HZ BOOKS
华章教育

计算机科学与技术学科研究生教材

计算机算法基础

沈孝钧　编著

ESSENTIALS OF
COMPUTER ALGORITHMS

机械工业出版社
China Machine Press

计算机科学与技术学科研究生教材

计算机算法基础

沈孝钧 编著

ESSENTIALS OF
COMPUTER ALGORITHMS

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

计算机算法基础 / 沈孝钧编著. —北京：机械工业出版社，2013.6
（计算机科学与技术学科研究生教材）

ISBN 978-7-111-42595-3

Ⅰ. 计… Ⅱ. 沈… Ⅲ. 电子计算机-算法理论-高等学校-教材 Ⅳ. TP301.6

中国版本图书馆 CIP 数据核字（2013）第 109019 号

本书是作者 20 多年在国内外教学与科研实践的结晶，深入浅出地介绍计算机算法中涉及的基本理论和方法。主要内容包括算法复杂度的概念和表达、分治法、贪心法、动态规划、图的遍历技术、平扫线技术、回溯法、分支限界法、α-β剪枝等。在讲述这些理论和方法的同时，介绍一系列重要问题的算法，包括排序问题、选择问题、最小支撑树问题、最短路径问题、网络流问题、二分图的匹配问题、字符串的匹配问题以及若干几何算法问题，并将这些问题的解法及所用技术紧密相连，有机地编排在一起。此外，本书还介绍了问题本身固有的计算复杂性的概念和 NP 完全问题的理论以及近似算法。

本书讲解细腻、分析透彻，以探索解决问题的方式深入分析了大量案例，使读者能清晰触摸到作者的思维方法，并建立起自己独立思考的学习习惯。本书可以作为计算机科学等相关专业本科生、研究生的教材，也可供从事计算机算法设计与分析工作的教师与研究人员参考。

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）
责任编辑：朱秀英
北京瑞德印刷有限公司印刷
2014 年 1 月第 1 版第 1 次印刷
185mm×260mm · 18.75 印张
标准书号：ISBN 978-7-111-42595-3
定 价：45.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

前　言

计算机算法是计算机科学的一个重要分支，是计算机专业的一门必修课。作者从事这门课的教学及相关研究工作二十余年，在走了不少弯路后逐渐领悟到算法的思维方法和设计技巧，积累了一些经验，希望通过书的形式让更多的读者一开始便得到正规的训练，为今后进一步深造奠定坚实的基础。即使是不再深造的学生，能正确地运用第一门算法课中的方法解决实际工作中的问题也是至关重要的。我们经常看到，有些学过算法的学生仍喜欢用自己习惯的复杂度高的方法来解决问题，这是还没真正入门的表现。这本书的主要目的就是使学生能养成自觉地运用所学方法去追求最好结果的良好习惯。

书是为学生而用，为读者而写。作者本着这一宗旨，以共同探索解决问题的方式来写作，使读者能触摸到作者的思维方法，并能建立起自己独立思考的学习习惯。培养独立思考和创新的欲望对从事算法工作的人来讲十分重要，没有哪一种具体的算法可以解决所有问题，也没有哪一本书是万能的，只有勤于思考的人才能最好地解决实际问题。

本书包含了任何算法书都会包含的基本内容，并进行了扩充，主要包括网络流、字符串匹配、计算几何算法、近似算法以及穷举搜索法等，并在附录中介绍了红黑树。每章后配有精心挑选的习题，其中相当一部分是作者在教学实践中设计的。打星号的章节和题目较难或较偏，供选择。全书的内容足够用于两个学期教学，经过适当地取舍，本书可用于高等院校计算机及相关专业高年级本科生、硕士生及博士生教材。

最后，作者要感谢启蒙导师朱宗正教授（已去世）、博士生导师 C. L. Liu 教授的指导和熏陶，感谢清华大学、南京理工大学、伊利诺大学香滨分校的培养和提供的优良学术环境，感谢密苏里大学提供的教学和研究机会，并感谢朋友和家人的鞭策和支持。

因作者水平有限，疏漏之处在所难免，欢迎读者批评指正。

沈孝钧

2013 年 3 月 20 日

教学建议

教学章节	教学要求			
	一学期课程	课时	二学期课程	课时
第1章 概　述	算法复杂度度量的方法	1	算法复杂度度量的方法	1
	函数增长快慢的比较 O、Ω、Θ记号	1	函数增长快慢的比较，O、Ω、Θ记号	1
第2章 分　治　法	分治法原理	1	分治法原理	1
	解递推关系	2	解递推关系	2
	主方法	1	主方法	1
第3章 基于比较的排序	插入排序	1	插入排序	1
	合并排序	1	合并排序	1
	堆排序	2	堆排序	2
	快排序（不详细分析）	1	快排序	2
第4章 不基于比较的排序	决策树和排序最坏情况下界	1	决策树和排序最坏情况下界	1
			外路径总长与平均情况下界	1
			全路径总长与堆排序最好情况下界	1
	计数排序	1	计数排序	1
	基数排序，桶排序（选讲）	（1）	基数排序，桶排序	（1）
第5章 中位数和任一顺序数的选择	最大数和最小数的选择（选讲）	（1）	最大数和最小数的选择	1
	任一顺序数选择（选讲）	（1）	任一顺序数选择	1
	找 k 个最大数讨论（选讲）	（1）	找 k 个最大数讨论	1
第6章 动态规划	动态规划原理，矩阵连乘	2	动态规划原理，矩阵连乘	2
	最长公共子序列	1	最长公共子序列	1
	最佳二元搜索树（只介绍问题）		最佳二元搜索树	（2）
	多级图	1	多级图	1
	最长递增子序列	（1）	最长递增子序列（只讲 n^2 算法）	1
第7章 贪心算法	原理、最佳邮局设置问题	1	原理、最佳邮局设置问题	1
	最佳活动安排问题	1	最佳活动安排问题	1
	哈夫曼编码问题	1	哈夫曼编码问题	1
	最佳加油计划	（1）	最佳加油计划	1
第8章 图的周游算法	图的表示、广度优先搜索算法	2	图的表示、广度优先搜索算法	2
	无向图的二着色问题	1	无向图的二着色问题	1
	深度优先搜索算法	2	深度优先搜索算法	2
	拓扑排序及应用	1	拓扑排序及应用	1
	强连通分支算法	1	强连通分支算法	1
			双连通分支算法	1
第9章 图的最小支撑树	问题定义和通用算法	2	问题定义和通用算法	2
	Kruskal 算法	（1）	Kruskal 算法	1
	Prim 算法	1	Prim 算法	2
第10章 单源最短路径	Dijkstra 算法	2	Dijkstra 算法	2
	Bellman-Ford 算法	（1）	Bellman-Ford 算法	1

（续）

教学章节	教学要求			
	一学期课程	课时	二学期课程	课时
第 11 章 网络流			网络流定义，割的定义	1
			最大流最小割定理	1
			Ford-Fulkerson 方法	1
			Edmond-Karp 算法	1
			Dinic 算法	（1）
			二部图匹配、Hall 氏定理	1
			Birkhoff-von Neuman 定理	（1）
			推进-重标号算法	（2）
第 12 章 计算几何基础			平面线段及相互关系	2
			平扫线技术	2
			平面点集的凸包	2
			最近点对问题	1
第 13 章 字符串匹配			一个朴素的字符串匹配算法	1
			Rabin-Karp 算法（选讲）	（1）
			基于有限状态自动机的匹配算法	2
			KMP 算法	2
第 14 章 NP 完全问题	预备知识、P 类语言	2	预备知识、P 类语言	2
	非确定图灵机、多项式检验算法	1	非确定图灵机、多项式检验算法	1
	NP 类语言和 NP 完全问题定义	1	NP 类语言和 NP 完全问题定义	1
	第一个 NPC 问题	1	第一个 NPC 问题	1
	若干著名 NPC 问题的证明	3	若干著名 NPC 问题的证明	4
	SAT，3-SAT（只定义），团，顶点覆盖	（2）	SAT，3-SAT（只定义），团，顶点覆盖	（2）
	哈密尔顿回路（只定义），TSP		哈密尔顿回路（只定义），TSP	
	子集和（只定义），集合划分（只定义）		子集和，集合划分	
第 15 章 近似算法			近似算法的性能评价、顶点覆盖问题	1
			货郎担问题	1
			集合覆盖问题	1
			MAX-3-SAT 问题，加权顶点覆盖问题	1
			子集和问题	1
			鸿沟定理和不可近似度	1
第 16 章 穷举搜索	搜索问题及方法的描述	（1）	搜索问题及方法的描述	1
	回溯法（选讲）	（2）	回溯法	2
			分支限界法	1
			博弈树（game tree）和α-β剪枝（选讲）	（1）
总课时	基本部分	40		80
	选讲部分	13		11

说明：

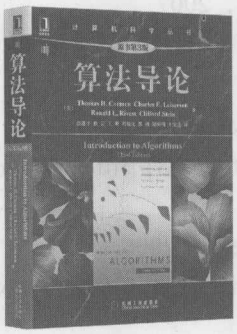
1）本建议仅供参考。学生也许已在前续课学过 Huffman 码或其他内容等，则应做适当调整。

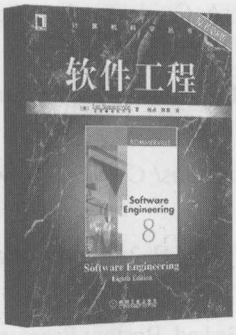
2）本建议以一学期最少 40 课时授课时间来设计的，最好能有 50 课时。为灵活起见，部分内容可由老师选讲，选讲部分用括号表示。

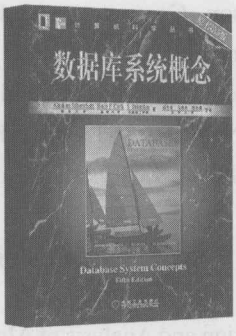
3）除选讲以外的内容为重点内容，建议不做删减，除非学生已学过。

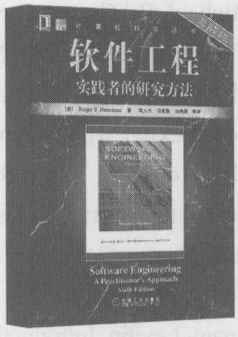
4）对各章节的课时分配是大概估计，有的也许要 1.5 课时，有的需 0.5 课时，老师应灵活掌握。

推荐阅读









算法导论（原书第3版）

2006、2007 CSDN、《程序员》杂志评选的十大IT好书之一　算法中的经典权威之作

编译原理（原书第2版）

编译领域无可替代的经典著作，被广大计算机专业人士誉为“龙书”

设计模式：可复用面向对象软件的基础

经典教材 权威之作

软件工程（原书第8版）

最受欢迎的软件工程指南

数据库系统概念（原书第5版）

数据库系统方面的经典教材，被美誉为“帆船书”

软件工程：实践者研究方法（原书第6版）

全球上百所大学和学院采用 最受欢迎的软件工程指南

目　录

前言
教学建议
第 1 章　概述 …… 1
1.1　算法与数据结构及程序的关系 …… 1
1.1.1　什么是算法 …… 1
1.1.2　算法与数据结构的关系 …… 1
1.1.3　算法与程序的关系 …… 1
1.1.4　选择排序的例子 …… 2
1.1.5　算法的伪码表示 …… 2
1.2　算法复杂度分析 …… 3
1.2.1　算法复杂度的度量 …… 3
1.2.2　算法复杂度与输入数据规模的关系 …… 3
1.2.3　输入数据规模的度量模型 …… 4
1.2.4　算法复杂度分析中的两个简化假设 …… 4
1.2.5　最好情况、最坏情况和平均情况的复杂度分析 …… 5
1.3　函数增长渐近性态的比较 …… 6
1.3.1　三种比较关系及 O、Ω、Θ 记号 …… 6
1.3.2　表示算法复杂度的常用函数 …… 6
1.4　算法复杂度与问题复杂度的关系 …… 8
1.4.1　问题复杂度是算法复杂度的下界 …… 8
1.4.2　问题复杂度与最佳算法 …… 8
1.4.3　易处理问题和难处理问题 …… 8
习题 …… 9
第 2 章　分治法 …… 10
2.1　分治法原理 …… 10
2.1.1　二元搜索的例子 …… 10
2.1.2　表示复杂度的递推关系 …… 11
2.2　递推关系求解 …… 11
2.2.1　替换法 …… 12
2.2.2　序列求和法和递归树法 …… 13
2.2.3　常用序列和公式 …… 14
2.2.4　主方法求解 …… 16
习题 …… 16
第 3 章　基于比较的排序算法 …… 19
3.1　插入排序 …… 19
3.1.1　插入排序的算法 …… 19
3.1.2　插入排序算法的复杂度分析 …… 20
3.1.3　插入排序的优缺点 …… 20
3.2　合并排序 …… 21
3.2.1　合并算法及其复杂度 …… 21
3.2.2　合并排序的算法及其复杂度 …… 22
3.2.3　合并排序的优缺点 …… 24
3.3　堆排序 …… 24
3.3.1　堆的数据结构 …… 24
3.3.2　堆的修复算法及其复杂度 …… 25
3.3.3　为输入数据建堆 …… 27
3.3.4　堆排序算法 …… 28
3.3.5　堆排序算法的复杂度 …… 29
3.3.6　堆排序算法的优缺点 …… 29
3.3.7　堆用作优先队列 …… 30
3.4　快排序 …… 31
3.4.1　快排序算法 …… 31
3.4.2　快排序算法最坏情况复杂度 …… 33
3.4.3　快排序算法平均情况复杂度 …… 34
3.4.4　快排序算法最好情况复杂度 …… 34
3.4.5　快排序算法优缺点 …… 36
习题 …… 36
第 4 章　不基于比较的排序算法 …… 39
4.1　比较排序的下界 …… 39
4.1.1　决策树模型及最坏情况下界 …… 39
*4.1.2　二叉树的外路径总长与平均情况下界 …… 41
*4.1.3　二叉树的全路径总长及堆排序最好情况下界 …… 43

4.2 不基于比较的线性时间排序算法……46
4.2.1 计数排序……46
4.2.2 基数排序……48
4.2.3 桶排序……49
习题……51

第 5 章 中位数和任一顺序数的选择……53
5.1 问题定义……53
5.2 最大数和最小数的选择……53
5.2.1 最大和最小顺序数的选择算法及其复杂度……53
5.2.2 同时找出最大数和最小数的算法……54
5.3 线性时间找出任一顺序数的算法……55
5.3.1 最坏情况复杂度为 $O(n)$的算法……56
5.3.2 平均情况复杂度为 $O(n)$的算法……58
5.4 找出 k 个最大顺序数的算法……58
5.4.1 一个理论联系实际的问题……58
5.4.2 利用堆来找 k 个最大的顺序数的算法……59
5.4.3 利用锦标赛树来找 k 个最大顺序数的算法……59
习题……60

第 6 章 动态规划……62
6.1 动态规划的基本原理……62
6.2 矩阵连乘问题……63
6.3 最长公共子序列问题……68
*6.4 最佳二元搜索树……71
6.5 多级图及其应用……76
6.6 最长递增子序列问题……78
习题……80

第 7 章 贪心算法……86
7.1 最佳邮局设置问题……86
7.2 最佳活动安排问题……87
7.3 哈夫曼编码问题……89
7.3.1 前缀码……89
7.3.2 最佳前缀码——哈夫曼编码……90
7.4 最佳加油计划问题……94
习题……96

第 8 章 图的周游算法……100
8.1 图的表示……100
8.1.1 邻接表……100
8.1.2 邻接矩阵……101
8.2 广度优先搜索及应用……101
8.2.1 广度优先搜索策略……101
8.2.2 广度优先搜索算法及距离树……102
8.2.3 无向图的二着色问题……105
8.3 深度优先搜索及应用……107
8.3.1 深度优先搜索策略……107
8.3.2 深度优先搜索算法和深度优先树……107
8.3.3 深度优先搜索算法举例和图中边的分类……110
8.3.4 拓扑排序……112
8.3.5 无回路有向图中最长路径问题及应用……114
8.3.6 有向图的强连通分支的分解……116
8.3.7 无向图的双连通分支的分解……119
习题……124

第 9 章 图的最小支撑树……128
9.1 计算最小支撑树的一个通用的贪心算法策略……129
9.2 Kruskal 算法……130
9.3 Prim 算法……133
习题……137

第 10 章 单源最短路径……140
10.1 Dijkstra 算法……140
10.2 Bellman-Ford 算法……144
习题……148

第 11 章 网络流……150
11.1 网络模型和最大网络流问题……150
11.2 网络中流与割的关系……152
11.2.1 网络中的割及其容量……153
11.2.2 剩余网络和增广路径……154
11.2.3 最大流最小割定理……156
11.3 Ford-Fulkerson 方法……157
11.3.1 Ford-Fulkerson 的通用方法……157
11.3.2 Edmonds-Karp 算法……159

11.3.3 Dinic 算法 …… 161
11.3.4 0-1 网络的最大流问题 …… 164
11.4 二部图的匹配问题 …… 165
11.4.1 用网络流求二部图的最大匹配算法 …… 166
11.4.2 Philip-Hall 婚配定理 …… 167
11.4.3 Birkhoff-von Neuman 定理 …… 169
11.5 推进-重标号算法简介 …… 170
11.5.1 预流和高度函数 …… 170
11.5.2 在剩余图中对顶点的两个操作 …… 171
11.5.3 推进-重标号算法的初始化 …… 172
11.5.4 推进-重标号的通用算法 …… 172
习题 …… 174

第 12 章 计算几何基础 …… 177
12.1 平面线段及相互关系 …… 177
12.1.1 向量的点积和叉积 …… 177
12.1.2 平面线段的相互关系 …… 178
12.2 平扫线技术和线段相交的确定 …… 181
12.2.1 平扫线的状态 …… 182
12.2.2 用平扫线确定线段相交的算法 …… 182
12.3 平面点集的凸包 …… 185
12.3.1 Graham 扫描法 …… 186
12.3.2 Jarvis 行进法 …… 190
12.4 最近点对问题 …… 192
习题 …… 195

第 13 章 字符串匹配 …… 197
13.1 一个朴素的字符串匹配算法 …… 197
13.2 Rabin-Karp 算法 …… 198
13.3 基于有限状态自动机的匹配算法 …… 199
13.3.1 有限状态自动机简介 …… 199
13.3.2 字符串匹配自动机 …… 200
13.3.3 基于有限状态自动机的串匹配算法 …… 202
13.4 Knuth-Morris-Pratt（KMP）算法 …… 203
13.4.1 模式的前缀函数 …… 203
13.4.2 KMP 算法的伪码 …… 205
习题 …… 207

第 14 章 NP 完全问题 …… 208
14.1 预备知识 …… 209
14.1.1 图灵机 …… 209
14.1.2 符号集和编码对计算复杂度的影响 …… 210
14.1.3 判断型问题和优化型问题及其关系 …… 210
14.1.4 判断型问题的形式语言表示 …… 211
14.1.5 多项式关联和多项式归约 …… 212
14.2 P 和 NP 语言类 …… 214
14.2.1 非确定图灵机和 NP 语言类 …… 214
14.2.2 多项式检验算法和 NP 类语言 …… 215
14.3 NP 完全语言类和 NP 完全问题 …… 216
14.3.1 第一个 NPC 问题 …… 217
14.3.2 若干著名 NPC 问题的证明 …… 220
习题 …… 231

第 15 章 近似算法 …… 236
15.1 近似算法的性能评价 …… 236
15.2 顶点覆盖问题 …… 237
15.3 货郎担问题 …… 238
15.3.1 满足三角不等式关系的货郎担问题 …… 238
15.3.2 无三角不等式关系的一般货郎担问题 …… 241
15.4 集合覆盖问题 …… 242
15.5 MAX-3-SAT 问题 …… 244
15.6 加权的顶点覆盖问题 …… 245
15.7 子集和问题 …… 247
15.7.1 一个保证最优解的指数型算法 …… 247
15.7.2 子集和问题的一个完全多项式近似机制 …… 248
*15.8 鸿沟定理和不可近似性 …… 251
15.8.1 鸿沟定理 …… 251
15.8.2 任务均匀分配问题 …… 252
习题 …… 254

第 16 章　穷举搜索 ……257
16.1　搜索问题及方法的描述 ……257
16.2　回溯法 ……259
16.2.1　回溯法的通用算法 ……259
16.2.2　*n* 皇后问题 ……259
16.2.3　子集和问题 ……260
16.2.4　回溯法的效率估计 ……262
16.3　分支限界法 ……263
16.3.1　分支限界法解 *n* 皇后问题 ……264
16.3.2　0/1 背包问题 ……266
16.4　博弈树和α-β剪枝 ……271
16.4.1　博弈树及其评估的方法 ……271
16.4.2　α-β剪枝法 ……275
习题 ……276
附录　红黑树 ……278
参考文献 ……289

第1章 概 述

计算机算法的设计与分析，简称为计算机算法，是计算机科学领域中一个重要分支，它主要研究两方面问题：

1）如何分析一个给定算法的时间复杂度。

2）如何为给定的计算问题设计一个算法，使得它的复杂度最低。

通俗讲，这里的时间复杂度就是用多少时间，显然以上两个问题是紧密相连的。因实用价值不大，计算机算法课一般已不太讨论空间复杂度，即用多少存储单元的问题，本书也是这样。这一章，我们介绍算法的时间复杂度的基本概念以及分析时遵循的基本原则。

1.1 算法与数据结构及程序的关系

计算机算法课是数据结构课和程序设计课的后续课程。一个简单的问题是，学了数据结构课和程序设计课之后，为什么还要学算法？程序本身是算法吗？这一节逐步给出简单回答。在深入学习之后，读者自己会有更准确的理解和领悟。

1.1.1 什么是算法

简单地说，算法（algorithm）是为解决某一计算问题而设计的一个过程，其每一步必须能在计算机上实现并在有限时间内完成。广义地讲，一个用某一语言（比如 C++ 或机器语言）编写的程序也可称为算法。但是，为了理论分析的严格性和方便，我们对算法的定义加了某些限制。经过下面的讨论，我们会了解是哪些限制及为什么要加这些限制。

1.1.2 算法与数据结构的关系

算法与数据结构是密不可分的，除极少数算法外，几乎所有算法都需要数据结构的支持，而且数据结构的优劣往往决定算法的好坏。数据结构把输入的数据及运算过程中产生的中间数据以某种方式组织起来以便于动态地寻找、更改、插入、删除等。没有一种数据结构是万能的，我们应根据问题和算法的需要选用和设计数据结构，而在讨论数据结构时也必定会讨论其适用的算法。所以，数据结构课程与算法课程的内容往往有很大重叠。但是，数据结构课程需要解释其在计算机上的具体实现，而算法课程着重讨论在更为抽象的层次上解决问题的技巧及分析方法。打个比方，数据结构就好像汽车零件，例如发动机、车轮、车窗、车闸、座椅、灯光、方向盘等，而算法就好像是汽车总体设计。我们假定读者熟悉常用的一些数据结构，包括数组、队列、堆栈、二叉树等，而略去对它们的介绍。读者还应当具有基本的编写程序的知识。

1.1.3 算法与程序的关系

一段用某种计算机语言写成的源码，如果可以在计算机上运行并正确地解决一个问题，则称为一个程序。程序必须严格遵守该语言规定的语法（包括标点符号），并且编程时往往还必须考虑到计算机的物理限制，例如，最大允许的整数在 32 位机上和 16 位机上是不同的。而算法则不依赖于某种语言，更不依赖具体计算机的限制。只要步骤和逻辑正确，一个算法可以用任何一种语言表达。当然这种语言必须清楚无误地定义每一步骤且能够让稍懂程序的人看懂。这样，设计算法者可以着重考虑解题方法而免去不必要的琐碎的语法细节。所以，算法通常不是程序，但一定

可以用任一种语言的程序来实现。(我们假定机器有足够大的内存。)

1.1.4 选择排序的例子

在这一节，我们举一个例子——选择排序（selection sort）——来说明算法与程序的关系。排序问题就是要求把 n 个输入的数字从小到大（或从大到小）排好。我们假定这 n 个输入的数字是存放在数组 $A[1..n]$ 中。下面的算法称为选择排序。

【例 1-1】选择排序。

输入：$A[1], A[2], \cdots, A[n]$

输出：把输入的 n 个数重排使得 $A[1] \leqslant A[2] \leqslant \cdots \leqslant A[n]$

```
Selection-Sort (A[1..n])
1   for (i ← 1, i ⩽ n, i++)
2       key ← i
3       for (j ← i, j ⩽ n, j++)
4           if A[j] < A[key]
5               then key ← j
6           endif
7       endfor
8       A[i] ↔ A[key]
9   endfor
10  End
```

这个算法看上去像 C++ 程序，但不是。实际上，它不遵守目前为止任一个可在计算机上编译的语言规定的语法，但它把算法的步骤描述得很清楚。这个算法含有 n 步，对应于变量 i 从 1 变到 n。第一步，它把最小数选出并放在 $A[1]$中；第二步，它把余下的在 $A[2..n]$中的最小数选出并放在 $A[2]$中，……，第 i 步，它把余下的在 $A[i..n]$中最小的数选出并放在 $A[i]$中。当 $i=n$ 时，排序完成。算法中第 2 行到第 7 行表明该算法是用顺序比较的方法找到 $A[i..n]$中最小的数所在的位置 $A[key]$，然后交换 $A[i]$和 $A[key]$，该算法显然是正确的。

1.1.5 算法的伪码表示

从上一节例子中，我们看到算法的描述不依赖于某一语言，但又必须用某种语言去描述。我们把这个语言称为伪语言（pseudo language），而用该语言所表达的算法称为伪码（pseudo code）。不同的人可用不同的伪码写算法。本书允许任何含义清楚的伪码，例如上一节中的例 1-1。但是我们应当注意符号的一致性，例如我们用“←”表示赋值，则不要与“=”或“:=”混用。用伪码可以方便我们对算法的描述，有时还可以大大简化描述。例如，例 1-1 中的算法还可以描述如下。

【例 1-2】选择排序的另一种描述。

```
Selection-Sort (A[1..n])
1   for (i ← 1, i ⩽ n, i++)
2       find j such that A[j] = Min{A[i], A[i+1], ⋯, A[n]}
3                   //这里，符号 Min 表示找出集合中有最小值的元素
4       A[i] ↔ A[j]
5   endfor
6   End
```

显然，这样的伪码大大简化了描述，突显了思路和方法，且易于分析。本书的伪码以英文为主，适当加入中文注释，以简洁、准确为原则。

1.2 算法复杂度分析

简单地说，某个算法的时间复杂度（time complexity）是它对一组输入数据执行一次运算并产生输出所需要的时间。早期的计算机使用者往往只注重程序的正确性而忽略了其时间的复杂度，因为当时计算机的速度与人算、珠算、机械式计算机或电动计算机相比太快了，似乎不需要时间。但后来人们困惑地发现，调试好的程序在实际应用时却迟迟不能算出结果，因而往往导致上百万美元投资的失败。原来，程序是正确的，但是，随着所解问题的规模增大，每次执行的时间却成十倍、百倍、千倍地增长。所以，算法复杂度的分析成了一个十分重要和必须考虑的问题。

1.2.1 算法复杂度的度量

既然算法复杂度很重要，那我们该如何度量呢？直接在机器上测量执行一次算法的时间显然不是好办法，因为即使是同样的算法和同样的输入数据，在快速计算机上执行和在慢速计算机上执行的时间也是不同的。而且，不同的编译程序也会影响计算时间。这样的测量不能反映算法本身的优劣。

一个客观的度量方法是数一数在执行一次算法时共需要多少次基本运算。基本运算指一条指令或若干条指令可以完成的操作，例如加法、减法、乘法、除法、赋值、两个数的比较，等等。不同语言提供的基本运算集合也许不完全相同，但大部分相同。某一语言所允许的不同的基本运算的个数是一个常数，且很少超出 300。但是，这一办法说起来容易做起来难。就拿例 1-1 中这一简单算法来说，有变量 i 和 n 的比较，有 i 每次加 1 的运算，有数组 A 中数字的比较，等等。数出一个准确的数很困难。对于稍微复杂一点的算法，这一办法很难实行。所以我们必须简化对复杂度的度量。

解决的办法是，我们只统计算法中一个主要的基本运算被执行的次数并把它作为算法的复杂度。这里，主要的基本运算指的是在算法中被执行得最多的一个运算。这样，度量会大为简化。例如，在例 1-1 中，第 4 行的比较运算是一个主要运算。因此，选择排序的复杂度可以这样得出：

选出最小的数并放入 $A[1]$需要（$n-1$）次比较；

选出第二小的数并放入 $A[2]$需要（$n-2$）次比较；

…

选出第 i 小的数并放入 $A[i]$需要（$n-i$）次比较；

…

所以，总共需要的比较次数即复杂度是

$$T(n) = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}. \qquad (1.1)$$

那么，这种简化是否合理呢？我们在下面会继续讨论。

1.2.2 算法复杂度与输入数据规模的关系

算法的基本功能是将一组输入数据进行处理和运算，并产生和输出符合所解问题需要的结果。显然，一个算法的复杂度与输入数据的规模（通常用 n 表示）有关系。例如，（1.1）式表明，选择排序中 n 越大，则所需比较的次数就越多。我们当然希望算法中主要的基本运算的执行次数要少，但更重要的是，我们希望其复杂度随着 n 的增长而缓慢地增长。所以，一个算法的复杂度，就像（1.1）式那样，是一个以输入数据规模 n 为自变量的函数 $f(n)$。假定某一问题有两个算法，

其复杂度分别为 $f(n)$ 和 $g(n)$。如果在 n 趋向无穷大时，$f(n) < g(n)$，我们则认为第一个算法优于第二个算法。例如，$f(n) = 200n, g(n) = n^2$，$f(n)$ 比 $g(n)$ 好。注意，在 n 小于 200 时，$f(n)$ 并不优于 $g(n)$，所以在解决具体问题时，我们应当理论联系实际以确定用哪个算法。但是，在算法分析领域，我们只注重考虑在 n 趋向无穷大时的情形以便于理论的分析，这是因为函数增长快慢的本质在这种情形下才真正体现出来。所以，算法分析的结果对解决实际问题起着巨大的和决定性的指导作用，而理论联系实际的工作需要由解决实际问题的工作者去完成。本书将尽量把理论的讨论与实际问题相结合。

因为我们必须讨论 n 趋向无穷大时的情形，所以算法所解决的问题必须允许无穷多个不同大小的输入数据。比如排序问题，算法必须是能将任意多的数字排好序的一般方法。例 1-1 中选择排序是一个算法，因为 n 可以任意大。作为反例，如果一个程序周期性地计算每周 7 天测量的温度的平均值，那么这个程序不在我们讨论的算法范畴内。这是我们对算法定义的最主要的限制，这也是对我们要解决的问题的限制，即所解问题必须有无穷多个可能的不同规模的输入数据，而每一个具体的输入数据则称为该问题的一个实例（instance）。例如，5.1、3.2、8.6、9.7、4.0 这 5 个数可认为是排序问题的一个实例。解决一个问题的算法必须能够对任何一组输入数据进行运算并输出正确结果。

通常讨论的算法还有一些其他限制，例如，算法不可以无结果地永远不停地运算下去等。因本书所讨论的算法不会涉及这些问题，我们略去对它们的解释。上面的限制，即要求输入规模 n 可趋向无穷大，是最主要的。

1.2.3 输入数据规模的度量模型

因为算法的复杂度是输入数据规模大小（简称输入规模）的函数，那么如何来度量输入规模呢？一般所采用的模型是用输入数据中数字的个数或输入的集合中元素的个数 n 作为输入的大小。在这个模型下，每个数，不论其数值大小，均占有一个存储单元，而任何一个基本运算所需时间是常数并不受被操作数大小的影响。例如，做 5+7 和做 10 225 566+45 787 899 所需时间是一样的。这本书也采用这个模型。这个模型也有其缺点。比如，它把变量 n 也作为一般变量处理。这似乎不太合理。一方面允许 n 趋向无穷大，而另一方面又认为任何与 n 有关的基本运算像其他数字一样只需常数时间。这是理论不完美的地方。但是，这部分的影响一般小于算法中主要部分的复杂度，不影响算法优劣的比较。这一模型在实践中被广泛采用而且非常成功。

另一个模型是用输入数据的二进制表示中所用的比特数作为输入数据的规模。这个模型在输入数据只含一个数或几个数时很有用。比如，我们希望判断一个数 a 是否为质数时，输入数据只有一个数。这时，前面的模型失败，因为 n 始终为 1，而且数 a 往往是个非常大的数，用一个存储单元存储 a 是一个不合理的假设。这时，用其比特数表示输入规模是一个合理的模型。本书基本上用不到这个模型。

1.2.4 算法复杂度分析中的两个简化假设

因为算法复杂度的优劣取决于在 n 趋向无穷大时它增长的快慢，所以两个复杂度在 n 趋向无穷大时如相差在常数倍之内，则可以认为是同阶的函数。例如，$f(n) = 500n$ 和 $g(n) = 3n$ 是同阶无穷大。而 $f(n) = 5n$ 和 $g(n) = 5n^{1.01}$，尽管指数相差很小，却有本质不同。它们不同阶，因为在 n 趋向无穷大时 $g(n)$ 和 $f(n)$ 之比会大于任何一常数而无限增大。所以，在分析算法复杂度时，我们注重的是函数增长的阶的大小，而不计较常数因子的影响。正因为这个原因，我们在分析算法复杂度时只统计一个主要的基本运算被执行的次数就是很合理的。这是因为任一语言提供的不同的基

本运算的种类（如加、减、乘、除、比较等）是一个不大的常数 c，而一个算法中用到的不同的基本运算的种类更是它的一个小子集。假定一个算法中用到 k（$\leqslant c$）种不同运算，而算法中主要的基本运算被执行的次数是 $f(n)$，那么所有基本运算被执行的总数 $g(n)$ 满足关系 $f(n) \leqslant g(n) \leqslant kf(n)$，可见 $g(n)$ 和 $f(n)$ 是同阶的函数。为了使这两个同阶函数所对应的实际所需时间也是同阶函数，我们还需要两个假设：

1）任一基本运算所需时间是一个常数，不因被操作数大小而改变。这一点在 1.2.3 节中已提到，即运算时间与被操作数大小无关。这一假设是合理的，因为不论数字大小，计算机中二进制表示所用的比特数是一样的且硬件操作的时钟周期数也往往一样。即使有所不同，其差别也是在常数倍之内。

2）任何两个不同基本运算所需要的时间相同。例如，做一次减法与做一次乘法所需时间被认为相同。实际上会有不同，但也是在常数倍之内。

因为差别在常数因子之内的两个函数是同阶的，以上两个假设既合理又可大大简化我们的分析。它们的合理性是建立在输入规模 n 趋向无穷大的假设之上。这个假设是我们对算法的定义所加的最主要的限制或要求。我们对算法的理论分析和设计都是建立在这个假设之上的。

1.2.5 最好情况、最坏情况和平均情况的复杂度分析

同一个算法可能遇到各种不同的输入数据。即使两组输入数据有相同的规模，算法执行的时间也会大不相同，因而我们往往需要估计在遇到最有利的一组输入数据时算法所需要的时间是多少，在遇到最不利的一组输入数据时算法所需要的时间是多少，以及对各种输入情况下平均所需要的时间是多少。下面我们看一个简单例子。例 1-3 是一个线性搜索的算法。这个算法搜索数组 $A[1..n]$中的 n 个数，如发现某个数 $A[i]$（$1 \leqslant i \leqslant n$），等于要找的数 x，则报告序号 i，否则报告 0。

【例 1-3】线性搜索的算法。

```
Linear- Search (x, A[1..n])
输入：x 和数组 A[1..n]
输出：如果 A[i] = x, 1≤ i ≤ n，输出 i，否则输出 0。
1    i←1
2    while (i ⩽ n and x ≠ A[i])
3         i ← i + 1
4    endwhile
5    if i ⩽ n
6         then return (i)
7         else return (0)
8    endif
9    End
```

我们把数 x 和数组 $A[1..n]$ 之间比较的次数作为复杂度。最好的情况是 $A[1] = x$，这时算法只需要一次比较。当 x 不在数组中或者发现 $A[n] = x$，算法则需要比较 n 次，这是最坏情况。算法平均情况下的复杂度与各种情况的分布有关。假设 x 总是可以在数组中找到，并且 x 等于任一个数 $A[i]$的概率都是 $1/n$，$1 \leqslant i \leqslant n$，那么平均情况复杂度为：

$$\frac{1}{n}(1+2+\cdots+n)=\frac{1}{n}\frac{n(n+1)}{2}=\frac{n+1}{2}$$

显然，最坏情况的复杂度是最重要的和必须考虑到的，否则会因运算时间过长导致整个软件系统失败，而平均复杂度帮助我们了解长时间重复使用一个算法时可能会有的效率。最好情况的复杂度相对来讲不太重要而往往不作分析，但为了理论分析的需要，我们有时也会讨论。

1.3 函数增长渐近性态的比较

由上面讨论知，算法复杂度的优劣取决于在输入规模 n 趋向无穷大时其增长速度的快慢，即所谓的阶。这一节我们介绍评估和比较两个函数随 n 增长时渐近性态优劣的常用方法和记号，以及常见的一些复杂度函数。

1.3.1 三种比较关系及 O、Ω、Θ 记号

在比较两个复杂度函数时一般有三种情况，即等阶、高阶、低阶。然而，我们用得更多的三种关系是“不高于”、“不低于”和“等阶于”，分别用记号 O（读作大 oh）、Ω（读作大 omega）和 Θ（读作大 theta）表示。下面逐一讨论。

定义 1.1 设 $f(n)$和 $g(n)$是两个定义域为自然数的正函数（即值域为正实数的函数）。如果存在一个常数 $c>0$ 和某个自然数 n_0 使得对任一 $n \geqslant n_0$，都有关系 $f(n) \leqslant cg(n)$，我们则说 $f(n)$的阶不高于 $g(n)$的阶，并记作 $f(n)=O(g(n))$。

【例 1-4】 证明 $n^3+2n+5=O(n^3)$。

证明： 因为当 $n \geqslant 1$ 时，我们有 $n^3+2n+5 \leqslant n^3+2n^3+5n^3=8n^3$，所以 $n^3+2n+5=O(n^3)$。（取 $c=8, n_0=1$。） ■

定义 1.2 设 $f(n)$和 $g(n)$是两个定义域为自然数的正函数。如果存在一个常数 $c>0$ 和某个自然数 n_0 使得对任一 $n \geqslant n_0$，都有关系 $f(n) \geqslant cg(n)$，我们则说 $f(n)$的阶不低于 $g(n)$的阶，并记作 $f(n)=\Omega(g(n))$。

【例 1-5】 证明 $n^2=\Omega(n\lg n)$。（注意，在计算机科学领域，不明示的对数底规定为 2。）

证明： 因为当 $n \geqslant 1$ 时，我们有 $n>\lg n$，所以 $n^2=\Omega(n\lg n)$。（取 $c=1, n_0=1$。） ■

定义 1.3 设 $f(n)$和 $g(n)$是两个定义域为自然数的正函数。如果关系 $f(n)=O(g(n))$和 $f(n)=\Omega(g(n))$同时成立，我们则说 $f(n)$与 $g(n)$同阶，并记作 $f(n)=\Theta(g(n))$。

【例 1-6】 证明 $n^3+2n+5=\Theta(n^3)$。

证明： 因为例 1-4 已经证明了 $n^3+2n+5=O(n^3)$，只需证明 $n^3+2n+5=\Omega(n^3)$。我们注意到当 $n \geqslant 1$ 时，$n^3+2n+5>n^3$，所以 $n^3+2n+5=\Omega(n^3)$，这样也就证明了 $n^3+2n+5=\Theta(n^3)$。 ■

1.3.2 表示算法复杂度的常用函数

多项式函数是用于表示算法复杂度的最常见的函数之一。容易证明任一个多项式函数与其首项同阶。我们把它总结在定理 1.1 中。

定理 1.1 假设 $p(n)=a_kn^k+a_{k-1}n^{k-1}+a_{k-2}n^{k-2}+\cdots+a_1n^1+a_0$ 是一个变量为 n 的多项式，其中系数 $a_k>0$，那么 $p(n)=\Theta(n^k)$。

证明： 我们先证明 $p(n)=O(n^k)$。有以下演算：

$$\begin{aligned} p(n) &= a_kn^k+a_{k-1}n^{k-1}+a_{k-2}n^{k-2}+\cdots+a_1n^1+a_0 \\ &\leqslant a_kn^k+|a_{k-1}|n^{k-1}+|a_{k-2}|n^{k-2}+\cdots+|a_1|n^1+|a_0| \\ &\leqslant a_kn^k+|a_{k-1}|n^k+|a_{k-2}|n^k+\cdots+|a_1|n^k+|a_0|n^k \\ &\leqslant (a_k+|a_{k-1}|+|a_{k-2}|+\cdots+|a_1|+|a_0|)n^k \\ &\leqslant Cn^k \end{aligned}$$

这里，$C=(a_k+|a_{k-1}|+|a_{k-2}|+\cdots+|a_1|+|a_0|)$是一个大于零的常数，所以 $p(n)=O(n^k)$。