广西重点教材建设基金资助

高等学校教材

计算机算法设计与分析

苏德富　钟　诚　编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL: http://www.phei.com.cn

高等学校教材

计算机算法设计与分析

苏德富　钟　诚　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

算法设计与分析是计算机科学的主要研究领域之一。本课程是计算机专业和其他相关专业高年级本科生、研究生的一门重要专业基础课程。

它的主要目的是讲授在计算机应用中常常遇到的重要的实际问题的解法，讲授设计和分析各种算法的基本原理、方法和技术。

本书共12章，取材先进、内容实用、重点突出、少而精、难易适当，便于自学。全书以非数值算法为主，兼顾数值算法；串行算法和并行算法并重；在附录中介绍并行MULTIPASCAL系统的使用方法，并给出一个并行程序实例。

本书可供计算机、管理信息系统、系统工程、应用数学和计算数学等专业本科生、研究生作为教材使用，也可供从事计算机科学研究、计算机软件开发的工程技术人员参考。

图书在版编目(CIP)数据

计算机算法设计与分析/苏德富，钟诚编著. - 北京：电子工业出版社，2001.1

高等学校教材

ISBN 7-5053-5871-5

Ⅰ.计… Ⅱ.①苏…②钟… 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(2000)第75187号

丛 书 名：高等学校教材

书　　名：**计算机算法设计与分析**

编 著 者：苏德富　钟　诚

责任编辑：赵家鹏

排版制作：电子工业出版社计算机排版室

印 刷 者：北京大中印刷厂

装 订 者：三河市万和装订厂

出版发行：电子工业出版社　URL:http://www.phei.com.cn

北京市海淀区万寿路173信箱　邮编100036

经　　销：各地新华书店

开　　本：787×1092　1/16　印张：12　字数：182.4千字

版　　次：2001年1月第1版　2001年2月第2次印刷

书　　号：ISBN 7-5053-5871-5/G·531

印　　数：5000册　　定价：18.00元

前　言

算法设计与分析是计算机科学的一个主要研究领域。本课程是计算机、管理信息系统、系统工程、应用数学和计算数学等专业高年级学生和研究生的一门重要专业课程。它的主要目的是讲授在计算机应用中常常遇到的实际问题的解法，讲授设计和分析各种算法的基本原理、方法和技术，以培养读者在选择或设计一个算法时，思考下列问题：这个算法是否有效？这个算法有多好？是否还有更好的算法？用什么方法和技巧去获得更好的算法？从而使得所设计算法的时空复杂性最优，进而为编写高效的程序、开发优秀软件奠定基础。

考虑到与离散数学、程序设计、计算方法、数据结构等前驱课程的联系与衔接，本书兼顾数值和非数值算法，以非数值算法为主；并力争做到取材先进、内容实用、重点突出、少而精，便于自学；并注意收录一些典型问题的最新研究成果。期望读者通过本课程的学习，在教师的指导下能较快地进入某个研究领域，接受基础研究和应用基础研究的初步训练，培养独立开展科研工作的能力和创新意识。

目前计算机正朝着微型化、并行化、网络化、智能化和多媒体化方向发展。并行分布计算技术正发挥着传统的串行计算技术所不能比拟的越来越重要的作用。因此，除了介绍串行算法之外，本书特别用较大的篇幅介绍并行算法设计技术及其分析方法。

从方便读者理解的目的出发，书中的算法用类 PASCAL 语言，或者用类 C 语言，也可以用接近自然语言的方式描述。读者可根据实际情况，用 PASCAL 或者 C 语言适当加以修改即可上机实现；对于并行算法部分，可采用支持多线程程序设计的 JAVA 语言或者采用并行 MULTIPASCAL 系统等编程上机实现。

本书由苏德富教授和钟诚副教授编写，并由钟诚统稿和配置习题。全书共分 12 章。第 1 章介绍算法分析的基本概念和基本理论，详细分析了搜索有序表二分查找算法的平均复杂性和最坏情形复杂性；第 2 章介绍设计算法的基本技术和分析算法复杂性的基本方法；第 3 章讨论若干经典数值计算问题，包括大整数相乘和矩阵乘积算法，以及数据加解密算法、数字签名和数据压缩技术；第 4 章主要介绍基于映射(散列)的排序算法和汉字字符串的排序方法；第 5 章讲授字符串精确匹配和近似匹配技术，重点剖析一些典型、能启迪人们思维的算法设计思想；第 6 章介绍并行计算基础知识和并行处理技术的应用；第 7 章主要讲授并行程序的特性，包括并行计算可扩展性、计算粒度、程序划分的条件、并行软件和硬件的匹配等问题；第 8 章从数据求和问题入手，讨论同步和异步并行求和算法的设计与分析方法；第 9 章阐述并行排序技术，其中展示了许多著名串行排序算法如何转换成并行算法，这对读者开发新的并行算法会有帮助；第 10 章介绍若干查找和串匹配问题的并行化算法，以拓宽读者的视野；第 11 章讨论一些数值问题的并行算法；第 12 章重点介绍数据库操作的并行选择算法、并行投影算法、并行集合操作算法和并行连接算法。最后，在附录中简介并行 MULTIPASCAL 系统及其使用方法，同时给出一个基于散列技术的

(m,n)选择并行算法和相应的并行程序实例。

编者自20世纪80年代中后期以来，一直从事算法设计与分析、并行计算等领域的学习、教学和研究。本书在授课讲义基础上，参考国内外有关论著编写而成。在编写过程中得到中国科技大学计算机系博士生导师、国家(合肥)高性能计算中心主任陈国良教授，武汉大学软件工程国家重点实验室博士生导师康立山教授，复旦大学计算机系博士生导师朱洪教授和暨南大学计算机系博士生导师苏运霖教授的关心和指导；国防科技大学计算机学院教授殷建平博士后也提出了宝贵的建议，在此深表感谢。

本书能在21世纪的新千年里顺利出版，我们要衷心感谢广西重点教材建设基金和广西大学的资助以及电子工业出版社和责任编辑赵家鹏老师的热情鼓励与大力帮助。同时，本书第二作者也要感谢他的家人的充分理解和支持；感谢他的研究生刘峻、廖永珺、肖立国帮助整理部分文档和绘制部分图表。

全国高校计算机教学指导委员会和中国计算机学会教育专业委员会制订的《计算机学科教学计划2000》强调要加强计算机科学与技术专业学生的算法设计与分析能力的培养。我们希望本书的出版有助于推动我国《计算机算法设计与分析》课程教学的普及和发展。

由于编者学识有限，加之编写时间较紧，书中如有不妥之处，敬请读者批评指正，以臻完善。

编著者

2000.6

目　录

第1章 引 论

众所周知，计算机要真正能够充分发挥作用离不开计算机软件，软件由计算机程序、文档和有关的控制数据组成。而计算机程序的核心则是计算机算法。可见，如果不发明更有效的算法，就不可能开发出更新更先进的软件。著名的方正排版软件系统每推出新的功能更强大的版本都是在发明新的更有效的算法基础上进行的。

一步一步解问题的过程称为算法，研究计算复杂性的典型模式是设计和提出解问题的算法所需的基本运算次数和证明其可能达到的界限。

通常所说的算法——在计算机上执行的计算过程的抽象描述和计算机程序是有区别的：程序是在指定的计算机上执行算法，而算法是抽象的，它凌驾于一切具体执行的计算机之上。我们不打算提出任何高级语言程序，但尽可能用大家熟悉的语法和句法对算法给出形式的描述。求解同一个问题 P 通常有若干种算法，因此我们要问：

(1) 什么是解问题 P 的“好”算法？为回答这个问题，我们需要比较不同算法的相对有效性。

(2) 什么是解问题 P 所需要基本运算的最小次数？

(3) 在指定的适当时间内是否有解该问题的算法？

上述问题一直是计算科学研究的核心问题之一，不少计算机科学家为此耗尽了毕生的精力，取得大量极富创造性的成果，并获得了计算机科学领域最高荣誉：计算图灵奖(Turing Awards)。

一年一度的计算图灵奖是国际计算机界公认的、权威的、影响重大和深远的一项荣誉，被誉为计算机科学领域的“诺贝尔奖”。设立图灵奖的一个主要目的是纪念计算机科学之父——图灵(Turing)。国际性图灵奖由 ACM(美国计算机学会)主持并于 1966 年开始颁奖。从 1966 年至 1999 年所颁发的图灵奖获得者中，约有 12 人是由于在算法与数据结构、计算复杂性理论、程序设计以及相关领域作出杰出贡献而荣获图灵奖。他们是：

美国斯坦福大学计算机科学系终身教授 D. E. Kunth 因撰写多卷巨著“The art of Computer programming”而于 1974 年成为最年轻的图灵奖获得者。

以色列特拉维夫大学 M. O. Rabin 教授和英国的 D. S. Scott 教授因发表著名的计算复杂性方面的论文“Finite automata and Their Decison problem”荣获 1976 年度图灵奖。

美国的 R. W. Floyd 教授由于在算法分析和程序设计正确性证明领域作出开创性的工作而荣获 1978 年度的图灵奖。

1980 年度的图灵奖则授予在程序设计和算法方面作出突出贡献的、发明著名 QUICKSORT 算法的英国的 C. A. R. Hoare 教授。

研究计算复杂性理论和 NP 完全性问题专家、加拿大 S. A. Cook 教授的论文“The complexity of Theorem Proving Procedures”荣获 1982 年度图灵奖。

提出著名公式“算法＋数据结构＝程序”瑞士的 N. Wirth 教授则成为 1984 年图灵奖

得主。

对组合优化算法、网络流有效算法、NP 完全性和随机算法研究很有造诣的、美国加州大学伯克利分校著名教授 R. M. Karp 的论文“Reducibility among combinatorial problems”荣获 1985 年度图灵奖。

更有意思的是美国教授 J. E. Hopcrft 和他的学生 R. E. Tarjan 博士密切合作、发明了一种非常重要的数据结构及其算法从而荣获 1986 年度图灵奖。

在计算复杂性领域作出卓越贡献的美国的 J. Hartmanis 和 R. E. Steams 教授荣幸地于 1993 年走上图灵奖领奖台。

两年之后的 1995 年，图灵奖颁发给在计算复杂性、密码学算法和程序验证技术取得重要成就的美国教授 M. Blum。

由此可见研究算法对推动计算机科学与技术的研究和发展具有十分重要的作用。

当 Karp 站在图灵奖的领奖台上时，思绪万千，他回顾自己的经历时说：“我进入计算机领域甚为偶然。1955 年从哈佛学院毕业并取得数学学位后，我面临的问题是决定下一步干什么。为谋生而工作，对我没有什么吸引力，因此鲜明地选择了研究生院。……，尽管课程缺乏深度和一致性，但学习空气特别浓厚，我们知道我们正在目睹着一门以计算机为中心的科学诞生。我发现我在算法的结构中找到了美和雅致，……，简言之，多少出于偶然，我摸索着进入了一个我极其喜欢的领域。”（参见 Karp：“组合论、复杂性和随机性”，CACM，Vol. 29，NO. 2，1986）。

另一位图灵奖获得者 Hopcrft 在谈到美国的现状时说：“教育机构和研究室对计算机科学家的需求增长快于这一领域为培养所需合格人材而构造必要的基础结构的速度。”并呼吁美国政府：“今天全球都在力争取得技术和经济的领导地位。计算技术在这一斗争中将起关键性的作用。除非我们提出一种全国性的政策来支持计算机科学，否则由于我们无所作为，就会使其他国家在计算机技术方面建立我们不可能赶上的领先地位。”（参见：CACM，Vol. 30，NO. 3，1987）。事实上，20 世纪 90 年代发生的两场战争的胜利事实上是高技术的胜利，而高性能计算技术则从中扮演非常重要的作用，这充分说明了 Hopcrft 的预言是正确的。

那么，面对这一世界性的挑战，有幸进入计算机科学领域的中国青年学者应该怎么办？

1.1 算法分析的基本概念和理论

在 20 世纪 60 年代，对于算法研究的现状是很不能令人满意的。那时，对算法的有效性没有一个令人满意的客观标准，一个研究者可能会在杂志上发表一个算法及对于少数例题的执行时间；而几年后，第二名研究者又会给出一个改进的算法以及对同样例题的执行时间，新的算法肯定更快，因为这相间的几年里，计算机性能和程序设计语言都得到改善。算法在不同的计算机上运行并且编程语言也不同，这使得我们对这种比较感到不满意。要弄清楚计算机性能提高与实现编程技术对算法执行时间的影响是很困难的，再者，

有可能第二个研究者无意中将其算法搞得对这些例题特别有效。可以预见，如果对于另外的例题再运行这两个算法，第一个算法就可能更快。

因此，对算法的分析必须脱离具体的计算机结构和程序设计语言。

比较两个算法的好坏，看其所需的运算时间，时间的长短是由算法所需的运算次数决定的。任何一个算法都可能有几种运算，因此，我们抓住其中影响算法运行时间最大的运算作为基本运算。如在一个字表中寻找 Z 的问题，把 Z 和表中元素的比较作为基本运算。两个实数矩阵相乘的问题中，则把两个实数相乘作为基本运算。

但是同一个问题对不同的输入，基本运算的次数亦可能不同。因此，我们引进问题大小(即规模，Size)的概念。例如，在一个姓名表中寻找给定的 Z，问题的大小可用表中姓名的数目表示。对于两矩阵相乘问题，其大小可用矩阵的阶表示。而对于遍历一棵二叉树的问题，其大小是用树中的节点数来表示等等。这样，一个算法的基本运算的次数就可能用问题的大小 n 的函数 $f(n)$来表示。$f(n)$是 $N\to N$ 的一个函数。

下面介绍本书常用的记号：

$|S|$：S 的基数(即 S 中元素的个数)，此处 S 是一个有限集合；

$|x|$：x 的长度(即 x 中字符的个数)，此处 x 是一个字符串；

$\Omega(f(n))$：阶至少为 $f(n)$的函数；

$O(f(n))$：阶至多为 $f(n)$的函数；

$\Theta(f(n))$：阶恰好为 $f(n)$的函数；

$\lfloor x \rfloor$：小于或等于 x 的最大整数，简称 x 的向下取整(floor of x)；

$\lceil x \rceil$：大于或等于 x 的最小整数，简称 x 的向上取整(ceiling of x)；

$\log n$：$\log_2 n$；

$\ln x$：x 的自然对数，即以 e 为底的对数；

∞：无穷大，或在讨论数据域或变量的值时，某个远大于数据论域中最大元素的某个数据；

对于算法的优劣，通常以平均性态和最坏情形两种结果来衡量。

1. 算法的平均情形复杂性(Average behavior)

设 D_n 是对于所考虑问题来说大小为 n 的输入的集合，并设 I 是 D_n 的一个元素，$p(I)$是 I 出现的概率，$t(I)$是算法在输入 I 时所执行的基本运算次数。那么，算法的平均复杂性定义为：

$$A(n) = \sum_{I \in D_n} p(I)t(I)$$

2. 最坏情形复杂性：(Worst-case complexity)

$$W(n) = \mathrm{MAX}_{I \in D_n}\, t(I)$$

例 1.1　考虑顺序查找算法：设 L 是一个有 n 个数据的线性表(数组)。对于某个指定的 z，

如果 z 在表 L 中，则查找出它在表 L 的下标；如果 z 不在表 L 中，则返回结果零。

算法 1.1　Sequential Search(L,n,z)

```
Begin
  j=1;
  while j<=n and L[j]≠z do
    j=j+1;
  if j>n then
    j=0;
  writeln ('j=',j);
End
```

上述顺序查找算法的基本运算为将 z 和表 L 中数据项作比较操作。现在分析算法的平均性态。

设 I_i 表示 z 出现在表中第 i 个位置的情况。$t(I_i)$ 表示输入 I_i 的比较次数。所以

$$t(I_i) = i,\ 1 \leqslant i \leqslant n,\ t(I_{n+1}) = n + 1$$

设 q 表示 z 出现在表 L 中的概率，并假定它出现在表 L 中任何位置的可能性(概率)是一样的。那么，$p(I_i)=q/n$ 且 $p(I_{n+1})=1-q$，从而

$$\begin{aligned} A(n) &= \sum_{i=1}^{n+1} p(I_i)t(I_i) = \sum_{i=1}^{n} (q/n^* i) + (1-q)(n+1) \\ &= q/n\sum_{i=1}^{n} i + (1-q)(n+1) \\ &= q/n^* n(n+1)/2 + (1-q)(n+1) \\ &= q(n+1)/2 + (1-q)(n+1) \end{aligned}$$

如果 z 肯定出现在表 L 中，即 $q=1$；则

$$A(n)=(n+1)/2$$

这说明平均要查找半个线性表。

如果 z 只有一半的可能在线性表 L 中，即 $q=1/2$，则

$$A(n)=(n+1)/4+(n+1)/2\approx 3/4 * n$$

这说明平均需要查找线性表中 3/4 的元素。

而算法在最坏情况下，所需的时间是：

$$w(n)=\mathrm{MAX}\{t(I_i): 1\leqslant i\leqslant n+1\}=n+1$$

即 z 不出现在表 L 中或出现在表 L 的最后一个位置时，都要查找整个线性表 L。

例 1.2　已知矩阵 $A=(a_{ij})_{n\times n}$，$B=(b_{ij})_{n\times n}$，求乘积 $C=A\times B$。

计算乘积的方法是：$C_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj},\ 1 \leqslant i,\ j \leqslant n$

算法 1.2　Martrix Multiplication (A,B,C,n)

```
Begin
  for i=1 to n do
    for 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;
End
```

矩阵相乘算法的基本操作是矩阵元素相乘操作。

算法复杂性分析：

对于矩阵 C 中的每一个元素都需要做 n 次乘法操作，而 C 共有 n^2 项，因此

$$A(n)=w(n)=n^3$$

但是，仅仅通过算法平均情形和最坏情形的分析还不能完全说明一个算法的有效性。例如，对于同一个问题 P，假设一个算法在最坏情形下的时间复杂性为 $w_1(n)=3n^2$，而另一个算法的最坏情形复杂性为 $W_2(n)=25n$。当 $n=8$ 时，$W_1(8)=192$，$W_2(8)=200$；显然，$W_1(8)<W_2(8)$，即 $W_1(n)$ 比 $W_2(n)$ 的运算次数少。但是，当 $n=9$ 时，$W_1(9)=243$，$W_2(9)=225$ 并且当 $n\geqslant 9$ 时，都有 $W_1(n)>W_2(n)$ 成立，即第一个算法比第二个算法速度慢。

因此，我们需要考虑当 n 足够大时对算法进行比较，即讨论算法的渐近性态分析。

定义 1.1 设 $f(n)$ 和 $g(n)$ 是定义域为自然数 N 的非负函数，如果存在正常数 C 和 n_0，使得对于所有的 $n\geqslant n_0$，$f(n)\leqslant Cg(n)$ 成立。那么，称 $f(n)$ 的阶低于 $g(n)$ 的阶，记为 $f(n)$ 是 $O(g(n))$ 或 $f(n)\in O(g(n))$ 或 $f(n)=O(g(n))$。读作 $f(n)$ 是 $g(n)$ 的大 O。

例如，$f(n)=3^n$ 不是 $O(2^n)$。因为假设存在正常数 C 与 n_0，使得当 $n\geqslant n_0$ 时，有 $3^n\leqslant C2^n$，那么，$C\geqslant(3/2)^n$ 对于一切 $n>n_0$ 成立。但是，我们知道随着 n 的增大，$(3/2)^n$ 可以无限地增大，因此只要 n 充分大，就可以使 $(3/2)^n$ 大于任意预先指定的常数 C。这说明所假设的常数 C 不存在。

定义 1.1　指出一个算法（程序）的运行时间函数为 $O(g(n))$，也就是说该算法（程序）的时间增长率的上界为 $g(n)$。

定义 1.2　设 $f(n)$ 和 $g(n)$ 是定义域为自然数 N 的非负函数，如果存在正常数 C_0，C_1 和 n_0，使得对于所有的 $n\geqslant n_0$，有 $C_0g(n)\leqslant f(n)\leqslant C_1g(n)$，则称 $f(n)$ 恰好与 $g(n)$ 同阶，记为 $f(n)=\Theta(g(n))$。

与 $f(n)$ 同阶的函数的集合记为 $\Theta(f(n))$。

例如，$f(n)=n^2/2$，$g(n)=307n^2=\Theta(n^2/2)$；又例如，$f(n)=n^4$，$g(n)=5n^4+3n^2+1=\Theta(n^4)$。

定义 1.3　对于定义在自然数 N 上的非负函数 $f(n)$ 和 $g(n)$，若存在正常数 C，对于无穷多个 n，使得 $f(n)>Cg(n)$ 成立，则称 $f(n)$ 是 $\Omega(g(n))$（读作大奥米伽 $g(n)$）。

当 $f(n)=\Omega(g(n))$ 时，称 $g(n)$ 是 $f(n)$ 在这无穷多个 n 上的增长率的下界。

例如，设 $f(n)=n^4+2n^2$，令 $C=1$，则 $f(n)>Cn^3$，对于无穷多个 n 成立，所以 $f(n)$ 是 $\Omega(n^3)$。

又例如，设 $f(n)=\begin{cases}n & \text{当 } n \text{ 为非负奇数时}\\ n^2/2 & \text{当 } n \text{ 为非负偶数时}\end{cases}$

则只需令 $C=1/100$，并考虑全体非负偶数无穷集合上的取值情况即可。

比较两个算法的时间复杂性函数 $g(n)$和 $f(n)$的阶的方法除了可用定义判断外，还可以采用求极限的方法来加以判断。

若$\lim_{n\to\infty} f(n)/g(n)=C$

则

(1)当 $C\neq 0$ 时，说明 $f(n)$和 $g(n)$同阶，记为 $f(n)=\Theta(g(n))$

(2)当 $C=0$ 时，说明 $f(n)$比 $g(n)$低阶，记为 $f(n)=O(g(n))$

(3)当 $C=\infty$时，说明 $f(n)$比 $g(n)$高阶，记为 $f(n)=\Omega(g(n))$

例如，设 $f(n)=n^2/2$，$g(n)=307n^2$，则

$$\lim_{n\to\infty}(n^2/2)/(307n^2)=1/614$$

因此，$f(n)=n^2/2$ 与 $g(n)=307n^2$ 是同阶的。

又例如，设 $f(n)=\log n$，$g(n)=n$，则

$$\lim_{n\to\infty}\log n/n=\lim_{n\to\infty}\ln n/(n\ln 2)=(1/\ln 2)\lim_{n\to\infty}((1/n)*\ln n)=0$$

所以，$f(n)=\log n$ 比 $g(n)=n$ 低阶。

定义 1.4　时间复杂性达到下界的算法称为最优算法。

1.2　搜索有序表算法的分析

为了阐述上节所提出的基本概念，我们将仔细地研究以下问题。

1. 搜索有序表问题

给定一个含有 n 个数据项的已按非减顺序排列的表 L，并给定一个值 z，在表 L 中寻找 z 出现的一个下标，若在表 L 中 z 不存在，则返回结果 0。

我们看到顺序搜索算法(算法 1.1)虽然可以解决这个问题，但是由于它没有利用表 L 中的数据项有序这个特点，所以查找速度较慢。我们能否改进这个算法使它能利用到这些附加(有序)信息，从而只需作较少的比较操作就可以完成查找问题呢？由于表 L 是非递减顺序的，所以一旦查找到了一个比 z 还要大的数据项，算法就可结束，其答案为 0。这个修改对时间复杂性分析有怎样的影响呢？很显然，修改过的算法在某些情况下有所改进，即对某些输入它能使得算法较快结束。然而，最坏情形复杂性仍然保持不变。若 z 是表中最大的数据项或 z 比表 L 中所有的数据项都要大，则算法需要做 n 次比较。

修改过的算法平均复杂性是否好一些呢？如果 z 有一半的机会出现在表 L 中，且 z 出现在表 L 中每一个位置的可能性是一样的话，则顺序搜索算法大约作 $3n/4$ 次比较，对于上述修改过的算法，我们还必须考虑 z 出现在表 L 中两个数据项之间的可能性。若定义间隙 g_i 是对于 $i=2,\ldots\ldots,n$ 满足 $L[i-1]<y<L[i]$ 的 y 的集合，同时设 g_1 是小于 $L[1]$的所有值，g_{n+1}是大于 $L[n]$的那些值。现在，为了对修改过的算法进行平均性态分析，

设 z 有一半的机会出现在表 L 中，若它出现在表 L 中的话，则它在表 L 中所有位置的机会是一样的(则概率为 $1/(2n)$)；假设若 z 不出现在表 L 中的话，则 z 落入所有间隙中的可能性是一样的(即概率为 $1/(2(n+1))$)。对于 $1\leqslant i\leqslant n$，需要作 i 次比较来确定 $z=L[i]$ 或者 z 落入 g_i 中，并且需要作 n 次比较来确定 z 是否落入 g_{n+1} 中。于是，我们可以计算上述修改算法的平均比较次数。

$$A(n)=\sum_{i=1}^{n}(1/2n*i)+1/(2(n+1))*n$$

上式的第一项为 z 出现在表 L 中的情况，而后一项对应于 z 不出现在表 L 中的情况。计算结果为 $A(n)\approx n/4$，即算法平均需要查找四分之一张表。

至此我们还没有对算法 1.1 进行实质性的改进，但是上述的努力是有意义的。我们并不期待对每一个问题在一开始尝试时就能设计出理想的算法。相信读者见到容易的算法分析的例子越多，他为处理复杂问题的准备就越充分。

现在，能否设计一个算法，在最坏情况下，实质上做少于 n 次的比较就能查找确定 z 是否出现在一个有 n 个数据项的有序表 L 中呢?假设我们写一个算法，比如将 z 与表 L 中的每第 4 个项进行比较，如果存在一次匹配，算法就结束，如果在某一次与 z 相比较的数据项大于 z，则说明 z 位于与它所比较的最后两个数据项之间，接着再进行很少的几次(多少次?)比较就能够确定出 z 的位置(若它在表中的话)，或者确定出 z 不在表 L 中。

这个算法的细节和分析留给读者，但是很容易看出表 L 中大约只有四分之一的数据项是被检查过的，即与 z 比较过的。因此，在最坏情形下，大约进行了 $n/4$ 次比较。这肯定是对顺序搜索算法的一个改进。

我们可以沿着这条思路，设计一个算法将 z 与表 L 中每第 k 个项进行比较，同时选取较大的 k 值，但是在减少为寻找包含 z 的表的子部分(若 z 在表中)所需比较次数的同时，已增加了子部分的尺寸从而增加了在其中确定的 z 位置所需的另外的比较次数。

假如我们采用稍微不同的技巧，则可以首先将 z 与表 L 中间的那个数据项进行比较，而不是选择一个特定的数 k 将 z 与第 k 个数据项进行比较。如果 z 较大，则它肯定出现在表 L 中的话，说明它将出现在表 L 的后半部分。这样，我们只用了一次比较，整个表 L 的前半部分就可以不加考虑了。反之，若 z 小于等于表 L 中间的那个数据项，则表 L 的后半部分也就可以不加考虑了(当然，若 z 等于表 L 中间的那个数据项，则搜索可马上结束，不需要再作更多的比较)。依次类推，继续将 z 与表 L 中当前所考虑部分的中间数据项进行比较，直到找到了 z 或者确定 z 不出现在表 L 中时为止。在每一次比较之后，表 L 中可能包含 z 的那部分的大小(数据个数)就削减一半。这种查找方法称为二分搜索(折半查找，*Binary Search*)。下面我们给出二分搜索算法的形式描述。

算法 1.3　Binary Search(L,n,z)

```
Begin
  k=1;
  m=n; /* k 和 m 分别是数组中目前正搜索的那一部分的第一项和末项的下标 */
  flag=0;    /* flag 为标志变量 */
```

```
(1) while k<=m and flag=0 do
        begin
(2)         j= ⌊(k+m)/2⌋ ;     /* 中间元素的下标 */
(3)         if z=L[j] then
                flag=1
(4)         else if z<L[j] then
                m=j-1
(5)         else
                k=j+1
        end;
    if flag=1 then
        writeln('j=',j)
    else
        writeln('j=',0);
End
```

2. 算法复杂性最坏情形分析

设 $W(n)$是最坏情况下，在 n 个数据项的有序表 L 上，二分搜索算法所执行的基本运算(即 z 和表 L 中数据项比较)的次数。假设在算法的第 3 行至第 5 行中，为测试 z，进行的是具有三叉分支的一次比较；因此，$W(n)$是通过 while 循环的次数。

设 $n>1$，在第一次碰到第 1 行时，算法的任务就是要在下标从 $k=1$ 到 $m=n$ 的 n 个数据项的表 L 中寻找 z，算法进行到第 3 行，并将 z 和 $L[\lfloor(1+n)/2\rfloor]$比较。在最坏情形下，它们不相等而且 k 或者 m 将发生变化，于是到下一次循环的任务就是在表 L 中下标从 k 到 m 的部分中继续寻找 z。

在这个新的部分中若 n 是偶数，表中在 $L[\lfloor(1+n)/2\rfloor]$的后半部分中有 $n/2$ 个数据项。在前半部分有$(n/2-1)$个数据项。若 n 为奇数，则前半部分和后半部分都有$(n-1)/2$ 个数据项。

于是，在下一次循环时，算法要在其中寻找 z 的表 L 的那一部分至多有 $\lfloor n/2\rfloor$ 个数据项。从第 2 次循环开始，算法所做的比较等于输入是一个具有 $\lfloor n/2\rfloor$ 个数据项的表 L 时算法所做的比较次数。因此，$W(n)=1+w(\lfloor n/2\rfloor)$。这是一个递推关系的方程例子。而且容易看出 $W(0)=0$。

我们也可以计算 $W(1)$。若 $n=1$，则 $k=m=j=1$，z 和 $L[1]$作比较；如果第 3 行的条件 n 没有满足，那么 m 将置成 0 或者 k 将置成 2，此时循环结束。于是，无论 z 是否出现在表 L 中，算法仅做了一次比较操作，所以 $W(1)=1$。因此，我们有下列递推方程成立

$$W(n)=1+w(\lfloor n/2\rfloor), \quad n>1$$

$$W(n)=1, \quad n=1$$

当一个函数用递推关系描述时，给出自变量取某个特定值的函数的方程称为边界条件。例如，方程 $W(1)=1$ 是自变量 n 的边界条件。