普通高等教育“十一五”国家级规划教材

21世纪大学本科
计算机专业系列教材

算法设计与分析

屈婉玲 刘 田
张立昂 王捍贫 编著

李晓明 主审

- 根据教育部“高等学校计算机科学与技术专业规范”组织编写
- 与美国 ACM 和 IEEE CS *Computing Curricula* 最新进展同步

http://www.tup.com.cn

青华大学出版社

“十一五”国家级规划教材

21世纪大学本科计算机专业系列教材

算法设计与分析

屈婉玲 刘田 张立昂 王捍贫 编著

李晓明 主审

清华大学出版社

北京

内 容 简 介

本教材为计算机科学技术专业核心课程“算法设计与分析”教材. 全书以算法设计技术和分析方法为主线来组织各知识单元，主要内容包括基础知识、分治策略、动态规划、贪心法、回溯与分支限界、算法分析与问题的计算复杂度、NP 完全性、近似算法、随机算法、处理难解问题的策略等. 书中突出对问题本身的分析和求解方法的阐述，从问题建模、算法设计与分析、改进措施等方面给出适当的建议，同时也简要介绍了计算复杂性理论的核心内容和处理难解问题的一些新技术.

本书有配套的学习指导与习题解析用书以及 PPT 电子教案.

本书可作为大学计算机科学与技术、软件工程、信息安全、信息与计算机科学等专业本科生和研究生教学用书，也可以作为从事实际问题求解的算法设计与分析工作的参考书.

图书在版编目（CIP）数据

算法设计与分析 / 屈婉玲等编著. —北京：清华大学出版社，2011.5
（21 世纪大学本科计算机专业系列教材）
ISBN 978-7-302-24756-2

Ⅰ. ①算… Ⅱ. ①屈… 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. ①TP301.6

中国版本图书馆 CIP 数据核字(2011)第 026141 号

责任编辑：张瑞庆　薛　阳
责任校对：梁　毅
责任印制：何　芊

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62795954，jsjjc@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市春园印刷有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　　张**：14.5　　**字　　数**：361 千字
版　　次：2011 年 5 月第 1 版　　**印　　次**：2011 年 5 月第 1 次印刷
印　　数：1～5000
定　　价：25.00 元

产品编号：041680-01

前言

FOREWORD

作为问题求解和程序设计的重要基础，“算法设计与分析”在计算机科学与技术专业的课程体系中是一门重要的必修课。通过该课程的学习，不但为学习其他专业课程奠定了扎实的基础，也对培养学生的逻辑思维和创造性有着不可替代的作用. ACM IEEE Computing Curricula 2004 与我国教育部计算机科学与技术专业教学指导委员会提出的《计算机科学与技术专业规范 2005》都把该课程列入本专业的核心课程之一. 纵观计算机学科数十年发展的历史，算法与计算复杂性理论一直是计算机科学研究的热点和活跃领域，也是获得图灵奖最多的研究领域之一. 面对计算机应用领域的大量问题，最重要的是根据问题的性质选择正确的求解思路，即找到一个好的算法. 特别在复杂的、海量信息的处理中，一个好的算法往往起着决定性的作用.

目前已经出版了许多算法教材，各有特色. 作为国家高等教育“十一五”规划教材，我们在多年从事算法设计分析及计算复杂性理论的教学和研究的基础上，精心选材，完成了本书的写作. 本书的主要特点是：

(1) 以算法设计技术为主线来组织素材，深入分析了各种设计技术的适用范围、设计步骤、算法的正确性证明与复杂度的分析方法、改进算法的途径、局限性等. 与通常的算法与数据结构教材有所不同，本书不打算过多地关注实现细节，算法描述采用伪码，力求突出对问题本身的分析和求解方法的阐述，从问题建模、算法设计与分析、改进措施等方面给出适当的建议，为从事实际问题的算法设计与分析工作在理论上提供了清晰的、整体的思路和方法.

(2) 从对具体算法的设计技术与分析方法，自然过渡到对问题难度的分析和界定. 这里处理的不是一个具体的算法，而是对求解该问题的一大类算法的评价和分析，是对问题本身计算复杂度的估计. 基于这种分析，就能回答“求解该问题的最好算法是什么”，“它能好到什么程度”等问题，从而为选择最好的算法给出依据. 一般的算法教材涉及这方面的知识比较少，本书比较系统地介绍了一些关于问题复杂度的分析方法.

(3) NP 完全理论是计算复杂性理论的核心内容，其中“**P≠NP**”的问题是 21 世纪最重要的数学难题之一. 计算复杂性理论关注的不仅仅是某个具体问题，而是希望了解每一个问题在计算难度的层次结构中到底处于什么位置，不同问题的难度之间有什么关系. 本书力求用清晰易懂的语言，对计算复杂性理论的核心内容和针对难解问题的处理策略加以简单的介绍，希望为那些从事复杂问题求解的读者提供一点帮助.

(4) 本书的素材来自多年的教学积淀，选材适当，组织合理，先引入基本概念和数学基

础知识，然后进入算法设计与分析的核心内容. 在叙述中不但注意理论的严谨，也精选了大量生动有趣的例子. 每章都配有难度适当的练习，适合教学使用.

全书共 10 章，第 1 章是基础知识，介绍和算法设计与分析有关的基本概念、符号和数学知识；第 2～5 章分别阐述分治策略、动态规划、贪心法、回溯与分支限界等算法设计技术；第 6 章介绍算法分析与问题的计算复杂度；第 7 章是 NP 完全性理论；第 8 章是近似算法；第 9 章是随机算法；第 10 章介绍处理难解问题的策略.

本书既可以作为本科生教材，也可以作为研究生教材. 对于本科生教学，建议讲授第 1～6 章的全部内容，第 7～10 章可根据情况选择部分内容做一些概括性的介绍. 研究生教学可以选择第 1～9 章的全部内容，第 10 章可根据情况选讲. 此外，对于从事实际问题求解的研究工作者，本书也可以作为一本算法设计与分析的入门参考书.

为了更好地为使用本教材的读者服务，作者正在撰写和开发与本教材配套的教学辅导书和 PPT 电子教案.

本书的第 1～4 章由屈婉玲完成，第 5～6 章由王捍贫完成，第 7～8 章由张立昂完成，第 9～10 章由刘田完成.

在编写过程中，作者参考了国内外多种版本的算法设计与分析以及计算复杂性方面的教材、论文和专著，从中吸取了一些好的思路和素材，在此一并向有关作者致谢. 特别感谢李晓明教授审阅了初稿并提出了宝贵意见，感谢清华大学出版社对本书出版的大力支持. 我们期待着广大读者，特别是使用本书的教师和学生对本书的批评、指正和建议.

作　者

2010 年 11 月写于北大

目 录

CONTENTS

第 1 章 基础知识

1.1 有关算法的基本概念

对于给定的问题，一个计算机算法就是用计算机求解这个问题的方法. 一般来说，算法由有限条指令构成，每条指令规定了计算机所要执行的有限次运算或者操作. 下面先看几个具体例子.

例 1.1 有 n 个客户带来 n 项任务等待在一个机器上加工，这些任务记为 $1,2,\cdots,n$. 设第 j 项任务的加工时间是 t_j，$j=1,2,\cdots,n$. 一个可行的调度方案是 $1,2,\cdots,n$ 的一个排列 $i_1, i_2,\cdots,i_n$. 对于排在第 j 个位置的任务 i_j，这个客户所需要等待的时间是任务 i_j 本身的加工时间与排在它前面的所有任务加工时间之和，即 $T(i_j)=t_{i_1}+t_{i_2}+\cdots+t_{i_{j-1}}+t_{i_j}$. 而所有客户的总等待时间就是每项任务的等待时间之和. 为了使得尽可能多的客户满意，我们希望找到使得总等待时间最少的调度方案.

为了完成算法的设计与分析，需要用数学语言将这个问题加以形式化描述. 先分析一下总等待时间的构成：第 1 项任务只需要等待自己的加工时间，第 2 项任务需要等待前两项任务的加工时间，……，第 n 项任务需要等待所有 n 项任务的加工时间. 因此，在总等待时间的计算公式中，第 1 项任务的加工时间出现 n 次，第 2 项任务的加工时间出现 $n-1$ 次，……，第 k 项任务的加工时间出现 $n-k+1$ 次. 因此这个调度问题可以描述如下：

问题：给定任务集合 $S=\{1,2,\cdots,n\}$，$\forall j\in S$，加工时间 $t_j\in\mathbf{Z}^+$，$\mathbf{Z}^+$ 为正整数集合，求 S 的排列 $i_1, i_2,\cdots,i_n$ 使得 $\sum_{k=1}^{n}(n-k+1)t_{i_k}$ 最小.

一种可能的想法就是“加工时间短的任务先做”. 把任务按照加工时间的递增顺序排序，即如果 $i<j$，那么 $t_i\leqslant t_j$. 然后按照标号从小到大的顺序进行加工. 例如，给定 $S=\{1,2,3,4\}$，$t_1=5$，$t_2=20$，$t_3=3$，$t_4=15$. 那么这种加工顺序就是 3，1，4，2. 按照这种顺序的总等待时间是 $T=3\times4+5\times3+15\times2+20=77$.

以上设计的算法是一种贪心算法. 我们需要考虑的问题是：

(1) 如何一步一步地给出这个算法的清晰描述？

(2) 这个算法是否对所有有效的输入都能得到正确的解？如果能，怎样证明？如果不能，能否举出反例？

我们还可以进一步考虑，什么类型的问题可以使用贪心算法的设计技术？如果对某些问题，贪心算法不能得到正确的解，还有没有其他设计技术？回答这些问题涉及算法的描述、算法的设计技术、算法的正确性证明等，这些都与算法设计与分析领域的基本理论及方法相关. 它们构成了这本书的主要内容.

例 1.2 排序问题是大家所熟悉的问题，已经存在的有插入排序、冒泡排序、选择排序、快速排序、堆排序等多种算法. 其中哪些算法运行效率更高？是否存在更好的算法？为解决这些问题，需要对算法的运行效率给出科学的量化定义——算法在最坏情况和平均情况下的时间复杂度，并给出计算这两种时间复杂度的方法. 例如上述排序算法中插入排序与冒泡排序都是 $O(n^2)$[①]时间的算法，而快速排序在最坏情况下是 $O(n^2)$ 时间的算法，平均情况下是 $O(n\log n)$ 时间的算法. 堆排序在两种情况下都是 $O(n\log n)$ 时间的算法. 这些都是针对一个给定算法的分析工作. 是否存在更有效的算法则涉及对问题难度的界定，与一个具体算法的分析相比，这是不同层次的问题. 问题难度是由问题本身的内在性质决定的，与求解它的具体算法无关. 希望能对问题难度给出界定，从而为设计与评价相关的算法提供可靠依据.

遗憾的是，目前只有少量问题得到了有关难度下界的分析结果. 比如排序问题，已经证明了对于 n 个元素进行排序，任何用元素的比较运算进行排序的算法在最坏及平均情况下至少要做 $O(n\log n)$ 次比较. 这说明在这个算法类中不可能找到时间复杂度比 $O(n\log n)$ 更好的排序算法(对某些特定数组的排序，如果算法的运算不限于元素之间的比较，则存在更好的算法)，这也意味着现有的某些算法已经是求解排序问题的最有效的算法.

有关算法以及问题复杂度的分析也是本书的主要内容.

例 1.3 货郎问题 设有 m 个城市，已知其中任何两个城市之间(不经过第三个城市)道路的距离. 一个货郎需要到每个城市巡回卖货，他从某个城市出发，每个城市恰好经过一次，最后回到出发的城市. 问怎样走使得总的路程最短？

先建立问题的数学模型. 即

问题：有穷个城市的集合 $C=\{c_1,c_2,\cdots,c_m\}$，距离 $d(c_i,c_j)=d(c_j,c_i)\in \mathbf{Z}^+$，$1\leqslant i<j\leqslant m$.

求 $1,2,\cdots,m$ 的排列 $k_1,k_2,\cdots,k_m$ 以求得最小值

$$\min\left\{\sum_{i=1}^{m-1} d(c_{k_i},c_{k_{i+1}})+d(c_{k_m},c_{k_1})\right\}$$

下面给出问题的一个实例，即

$$C=\{c_1,c_2,c_3,c_4\},\quad d(c_1,c_2)=10,$$
$$d(c_1,c_3)=5,\quad d(c_1,c_4)=9,$$
$$d(c_2,c_3)=6,\quad d(c_2,c_4)=9,\quad d(c_3,c_4)=3$$

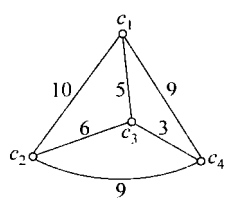

图 1.1 货郎问题的实例

这个实例也可以用图来表示，见图 1.1. 上述问题就是著名的货郎问题. 使用图论的术语，这个问题可以表述为：如何在带权完全图 G 中找一条最短哈密顿(Hamilton)回路.

假定从 c_1 出发，一种可行的算法就是穷举所有可能的排列 $i_1=1,i_2,\cdots,i_m$(代表旅行路

① 关于 $O(\ )$ 记号，后面将给出定义.

线 $c_1, c_{i_2}, \cdots, c_{i_m}, c_1$），然后针对每个排列计算路径长度之和，从而确定最短路径. 一般把这种算法称为“蛮力算法”. 对于任何组合问题都有一个搜索空间，搜索空间的某些结点对应了问题的可行解，而其中有的结点对应了最优解. 蛮力算法不需要分析解之间的关系以利用其相关的信息，也不需要考虑其他加快搜索的技术，因此是一种很笨的算法，效率不高. 我们所考虑的算法设计技术，是通过分析问题的结构找到最好的搜索策略，以得到比蛮力算法更高的效率.

和例 1.1 不同，对于货郎问题，到目前为止还没有找到比蛮力算法有着本质改进的高效算法（指算法的时间复杂度由指数函数降为多项式函数）. 这就促使我们考虑更多的问题：如果对一个问题经过多年的努力还没有找到有效的算法，这是否揭示了这个问题本质上的“难解性”？这类问题大量存在于各个不同的研究和应用领域. 自从 20 世纪 70 年代建立了计算复杂性的核心理论——NP 完全性理论以来，已经发现了数千个判定问题（问题的解是 Yes 或者 No. 货郎问题是其中之一. 考虑图 1.1 的例子，如果问：是否存在一条不超过 20 的旅行路线？这就变成了货郎判定问题的一个实例）. 可以证明这些判定问题的难度彼此等价. 换句话说，如果其中一个问题能够有效求解，那么所有这些问题都能有效求解；如果证明了其中一个问题没有有效的算法，那么所有这些问题都是难解的. 在计算复杂性理论中把这类判定问题称为 NP 完全问题①. 我们关心的是：NP 完全问题到底是不是难解的？经过数十年的研究已经得到一些结果，但是到目前为止还没有本质上的突破. 这个问题就是著名的“**P=NP**?”的问题，已列入 21 世纪数学界最重要的问题之一. 与此相关的是在各个应用领域有大量的实际问题，已经证明其难度至少和 NP 完全问题一样，人们不得不为处理这类问题寻找各种应对的策略.

上面三个例子分别显示了从某个具体算法的设计与分析技术，到针对某个问题的算法类的分析技术——问题难度的确认，最后到不同问题之间的难度分析——计算复杂性理论这样三种不同的研究层次. 我们将按照这样一种由低到高、由具体到抽象的思路来安排各章的内容. 在介绍了 NP 完全理论之后，将对近似算法、概率算法等比较活跃的研究方向，以及面对 NP 难问题的应对策略加以介绍.

先说明一些有关算法的基本概念.

问题是需要回答的一般性提问，通常含有若干参数. 为了清晰地描述一个问题，需要说明参数的含义和解所满足的条件. 如果对问题参数给出一组赋值，就得到了问题的一个实例. 例 1.3 的货郎问题就是这样描述的. 今后所提到的问题都是指它的所有实例构成的抽象描述，不是特指一个具体的实例.

算法是有限条指令的集合，这个指令集确定了解决某个问题的运算或操作的序列. 算法 A 解问题 P 是指：把问题 P 的任何实例作为算法 A 的输入，A 能够在有限步停机，并输出该实例的正确解.

算法的时间复杂度是对算法效率的度量. 如何度量一个算法的效率？理论上不能用算法在机器上真正的运行时间作为度量标准. 因为运行时间依赖于机器的硬件性能，如 CPU 速度等，也依赖于程序的代码质量. 我们希望这个度量能够反映算法本身的性能. 因此，一般的做法是针对问题选择基本运算，用基本运算次数来表示算法的效率，运算次数越多，效率

① 关于 NP 完全问题将在第 7 章给出定义.

就越低. 为了给出时间复杂度的清晰定义,有两个问题需要解决. 第一个问题是算法运算次数与问题的实例规模有关(关于实例规模的精确定义见第 7 章),如何表示这种依赖关系? 比如对整数数组排序,其基本运算是数之间的比较. 显然 100 000 个数的数组比10 个数的数组所需要的比较次数要多得多,在这里数组的大小 $n=100\ 000$ 或 10 代表了不同的实例规模. 可以用函数来解决这个问题,即把排序算法对规模为 n 的输入实例所做的比较次数看作 n 的函数 $T(n)$,即算法的时间复杂度函数. 第二个问题是对规模为 n 的两个不同的输入实例,算法的运算次数也可能不一样,选择哪一个作为它的时间复杂度函数呢? 比如插入排序. 假如算法的输入是 3,2,5,4,1. 初始输出数组只保留第一个数 3,然后算法依次在这个数组中插入 2,5,4,1. 插入时,把要插入的元素从后向前与数组中的数比较,找到应该插入的正确位置. 比如,首先把 2 与 3 比较;因为 2 比 3 小,把 2 插到 3 的前面,数组变成 2, 3. 接着把 5 与 3 比较,5 比 3 大,因此 5 插到 3 的后面,数组变成 2, 3, 5. 类似地,在3 和 5 之间插入 4;最后在 2 前面插入 1,从而得到输出 1,2,3,4,5. 如果输入数组是 $1,2,\cdots,n$,那么插入 2 需要 1 次比较,插入 3 需要 1 次比较,……,插入 n 也仅需要 1 次比较,总计 $n-1$ 次就够了. 但是,对于以相反次序排列的输入数组 $n,n-1,\cdots,2,1$,同样的插入排序算法所需要的比较次数则是 $1+2+\cdots+n-1=n(n-1)/2$ 次比较. 上述两个实例的规模都等于 n,但插入算法所做的比较次数不一样. 为了解决这个问题,通常将算法的时间复杂度分为最坏情况下的时间复杂度与平均情况下的时间复杂度. 所谓**最坏情况下的时间复杂度**代表了该算法求解输入规模为 n 的实例所需要的最长时间 $W(n)$;而**平均情况下的时间复杂度**代表了该算法求解输入规模为 n 的实例所需要的平均时间 $A(n)$.

以检索问题为例,给定按照递增顺序排列的数组 L,其中含有 n 个不同的元素. 现在需要在 L 中检索 x. 如果 x 在 L 中出现,那么输出 x 所在的位置,否则输出 0. 如下设计顺序检索算法:从 L 的第一个元素开始采用顺序与 x 比较的方法,如果第 j 个元素等于 x,那么输出 j,算法结束. 如果直到最后一个元素为止,没有元素等于 x,那么输出 0,算法结束. 对于含有 n 个元素的数组 L,x 可能是第一个元素,算法只要比较 1 次就结束了;x 也可能是最后一个元素或者 x 根本不在数组中出现,那么算法就需要比较 n 次才能结束. 对于规模为 n 的不同输入实例,算法的比较次数是不一样的,最坏的情况是指对于某个输入运算次数达到最多的那种情况. 显然对于顺序检索最坏情况下的时间复杂度函数 $W(n)=n$. 为了计算平均情况的时间复杂度函数,需要给出规模为 n 的所有输入实例的概率分布. 假定实例集合为 S,实例 $I\in S$ 出现的概率是 p_I,算法对于 I 所做的基本运算次数为 t_I,那么平均情况下的时间复杂度为 $\sum_{I\in S}t_Ip_I$. 以顺序检索算法为例,假定 x 在 L 中的概率是 p,并且处在 L 每个位置的概率相等,那么平均情况下的时间复杂度函数是:

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

对于同样的问题可以设计不同的算法,不同算法的时间复杂度函数可能是不一样的. 什么算法在实践中是可以接受的算法? 划定这个界限对处理实际问题有着重要的意义. 这个界限就是考查算法是不是多项式时间的. 以最坏情况下的时间复杂度函数为例,多项式时间的算法是指在最坏情况下的时间复杂度函数在 n 充分大时以 $P(n)$ 为上界的算法,其中,$P(n)$ 是 n 的多项式. 而对于许多实际的算法,比如 Hanoi 塔的递归算法,根本不存在多项式

$P(n)$，使得该算法最坏情况下的时间复杂度函数在 n 充分大时以 $P(n)$ 为上界，这些算法可能是指数时间的算法或者时间复杂度更高的算法. 如果使用指数时间的算法，对于比较大的 n，实践中是根本不能工作的. 根据这个分析，就可以对实际上可求解问题的范围进行更清晰的划分，即多项式时间可解的问题与难解的问题. 关于这个问题将在第 7 章进行详细的讨论.

1.2 算法的伪码描述

我们将使用伪码描述算法. 伪码由带标号的指令构成，但是它不是 C、C++ 、Java 等通常使用的程序设计语言，而是算法步骤的描述. 它包含赋值语句，并具有程序的主要结构，如顺序、分支(if then else)、循环(while，for，repeat until)等，为了叙述的方便，我们也允许使用转向语句(goto). 有时也可以在某些语句后面加上注释. 输出语句由关键字 return 后面跟随着输出变量或函数值等构成. 当在循环体内遇到输出语句时，不管是否满足循环的条件，算法将停止进一步的迭代，立刻进行输出，然后算法停止运行. 下面是伪码的具体表示：

```
赋值语句：←
分支语句：if…then…[else…]
循环语句：while, for, repeat until
转向语句：goto
输出语句：return
调用
注释：//…
```

伪码程序中常常忽略变量或函数的说明，有时也不指明算法所使用的数据结构. 伪码程序也常常忽略程序的实现细节. 忽略这些，是为了更加专注于算法的设计技术与分析方法，即求解问题的思路与方法.

伪码程序中的语句通常带有标号，这是为了在分析算法的工作量时更为方便.

算法调用子过程时，只需要在相应的语句写明该子过程的名字. 有时甚至可以直接使用自然语言来指出使用的函数或者过程. 例如“将数组 $A[1..n]$ 复制到数组 B”，“将数组 A 的元素按照递增的顺序排序”都是合法的语句. 一般来说，所调用的子过程应该是前面已经说明的过程.

下面是一些伪码的例子. 其中，Euclid 算法计算 m 和 n 的最大公约数，顺序检索算法 Search 检查 x 是否在数组 L 中出现. 如果出现，则输出 x 第一次出现的位置；否则输出 0.

算法 1.1 Euclid(m,n)
输入：非负整数 m,n，其中 m 与 n 不全为 0
输出：m 与 n 的最大公约数
1. while $m>0$ do
2. $\quad r\leftarrow n \bmod m$
3. $\quad n\leftarrow m$
4. $\quad m\leftarrow r$
5. return n

Euclid 算法中第 2 行的 mod 是取模运算，n mod m 表示 n 除以 m 所得的余数，一般余数的值应该在 0 到 $m-1$ 之间.

算法 1.2 Search(L,x)
输入：数组 $L[1..n]$，其元素按照从小到大排列，数 x
输出：若 x 在 L 中，输出 x 的位置下标 j；否则输出 0
1. $j\leftarrow 1$;
2. while $j\leqslant n$ and $x>L[j]$ do $j\leftarrow j+1$
3. if $x<L[j]$ or $j>n$ then $j\leftarrow 0$
4. return j

不难看出算法 Search 在行 2 把 x 与 L 中的元素依次进行比较. while 循环的结束条件是 $x\leqslant L[j]$或者 $j>n$. 当 $x=L[j]$时，这个 j 就是 x 在数组中第一次出现的下标；当 $x<L[j]$时或者 $j>n$ 时，x 不在数组 L 中. 因此算法在最坏的情况下所执行的比较次数是 n. 不难看出，算法 Search 比起 1.1 节提到的顺序检索算法已经有了改进. 当 x 不在 L 中时，它利用 L 中元素的有序性减少了比较次数. 如果输入实例具有与 1.1 节顺序检索算法同样的概率分布，请读者给出 Search 算法在平均情况下的时间复杂度函数.

1.3 算法的数学基础

在讨论算法设计与分析技术之前，我们需要介绍一些相关的数学知识，包括函数的渐近的界的定义与性质、算法分析中常用的证明方法、序列求和的技术以及递推方程的求解.

1.3.1 函数的渐近的界

在算法分析中经常用到定义在自然数集合上的函数 f：$\mathbf{N}\rightarrow\mathbf{N}$. 例如二分检索算法最坏情况下的时间复杂度 $O(\log n)$，插入排序算法最坏情况下的时间复杂度为 $O(n^2)$等. 这里的 n 表示输入规模，检索问题中的 n 表示被检索的线性表中的元素个数，排序问题中的 n 表示被排序的数组中的元素个数. $f(n)=O(\log n)$表示函数 $f(n)$在 n 充分大时以 $c\log n$ 为上界，其中 c 为某个正数称为函数 $f(n)$的渐近的上界. 类似地，也可以定义渐近的下界. 如果函数 $f(n)$的渐近的上界与下界相等，都是 $g(n)$，这时也称 $g(n)$是 $f(n)$的**渐近的紧的界**，或者称函数 $f(n)$的阶是$g(n)$. 考查函数渐近的性质，主要基于以下的考虑：首先，当 n 充分大时，不同阶的函数的值差别非常大，例如前面看到过的 2^n 与 n^2 就是这样的函数. 因此，为了考查算法的性能，我们更应该关心在实例规模很大的时候算法所表现的计算效率. 其次，这里渐近的界的概念，也抓住了算法分析中影响时间复杂度函数的关键因素，使得分析过程更为简单和清晰. 很可能算法的时间复杂度函数的表达式 $T(n)$有多个项，比如 $T(n)=n^2+8n-5$，其中含有三项：n^2，$8n$，-5. 在 n 充分大时，与 n^2 比较，后面两项的影响可以忽略不计. 而 $T(n)$的阶就是 n^2. 这种阶的表示恰好反映了影响算法性能的关键因素. 下面给出关于函数渐近的界的定义.

定义 1.1 设 f 和 g 是定义域为自然数集 $\mathbf{N}$ 上的函数.

(1) 若存在正数 c 和 n_0 使得对一切 $n\geqslant n_0$ 有 $0\leqslant f(n)\leqslant cg(n)$成立，则称 $f(n)$的**渐近的上界**是 $g(n)$，记作 $f(n)=O(g(n))$.