

TP31
R93

8062133



E8052133

ALGORITHMS : THEIR COMPLEXITY AND EFFICIENCY

Lydia I. Kronsjö

*The Computer Centre
The University of Birmingham*



A Wiley-Interscience Publication

JOHN WILEY & SONS

Chichester • New York • Brisbane • Toronto

Copyright © 1979 by John Wiley & Sons, Ltd.

All rights reserved.

No part of this book may be reproduced by any means, nor transmitted, nor translated into a machine language without the written permission of the publisher.

Library of Congress Cataloging in Publication Data:

Kronsjö, Lydia I.

Algorithms: their complexity and efficiency.

(Wiley series in computing)

'A Wiley-Interscience publication.'

Includes index.

1. Electronic digital computers—Programming.
2. Algorithms. 3. Numerical analysis—Data processing.
4. Computational complexity. I. Title.

QA76.6.K76

519.4

78-22090

ISBN 0 471 99752 8

Photosetting by Thomson Press (India) Limited, New Delhi
and printed in Great Britain by The Pitman Press, Bath, Avon.

**ALGORITHMS :
THEIR COMPLEXITY
AND EFFICIENCY**

WILEY SERIES IN COMPUTING

Consulting Editor

Professor D. W. Barron, *Department of Mathematics, Southampton University*

Numerical Control—Mathematics and Applications

P. Bézier

Communication Networks for Computers

D. W. Davies and D. L. A. Barber

Macro Processors and Techniques for Portable Software

P. J. Brown

A Practical Guide to Algol 68

Frank G. Pagan

Programs and Machines

Richard Bird

The Codasyl Approach to Data Base Management

T. William Olle

Data Structures and Operating Systems

Teodor Rus

Computer Networks and their Protocols

D. W. Davies, D. L. A. Barber, W. L. Price and C. M. Solomonides

Algorithms: Their Complexity and Efficiency

Lydia I. Kronsjö

To
my parents Ivan and Varvara,
my father-in-law Erik,
my husband Tom,
my son Tim.

Preface

This book is concerned with the study of algorithms and evaluation of their performance. Analysis of algorithms is a new area of research, and emerged as a new scientific subject during the sixties and has been quickly established as one of the most active fields of study, becoming an important part of computer science. The reason for this sudden interest in the study of algorithms is not difficult to trace as the fast and successful development of digital computers and their uses in many different areas of human activity has led to the construction of a great variety of computer algorithms. At present it is often the case that several different algorithms exist for the solution of a single problem or a class of problems and these algorithms need to be carefully analysed in order to provide a basis for selecting the best one for the purpose. In many cases, analysis of algorithms also leads to the revelation of completely new algorithms that are even faster than all algorithms known before. On the other hand, the study of algorithms has brought about many no less startling discoveries of certain natural problems for which all algorithms are inefficient.

The book is intended as a text for an intermediate course in computer science or computational mathematics which focuses on the basic principles and concepts of algorithms. It requires general familiarity with computers, preferably some courses on programming and introductory computer science. Each chapter is devoted to one particular class of problems and their algorithms. Chapter 1 introduces the subject, while Chapters 2, 3, 4, and 5 discuss four different classes of problems that are termed as numerical, i.e. the mathematical problems, solution of which is of a numerical nature. For such problems numerical accuracy of the computed results is of particular importance. Chapter 6 discusses the problem of (asymptotically) fast multiplication of two numbers and is based on the fast Fourier transforms and Chapters 7, 8, and 9 discuss sorting and searching, the most common non-numerical problems encountered in computing.

The exercises at the end of each chapter are used to provide examples as well as to complete or generalize some proofs.

All algorithms in the text are given in a form of a sequence of steps, each

step describing the actions to be undertaken, in natural English. This seems to be a most neutral way of introducing the algorithms. Actual implementation details, e.g. a specific programming language, various programming tricks etc., are left to an interested reader. The reference list at the end of the book contains published sources for the algorithms and theoretical results discussed in the text.

I would like to thank all the people who have critically read various portions of the manuscript and offered many helpful improvements. In particular, I would like to thank Peter Jarratt, Stuart Hollingdale, Michael Atkinson, Nelson Stevens, and Tom Axford. Sincere thanks go to Ilse Browne for her excellent typing of the manuscript.

L. I. K.

Birmingham, November 1978

Table of Notations

Written	Denotes
$a \in A$	a is contained in the set A
$x \leftarrow y + z$	x is assigned the value $y + z$
$\lceil a \rceil$	the least integer greater than or equal to a
$\lfloor b \rfloor$	the greatest integer less than or equal to b
$\ln a$	natural logarithm
$a \approx b$	a approximately equal to b
\mathbf{A}^*	conjugate transpose of the matrix \mathbf{A}
\mathbf{a}^*	conjugate transpose of the vector \mathbf{a}
\mathbf{A}^T	transpose of the real matrix \mathbf{A}
\mathbf{a}^T	transpose of the real vector \mathbf{a}
$\mu[y]$	mean value of the statistical variable y
$\sigma[y]$	standard deviation of the statistical variable y
$a * b$	a is multiplied by b
$a \equiv b$	a is equivalent to b
$x \gg y$	x is much larger than y



Contents

Preface	xiii
Table of Notations	xv
1. Introduction	1
1.1 Definition of an Algorithm	1
1.2 Measures of Efficiency	2
1.3 Algorithms and Complexity	3
1.4 Numerical Algorithms and Computational Complexity	4
1.5 Numerical Algorithms and Numerical Accuracy	5
1.6 Types of Algorithm Analysis	6
1.7 Bounds on Complexity and Models of Computation	7
2. Evaluation of Polynomials	10
2.1 Polynomial Forms and Their Evaluation Algorithms	10
2.2 Preprocessing the Coefficients	15
2.3 On Optimality of the Algorithms for Polynomial Evaluation. Optimality of the Horner Algorithm	16
2.4 The Belaga Theorems	20
2.5 Polynomials of Degree Less Than or Equal to 6	26
2.5.1 Polynomials of Degree Four	26
2.5.2 Polynomials of Degree Five	28
2.5.3 Polynomials of Degree Six	29
2.6 Accuracy of the Numerical Solution and Conditioning of the Problem	31
2.6.1 Floating-Point Number Representation and Summary of Some Basic Results on Floating-Point Arithmetic	31
2.6.2 Numerical Accuracy and Conditioning	32
2.6.3 Error Analysis of Evaluation Algorithms	35
2.7 Evaluation of the Derivatives of a Polynomial	40

2.8	Evaluation of Polynomials with Complex Argument and Complex Coefficients	43
2.9	Final Comments	45
	Exercises	45
3.	Iterative Processes	47
3.1	Definition of an Iterative Process	47
3.2	The Bisection Method	48
3.3	The Order of Convergence of an Iterative Process	49
3.4	The Newton–Raphson Method. Convergence	49
3.5	The Secant Method. Convergence	52
3.6	Method of False Position (<i>Regula Falsi</i>)	55
3.7	Formulation of the Optimality Problem	55
3.8	Iterative Methods for Finding Zeros of Functions that Change Sign in an Interval. Dekker's Algorithm	57
3.9	Optimality of Root-Finding Algorithms for Functions that Change Sign in an Interval	63
3.10	Complexity Parameters and Efficiency Measures	65
3.11	Interpolation and One-Point Iterative Methods	68
3.11.1	Direct Polynomial Interpolation	68
3.11.2	Inverse Polynomial Interpolation	71
3.11.3	Derivative Estimated Iterative Methods	74
3.12	Order of Convergence and Optimality of One-Point Iterative Methods	75
3.13	Order of Convergence of Polynomial Schemes	76
3.14	Multi-Point Iterative Methods	78
3.15	Further Examples of Multi-Point Iterative Methods	82
3.16	Optimality of Multi-Point Iterative Methods	84
3.17	Conditioning of the Root-Finding Problem and Stopping Criteria for Computations	84
3.18	The Numerical Stability of Iterative Algorithms	86
	Exercises	87
4.	Direct Methods for Solving Sets of Linear Equations	90
4.1	Gaussian Elimination	91
4.1.1	Ordinary Gaussian Elimination	91
4.1.2	Gaussian Elimination and Evaluation of a Matrix Determinant	92
4.1.3	Triangular Decomposition	95
4.1.4	Storage Requirements for Triangular Decomposition	97
4.2	Algebraic Complexity: the Number of Arithmetic Operations Involved in Gaussian Elimination	98
4.3	Matrix Inversion and its Algebraic Complexity	100
4.4	Error Analysis	101
4.4.1	Conditioning of the Problem. Backward Error Analysis and Solution of Linear Simultaneous Equations	102

4.4.2	Numerical Stability of Triangular Decomposition	103
4.5	Iterative Refinement of the Solution	109
4.6	Cholesky Decomposition of Symmetric Matrices	111
4.6.1	The Cholesky Method	111
4.6.2	Algebraic Complexity: the Number of Arithmetic Operations Required to Solve a Set of Linear Equations	113
4.6.3	Matrix Inversion	114
4.6.4	Numerical Stability	115
4.7	The Orthogonal Reduction Methods	116
4.7.1	The Householder Reduction	116
4.7.2	Algebraic Complexity: the Number of Arithmetic Operations Required to Solve a Set of Linear Equations Using the Householder Reduction	122
4.7.3	Matrix Inversion by the Householder Method and its Computational Complexity	125
4.7.4	Error Analysis of Householder Reduction	126
4.8	How the Efficiency of a Direct Method may be Increased	130
4.9	The Winograd Method	131
4.9.1	The Winograd Identity	131
4.9.2	The Winograd Matrix Multiplication Algorithm	132
4.9.3	Optimality of Winograd's Formula	132
4.9.4	Algebraic Complexity: the Number of Arithmetic Operations Required by Winograd's Method	133
4.9.5	Error Analysis of Winograd's Identity	136
4.9.6	Numerical Stability of LU Decomposition Obtained Using Winograd's Matrix Multiplication Algorithm	140
4.9.7	Cholesky Decomposition Using Winograd's Algorithm	144
4.10	The Strassen Method	144
4.10.1	Matrix Multiplication	145
4.10.2	Matrix Inversion	147
4.11	Winograd's Variant of the Strassen Algorithm	148
4.12	The Karatsuba–Makarov Method	148
4.13	Summary of the Direct Methods Studied	149
4.14	Epilogue	151
	Exercises	152
5.	The Fast Fourier Transform	155
5.1	Introduction	155
5.2	The Continuous Fourier Transform	156
5.3	The Discrete Fourier Transform	158
5.4	The Fourier Transform and Operations on Polynomials	161
5.5	The Fast Fourier Transform	163
5.5.1	The FFT Algorithm: Elementary Derivation	164
5.5.2	Matrix Form of the FFT	168
5.5.3	The FFT Algorithm for a Non-Uniform Factorization	170
5.5.4	The FFT Algorithm <i>in situ</i>	172

5.6	Optimal Factorization for the FFT Algorithm	174
5.7	The FFT Algorithm of Radix 2	177
5.7.1	General Formula and a Flowgraph of the FFT Algorithm of Radix 2	178
5.7.2	Analysis of the FFT Flowgraph	180
5.7.3	Unscrambling the Fast Fourier Transform	183
5.8	Basic Discrete Fourier Transform (DFT) Computational Algorithms for Different Types of Data	186
5.9	Round-off Errors in the Fast Fourier Transform	189
5.10	Conclusion	195
	Exercises	196
6.	Fast Multiplication of Numbers: Use of the Convolution Theorem	199
6.1	On the Minimum Computation Time of Functions	199
6.2	An Initial Reduction	200
6.3	Schönhage–Strassen Algorithm for Fast Multiplication of Integers	202
6.3.1	Generalization of the Initial Reduction	202
6.3.2	Basic Modular (or Residue) Arithmetic	204
6.3.3	The Integer Fourier Transform	207
6.3.4	Multiplication of Two Numbers Using Modular Arithmetic	208
6.3.5	Calculation of the Reduced Product Coefficients Exactly	210
6.3.6	Estimation of the Work Involved	215
	Exercises	217
7.	Internal Sorting	218
7.1	Introduction	219
7.2	Find-the-Largest Algorithm	219
7.2.1	Frequency Analysis of the Algorithm	219
7.3	Comparison Sorting by Selection	224
7.3.1	The Straight Selection Sort	224
7.3.2	Analysis of the Method	224
7.3.3	Heapsort—a Comparison Sort of Complexity $n \log_2 n$	226
7.3.4	The Number of Comparisons Required by the Heapsort	229
7.3.5	The Number of Exchanges Required by the Heapsort	230
7.4	Comparison Sorting by Exchanging	231
7.4.1	The Bubble Sort	231
7.4.2	Analysis of the Bubble Sort	232
7.4.3	Quicksort—a Comparison Sort of Complexity $n \log_2 n$, on Average	235
7.4.4	Storage Analysis of Quicksort	237
7.4.5	Average Time Required by Quicksort	239
7.5	Related Problem: Insertion of a Key in an Ordered Array	240
7.6	Optimum Comparison Sorting	241

7.6.1	Lower Bound on the Maximum Number of Comparisons	243
7.6.2	Lower Bound on the Average Number of Comparisons	246
7.7	Related Problem: Finding the k th Largest of n	247
7.7.1	Bounds on the Maximum Number of Comparisons	248
7.7.2	Upper Bound on the Average Number of Comparisons	251
	Exercises	252
8.	Large Scale Data Processing: External Sorting Using Magnetic Tapes	255
8.1	The 2-way Merge	256
8.1.1	Comparisons in the 2-Way Merge	256
8.2	Merge Sorting	259
8.2.1	Maximum Number of Comparisons Required by the Balanced 2-Way Merge	260
8.3	The Use of Merge in External Sorting	261
8.3.1	Main Characteristics Which Effect the Performance of an External Sorting Algorithm	262
8.4	The Number of Complete Passes Required by the Balanced P -Way Merge	267
8.5	Polyphase Merge and Perfect Fibonacci Distributions	269
8.5.1	The Number of Complete Passes Required	273
8.6	The Cascade Merge	278
8.6.1	The Number of Complete Passes Required	280
8.7	The Oscillating Sort	281
8.7.1	The Number of Complete Passes Required	282
8.8	Generation of the Initial Subfiles	282
8.9	Merge Trees and Optimum Merge Sorting	283
	Exercises	287
9.	Searching	289
9.1	Introduction	289
9.2	Classification of Search Algorithms	291
9.3	Ordered Tables	293
9.3.1	Algorithm for Sequential Search	293
9.3.2	Algorithm for Binary Search	293
9.3.3	Algorithm for Fibonaccian Search	293
9.4	Search Trees	294
9.4.1	Binary Tree Search Methods and Data Structures	294
9.4.2	Search Methods for Unequal Distribution Tables	297
9.4.3	Optimum Cost Trees	298
9.4.4	Algorithm for Computing and Construction of Minimum-Cost Binary Tree	300
9.5	A Tree Search Followed by Insertion or Deletion of the Key	303
9.5.1	A Tree Search and Insertion Algorithm	304
9.5.2	A Tree Search and Deletion Algorithm	306
9.6	Methods for Rebalancing the Search Trees	307

9.6.1	The Balanced Trees Method of Adelson–Velski and Landis (The AVL Trees)	308
9.7	Hashing	309
9.8	Computing the Initial Hashing Function	310
9.8.1	Distribution-Dependent Hashing Functions	310
9.8.2	Cluster-Separating Hashing Functions	311
9.8.3	Distribution-Independent Hashing Functions	313
9.8.4	A Multiplicative Hashing Function	314
9.9	Collision Resolution by Open Addressing	316
9.9.1	The Pile-Up and Secondary Clustering Phenomena	316
9.9.2	Open Addressing Algorithms	317
9.10	Efficiency Analysis of the Open Addressing Algorithms	321
9.10.1	Analysis of Linear Probing	323
9.10.2	Analysis of the Uniform Hashing Model and Optimality Considerations	328
	Exercises	331
Appendix A.	Some Basic Results on the Error Analysis of the Floating-Point Matrix Multiplication and the Solution of Sets of Linear Equations	334
Appendix B.	Some Basic Preliminaries on Laws of Probability and Statistical Analysis	341
	Bibliography and References	344
	Index	355

8062133



Chapter 1

Introduction

This book will be chiefly concerned with an investigation of algorithms in order to evaluate their performance. This comparatively new field of study is known as algorithmic analysis and forms part of the more general discipline of computer science. In practical terms, a goal of algorithmic analysis is 'to obtain sufficient understanding about the relative merits of complicated algorithms to be able to provide useful advice to someone undertaking an actual computation' (Gentleman, 1973). In broader interpretation, however, algorithmic analysis includes the study of all aspects of performance in computational problem solving, from the preliminary formulation, through the programming stages, to the final task of interpreting the results obtained. We would also like to prove lower bounds on the computation time of various classes of algorithms. In order to show that there is no algorithm to perform a given task in less than a certain amount of time, we need a precise definition of what constitutes an algorithm. First, then, what is an algorithm?

1.1 Definition of an Algorithm

A procedure consisting of a finite set of unambiguous rules which specify a finite sequence of operations that provides the solution to a problem, or to a specific class of problems, is called an algorithm.

Several important features of this definition must now be emphasized.

First, each step of an algorithm must be unambiguous and precisely defined. The actions to be carried out must be rigorously specified for each case.

Secondly, an algorithm must always arrive at a problem solution after a finite number of steps. Indeed, the general restriction of finiteness is not sufficient in practice, as the number of steps needed to solve a specific problem, although finite, may be too large for practicable computation. A useful algorithm must require not only a finite number of steps, but a reasonable number.

Thirdly, every meaningful algorithm possesses zero or more inputs and provides one or more outputs. The inputs may be defined as quantities which are

given to the algorithm initially, before it is executed, and the outputs as quantities which have a specified relation to the inputs and which are delivered at the completion of its execution.

Fourthly, it is preferable that the algorithm should be applicable to any member of a class of problems rather than only to a single problem. This property of generality, though not a necessity, is certainly a desirable attribute of a useful algorithm.

Finally, we would like to mention that although the concept of an algorithm is a very broad one, in this book we restrict ourselves to algorithms designed to be executed on a computer. Such an algorithm must be embodied in a computer program (or set of programs), and so in the sequel the two terms will be used interchangeably.

1.2 Measures of Efficiency

It is relatively easy to invent algorithms. In practice, however, one wants not only algorithms, one wants good algorithms. Thus, the objective is to invent good algorithms and prove that they are good. The 'goodness' of an algorithm can be appraised by a variety of criteria. One of the most important is the time taken to execute it. There are several aspects of such a time criterion. One might be concerned with the execution time required by different algorithms for solution of a particular problem on a particular computer. However, such an empirical measure is strongly dependent upon both the program and the machine used to implement the algorithm. Thus, a change in a program may not represent a significant change in the underlying algorithm but may, nevertheless, affect the speed of execution. Furthermore, if two programs are compared first on one machine and then another, the comparisons may lead to different conclusions. Thus, while comparison of actual programs running on real computers is an important source of information, the results are inevitably affected by programming skill and machine characteristics.

A useful alternative to such empirical measurements is a mathematical analysis of the intrinsic difficulty of solving a problem computationally. Judiciously used, such an analysis provides an important means of evaluation the cost of algorithm execution.

The performance time of an algorithm is a function of the size of the computational problem to be solved. However, assuming we have a computer program which eventually terminates, solving a particular problem requires only sufficient time and sufficient storage. Of more general interest are algorithms which can be applied to a collection of problems of a certain type. For these algorithms, the time and storage space required by a program will vary with the particular problem being solved. Consider, for example, the following classes of problems, and note the role of the value of the parameter n .

1. Find the largest in a sequence of n integers.
2. Solve a set of linear algebraic equations $\mathbf{Ax} = \mathbf{b}$, where \mathbf{A} is an $n \times n$ real matrix and \mathbf{b} is a real vector of length n .