重点大学计算机教材

HZ BOOKS

数据结构与算法

Java语言描述

邓俊辉 编著

清华大学

机械工业出版社
China Machine Press

重点大学计算机教材

数据结构与算法
Java语言描述

邓俊辉 编著
清华大学

机械工业出版社
China Machine Press

本书充分展示了面向对象技术在现代数据结构理论中的应用，普遍采用了抽象、封装及继承等技术。本书既介绍了基本的数据结构，包括栈、队列、向量、列表结构；也介绍了若干高级数据结构，包括优先队列结构、映射和词典结构、查找树结构等。并结合具体问题介绍了算法的应用、实现及其分析方法，涉及的算法包括堆结构的生成及调整算法、Huffman编码树算法、平衡查找树的生成、插入和删除算法，并着重介绍了串匹配的KMP和BM算法。本书还通过遍历算法框架将各种图算法统一起来，并基于遍历算法模板加以实现，在同类教材中独树一帜。

本书图文并茂，循序渐进。书中代码都配有详尽而简洁的注释。书中还结合各部分的具体内容穿插了大量问题，以激发读者的求知欲，培养良好的自学习惯和自学能力。本书适合用作计算机专业本科生教材或参考书。

图书在版编目（CIP）数据

数据结构与算法（Java语言描述）/邓俊辉编著．—北京：机械工业出版社，2006.1
（重点大学计算机教材）
ISBN 7-111-18204-9

Ⅰ.数… Ⅱ.邓… Ⅲ.①数据结构—高等学校—教材②算法分析—高等学校—教材③JAVA语言—程序设计—高等学校—教材 Ⅳ.①TP311.12②TP312

中国版本图书馆CIP数据核字（2005）第157508号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）
策划编辑：温莉芳
责任编辑：刘立卿
北京慧美印刷有限公司印刷·新华书店北京发行所发行
2006年2月第1版第1次印刷
787mm×1092mm 1/16 · 20.00印张
印数：0 001-4 000册
定价：33.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：(010)68326294

前　言

关于计算机教育规范，最有权威、影响最大的莫过于 ACM 与 IEEE-CS 联合制订的 Computing Curricula。比如，Computing Curricula 1991（CC1991）就曾对我国高校的计算机教育产生深刻的影响。正在制订中的 CC2001（后改称 CC2004）认为，随着近年来计算概念的快速发展，计算学科已经发展成为一个内涵繁杂的综合性学科，至少可以划分为计算机工程（CE）、计算机科学（CS）、信息系统（IS）、信息技术（IT）和软件工程（SE）五个领域，而且不同领域的人才所应具备的知识结构与能力侧重也不尽相同。尽管如此，从目前已经完成的部分来看，数据结构与算法在各领域的知识体系中仍占据重要的位置。比如在 CE 分卷（草案）中，Programming Fundamentals 共 39 个核心学时，其中 Algorithms and Problem Solving 占 8 个学时，Data Structures 占 13 个学时；在 CS 分卷（正式稿）中，Programming Fundamentals 共 38 个核心学时，其中 Algorithms and Problem Solving 占 6 个学时，Data Structures 占 14 个学时。

为什么数据结构与算法长期以来一直受到如此重视呢？数据结构和算法是一对孪生兄弟，数据结构研究的问题包括数据在计算机中存储、组织、传递和转换的过程及方法，这些也是构成与支撑算法的基础。我们在编写计算机程序解决应用问题时，最终都要归结并落实到这两个问题上。正因为此，N. Wirth 早在 20 世纪 70 年代就曾指出“程序＝数据结构＋算法”。近年来，随着面向对象技术的广泛应用，从数据结构的定义、分类、组成，到设计、实现与分析的模式和方法都有了长足的发展，现代数据结构更加注重和强调数据结构的整体性、通用性、复用性、简洁性和安全性。

为遵循上述原则，本书选择 Java 作为描述语言，因为相对于其他语言，Java 语言比较完整、彻底地体现了面向对象的设计思想，这也是目前国际上同行们的一个主要取向。不过，关于 Java 语言本身的介绍，本书并未花费多少笔墨，而是直接假定读者已经熟练掌握了该语言。这既能缩减本书篇幅，也符合此领域教学今后的规范。比如，CC2001 的 CS 分卷将 Data Structures and Algorithms 课程编号为 CS103，并明确指出该课程是 Programming Fundamentals（CS101）和 The Object-Oriented Paradigm（CS102）的后续课程。

在内容选取、剪裁与体例结构上，作者力图突破现成的模式。这里并未对各种数据结构面面俱到，而是通过分类和讲解典型结构，力图使读者形成对数据结构的宏观认识；并结合各部分的具体内容，在书中穿插了大量的问题，把更多的思考空间留给读者。

根据内容侧重，本书具体分为 10 章。

第 1 章是全书的基础，重点在于介绍算法与数据结构的关系，以及算法时间、空间复杂度的概念及其度量方法。

第 2～4 章覆盖了基本的数据结构，既有传统的栈与队列，也有更为抽象和通用的向量和列表。面向对象技术在现代数据结构理论中的重要地位在此得到体现，我们利用接口实现数据结构的封装，抽象出位置的概念，在向量和列表的基础上定义并实现序列结构，引入迭代器概念并针对上述结构具体实现。通过结合数组与链表结构的优点，第 4 章自然地导出了树结构的概念、实现及算法。

第 5～7 章介绍了若干高级数据结构。通过在一般性队列中引入优先级概念，导出了优

先队列结构；面向实际问题中的查找需求，导出了映射和词典结构；针对全序集的查找问题，引入了查找树结构，并结合 AVL 树、伸展树和 B-树的实例介绍了平衡查找树结构的原理及其实现。面向对象的思想在这里继续得到贯彻，普遍采用了抽象、封装及继承等技术。

这一部分的侧重点逐渐转向算法，并结合具体问题介绍其应用与实现，包括堆结构的生成及调整算法、Huffman(赫夫曼)编码树算法、散列算法以及二路或多路平衡查找树的生成、插入、删除算法。在这三章中，读者也将对算法复杂度的各种分析方法有所了解。

第 8～10 章的论述重点完全转向算法。第 8 章对此前各章中陆续介绍过的排序算法进行了归纳与分类，着重介绍了归并排序算法与快速排序算法，并给出了此类算法的复杂度下界。结合串结构的应用，第 9 章着重讨论了串匹配问题，从蛮力算法出发，采用不同的启发式加速策略，分别介绍了 KMP 算法及 BM 算法。

关于图结构，第 10 章将重点放在相关算法上，并力图使初学者从各种图算法中梳理出清晰的脉络。为此，这里尝试通过一条主线——遍历——将各种算法串接起来。在这里，面向对象技术的魅力再次得以展现：基于统一的图遍历算法模板，分别实现了深度优先、广度优先和最佳优先三大类遍历算法。实际上，这三类算法本身仍然以模板形式实现，包括边分类、可达分量、连通分量、最短路径以及最小生成树在内的各种具体算法，都进而基于这三个模板分别得到了实现。

书中涉及的所有代码，都符合 J2SDK-1.4.1 规范，并构成一个名为 dsa(Data Structures & Algorithms)的包；所有类之间的扩展、继承关系，可以参见书后的“DSA 类关系图”。

要获取本书中的代码及相关教学资料，请访问：http://vis.cs.tsinghua.edu.cn/～deng/dsaj.htm。

本书从筹划、撰写、审订到定稿，其间跨越三个年头，在此漫长的过程中，我的父母和妻子给了我巨大的支持，就这个意义而言，他们也是本书的完成者。年幼的女儿总是以她独有的方式，不时将我从写作和调试的苦海中解救出来，她让我体会了禁用 PowerOff 按钮的妙用（尽管除了拔掉接头外我还没有找到更好的办法，使得在她恶作剧地按下 Reset 按钮后依然能够继续写作）。我的确需要感谢她，她让我养成了及时备份的良好习惯，更使我意识到在离开电脑后自己依然能够而且更好地思考。

在清华大学执教多年，我日益深刻地体会到教育的伟大和神圣，是我的同事、学生使我懂得了如何忘却劳动的艰辛，并从教育的过程中获得乐趣。

我还要特别感谢机械工业出版社的温莉芳女士，在我一次次地推迟交稿时，她表现出的耐心与理解都令我既钦佩又惭愧，正是这非凡的宽容激励着我不断追求完美。

虽反复斟酌，在本书即将出版之际，内心依然惶恐。“丑媳妇终要见公婆”，恳请读者批评指正。

邓俊辉

2005 年岁末定稿

目　录

第 1 章

算法及其复杂度

1.1 计算机与算法

现代意义上的电子计算机诞生于上世纪中叶，然而实际上，在人类历史发展的长河中，计算与计算“机”早就伴随在我们的周围。打结的绳子[1]，刻痕的石头，都是人类的计算工具。下面我们就来看几个例子。

1.1.1 过指定垂足的直角边

在规划和实施复杂而规模浩大的土木工程的过程中，古代埃及人逐渐归纳并掌握了一整套基本方法。比如，早在公元前 2000 年，他们就已经知道了如下实际问题的解决方法：通过直线上给定的一点，作该直线的垂线。具体方法如图 1-1 及算法 1-1 所示。

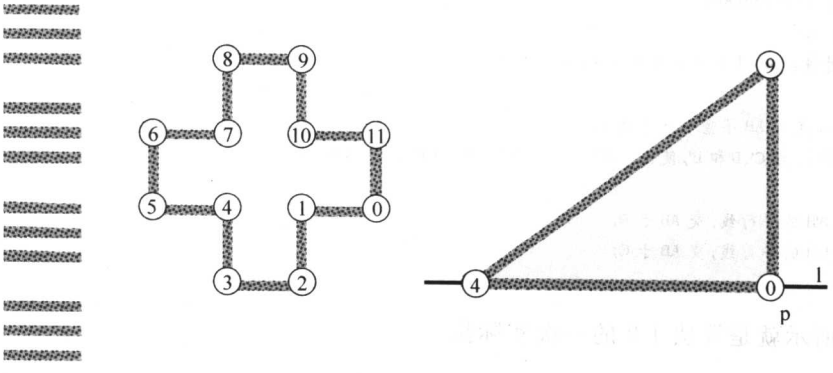


图 1-1 公元前 2000 年古埃及人使用的“绳索计算机”及其算法[2]

算法 1-1 过直线上给定点作直角的算法

算法：RightAngle(l,p)

输入：直线 l 及其上面的一点 p

---

[1] 易系辞云：上古结绳而治，后世圣人，易之以书契。

[2] 不难看出，其原理就是勾股定理的逆命题。这一定理最早记载于《周髀算经》(成书于公元前 100～200 年)，后由赵爽给出了初等而严格的证明(公元 220 年前后)。关于这一定理，《周髀算经》记载了商高向周公的一段解释：故禹之所以治天下者，此数之所由生也。也就是说，早在大禹治理黄河时，中国人就已经掌握了这一定理并应用于社会生产实践，比古埃及人的记载要早 100 年。

```
输出:经过 p、垂直于 l 的一条直线
{
  1. 取 12 段等长的绳索,首尾依次联结成一个环; //联结处称作"结",按顺时针方向编号
  2. 将 0 号结固定于给定的点 p 处;
  3. 第一个奴隶牵动 4 号结,将绳索沿直线 l 方向拉直;
  4. 另一个奴隶牵动 9 号结,将绳索尽可能地拉直;
  5. 记录下由 0 号和 9 号结确定的直线;
}
```

从外表看，古埃及奴隶用绳索制成的这一简易工具与现代电子计算机相去甚远，然而就其本质而言，二者却有着很多相似之处。首先，它们都明确地定义了问题输入的形式与要求。其次，基于若干基本的操作(比如取等长绳索、联结绳索、将绳结固定在指定点以及拉直绳索等)，具体的操作方法与规程也可以明白无误地得到描述。最后也是最重要的，只要按照这种方法来进行操作，对指定问题的任何输入，它们都能在有限的时间内给出相应的解答。因此从这个意义上讲，将 4000 年前的这一计算工具称作"绳索计算机"，一点也不过分[⊖]。

1.1.2 三等分线段

欧几里德几何是现代公理系统的鼻祖。如果从计算机的角度来看，直尺和圆规就相当于我们今天的计算机；而为了解决某一特定几何问题而设计的一套几何作图流程，则相当于今天的一个算法或程序。比如典型的三等分线段过程，就可以描述为算法 1-2：

算法 1-2 三等分给定线段的算法

```
算法:Tripartition(AB)
输入:线段 AB
输出:通过线段 AB 上的两点 C 和 D,将其三等分
{
从 A 画出一条与 AB 不重合的射线 ρ;
在 ρ 上任取三点 C'、D'和 B',使得 |AC'| = |C'D'| = |D'B'| = |AB'|/3;
连接 B'B;
经过 D'作 B'B 的平行线,交 AB 于 D;
经过 C'作 B'B 的平行线,交 AB 于 C;
}
```

图 1-2 所示就是算法 1-2 的一次实际执行结果。

当然，算法 1-2 中涉及的操作并不都是基本的，比如，"经过直线外一点作其平行线"本身就是一个几何问题。幸运的是，这些问题都可以借助相应的更为基本的算法来解决(请读者根据初中平面几何的知识给出具体的描述)，相对算法 1-2，这些算法的作用相当于"子程序"。

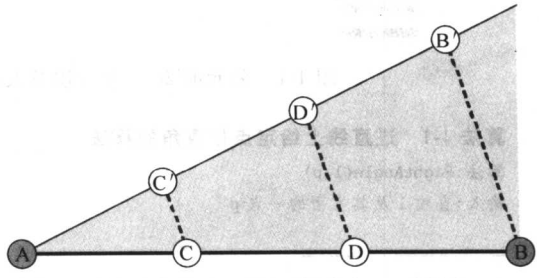


图 1-2 古希腊人的尺规计算机

⊖ 事实上，这类计算机的功能极其强大，某些方面甚至超过了现代计算机。

1.1.3 排序

我们再来看另一个算法实例。

据统计，计算机中80%的计算工作，都可以归结为同一类操作：按照某种次序，将给定的一组元素顺序排列——即排序(sorting)。比如，将 n 个整数排成一个非降序列。

1. 排序器接口

排序算法种类繁多，在第8章中我们将专门讨论这一问题。为了遵循面向对象的规范，本书中涉及的排序算法都符合如代码1-1所示的排序器接口。

代码1-1 排序器接口

```
/*
 * 排序器接口
 */
package dsa;

public interface Sorter {
  public void sort(Sequence s);
}
```

这里，我们将输入统一表示为Sequence类，第3.3节将详细介绍该类的定义及实现。

2. 起泡排序

在由一组元素组成的序列A[0..n-1]中，若某一对相邻元素A[i-1]和A[i]满足A[i-1]≤A[i]，我们就称它们是顺序的；否则是逆序的。于是，排序操作的目的，就是通过元素位置的交换，使得所有相邻元素都是顺序的，即对任意的0<i<n，都有A[i-1]≤A[i]。

由上述分析可以立即得出如下算法：从前向后逐对检查相邻元素，一旦发现逆序，就交换其位置。这一过程，称作一趟扫描交换。在图1-3中，对于如图1-3a所示的由7个整数组成的序列A[0..6]，经过一趟扫描交换之后的结果如图1-3b所示。

经过这样的一趟扫描，如果序列仍未达到完全有序，则可以再次对其进行一趟扫描交换，所得结果如图1-3c所示。事实上，我们将不断对序列进行扫描交换，直到最终其中不再含有逆序的相邻元素，结果如图1-3g所示。

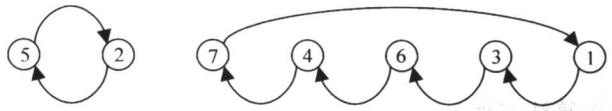
起泡排序的过程可以描述为算法1-3。

算法1-3 起泡排序算法

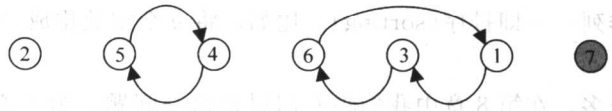
```
算法:Bubblesort(S[],n)
输入:n个元素组成的一个序列S[],每个元素由[0..n-1]之间的下标确定,元素之间可以比较大小
输出:重新调整S[]中元素的次序,使得它们按照非降次序排列
{
  从S[0]和S[1]开始,依次检查每一对相邻的元素;
  只要它们位置颠倒,则交换其位置;
  反复执行上述操作,直到每一对相邻元素的次序都符合要求;
}
```

3. 实现

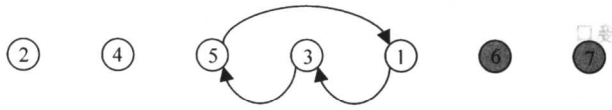
基于代码1-1所定义的排序器接口，可以实现如代码1-2所示的起泡排序器。



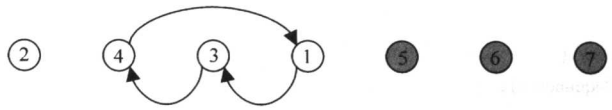
a) 初始序列



b) 经过一趟扫描交换



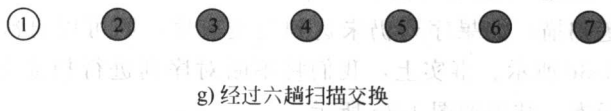
c) 经过两趟扫描交换



d) 经过三趟扫描交换

e) 经过四趟扫描交换

f) 经过五趟扫描交换



g) 经过六趟扫描交换

图 1-3 对 7 个整数做起泡排序

代码 1-2 起泡排序器

```
/*
 * 起泡排序器
 */

package dsa;

public class Sorter_Bubblesort implements Sorter {
  private Comparator C;

  public Sorter_Bubblesort()
  { this(new ComparatorDefault()); }

  public Sorter_Bubblesort(Comparator comp)
  { C = comp; }
```

```
public void sort(Sequence S) {
  int n = S.getSize();
  for (int i = 0; i<n; i ++ )
  for (int j = 0; j<n - i - 1; j ++ )
    if (0<C.compare(S.getAtRank(j),S.getAtRank(j + 1))) {
      Object temp = S.getAtRank(j);
      S.replaceAtRank(j,S.getAtRank(j + 1));
      S.replaceAtRank(j + 1,temp);
    }
  }
}
```

4. 正确性

关于起泡排序算法，很自然的一个问题是，需要经过多少趟扫描转换才能完成排序？我们甚至可以怀疑，经过有限趟扫描转换之后，是否总是能够完成排序？也就是说，它是不是一个名副其实的算法？

从图 1-3 不难看出，每经过一趟扫描，尽管并不能保证序列达到完全有序，我们却总是能够在某些方面取得一些进展。如图 1-3b 所示，经过第一趟扫描之后，序列中的最大元素必然就位，而且在此后的各轮扫描交换中，该元素将绝不会参与任何交换。因此，经过一趟扫描交换之后，我们可以只关注前面的 $n-1$ 个元素——也就是说，问题的规模减少了一个单位。实际上，这一结论对后续的每一趟扫描交换都成立。由此，根据数学归纳法可以得出如下结论：

引理 1.1　利用起泡排序算法对长度为 n 的序列进行排序，至多经过 n 轮扫描交换，所有元素都将就位，即实现完全有序。

1.1.4　算法的定义

上面的三个例子都可以称作算法。那么，究竟什么是算法呢？

所谓算法，就是在特定计算模型下，在信息处理过程中为了解决某一类问题而设计的一个指令序列。比如，对于“过直线上某一点作垂直线”这一问题，绳索及奴隶就构成了一个特定的计算模型，古埃及人基于这一模型所设计的算法 1-1，就是在这一计算模型下解决这一问题的一个算法。

更准确地说，一个算法还必须具备以下要素：

- 输入：待处理的信息，即对具体问题的描述。比如，对于上述三个例子来说，输入分别是“任意给定的直线以及其上的一点”、“任意给定的一条线段”以及“由 n 个可比较元素组成的序列”。
- 输出：经过处理之后得到的信息，即问题的答案。比如，对于上述三个例子来说，输出分别是我们所要得到的“垂直线”、“三等分点”以及“完全有序的序列”。
- 确定性：任一算法都可以描述为由若干个基本操作组成的序列。在垂直线算法中，“取等长绳索”、“联结绳索”、“将绳结固定于一点”、“沿特定方向拉直绳索”等操作都属于基本操作。在三等分线段算法中，基本操作就是欧氏作图法所允许的所有尺规操作。而在起泡排序算法中，基本操作就是图灵机所允许的各种操作——“读取某一元素的内容”、“比较两个元素的大小”以及“修改某一元素的内容”等等。
- 可行性：在相应的计算模型中，每一基本操作都可以实现，且能够在常数时间内完成。
- 有穷性：对于任何输入，按照算法，经过有穷次基本操作都可以得到正确的输出。

1.2 算法性能的分析与评价

1.2.1 三个层次

相信本书的读者都已学习并掌握了至少一种程序设计语言，比如Java。学习程序设计语言的目的，就是学会如何编写合法的程序。这里所说的“合法”，指的是合乎该语言的语法，从而保证所编写的程序能够顺利通过编译器生成可执行代码，或者能够通过解释器解释执行。从利用计算机解决实际问题的角度来看，这只是第一个层次，当然也是最基本的要求。

对于算法而言，前面提到的有穷性应该是一项起码的要求；然而遗憾的是，合法的程序却未必满足有穷性——相信大多数读者都曾编写过导致无穷循环或者递归溢出的合法程序。

实际上，我们不仅需要确定算法对任何输入都能够在有穷次操作后终止并输出结果，而且希望等待的时间尽可能地短。为此，我们首先需要确立一种尺度，以度量不同算法的效率（执行速度）；另外，我们还需要学习如何设计和使用适宜的数据结构，编写出效率高、能够处理大规模数据的程序。这些都属于第二层次的问题。前一问题将通过对算法分析理论的学习来解决，后一问题则需要通过对算法设计技巧以及相应的数据结构的学习来解决，这些也正是本书的主题。

如果将借助计算机解决实际问题比作书法，那么在第一层次上就相当于学习点、横、竖、撇和捺等基本笔画；而第二层次则相当于学习如何将不同的笔画按照一定的间架结构组成有意义的不同汉字。然而这还远远不够，作为一幅完整的书法作品，还需要在汉字之间形成大小、粗细、疏密、浓淡、奇正及枯润等方面的搭配与呼应，也就是要讲求章法。在利用计算机解决实际问题的过程中，同样存在这样一个更高层次的问题：倘若软件的规模大到任何个人或少数人都不足以开发出来的地步，而且在软件生命期内也需要很多人的协作才能得以维护，则需要进一步考虑一些更为全局性的问题，这些问题都属于软件工程学的范畴，本书将不做深入介绍。

1.2.2 时间复杂度及其度量

我们首先来解决第二层次的前一个问题：如何度量一个算法的执行速度并评价其效率？具体地讲，对于不同的输入，算法需要运行多少时间才能得出结果？

1. 问题规模、运行时间及时间复杂度

直接回答上述问题并非易事，原因在于，即使是同一算法，针对不同的输入，运行的时间并不相同。以排序问题为例，输入序列的规模、组成和次序都不是确定的，这些因素都会影响到排序算法的运行时间。在所有这些因素中，输入的规模是最重要的。还是以排序问题为例，如果是操作系统对某一文件夹内的文件按名字排序，则输入的规模通常不超过100，故可以在瞬间完成排序；反之，若需要对全中国人口普查的数据进行排序，则输入的规模将高达10^9，此时若采用起泡算法，恐怕需要经过数月甚至数年才能完成排序。

因此，为了简化分析，我们通常只考虑输入规模这一主要因素。如此一来，本节开头所提出的问题就转化为：针对不同规模的输入，算法的执行时间各是多少？如果将某一算法处理规模为n的问题所需的时间记作$T(n)$，那么随着问题规模n的增长，运行时间$T(n)$将如何增长？我们将$T(n)$称作算法的时间复杂度。

2. 渐进复杂度及大 O 记号

新的问题依然不好回答，原因在于，同一规模的输入仍不确定，通常都有很多个，算法对它们进行处理时所需的时间也不尽相同。仍以排序问题为例，由 n 个元素组成的输入序列有 $n!$ 种。因此严格说来，上面对 $T(n)$的定义并不明确(not well-defined)。如果仍然希望藉此度量算法的执行速度，那么在规模为 n 的所有输入中，我们究竟应该以哪一个输入所对应的时间作为 $T(n)$呢？

幸运的是，在评价算法的运行时间时，我们往往可以忽略其在处理小规模问题时的性能，转而关注其在处理足够大规模问题时的性能，即所谓的渐进复杂度(asymptotic complexity)。原因不难理解，小规模的问题所需的处理时间相对更少，不同算法在效率方面的差异并不明显；只有在处理大规模的问题时，这方面的差异才有质的区别。

另外，通常我们也不需要知道 $T(n)$的确切大小，而只需要对其上界作出估计。比如说，如果存在正常数 a、N 和一个函数 $f(n)$，使得对于任何 $n>N$，都有

$$T(n) < a \times f(n)$$

我们就可以认为在 n 足够大之后，$f(n)$给出了 $T(n)$的一个上界。对于这种情况，我们记之为

$$T(n) = O(f(n))$$

这里的 O 称作“大 O 记号”(big-O notation)。

3. 机器差异与基本操作

在实际计算环境中，如上定义的 $T(n)$依然无法度量。即便是同一算法、同一输入，在不同的硬件平台上、使用不同的操作系统所需要的计算时间都不相同。然而实际上，无论在何种计算环境中，每一次基本操作都可以在常数时间内完成，因此如果根据算法所需执行的基本操作次数来表示 $T(n)$，就可以更加客观地反映算法的效率。

以第 1.1.3 节介绍的起泡排序算法为例，如果将该算法处理长度为 n 的序列所需的时间记作 $T(n)$，$T(n)$的上界是多少呢？根据上面的分析，我们只需估计出该算法所需执行的基本操作次数。

从算法 1-3 和代码 1-2 都可以看出，该算法由内、外两层循环组成。内循环从前向后依次检查各相邻的元素对，如有必要，则交换逆序的元素对，因此在每一轮内循环中，至多需要扫描和比较 $n-1$ 对元素，至多需要交换 $n-1$ 对元素。无论是比较元素大小还是交换元素的位置，都属于基本操作，故每一轮内循环至多需要执行 $2(n-1)$次基本操作。

另外，根据引理 1.1，外循环至多执行 $n-1$ 轮。因此，总共需要执行的基本操作不超过 $2(n-1)^2$ 次。若取 $a=2$ 和 $f(n)=n^2$，则有

$$T(n) < 2 \times n^2$$

也就是说

$$T(n) = O(n^2)$$

以上可以看出，大 O 记号实质上是对算法执行效率的一种保守估计——对于规模为 n 的任意输入，算法的运行时间都不会超过 $O(f(n))$。

4. 算法运行时间的实测统计

上面对起泡排序算法的分析，是估计算法时间复杂度的最直接的方法，我们在本书中还将看到更多这样的例子。然而有些算法的时间复杂度极难从理论上作出分析，此时我们可以采用实验的方法，随机选择足够多规模不同的输入，通过实测统计得出运行时间随输入规模而增长的趋势。

5. 大 Ω 记号

如果存在正常数 a、N 和一个函数 $g(n)$，使得对于任何 $n>N$，都有

$$T(n) > a \times g(n)$$

我们就可以认为在 n 足够大之后，$g(n)$给出了 $T(n)$的一个下界。对于这种情况，我们记之为

$$T(n) = \Omega(g(n))$$

这里的 Ω 称作“大 Ω 记号”(big-Ω notation)。

大 Ω 记号与大 O 记号正好相反，它是对算法执行效率的一种乐观估计——对于规模为 n 的任意输入，算法的运行时间都不会低于 $\Omega(g(n))$。

6. Θ 记号

如果存在正常数 $a<b$、N 和一个函数 $h(n)$，使得对于任何 $n>N$，都有

$$a \times h(n) < T(n) < b \times h(n)$$

我们就可以认为在 n 足够大之后，$h(n)$给出了 $T(n)$的一个确界。对于这种情况，我们记之为

$$T(n) = \Theta(h(n))$$

Θ 记号是对算法执行效率的一种准确估计——对于规模为 n 的任意输入，算法的运行时间都与 $\Theta(h(n))$同阶。

1.2.3 空间复杂度

衡量算法性能的另一个重要方面，就是算法所需使用的存储空间量，即算法空间复杂度。显然，对于同样的输入规模，在时间复杂度相同的前提下，我们希望算法所占用的空间越少越好。为此，我们可以借助第 1.2.2 节所引入的各种记号来度量算法的空间复杂度，在此不再赘述。

不过，在通常情况下，我们将更多地分析和讨论算法的时间复杂度，甚至只关注时间复杂度。之所以能够这样做，是基于以下事实：

观察结论 1.1 就渐进复杂度的意义而言，在任何一个算法的任何一次运行过程中，其实际占用的存储空间都不会多于其间执行的基本操作次数。

实际上，每次基本操作只会涉及常数规模的空间。于是，纵然每次基本操作所占用的存储空间都是新开辟的，所需的空间总量也不过与基本操作的次数同阶。从这个意义上说，时间复杂度本身就是空间复杂度的一个上界。

当然，空间复杂度本身也有其存在的意义，尤其是非常在乎空间效率的应用场合，或者是当问题的输入规模极为庞大时。在本书后续章节中，我们将结合一些实际问题就此进行讨论。

1.3 算法复杂度及其分析

我们不仅要了解算法复杂度的度量规则，更要学会如何对各个具体算法的复杂度进行分析。在第 1.2.2 节所引入的度量算法复杂度的三种记号中，大 O 记号是最基本的，也是最常用到的。按照渐进复杂度的思想，可以将算法的复杂度按照高低划分为若干典型的级别。

1.3.1 O(1)——取非极端元素

首先来看这样一个问题：给定整数子集 S，$+\infty > |S| = n \geqslant 3$，从中找出一个元素 $a \in S$，使得 $a \neq \max(S)$且 $a \neq \min(S)$。也就是说，在最大、最小者之外，取出任意一个数。这一问题可以用算法 1-4 解决。