THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of Computer Programming

VOLUME 3
Sorting and Searching
Second Edition

计算机程序设计艺术
第 3 卷 排序和查找
(第 2 版)

[美] DONALD E. KNUTH 著

(英文影印版)

清华大学出版社

计算机程序设计艺术

[美] DONALD E. KNUTH 著

这部多卷专著是公认的对经典计算机科学最权威的描述。几十年来，无论是学生、研究人员还是编程从业人员，本套书的前三卷都是他们学习编程理论、进行编程实践的宝贵资源。

这是一套集所有基础算法之大成的经典之作，当今软件开发人员所掌握的绝大多数计算机程序设计的知识都来源于此。 ——Byte

这套书本来作为参考之用，但后来人们发现，把这套书的每一卷从头读到尾不仅是可能的，而且也是非常有意义的。一位中国的程序员甚至把他的阅读经历比做吟诗。

如果你是一名真正优秀的程序员……读 Knuth 的《计算机程序设计艺术》。如果你读懂整套书，请给我发一份简历。 ——Bill Gates

无数读者都曾谈起过 Knuth 专著对他们个人产生的巨大影响。科学家们惊讶于他精美、雅致的问题分析方式，而普通程序员则利用他提供的方案成功地解决日常工作中遇到的问题。书的恢宏、透彻、精确与幽默赢得了所有人的尊敬。

Knuth 专著伴我在学习和生活中度过了无数欢乐时光。我在车里，在餐馆里，在家里……甚至在我儿子小联赛的间隙都忘不了带上它们，一有空就捧出来阅读。 ——Charles Long

不管基础如何，只要你想认真地编写任何计算机程序，你都有必要把这套书的任何一卷抱回家，以便在你学习和工作的时候随时翻阅。

要是有一个问题难到要把《计算机程序设计艺术》请下书架，那将是一种莫大的荣幸。我发现，这套书仅仅是开卷展读，就可能会对计算机产生巨大的影响。 ——Jonathan Laventhol

为反映计算机领域的最新发展，Knuth 二十多年来第一次将三卷书全部做了修订。他的修订主要集中在自上一版以来得到众人认可的新知识，已经解决的问题，以及有所变化的问题。为保持本书的权威性，在必要的地方对计算机领域先驱工作的历史信息做了更新；为维护作者苦心孤诣追求至善至美的盛誉，新版本对读者发现的少量技术性错误做了更正；为增加本书的挑战性，作者还添加了数百道习题。

Addison Wesley

ISBN 7-302-05816-4

9 787302 058168 >

3 卷 1 套总定价：248 元（精装）

定价：85.00 元

THE CLASSIC WORK

NEWLY UPDATED AND REVISED

The Art of Computer Programming

VOLUME 3
Sorting and Searching
Second Edition

计算机程序设计艺术

第 3 卷　排序和查找

（第 2 版）

[美] DONALD E.KNUTH　著

（英文影印版）

清华大学出版社

（京）新登字 158 号

书　　名：计算机程序设计艺术　第 3 卷：排序和查找（第 2 版）
作　　者：Donald E.Knuth
出 版 者：清华大学出版社（北京清华大学学研大厦，邮编 100084）
　　　　　http://www.tup.tsinghua.edu.cn
印 刷 者：北京市耀华印刷有限公司
发 行 者：新华书店总店北京发行所
开　　本：787×1092　1/16　印张：50
版　　次：2002 年 9 月第 1 版　2002 年 9 月第 1 次印刷
印　　数：0001~3000
书　　号：ISBN 7-302-05816-4/TP・3441
定　　价：一套（三册）总定价：248 元

本套书由 3 卷组成

第 1 卷　基本算法

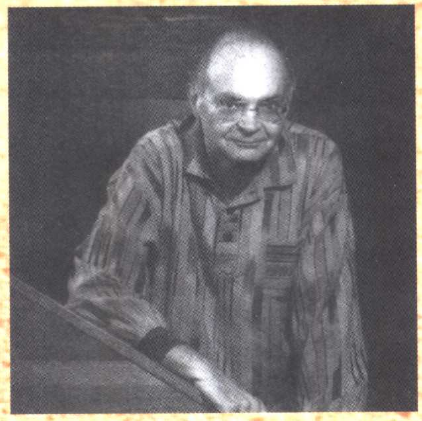
第 1 卷首先介绍编程的基本概念和技术，然后详细讲解信息结构方面的内容，包括信息在计算机内部的表示方法、数据元素之间的结构关系，以及有效的信息处理方法。此外，书中还描述了编程在模拟、数值方法、符号计算、软件与系统设计等方面的初级应用。新版本增加了数十项简单但重要的算法和技术，并根据当前研究发展趋势在数学预备知识方面做了大量修改。

第 2 卷　半数值算法

第 2 卷对半数值算法领域做了全面介绍，分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论，广泛剖析了计算机程序设计与数值分析间的相互联系。第 3 版中最引人注目的是，Knuth 对随机数生成器进行了重新处理，对形式幂级数计算作了深入讨论。

第 3 卷　排序和查找

这是对第 3 卷的头一次修订，不仅是对经典计算机排序和查找技术的最全面介绍，而且还对第 1 卷中的数据结构处理技术作了进一步的扩充，通盘考虑了大小型数据库和内外存储器。它遴选了一些经过反复检验的计算机方法，并对其效率做了定量分析。第 3 卷的突出特点是对“最优排序”一节作了修订，对排列论原理与通用散列法作了全新讨论。



Donald.E.Knuth（唐纳德.E.克努特，中文名高德纳）是算法和程序设计技术的先驱者，是计算机排版系统 $\TeX$ 和 METAFONT 的发明者，他因这些成就和大量创造性的影响深远的著作（19 部书和 160 篇论文）而誉满全球。作为斯坦福大学计算机程序设计艺术的荣誉退休教授，他当前正全神贯注于完成其关于计算机科学的史诗性的七卷集。这一伟大工程在 1962 年他还是加利福尼亚理工学院的研究生时就开始了。Knuth 教授获得了许多奖项和荣誉，包括美国计算机协会图灵奖（ACM Turing Award），美国前总统卡特授予的科学金奖（Medal of Science），美国数学学会斯蒂尔奖（AMS Steele Prize），以及 1996 年 11 月由于发明先进技术而荣获的备受推崇的京都奖（Kyoto Prize）。Knuth 教授现与其妻 Jill 生活于斯坦福校园内。

访问 Addison-Wesley 网站可以获得有关杰出的科学家和作者 Knuth 教授的更多信息：

www.aw.com/cseng/authors/knuth

访问 Knuth 教授的个人主页，可以获得有关本书及本系列其他未出版图书的更多信息：

www-cs-faculty.stanford.edu/~knuth

PREFACE

Cookery is become an art,
a noble science;
cooks are gentlemen.
— TITUS LIVIUS, *Ab Urbe Condita* XXXIX.vi
(Robert Burton, *Anatomy of Melancholy* 1.2.2.2)

THIS BOOK forms a natural sequel to the material on information structures in Chapter 2 of Volume 1, because it adds the concept of linearly ordered data to the other basic structural ideas.

The title "Sorting and Searching" may sound as if this book is only for those systems programmers who are concerned with the preparation of general-purpose sorting routines or applications to information retrieval. But in fact the area of sorting and searching provides an ideal framework for discussing a wide variety of important general issues:

- How are good algorithms discovered?
- How can given algorithms and programs be improved?
- How can the efficiency of algorithms be analyzed mathematically?
- How can a person choose rationally between different algorithms for the same task?
- In what senses can algorithms be proved "best possible"?
- How does the theory of computing interact with practical considerations?
- How can external memories like tapes, drums, or disks be used efficiently with large databases?

Indeed, I believe that virtually *every* important aspect of programming arises somewhere in the context of sorting or searching!

This volume comprises Chapters 5 and 6 of the complete series. Chapter 5 is concerned with sorting into order; this is a large subject that has been divided chiefly into two parts, internal sorting and external sorting. There also are supplementary sections, which develop auxiliary theories about permutations (Section 5.1) and about optimum techniques for sorting (Section 5.3). Chapter 6 deals with the problem of searching for specified items in tables or files; this is subdivided into methods that search sequentially, or by comparison of keys, or by digital properties, or by hashing, and then the more difficult problem of secondary key retrieval is considered. There is a surprising amount of interplay between both chapters, with strong analogies tying the topics together. Two important varieties of information structures are also discussed, in addition to those considered in Chapter 2, namely priority queues (Section 5.2.3) and linear lists represented as balanced trees (Section 6.2.3).

Like Volumes 1 and 2, this book includes a lot of material that does not appear in other publications. Many people have kindly written to me about their ideas, or spoken to me about them, and I hope that I have not distorted the material too badly when I have presented it in my own words.

I have not had time to search the patent literature systematically; indeed, I decry the current tendency to seek patents on algorithms (see Section 5.4.5). If somebody sends me a copy of a relevant patent not presently cited in this book, I will dutifully refer to it in future editions. However, I want to encourage people to continue the centuries-old mathematical tradition of putting newly discovered algorithms into the public domain. There are better ways to earn a living than to prevent other people from making use of one's contributions to computer science.

Before I retired from teaching, I used this book as a text for a student's second course in data structures, at the junior-to-graduate level, omitting most of the mathematical material. I also used the mathematical portions of this book as the basis for graduate-level courses in the analysis of algorithms, emphasizing especially Sections 5.1, 5.2.2, 6.3, and 6.4. A graduate-level course on concrete computational complexity could also be based on Sections 5.3, and 5.4.4, together with Sections 4.3.3, 4.6.3, and 4.6.4 of Volume 2.

For the most part this book is self-contained, except for occasional discussions relating to the MIX computer explained in Volume 1. Appendix B contains a summary of the mathematical notations used, some of which are a little different from those found in traditional mathematics books.

Preface to the Second Edition

This new edition matches the third editions of Volumes 1 and 2, in which I have been able to celebrate the completion of TeX and METAFONT by applying those systems to the publications they were designed for.

The conversion to electronic format has given me the opportunity to go over every word of the text and every punctuation mark. I've tried to retain the youthful exuberance of my original sentences while perhaps adding some more mature judgment. Dozens of new exercises have been added; dozens of old exercises have been given new and improved answers. Changes appear everywhere, but most significantly in Sections 5.1.4 (about permutations and tableaux), 5.3 (about optimum sorting), 5.4.9 (about disk sorting), 6.2.2 (about entropy), 6.4 (about universal hashing), and 6.5 (about multidimensional trees and tries).

The Art of Computer Programming is, however, still a work in progress. Research on sorting and searching continues to grow at a phenomenal rate. Therefore some parts of this book are headed by an "under construction" icon, to apologize for the fact that the material is not up-to-date. For example, if I were teaching an undergraduate class on data structures today, I would surely discuss randomized structures such as treaps at some length; but at present, I am only able to cite the principal papers on the subject, and to announce plans for a future Section 6.2.5 (see page 478). My files are bursting with important material that I plan to include in the final, glorious, third edition of Volume 3, perhaps 17 years from now. But I must finish Volumes 4 and 5 first, and I do not want to delay their publication any more than absolutely necessary.

I am enormously grateful to the many hundreds of people who have helped me to gather and refine this material during the past 35 years. Most of the hard work of preparing the new edition was accomplished by Phyllis Winkler (who put the text of the first edition into TEX form), by Silvio Levy (who edited it extensively and helped to prepare several dozen illustrations), and by Jeffrey Oldham (who converted more than 250 of the original illustrations to METAPOST format). The production staff at Addison–Wesley has also been extremely helpful, as usual.

I have corrected every error that alert readers detected in the first edition — as well as some mistakes that, alas, nobody noticed — and I have tried to avoid introducing new errors in the new material. However, I suppose some defects still remain, and I want to fix them as soon as possible. Therefore I will cheerfully pay $2.56 to the first finder of each technical, typographical, or historical error. The webpage cited on page iv contains a current listing of all corrections that have been reported to me.

Stanford, California D. E. K.
February 1998

There are certain common Privileges of a Writer,
the Benefit whereof, I hope, there will be no Reason to doubt;
Particularly, that where I am not understood, it shall be concluded,
that something very useful and profound is coucht underneath.
— JONATHAN SWIFT, *Tale of a Tub*, Preface (1704)

NOTES ON THE EXERCISES

THE EXERCISES in this set of books have been designed for self-study as well as classroom study. It is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying the information to specific problems and thereby being encouraged to think about what has been read. Furthermore, we all learn best the things that we have discovered for ourselves. Therefore the exercises form a major part of this work; a definite attempt has been made to keep them as informative as possible and to select problems that are enjoyable as well as instructive.

In many books, easy exercises are found mixed randomly among extremely difficult ones. This is sometimes unfortunate because readers like to know in advance how long a problem ought to take — otherwise they may just skip over all the problems. A classic example of such a situation is the book *Dynamic Programming* by Richard Bellman; this is an important, pioneering work in which a group of problems is collected together at the end of some chapters under the heading "Exercises and Research Problems," with extremely trivial questions appearing in the midst of deep, unsolved problems. It is rumored that someone once asked Dr. Bellman how to tell the exercises apart from the research problems, and he replied, "If you can solve it, it is an exercise; otherwise it's a research problem."

Good arguments can be made for including both research problems and very easy exercises in a book of this kind; therefore, to save the reader from the possible dilemma of determining which are which, *rating numbers* have been provided to indicate the level of difficulty. These numbers have the following general significance:

Rating	*Interpretation*
00	An extremely easy exercise that can be answered immediately if the material of the text has been understood; such an exercise can almost always be worked "in your head."
10	A simple problem that makes you think over the material just read, but is by no means difficult. You should be able to do this in one minute at most; pencil and paper may be useful in obtaining the solution.
20	An average problem that tests basic understanding of the text material, but you may need about fifteen or twenty minutes to answer it completely.

30 A problem of moderate difficulty and/or complexity; this one may involve more than two hours' work to solve satisfactorily, or even more if the TV is on.

40 Quite a difficult or lengthy problem that would be suitable for a term project in classroom situations. A student should be able to solve the problem in a reasonable amount of time, but the solution is not trivial.

50 A research problem that has not yet been solved satisfactorily, as far as the author knew at the time of writing, although many people have tried. If you have found an answer to such a problem, you ought to write it up for publication; furthermore, the author of this book would appreciate hearing about the solution as soon as possible (provided that it is correct).

By interpolation in this "logarithmic" scale, the significance of other rating numbers becomes clear. For example, a rating of *17* would indicate an exercise that is a bit simpler than average. Problems with a rating of *50* that are subsequently solved by some reader may appear with a *45* rating in later editions of the book, and in the errata posted on the Internet (see page iv).

The remainder of the rating number divided by 5 indicates the amount of detailed work required. Thus, an exercise rated *24* may take longer to solve than an exercise that is rated *25*, but the latter will require more creativity.

The author has tried earnestly to assign accurate rating numbers, but it is difficult for the person who makes up a problem to know just how formidable it will be for someone else to find a solution; and everyone has more aptitude for certain types of problems than for others. It is hoped that the rating numbers represent a good guess at the level of difficulty, but they should be taken as general guidelines, not as absolute indicators.

This book has been written for readers with varying degrees of mathematical training and sophistication; as a result, some of the exercises are intended only for the use of more mathematically inclined readers. The rating is preceded by an *M* if the exercise involves mathematical concepts or motivation to a greater extent than necessary for someone who is primarily interested only in programming the algorithms themselves. An exercise is marked with the letters "*HM*" if its solution necessarily involves a knowledge of calculus or other higher mathematics not developed in this book. An "*HM*" designation does *not* necessarily imply difficulty.

Some exercises are preceded by an arrowhead, "▸"; this designates problems that are especially instructive and especially recommended. Of course, no reader/student is expected to work *all* of the exercises, so those that seem to be the most valuable have been singled out. (This is not meant to detract from the other exercises!) Each reader should at least make an attempt to solve all of the problems whose rating is *10* or less; and the arrows may help to indicate which of the problems with a higher rating should be given priority.

Solutions to most of the exercises appear in the answer section. Please use them wisely; do not turn to the answer until you have made a genuine effort to solve the problem by yourself, or unless you absolutely do not have time to work this particular problem. *After* getting your own solution or giving the problem a decent try, you may find the answer instructive and helpful. The solution given will often be quite short, and it will sketch the details under the assumption that you have earnestly tried to solve it by your own means first. Sometimes the solution gives less information than was asked; often it gives more. It is quite possible that you may have a better answer than the one published here, or you may have found an error in the published solution; in such a case, the author will be pleased to know the details. Later editions of this book will give the improved solutions together with the solver's name where appropriate.

When working an exercise you may generally use the answers to previous exercises, unless specifically forbidden from doing so. The rating numbers have been assigned with this in mind; thus it is possible for exercise $n+1$ to have a lower rating than exercise n, even though it includes the result of exercise n as a special case.

Summary of codes:

		00	Immediate
		10	Simple (one minute)
		20	Medium (quarter hour)
▶	Recommended	*30*	Moderately hard
M	Mathematically oriented	*40*	Term project
HM	Requiring "higher math"	*50*	Research problem

EXERCISES

▶ **1.** [*00*] What does the rating "*M20*" mean?

2. [*10*] Of what value can the exercises in a textbook be to the reader?

3. [*HM45*] Prove that when n is an integer, $n > 2$, the equation $x^n + y^n = z^n$ has no solution in positive integers x, y, z.

Two hours' daily exercise ... will be enough to keep a hack fit for his work.

— M. H. MAHON, *The Handy Horse Book* (1865)

CONTENTS

CHAPTER FIVE

SORTING

There is nothing more difficult to take in hand,
more perilous to conduct, or more uncertain in its success,
than to take the lead in the introduction of
a new order of things.

— NICCOLÒ MACHIAVELLI, *The Prince* (1513)

"But you can't look up all those license
numbers in time," Drake objected.
"We don't have to, Paul. We merely arrange a list
and look for duplications."

— PERRY MASON, in *The Case of the Angry Mourner* (1951)

"Treesort" Computer — With this new 'computer-approach'
to nature study you can quickly identify over 260
different trees of U.S., Alaska, and Canada,
even palms, desert trees, and other exotics.
To sort, you simply insert the needle.

— Catalog of Edmund Scientific Company (1964)

IN THIS CHAPTER we shall study a topic that arises frequently in programming: the rearrangement of items into ascending or descending order. Imagine how hard it would be to use a dictionary if its words were not alphabetized! We will see that, in a similar way, the order in which items are stored in computer memory often has a profound influence on the speed and simplicity of algorithms that manipulate those items.

Although dictionaries of the English language define "sorting" as the process of separating or arranging things according to class or kind, computer programmers traditionally use the word in the much more special sense of marshaling things into ascending or descending order. The process should perhaps be called *ordering*, not sorting; but anyone who tries to call it "ordering" is soon led into confusion because of the many different meanings attached to that word. Consider the following sentence, for example: "Since only two of our tape drives were in working order, I was ordered to order more tape units in short order, in order to order the data several orders of magnitude faster." Mathematical terminology abounds with still more senses of order (the order of a group, the order of a permutation, the order of a branch point, relations of order, etc., etc.). Thus we find that the word "order" can lead to chaos.

Some people have suggested that "sequencing" would be the best name for the process of sorting into order; but this word often seems to lack the right connotation, especially when equal elements are present, and it occasionally conflicts with other terminology. It is quite true that "sorting" is itself an overused word ("I was sort of out of sorts after sorting that sort of data"), but it has become firmly established in computing parlance. Therefore we shall use the word "sorting" chiefly in the strict sense of sorting into order, without further apologies.

Some of the most important applications of sorting are:

a) *Solving the "togetherness" problem*, in which all items with the same identification are brought together. Suppose that we have 10000 items in arbitrary order, many of which have equal values; and suppose that we want to rearrange the data so that all items with equal values appear in consecutive positions. This is essentially the problem of sorting in the older sense of the word; and it can be solved easily by sorting the file in the new sense of the word, so that the values are in ascending order, $v_1 \leq v_2 \leq \cdots \leq v_{10000}$. The efficiency achievable in this procedure explains why the original meaning of "sorting" has changed.

b) *Matching items in two or more files.* If several files have been sorted into the same order, it is possible to find all of the matching entries in one sequential pass through them, without backing up. This is the principle that Perry Mason used to help solve a murder case (see the quotation at the beginning of this chapter). We can usually process a list of information most quickly by traversing it in sequence from beginning to end, instead of skipping around at random in the list, unless the entire list is small enough to fit in a high-speed random-access memory. Sorting makes it possible to use sequential accessing on large files, as a feasible substitute for direct addressing.

c) *Searching for information by key values.* Sorting is also an aid to searching, as we shall see in Chapter 6, hence it helps us make computer output more suitable for human consumption. In fact, a listing that has been sorted into alphabetic order often looks quite authoritative even when the associated numerical information has been incorrectly computed.

Although sorting has traditionally been used mostly for business data processing, it is actually a basic tool that every programmer should keep in mind for use in a wide variety of situations. We have discussed its use for simplifying algebraic formulas, in exercise 2.3.2–17. The exercises below illustrate the diversity of typical applications.

One of the first large-scale software systems to demonstrate the versatility of sorting was the LARC Scientific Compiler developed by J. Erdwinn, D. E. Ferguson, and their associates at Computer Sciences Corporation in 1960. This optimizing compiler for an extended FORTRAN language made heavy use of sorting so that the various compilation algorithms were presented with relevant parts of the source program in a convenient sequence. The first pass was a lexical scan that divided the FORTRAN source code into individual tokens, each representing an identifier or a constant or an operator, etc. Each token was assigned several sequence numbers; when sorted on the name and an appropriate sequence number, all the uses of a given identifier were brought together. The "defining entries" by which a user would specify whether an identifier stood for a function name, a parameter, or a dimensioned variable were given low sequence numbers, so that they would appear first among the tokens having a given identifier; this made it easy to check for conflicting usage and to allocate storage with respect to EQUIVALENCE declarations. The information thus gathered about each identifier was now attached to each token; in this way no "symbol table" of identifiers needed to be maintained in the high-speed memory. The updated tokens were then sorted on another sequence number, which essentially brought the source program back into its original order except that the numbering scheme was cleverly designed to put arithmetic expressions into a more convenient "Polish prefix" form. Sorting was also used in later phases of compilation, to facilitate loop optimization, to merge error messages into the listing, etc. In short, the compiler was designed so that virtually all the processing could be done sequentially from files that were stored in an auxiliary drum memory, since appropriate sequence numbers were attached to the data in such a way that it could be sorted into various convenient arrangements.

Computer manufacturers of the 1960s estimated that more than 25 percent of the running time on their computers was spent on sorting, when all their customers were taken into account. In fact, there were many installations in which the task of sorting was responsible for more than half of the computing time. From these statistics we may conclude that either (i) there are many important applications of sorting, or (ii) many people sort when they shouldn't, or (iii) inefficient sorting algorithms have been in common use. The real truth probably involves all three of these possibilities, but in any event we can see that sorting is worthy of serious study, as a practical matter.

Even if sorting were almost useless, there would be plenty of rewarding reasons for studying it anyway! The ingenious algorithms that have been discovered show that sorting is an extremely interesting topic to explore in its own right. Many fascinating unsolved problems remain in this area, as well as quite a few solved ones.

From a broader perspective we will find also that sorting algorithms make a valuable *case study* of how to attack computer programming problems in general. Many important principles of data structure manipulation will be illustrated in this chapter. We will be examining the evolution of various sorting techniques in an attempt to indicate how the ideas were discovered in the first place. By extrapolating this case study we can learn a good deal about strategies that help us design good algorithms for other computer problems.

Sorting techniques also provide excellent illustrations of the general ideas involved in the *analysis of algorithms*—the ideas used to determine performance characteristics of algorithms so that an intelligent choice can be made between competing methods. Readers who are mathematically inclined will find quite a few instructive techniques in this chapter for estimating the speed of computer algorithms and for solving complicated recurrence relations. On the other hand, the material has been arranged so that readers without a mathematical bent can safely skip over these calculations.