

Design and Analysis of Algorithms

The text introduces readers to different paradigms of computing in addition to the traditional approach of discussing fundamental computational problems and design techniques in the random access machine model. Alternate models of computation including parallel, cache-sensitive design and streaming algorithms are dealt in separate chapters to underline the significant role of the underlying computational environment in the algorithm design. The treatment is made rigorous by demonstrating new measures of performances along with matching lower bound arguments.

The importance of greedy algorithms, divide-and-conquer technique and dynamic programming is highlighted by additional applications to approximate algorithms that come with guarantees. In addition to several classical techniques, the book encourages liberal use of probabilistic analysis and randomized techniques that have been pivotal for many recent advances in this area. There is also a chapter introducing techniques for dimension reduction which is at the heart of many interesting applications in data analytics as well as statistical machine learning. While these techniques have been known for a while in other communities, their adoption into mainstream computer science has been relatively recent.

Concepts are discussed with the help of rigorous mathematical proofs, theoretical explanations and their limitations. Problems have been chosen from a diverse landscape including graphs, geometry, strings, algebra and optimization. Some exposition of approximation algorithms has also been included, which has been a very active area of research in algorithms. Real life applications and numerical problems are spread throughout the text. The reader is expected to test her understanding by trying out a large number of exercise problems accompanying every chapter.

The book assumes familiarity with basic data structures, to focus on more algorithmic aspects and topics of contemporary importance.

Sandeep Sen has been a faculty member in the Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India, since 1991 where he is currently a professor. His research spans randomized algorithms, computational geometry, dynamic graph algorithms and models of computation. He is a Fellow of the Indian National Science Academy and the Indian Academy of Sciences.

Amit Kumar is a faculty member in the Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India. His research lies in the area of combinatorial optimization, with emphasis on problems arising in scheduling, graph theory and clustering. He is a Fellow of Indian Academy of Sciences, and is a recent recipient of the Shanti Swarup Bhatnagar Award for Mathematical Sciences.

Design and Analysis of Algorithms

A Contemporary Perspective

Sandeep Sen
Amit Kumar



Cambridge University Press
978-1-108-49682-7 — Design and Analysis of Algorithms
Sandeep Sen , Amit Kumar
Frontmatter
[More Information](#)

CAMBRIDGE UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom
One Liberty Plaza, 20th Floor, New York, NY 10006, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
314 to 321, 3rd Floor, Plot No.3, Splendor Forum, Jasola District Centre, New Delhi 110025, India
79 Anson Road, #06–04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org

Information on this title: www.cambridge.org/9781108496827

© Sandeep Sen and Amit Kumar 2019

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2019

Printed in India

A catalogue record for this publication is available from the British Library

Library of Congress Cataloging-in-Publication Data

Names: Sen, Sandeep, author. | Kumar, Amit, 1976- author.

Title: Design and analysis of algorithms / Sandeep Sen, Amit Kumar.

Description: New York, NY, USA : University Printing House, 2019. | Includes bibliographical references and index.

Identifiers: LCCN 2019002080 | ISBN 9781108496827 (hardback : alk. paper) |

ISBN 9781108721998 (paperback : alk. paper)

Subjects: LCSH: Algorithms.

Classification: LCC QA9.58 .S454 2019 | DDC 005.1–dc23

LC record available at <https://lcn.loc.gov/2019002080>

ISBN 978-1-108-49682-7 Hardback

ISBN 978-1-108-72199-8 Paperback

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party internet websites referred to in this publication, and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.

To the loving memory of my parents, Sisir Sen and Krishna Sen who nourished and inspired my academic pursuits and all my teachers who helped me imbibe the beauty and intricacies of various subjects

– Sandeep Sen

To my parents
– Amit Kumar

Content

<i>List of Figures</i>	xv
<i>List of Tables</i>	xix
<i>Preface</i>	xxi
<i>Acknowledgments</i>	xxv
1 Model and Analysis	1
1.1 Computing Fibonacci Numbers	1
1.2 Fast Multiplication	3
1.3 Model of Computation	4
1.4 Randomized Algorithms: A Short Introduction	6
1.4.1 A different flavor of randomized algorithms	8
1.5 Other Computational Models	10
1.5.1 External memory model	10
1.5.2 Parallel model	11
Further Reading	12
<i>Exercise Problems</i>	13
2 Basics of Probability and Tail Inequalities	16
2.1 Basics of Probability Theory	16
2.2 Tail Inequalities	21
2.3 Generating Random Numbers	26
2.3.1 Generating a random variate for an arbitrary distribution	26

2.3.2	Generating random variables from a sequential file	27
2.3.3	Generating a random permutation	29
	Further Reading	31
	<i>Exercise Problems</i>	31
3	Warm-up Problems	34
3.1	Euclid's Algorithm for the Greatest Common Divisor (GCD)	34
3.1.1	Extended Euclid's algorithm	35
3.1.2	Application to cryptography	36
3.2	Finding the k th Smallest Element	37
3.2.1	Choosing a random splitter	38
3.2.2	Median of medians	39
3.3	Sorting Words	41
3.4	Mergeable Heaps	43
3.4.1	Merging binomial heaps	44
3.5	A Simple Semi-dynamic Dictionary	45
3.5.1	Potential method and amortized analysis	46
3.6	Lower Bounds	47
	Further Reading	50
	<i>Exercise Problems</i>	50
4	Optimization I: Brute Force and Greedy Strategy	54
4.1	Heuristic Search Approaches	55
4.1.1	Game trees*	57
4.2	A Framework for Greedy Algorithms	60
4.2.1	Maximum spanning tree	64
4.2.2	Finding minimum weight subset	64
4.2.3	A scheduling problem	65
4.3	Efficient Data Structures for Minimum Spanning Tree Algorithms	66
4.3.1	A simple data structure for Union-Find	68
4.3.2	A faster scheme	69
4.3.3	The slowest growing function?	71
4.3.4	Putting things together	72
4.3.5	Path compression only*	73
4.4	Greedy in Different Ways	74
4.5	Compromising with Greedy	76

Contents	ix
4.6 Gradient Descent*	77
4.6.1 Applications	83
Further Reading	87
<i>Exercise Problems</i>	88
5 Optimization II: Dynamic Programming	92
5.1 Knapsack Problem	94
5.2 Context Free Parsing	95
5.3 Longest Monotonic Subsequence	97
5.4 Function Approximation	99
5.5 Viterbi's Algorithm for Maximum Likelihood Estimation	100
5.6 Maximum Weighted Independent Set in a Tree	102
Further Reading	102
<i>Exercise Problems</i>	103
6 Searching	109
6.1 Skip-Lists – A Simple Dictionary	110
6.1.1 Construction of skip-lists	110
6.1.2 Analysis	111
6.1.3 Stronger tail estimates	113
6.2 Treaps: Randomized Search Trees	114
6.3 Universal Hashing	117
6.3.1 Existence of universal hash functions	120
6.4 Perfect Hash Function	121
6.4.1 Converting expected bound to worst case bound	122
6.5 A log log N Priority Queue*	122
Further Reading	124
<i>Exercise Problems</i>	125
7 Multidimensional Searching and Geometric Algorithms	128
7.1 Interval Trees and Range Trees	129
7.1.1 Two-dimensional range queries	131
7.2 k - d Trees	132
7.3 Priority Search Trees	135
7.4 Planar Convex Hull	137
7.4.1 Jarvis march	139
7.4.2 Graham's scan	140
7.4.3 Sorting and convex hulls	141

x	Contents
7.5 Quickhull Algorithm	142
7.5.1 Analysis	143
7.5.2 Expected running time*	145
7.6 Point Location Using Persistent Data Structure	146
7.7 Incremental Construction	149
Further Reading	152
<i>Exercise Problems</i>	153
8 String Matching and Finger Printing	157
8.1 Rabin–Karp Fingerprinting	157
8.2 KMP Algorithm	161
8.2.1 Analysis of the KMP algorithm	165
8.2.2 Pattern analysis	165
8.3 Tries and Applications	165
Further Reading	168
<i>Exercise Problems</i>	169
9 Fast Fourier Transform and Applications	171
9.1 Polynomial Evaluation and Interpolation	171
9.1.1 Multiplying polynomials	172
9.2 Cooley–Tukey Algorithm	173
9.3 The Butterfly Network	175
9.4 Schonage and Strassen’s Fast Multiplication*	176
9.5 Generalized String Matching	179
9.5.1 Convolution based approach	180
Further Reading	182
<i>Exercise Problems</i>	182
10 Graph Algorithms	184
10.1 Depth First Search	184
10.2 Applications of DFS	188
10.2.1 Strongly connected components (SCC)	188
10.2.2 Biconnected components	191
10.3 Path Problems	193
10.3.1 Bellman–Ford SSSP algorithm	194
10.3.2 Dijkstra’s SSSP algorithm	195
10.3.3 All pair shortest paths algorithm	197

Contents	xi
10.4 Computing Spanners for Weighted Graphs	198
10.5 Global Min-cut	201
10.5.1 The contraction algorithm	202
10.5.2 Probability of min-cut	203
Further Reading	204
<i>Exercise Problems</i>	205
11 Maximum Flow and Applications	208
11.0.1 Max-Flow Min-Cut	212
11.0.2 Ford and Fulkerson algorithm	213
11.0.3 Edmond–Karp augmentation strategy	214
11.0.4 Monotonicity lemma and bounding the number of iterations	215
11.1 Applications of Max-Flow	216
11.1.1 Disjoint paths	216
11.1.2 Bipartite matching	217
11.1.3 Circulation problems	222
11.1.4 Project planning	224
Further Reading	226
<i>Exercise Problems</i>	227
12 NP Completeness and Approximation Algorithms	230
12.1 Classes and Reducibility	233
12.2 Cook–Levin Theorem	235
12.3 Common NP-Complete Problems	237
12.4 Proving NP Completeness	240
12.4.1 Vertex cover and related problems	241
12.4.2 Three coloring problem	242
12.4.3 Knapsack and related problems	244
12.5 Other Important Complexity Classes	247
12.6 Combating Hardness with Approximation	249
12.6.1 Maximum knapsack problem	251
12.6.2 Minimum set cover	252
12.6.3 The metric TSP problem	253
12.6.4 Three coloring	253
12.6.5 Max-cut problem	254
Further Reading	254
<i>Exercise Problems</i>	255

13 Dimensionality Reduction*	258
13.1 Random Projections and the Johnson–Lindenstrauss Lemma	259
13.2 Gaussian Elimination	262
13.3 Singular Value Decomposition and Applications	264
13.3.1 Some matrix algebra and the SVD theorem	265
13.3.2 Low-rank approximations using SVD	267
13.3.3 Applications of low-rank approximations	269
13.3.4 Clustering problems	271
13.3.5 Proof of the SVD theorem	273
Further Reading	275
<i>Exercise Problems</i>	275
14 Parallel Algorithms	277
14.1 Models of Parallel Computation	277
14.2 Sorting and Comparison Problems	278
14.2.1 Finding the maximum	278
14.2.2 Sorting	282
14.3 Parallel Prefix	287
14.4 Basic Graph Algorithms	291
14.4.1 List ranking	292
14.4.2 Connected components	294
14.5 Basic Geometric Algorithms	298
14.6 Relation between Parallel Models	300
14.6.1 Routing on a mesh	301
Further Reading	303
<i>Exercise Problems</i>	304
15 Memory Hierarchy and Caching	308
15.1 Models of Memory Hierarchy	308
15.2 Transposing a Matrix	310
15.2.1 Matrix multiplication	311
15.3 Sorting in External Memory	313
15.3.1 Can we improve the algorithm?*	314
15.4 Cache Oblivious Design	316
15.4.1 Oblivious matrix transpose	317
Further Reading	320
<i>Exercise Problems</i>	321

Contents	xiii
16 Streaming Data Model	323
16.1 Introduction	323
16.2 Finding Frequent Elements in a Stream	324
16.3 Distinct Elements in a Stream	327
16.4 Frequency Moment Problem and Applications	331
16.4.1 The median of means trick	334
16.4.2 The special case of second frequency moment	335
16.5 Proving Lower Bounds for Streaming Model	337
Further Reading	339
<i>Exercise Problems</i>	340
Appendix A Recurrences and Generating Functions	343
A.1 An Iterative Method – Summation	344
A.2 Linear Recurrence Equations	345
A.2.1 Homogeneous equations	345
A.2.2 Inhomogeneous equations	346
A.3 Generating Functions	346
A.3.1 Binomial theorem	348
A.4 Exponential Generating Functions	348
A.5 Recurrences with Two Variables	349
<i>Bibliography</i>	351
<i>Index</i>	363

Figures

1.1	Algorithm for verifying matrix product	9
2.1	Generating a random permutation of n distinct objects	29
3.1	Euclid's algorithm	35
3.2	Algorithm based on median of medians	40
3.3	(a) Recursive construction of binomial tree; (b) Binomial heap of 11 elements consisting of three binomial trees	44
4.1	Illustration of the induction proof when root node is \vee	59
4.2	Algorithm Gen_Greedy	61
4.3	The matching (a, d) is a maximal independent set, but $(a, b), (c, d)$ is a larger maximal independent set	63
4.4	Kruskal's minimum spanning tree algorithm	64
4.5	Successive iterations in Kruskal's greedy algorithm	67
4.6	An example of a Union-Find data structure storing elements $\{1, 2, \dots, 12\}$	69
4.7	An example of a path compression heuristic	70
4.8	Prim's minimum spanning tree algorithm	74
4.9	Boruvka's minimum spanning tree algorithm	76
4.10	A convex function of one variable	78
4.11	Gradient descent algorithm	80
4.12	The convex function is non-differentiable at x	82
4.13	The point P should ideally lie on the intersection of the three circles, but there are some measurement errors	83
4.14	A perceptron with inputs x_1, x_2, \dots, x_n and output determined by the sign of $w_0 + w_1x_1 + \dots + w_nx_n$	84

xvi	Figures
4.15 A plot of the function g as an approximation to the sgn function	85
4.16 The points in P are denoted by dots, and those in N by squares	86
5.1 Recursive Fibonacci sequence algorithm	93
5.2 The recursive unfolding of computing F_6	93
5.3 Table (a) implies that the string aba does not belong to the grammar whereas Table (b) shows that $baaba$ can be generated from S	96
5.4 In (a), the constant function is an average of the y values which minimizes the sum of squares error. In (b), a 3 step function approximates the 7 point function	99
5.5 For the label aba and starting vertex v_1 , there are several possible labeled paths like $[v_1, v_3, v_4, v_6]$, $[v_1, v_4, v_5, v_6]$, etc.	101
5.6 In the sequence 13, 5, 8, 12, 9, 14, 15, 2 we have predefined the tree structure but only the first four numbers have been scanned, i.e., 13, 5, 8, 12	104
6.1 The path traversed while searching for the element 87	111
6.2 Diagrams (a) to (d) depict the rotations required to insert the element 20 having priority 58 starting with the treap for the first four elements. Diagram (e) is the final tree after the entire insertion sequence. Diagram (f) shows the schematic for left/right rotations	117
6.3 The shaded leaf nodes correspond to the subset S	123
7.1 The structure of a one-dimensional range search tree where a query interval is split into at most $2\log n$ disjoint canonical (half)-intervals	130
7.2 The rectangle is the union of the slabs represented by the darkened nodes plus an overhanging left segment containing p_6	132
7.3 Rectangular range query used in a k - d tree	133
7.4 Each rectangular subdivision corresponds to a node in the k - d tree and is labeled by the splitting axis – either vertical or horizontal	134
7.5 The query is the semi-infinite upper slab supported by the two bottom points $(0, 4.5)$ and $(10, 4.5)$	137
7.6 The figure on the left is convex, whereas the one on the right is not convex	137
7.7 Convex hull of points shown as the shaded region	138
7.8 Jarvis March algorithm for convex hull	140
7.9 Merging upper hulls	141
7.10 Left turn(p_m, p_{2j-1}, p_{2j}) is true but $\text{slope}(\overline{p_{2j-1}p_{2j}})$ is less than the median slope given by L	144
7.11 The shaded vertical region does not contain any intersection points	147
7.12 An example depicting $\Omega(n^2)$ space complexity for n segments	149
7.13 Path copying technique on adjacent slabs s_5 and s_6	149
7.14 Incremental algorithm for closest pair computation	150

Figures	xvii
7.15 Maximum number of D -separated points per cell is 4 and the shaded area is the region within which a point can lie with distance less than D from p	152
8.1 Testing equality of two large numbers	159
8.2 Karp–Rabin string matching algorithm	160
8.3 Knuth–Morris–Pratt string matching algorithm	164
8.4 The suffix tree construction corresponding to the string catca \$: (i) Suffix a starting at position 5, (ii) Suffixes at position 4 and 5, etc.	167
9.1 FFT computation	174
9.2 Computing an eight point FFT using a butterfly network	176
9.3 Matching with wildcards: The pattern is $X = 3\ 2\ 1\ *$ and $r_1 = 6, r_2 = 4, r_3 = 11, r_4 = 0$ all chosen from $[1 \dots 16]$ corresponding to $p = 17$	182
10.1 Algorithm for Depth First Search	186
10.2 The pair of numbers associated with each vertex denotes the starting time and finishing time respectively as given by the global counter	187
10.3 The pair of numbers associated with the vertices represent the start and finish time of the DFS procedure	189
10.4 Finding strongly connected components using two DFS	190
10.5 The component graph for the graph on the left is shown on the right	192
10.6 Bellman–Ford single-source shortest path problem	194
10.7 For every vertex, the successive labels over the iterations of the Bellman–Ford algorithm are indicated where i denotes ∞	196
10.8 Dijkstra’s single source shortest path algorithm	196
10.9 An algorithm for weighted 3-spanner	199
10.10 The 3-spanner algorithm – Stages 1 and 2	200
10.11 Stretch bound: (i) Intracluster; (ii) Intercluster	201
10.12 Algorithm for computing t -partition	202
11.1 Greedy algorithm for max-flow: it may not give optimal solution	211
11.2 Example of residual graph	211
11.3 Example of disjoint paths in a graph	217
11.4 Reduction from a matching instance on the left to a max-flow instance on the right	218
11.5 The matching M is shown by dotted edges	220
11.6 Illustration of Hall’s theorem	221
11.7 The shaded region consisting of s, A, C, D, E, F represents a min s - t cut of capacity 3	222
11.8 Example of circulation on the left	223
11.9 Figure on the left shows an example of DAG on a set of tasks	225
11.10 Figure for Exercise 11.3. Numbers denote edge capacities	227
12.1 Many-to-one reduction from Π_1 to Π_2 by using a function $f : \mathbb{N} \rightarrow \mathbb{N}$	234

xviii	Figures
12.2 Graph illustrating the reduction for the 3-CNF formula	242
12.3 Illustration of the reduction proving NP-completeness of the three coloring problem.	243
13.1 Gaussian elimination algorithm	263
13.2 A two-dimensional illustration of the SVD subspace V_1 of points represented by circular dots, where V_1 denotes the subspace spanned by the first column of V	273
14.1 Parallel odd–even transposition sort	282
14.2 Sorting two rows by alternately sorting rows and columns	284
14.3 Shearsort algorithm for a rectangular mesh	285
14.4 Partition sort in parallel	285
14.5 Parallel prefix computation: this procedure computes prefix of x_a, x_{a+1}, \dots, x_b	287
14.6 Parallel prefix computation using blocking	289
14.7 Recursive unfolding of the prefix circuit with 8 inputs in terms of 4-input and 2-input circuits	289
14.8 Parallel list ranking	292
14.9 The star rooted at vertex 5 is connected to all the stars (dotted edges) on the right that have higher numbered vertices	296
14.10 Parallel connectivity: We assume that there is a processor assigned to every vertex $v \in V$ and to every edge $(u, w) \in E$	297
14.11 Starting from (r, c) , the packet is routed to a random row r' within the same column c	303
15.1 The tiling of a $p \times q$ matrix in a row-major layout	310
15.2 Transposing a matrix using minimal transfers	311
15.3 Computing the product $Z = X \cdot Y$ using tiles of size s	312
15.4 Searching a dictionary in external memory	316
15.5 Consider numbers from 1 to 16 arranged according to the Algorithm in Fig. 15.4	316
15.6 Algorithm for matrix transpose	317
15.7 Base case: Both A, B fit into cache – no further cache miss	318
15.8 The subsequence $\sigma_{i_1} \sigma_{i_1+1} \dots \sigma_{i_1+r_1} \sigma_{i_2}$ have $k+1$ distinct elements, whereas the subsequence $\sigma_{i_1} \sigma_{i_1+1} \dots \sigma_{i_1+r_1}$ have k distinct elements	319
16.1 The algorithm A receives input x_t at time t , but has limited space	324
16.2 Boyer–Moore majority voting algorithm	325
16.3 Misra–Gries streaming algorithm for frequent elements	327
16.4 Counting number of distinct elements	328
16.5 Combining reservoir sampling with the estimator for F_k	332
16.6 Estimating F_2	336

Tables

5.1	The dynamic programming table for Knapsack	95
8.1	Finite automaton transition function for the string <i>aabb</i> matching	163
8.2	Illustration of matching using KMP failure function f for the pattern <i>abababca</i> .	164
12.1	Creating an instance of decision-knapsack from a given instance of 3-SAT	245
14.1	Consecutive snapshots of the list ranking algorithm on 15 elements	293

Preface

This book embodies a distillation of topics that we, as educators, have frequently covered in the past two decades in various postgraduate and undergraduate courses related to *Design and Analysis of Algorithms* in IIT Delhi. The primary audience were the junior level (3rd year) computer science (CS) students and the first semester computer science post-graduate students. This book can also serve the purpose of material for a more advanced level algorithm course where the reader is exposed to alternate and more contemporary computational frameworks that are becoming common and more suitable.

A quick glance through the contents will reveal that about half of the topics are covered by many standard textbooks on algorithms like those by Aho et al. [7], Horowitz et al. [65], Cormen et al. [37], and more recent ones like those by Kleinberg and Tardos [81] and Dasgupta et al. [40]. The first classic textbook in this area, viz., that by Aho et al., introduces the subject with the observation ‘The study of algorithms is at the very heart of computer science’ and this observation has been reinforced over the past five decades of rapid development of computer science as well as of the more applied field of information technology. Because of its foundational nature, many of the early algorithms discovered about five decades ago continue to be included in every textbook written including this one – for example, algorithms like FFT, quicksort, Dijkstra’s shortest paths, etc.

What motivated us to write another book on algorithms are the several important and subtle changes in the understanding of many computational paradigms and the relative importance of techniques emerging out of some spectacular discoveries and changing technologies. As teachers and mentors, it is our responsibility to inculcate the right focus

in the younger generation so that they continue to enjoy this intellectually critical activity and contribute to the enhancement of the field of study. As more and more human activities are becoming computer-assisted, it becomes obligatory to emphasize and reinforce the importance of efficient and faster algorithms, which is the core of any automated process. We are often limited and endangered by the instinctive use of ill-designed and brute force algorithms, which are often erroneous, leading to fallacious scientific conclusions or incorrect policy decisions. It is therefore important to introduce some formal aspects of algorithm design and analysis into the school curriculum at par with maths and science, and sensitize students about this subject.

Who can use it

The present book is intended for students who have acquired skills in programming as well as basic data structures like arrays, stacks, lists, and even some experience with balanced trees. The authors, with a long experience behind them in teaching this subject, are convinced that algorithm design can be a deceptively hard subject and a gentle exposure is important for, both, understanding and sustaining interest. In IIT Delhi, CS undergraduates do a course in programming followed by a course in data structures with some exposure to basic algorithmic techniques. This book is intended for students having this background and so we have avoided any formal introduction of basic data structures including elementary graph searching methods like BFS/DFS. Instead, the book focusses on a mathematical treatment of the previously acquired knowledge and emphasizes a clean and crisp analysis of any new idea and technique. The CS students in IIT Delhi would have done a course in discrete mathematics and probability before they do this course. The design of efficient algorithms go hand-in-hand with our ability to quickly screen intuitions that lead to poor algorithms – both in terms of efficiency and correctness. We have consciously avoided topics that require long and dry formalism, although we have emphasized rigor at every juncture.

An important direction that we have pursued is based on the significance of adapting algorithm design to the computational environment. Although there has been a long history of research in designing algorithms for real-world models such as parallel and cache-hierarchy models, these have remained in the realms of niche and specialized graduate courses. The tacit assumption in basic textbooks is that we are dealing with uniform cost random access machines (RAMs). It is our firm belief that algorithm design is as much a function of the specific problem as the target model of execution, and failing to recognize this aspect makes the exercise somewhat incomplete and ineffective. Therefore, trying to execute the textbook data structures on a distributed model or Dijkstra's algorithm in a parallel computer would be futile. In summary,

$$\textit{Algorithms} = \textit{ProblemDefinition} + \textit{Model}$$

The last three chapters specifically address three very important environments, namely parallel computing, memory hierarchy, and streaming. They form the core of a course taught in IIT Delhi, *Model Centric Algorithm Design* – some flavor can add diversity to a core course in algorithms. Of course, any addition to a course would imply proportionate exclusion of some other equally important topic – so it is eventually the instructor’s choice.

Another recurring theme in the book is the liberal use of randomized techniques in algorithm design. To help students appreciate this aspect, we have described some basic tools and applications in Chapter 2. Even for students who are proficient in the use of probabilistic calculations (we expect all CS majors to have one college level course in probability), may find these applications somewhat non-intuitive and surprising – however, this may also turn into a very versatile and useful tool for anyone who is mathematically minded.

The other major development over the past decade is an increasing popularity of algebraic (particularly spectral) methods for combinatorial problems. This has made the role of conventional continuous mathematics more relevant and important. Reconciling and bridging the two distinct worlds of discrete and continuous methods is a huge challenge to even an experienced researcher, let alone an average student. It is too difficult to address this in a book like ours but we have tried to present some flavor in Chapter 12, which is an introduction to the technique of random projections.

Each chapter is followed by some brief discussion on some historical origins of the problem and pointers to relevant existing literature. The subsections/sections/chapters marked with * are more suitable for the advanced reader and may be skipped by others without loss of continuity.

One of the primary objectives of a course on algorithms is to encourage an appreciation for creativity without sacrificing rigor – this aspect makes algorithm design one of the most challenging and fascinating intellectual pursuit.

Suggested use of the chapters

The material presented in the sixteen chapters can be taught over two semesters at a leisurely pace, for example, in a two sequence course on algorithms. Alternately, for a first course on algorithms (with prior background in basic data structures), the instructor can choose majority portions from Chapters 3 to 11 and parts of Chapter 12. An advanced course can be taught using material from Chapters 12–16. Chapters 14–16 can form the crux of a course on *model centric algorithm design* which can be thought of as a more pragmatic exposure to theory of computation using contemporary frameworks.

Sandeep Sen
Amit Kumar
New Delhi, 2019

Acknowledgments

The authors would like to acknowledge their respective PhD advisors, John Reif and Jon Kleinberg, as their inspiring *gurus* in their journey in the algorithmic world. Sandeep Sen would also like to acknowledge Isaac Scherson, his Masters supervisor, for his motivating support to pursue algorithmic research.

The authors would like to thank many experienced researchers and teachers for their encouraging comments, including Pankaj Agarwal, Gary Miller, Sarel Har Peled, Jeff Vitter, Ravi Kannan, Sachin Maheshwari, Bernard Chazelle, Sartaj Sahni, Arijit Bishnu, and Saurabh Ray. More specifically, the authors would like to acknowledge Kurt Mehlhorn for sharing his notes on Gaussian Elimination in the finite precision model and Surender Baswana for his careful reading of the section on Graph Spanners.

It will also be fitting to acknowledge the numerous students in IIT Delhi, CSE department who have taken our courses in the area of algorithms, whose probing questions and constant prodding have contributed immensely to the shaping of the current contents.

Sandeep Sen would like to acknowledge the patience and support of his wife, Anuradha, who showed exemplary tolerance in sacrificing many hours that was due to her and his son, Aniruddha, for keeping the adrenaline going as a reminder of what the next generation is likely to be interested in. Amit Kumar would like to acknowledge his wife, Sonal, for her unwavering support and patience, and daughters, Aanvi and Anshika, for their love and affection. He would also like to acknowledge his parents for their encouragement and inspiration.

Since typists have long become defunct, it is obligatory to acknowledge the contributions of Donald Knuth for \TeX and Leslie Lamport's \LaTeX manual for the ease and convenience of writing such textbooks. For the sake of the environment and future of the world, we hope that long-term dissemination of such books will be in the electronic medium.