TURING 图灵计算机科学丛书

Foundations Of Algorithms Fifth Edition

算法基础（第5版）

[美] Richard E. Neapolitan 著
贾洪峰 译

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵计算机科学丛书

Foundations Of Algorithms Fifth Edition

算法基础（第5版）

[美] Richard E. Neapolitan 著
贾洪峰 译

人民邮电出版社
北京

图书在版编目（CIP）数据

算法基础 : 第5版 / (美) 那不勒坦(Neapolitan,R.E.) 著 ; 贾洪峰译. -- 北京 : 人民邮电出版社, 2016.3
（图灵计算机科学丛书）
ISBN 978-7-115-41657-5

Ⅰ. ①算… Ⅱ. ①那… ②贾… Ⅲ. ①电子计算机－算法理论 Ⅳ. ①TP301.6

中国版本图书馆CIP数据核字(2016)第023362号

内 容 提 要

本书通过大量示例介绍了算法设计、算法的复杂度分析以及计算复杂度。主要内容有：算法设计与分析、分而治之方法、动态规划方法、贪婪方法、回溯算法、分支定界算法、计算复杂度、难解性和 *NP* 理论、遗传算法和遗传编程、数论算法、并行算法等。此外，本书在每章末尾都提供了大量练习，而且还提供了全面的教辅材料及答案，是教授和学习算法设计与分析的理想教材。

本书适合高等院校学生、程序员及算法分析和设计人员。

◆ 著　[美] Richard E. Neapolitan
译　贾洪峰
责任编辑　岳新欣
执行编辑　李舒扬
责任印制　杨林杰
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷
◆ 开本：880×1230　1/16
印张：25.5
字数：775千字　2016年3月第1版
印数：1－3 000册　2016年3月河北第1次印刷
著作权合同登记号　图字：01-2015-1434号

定价：99.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广字第8052号

版 权 声 明

纪念我的朋友 Jack，他让生活变得充满乐趣！

Richard E. Neapolitan

前　言

本书这一版保留了之前各版赖以成功的特点。和之前的版本一样，这一版仍然使用伪代码，而非真正的C++代码。在呈现复杂算法时，无论使用何种程序设计语言，如果毫无节制地运用该语言的所有细节，都只会模糊学生对算法的理解。此外，伪代码应当能让精通任意高级编程语言的人们都可以看懂，也就是说，它应当尽可能避免使用某种语言特有的细节。1.1 节中讨论了明显与 C++不一致的内容。本书讨论的是算法设计、算法的复杂度分析和计算复杂度（对问题的分析），没有涉及其他类型的分析，比如正确性分析。之所以要编写本书，是因为我找不出一本教科书，既严密准确地讨论了算法的复杂度分析，又能让主流大学（比如东北伊利诺伊大学）计算机科学专业的学生读懂。东北伊利诺伊大学的大多数学生都还没有学习微积分，这就意味着他们不熟悉抽象数学和数学符号。就我所知，现有教科书中都采用了一些数学符号，这对于精通数学的学生来说完全没有问题，但对东北伊利诺伊大学的多数学生来说，就显得有些过于简练而难以理解了。

为使本书更易于理解，我采取了以下做法：

- 假定学生的数学背景知识仅包括大学代数和离散结构；
- 更多地使用日常语言来解释数学概念；
- 在正式证明中给出更多细节；
- 提供大量范例。

本教科书是为高年级本科生或研究生为时一学期的“算法设计与分析”课程编写的，旨在让学生基本掌握编写和分析算法的方法，并向他们传授一些运用标准的算法设计策略来编写算法的必要技能。过去，这些策略包括分而治之、动态规划、贪婪方法、回溯和分支定界。但近年来，遗传算法的应用对计算机科学家来说越来越重要。而学生只有在学习人工智能的相关课程时才可能接触到此类算法。但并没有什么本质性的特征将遗传算法归入人工智能领域。因此，为了更全面地向学生提供当前流行的有用方法，这一版增加了一章内容，讨论遗传算法和遗传编程。

绝大多数复杂度分析都只需要有限数学的知识，所以在大多数讨论中，可以假定读者只有大学代数和离散结构的背景知识。也就是说，对于大多数内容来说，并不需要依靠那些只会在微积分课上学到的概念。没有微积分背景知识的学生通常会对数学符号感到不适应。因此，我会尽量使用日常语言来介绍数学概念（如“大 O”符号），减少数学符号的使用。要在两者之间找到一个最佳平衡点并不那么容易；为使表述清晰，有必要使用一定数量的符号，但使用过多，又会让许多学生感到困惑。根据学生们的反馈情况，我找到了一个合适的平衡点。

这并不是说我不忠实于数学的严格性。我对所有结果都给出了正式证明。但在给出这些证明时，我还给出了较平常而言更多的细节，而且提供了大量范例。学生们在看到具体范例时，通常能够更好地掌握理论概念。因此，数学背景知识不够扎实的学生只要愿意付出足够的努力，应当可以理解这些数学论证，从而更深入地掌握相关内容。此外，书中的确包含了一些需要微积分知识才能理解的内容（比如使用极限来确定阶数和证明一些定理）。但是，学生们没掌握这些内容也能理解书中其余部分。对于需要微积分知识的内容，在目录和正文中的空白处都标有一个◉符号；并不需要更多的数学背景知识，但难度高于书中其他部分的内容，则标注了◈符号。

预备知识

前面已经说过，本书假定学生的数学背景知识仅限于大学代数和有限数学。真正需要的数学知识在附录 A 中复习。关于计算机科学方面的背景知识，假定学生们已经学习了数据结构课程。因此，通常会在数据结构教科书中出现的内容，本书不再给出。

各章内容

本书大部分内容都是根据问题的解决方法而非应用领域来组织的。我感觉这种组织形式可以让算法的设计与分析领域显得更连贯。另外，学生也能更轻松地学到一整套技巧，在遇到新问题时，从中找出也许可行的解决方法。各章内容如下。

- 第 1 章介绍算法设计与分析，其中对阶的概念进行了直观、正式的介绍。
- 第 2 章介绍算法设计的分而治之方法。
- 第 3 章给出动态规划方法，讨论了应当使用动态规划而不是分而治之方法的场景。
- 第 4 章讨论了贪婪方法，并在最后对比了用于解决最优化问题的动态规划和贪婪方法。
- 第 5 章和第 6 章分别介绍回溯算法和分支定界算法。
- 第 7 章从算法分析转到计算复杂度，它是对问题本身的分析。我们通过分析“排序问题”来介绍计算复杂度。之所以选择这一问题，一是因为它重要，二是因为排序算法非常多，还有最重要的一点，是因为一些性能非常出色的排序算法几乎可以达到“排序问题”的时间下限（这里所说的下限，是指仅通过比较键进行排序的算法所花费的时间）。在比较了排序算法之后，分析了通过比较键进行排序的问题。这一章最后讨论了基排序，这种排序算法不是通过比较键来实现排序的。
- 第 8 章通过分析“查找问题”进一步讨论了计算复杂度。这一章分析了在列表中查找键的问题，还分析了“选择问题”，也就是在一个列表中找出第 k 小的键的问题。
- 第 9 章专门讨论难解性（intractability）和 *NP* 理论。为使本书既通俗易懂又准确严谨，对这一内容的讨论要比一般算法教科书中更为全面。首先明确划分了三类问题之间的界限：一类是已经为其找到多项式时间算法的问题，一类是已经证明为难解的问题，还有一类是尚未证明是难解的，但还从来没有为其找到多项式时间算法的问题。接下来讨论了 *P* 问题、*NP* 问题、*NP* 完全问题和 *NP* 等价问题。我发现，如果学生没有真正明白这几类问题之间的关系，经常会糊里糊涂。本章最后讨论了近似算法。
- 第 10 章介绍遗传算法和遗传编程，其中提供了有理论、实践两方面的应用，比如金融贸易算法。
- 第 11 章介绍数论算法，包括欧氏算法和用于判定一个数字是否为质数的新的多项式时间算法。
- 第 12 章简要介绍并行算法，包括并行体系结构和 PRAM 模型。
- 附录 A 复习了理解本书所需要的数学知识。
- 附录 B 介绍了求解递归方程的方法。在第 2 章分析分而治之算法时用到了附录 B 中的结果。
- 附录 C 给出了一种不交集数据结构，在实现第 4 章的两个算法时用到了这一结构。

教授方法

为激发学生的兴趣，每一章都以一个与该章内容有关的故事开头。此外，还使用了许多示例，并在各章最后给出大量习题，按节进行分组。在各节的习题之后是补充习题，其挑战性通常要更大一些。

为表明一个问题可以有多种解决方法，有些问题采用了多种解法。例如，在解决“旅行推销员问题”时使用了动态规划、分支定界和近似算法。在解决“0-1 背包问题”时使用了动态规划、回溯和分支定界算法。为

使内容更加完整，我给出了一个横跨多章的题目，其中有一个名为 Nancy 的推销员，她要找出一条最佳推销旅行路线。

课程概述

如前所述，本书适用于高年级本科生或研究生算法课程。

在为时一学期的课程中，建议依次讲授以下内容。

第 1 章：全部

附录 B：B.1、B.3 节

第 2 章：2.1~2.5、2.8 节

第 3 章：3.1~3.4、3.6 节

第 4 章：4.1、4.2、4.4 节

第 5 章：5.1、5.2、5.4、5.6、5.7 节

第 6 章：6.1、6.2 节

第 7 章：7.1~7.5、7.7、7.8.1、7.8.2、7.9 节

第 8 章：8.1.1、8.5.1、8.5.2 节

第 9 章：9.1~9.4 节

第 10 章：10.1~10.3.2 节

第 2~6 章均包含若干节，其中每一节都利用该章给出的设计方法解决一个问题。我选择了最感兴趣的几节，你可以选择其中任意一节。

你可能来不及讲授第 11 章和第 12 章。但是，在学习前 10 章之后，学生们应当可以很轻松地理解第 12 章中的内容。数学知识扎实的学生，比如在学习了微积分之后，应当可以自学第 11 章的内容。

教师资源

具备教师资格的读者可以申请教师手册、PowerPoint 演示文稿和完整的答案手册。Jones & Bartlett Learning 保留对所有申请进行评估的权利。

致谢

我要感谢所有阅读了本书草稿并提出许多有用建议的人们。特别感谢我的同事 William Bultman、Jack Hade、Mary Kenevan、Jim Kenevan、Stuart Kurtz、Don La Budde 和 Miguel Vian，他们欣然全面地审阅了相关内容。还要感谢学术与专业同行审查员们，他们富有深刻见解的批评意见大大提升了本书的质量。他们许多人所做的详尽工作都远远超出了我们的预期。他们是：泽维尔大学的 David D. Berry，瓦尔德斯塔州立大学的 David W. Boyd，圣何塞州立大学的 Vladimir Drobot，加利福尼亚大学欧文分校的 Dan Hirschberg，东北伊利诺伊大学的 Xia Jiang，西弗吉尼亚大学的 Raghu Karinthi，东北伊利诺伊大学的 Peter Kimmel，东北伊利诺伊大学的 C. Donald La Budde，印第安纳大学–普渡大学韦恩堡分校的 Y. Daniel Liang，德雷塞尔大学的 David Magagnosc，南伊利诺伊大学卡本代尔分校的 Robert J. McGlinn，密西西比大学的 Laurie C. Murphy，梅西山学院的 Paul D. Phillips，加州州立理工大学波莫纳分校的 H. Norton Riley，西北大学的 Majid Sarrafzadeh，弗吉尼亚理工学院暨州立大学的 Cliff Shaffer，得克萨斯理工大学的 Nancy Van Cleave，纽约州立大学宾汉姆顿分校的 William L. Ziegler。最后，我还要感谢 Taylor 和 Francis，特别是 Randi Cohen，他们允许将我在 2012 年出版的 *Contemporary*

Artificial Intelligence 一书中的内容放在本书第 10 章中。

勘误

在一本如此篇幅的书中，肯定会存在一些错误。如果你发现了任何错误，或有任何改进建议，我非常愿意收到你的来信。请将你的意见发给 Rich Neapolitan。电子信箱：RE-Neapolitan@neiu.edu。谢谢！

R. N.

目　录

第 1 章

算法：效率、分析和阶

本书讨论用计算机解决问题的方法。这里所说的“方法”并不是指一种程序设计风格或者一种程序设计语言，而是用于解决问题的方法或方法学。例如，假设 Barney Beagle 希望在电话簿中找到名字“Collie, Colleen”。一种方法是从第一个名字开始，依次查看每个名字，直到找出“Collie, Colleen”为止。但是，没有人会这样来找一个名字。电话簿中的名字都是有序的，所以 Barney 会利用这一事实，将电话簿翻到他认为 C 字头名字所在的位置。如果他翻过了，可以往回翻一些。他不停地前后翻动，直到找出“Collie, Colleen”所在的页面。你可能看出来了，第二种方法就是一种经过修改的二分查找，而第一种方法是一种顺序查找。1.2 节会进一步讨论这两种查找方法。这里要说的是，这一问题有两种不同的解决方法，而这两种方法与程序设计语言或风格没有任何关系。计算机程序只不过是实现这些方法的一种途径。

第 2 章至第 6 章讨论各种解决问题的方法，并运用这些方法解决各种问题。将某一方法应用于某一问题，会得到解决该问题的一个逐步过程。这个逐步过程称为该问题的**算法**（algorithm）。研究这些方法及其应用的目的就是掌握许多方法，以便在遇到新问题时，可以从中找出解决该问题的方法。后面经常会看到，可以使用几种不同的方法来解决同一问题，但其中一种方法得出的算法要远快于其他方法得出的算法。在电话簿中查找姓名时，经过修改的二分查找法当然要快于顺序查找法。因此，我们不仅要判断某个问题能否用某一给定方法解决，还要分析所得到的算法在时间和存储方面的效率如何。当在计算机上实现算法时，时间是指 CPU 周期，存储是指内存。你可能会感到奇怪，计算机变得越来越快，内存变得越来越便宜，为什么还需要考虑效率。本章将讨论一些基本概念，它们是理解本书内容所必需的。在此过程中我会向你说明，为什么无论计算机变得多快、内存变得多便宜，还是要考虑效率。

1.1 算法

到目前为止，我们已经提到了“问题”“答案”和“算法”等词。我们大多数人都能很好地理解这些词语的含义。但为了打下一个坚实的基础，还是再具体定义一下这些术语。

计算机程序由完成特定任务（比如排序）的各个模块组成，计算机是可以理解这些模块的。本书关注的不是整个程序的设计，而是这些完成特定任务的各个模块的设计。这些特定的任务称为“问题”。明确地说，**问题**（problem）就是要为其寻求答案的疑问。下面给出问题的一些例子。

例 1.1 将一个包含 n 个数字的列表 S 按非递减顺序排序。其答案就是这些数字排序后的结果。

这里所说的**列表**（list）是指一组按特定顺序排列的项目。例如，

$$S = [10, 7, 11, 5, 13, 8]$$

是一个包含六个数字的列表，其中第一个数字为 10，第二个为 7，等等。例 1.1 中说到要按“非递减顺序”对这个列表排序，而不是按升序排列，这样就存在同一数字在列表中出现多次的可能性。

例 1.2 判断数字 x 是否在一个包含 n 个数字的列表 S 中。如果 x 在 S 中，则答案为“是”，否则为“否”。

一个问题可能会在问题描述中包含一些未指定取值的变量。这些变量称为问题的**参数**（parameter）。例 1.1 中有两个参数：S（列表）和 n（S 中项目的个数）。例 1.2 中有三个参数：S、n 和数字 x。在这两个例子中，并不需要让 n 成为一个参数，因为它的值由 S 唯一确定。但是，让 n 成为参数有利于问题的描述。

因为问题中包含参数，所以它代表着一类问题，每对参数进行一次赋值，就得到其中一个问题。对参数的

每次特定赋值就称为该问题的一个**实例**（instance）。一个问题实例的**答案**（solution）就是该实例中所提问题的回答。

例 1.3　例 1.1 所提问题的一个实例为：

$$S = [10, 7, 11, 5, 13, 8],\ n = 6$$

这一实例的答案为[5, 7, 8, 10, 11, 13]。

例 1.4　例 1.2 所提问题的一个实例为：

$$S = [10, 7, 11, 5, 13, 8],\ n = 6,\ x = 5$$

这一实例的答案为："是，x 在 S 中。"

对于例 1.3 中的实例，只需审视 S，用一种无法具体描述的认知步骤在大脑中生成有序序列，就能给出该实例的答案。之所以能这样做，是因为 S 非常小，在人类的意识水平，大脑可以快速扫描 S，几乎马上就可以给出答案（因此，我们无法描述大脑用于获得此答案的具体步骤）。但是，如果这个例子中的 n 值为 1000，那人们就无法使用这一方法，当然也就不可能将这样一种数字排序方法转换为计算机程序。为了给出能够解答某一问题所有实例的计算机程序，我们必须给出一个通用的逐步过程，用于给出每个实例的答案。这一逐步过程称为**算法**。我们说算法**解决**了问题。

例 1.5　例 1.2 所示问题的一种算法。从 S 中的第一项开始，依次将 x 与 S 中的每一项对比，直到找出 x 或 S 中的项目耗尽为止。如果找到了 x，则答案为"是"；如果没有找到 x，则答案为"否"。

任何算法都可以像例 1.5 中那样用日常语言来描述。但是，以这种方式来书写算法有两个缺点。第一，这样很难书写复杂算法，即使勉强写出，人们也很难理解。第二，根据算法的日常语言描述，无法清晰地知道如何生成对该算法的计算机语言描述。

因为 C++是当前学生比较熟悉的一门语言，所以我们使用一种类似于 C++的伪代码来书写算法。只要拥有类 Algol 命令式语言（比如 C、Pascal 或 Java）的编程经验，应当不难看懂这种伪代码。

为演示该伪代码，我们以一种算法以例，该算法可以解决例 1.2 所示问题的一般形式。为简单起见，例 1.1 和例 1.2 的表述都是针对数字的。但是，一般来说，我们希望对来自任意有序集合中的项目进行查找和排序。通常，每一项都唯一地确定了一条记录，因此，这些项目通常称为**键**（key）。例如，一条记录可能由某个人的个人信息组成，以这个人的社会保障号作为键。为这些项目定义一种数据类型 keytype，在编写查找和排序算法时就使用这一数据类型。这就是说，这些项目来自任意有序集合。

下面的算法用数组表示列表 S，它返回的不只是"是"或"否"，如果 x 在 S 中，还会返回 x 在数组中的位置；如果不在，则返回 0。这一查找算法并不要求这些项目来自有序集合，但我们仍然使用标准数据类型 keytype。

算法 1.1　顺序查找

问题：键 x 是否存在于拥有 n 个键的数组 S 中？

输入（参数）：正整数 n；键的数组 S，其索引范围为 1 至 n；键 x。

输出：location，x 在 S 中的位置（若 x 不在 S 中，则为 0）。

```
void seqsearch (int n,
                const keytype S[ ],
                keytype x,
                index& location)
{
    location = 1;
    while (location <= n && S[location] != x)
       location++;
    if (location > n)
       location = 0;
}
```

这段伪代码与 C++非常类似，但不完全等同。一个很明显的不同就是对数组的使用。C++中的数组索引只

能是从 0 开始的整数。很多时候，使用以其他整数范围为索引的数组可以更清楚地解释算法；而有时，使用非整数索引可以将算法解释得最为清楚。因此，在伪代码中，允许数组的索引是任意集合。在指明算法的“输入”和“输出”时，总会规定索引的范围。例如，算法 1.1 中指明 S 的索引范围为 1 至 n。人们在计算列表中的项目数时，习惯于从 1 开始，所以为列表使用这一索引范围是很不错的选择。当然，这个算法可以直接用 C++实现，声明如下：

```
keytype S[n + 1];
```

跳过 $S[0]$位置不用。下文不再讨论以任何程序设计语言实现算法。我们的目的只是清楚明了地给出算法，从而使其便于理解和分析。

关于伪代码中的数组，还有另外两点明显不同于 C++的地方。第一，允许采用变长两维数组作为例程的参数，比如算法 1.4。第二，我们声明了局部变长数组。例如，如果 n 是过程 example 的一个参数，而且需要一个索引范围为 2 到 n 的局部数组，可以声明如下：

```
void example (int n)
{
   keytype S[2..n];
   ⋮
}
```

符号 $S[2..n]$表示以 2 到 n 为索引范围的数组，这完全是伪代码；也就是说，它不是 C++语言的组成部分。

如果与使用实际的 C++指令相比，使用数学表达式或日常语言可以更简洁、更清楚地描述算法步骤，我们就会这么做。例如，假定仅当变量 x 介于取值 low 和 high 之间时，某些指令才会执行。我们会写为：

```
if (low ≤ x ≤ high) {
      ⋮
}
```

而不是：

```
if (low <= x && x <= high){
      ⋮
}
```

假定我们希望变量 x 取变量 y 的值，y 取 x 的值。我们将写为：

```
交换 x 和 y;
```

而不是写为：

```
temp = x;
x = y;
y = temp;
```

除了数据类型 keytype 之外，我们还经常使用以下数据类型，它们也不是预定义的 C++数据类型。

数据类型	含　义
index	用作索引的整数变量
number	一个可以定义为整数（int）或实数（float）的变量
bool	一个可以取“true”或“flase”的变量

有时数字可以取任意实数，有时只能取整数。如果这一点对算法来说并不重要，我们将使用数据类型 number。

我们有时会使用下面的非标准控制结构：

```
repeat ( n times) {
  ⋮
}
```

其含义是将代码重复 n 次。在C++中，需要另外引入一个控制变量，并编写 for 循环。只有当真正需要引用循环中的控制变量时，我们才会使用 for 循环。

当算法的名字看起来与其返回值很吻合时，我们就将算法写为*函数*（function）。否则，就将其写为*进程*（procedure，C++中的 void 函数），并使用*引用参数*（reference parameter，也就是按地址传递的参数）来返回值。如果参数不是数组，在声明它时，会在数据类型名的末尾添加一个&符号。它在这里的意思是：这个参数包含算法的一个返回值。因为数组在C++中是自动按引用传递的，而且在传递数组时，C++中也没有使用&符号，所以我们没有使用&来表示数组中包含算法的返回值。相反，由于C++中使用保留字 const 来防止对所传递数组进行修改，所以我们使用 const 表示数组中不包含算法的返回值。

一般情况下，我们尽量避免使用C++特有的功能，以便那些只了解其他高级语言的人也能读懂伪代码。但是，我们的确会编写一些类似于 i++之类的指令，其含义是指将 i 递增1。

如果不了解C++，你可能会觉得逻辑运算符和某些关系运算符有些陌生。这些符号列出如下。

运算符	C++符号
and	&&
or	||
not	!

比　较	C++代码
$x = y$	(x == y)
$x \neq y$	(x != y)
$(x \leq y)$	(x <= y)
$x \geq y$	(x >= y)

下面给出更多的示例算法。第一个例子演示函数的使用。过程的例程名之前带有关键字 void，而函数的例程名之前带有函数返回的数据类型。这个值在函数中通过 return 语句返回。

算法 1.2　数组成员求和

问题：将包含 n 个数字的数组 S 中的所有成员加在一起。

输入：正整数 n；数字数组 S，其索引范围为1至 n。

输出：sum，S 中的数字之和。

```
number sum (int n, const number S[ ])
{
   index  i;
   number result;

   result = 0;
   for (i = 1; i <= n; i++)
      result = result + S[i];
   return result;
}
```

本书将讨论许多排序算法。下面是其中很简单的一个。

算法 1.3　交换排序

问题：按非递减顺序对 n 个键排序。

输入：正整数 n；键的数组 S，其索引范围为1至 n。

输出：数组 S，其中的键按非递减顺序排列。

```
void exchangesort (int n; keytype S[])
{
    index i, j;
    for (i=1;i<=n;i++)
       for (j=i+1;j<=n;j++)
```

```
        if (S[j] < S[i])
          交换 S[i]和 S[j];
}
```

指令

```
交换 S[i]和 S[j];
```

的含义是，$S[i]$将取 $S[j]$的值，$S[j]$将取 $S[i]$的值。这条命令一点都不像 C++指令；如果不使用 C++指令的细节可以将事情描述得更简单，我们就一定会这么做。“交换排序”是将第 i 个位置的数字与第 i+1 个位置到第 n 个位置的数字进行比较。只要发现给定位置的数字小于第 i 个位置的数字，就交换这两个数字。这样，在完成第一遍 for-i 循环后，最小的数字将放在第一位，在第二遍循环后，第二小的数字将放在第二位，以此类推。

下面的算法执行矩阵乘法。回想一下，如果有两个 2×2 矩阵：

$$A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{和} B=\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

则它们的乘积 $C=A\times B$ 为：

$$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}$$

例如，

$$\begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix}\times\begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}=\begin{bmatrix} 2\times5+3\times6 & 2\times7+3\times8 \\ 4\times5+1\times6 & 4\times7+1\times8 \end{bmatrix}=\begin{bmatrix} 28 & 38 \\ 26 & 36 \end{bmatrix}$$

一般情况下，如果有两个 $n\times n$ 矩阵 A 和 B，则其乘积 C 如下：

$$c_{ij}=\sum_{k=1}^{n}a_{ik}b_{kj} \quad (1\leqslant i,j\leqslant n)$$

由这一定义，可以直接得出矩阵乘法的以下算法。

算法 1.4 矩阵乘法

问题：计算两个 $n\times n$ 矩阵的乘积。

输入：一个正整数 n；两维数字数组 A 和 B，每个数组的行和列都以 1 至 n 为索引范围。

输出：一个二维数字数组 C，包含了 A 和 B 的乘积，它的行和列都以 1 至 n 为索引范围。

```
void matrixmult (int n,
                 const number A[][],
                 const number B[][],
                       number C[][])
{
    index i, j, k;

    for (i = 1; i<=n; i++)
       for (j=1; j≤n; j++){
          C[i][j] = 0;
          for (k=1; k<=n; k++)
             C[i][j] = C[i][j]+A[i][k]*B[k][j];
       }
}
```

1.2 开发高效算法的重要性

前面曾经提到，无论计算机变得多么快速，内存变得多么廉价，效率永远都是重要的考虑因素。接下来，我们通过对比同一问题的两种算法来说明其原因。

1.2.1　顺序查找与二分查找的对比

前面曾经提到，在电话簿中查找姓名的方法是一种经过修改的二分查找，它通常要远快于顺序查找。下面将对比这两种方法的算法，以说明二分查找法要快多少。

我们已经编写了完成顺序查找的算法，即算法 1.1。在一个非递减顺序数组中进行二分查找的算法类似于用大拇指前后翻动电话簿。也就是说，假定正在查找 x，算法首先将 x 与数组的中间项进行对比。如果相等，则算法完成。如果 x 小于中间项，则 x 必然在数组的前半部分（如果存在的话），算法将对数组的前半部分重复该查找过程。（也就是说，将 x 与数组前半部分的中间项进行比较，等等。）如果 x 大于数组的中间项，则对数组的后半部分重复查找过程。一直重复此过程，直到找出 x，或者确认 x 不在数组中为止。这一方法的算法如下。

算法 1.5　二分查找

问题：判断 x 是否在一个包含 n 个键的有序数组 S 中。

输入：正整数 n；有序（非递减顺序）键数组 S，其索引范围为 1 至 n；键 x。

输出：location，x 在 S 中的位置（如果 x 不在 S 中，则为 0）。

```
void binsearch (int n,
                const keytype S[],
                keytype x,
                index& location)
{
    index low, high, mid;

    low = 1; hign = n;
    location = 0;
    while (low <= high && location == 0){
       mid = |(low + high)/2|;
       if (x = = S[mid])
          location = mid;
       else if (x < S[mid])
          high = mid - 1;
       else
          low = mid + 1;
    }
}
```

我们来对比顺序查找与二分查找所做的工作。重点确定每种算法执行的比较次数。如果数组 S 包含 32 项，且 x 不在数组中，则算法 1.1（顺序查找）需要将 x 与所有 32 项进行比较后，才能确定 x 不在数组中。一般情况下，对于一个大小为 n 的数组，顺序查找需要完成 n 次比较才能确定 x 不在该数组中。显然，在搜索一个大小为 n 的数组时，这是顺序查找所执行的最大比较次数。也就是说，如果 x 在数组中，则比较次数不大于 n。

下面考虑算法 1.5（二分查找）。每执行一遍 while 循环，会将 x 与 S[mid]比较两次（找到 x 时除外）。在用高效汇编语言实现该算法时，每执行一遍 while 循环，仅将 x 与 S[mid]比较一次，比较结果将确定条件代码，然后根据条件代码的取值执行适当的跳转。这意味着每执行一遍 while 循环，仅将 x 与 S[mid]比较一次。我们假定算法就是以这种方式实现的。在这一假设条件下，图 1-1 表明，对于一个大小为 32 的数组，若 x 大于其中所有项目，该算法将执行 6 次比较。注意，$6 = \lg 32 + 1$。这里的 lg 是指 $\log_2$。在算法分析中经常会遇到 $\log_2$，所以我们为它保留了特殊符号 lg。你应当相信，二分查找最多只需要执行这么多的比较次数。也就是说，如果 x 在数组中，或者 x 小于数组中的所有项目，或者 x 在两个数组项目之间，所执行的比较次数都不会超过 x 大于所有数组项目时的次数。

图 1-1　对于一个大小为 32 的数组，当 x 大于其中所有项目时，二分查找法用来与 x 进行比较的数组项目。这些项目的编号就是它们与 x 的比较顺序