

COURSEBOOK for COMPSCI.220FT
Algorithms and Data Structures

A/Prof. Georgy Gimel'farb
Dr. Mark Wilson
Mr. Andrew MacGibbon

Department of Computer Science
Tamaki Campus
University of Auckland

2002

Contents

1	ALGORITHM ANALYSIS	15
1.1	Basics of Algorithm Analysis	15
1.1.1	Some informal definitions	15
1.1.2	Running time for simple loops and other operations	25
1.1.3	Tools for analysing time complexity	28
1.1.4	Worst-case and average-case performance	37
1.1.5	Algorithm analysis: possibilities and limitations	43
1.2	Efficiency of Sorting	44
1.2.1	Order and ordering relations.	45
1.2.2	Insertion sorting	47
1.2.3	Algorithm Shellsort	51
1.2.4	Algorithm Mergesort	54
1.2.5	Algorithm Quicksort	59
1.2.6	Algorithm Heapsort	66
1.2.7	Data selection	75
1.2.8	Lower complexity bound for sorting	77
1.3	Efficiency of Searching	81
1.3.1	Sequential search	82
1.3.2	Binary search	83
1.3.3	Binary search tree	92
1.3.4	Performance analysis of binary search tree operations	98
1.3.5	Balanced trees	102
1.3.6	List ADT in Java	107
1.4	Array-Based Symbol Tables	124
1.4.1	Table ADT implementations	125
1.4.2	Hashing	126
1.4.3	Choosing a hash function	135
1.4.4	Efficiency of search in hash tables	138
1.5	Exercises	143
1.5.1	Solutions	144

2	INTRODUCTION TO GRAPH ALGORITHMS	151
2.1	The Graph Abstract Data Type	151
2.1.1	Basic definitions	151
2.1.2	Computer representations of graphs	154
2.1.3	Abstract data types and graphs	155
2.2	Graph Searching	156
2.2.1	Properties of BFS and DFS	160
2.2.2	Connectivity algorithms	161
2.3	Cycles in Graphs	163
2.3.1	Acyclic graphs and DAGs	164
2.3.2	Computing the girth of a graph	165
2.4	Topological Sorting	165
2.4.1	Zero in-degree sorting algorithm	167
2.4.2	Depth-first search sorting algorithm	168
2.5	Computing Distances and Diameter	168
2.5.1	Dijkstra's algorithm	170
2.5.2	Floyd's algorithm	172
2.6	Final Remarks	174
3	INTRODUCTION TO AUTOMATA THEORY	175
3.1	Deterministic Finite-State Machines	175
3.1.1	The cheap engineer's elevators	175
3.1.2	Finite-state machines that accept/reject strings	176
3.1.3	Recognizing patterns with DFA	178
3.2	Nondeterministic Finite-State Machines	179
3.2.1	Using nondeterministic automata	180
3.2.2	The reverse $R(L)$ of a language L	181
3.2.3	The closure $C(L)$ of a language L	182
3.3	Recognition Capabilities of NFA's and DFA's	182
3.4	Regular Expressions	184
3.4.1	The UNIX extensions to regular expressions	185
3.5	Regular Sets and Finite-State Automata	186
3.6	Minimizing Deterministic Finite-State Machines	188
4	CONTEXT-FREE GRAMMARS AND PARSING	193
4.1	A Grammar for Arithmetic Expressions	193
4.2	A Grammar for Java Statements	195
4.2.1	Exercises	196
4.3	Parse Trees	196
4.4	Ambiguity and the Design of Grammars	198
4.5	An Unambiguous Grammar for Expressions	199

4.5.1	Exercises	201
4.6	Constructing Parse Trees	201
4.6.1	Limitations of recursive descent	203
4.6.2	Exercises	203
4.7	Tokenizing the Input Stream	204
5	JAVA COLLECTIONS FRAMEWORK	207
5.1	Generic Programming	207
5.1.1	Interfaces and implementations	208
5.1.2	Polymorphic algorithms	208
5.1.3	Benefits	208
5.2	The Collections Framework	208
5.2.1	What is a collection?	209
5.2.2	The Framework interfaces	210
5.3	Interface Contracts	210
5.4	The Interfaces in Detail	211
5.4.1	Collection	211
5.4.2	Iterators	212
5.4.3	Implementing a Collection	213
5.4.4	Set	213
5.4.5	List	213
5.4.6	ListIterators	214
5.4.7	Adapter classes	214
5.4.8	Map	215
5.5	Sorting	215
5.5.1	Natural ordering	215
5.5.2	Unnatural ordering	216
5.5.3	SortedSet and SortedMap	216
5.6	The Collections Class	217
5.7	General Purpose Implementations	218
5.8	Writing Your Own Collection Classes	218
5.8.1	The data structure	218
5.8.2	get() and size()	219
5.8.3	Modifiable collections and efficiency	219
A	ABSTRACT DATA TYPES IN JAVA	221
A.1	Abstract Data Types	222
A.2	ADTs and Java classes	227

B	JAVA GRAPH ADT	233
B.1	Java adjacency matrix implementation	236
B.2	Java adjacency lists implementation	241
B.3	Standardized Java graph class	245
B.4	Graph algorithms	247

List of Figures

1.1	Time-space boundaries.	21
1.2	Telescoping as a recursive substitution.	40
1.3	Simple insertion sorting.	47
1.4	Simple insertion sorting giving a standard run-time exception.	49
1.5	Shellsort implementation with divide a gap by 2.2 which gives excellent performance in practice.	52
1.6	Linear-time merging of two sorted arrays.	55
1.7	Basic Mergesort routines	57
1.8	Merging two sorted halves of a subarray.	58
1.9	Quicksort: median-of-three, partitioning, cutoff of small arrays	64
1.10	Methods used in the basic Quicksort routine in Figure 1.9.	65
1.11	A complete binary tree and its array representation.	67
1.12	A binary heap and its array representation.	68
1.13	Basic Heapsort routines.	73
1.14	Quickselect: median-of-three, partitioning, cutoff of small arrays	76
1.15	Decision tree for $n = 3$	78
1.16	Sequential search.	82
1.17	Binary search using three-way comparisons.	84
1.18	Binary search for the integer key 42.	85
1.19	Binary tree representation of the sorted array using the binary search.	87
1.20	Faster binary search using two-way comparisons.	90
1.21	Binary trees: only the leftmost tree is a binary search tree.	93
1.22	Examples of <code>find</code> and <code>insert</code> operations in the binary search tree.	94
1.23	Binary search tree with depth about $\log n$	95
1.24	Binary search trees with depth about n	96
1.25	Removal of the node with key 10 from the given binary search tree.	97
1.26	Binary search tree with order statistics.	99
1.27	Binary search trees obtained by inserting permutations of 1, 2, 3, 4 (<i>a</i>).	99
1.28	Binary search trees obtained by inserting permutations of 1, 2, 3, 4 (<i>b</i>).	100
1.29	Binary search trees obtained by inserting permutations of 1, 2, 3, 4 (<i>c</i>).	100
1.30	Multiway search tree of order $m = 4$	105

1.31	B-tree of order $m = 4$ with storage size $l = 7$: 2..4 children per node and 4..7 data items per leaf.	106
1.32	Simple singly linked (SL) and doubly linked (DL) lists.	109
1.33	Insertion of a new element.	113
1.34	Advancing an iterator.	118
1.35	Open addressing with linear probing.	128
1.36	Open addressing with double hashing.	133
A.1	Graph $G = [V, E]$ with the four vertices and three edges.	223
A.2	Levels of data abstraction.	228
A.3	The Collections Framework hierarchy in Java 1.2. Solid lines show relationships between the interfaces and the concrete classes that implement the interfaces and dot-dashed lines show relationships between the abstract classes and the concrete classes extending the abstract ones.	229
A.4	Historical classes in the Collections Framework.	230

List of Tables

1.1	Typical algorithmic operations and operands (note that in Java the logical operation AND , or \wedge , is denoted <code>&&</code> , the logical OR , or \vee , is <code> </code> , and the logical NOT , or \neg , is denoted <code>!</code>).	16
1.2	Input data size for different algorithms.	29
1.3	Relative growth of time complexity as input size increases from $n = 5$ to $n = 625$.	33
1.4	The largest sizes n of input data that can be processed by an algorithm with running time $O(f(n))$ for various $f(n)$, assuming that when $n = 10$, the algorithm takes one minute to run.	33
1.5	Example of the insertion sort (C denotes a number of comparisons and M is the number of data movements in the inner while-loop; previously sorted elements are in italic, a box shows the position where the newly sorted element is placed, and the element to be sorted next is boldfaced).	48
1.6	Number of inversions for an array element with respect to all preceding elements	50
1.7	Example of Shellsort (lines for $i = \text{gap}, \dots, n$ show how the elements are compared and sorted under a fixed gap; the swapped elements are boldfaced).	53
1.8	Example of pivot positioning in Quicksort ($\text{low} = 0$, $\text{middle} = 4$, and $\text{high} = 9$; the pivot $p = 65$ is boldfaced.)	62
1.9	Successive steps of Heapsort (the keys moved are shown in italic and the items sorted are boldfaced).	74
1.10	Average running times of various operations using array and singly-linked list representations.	122
1.11	Running times of priority queue operations for three representations.	123
1.12	Airport codes and names.	125
1.13	Values of $P_{365}(n)$ for various n .	131
1.14	Theoretical and measured experimental results for various load factors λ (in a pair t/m below, the numbers t and m give the theoretical and measured value, respectively). The abbreviations used are: <i>SC</i> – separate chaining, <i>OA-LP</i> – open addressing with linear probing, and <i>OA-DH</i> – open addressing with double hashing.	140

1.15 Comparative performance of Table ADT representations. 141

Introduction

Chapter 1.1 prepared by Dr. Georgy Gimel'farb covers basic practical and theoretical aspects of algorithm analysis, as applied to searching and sorting algorithms you started to study in the COMPSCI.105 course. Also it overviews array-based symbol tables. Techniques are presented for measuring program performance, in particular, the Big-Oh notation for expressing efficiency of a program, basic recurrences to estimate running times, and similar tools. Performance of basic sorting and searching algorithms is analysed in detail with respect to most popular data types such as lists and tables. For example, simple but inefficient Insertion sort is compared to much more efficient Mergesort, Quicksort, and Heapsort algorithms that implement powerful divide-and-conquer techniques. Note that both Quicksort and Mergesort are implemented in the Java Collections framework.

Although these subjects had been already discussed in brief in COMPSCI.105, the present course studies efficiency of sorting and searching algorithms more thoroughly. You will find more details in the recommended textbooks [6, 12] (see Bibliography on pp. 13–14), as well as in numerous other books on algorithms and data structures you can find in libraries and bookshops, in particular, in [1, 2, 5, 8, 9, 13, 14, 16] and so forth. Although the older books may use outdated or even antiquated programming languages (such as Pascal, Modula, or Fortran), the algorithms and data structures are basically the same as they reflect programming experience of many years...

Chapters 2 and 3 were written by Dr. Michael Dinneen, with modifications by Dr. Mark Wilson.

Chapter 2 contains the reference material on graph algorithms. Topics include computer representations (adjacency matrices and lists), graph searching (BFS and DFS), and basic graph algorithms such as computing the shortest distances between vertices and finding (strongly) connected components.

Chapter 3 contains the reference material on automata theory. Topics include deterministic and nondeterministic finite-state machines (DFA/NFA), regular expressions, and algorithms dealing with regular languages. More information can be found in [7].

Chapters 4 and 5 are prepared by Andrew MacGibbon. Chapter 4 presents the reference material on context-free grammars and parsing adapted from Chapter 11 of [2].

Chapter 5 is adapted from the Java “Collections” tutorial and the Java API documentation.

Appendices A and B give additional information about the above topics.

Appendix A compiled by Dr. Georgy Gimel'farb presents Java abstract data types (ADT).

Appendix B written by Dr. Michael Dinneen describes a graph ADT.

Bibliography

- [1] A.V. Aho, J.E. Hopcroft and J.D. Ullman: *Data Structures and Algorithms*, Addison-Wesley, 1983.
- [2] A.V.Aho, J.D.Ullman: *Foundations of Computer Science*, Computer Science Press, 1992.
- [3] D.A.Bailey: *JavaTM Structures*, McGraw-Hill, 1999.
- [4] F.J. Brandenburg: *The Graph Template Library – GTL*, (see <http://www.fmi.uni-passau.de/Graphlet/>).
- [5] T.H. Cormen, C.E. Leiserson and R.L. Rivest: *Introduction to Algorithms*, McGraw-Hill, New York, 1990.
- [6] M.T. Goodrich and R. Tamassia: *Data Structures and Algorithms in Java*, John Wiley and Sons, Inc., 2001 (see <http://www.cgc.cs.jhu.edu/~goodrich/dsa/toc.html>).
- [7] J.E.Hopcroft, R.Motwani, and J.D.Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 2001
- [8] J. H. Kingston: *Algorithms and Data Structures: Design, Correctness, Analysis*, Addison-Wesley, 1991.
- [9] M. Main: *Data Structures & Other Objects Using JavaTM*, Addison-Wesley, 1999.
- [10] K. Mehlhorn and St. Nadher: *The LEDA Platform of Combinatorial and Geometric Computing*, Cambridge University Press, 1999 (see <http://www.mpi-sb.mpg.de/LEDA/leda.html>).
- [11] J. Orwant, J. Hietaniemi, and J. Macdonald: *Mastering Algorithms with Perl*, August 1999.
- [12] C. A. Shaffer: *A Practical Introduction to Data Structures and Algorithms*, Prentice Hall, 2001
- [13] T.A.Standish: *Data Structures in JavaTM*, Addison-Wesley, 1998.
- [14] R.E. Tarjan: *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, 1983.

- [15] J. Van Leeuwin: *Handbook of Theoretical Computer Science*, Vol. A, North Holland, 1990.
- [16] M.A. Weiss: *Data Structures and Algorithm Analysis in Java*, Addison-Wesley, 1999.