普通高等教育“十一五”国家级规划教材

算法设计与分析

Design and Analysis of Algorithms

朱大铭　马绍汉　编著

高等教育出版社
HIGHER EDUCATION PRESS

普通高等教育“十一五”国家级规划教材

算法设计与分析

朱大铭　马绍汉　编著

高等教育出版社

内容提要

本书是普通高等教育“十一五”国家级规划教材。本书以算法设计策略为知识单元，系统地介绍计算机算法的设计方法与分析技巧，以期为计算机科学与技术专业的学生提供广泛而坚实的计算机基础知识。主要内容包括算法分析技术，算法设计技术，P 类、NP 类及 NPC 类，证明问题属于 NPC 类的技术，NPC 问题子问题的复杂性，拟多项式变换和图灵归约，NP-难解问题近似算法，近似算法设计技术，等等。

本书包括了算法与复杂性领域的主要内容，可以作为高等学校计算机专业高年级本科生和研究生学习计算机算法设计的教材，也可供广大工程技术人员和自学者学习参考。

图书在版编目（CIP）数据

算法设计与分析/朱大铭，马绍汉编著．—北京：高等教育出版社，2009.1
ISBN 978-7-04-025871-4

Ⅰ．算… Ⅱ．①朱…②马… Ⅲ．①电子计算机-算法设计-高等学校-教材②电子计算机-算法分析-高等学校-教材 Ⅳ．TP301.6

中国版本图书馆 CIP 数据核字（2008）第 195915 号

策划编辑 刘 艳 **责任编辑** 萧 潇 **封面设计** 王凌波 **责任绘图** 吴文信
版式设计 陆瑞红 **责任校对** 刘 莉 **责任印制** 陈伟光

出版发行	高等教育出版社	**购书热线**	010-58581118
社 址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100120	**网 址**	http:// www.hep.edu.cn
总 机	010-58581000		http:// www.hep.com.cn
		网上订购	http:// www.landraco.com
经 销	蓝色畅想图书发行有限公司		http:// www.landraco.com.cn
印 刷	涿州市京南印刷厂	**畅想教育**	http:// www.widedu.com
开 本	787×1092 1/16	**版 次**	2009 年 1 月第 1 版
印 张	16	**印 次**	2009 年 1 月第 1 次印刷
字 数	320 000	**定 价**	20.30 元

物料号 25871-00

前　言

计算机是一种现代化的信息处理工具，它对信息进行处理并提供所需结果，其结果取决于所接受的信息及相应的处理算法。算法是以计算机能够理解的语言描述的解题过程。当算法作用于所求解问题的给定输入集或作用于问题自身的描述上，将产生唯一确定的有限动作序列。此序列或终止于给定问题的解，或终止于对此输入信息无解。对于给定的问题，基于对其的深刻理解，可设计出解答该问题的一个算法。在这个意义上，设计算法是将有关问题的知识，转化为解决问题的智慧。知识诚可贵，智慧价更高。设计算法就是揭示所研究问题的基本特征，以寻求其解答策略的过程，这是一项艰苦的创造性工作。

对于给定的问题，有可能设计出多个算法，但不同的算法质量会不一样。衡量算法质量的主要因素是算法执行所耗费的时间和所需存储空间，以及算法适用范围等。对算法质量的分析研究称为算法分析。算法设计和算法分析是计算机科学的核心基础。国内外大学的计算机专业中，都将“算法设计与分析”作为专业基础课。

近半个世纪以来，算法研究始终是计算机科学的一个研究热点。以 E. W. Dijkstra、S. A. Cook、R. M. Karp、J. H. Hopcroft 及 R. E. Tarjan 等为代表的一批计算机科学家，以创造性工作推动着算法研究不断深入发展。在研究过程中，算法理论研究与软件技术研究之间产生了鸿沟，使得算法研究缺乏足够的实验支持，而实验工作又没有充分的理论分析。这种现状引起了人们的忧虑。在 1996 年的 ACM 计算理论学术年会(STOC'96)上，一些计算机科学家提出“算法工程”的概念，强调研究算法实现技术的重要性，呼吁算法研究者应重视理论与实践的结合，将算法设计、分析、实现、测试及改进等过程一体化、工程化。这种研究思路越来越受到人们的关注，促使算法研究健康发展。

本书原稿写于 20 世纪 80 年代中期、笔者在山东大学为计算机专业研究生讲授“算法设计与分析”课程期间，先后几经修改，于 1992 年定稿并由山东大学出版社正式出版。此后一直使用本书作为计算机科学与技术专业的学位课教材，由笔者和朱大铭教授共同为研究生讲授，效果尚好。经过 20 多年的研究沉淀，算法设计与分析虽然没有太多变迁，但其研究内容更加丰富和成熟。这期间我们始终坚持在这个研究方向上从

事教学和科研工作，连续主持了10多项国家自然科学基金课题，培养了一批博士、硕士研究生。为适应培养创新型人才的需要，经过反复酝酿，决定由朱大铭教授执笔，重新修改编著这本书，增加近10年来的研究成果，使之能反映21世纪以来算法设计与分析的研究现状。可以说，本书是山东大学计算机科学与技术学院两代人坚持不懈、刻苦努力完成的。在本书写作过程中，得到朱洪教授的指导和帮助，在此表示感谢。

本书主要面向计算机相关专业的高年级本科生和研究生。全书共分8章，其中：第1、2章讨论算法分析技术和算法设计技术；第3、4、5、6章论述NP-完全性理论，特别强调多项式变换和多项式图灵归约的方法及应用；第7章论述NP-难解问题近似算法的基本概念和设计NP-难解问题近似算法的基本方法；第8章讨论近似算法设计的基本理论与技术。

算法研究是一项富有挑战性的工作，对提高计算机学科的研究水平极为重要，希望能有更多的有识之士投身这项工作。由于我们学术水平有限，本书内容必有疏漏、欠妥、谬误之处，敬请读者指正。作者的联系方式为dmzhu@sdu.edu.cn。

马绍汉

二〇〇八年仲夏于山东大学

目　录

第1章 算法分析技术

计算机的计算是在程序控制下进行的。程序和数据存放在存储器中，中央处理器执行程序规定的操作步骤，计算出所要解答问题的结果。设计程序首先需要明确程序要解答什么样的问题，这就要形式化地描述问题的输入数据、输出数据和输出数据应满足的条件。好的程序不仅是正确的，还应该是高效的。计算机运行一个程序，输出正确结果或有效结果需要多少时间和多少存储空间，是标志这个程序解答问题优劣的主要量化指标。影响程序运行的计算时间和存储空间的因素很多，仅从程序在具体计算机上运行一次或几次所花费的时间和占用的存储空间难以断定程序的优劣。要客观地评价程序，就要脱离具体的计算机，分析程序解答一定规模的问题实例所使用的基本操作次数和占用的存储单元个数。脱离了具体的计算机、只描述解答问题的基本操作和操作顺序的计算过程就叫做解答问题的算法。实际上算法并不能真正脱离计算机，设想所有算法都在一个通用的计算模型上运行，这个计算模型就是第3章将会讲述到的图灵机模型。分析算法在计算模型上解答问题需要多少基本操作、多少存储单元，有助于客观、精确地评价算法，还有助于我们设计更好的算法。但在实际的算法分析中，通常不会去分析算法在计算模型上运行所需要的操作次数和存储单元个数，只需要脱离具体计算机，分析算法解答问题所使用的基本操作次数和占用的存储单元数，就足够了。

§1.1 算法及其复杂性

所谓问题(problem)，就是一个要求给出解答的一般性提问。它由两个要素组成：第一个要素描述问题的所有参量和参量格式，称为实例；第二个要素陈述问题解的格式和应当满足的条件，称为询问。所谓算法(algorithm)是一个过程，这个过程是若干语句的集合，每条语句都由明确指定计算机操作和操作顺序的规则构成。只要按照语句一步步地执行，便可得到问题的解。通常可将程序(program)看成是算法在具体计算机上运行的描述形式。在本书中，常常将程序和算法两个词混用。如果一个算法能应用于问题的任何实例，并保证得到正确的解，那么称这个算法解答了该问题。问题的实例又称为算法的输入，而问题的询问又称为算法的输出。评价算法的好坏，是算法分析(algorithm analysis)的中心内容。下面先从一个简单例子谈起。

例1.1：求幂问题。

实例：实数 x 和正整数 n。

询问：$x^n=?$

关于求幂问题应有很多算法。例如，当限制 n=10 时，求 x^{10} 的值至少有下述三种计算方法。

(1) 计算 $x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x\cdot x$。

(2) 先求出 $y=x\cdot x$，$z=y\cdot y$，$w=z\cdot z$，再计算 $y\cdot w$。

(3) 计算$((x\cdot x)^2\cdot x)^2$。

若想比较这三个算法的好坏，可以对特定实例将计算机的运算时间实际测出来，但是实际测出的时间不仅与算法的好坏有关，还与计算者使用计算机的熟练程度及所用计算机的性能都有关系。为了排除使用者熟练程度和计算机性能等因素的影响，在考虑算法计算 x^n 的时间时，只计入算法使用乘法操作的次数。

只用乘法操作次数表示算法的运行时间，使我们能够客观地比较两个算法的好坏。在算法分析中总是只考虑算法的主要操作步骤，并估计算法执行过程中所需的总操作次数，将这个总操作次数称为算法的时间复杂性(time complexity)。若以乘法为其主要操作，则上述算法(1)、(2)、(3)的时间复杂性分别为 9、4、4。同样地，算法在执行过程中所使用的存储器单元数目称为算法的空间复杂性(space complexity)。若以寄存一个数的存储空间作为一个单元，则算法(1)、(2)、(3)的空间复杂性分别为 2、4、2。

由这些数据可以看出，在这三个求 x^{10} 的算法中，以算法(3)为最好。我们先来导出算法(3)计算 x^n 的一般性规律，然后设计对任意正整数 n>0，计算 x^n 的算法。考虑 n 的二进制数表示，设 $n=(b_{m-1}b_{m-2}\cdots b_0)_2$，$b_{m-1}$=1，则

$$\begin{aligned} x^n &= x^{2^{m-1}+2^{m-2}b_{m-2}+2^{m-3}b_{m-3}+\cdots+2b_1+b_0} \\ &=((((x^2\times x^{b_{m-2}})^2\times x^{b_{m-3}})^2\times\cdots)^2\times x^{b_1})^2\times x^{b_0} \end{aligned} \tag{1.1}$$

根据(1.1)式，给出计算 x^n 的算法如下。

```
算法：Power(x,n)
//输入为 x 和 n，n=(b_{m-1}, b_{m-2},…, b_0)_2，m>1
1   y←x;
2   For i←m-2 down to 0 do
3       y←y*y;
4       if  b_i=1 then y←y*x;
5   End For
6   输出 y
```

将算法 Power(x,n)计算 x^n 的时间复杂性函数记为 $T(n)$，仍然只考虑计算中的乘法次数，作为算法的时间复杂性。假设 n 为一个 m 位二进制数，最高位为 1，显然除其最高位不计乘法次数外，从 n 的二进制数高位到低位，计算乘法次数的规律是逢 0 加

1，逢 1 加 2。若 $n=(10\cdots0)_2$，则乘法次数为 $m-1$；若 $n=(11\cdots1)_2$，则乘法次数为 $2(m-1)$。对于 n 的其他取值，算法所需乘法次数在 $m-1$ 与 $2(m-1)$之间。从 n 的二进制数表示可知，$\lfloor\log_2 n\rfloor=m-1$。因此算法的时间复杂性函数 $T(n)$可估计为

$$\lfloor\log_2 n\rfloor\leqslant T(n)\leqslant 2\lfloor\log_2 n\rfloor \tag{1.2}$$

容易看出，算法的空间复杂性函数 $S(n)=2$。在上述讨论中，算法的时间与空间复杂性总与参数 n 有关，n 代表了问题实例的规模。在算法分析中，通常将表达问题实例规模的参量 n 称为问题实例的输入长度(size)。

例如，在矩阵求积问题中矩阵的行数、多项式求值问题中多项式的阶数、排序问题中数值的个数，都可被用来表示问题实例的输入长度。在图上有关问题中，一般将图的顶点个数看做输入长度。如图上最短路径问题、最小生成树问题、着色问题，均可将图的顶点个数看做问题实例的输入长度。利用第 3 章的图灵机模型，将能够更加明确地解释什么是输入长度。一般地讲，一个算法解答输入长度为 n 的问题实例，其时间复杂性函数 $T(n)$和空间复杂性函数 $S(n)$会随 n 的增加而增加，但往往又不是 n 的严格递增函数。

一个算法解答输入长度为 n 的不同实例，其时间复杂性或空间复杂性一般是不同的。精确地给出算法解答一个具体实例的时间和空间复杂性通常难以做到；也没有必要专门去分析算法解答一个问题实例的时间或空间复杂性函数的精确数值。这也是前面定义的时间和空间复杂性函数只与实例输入长度有关，而与实例无关的原因。

仅凭算法解答问题一个或几个实例的时间和空间复杂性，也不可能客观地评价一个算法的好和坏。因此在实际的算法复杂性分析中，通常估计算法时间和空间复杂性函数 $T(n)$和 $S(n)$的界，以此来表示算法的好与坏。如例 1.1 中，给出了求幂算法计算 x^n 的时间复杂性函数上界为 $2\lfloor\log_2 n\rfloor$，下界为$\lfloor\log_2 n\rfloor$。

在实际应用中，人们更多地使用算法的时间复杂性来表示算法的好和坏。多数情况下，算法时间复杂性比空间复杂性更适合用来比较算法好坏。在分析算法时间复杂性时，通常也不需要同时估计算法时间复杂性的上界和下界。

设算法 A 解答输入长度为 n 的问题实例的时间复杂性函数为 $T_A(n)$。估计 $T_A(n)$需要分析算法最坏情况下所需要的基本操作次数，从而给出算法的时间复杂性上界。如求幂算法计算 x^n 的时间复杂性不超过 $2\lfloor\log_2 n\rfloor$，以后如不特别强调，都直接将算法的时间复杂性上界称为算法的时间复杂性。算法空间复杂性函数 $S(n)$也可类似地定义。

若解答一个问题有多种算法，则这些算法的时、空复杂性函数不一定相同。假设解答某问题的所有算法的集合为 $\mathcal{F}$，其中算法 A 的时间复杂性函数为 $T_A(n)$，$T(n)=\min\limits_{A\in\mathcal{F}} T_A(n)$。$T(n)$为解答这个问题最优算法的时间复杂性函数，亦表示解答这个问题长度为 n 的实例所需时间的下界，称 $T(n)$为该问题的时间复杂性函数。类似地，可定义问题的空间复杂性函数 $S(n)$。问题的时、空复杂性刻画了该问题的计算复杂性。

有时还考虑算法解答问题实例所需基本操作数目的平均值，以此来评价算法好坏。这种平均值估计只有在知道问题实例的概率分布时才行得通，而知道问题实例的概率分布一般很难做到。在讨论算法时间复杂性时，更多的是去分析算法解答各种实例时间复杂性的最坏值，给出算法的时间复杂性上界。本书不涉及算法平均时间复杂性的分析的内容，以下提及的复杂性函数，如不特别指出，通常指时间复杂性函数。

§1.2 渐近估计技术及基本规则

算法复杂性函数 $T(n)$的渐近估计即讨论随着实例输入长度 n 的增长，$T(n)$的增长趋势。在估计算法的时间复杂性时，经常用数学函数 $O(\bullet)$来表示算法时间复杂性的上界，用$\Omega(\bullet)$来表示算法时间复杂性的下界，另外，人们也经常使用$\Theta(\bullet)$表示算法时间复杂性的确切界。

复杂性函数 $T(n)$是定义在非负整数域上的函数，但其函数值 $T(n)\geqslant 0$ 不一定也为整数。当我们说某算法的复杂性函数 $T(n)=O(n^2)$时，其意思为：存在一个确定常数 C 和 n_0，使得当 $n\geqslant n_0$ 时，有 $T(n)\leqslant Cn^2$。一般地，若存在常数 C、n_0 和函数 $f(n)$，使得当 $n\geqslant n_0$ 时有 $T(n)\leqslant Cf(n)$，则记为 $T(n)=O(f(n))$。函数 $O(f(n))$提供了时间复杂性函数 $T(n)$的一个上界，它并不一定是算法的实际运行时间。用另外的说法，$T(n)=O(f(n))$表示当 n 逐渐增大时，$Cf(n)$是 $T(n)$的上界，其中 C 一定存在，但无须关心 C 的取值。

例如，函数 $T(n)=3n^3+2n^2=O(n^3)$，因为对于所有 $n\geqslant 0$ 有 $3n^3+2n^2\leqslant 5n^3$。但函数 $T(n)=3^n$ 不能记为 $O(2^n)$。这是因为，找不到 n_0 和 C，使得对于所有 $n\geqslant n_0$ 有 $C2^n\geqslant 3^n$。有时 $O(\bullet)$也作为简化形式用于一个表达式中，例如 $T(n)=3n^3+2n^2+5n-10=3n^3+O(n^2)$。

在说明一个算法较好时，就需要用到函数 $O(\bullet)$。例如，假设算法 A 与算法 B 是解决同一个问题的算法。算法 A 的时间复杂性函数为 $T_A(n)=O(n^2)$，算法 B 的时间复杂性函数为 $T_B(n)=O(n^3)$。我们就认为算法 A 好于算法 B。这里算法 A 好于算法 B 的说法，仅说明算法 A 和 B 解答同样输入长度的问题实例，A 的时间复杂性上界不超过 B 的时间复杂性上界，并非算法 A 解答每个实例的时间复杂性都小于算法 B 解答同样实例的时间复杂性。

当我们说某算法的复杂性函数 $T(n)=\Omega(n^2)$时，其意思为，存在一个确定常数 C 和无穷多个 n 值，使得 $T(n)\geqslant Cn^2$。一般地，$T(n)$增长率的下界记为$\Omega(g(n))$，其意指，存在一个常数 C 和无穷多个 n 值，使 $T(n)\geqslant Cg(n)$。注意，这里不要求对所有足够大的 n 都有不等式 $T(n)\geqslant Cg(n)$成立。记号 $O(\bullet)$和$\Omega(\bullet)$之间并不对称，这种不对称性对算法分析是有益的。例如，测试 n 是否为素数的算法，当 n 为偶数时，判定极其简单；而当 n 为奇数时，则需进一步判定。在这种情况下，我们只需估计当 n 为奇数时，算法复杂性函数 $T(n)$的下界就可以了。再如，假定

$$T(n)=\begin{cases} n, & n \text{ 为奇数} \\ \dfrac{n^2}{100}, & n \text{ 为偶数} \end{cases} \tag{1.3}$$

则可记为 $T(n)=\Omega(n^2)$，即当 $n=2,4,\cdots$ 时，有 $T(n)\geqslant\dfrac{n^2}{100}$。另外(1.3)的时间复杂性函数也可以表示为 $T(n)=\Omega(n)$和 $T(n)=O(n^2)$，但不能记为 $T(n)=O(n)$。一般地，说明一个算法好，需要估计算法时间复杂性的上界；而说明一个算法不好，则通常需要估计算法时间复杂性的下界。如果已有一个算法 A，其时间复杂性函数为 $T_A(n)=O(f(n))$，人们通常会设计解答同样问题的新算法 B，时间复杂性函数 $T_B(n)=O(g(n))$，满足 $g(n)<f(n)$，从而说明算法 B 好于算法 A。有时为了说明算法 A 不好，人们会分析算法 A 的时间复杂性，给出一个下界 $T_A(n)=\Omega(f(n))$，若 $f(n)$数量级较高，如 $f(n)=2^n$，即可说明算法 A 不好。

例如，算法 A、B 的时间复杂性函数分别为 $T_A(n)=\Omega(2^n)$，$T_B(n)=O\left(\left(\dfrac{3}{2}\right)^n\right)$，则算法 A 的时间复杂性下界不如算法 B 的时间复杂性上界好，因此可以说算法 A 不如算法 B 好，但这并不代表算法 A 解答每一个实例的时间复杂性都比算法 B 解答同样实例的时间复杂性大。

设 P_1、P_2 是一个算法的两个过程，两个过程互不调用。若 $T_1(n)$和 $T_2(n)$分别是 P_1 和 P_2 的时间复杂性函数，则算法从 P_1 到 P_2 的时间复杂性函数为 $T_1(n)+T_2(n)$。若 $T_1(n)=O(f(n))$，$T_2(n)=O(g(n))$，则存在常数 C_1，C_2，n_1，n_2，当 $n\geqslant n_1$ 时有 $T_1(n)\leqslant C_1f(n)$；当 $n\geqslant n_2$ 时有 $T_2(n)\leqslant C_2f(n)$。令 $n_0=\max\{n_1, n_2\}$，当 $n\geqslant n_0$ 时有 $T_1(n)+T_2(n)\leqslant C_1f(n)+C_2g(n)\leqslant(C_1+C_2)\max\{f(n),g(n)\}$。所以有 $T_1(n)+T_2(n)=O(\max\{f(n),g(n)\})$。

另外，设 $f(n)$和 $g(n)$是从自然数集到非负实数集的两个函数，如果存在一个自然数 n_0 和两个正常数 c_1,c_2，使得：$\forall n\geqslant n_0$，$c_1g(n)\leqslant f(n)\leqslant c_2g(n)$，则记 $f(n)=\Theta(g(n))$。$f(n)=\Theta(g(n))$意味着 $f(n)=O(g(n))$且 $f(n)=\Omega(g(n))$。

若在过程 P_1 中调用过程 P_2，调用次数也估计为 $T_1(n)$，则在估计算法时间复杂性时需要估计 $T_1(n)\cdot T_2(n)$，类似地讨论可得到，$T_1(n)\cdot T_2(n)=O(f(n)\cdot g(n))$。

现在讨论估计算法时间复杂性函数的基本方法。下面先来分析一个例子。

例 1.2：冒泡排序算法。冒泡排序算法是计算机工作者所熟知的排序算法，下面的冒泡排序算法将整数数组中的元素排列成递增次序。

```
算法：BubbleSort(A[1,n])
//A[1,n]为 n 维整型数组，i,j,temp 为整数
1    For i=1 to n–1 do
2        For j=n down to i+1 do
3            If A[j–1]>A[j] then
```

```
4              temp←A[j−1];
5              A[j−1]← A[j];
6              A[j]←temp;
7           End If
8        End For
9     End For
```

这个程序的输入长度是数组的元素数目 n。每个赋值语句所耗费的时间为与 n 无关的常数，记为 $O(1)$。在程序中，语句 4、5、6 三个赋值操作耗费时间总和为 $O(1)$。现在，考虑执行条件语句和循环语句所耗费的时间。对于条件语句，判定条件需 $O(1)$ 时间，但条件语句的主体(语句 4、5、6)并不一定要执行。由于我们考虑最坏情况下的运行时间，因此可设想每次循环条件语句的主体都执行。这样在程序中语句 3—7 组成的条件语句运行所需时间为 $O(1)$。注意到这个排序算法的程序是一种嵌套结构，我们从里层向外层来讨论条件语句和循环语句的运行时间。考虑由语句 2—8 组成的循环语句，每执行一次循环，循环体耗费时间为 $O(1)$，变量 j 有 $n-i$ 个选择，即循环次数为 $n-i$，因此执行循环语句 2—8 所耗费的时间为 $O(n-i)$。进一步考虑最外层的循环，它包含程序中所有可执行的语句，循环次数为 $n-1$，因此执行整个程序耗费时间为

$$T(n)=O\left(\sum_{i=1}^{n-1}(n-i)\right)=O\left(\frac{n(n-1)}{2}\right)=O(n^2) \tag{1.4}$$

当然，这个排序算法并不是最快的排序算法。我们知道最快的排序算法的时间复杂性函数为 $O(n\log n)$。一般地讲，算法运行时间估计是相当复杂的，没有一套固定的方法可供使用。下面介绍分析算法时间复杂性应遵循的一般规则。

(1) 各赋值、读、写语句的运行时间为 $O(1)$，但也有例外情况，例如，若在赋值语句中允许函数调用，则需考虑函数调用所耗费的时间。

(2) 在忽略常数因素的情况下，语句序列的运行时间为其中需时间最长的语句所耗费的时间。即，若语句序列为 $P_1, P_2, \cdots, P_k$，它们相应的复杂性函数分别为 $T_1(n), T_2(n), \cdots, T_k(n)$，则执行这个语句序列所需时间为 $O(\max\limits_{1\leqslant i\leqslant k}\{T_i(n)\})$。

(3) 条件(IF)语句的运行时间是执行条件主体语句所需时间加上判定条件的时间，通常判定条件的时间为 $O(1)$。If-Then-Else 语句的运行时间为：判定条件所需时间，加上条件为真时执行语句时间和条件为假时执行语句时间之总和。

(4) 执行循环语句的时间为：执行循环体的时间与循环次数之积，加上判定循环结束所需时间。一般的循环次数是清楚的，但有时也可能不清楚或不确定。判定循环结束时间通常为 $O(1)$，可以忽略。

§1.3 递归算法分析

递归算法分析与上节讨论的程序分析不同，这种分析过程可导出递推方程，可通过解递推方程得到其复杂性函数的渐近表示。而解递推方程与解常微分方程有很多相似之处，常借用解常微分方程的模式和术语来解递推方程。本节以合并排序算法为例讨论递归算法的时间复杂性分析法。

1.3.1 合并排序算法分析

将 n 个整数按照递增次序排列，合并排序算法如下。

```
算法：MergeSort(L: List of Integer; n: Integer)
//L 是长度为 n 的表，输出为 L 的排序形式，假定⌊log2n⌋=log2n
1       If n=1 then return (L);
2       Else
3           Break L into halves L1 and L2, each of length n/2;
4           Return(merge(MergeSort(L1, n/2), MergeSort(L2, n/2)))
5       End Else
```

该算法中，merge(L_1,L_2)以两个排序表 L_1 和 L_2 作为输入，将 L_1 和 L_2 合并为一个排序表。只需用两个指针分别指向 L_1 和 L_2 的头，比较两个指针指向数值的大小，将较小的放入另一个临时表，并相应移动指向较小数的指针。直到两个指针分别移动到 L_1 和 L_2 的尾时，合并完毕。假定输入表 L_1 和 L_2 长度均为 $\frac{n}{2}$，执行过程 merge(L_1, L_2)所需比较次数为 $O(n)$。

设 $T(n)$是合并排序算法中递归过程 MergeSort(•)的时间复杂性函数。我们用算法所用比较次数来代表算法的时间复杂性，$T(n)$的递推方程形式为

$$T(n) \leqslant \begin{cases} C_1, & n=1 \\ 2T\left(\frac{n}{2}\right)+C_2 n, & n>1 \end{cases} \tag{1.5}$$

当输入表长度 n=1 时，$T(n)=C_1$，算法计算时间为常数。在输入表 L 长度 n>1 的情况下，应对两个长度为 $\frac{n}{2}$ 的表递归调用 MergeSort(•)过程，每次调用这个过程耗费时

间为 $T\left(\frac{n}{2}\right)$，两次调用共需时间为 $2T\left(\frac{n}{2}\right)$，然后调用过程 merge(•)合并 L_1 和 L_2，需耗费时间为 C_2n，其中 C_2 为常数。注意到，式(1.5)仅当 n 为偶数时才能应用，因此当 n 为 2 的幂次时，式(1.5)提供了一种渐近形式的上界。对于任意的 n，假定 n 介于 2^i 和 2^{i+1} 之间，可以用 $T(2^i)$ 和 $T(2^{i+1})$ 估计出 $T(n)$的一种渐进表达的界。

因此分析递归程序运行时间的上界，可转化成求解关于时间复杂性函数的递推方程。解递推方程一般采用 3 种方法：

(1) 猜测一个解 $f(n)$，并归纳证明 $T(n)\leqslant f(n)$。多数情况下，仅能猜测出 $f(n)$的结构形式，其中某些参量尚需进一步确定。例如，在 $f(n)=an^2$ 中，a 为待定参量。在证明 $T(n)\leqslant f(n)$的过程中，可推演出待定参量的取值范围。

(2) 在递推方程右端，利用递推关系将 $m<n$ 的所有项 $T(m)$都用仅含有 $T(1)$的项代替。由于 $T(1)$是常数，因此得到的 $T(n)$表达式仅含有 n 和常数，以此整理后作为 $T(n)$的渐近表达式。这种方法实际上是展开递推关系。

(3) 利用组合数学中解递推方程的一般方法，可给出某些特殊类型递推方程的解。

本节讨论前两种方法，方法(3)可以参考组合数学的有关文献。首先讨论利用方法(1)解递推方程(1.5)。可以猜测 $f(n)=an\log n$，因为 $n=1$ 时，$an\log n=0$，所以这个猜测不对。进一步猜测 $f(n)=an\log n+b$，当 $n=1$ 时有 $f(n)=b$，只需令 $b\geqslant C_1$，即是合理的。为进行归纳证明，假定对所有 $n\leqslant k$ 有

$$T(n)\leqslant an\log n+b \tag{1.6}$$

我们证明当 $n=k+1$ 时结论成立。当 $n\geqslant 2$ 时，从递推方程(1.5)知 $T(n)\leqslant 2T\left(\frac{n}{2}\right)+C_2n$。若令 $\frac{n}{2}<n=k+1$，则由(1.5)和(1.6)可得到

$$T(n)\leqslant 2\left[a\frac{n}{2}\log\frac{n}{2}+b\right]+C_2n$$

$$\leqslant an\log n-an+2b+C_2n \tag{1.7}$$

由 $n\geqslant 1$ 可知，当 $a\geqslant C_2+b$ 时，不等式 $an\log n-an+2b+C_2n\leqslant an\log n+b$ 成立。于是在条件 $b\geqslant C_1$，$a\geqslant C_2+b$ 的约束下，不等式 $T(n)\leqslant an\log n+b$ 总成立。因此取 $b=C_1$，$a=C_1+C_2$，便有 $T(n)\leqslant an\log n+b$。根据 $T(n)$关于 n 的归纳法，可以推出对于所有 $n\geqslant 1$，有

$$T(n)\leqslant (C_1+C_2)n\log n+C_1 \tag{1.8}$$

即 $T(n)=O(n\log n)$。

在上述讨论中，应注意到下述事实：(1)如果猜测 $T(n)=O(f(n))$，但在证明过程中不能断定 $T(n)\leqslant Cf(n)$成立，此时也不能否定 $T(n)=O(f(n))$的猜测，因为 $T(n)\leqslant Cf(n)+C_1$ 还可能成立。(2)虽然已经证明 $T(n)$不比 $O(n\log n)$增长速度快，但一般还不能确定，关

于 $T(n)$，是否还有比 $O(n\log n)$数量级更小的上界。如果考虑到递推关系式(1.5)是对合并排序算法建立的，并且可以证明对合并排序算法的复杂性函数 $T(n)$有 $T(n)=\Omega(n\log n)$，就可以断定对排序算法，$O(n\log n)$应是复杂性函数 $T(n)$的最好的上界。即 $T(n)=\Theta(n\log n)$。

将递推方程(1.5)更一般地表述为

$$\begin{cases} T(1)=C \\ T(n)\leqslant g\left(T\left(\dfrac{n}{2}\right),n\right), \quad n>1 \end{cases} \tag{1.9}$$

在式(1.5)中 $g(x,y)$的形式为 $2x+C_2y$。当然还可以建立较递推方程(1.9)更一般的形式。例如，函数 $g(\bullet)$不仅包含 $T\left(\dfrac{n}{2}\right)$，还可包含有 $T(n-1), T(n-2),\cdots, T(1)$。现在仅就(1.9)形式的递推方程，讨论如何解递推方程，得到合并排序算法的时间复杂性函数。

假设猜测一个函数 $f(a_1, a_2,\cdots, a_j, n)$，其中，$a_1, a_2,\cdots, a_j$ 是待定参数，现在试着用关于 n 的归纳法证明 $T(n)\leqslant f(a_1, a_2,\cdots, a_j, n)$。为检查对于 $a_1, a_2,\cdots, a_j$ 的某些值及所有 $n\geqslant 1$，都有 $T(n)\leqslant f(a_1, a_2,\cdots, a_j, n)$成立，$f(a_1, a_2,\cdots, a_j, n)$必须满足

$$\begin{cases} f(a_1,\cdots,a_j,1)\geqslant C \\ f(a_1,\cdots,a_j,n)\geqslant g\left(f\left(a_1,\cdots,a_j,\dfrac{n}{2}\right),n\right) \end{cases} \tag{1.10}$$

根据归纳假定，在递推方程(1.9)第 2 式的右端可用 f 代替 T，得到

$$T(n)\leqslant g\left(T\left(\frac{n}{2}\right),n\right)\leqslant g\left(f\left(a_1,a_2,\cdots,a_j,\frac{n}{2}\right),n\right) \tag{1.11}$$

当(1.10)式成立，可将式(1.10)与(1.11)结合起来，证明 $T(n)\leqslant g\left(f\left(a_1,a_2,\cdots,a_j,\dfrac{n}{2}\right),n\right)$成立。这就是关于 n 的归纳法所要进行的证明。例如，对递推方程(1.5)有 $g(x,y)=2x+C_2y$，并且 $f(a_1,a_2,n)=a_1n\log n+a_2$，这里函数 $f(\bullet)$应当满足

$$f(a_1, a_2, 1)=a_2\geqslant C_1 \tag{1.12}$$

$$\begin{aligned} f(a_1, a_2, n)&=a_1n\log n+a_2 \\ &\geqslant 2\left(a_1\cdot\frac{n}{2}\cdot\log\left(\frac{n}{2}\right)+a_2\right)+C_2\cdot n \end{aligned} \tag{1.13}$$

如同前面讨论，取 $a_2=C_1, a_1=C_1+C_2$，则不等式(1.13)成立。

但有时并不能根据递推关系猜测出一个解，或者不能确定解的一个合适的上界，此时可应用方法(2)展开递推关系，直接求出 $T(n)$的表达式。原则上讲，这种方法总是可行的，但实际上常常遇到连续地求和及估计这个和的上界等问题。这些问题通常很