教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书

计算机算法

——设计与分析导论

（第三版　影印版）

COMPUTER ALGORITHMS

Introduction to Design and Analysis

(Third Edition)

Sara Baase
Allen Van Gelder

高等教育出版社
Higher Education Press
Pearson Education
出版集团

教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书

计算机算法

——设计与分析导论

（第三版 影印版）

COMPUTER ALGORITHMS

Introduction to Design and Analysis

(Third Edition)

Sara Baase
Allen Van Gelder

高等教育出版社
Pearson Education 出版集团

图字：01-2001-2170 号

图书在版编目(CIP)数据

计算机算法——设计与分析导论：第三版 /（美）巴斯（Baase，B.）等. —影印本. —北京：高等教育出版社，2001.7

ISBN 7-04-010048-7

Ⅰ.计… Ⅱ.巴… Ⅲ.①电子计算机-算法设计-高等学校-教材-英文②电子计算机-算法分析-高等学校-教材-英文 Ⅳ.TP301.6

中国版本图书馆 CIP 数据核字（2001）第 045055 号

计算机算法——设计与分析导论(第三版 影印版)

Sara Baase 等

出版发行	高等教育出版社		
社　　址	北京市东城区沙滩后街 55 号	邮政编码	100009
电　　话	010—64054588	传　　真	010—64014048
网　　址	http://www.hep.edu.cn http://www.hep.com.cn		
经　　销	新华书店北京发行所		
印　　刷	高等教育出版社印刷厂		
开　　本	787×1092　1/16	版　　次	2001 年 7 月影印版
印　　张	44.5	印　　次	2001 年 7 月第 1 次印刷
字　　数	1 066 000	定　　价	39.50 元

前　言

20世纪末，以计算机和通信技术为代表的信息科学和技术，对世界的经济、军事、科技、教育、文化、卫生等方面的发展产生了深刻的影响，由此而兴起的信息产业已经成为世界经济发展的支柱。进入21世纪，各国为了加快本国的信息产业，加大了资金投入和政策扶持。

为了加快我国信息产业的进程，在我国《国民经济和社会发展第十个五年计划纲要》中，明确提出“以信息化带动工业化，发挥后发优势，实现社会生产力的跨越式发展。”信息产业的国际竞争将日趋激烈。在我国加入WTO后，我国信息产业将面临国外竞争对手的严峻挑战。竞争成败最终将取决于信息科学和技术人才的多少与优劣。

在20世纪末，我国信息产业虽然得到迅猛发展，但与国际先进国家相比，差距还很大。为了赶上并超过国际先进水平，我国必须加快信息技术人才的培养，特别要培养一大批具有国际竞争能力的高水平的信息技术人才，促进我国信息产业和国家信息化水平的全面提高。为此，教育部高等教育司根据教育部吕福源副部长的意见，在长期重视推动高等学校信息科学和技术教学的基础上，将实施超前发展战略，采取一些重要举措，加快推动高等学校的信息科学和技术等相关专业的教学工作。在大力宣传、推荐我国专家编著的面向21世纪和“九五”重点的信息科学和技术课程教材的基础上，在有条件的高等学校的某些信息科学和技术课程中推动使用国外优秀教材的影印版进行英语或双语教学，以缩短我国在计算机教学上与国际先进水平的差距，同时也有助于强化我国大学生的英语水平。

为了达到上述目的，在分析一些出版社已影印相关教材，一些学校已试用影印教材进行教学的基础上，教育部高等教育司组织并委托高等教育出版社开展国外优秀信息科学和技术优秀教材及其教学辅助材料的引进研究与影印出版的试点工作。为推动用影印版教材进行教学创造条件。

本次引进的系列教材的影印出版工作，是在对我国高校信息科学和技术专业的课程与美国高校的进行对比分析的基础上展开的；所影印出版的教材均由我国主要高校

的信息科学和技术专家组成的专家组，从国外近两年出版的大量最新教材中精心筛选评审通过的内容新、有影响的优秀教材；影印教材的定价原则上应与我国大学教材价格相当。

教育部高等教育司将此影印系列教材推荐给高等学校，希望有关教师选用，使用后有什么意见和建议请及时反馈。也希望有条件的出版社，根据影印教材的要求，积极参加此项工作，以便引进更多、更新、更好的外国教材和教学辅助材料。

同时，感谢国外有关出版公司对此项引进工作的配合，欢迎更多的国外公司关心并参与此项工作。

教育部高等教育司

二〇〇一年四月

To Keith—always part of what I do S.B.

To Jane—for her patience A.V.G.

Preface

Purpose

This book is intended for an upper-division or graduate course in algorithms. It has sufficient material to allow several choices of topics.

The purpose of the book is threefold. It is intended to teach algorithms for solving real problems that arise frequently in computer applications, to teach basic principles and techniques of computational complexity (worst-case and average behavior, space usage, and lower bounds on the complexity of a problem), and to introduce the areas of $\mathcal{NP}$-completeness and parallel algorithms.

Another of the book's aims, which is at least as important as teaching the subject matter, is to develop in the reader the habit of always responding to a new algorithm with the questions: How good is it? Is there a better way? Therefore, instead of presenting a series of complete, "pulled-out-of-a-hat" algorithms with analysis, the text often discusses a problem first, considers one or more approaches to solving it (as a reader who sees the problem for the first time might), and then begins to develop an algorithm, analyzes it, and modifies or rejects it until a satisfactory result is produced. (Alternative approaches that are ultimately rejected are also considered in the exercises; it is useful for the reader to know why they were rejected.)

Questions such as: How can this be done more efficiently? What data structure would be useful here? Which operations should we focus on to analyze this algorithm? How must this variable (or data structure) be initialized? appear frequently throughout the text. Answers generally follow the questions, but we suggest readers pause before reading the ensuing text and think up their own answers. Learning is not a passive process.

We hope readers will also learn to be aware of how an algorithm actually behaves on various inputs—that is, Which branches are followed? What is the pattern of growth and shrinkage of stacks? How does presenting the input in different ways (e.g., listing the vertices or edges of a graph in a different order) affect the behavior? Such questions are raised in some of the exercises, but are not emphasized in the text because they require carefully going through the details of many examples.

Most of the algorithms presented are of practical use; we have chosen not to emphasize those with good asymptotic behavior that are poor for inputs of useful sizes (though some important ones are included). Specific algorithms were chosen for a variety of reasons

including the importance of the problem, illustrating analysis techniques, illustrating techniques (e.g., depth-first search) that give rise to numerous algorithms, and illustrating the development and improvement of techniques and algorithms (e.g., Union-Find programs).

Prerequisites

The book assumes familiarity with data structures such as linked lists, stacks, and trees, and prior exposure to recursion. However, we include a review, with specifications, for the standard data structures and some specialized ones. We have also added a student-friendly review of recursion.

Analysis of algorithms uses simple properties of logarithms and some calculus (differentiation to determine the asymptotic order of a function and integration to approximate summations), though virtually no calculus is used beyond Chapter 4. We find many students intimidated when they see the first log or integral sign because a year or more has passed since they had a calculus course. Readers will need only a few properties of logs and a few integrals from first-semester calculus. Section 1.3 reviews some of the necessary mathematics, and Section 1.5.4 provides a practical guide.

Algorithm Design Techniques

Several important algorithm design techniques reappear in many algorithms. These include divide-and-conquer, greedy methods, depth-first search (for graphs), and dynamic programming. This edition puts more emphasis on algorithm design techniques than did the second edition. Dynamic programming, as before, has its own chapter and depth-first search is presented with many applications in the chapter on graph traversals (Chapter 7). Most chapters are organized by application area, rather than by design technique, so we provide here a list of places where you will find algorithms using divide-and-conquer and greedy techniques.

The divide-and-conquer technique is described in Section 4.3. It is used in Binary Search (Section 1.6), most sorting methods (Chapter 4), median finding and the general selection problem (Section 5.4), binary search trees (Section 6.4), polynomial evaluation (Section 12.2), matrix multiplication (Section 12.3), the Fast Fourier Transform (Section 12.4), approximate graph coloring (Section 13.7), and, in a slightly different form, for parallel computation in Section 14.5.

Greedy algorithms are used for finding minimum spanning trees and shortest paths in Chapter 8, and for various approximation algorithms for $\mathcal{NP}$-hard optimization problems, such as bin packing, knapsack, graph coloring, and traveling salesperson (Sections 13.4 through 13.8).

Changes from the Second Edition

This edition has three new chapters and many new topics. Throughout the book, numerous sections have been extensively rewritten. A few topics from the second edition have been moved to different chapters where we think they fit better. We added more than 100 new exercises, many bibliographic entries, and an appendix with Java examples. Chapters 2, 3, and 6 are virtually all new.

Chapter 2 reviews abstract data types (ADTs) and includes specifications for several standard ADTs. The role of abstract data types in algorithm design is emphasized throughout the book.

Chapter 3 reviews recursion and induction, emphasizing the connection between the two and their usefulness in designing and proving correctness of programs. The chapter also develops recursion trees, which provide a visual and intuitive representation of recurrence equations that arise in the analysis of recursive algorithms. Solutions for commonly occurring patterns are summarized so they are available for use in later chapters.

Chapter 6 covers hashing, red-black trees for balanced binary trees, advanced priority queues, and dynamic equivalence relations (Union-Find). The latter topic was moved from a different chapter in the second edition.

We rewrote all algorithms in a Java-based pseudocode. Familiarity with Java is not required; the algorithms can be read easily by anyone familiar with C or C++. Chapter 1 has an introduction to the Java-based pseudocode.

We significantly expanded the section on mathematical tools for algorithm analysis in Chapter 1 to provide a better review and reference for some of the mathematics used in the book. The discussion of the asymptotic order of functions in Section 1.5 was designed to help students gain a better mastery of the concepts and techniques for dealing with asymptotic order. We added rules, in informal language, that summarize the most common cases (Section 1.5.4).

Chapter 4 contains an accelerated version of Heapsort in which the number of key comparisons is cut nearly in half. For Quicksort, we use the Hoare partition algorithm in the main text. Lomuto's method is introduced in an exercise. (This is reversed from the second edition.)

We split the old graph chapter into two chapters, and changed the order of some topics. Chapter 7 concentrates on (linear time) traversal algorithms. The presentation of depth-first search has been thoroughly revised to emphasize the general structure of the technique and show more applications. We added topological sorting and critical path analysis as applications and because of their intrinsic value and their connection to dynamic programming. Sharir's algorithm, rather than Tarjan's, is presented for strongly connected components.

Chapter 8 covers greedy algorithms for graph problems. The presentations of the Prim algorithm for minimum spanning trees and the Dijkstra algorithm for shortest paths were rewritten to emphasize the roles of priority queues and to illustrate how the use of abstract data types can lead the designer to efficient implementations. The asymptotically optimal $\Theta(m + n \log n)$ implementation is mentioned, but is not covered in depth. We moved Kruskal's algorithm for minimum spanning trees to this chapter.

The presentation of dynamic programming (Chapter 10) was substantially revised to emphasize a general approach to finding dynamic programming solutions. We added a new application, a text-formatting problem, to reinforce the point that not all applications call for a two-dimensional array. We moved the approximate string matching application (which was in this chapter in the second edition) to the string matching chapter (Section 11.5). The exercises include some other new applications.

Our teaching experience has pinpointed particular areas where students had difficulties with concepts related to $\mathcal{P}$ and $\mathcal{NP}$ (Chapter 13), particularly nondeterministic algorithms and polynomial transformations. We rewrote some definitions and examples to make the concepts clearer. We added a short section on approximation algorithms for the traveling salesperson problem and a section on DNA computing.

Instructors who used the second edition may particularly want to note that we changed some conventions and terminology (usually to conform to common usage). Array indexes now often begin at 0 instead of 1. (In some cases, where numbering from 1 was clearer, we left it that way.) We now use the term *depth* rather than *level* for the depth of a node in a tree. We use *height* instead of *depth* for the maximum depth of any node in a tree. In the second edition, a *path* in a graph was defined to be what is commonly called a *simple path*; we use the more general definition for *path* in this edition and define *simple path* separately. A directed graph may now contain a self-edge.

Exercises and Programs

Some exercises are somewhat open-ended. For example, one might ask for a good lower bound for the complexity of a problem, rather than asking students to show that a given function is a lower bound. We did this for two reasons. One is to make the form of the question more realistic; a solution must be discovered as well as verified. The other is that it may be hard for some students to prove the best known lower bound (or find the most efficient algorithm for a problem), but there is still a range of solutions they can offer to show their mastery of the techniques studied.

Some topics and interesting problems are introduced only in exercises. For example, the maximum independent set problem for a tree is an exercise in Chapter 3, the maximum subsequence sum problem is an exercise in Chapter 4, and the sink finding problem for a graph is an exercise in Chapter 7. Several $\mathcal{NP}$-complete problems are introduced in exercises in Chapter 13.

The abilities, background, and mathematical sophistication of students at different universities vary considerably, making it difficult to decide exactly which exercises should be marked ("starred") as "hard." We starred exercises that use more than minimal mathematics, require substantial creativity, or require a long chain of reasoning. A few exercises have two stars. Some starred exercises have hints.

The algorithms presented in this book are not programs; that is, many details not important to the method or the analysis are omitted. Of course, students should know how to implement efficient algorithms in efficient, debugged programs. Many instructors may teach this course as a pure "theory" course without programming. For those who want to assign programming projects, most chapters include a list of programming assignments. These are brief suggestions that may need amplification by instructors who choose to use them.

Selecting Topics for Your Course

Clearly the amount of material and the particular selection of topics to cover depend on the particular course and student population. We present sample outlines for two undergraduate courses and one graduate course.

This outline corresponds approximately to the senior-level course Sara Baase teaches at San Diego State University in a 15-week semester with 3 hours per week of lecture.

Chapter 1: The whole chapter is assigned as reading but I concentrate on Sections 1.4 and 1.5 in class.

Chapter 2: Sections 2.1 through 2.4 assigned as reading.

Chapter 3: Sections 3.1 through 3.4, 3.6, and 3.7 assigned as reading with light coverage in class.

Chapter 4: Sections 4.1 through 4.9.

Chapter 5: Sections 5.1 through 5.2, 5.6, and some of 5.4.

Chapter 7: Sections 7.1 through 7.4 and either 7.5 or 7.6 and 7.7.

Chapter 8: Sections 8.1 through 8.3 and brief mention of 8.4.

Chapter 11: Sections 11.1 through 11.4.

Chapter 13: Sections 13.1 through 13.5, 13.8, and 13.9.

The next outline is the junior-level course Allen Van Gelder teaches at the University of California, Santa Cruz, in a 10-week quarter with 3.5 hours per week of lecture.

Chapter 1: Sections 1.3 and 1.5, and remaining sections as reading.

Chapter 2: Sections 2.1 through 2.3, and remaining sections as reading.

Chapter 3: All sections are touched on; a lot is left for reading.

Chapter 4: Sections 4.1 through 4.9.

Chapter 5: Possibly Section 5.4, the average linear time algorithm only.

Chapter 6: Sections 6.4 through 6.6.

Chapter 7: Sections 7.1 through 7.6.

Chapter 8: The entire chapter.

Chapter 9: Sections 9.1 through 9.4.

Chapter 10: Possibly Sections 10.1 through 10.3, but usually no time.

For the first-year graduate course at the University of California, Santa Cruz (also 10 weeks, 3.5 hours of lecture), the above material is compressed and the following additional topics are covered.

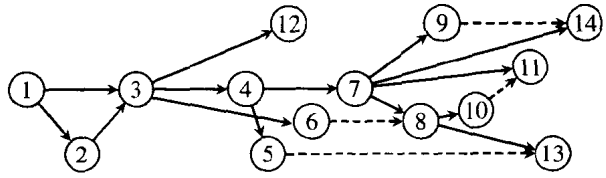
Chapter 5: The entire chapter.

Chapter 6: The remainder of the chapter, with emphasis on amortized analysis.

Chapter 10: The entire chapter.

Chapter 13: Sections 13.1 through 13.3, and possibly Section 13.9.

The primary dependencies among chapters are shown in the following diagram with solid lines; some secondary dependencies are indicated with dashed lines. A secondary dependency means that only a few topics in the earlier chapter are needed in the later chapter, or that only the more advanced sections of the later chapter require the earlier one.



While material in Chapters 2 and 6 is important to have seen, a lot of it might have been covered in an earlier course. Some sections in Chapter 6 are important for the more advanced parts of Chapter 8.

We like to remind readers of common themes or techniques, so we often refer back to earlier sections; many of these references can be ignored if the earlier sections were not covered. Several chapters have a section on lower bounds, which benefits from the ideas and examples in Chapter 5, but the diagram does not show that dependency because many instructors do not cover lower bounds.

We marked ("starred") sections that contain more complicated mathematics or more complex or sophisticated arguments than most others, but only where the material is not central to the book. We also starred one or two sections that contain optional digressions. We have not starred a few sections that we consider essential to a course for which the book is used, even though they contain a lot of mathematics. For example, at least some of the material in Section 1.5 on the asymptotic growth rate of functions and in Section 3.7 on solutions of recurrence equations should be covered.

Acknowledgments

We are happy to take this opportunity to thank the people who helped in big and small ways in the preparation of the third edition of this book.

Sara Baase acknowledges the influence and inspiration of Dick Karp, who made the subject of computational complexity exciting and beautiful in his superb lectures. Allen Van Gelder acknowledges the insights gained from Bob Floyd, Don Knuth, Ernst Mayr, Vaughan Pratt, and Jeff Ullman; they all teach more than is "in the book." Allen also wishes to acknowledge colleagues David Helmbold for many discussions on how to present algorithms effectively and on fine points of many algorithms, and Charlie McDowell for help on many of the aspects of Java that are covered in this book's appendix. We thank Lila Kari for reading an early draft of the section on DNA computing and answering our questions.

Of course, we'd have nothing to write about without the many people who did the original research that provided the material we enjoy learning and passing on to new generations of students. We thank them for their work.

In the years since the second edition appeared, several students and instructors who used the book sent in lists of errors, typos, and suggestions for changes. We don't have a complete list of names, but we appreciate the time and thought that went into their letters.

The surveys and manuscript reviews obtained by Addison-Wesley were especially helpful. Our thanks to Iliana Bjorling-Sachs (Lafayette College), Mohammad B. Dadfar (Bowling Green State University), Daniel Hirschberg (University of California at Irvine),

Mitsunori Ogihara (University of Rochester), R. W. Robinson (University of Georgia), Yaakov L. Varol (University of Nevada, Reno), William W. White (Southern Illinois University at Edwardsville), Dawn Wilkins (University of Mississippi), and Abdou Youssef (George Washington University).

We thank our editors at Addison-Wesley, Maite Suarez-Rivas and Karen Wernholm, for their confidence and patience in working with us on this project that often departed from standard production procedures and schedules. We thank Joan Flaherty for her painstakingly careful copy editing and valuable suggestions for improving the presentation. Brooke Albright's careful proofreading detected many errors that had survived earlier scrutiny; of course, any that remain are the fault of the authors.

We thank Keith Mayers for assisting us in various ways. Sara thanks him for not reminding her too often that she broke her wedding vow to work less than seven days a week.

Sara Baase, *San Diego, California*
http://www-rohan.sdsu.edu/faculty/baase

Allen Van Gelder, *Santa Cruz, California*
http://www.cse.ucsc.edu/personnel/faculty/avg.html

June, 1999

Contents