计 算 丛 书

原书第3版

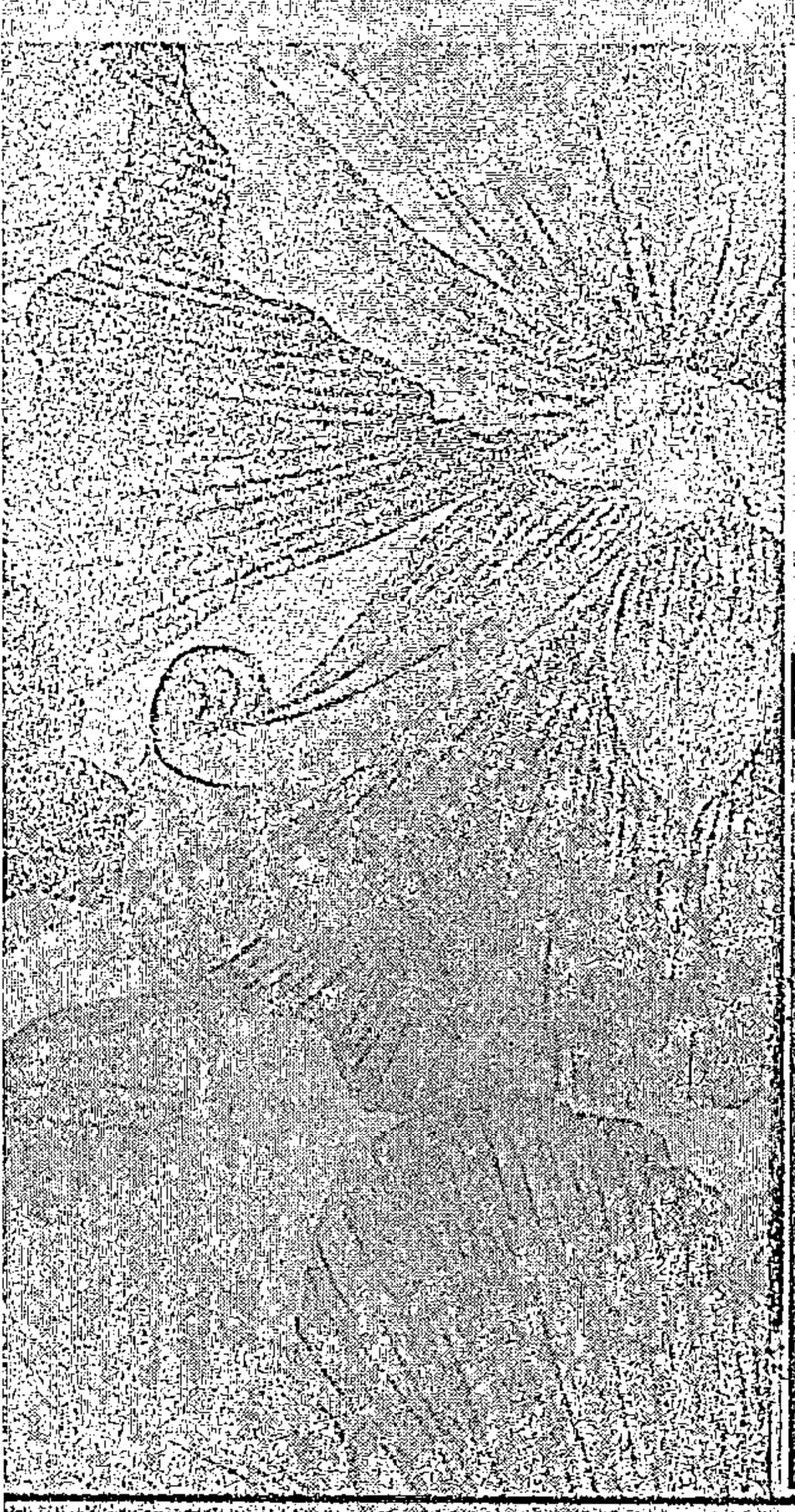
算法导论

(美) Thomas H. Cormen Charles E. Leiserson
Ronald L. Rivest Clifford Stein 著

殷建平 徐 云 王 刚 刘晓光 苏 明 邹恒明 王宏志 译

Introduction to Algorithms

Third Edition



机械工业出版社
China Machine Press

基础知识

这一部分将引导读者开始思考算法的设计和分析问题，简单介绍算法的表达方法、将在本书中用到的一些设计策略，以及算法分析中用到的许多基本思想。本书后面的内容都是建立在这些基础知识之上的。

第 1 章是对算法及其在现代计算系统中地位的一个综述。本章给出了算法的定义和一些算法的例子。此外，本章还说明了算法是一项技术，就像快速的硬件、图形用户界面、面向对象系统和网络一样。

在第 2 章中，我们给出了书中的第一批算法，它们解决的是对 n 个数进行排序的问题。这些算法是用一种伪代码形式给出的，这种伪代码尽管不能直接翻译为任何常规的程序设计语言，但是足够清晰地表达了算法的结构，以便任何一位能力比较强的程序员都能用自己选择的语言将算法实现出来。我们分析的排序算法是插入排序，它采用了一种增量式的做法；另外还分析了归并排序，它采用了一种递归技术，称为“分治法”。尽管这两种算法所需的运行时间都随 n 的值而增长，但增长的速度是不同的。我们在第 2 章分析了这两种算法的运行时间，并给出了一种有用的表示方法来表达这些运行时间。

第 3 章给出了这种表示法的准确定义，称为渐近表示。在第 3 章的一开始，首先定义几种渐近符号，它们主要用于表示算法运行时间的上界和下界。第 3 章余下的部分主要给出了一些数学表示方法。这一部分的作用更多的是为了确保读者所用的记号能与本书的记号体系相匹配，而不是教授新的数学概念。

第 4 章更深入地讨论了第 2 章引入的分治法，给出了更多分治法的例子，包括用于两方阵相乘的 Strassen 方法。第 4 章包含了求解递归式的方法。递归式用于描述递归算法的运行时间。“主方法”是一种功能很强的技术，通常用于解决分治算法中出现的递归式。虽然第 4 章中的相当一部分内容都是在证明主方法的正确性，但是如果跳过这一部分证明内容，也没有什么太大的影响。

第5章介绍概率分析和随机化算法。概率分析一般用于确定一些算法的运行时间，在这些算法中，由于同一规模的不同输入可能有着内在的概率分布，因而在这些不同输入之下，算法的运行时间可能有所不同。在有些情况下，我们假定算法的输入服从某种已知的概率分布，于是，算法的运行时间就是在所有可能的输入之下，运行时间的平均值。在其他情况下，概率分布不是来自于输入，而是来自于算法执行过程中所做出的随机选择。如果一个算法的行为不仅由其输入决定，还要由一个随机数生成器生成的值来决定，那么它就是一个随机化算法。我们可以利用随机化算法强行使算法的输入服从某种概率分布，从而确保不会有某一输入会始终导致算法的性能变坏；或者，对于那些允许产生不正确结果的算法，甚至能够将其错误率限制在某个范围之内。

附录A～D包含了一些数学知识，它们对读者阅读本书可能会有所帮助。在阅读本书之前，读者很有可能已经知道了附录中给出的大部分知识(我们采用的某些符号约定与读者过去见过的可能会有所不同)，因而可以将附录视为参考材料。另外，你很可能从未见过第一部分中给出的内容。第一部分中的所有各章和附录都是以一种入门指南的风格来编写的。

第1章

Introduction to Algorithms, Third Edition

算法在计算中的作用

什么是算法？为什么算法值得研究？相对于计算机中使用的其他技术来说算法的作用是什么？本章我们将回答这些问题。

1.1 算法

非形式地说，**算法**(algorithm)就是任何良定义的计算过程，该过程取某个值或值的集合作为输入并产生某个值或值的集合作为输出。这样算法就是把输入转换成输出的计算步骤的一个序列。

我们也可以把算法看成是用于求解良说明的**计算问题**的工具。一般来说，问题陈述说明了期望的输入/输出关系。算法则描述一个特定的计算过程来实现该输入/输出关系。

例如，我们可能需要把一个数列排成非递减序。实际上，这个问题经常出现，并且为引入许多标准的设计技术和分析工具提供了足够的理由。下面是我们关于**排序问题**的形式定义。

输入：n 个数的一个序列$\langle a_1, a_2, \cdots, a_n\rangle$。

输出：输入序列的一个排列$\langle a'_1, a'_2, \cdots, a'_n\rangle$，满足 $a'_1 \leqslant a'_2 \leqslant \cdots \leqslant a'_n$。

例如，给定输入序列〈31，41，59，26，41，58〉，排序算法将返回序列〈26，31，41，41，58，59〉作为输出。这样的输入序列称为排序问题的一个**实例**(instance)。一般来说，**问题实例**由计算该问题解所必需的(满足问题陈述中强加的各种约束的)输入组成。

因为许多程序使用排序作为一个中间步，所以排序是计算机科学中的一个基本操作。因此，已有许多好的排序算法供我们任意使用。对于给定应用，哪个算法最好依赖于以下因素：将被排序的项数、这些项已被稍微排序的程度、关于项值的可能限制、计算机的体系结构，以及将使用的存储设备的种类(主存、磁盘或者磁带)。

若对每个输入实例算法都以正确的输出停机，则称该算法是**正确的**，并称正确的算法**解决了**给定的计算问题。不正确的算法对某些输入实例可能根本不停机，也可能以不正确的回答停机。与人们期望的相反，不正确的算法只要其错误率可控有时可能是有用的。在第 31 章，当我们研究求大素数算法时，将看到一个具有可控错误率的算法例子。但是通常我们只关心正确的算法。

算法可以用英语说明，也可以说明成计算机程序，甚至说明成硬件设计。唯一的要求是这个说明必须精确描述所要遵循的计算过程。

算法解决哪种问题

排序绝不是已开发算法的唯一计算问题(当看到本书的厚度时，你可能觉得算法也同样多)。算法的实际应用无处不在，包括以下例子：

- 人类基因工程已经取得重大进展，其目标是识别人类 DNA 中的所有 10 万个基因，确定构成人类 DNA 的 30 亿个化学基对的序列，在数据库中存储这类信息并为数据分析开发工具。这些工作都需要复杂的算法。虽然对涉及的各种问题的求解超出了本书的范围，但是求解这些生物问题的许多方法采用了本书多章内容的思想，从而使得科学家能够有效地使用资源以完成任务。因为可以从实验技术中提取更多的信息，所以既能节省人和机器的时间又能节省金钱。
- 互联网使得全世界的人都能快速地访问与检索大量信息。借助于一些聪明的算法，互联网上的网站能够管理和处理这些海量数据。必须使用算法的问题示例包括为数据传输寻找好的路由(求解这些问题的技术在第 24 章给出)，使用一个搜索引擎来快速地找到特定

信息所在的网页(有关技术在第 11 章和第 32 章中)。

- 电子商务使得货物与服务能够以电子方式洽谈与交换，并且它依赖于像信用卡号、密码和银行结单这类个人信息的保密性。电子商务中使用的核心技术包括(第 31 章中包含的)公钥密码与数字签名，它们以数值算法和数论为基础。
- 制造业和其他商务企业常常需要按最有益的方式来分配稀有资源。一家石油公司也许希望知道在什么地方设置其油井，以便最大化其预期的利润。一位政治候选人也许想确定在什么地方花钱购买竞选广告，以便最大化赢得竞选的机会。一家航空公司也许希望按尽可能最廉价的方式把乘务员分配到班机上，以确保每个航班被覆盖并且满足政府有关乘务员调度的法规。一个互联网服务提供商也许希望确定在什么地方放置附加的资源，以便更有效地服务其顾客。所有这些都是可以用线性规划来求解的问题的例子，我们将在第 29 章学习这种技术。

虽然这些例子的一些细节已超出本书的范围，但是我们确实说明了一些适用于这些问题和问题领域的基本技术。我们还说明如何求解许多具体问题，包括以下问题：

- 给定一张交通图，上面标记了每对相邻十字路口之间的距离，我们希望确定从一个十字路口到另一个十字路口的最短道路。即使不允许穿过自身的道路，可能路线的数量也会很大。在所有可能路线中，我们如何选择哪一条是最短的？这里首先把交通图(它本身就是实际道路的一个模型)建模为一个图(第六部分和附录 B 将涉及这个概念)，然后寻找图中从一个顶点到另一个顶点的最短路径。第 24 章将介绍如何有效地求解这个问题。
- 给定两个有序的符号序列 $X=\langle x_1, x_2, \cdots, x_m\rangle$和 $Y=\langle y_1, y_2, \cdots, y_n\rangle$，求出 X 和 Y 的最长公共子序列。X 的子序列就是去掉一些元素(可能是所有，也可能一个没有)后的 X。例如，$\langle A, B, C, D, E, F, G\rangle$的一个子序列是$\langle B, C, E, G\rangle$。$X$ 和 Y 的最长公共子序列的长度度量了这两个序列的相似程序。例如，若两个序列是 DNA 链中的基对，则当它们具有长的公共子序列时我们认为它们是相似的。若 X 有 m 个符号且 Y 有 n 个符号，则 X 和 Y 分别有 2^m 和 2^n 个可能的子序列。除非 m 和 n 很小，否则选择 X 和 Y 的所有可能子序列做匹配将花费使人望而却步多的时间。第 15 章将介绍如何使用一种称为动态规划的一般技术来有效地求解这个问题。

7

- 给定一个依据部件库的机械设计，其中每个部件可能包含其他部件的实例，我们需要依次列出这些部件，以使每个部件出现在使用它的任何部件之前。若该设计由 n 个部件组成，则存在 $n!$ 种可能的顺序，其中 $n!$ 表示阶乘函数。因为阶乘函数甚至比指数函数增长还快，(除非我们只有几个部件，否则)先生成每种可能的顺序再验证按该顺序每个部件出现在使用它的部件之前，是不可行的。这个问题是拓扑排序的一个实例，第 22 章将介绍如何有效地求解这个问题。
- 给定平面上的 n 个点，我们希望寻找这些点的凸壳。凸壳就是包含这些点的最小的凸多边形。直观上，我们可以把每个点看成由从一块木板钉出的一颗钉子来表示。凸壳则由一根拉紧的环绕所有钉子的橡皮筋来表示。如果橡皮筋因绕过某颗钉子而转弯，那么这颗钉子就是凸壳的一个顶点(例子参见图 33-6)。n 个点的 2^n 个子集中的任何一个都可能是凸壳的顶点集。仅知道哪些点是凸壳的顶点还很不够，因为我们还必须知道它们出现的顺序。所以为求凸壳的顶点，存在许多选择。第 33 章将给出两种用于求凸壳的好方法。

虽然这些问题的列表还远未穷尽(也许你已经再次从本书的重量推测到这一点)，但是它们却展示了许多有趣的算法问题所共有的两个特征：

1. 存在许多候选解，但绝大多数候选解都没有解决手头的问题。寻找一个真正的解或一个最好的解可能是一个很大的挑战。

2. 存在实际应用。在上面所列的问题中，最短路径问题提供了最易懂的例子。一家运输公司(如公路运输或铁路运输公司)对如何在公路或铁路网中找出最短路径，有着经济方面的利益，因为采用的路径越短，其人力和燃料的开销就越低。互联网上的一个路由结点为了快速地发送一条消息可能需要寻找通过网络的最短路径。希望从纽约开车去波士顿的人可能想从一个恰当的网站寻找开车方向，或者开车时她可能使用其 GPS。

8

算法解决的每个问题并不都有一个容易识别的候选解集。例如，假设给定一组表示信号样本的数值，我们想计算这些样本的离散傅里叶变换。离散傅里叶变换把时域转变为频域，产生一组数值系数，使得我们能够判定被采样信号中各种频率的强度。除了处于信号处理的中心之外，离散傅里叶变换还应用于数据压缩和大多项式与整数相乘。第 30 章为该问题给出了一个有效的算法——快速傅里叶变换(通常称为 FFT)，并且这章还概述了计算 FFT 的硬件电路的设计。

数据结构

本书也包含几种数据结构。**数据结构**是一种存储和组织数据的方式，旨在便于访问和修改。没有一种单一的数据结构对所有用途均有效，所以重要的是知道几种数据结构的优势和局限。

技术

虽然可以把本书当做一本有关算法的“菜谱”来使用，但是也许在某一天你会遇到一个问题，一时无法很快找到一个已有的算法来解决它(例如本书中的许多练习和思考题就是这样的情况)。本书将教你一些算法设计与分析的技术，以便你能自行设计算法、证明其正确性和理解其效率。不同的章介绍算法问题求解的不同方面。有些章处理特定的问题，例如，第 9 章的求中位数和顺序统计量，第 23 章的计算最小生成树，第 26 章的确定网络中的最大流。其他章介绍一些技术，例如第 4 章的分治策略，第 15 章的动态规划，第 17 章的摊还分析。

难题

本书大部分讨论有效算法。我们关于效率的一般量度是速度，即一个算法花多长时间产生结果。然而有一些问题，目前还不知道有效的解法。第 34 章研究这些问题的一个有趣的子集，其中的问题被称为 NP 完全的。

为什么 NP 完全问题有趣呢？第一，虽然迄今为止不曾找到对一个 NP 完全问题的有效算法，但是也没有人能证明 NP 完全问题确实不存在有效算法。换句话说，对于 NP 完全问题，是否存在有效算法是未知的。第二，NP 完全问题集具有一个非凡的性质：如果任何一个 NP 完全问题存在有效算法，那么所有 NP 完全问题都存在有效算法。NP 完全问题之间的这种关系使得有效解的缺乏更加诱人。第三，有几个 NP 完全问题类似于(但又不完全同于)一些有着已知有效算法的问题。计算机科学家迷恋于如何通过对问题陈述的一个小小的改变来很大地改变其已知最佳算法的效率。

9

你应该了解 NP 完全问题，因为有些 NP 完全问题会时不时地在实际应用中冒出来。如果要求你找出某一 NP 完全问题的有效算法，那么你可能花费许多时间在毫无结果的探寻中。如果你能证明这个问题是 NP 完全的，那么你可以把时间花在开发一个有效的算法，该算法给出一个好的解，但不一定是最好的可能解。

作为一个具体的例子，考虑一家具有一个中心仓库的投递公司。每天在中心仓库为每辆投递车装货并发送出去，以将货物投递到几个地址。每天结束时每辆货车必须最终回到仓库，以便准备好为第二天装货。为了减少成本，公司希望选择投递站的一个序，按此序产生每辆货车行驶的最短总距离。这个问题就是著名的“旅行商问题”，并且它是 NP 完全的。它没有已知的有效算法。然而，在某些假设条件下，我们知道一些有效算法，它们给出一个离最小可能解不太远的总

距离。第 35 章将讨论这样的“近似算法”。

并行性

我们或许可以指望处理器时钟速度能以某个持续的比率增加多年。然而物理的限制对不断提高的时钟速度给出了一个基本的路障：因为功率密度随时钟速度超线性地增加，一旦时钟速度变得足够快，芯片将有熔化的危险。所以，为了每秒执行更多计算，芯片被设计成包含不止一个而是几个处理“核”。我们可以把这些多核计算机比拟为在单一芯片上的几台顺序计算机；换句话说，它们是一类“并行计算机”。为了从多核计算机获得最佳的性能，设计算法时必须考虑并行性。第 27 章给出了充分利用多核的“多线程”算法的一个模型。从理论的角度来看，该模型具有

10 一些优点，它形成了几个成功的计算机程序的基础，包括一个国际象棋博弈程序。

练习

1.1-1 给出现实生活中需要排序的一个例子或者现实生活中需要计算凸壳的一个例子。

1.1-2 除速度外，在真实环境中还可能使用哪些其他有关效率的量度？

1.1-3 选择一种你以前已知的数据结构，并讨论其优势和局限。

1.1-4 前面给出的最短路径与旅行商问题有哪些相似之处？又有哪些不同？

1.1-5 提供一个现实生活的问题，其中只有最佳解才行。然后提供一个问题，其中近似最佳的一个解也足够好。

1.2 作为一种技术的算法

假设计算机是无限快的并且计算机存储器是免费的，你还有什么理由来研究算法吗？即使只是因为你还想证明你的解法会终止并以正确的答案终止，那么回答也是肯定的。

如果计算机无限快，那么用于求解某个问题的任何正确的方法都行。也许你希望你的实现在好的软件工程实践的范围内(例如，你的实现应该具有良好的设计与文档)，但是你最常使用的是最容易实现的方法。

当然，计算机也许是快的，但它们不是无限快。存储器也许是廉价的，但不是免费的。所以计算时间是一种有限资源，存储器中的空间也一样。你应该明智地使用这些资源，在时间或空间

11 方面有效的算法将帮助你这样使用资源。

效率

为求解相同问题而设计的不同算法在效率方面常常具有显著的差别。这些差别可能比由于硬件和软件造成的差别要重要得多。

作为一个例子，第 2 章将介绍两个用于排序的算法。第一个称为**插入排序**，为了排序 n 个项，该算法所花时间大致等于 $c_1 n^2$，其中 c_1 是一个不依赖于 n 的常数。也就是说，该算法所花时间大致与 n^2 成正比。第二个称为**归并排序**，为了排序 n 个项，该算法所花时间大致等于 $c_2 n \lg n$，其中 $\lg n$ 代表 $\log_2 n$ 且 c_2 是另一个不依赖于 n 的常数。与归并排序相比，插入排序通常具有一个较小的常数因子，所以 $c_1 < c_2$。我们将看到就运行时间来说，常数因子可能远没有对输入规模 n 的依赖性重要。把插入排序的运行时间写成 $c_1 n \cdot n$ 并把归并排序的运行时间写成 $c_2 n \cdot \lg n$。这时就运行时间来说，插入排序有一个因子 n 的地方归并排序有一个因子 $\lg n$，后者要小得多。(例如，当 $n=1\,000$ 时，$\lg n$ 大致为 10，当 n 等于 100 万时，$\lg n$ 大致仅为 20。)虽然对于小的输入规模，插入排序通常比归并排序要快，但是一旦输入规模 n 变得足够大，归并排序 $\lg n$ 对 n 的优点将足以补偿常数因子的差别。不管 c_1 比 c_2 小多少，总会存在一个交叉点，超出这个点，归并排序更快。

作为一个具体的例子，我们让运行插入排序的一台较快的计算机(计算机 A)与运行归并排序的一台较慢的计算机(计算机 B)竞争。每台计算机必须排序一个具有 1 000 万个数的数组。(虽然

1 000 万个数似乎很多，但是，如果这些数是 8 字节的整数，那么输入将占用大致 80MB，即使一台便宜的便携式计算机的存储器也能多次装入这么多数。)假设计算机 A 每秒执行百亿条指令(快于写本书时的任何单台串行计算机)，而计算机 B 每秒仅执行 1 000 万条指令，结果计算机 A 就纯计算能力来说比计算机 B 快 1 000 倍。为使差别更具戏剧性，假设世上最巧妙的程序员为计算机 A 用机器语言编码插入排序，并且为了排序 n 个数，结果代码需要 $2n^2$ 条指令。进一步假设仅由一位水平一般的程序员使用某种带有一个低效编译器的高级语言来实现归并排序，结果代码需要 $50n\lg n$ 条指令。为了排序 1 000 万个数，计算机 A 需要

$$\frac{2 \cdot (10^7)^2 \text{ 条指令}}{10^{10} \text{ 条指令 / 秒}} = 20\,000 \text{ 秒(多于 5.5 小时)}$$

而计算机 B 需要

$$\frac{50 \cdot 10^7 \lg 10^7 \text{ 条指令}}{10^7 \text{ 条指令 / 秒}} \approx 1\,163 \text{ 秒(少于 20 分钟)}$$

通过使用一个运行时间增长较慢的算法，即使采用一个较差的编译器，计算机 B 比计算机 A 还快 17 倍！当我们排序 1 亿个数时，归并排序的优势甚至更明显：这时插入排序需要 23 天多，而归并排序不超过 4 小时。一般来说，随着问题规模的增大，归并排序的相对优势也会增大。

算法与其他技术

上面的例子表明我们应该像计算机硬件一样把算法看成是一种**技术**。整个系统的性能不但依赖于选择快速的硬件而且还依赖于选择有效的算法。正如其他计算机技术正在快速推进一样，算法也在快速发展。

你也许想知道相对其他先进的计算机技术(如以下列出的)，算法对于当代计算机是否真的那么重要：

- 先进的计算机体系结构与制造技术
- 易于使用、直观的图形用户界面(GUI)
- 面向对象的系统
- 集成的万维网技术
- 有线与无线网络的快速组网

回答是肯定的。虽然某些应用在应用层不明确需要算法内容(如某些简单的基于万维网的应用)，但是许多应用确实需要算法内容。例如，考虑一种基于万维网的服务，它确定如何从一个位置旅行到另一个位置。其实现依赖于快速的硬件、一个图形用户界面、广域网，还可能依赖于面向对象技术。然而，对某些操作，如寻找路线(可能使用最短路径算法)、描绘地图、插入地址，它还是需要算法。

而且，即使是那些在应用层不需要算法内容的应用也高度依赖于算法。该应用依赖于快速的硬件吗？硬件设计用到算法。该应用依赖于图形用户界面吗？任何图形用户界面的设计都依赖于算法。该应用依赖于网络吗？网络中的路由高度依赖于算法。该应用采用一种不同于机器代码的语言来书写吗？那么它被某个编译器、解释器或汇编器处理过，所有这些都广泛地使用算法。算法是当代计算机中使用的大多数技术的核心。

进一步，随着计算机能力的不断增强，我们使用计算机来求解比以前更大的问题。正如我们在上面对插入排序与归并排序的比较中所看到的，正是在较大问题规模时，算法之间效率的差别才变得特别显著。

是否具有算法知识与技术的坚实基础是区分真正熟练的程序员与初学者的一个特征。使用现代计算技术，如果你对算法懂得不多，你也可以完成一些任务，但是，如果有一个好的算法背景，那么你可以做的事情就多得多。

练习

1.2-1　给出在应用层需要算法内容的应用的一个例子，并讨论涉及的算法的功能。

1.2-2 假设我们正比较插入排序与归并排序在相同机器上的实现。对规模为 n 的输入，插入排序运行 $8n^2$ 步，而归并排序运行 $64n\lg n$ 步。问对哪些 n 值，插入排序优于归并排序？

1.2-3 n 的最小值为何值时，运行时间为 $100n^2$ 的一个算法在相同机器上快于运行时间为 2^n 的另一个算法？

思考题

1-1 （运行时间的比较） 假设求解问题的算法需要 $f(n)$ 毫秒，对下表中的每个函数 $f(n)$ 和时间 t，确定可以在时间 t 内求解的问题的最大规模 n。

14

	1秒钟	1分钟	1小时	1天	1月	1年	1世纪
$\lg n$							
$\sqrt{n}$							
n							
$n\lg n$							
n^2							
n^3							
2^n							
$n!$							

本章注记

关于算法的一般主题存在许多优秀的教科书，包括由以下作者编写的那些：Aho、Hopcroft 和 Ullman[5，6]，Baase 和 Van Gelder[28]，Brassard 和 Bratley[54]，Dasgupta、Papadimitriou 和 Vazirani[82]，Goodrich 和 Tamassia[148]，Hofri[175]，Horowitz、Sahni 和 Rajasekaran [181]，Johnsonbaugh 和 Schaefer[193]，Kingston[205]，Kleinberg 和 Tardos[208]，Knuth[209，210，211]，Kozen[220]，Levitin[235]，Manber[242]，Mehlhorn[249，250，251]，Purdom 和 Brown[287]，Reingold、Nievergelt 和 Deo[293]，Sedgewick[306]，Sedgewick 和 Flajolet[307]，Skiena[318]，以及 Wilf[356]。Bentley[42，43]和 Gonnet[145]讨论了算法设计的一些更实际的方面。算法领域的全面评述也可以在《Handbook of Theoretical Computer Science，Volume A》[342]以及 CRC 出版的《Algorithms and Theory of Computation Handbook》[25]中找到。计算生物学中使用的算法的概述可以在由 Gusfield[156]、Pevzner[275]、Setubal 和 Meidanis[310]以及 Waterman[350]编写的教材中找到。

15