 $\lim_{n\to\infty}\frac{f(n)}{g(n)} \le c$，它表明存在一个$n_0$，*使得对于所有的*$n\ge n_0$，有$f(n)\le c\,g(n)$。证毕。

e.g. $\lim_{n\to\infty}\frac{5n+2}{n}=5$ ⟹ $5n+2 = O(n)$

$\lim_{n\to\infty}\frac{7n^2+5n+2}{n^2}=7$ ⟹ $7n^2+5n+2 = O(n^2)$

$\lim_{n\to\infty}\frac{5\times 2^n+n^2}{2^n}=5$ ⟹ $5\times 2^n+n^2 = O(2^n)$

考察下式:

因为 $\lim_{n\to\infty}\frac{n^9+3n^2}{2^n}=0$, 所以 $n^9+3n^2 = O(2^n)$

是否合适？显然这不是一个好的上界估计。

原因在于：极限值不是一个正常数（见O的定义）。

Ω-notation

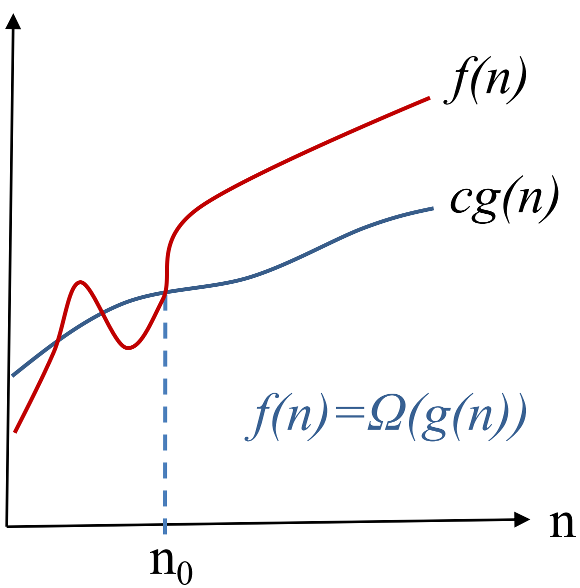
- ***Math:***

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

- Ω-notation gives a ***lower bound*** on the ***best-case*** running time of an algorithm.
- When we say that the running time of an algorithm is Ω(g(n)), we mean that no matter what particular input of size n is chosen for each value of n, the running time on that input is at least a constant times g(n), for sufficiently large n.

Math:

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



(c)

We write **f (n) = Ω(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).

[定理1.3 大Ω比率定理] 对于函数$f(n)$和$g(n)$，若 $\lim_{n\to\infty}\frac{g(n)}{f(n)}$ 存在，

则 $f(n)=\Omega(g(n))$，当且仅当存在确定的常数c，有

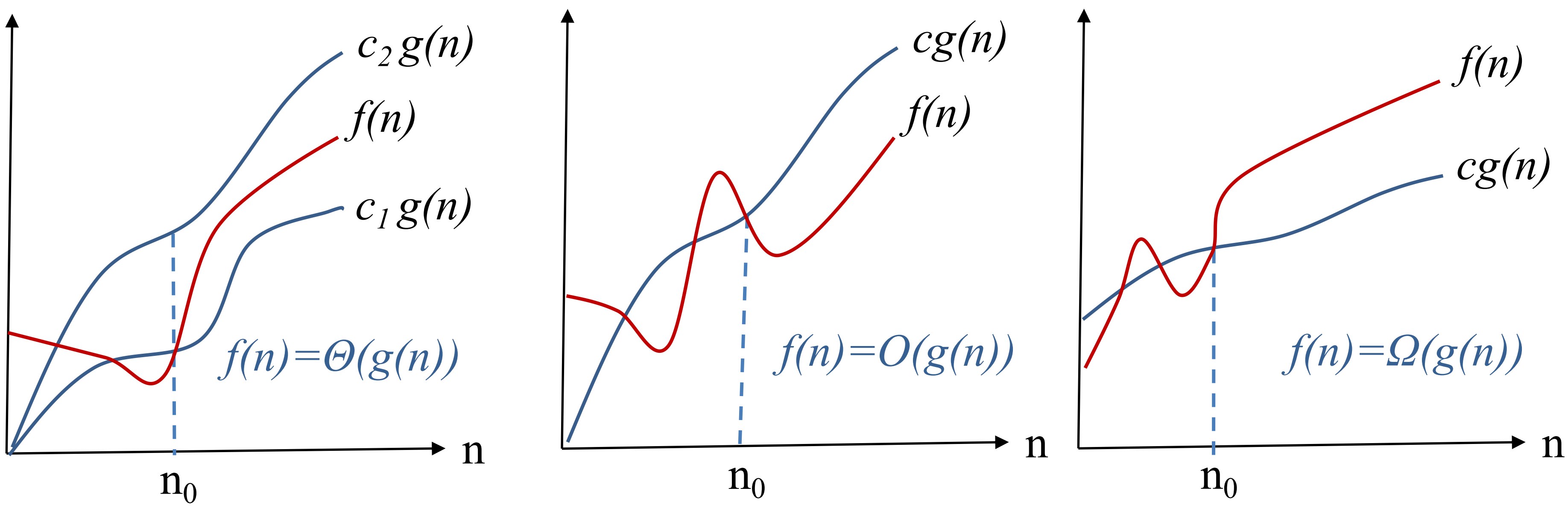
$\lim_{n\to\infty}\frac{g(n)}{f(n)} \leq c$。

注：

这里，当n充分大时，$g(n) \leq cf(n)$意味着 $f(n)\geq\frac{1}{c}g(n)$ ，

由此不难看出上述判别规则的正确性。

Comparison of Asymptotic notations (Θ,O,Ω)



(a) (b) (c)

o-notation

- o denotes an upper bound that is NOT asymptotically tight.

$$o(g(n)) = \{f(n): for\ any\ positive\ constant\ c > 0,\ there\ exists\ a\ constant\ n_0 > 0\ such\ that\ 0 \le f(n) < cg(n)\ \ for\ \ all\ n \ge n_0\}$$

  – For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$.

- The main difference between O-notation and o-notation is that in f(n) =O(g(n)), the bound 0≤ f(n) ≤ c(g(n)) holds for ***some* constant c > 0**, but in f(n) =o(g(n)), the bound 0≤ f(n) < c(g(n)) holds for ***all* constants c** > 0.

- Intuitively, in o-notation, the function f(n) becomes insignificant relative to g(n) as n approaches infinity; that is,

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

O和o的区别

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

o: $f(n)=o(g(n)) \Leftrightarrow \boxed{\forall c}, \exists n_0 : (n \geq n_0 \Rightarrow f(n) < cg(n))$

在o表示中，当n趋于无穷时，f(n)相对于g(n)来说已经不重要了。

5. ω-notation

- ω denotes a lower bound that is NOT asymptotically tight.

$$\omega(g(n)) = \{f(n): for\ any\ positive\ constant\ c > 0,\ there\ exists\ a\ constant\ n_0 > 0\ such\ that\ 0 \le cg(n) < f(n)\ \ for\ \ all\ n \ge n_0\}$$

For example, $n^2/2 = \omega(n)$, but $n^2/2 \neq \omega(n^2)$.

- The relation $f(n) = \omega(g(n))$ implies that

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

- That is, f(n) becomes arbitrarily large relative to g(n) as n approaches infinity.

Ω和ω的区别

Ω：

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

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

ω：

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。

Properties of those **notations**

Transitivity: 传递性

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

Reflexivity: 自反性

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

Symmetry: 对称性

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

Transpose symmetry: 转置对称性

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

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：

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

复杂性越来越高

- 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：

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

复杂性越来越高

■ 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比较

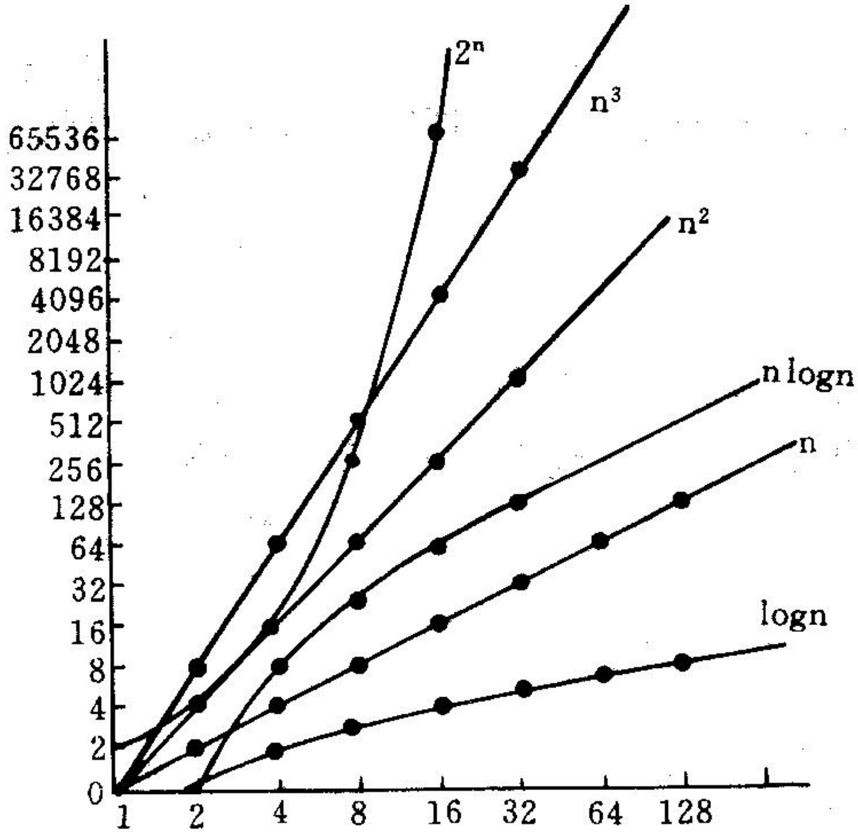


图 1.1　一般计算时间函数的曲线

□ 计算时间函数值的比较

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 (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。

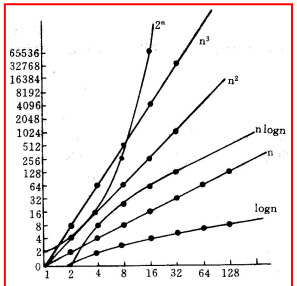


图 1.1 一般计算时间函数的曲线

作业

- 1）3.1-1
- 2）3.1-2
- 3）3.1-4

Thank You!

Q&A