

Analysis of Algorithms: An Active Learning Approach

Jeffrey J. McConnell
Canisius College



JONES AND BARTLETT PUBLISHERS

Sudbury, Massachusetts

BOSTON TORONTO LONDON SINGAPORE

The two major goals of this book are to raise awareness of the impact that algorithms can have on the efficiency of a program and to develop the skills necessary to analyze any algorithms that are used in programs. In looking at many commercial products today, it appears that some software designers are unconcerned about space and time efficiency. If a program takes too much space, they expect that the user will buy more memory. If a program takes too long, they expect that the user will buy a faster computer.

There are limits, however, on how fast computers can ever become because there are limits on how fast electrons can travel down "wires," how fast light can travel along fiber optic cables, and how fast the circuits that do the calculations can switch. There are other limits on computation that go beyond the speed of the computer and are directly related to the complexity of the problems being solved. There are some problems for which the fastest algorithm known will not complete execution in our lifetime. Since these are important problems, algorithms are needed that provide approximate answers.

In the early 1980s, computer architecture severely limited the amount of speed and space on a computer. Some computers of that time frequently limited programs and their data to 64K of memory, where today's personal computers regularly come equipped with more than 1,000 times that amount. Though today's software is much more complex than that in the 1980s and today's computers are much more capable, these changes do not mean we can ignore efficiency in our program design. Some project specifications will include time and space limitations on the final software that may force programmers to look for places to save memory and increase speed. The com-

pact size of personal digital assistants (PDAs) also limits the size and speed of software.

Pedagogy

What I hear, I forget.

What I see, I remember.

What I do, I understand.

—*Confucius*

The material in this book is presented with the expectation that it can be read independently or used as part of a course that incorporates an active and cooperative learning methodology. To accomplish this, the chapters are clear and complete so as to be easy to understand and to encourage readers to prepare by reading before group meetings. All chapters include study suggestions. Many include additional data sets that the reader can use to hand-execute the algorithms for increased understanding of them. The results of the algorithms applied to this additional data are presented in Appendix C. Each section has a number of exercises that include simple tracing of the algorithm to more complex proof-based exercises. The reader should be able to work the exercises in each chapter. They can, in connection with a course, be assigned as homework or can be used as in-class assignments for students to work individually or in small groups. An instructor's manual that provides background on how to teach this material using active and cooperative learning as well as giving exercise solutions is available. Chapters 2, 3, 5, 6, and 9 include programming exercises. These programming projects encourage readers to implement and test the algorithms from the chapter, and then compare actual algorithm results with the theoretical analysis in the book.

Active learning is based on the premise that people learn better and retain information longer when they are participants in the learning process. To achieve that, students must be given the opportunity to do more than just listen to the professor during class. This can best be accomplished in an algorithms course by the professor giving a short introductory lecture on the material, and then having students work problems while the instructor circulates around the room answering questions that this application of the material raises.

Cooperative work gives students an opportunity to answer simple questions that others in their group have and allows the professor to deal with bigger questions that have stumped an entire group. In this way, students have a greater opportunity to ask questions and have their concerns addressed in a timely manner. It is important that the professor observe group work to make sure that group-wide misconceptions are not reinforced. An additional way for the professor to identify and correct misunderstandings is to have groups regularly submit exercise answers for comments or grading.

To support student preparation and learning, each chapter includes the prerequisites needed, and the goals or skills that students should have on completion, as well as suggestions for studying the material.

Algorithms

Since the analysis of algorithms is independent of the computer or programming language used, algorithms are given in pseudo-code. These algorithms are readily understandable by anyone who knows the concepts of conditional statements (for example, IF and CASE/SWITCH), loops (for example, FOR and WHILE), and recursion.

Course Use

One way that this material could be covered in a one-semester course is by using the following approximate schedule:

Chapter 1	2 weeks
Chapter 2	1 week
Chapter 3	2 weeks
Chapter 4	1 week
Chapter 5	1 week
Chapter 6	2 weeks
Chapter 7	2 weeks
Chapter 8	1 week
Chapter 9	2 weeks

Chapters 2, 4, and 5 are not likely to need a full week, which will provide time for an introduction to the course, an explanation of the active and cooperative learning pedagogy, and hour examinations. Depending on the background of the students, Chapter 1 may be covered more quickly as well.

Acknowledgements

I would like to acknowledge all those who helped in the development of this book. First, I would like to thank the students in my “Automata and Algorithms” course (Spring 1997, Spring 1998, Spring 1999, Spring 2000, and Fall 2000) for all of their comments on earlier versions of this material.

The reviews that Jones and Bartlett Publishers obtained about the manuscript were most helpful and produced some good additions and clarifications. I would like to thank Douglas Campbell (Brigham Young University), Nancy Kinnersley (University of Kansas), and Kirk Pruhs (University of Pittsburgh) for their reviews.

At Jones and Bartlett, I would like to thank my editors Amy Rose and Michael Stranz, and production assistant Tara McCormick for their support of this book. I am especially grateful to Amy for remembering a brief conversation about this project from a few years ago. Her memory and efforts are appreciated very much. I would also like to thank Nancy Young for her copy-editing and Brooke Albright for her proofreading. Any errors that remain are solely the author’s responsibility.

Lastly, I am grateful to Fred Dansereau for his support and suggestions during the many stages of this book, and to Barney for the wonderful diversions that only a dog can provide.

Preface v

Chapter 1

Analysis Basics 1

- 1.1 **What is Analysis?** 3
 - 1.1.1 Input Classes 7
 - 1.1.2 Space Complexity 9
 - 1.1.3 Exercises 10
- 1.2 **What to Count and Consider** 10
 - 1.2.1 Cases to Consider 11
 - Best Case 11
 - Worst Case 12
 - Average Case 12
 - 1.2.2 Exercises 13
- 1.3 **Mathematical Background** 13
 - 1.3.1 Logarithms 14
 - 1.3.2 Binary Trees 15
 - 1.3.3 Probabilities 15
 - 1.3.4 Summations 16
 - 1.3.5 Exercises 18
- 1.4 **Rates of Growth** 20
 - 1.4.1 Classification of Growth 22
 - Big Omega 22
 - Big Oh 22
 - Big Theta 23
 - Finding Big Oh 23
 - Notation 23
 - 1.4.2 Exercises 23
- 1.5 **Divide and Conquer Algorithms** 24
 - Recursive Algorithm Efficiency 26
 - 1.5.1 Tournament Method 27
 - 1.5.2 Lower Bounds 28
 - 1.5.3 Exercises 31

- 1.6 **Recurrence Relations** 32
 - 1.6.1 Exercises 37
- 1.7 **Analyzing Programs** 38

Chapter 2 Searching and Selection Algorithms 41

- 2.1 **Sequential Search** 43
 - 2.1.1 Worst-Case Analysis 44
 - 2.1.2 Average-Case Analysis 44
 - 2.1.3 Exercises 46
- 2.2 **Binary Search** 46
 - 2.2.1 Worst-Case Analysis 48
 - 2.2.2 Average-Case Analysis 49
 - 2.2.3 Exercises 52
- 2.3 **Selection** 53
 - 2.3.1 Exercises 55
- 2.4 **Programming Exercise** 55

Chapter 3 Sorting Algorithms 57

- 3.1 **Insertion Sort** 59
 - 3.1.1 Worst-Case Analysis 60
 - 3.1.2 Average-Case Analysis 60
 - 3.1.3 Exercises 62
- 3.2 **Bubble Sort** 63
 - 3.2.1 Best-Case Analysis 64
 - 3.2.2 Worst-Case Analysis 64
 - 3.2.3 Average-Case Analysis 65
 - 3.2.4 Exercises 67
- 3.3 **Shellsort** 68
 - 3.3.1 Algorithm Analysis 70
 - 3.3.2 The Effect of the Increment 71
 - 3.3.3 Exercises 72
- 3.4 **Radix Sort** 73
 - 3.4.1 Analysis 74
 - 3.4.2 Exercises 76
- 3.5 **Heapsort** 77
 - FixHeap 78
 - Constructing the Heap 79
 - Final Algorithm 80

3.5.1	Worst-Case Analysis	80
3.5.2	Average-Case Analysis	82
3.5.3	Exercises	83
3.6	Merge Sort	83
3.6.1	MergeLists Analysis	85
3.6.2	MergeSort Analysis	86
3.6.3	Exercises	88
3.7	Quicksort	89
	Splitting the List	90
3.7.1	Worst-Case Analysis	91
3.7.2	Average-Case Analysis	91
3.7.3	Exercises	94
3.8	External Polyphase Merge Sort	95
3.8.1	Number of Comparisons in Run Construction	99
3.8.2	Number of Comparisons in Run Merge	99
3.8.3	Number of Block Reads	100
3.8.4	Exercises	100
3.9	Additional Exercises	100
3.10	Programming Exercises	102

Chapter 4 **Numeric Algorithms 105**

4.1	Calculating Polynomials	107
4.1.1	Horner's Method	108
4.1.2	Preprocessed Coefficients	108
4.1.3	Exercises	111
4.2	Matrix Multiplication	112
4.2.1	Winograd's Matrix Multiplication	113
	Analysis of Winograd's Algorithm	114
4.2.2	Strassen's Matrix Multiplication	115
4.2.3	Exercises	116
4.3	Linear Equations	116
4.3.1	Gauss-Jordan Method	117
4.3.2	Exercises	119

Chapter 5 **Matching Algorithms 121**

5.1	String Matching	122
5.1.1	Finite Automata	124
5.1.2	Knuth-Morris-Pratt Algorithm	125
	Analysis of Knuth-Morris-Pratt	127

- 5.1.3 Boyer-Moore Algorithm 128
 - The Match Algorithm 129
 - The Slide Array 130
 - The Jump Array 132
 - Analysis 135
- 5.1.4 Exercises 135
- 5.2 Approximate String Matching 136**
 - 5.2.1 Analysis 138
 - 5.2.2 Exercises 139
- 5.3 Programming Exercises 139**

Chapter 6 Graph Algorithms 141

- 6.1 Graph Background and Terminology 144**
 - 6.1.1 Terminology 145
 - 6.1.2 Exercises 146
- 6.2 Data Structure Methods for Graphs 147**
 - 6.2.1 The Adjacency Matrix 148
 - 6.2.2 The Adjacency List 149
 - 6.2.3 Exercises 149
- 6.3 Depth-First and Breadth-First Traversal Algorithms 150**
 - 6.3.1 Depth-First Traversal 150
 - 6.3.2 Breadth-First Traversal 151
 - 6.3.3 Traversal Analysis 153
 - 6.3.4 Exercises 154
- 6.4 Minimum Spanning Tree Algorithm 155**
 - 6.4.1 The Dijkstra-Prim Algorithm 155
 - 6.4.2 The Kruskal Algorithm 159
 - 6.4.3 Exercises 162
- 6.5 Shortest-Path Algorithm 163**
 - 6.5.1 Dijkstra's Algorithm 164
 - 6.5.2 Exercises 167
- 6.6 Biconnected Component Algorithm 168**
 - 6.6.1 Exercises 171
- 6.7 Partitioning Sets 172**
- 6.8 Programming Exercises 174**

Chapter 7**Parallel Algorithms 177**

- 7.1 Parallelism Introduction 178**
 - 7.1.1 Computer System Categories 179
 - Single Instruction Single Data 179
 - Single Instruction Multiple Data 179
 - Multiple Instruction Single Data 179
 - Multiple Instruction Multiple Data 180
 - 7.1.2 Parallel Architectures 180
 - Loosely versus Tightly Coupled Machines 180
 - Processor Communication 181
 - 7.1.3 Principles for Parallelism Analysis 182
 - 7.1.4 Exercises 183
- 7.2 The PRAM Model 183**
 - 7.2.1 Exercises 185
- 7.3 Simple Parallel Operations 185**
 - 7.3.1 Broadcasting Data in a CREW PRAM Model 186
 - 7.3.2 Broadcasting Data in a EREW PRAM Model 186
 - 7.3.3 Finding the Maximum Value in a List 187
 - 7.3.4 Exercises 189
- 7.4 Parallel Searching 189**
 - 7.4.1 Exercises 191
- 7.5 Parallel Sorting 191**
 - 7.5.1 Linear Network Sort 191
 - 7.5.2 Odd-Even Swap Sort 195
 - 7.5.3 Other Parallel Sorts 196
 - 7.5.4 Exercises 197
- 7.6 Parallel Numerical Algorithms 198**
 - 7.6.1 Matrix Multiplication on a Parallel Mesh 198
 - Analysis 203
 - 7.6.2 Matrix Multiplication in a CRCW PRAM Model 204
 - Analysis 204
 - 7.6.3 Solving Systems of Linear Equations with an SIMD Algorithm 205
 - 7.6.4 Exercises 206
- 7.7 Parallel Graph Algorithms 207**
 - 7.7.1 Shortest-Path Parallel Algorithm 207
 - 7.7.2 Minimum Spanning Tree Parallel Algorithm 209
 - Analysis 210
 - 7.7.3 Exercises 211

Chapter 8 Nondeterministic Algorithms 213

- 8.1 **What is NP? 214**
 - 8.1.1 Problem Reductions 217
 - 8.1.2 NP-Complete Problems 218
- 8.2 **Typical NP Problems 220**
 - 8.2.1 Graph Coloring 220
 - 8.2.2 Bin Packing 221
 - 8.2.3 Backpack Problem 222
 - 8.2.4 Subset Sum Problem 222
 - 8.2.5 CNF-Satisfiability Problem 222
 - 8.2.6 Job Scheduling Problem 223
 - 8.2.7 Exercises 223
- 8.3 **What Makes Something NP? 224**
 - 8.3.1 Is $P = NP$? 225
 - 8.3.2 Exercises 226
- 8.4 **Testing Possible Solutions 226**
 - 8.4.1 Job Scheduling 227
 - 8.4.2 Graph Coloring 228
 - 8.4.3 Exercises 229

Chapter 9 Other Algorithmic Techniques 231

- 9.1 **Greedy Approximation Algorithms 232**
 - 9.1.1 Traveling Salesperson Approximations 233
 - 9.1.2 Bin Packing Approximations 235
 - 9.1.3 Backpack Approximation 236
 - 9.1.4 Subset Sum Approximation 236
 - 9.1.5 Graph Coloring Approximation 238
 - 9.1.6 Exercises 239
- 9.2 **Probabilistic Algorithms 240**
 - 9.2.1 Numerical Probabilistic Algorithms 240
 - Buffon's Needle 241
 - Monte Carlo Integration 242
 - Probabilistic Counting 243
 - 9.2.2 Monte Carlo Algorithms 244
 - Majority Element 245
 - Monte Carlo Prime Testing 246
 - 9.2.3 Las Vegas Algorithms 246
 - 9.2.4 Sherwood Algorithms 249

9.2.5	Probabilistic Algorithm Comparison	250
9.2.6	Exercises	251
9.3	Dynamic Programming	252
9.3.1	Array-Based Methods	252
9.3.2	Dynamic Matrix Multiplication	255
9.3.3	Exercises	257
9.4	Programming Exercises	258
Appendix A	Random Number Table	259
Appendix B	Pseudorandom Number Generation	261
Appendix C	Results of Chapter Study Suggestion	265
Appendix D	References	279
Index		285