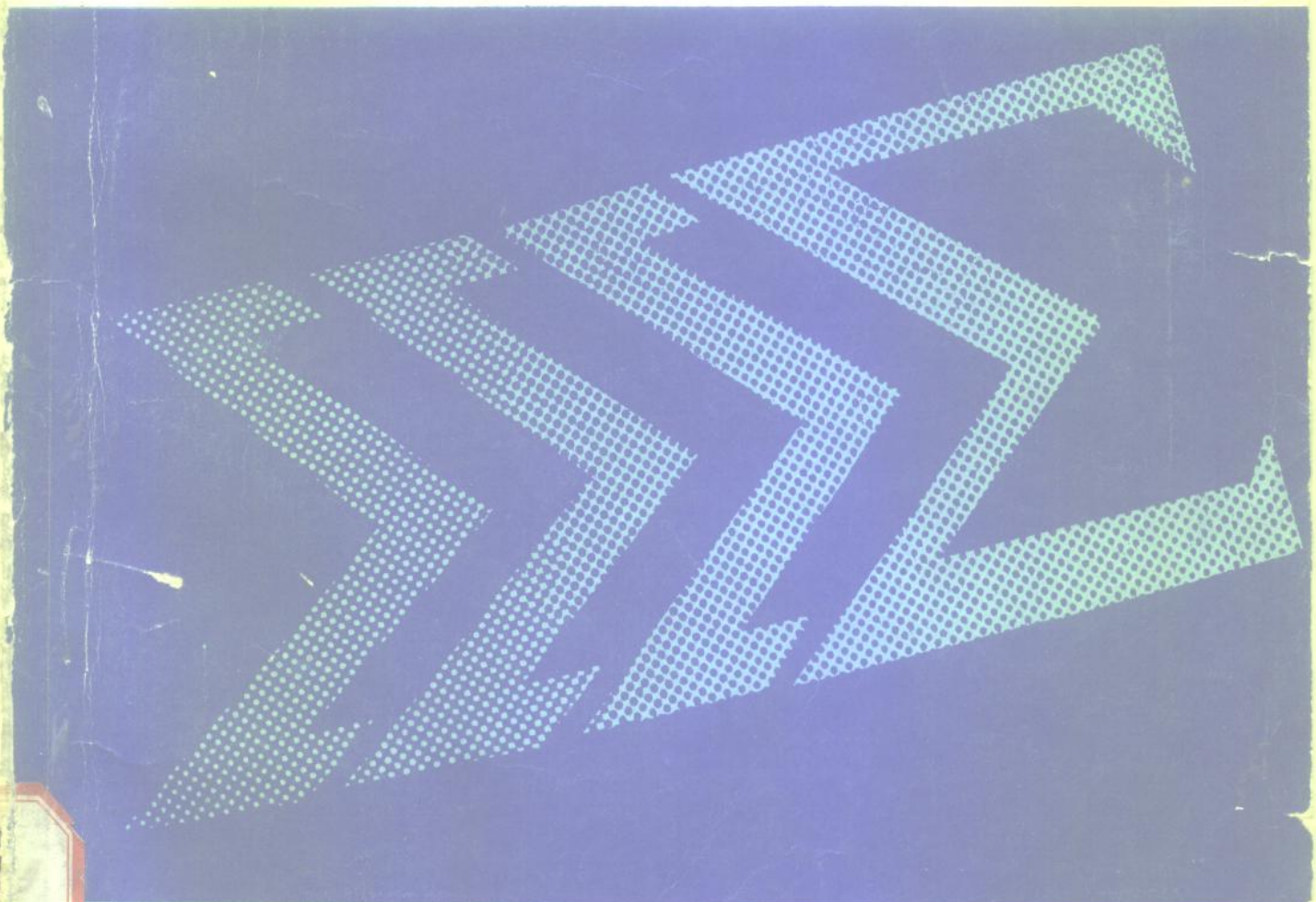
算法设计和分析

朱　洪　陈增武　段振华　周克成编著



上海科学技术文献出版社

算法设计和分析

朱 洪 陈增武 段振华 周克成 编著

上海科学技术文献出版社

算法设计和分析
朱 洪 陈增武 段振华 周克成 编著
*
上海科学技术文献出版社出版发行
(上海市武康路2号)
新华书店经销
昆山亭林印刷厂印刷
*
开本 787×1092 1/16 印张 16.5 字数 418,000
1989年9月第1版 1989年9月第1次印刷
印数: 1—4,000
ISBN 7-80513-395-6/T·134
定价: 8.50元
《科技新书目》195-255

序　言

本书主要介绍如何设计和分析离散系统中出现的各种问题的计算机解法。作者期望读者通过本书的学习，不仅能掌握一些常用算法，而且具有设计有效算法，分析算法时间和空间效率的初步能力。主要对象是大学高年级学生和同等水平的计算机工作者。

近十余年来，算法研究成果和发表的论文极为丰富，要想在这么一本书中概括全貌是不可能的。为了使读者能较快地进入算法研究的前沿，作者在各章末写了注解和参考文献，目的是使读者了解某些问题的新发展。

本书还力求反映中国学者在计算机算法领域中的研究成果。中国人对数学中的算法研究是有悠久历史的；近年来，在计算机理论方面有一些比较先进的研究成果。作者限在算法设计和分析的范围内把一些成果写进本书。由于作者本身水平和视野不够宽广，也许不够全面，敬请读者多提意见。以便今后能反映中国学者的成绩准确些。

如果把本书作为一学期四学时的教材，则可酌情删去第§2.5，§2.6.3，§2.7，§4.5～4.8，§5.7～5.9，§7.4～7.5，§10.3等节内容的全部或部分。

本书的初稿是朱洪于1983年夏季在北京工业大学第二分校（现北京计算机学院），由中国数学会举办的计算机科学理论讲习班上的讲义。后来陈增武、周克成和段增华通过教学实践进行改写。其中第二、五和七章由段增华执笔，第四和六章由周克成执笔，第八和九章由陈增武执笔，第一、三、十和十一章由朱洪重写。最后由朱洪审定全稿，并适当增删内容，还加写了各章注解和参考文献和大部分章节的习题。

顾强协助作者写第十一章，王君英打印本书所有程序，赵惠莲绘画本书大部分插图，胡美琛，吴京，臧斌宇，林国强等对本书原稿提过不少意见，作者在此表示感谢。

编著者

1987年11月

目　录

算法一览表

第一章　算法设计和分析的原则

§1.1　引　　言

什么是一个算法？形式地说，任何一个对所有有效输入总要停机的图灵机是一个算法。为着本课程的目的，我们宁愿取下述非形式的一种描述：算法是一个有限条指令的集合。这些指令确定了解决某一问题的运算或操作的序列。每条指令必须是确定的与可行的，即每条指令必须清楚明了，无二义性，例如不允许"add 6 or 7 to x"，每条指令还必须充分基本，使得可在有限时间内完成，例如实数的算术运算未必可行，因为可能产生无限长的十进小数。此外，一个算法还必须满足有限性，即它必须在执行有限条指令后终止；$\geqslant 0$ 个输入，它是算法所要求的初始信息量；$\geqslant 1$ 个输出，它一般被解释为对输入的计算结果。可以说，非形式的描述是形式定义所蕴涵特征的尽可能具体的刻划；形式定义是非形式描述的数学抽象。

一个问题 P 是由无穷多个实例组成的一个类。其中每个实例由问题定义域上的某个实体（俗称输入）以及对实体的提问（或加工要求）组成。

例如，整数乘法问题。问题定义域是全体整数偶对。某一实例的实体是某个数对 a, b。提问是它们的积是多少？问题的一个实例是 $2\times 4=?$

又例如，语言识别问题，问题定义域是 Σ^*——某字符集上字符串全体。某一实例的实体是某字符串 $\omega\in\Sigma^*$。提问是 $\omega\in L(G)$？这里 G 是某个给定的语法，$L(G)$ 是 G 生成的语言 $\subseteq\Sigma^*$。

必须注意，个别的实例不认为是一个问题。

一个算法 A 称为解算问题 P，或称 A 是问题 P 的一个算法，是指如果把 P 的任一实例的实体作为 A 的输入，A 在有限步之内总输出一个关于此实例的正确答案。一个问题 P 称为算法可解的，如果存在一个算法解算 P。

一个问题是算法可解的，是否就可以说它在实践中可解的呢？回答是并不尽然，因为解算这个问题的算法可能需要极大的时间与空间以致于在现有的计算机上无法计算，也即它们不是有效算法。这就导至对一个算法的定性与定量。定性是指一个算法的可行性，即它是否总在有限步内终止，至于究竟多少步则不论，反正有限步即可。定量是一个算法的有效性，即它到底在多少步内终止。这个步数必须要有限度，必须要是可以接受的，现实可行的。对于后者的研究，就导至计算复杂性这一重大课题。

算法的设计和分析主要要解决两个课题：

第一，对一个给定的问题，如何设计出解算它的有效算法？常用的设计技术是什么？

第二，试图提供用来分析，判断一个算法的质量（主要是效率的度量）的准则和技术。

表 1.1 和 1.2 说明算法的好坏对于用计算机来解算问题的能力（速度和规模）有着十分重要的作用。表中函数 $T(n)$ 是一个问题用某算法来解算时，对各种规模为 n 的输入所耗费的最大时间（今后将定义它为时间复杂度函数）。n 是该问题输入的规模（尺寸）。

表 1.1 若干种多项式和指数时间复杂度解题速率的比较

$T(n)$	n		
	10	30	60
n	0.01 ms	0.03 ms	0.06 ms
n^5	0.1 s	24.3 s	13 min
2^n	1 ms	17.9 min	366 世纪

表 1.2 随着计算机速度提高，同一单位时间内，不同算法的解题规模扩大程度的比较

$T(n)$	输入规模		
	现有计算机	提高 100 倍后	提高 1000 倍后
n	N_1	$100N_1$	$1000N_1$
n^5	N_2	$2.5N_2$	$3.98N_2$
2^n	N_3	$N_3+4.19$	$N_3+6.29$

如果把算法设计比作剧作家创作电视剧，那么算法分析就可以比作欣赏者对电视剧的评论。正如一个优秀电视剧在创作与评论的交替中诞生一样，一个优良的算法是在设计与分析的交替中问世的。

在本书中，我们的注意力将集中于下述六点：

1．对一个给定的问题，如何设计出它的有效算法。

2．给定的一个算法，如何分析它的有效性。

3．在同一问题的若干算法之中，应采用哪些准则来判断它们的优劣？

4．对一个问题还能提出新的改进的算法吗？

5．在什么意义下，算法被证明为最佳。

6．这些理论的实际应用是什么？

我们希望读者在读完本书之后，在面临新的需要解决的实际问题时，比较善于和习惯于从上述六点设计、分析和研究该问题的算法。

本课程中，用 pascal 语言来描述算法，但我们倾向于常常用非形式化的描述。例如，在一些显然的场合，省略语句括号 begin 与 end；省略常量、变量说明；为着方便起见，我们使用 return 语句等等。

§1.2 分析算法的若干准则

评价一个算法的性能质量，包括下述五条准则：

1．正确性。2．工作量。3．空间用量。4．简单性。5．最佳性。

正确性

称一个解算某问题的算法是正确的，如果对任给的一个有效输入（即问题定义域上一个成员），这算法总在一有限时间内给出这问题的正确答案。一个算法的正确性显然包括问题的解法在数学上是正确的，执行算法的指令序列是确定的。我们主要关心前者。

工作量

工作量是指一个程序的运行时间，一个算法的工作量如何度量？由于度量的目的是试图

给出一个算法的有效性信息，因此度量的结果必须要在某种恰当的程度上反映出算法的实际执行时间。

采用下列三种度量标准，被认为是不恰当的：

第一，算法的实际执行时间。它显然随不同的机器(包括运行速度，软件质量等)而异，而我们并不想把工作量的研究限于一特定机器。事实上，算法的分析应独立于机器。

第二，算法执行的指令条数。它依赖于所用的程序设计语言和程序员的风格，为了减少工作量，需要我们花费力气去写出和修改要研究的每一个程序。

第三，算法执行时，通过循环的次数。在一个算法中，一个给定循环的两次通过所做工作量可能相差甚远，通过二个不同的循环则工作量可能相差更大。故单用通过算法的循环的次数不足以反映一个算法的工作量。此外，解算同一问题的两个算法可能有完全不同的控制结构，通过它们各自循环的一圈工作量可能完全不同。

我们所要采用的度量，应该与计算机，程序设计语言，程序员无关，还应该与许多实现细节(或称为'簿记'操作，例如循环下标计数，数组下标计数或设置指针等)无关，但是它又必须反映算法实际执行时间方面的信息。这个度量标准必须既足够一般、又足够精确，以发展一套对各种算法和应用都适用的理论。

为了分析一个算法，我们可以抽出一个特定的操作，称为基本操作，它对被研究的问题(或对被讨论的算法类)是基本的。'基本'的含义包括这一操作是解算这一问题的关键操作，操作次数将随着输入尺寸增大而增加，而簿记操作则不一样，它也许保持常数或以较小的速度增加。因此，弃置簿记工作量不顾，只计算算法所执行的基本操作次数。若这算法所执行的操作总数粗略地与基本操作数成比例(当输入尺寸增大时)，则这一指定是合理的。更精确地说，如果基本操作的指定是合理的，那么若这算法对输入 I 做 $t(I)$ 次基本操作，则操作基本总数至多是 $ct(I)$，而实际执行时间至多是 $c_1t(I)$ 秒。这里 c 和 c_1 是常数。它们仅依赖于这个算法的簿记方法和实现这算法的计算机，而与 I 的大小无关。因此，只要基本操作选取得合理，对算法的工作量就有一个好的度量。对解算同一问题的若干算法，只要选取的基本操作相同，也就有了比较它们优劣的准绳。

表 1.3　几个问题的基本操作

问　　题	基　本　操　作
1. 在一表中搜索 x	比较(x 与表中一元相比较)
2. 两个实矩阵相乘	乘法(两个实数相乘)(或实数的乘、加法)
3. 整序一个表	比较(表中两元比较)
4. 遍历一二叉树(链接结构)	访问一个链接，置一个指针

三点注记

1. 可能要考虑一种以上基本操作　设对一个问题已选择一基本操作，然后又发现解这一问题的若干算法，还做别的操作，这操作的工作量如此之大，以致于没有理由认为它是簿记，那么应指定这操作是另一基本操作。当有一种以上基本操作时，要分别计数，或再(加权)求总数以反映复杂性。
2. 选择什么样基本操作，允许有较大的灵活性。虽然，我们常常试图选择一个，至多两个基本操作，但是我们也可选择更多的操作为基本操作。在极端情形，可把一台特定计算机

的指令系统〔如 Knuth 在他的书"程序设计技巧"中采用的 MIX 语言及指令〕选择为基本操作。因此，借助于改变基本操作的选择，可改变分析的精度和抽象程度，以适应实际需要。

3. 常常用指定基本操作来定义一个算法类。解算同一问题，且选用相同基本操作的算法可归入一类。在同一类里，各个算法可用所执行的基本操作数来标志它们的优劣。在不同类里，两个算法的比较可能不准确。

平均(average)和最坏情况(worst-case)分析

如上所述，我们可计数基本操作个数来度量一个算法的工作量。现在我们要讨论如何精确地表示计数结果的方法。

因为执行的基本操作数总是随着输入的不同而异，算法的工作量不可能为固定常数。首先它依赖于输入的规模。其次，即使考虑一种规模的输入，它也依赖于特定的具体的输入。

输入规模对工作量大小起主要作用，应当给一度量。一个问题的规模可以用一个整数(或若干个整数)来表示，它是对输入数据量的一个测度。例如

表 1.4 问题和输入规模举例

问　　题	输　入　的　规　模
1. 在一个表中搜索 x	表中元素的个数
2. 两个矩阵相乘	矩阵的维数
3. 整序一个表	表中元素的个数
4. 遍历一二叉树	树中顶点个数
5. 解一线性方程组	方程的个数，或未知数个数，或两者兼而有之

相同规模的输入，算法实际执行情况可能会大不相同，这就需要区分平均情况与最坏情况两种分析。

平均性态

要给出一个算法的平均性态，可对每一输入 I，$|I|=n$，计数算法执行的基本操作数，然后取平均。在实用中，由于不同的输入出现的概率未必相同。采用加权平均可给出更有意义的结果。

设 $\mathscr{D}_n$ 是一个问题规模为 n 的输入集，$I\in\mathscr{D}_n$，$p(I)$ 是输入为 I 的概率(事件'输入为 I' 发生的可能性大小叫做输入 I 的概率)。设 $t(I)$ 为算法接收 I 所执行的基本操作个数，则加权平均特性定义为

$$A(n)=\sum_{I\in\mathscr{D}_n}p(I)t(I)$$

其中 $p(I)$ 一般由这算法要用到的应用中的经验和/或外界特别的信息来确定。当然，这时计算出来的 $A(n)$ 仅对这些应用有效。

最坏情况性态

沿用上面记号，最坏情况特性定义为

$$W(n)=\max_{I\in\mathscr{D}_n}t(I)$$

显然有

$$A(n)\leqslant W(n)$$

若一个算法在平均情况(或最坏情况)做 $A(n)$(或 $W(n)$)次基本操作，则它在平均情况

(或最坏情况)下总操作次数 $\leqslant cA(n)$(或 $cW(n)$), 运行时间 $\leqslant c_1A(n)$(或 $c_1W(n)$)。其中 c, c_1 独立于 n, I。上面所定义的算法工作量称为算法的时间复杂性。$A(n)$, $W(n)$分别称为平均复杂性(有时称为平均复杂度)和最坏情况复杂性(或最坏情况复杂度)。它们不是个别的常量, 而强调了算法工作量随同问题规模而变化的一种趋势(或渐近性态)。在不致混淆时, 也采用记号 $T(n)$。

例 1.1 搜索问题

算法 1.1 SEQUENSEARCH 表搜索 {L 是一含 n 元的表, x 是任一给定的元。若 x 在 L 中, 则找出 j 使得 $L(j)=x$。若 x 不在 L 中, 则 $j=0$}

```
j: =1;
while j≤n and L[j]≠x do j: =j+1;
if j>n then j: =0;
```

平均性态:

这问题的输入 L, x 能依 x 在 L 中何处出现而分类, 即有 $(n+1)$ 个输入类要考虑。对 $1\leqslant i\leqslant n$, 令 I_i 表示 x 在 L 中第 i 位置的输入(类), I_{n+1} 表示 x 不在 L 中的输入(类)。则显然, 算法对输入 I_i, I_{n+1} 的比较个数为 $t(I_i)=i$, $t(I_{n+1})=n$。为计算平均比较个数, 必须给出 x 在 L 中的概率和 x 在 L 中任一特定位置的概率。设 x 在 L 中的概率为 q, 且假设 x 在 L 中每个位置出现的概率相同, 则 $p(I_i)=q/n(1\leqslant i\leqslant n)$, $p(I_{n+1})=1-q$。故

$$A(n)=\sum_{i=1}^{n+1}p(I_i)t(I_i)\ =q\cdot\frac{n+1}{2}+(1-q)n$$

譬如说, 已知 x 在 L 中, 则 $q=1$, $A(n)=(n+1)/2$。这表明若预先知道 x 必在 L 中, 在平均情形下, 需要搜索表的一半。若 $q=1/2$, 即 x 在 L 中或 x 不在 L 中机会相等, 则 $A(n)\approx 3/4\,n$。这表明在这种情况下的平均特性: 表的 3/4 部份要搜索到。

最坏情况:

显然, $W(n)=\max\{t(I_i): 1\leqslant i\leqslant n+1\}=n$。当 x 是 L 中最末一元或 x 不在 L 中出现是搜索的最坏情况。在这时, x 和 L 的全部元素相比较。

两点注记:

1 例 1.1 表明我们应当怎样解释 $\mathscr{D}_n$。我们仅考虑输入的那些影响到算法性态的性质, 即 x 在 L 中否? 在哪个位置等等, 而不考虑输入是实数, 字符串或其他数据类型等等。$\mathscr{D}_n$ 中的一个元素 I 可视为一个集合(或等价类)。这一集合由 x 出现在 L 中某一特定位置为表征。$t(I)$ 是 I 中任一输入所做的操作数。

2 一个算法表现出最坏性态的输入依赖于该特定算法, 而不是由问题所决定的。例 1.1 的最坏性态输入即最坏情况是 'x 在 L 中最末一元', 但在另一算法(由右至左顺序搜索)中却为最佳输入。

例 1.2 矩阵乘法问题

算法 1.2 MULMATRIX 矩阵乘法

```
for i, j: =1 to n do begin
    c[i, j]: =0;
    for k: =1 to n do
        c[i, j]: =c[i, j]+a[i, k]*b[k, j]
end;
```

矩阵的元的乘法作为基本操作。对 c 的每一个元 c_{ij} 做 n 次乘法，c 有 n^2 个元，$\therefore A(n)=W(n)=n^3$。算法 MULMATRIX 的时间复杂性是数据无关的。它们的最坏情况复杂性和平均情况复杂性相等。

记号 O, θ, Ω

记号 O, θ, Ω 是用来表示函数渐近性态的三个记号。

称 $T(n)$ 是 $O(f(n))$，记作 $T(n)=O(f(n))$，如果存在常数 c 和 n_0，使得当 $n\geqslant n_0$ 时有 $T(n)\leqslant cf(n)$。当 $T(n)=O(f(n))$ 时，$f(n)$ 是 $T(n)$ 的渐长率（或渐近性质）的一个上界（在相差一个常数因子意义下）。一般地，若有 $g(n)=O(f(n))$，未必有 $f(n)=O(g(n))$，如 $f(n)=n^2$，$g(n)=n$。

称 $T(n)$ 是 $\theta(f(n))$，记作 $T(n)=\theta(f(n))$，如果 $T(n)=O(f(n))$ 和 $f(n)=O(T(n))$ 同时成立。当 $T(n)=\theta(f(n))$ 时，$T(n)$，$f(n)$ 的增长率是同阶的。

如果存在一个常数 $c\neq 0$，使得对无穷多个 n 成立 $T(n)\geqslant cf(n)$，称 $T(n)$ 是 $\Omega(f(n))$，记作 $T(n)=\Omega(f(n))$。当 $T(n)=\Omega(f(n))$ 时，$f(n)$ 是 $T(n)$ 在这无穷多个 n 上的增长率的一个下界。通常，如果我们不特别申明，$T(n)=\Omega(f(n))$ 蕴含 $f(n)$ 在 $n\geqslant n_0$（某个常数 n_0）上给出 $T(n)$ 增长率的下界。此时，$T(n)=O(f(n))$ 和 $T(n)=\Omega(f(n))$ 的同时成立蕴含 $T(n)=\theta(f(n))$。

最后，我们用一个进一步的注记来结束本段有关'时间工作量'的讨论。

当我们分析一个新算法时，首先要知道它的复杂性与已有的算法相比是低阶，同阶或高阶。如果分析结果，两个算法非同阶，那么我们认为计数基本操作个数而忽略簿记和实现细节是合理的，此时，我们可以区别出两个算法的优劣。如果两个算法同阶，那么我们要进而比较它们的基本操作估计项的系数（如 c_1n^α 和 c_2n^α 中的 c_1 和 c_2）。如果两个算法的时间复杂性基本操作估计项不但同阶而且同系数，乃至估计的低阶项也都相等，那么我们要进而比较它们的簿记和实现细节。这一想法可用来开发或修改一个算法：首先寻找减小复杂性阶的修改（粗调），其次研究约简一些细节（细调）。

空间用量

一个算法的空间用量，如同它的实际运行时间一样，依赖于这算法的特定的实现程序。然而，关于空间用量的某些结论则可仅仅考察算法而得出。一个程序要用存贮空间来存贮指令/常数/变数/输入数据，也要用一些工作空间来存贮这程序运行中的数据/信息。输入数据可能以几种结构表示，其中一些结构可能比另一些结构要求更多的存贮空间。若输入数据以一种自然形式而存贮，例如一个数组或矩阵，则我们考虑除去程序和输入数据的存贮空间以外要用的额外空间量。若额外空间量关于输入规模是常数，则算法称为就地工作（work in place）。如果输入能由不同形式表示（例如图在计算机中就有几种表示法，详见有关图的算法之章节），则我们将考虑输入本身所要的空间，同时还考虑所用的任何额外空间。空间的计量单位是'单元'，一般地，我们这里所说的'单元'是笼统的，有时我们设想它可放一个很长的数，有时我们又设想它只放 0，1 这样的一个很小的数。在具体应用中，我们才把这里的'单元'与计算机的'字长'联系起来。空间用量（或空间复杂性）类似于时间复杂性，用一个关于输入规模的函数 $S(n)$ 来刻划它的渐近性质。如果空间复杂性依赖于特定的输入，则可采用最坏情况和平均情况空间复杂性来度量，详细而具体的例子将在讨论整序（Sorting）的章节中给出。

简单性

一个算法的简单性是指它的直观，清晰易读。当然这里没有定量的标准。一个简单的算法易于证明它的正确性，也易于分析它的效率等等。不过，二者相比，效率是决定因素，我们只能在保持尽可能高的效率前提下，力求算法简单些。因为事实上存在着简单但不是效率高的算法。

最佳性

为了分析一个问题的复杂性，我们选择一类算法 $\mathscr{A}$（常常由指定基本操作来确定一算法类）和复杂性的一个度量（例如，$T(n)=W(n)$ 或 $A(n)$），然后试图求出 $\mathscr{A}$ 中任一解算这问题的算法至少应有怎样的复杂性。称一个算法 A 在 $\mathscr{A}$ 中是最佳的，如果 $\mathscr{A}$ 中其他算法的复杂性不低于 A 算法的复杂性。

形式地说，设为解某问题 P，有一特定的（一般由基本操作而定）算法类 $\mathscr{A}$。对任一 $A\in\mathscr{A}$，其时间复杂性为 $T_A(n)$（最坏或平均情况复杂性），定义

$$C_p(n)\triangleq\min_{A\in\mathscr{A}}\{T_A(n)\}$$

它可称为问题 P 关于算法类 $\mathscr{A}$ 的（最坏或平均）计算复杂性。一个问题的计算复杂性是这个问题（在限制算法类下）所固有的，因此常称为问题 P 的固有计算复杂度。

要估计量 $C_p(n)$ 是很困难的，是算法研究的中心课题。算法研究者总是对给定的问题求出它在特定算法类下的固有计复难度（即复杂度）下限，然后努力设法设计出达到这一计算复杂度的算法。前者简称为'下界问题'，后者，即努力设计出好算法，并估计该算法的计算复杂性问题简称为'上界问题'。具体地说是：

1　求 $C_p(n)$ 的下界

要求构造一个函数 $g(n)$（例如 cn，$cn\log n$，cn^2 等等），使得可以证明，对任何 $A\in\mathscr{A}$，均有 $T_A(n)\geqslant g(n)$ 成立（对适当大 $n\geqslant n_0$）。则 $g(n)$ 是 $C_p(n)$ 的一个下界。

在最坏情形时，注意到 $W_A(n)=\max\limits_{I\in\mathscr{D}_n}t(I)$，所以要证明 $g(n)$ 是 $C_p(n)$ 的下界，等价于证明对任何 $A\in\mathscr{A}$，总存在某个规模为 n 的输入 $I\in\mathscr{D}_n$，使得 $t(I)\geqslant g(n)$（对 $n\geqslant n_0$）。

2　求 $C_p(n)$ 的上界

设计出一个解算 P 的算法 $A\in\mathscr{A}$，分析出 A 的算法复杂性 $T_A(n)$，于是有

$$C_p(n)\leqslant T_A(n)$$

$T_A(n)$ 是 $C_p(n)$ 的一个上界。

设 $g(n)$ 是 $C_p(n)$ 的一个下界，若存在某个算法 $A^*\in\mathscr{A}$，使得 $T_{A^*}(n)\leqslant g(n)$，$n\geqslant n_0$，则 A^* 是最佳算法。事实上，有 $T_{A^*}(n)=g(n)$，$(n\geqslant n_0)$。一般地，设 $f(n)$，$g(n)$ 分别为 $C_p(n)$ 的上界和下界，如果有 $f(n)=\theta(g(n))$ $(n\geqslant n_0)$，则可以认为 $T_A(n)=f(n)$ 的算法 A 是一个最佳算法（在最佳阶的意义下）。这时 $T_A(n)$ 仅与 $g(n)$ 相差一个常数因子，且没有实质上比 A 更小复杂性的算法解算这一问题，但其常数因子也许还有减少的余地。这可以从不同整序方法的讨论中得到验证（见第二章）。

例 1.3　在一个 n 个元的表中找最大元

算法类：限于在表的元上仅做比较和复制操作的算法类 $\mathscr{A}$（比较算法类）

基本操作：两个元的比较