

Computational Complexity: A Modern Approach

Sanjeev Arora and Boaz Barak
Princeton University

<http://www.cs.princeton.edu/theory/complexity/>
complexitybook@gmail.com

Not to be reproduced or distributed without the authors' permission

To our wives— Silvia and Ravit

About this book

Computational complexity theory has developed rapidly in the past three decades. The list of surprising and fundamental results proved since 1990 alone could fill a book: these include new probabilistic definitions of classical complexity classes ($\mathbf{IP} = \mathbf{PSPACE}$ and the \mathbf{PCP} Theorems) and their implications for the field of approximation algorithms; Shor's algorithm to factor integers using a quantum computer; an understanding of why current approaches to the famous \mathbf{P} versus \mathbf{NP} will not be successful; a theory of derandomization and pseudorandomness based upon computational hardness; and beautiful constructions of pseudorandom objects such as extractors and expanders.

This book aims to describe such recent achievements of complexity theory in the context of more classical results. It is intended to both serve as a textbook and as a reference for self-study. This means it must simultaneously cater to many audiences, and it is carefully designed with that goal. We assume essentially no computational background and very minimal mathematical background, which we review in Appendix A. We have also provided a *web site* for this book at <http://www.cs.princeton.edu/theory/complexity/> with related auxiliary material including detailed teaching plans for courses based on this book, a draft of all the book's chapters, and links to other online resources covering related topics. Throughout the book we explain the context in which a certain notion is useful, and *why* things are defined in a certain way. We also illustrate key definitions with examples. To keep the text flowing, we have tried to minimize bibliographic references, except when results have acquired standard names in the literature, or when we felt that providing some history on a particular result serves to illustrate its motivation or context. (Every chapter has a notes section that contains a fuller, though still brief, treatment of the relevant works.) When faced with a choice, we preferred to use simpler definitions and proofs over showing the most general or most optimized result.

The book is divided into three parts:

Part I: Basic complexity classes. This part provides a broad introduction to the field. Starting from the definition of Turing machines and the basic notions of computability theory, it covers the basic time and space complexity classes, and also includes a few more modern topics such as probabilistic algorithms, interactive proofs, cryptography, quantum computers, and the \mathbf{PCP} Theorem and its applications.

Part II: Lower bounds on concrete computational models. This part describes lower bounds on resources required to solve algorithmic tasks on concrete models such as circuits, decision trees, etc. Such models may seem at first sight very different from Turing machines, but looking deeper one finds interesting interconnections.

Part III: Advanced topics. This part is largely devoted to developments since the late 1980s. It includes counting complexity, average case complexity, hardness amplification, derandomization and pseudorandomness, the proof of the \mathbf{PCP} theorem, and natural proofs.

Almost every chapter in the book can be read in isolation (though Chapters 1, 2 and 7 must not be skipped). This is by design, because the book is aimed at many classes of readers:

- *Physicists, mathematicians, and other scientists.* This group has become increasingly interested in computational complexity theory, especially because of high-profile results such as Shor's algorithm and the recent deterministic test for primality. This

intellectually sophisticated group will be able to quickly read through Part I. Progressing on to Parts II and III they can read individual chapters and find almost everything they need to understand current research.

- *Computer scientists who do not work in complexity theory per se.* They may use the book for self-study, reference, or to teach an undergraduate or graduate course in theory of computation or complexity theory.
- *All those —professors or students— who do research in complexity theory or plan to do so.* The coverage of recent results and advanced topics is detailed enough to prepare readers for research in complexity and related areas.

This book can be used as a textbook for several types of courses:

- *Undergraduate Theory of Computation.* Many Computer Science departments offer an undergraduate *Theory of Computation* course, using say Sipser’s book [Sip96]. Our text could be used to supplement Sipser’s book with coverage of some more modern topics such as probabilistic algorithms, cryptography and quantum computing. Undergraduate students may find these more exciting than traditional topics such as automata theory and the finer distinctions of computability theory. The prerequisite mathematical background would be some comfort with mathematical proofs and discrete mathematics, as covered in the typical “discrete math”/“math for CS” courses currently offered in many CS departments.
- *Introduction to computational complexity for advanced undergrads/beginning grads.* The book can be used as a text for an introductory complexity course aimed at advanced undergraduate or graduate students in computer science (replacing books such as Papadimitriou’s 1994 text [Pap94], that do not contain many recent results). Such a course would probably include many topics from Part I and then a sprinkling from Parts II and III, and assume some background in algorithms and/or the theory of computation.
- *Graduate Complexity course.* The book can serve as a text for a graduate complexity course that prepares graduate students for research in complexity theory or related areas like algorithms, machine learning, etc. Such a course can use Part I to review basic material, and then move on to the advanced topics of Parts II and III. The book contains far more material than can be taught in one term, and we provide on our website several alternative outlines for such a course.
- *Graduate seminars or advanced courses.* Individual chapters from Parts II and III can be used in seminars or advanced courses on various topics in complexity theory (e.g., derandomization, the **PCP** Theorem, lower bounds).

We provide several teaching plans and material for such courses on the book’s web site. If you use the book in your course, we’d love to hear about it and get your feedback. We ask that you do not publish solutions for the book’s exercises on the web though, so other people can use them as homework and exam questions as well.

As we finish this book, we are sorely aware of many more exciting results that we had to leave out. We hope the copious references to other texts will give the reader plenty of starting points for further explorations. We also plan to periodically update the book’s website with pointers to newer results or expositions that may be of interest to you.

Above all, we hope that this book conveys our excitement about computational complexity and the insights it provides in a host of other disciplines.

Onward to **P** versus **NP**!

Acknowledgements

Our understanding of complexity theory was shaped through interactions with our colleagues, and we have learned a lot from far too many people to mention here. Boaz would like to especially thank two mentors— Oded Goldreich and Avi Wigderson— who introduced to him the world of theoretical computer science and still influence much of his thinking on this area.

We thank Luca Trevisan for coconceiving the book (7 years ago!) and helping write the first drafts of a couple of chapters. Several colleagues have graciously agreed to review for us early drafts of parts of this book. These include Scott Aaronson, Noga Alon, Paul Beame, Irit Dinur, Venkatesan Guruswami, Valentine Kavanets, Jonathan Katz, Subhash Khot, Jiří Matoušek, Klaus Meer, Or Meir, Moni Naor, Alexandre Pinto, Alexander Razborov, Oded Regev, Omer Reingold, Ronen Shaltiel, Madhu Sudan, Amnon Ta-Shma, Iannis Tourlakis, Chris Umans, Salil Vadhan, Dieter van Melkebeek, Umesh Vazirani, and Joachim von zur Gathen. Special thanks to Jiří, Or, Alexandre, Dieter and Iannis for giving us very detailed and useful comments on many chapters of this book.

We also thank many others who have sent notes on typos or bugs, comments that helped improve the presentations, or answered our questions on a particular proof or reference. These include Emre Akbas, Djangir Babayev, Miroslav Balaz, Arnold Beckmann, Ido Ben-Eliezer, Goutam Biswas, Shreeshankar Bodas, Josh Bronson, Arkadev Chattopadhyay, Bernard Chazelle, Maurice Cochand, Nathan Collins, Tim Crabtree, Morten Dahl, Ronald de Wolf, Scott Diehl, Dave Doty, Michael Fairbank, Joan Feigenbaum, Lance Fortnow, Ali Ghodsi, Parikshit Gopalan, Vipul Goyal, Venkat Guruswami, Stephen Harris, Johan Håstad, Andre Hernich, Yaron Hirsch, Thomas Holenstein, Moukarram Kabbash, Bart Kastermans, Joe Kilian, Tomer Kotek, Michal Koucky, Sebastian Kuhnert, Katrina LaCurts, Chang-Wook Lee, James Lee, John Lenz, Meena Mahajan, Mohammad Mahmoody-Ghidary, Shohei Matsuura, Mauro Mazzieri, John McCullough, Eric Miles, Shira Mitchell, Mohsen Momeni, Kamesh Munagala, Rolf Neidermeier, Robert Nowotniak, Taktin Oey, Emily Pitler, Toni Pitassi, Manoj Prabhakaran, Anup Rao, Saikiran Rapaka, Nicola Rebagliati, Johan Richter, Ron Rivest, Mohammad Sadeq Dousti, Rahul Santhanam, Cem Say, Robert Schweiker, Thomas Schwentick, Joel Seiferas, Jonah Sherman, Amir Shpilka, Yael Snir, Nikhil Srivastava, Thomas Starbird, Jukka Suomela, Elad Tsur, Leslie Valiant, Vijay Vazirani, Suresh Venkatasubramanian, Justin Vincent-Foglesong, Jirka Vomlel, Daniel Wichs, Avi Wigderson, Maier Willard, Roger Wolff, Jureg Wullschleger, Rui Xue, Jon Yard, Henry Yuen, Wu Zhanbin, and Yi Zhang. Thank you!

Doubtless this list is still missing some of the people that helped us with this project over the years— if you are one of them we are both grateful and sorry.

This book was typeset using L^AT_EX, for which we're grateful to Donald Knuth and Leslie Lamport. Stephen Boyd and Lieven Vandenberghé kindly shared with us the L^AT_EX macros of their book *Convex Optimization*.

Most of all, we'd like to thank our families— Silvia, Nia and Rohan Arora, and Ravit and Alma Barak.

Contents at a glance

Introduction	1
Part I: Basic Complexity Classes	
Chapter 1: The computational model - and why it doesn't matter	13
Chapter 2: NP and NP completeness	37
Chapter 3: Diagonalization	61
Chapter 4: Space complexity	71
Chapter 5: The polynomial hierarchy and alternations	85
Chapter 6: Boolean circuits	95
Chapter 7: Randomized Computation	109
Chapter 8: Interactive proofs	127
Chapter 9: Cryptography	151
Chapter 10: Quantum computation	175
Chapter 11: PCP Theorem and Hardness of Approximation: An introduction	205
Part II: Lower bounds for concrete computational models	
Chapter 12: Decision trees	223
Chapter 13: Communication complexity	233
Chapter 14: Circuit lower bounds	247
Chapter 15: Proof Complexity	265
Chapter 16: Algebraic computation models	275
Part III: Advanced topics	
Chapter 17: Complexity of counting	295
Chapter 18: Average case complexity: Levin's theory	313
Chapter 19: Hardness amplification and error correcting codes	323
Chapter 20: Derandomization	349
Chapter 21: Pseudorandom constructions: expanders and extractors	365
Chapter 22: Proofs of PCP theorems and the Fourier transform technique	397
Chapter 23: Why are circuit lower bounds so difficult?	429
Appendix A: Mathematical background	439

Contents

About this book	v
Introduction	1
0 Notational Conventions	7
0.1 Representing objects as strings	7
0.2 Decision problems / languages	8
0.3 Big-Oh notation	9
Exercises	9
I Basic Complexity Classes	11
1 The computational model —and why it doesn't matter	13
1.1 Modeling computation: What you really need to know	14
1.2 The Turing Machine	15
1.2.1 The expressive power of Turing machines	18
1.3 Efficiency and running time	19
1.3.1 Robustness of our definition	19
1.4 Machines as strings and the universal Turing machine	22
1.4.1 The Universal Turing Machine	22
1.5 Uncomputability: An introduction	24
1.5.1 The Halting Problem (first encounter with reductions)	25
1.5.2 Gödel's Theorem	25
1.6 The class P	26
1.6.1 Why the model may not matter	27
1.6.2 On the philosophical importance of P	28
1.6.3 Criticisms of P and some efforts to address them	28
1.6.4 Edmonds' quote	29
Chapter notes and history	30
Exercises	31
1.A Proof of Theorem 1.9: Universal Simulation in $O(T \log T)$ -time	34
2 NP and NP completeness	37
2.1 The class NP	38
2.1.1 Relation between NP and P	39
2.1.2 Non-deterministic Turing machines.	40
2.2 Reducibility and NP-completeness	40
2.3 The Cook-Levin Theorem: Computation is Local	42
2.3.1 Boolean formulae, CNF and SAT.	42
2.3.2 The Cook-Levin Theorem	43
2.3.3 Warmup: Expressiveness of Boolean formulae	43
2.3.4 Proof of Lemma 2.11	44
2.3.5 Reducing SAT to 3SAT.	46
2.3.6 More thoughts on the Cook-Levin theorem	47

2.4	The web of reductions	47
2.5	Decision versus search	51
2.6	coNP , EXP and NEXP	52
2.6.1	coNP	52
2.6.2	EXP and NEXP	53
2.7	More thoughts about P , NP , and all that	53
2.7.1	The philosophical importance of NP	53
2.7.2	NP and mathematical proofs	54
2.7.3	What if P = NP ?	54
2.7.4	What if NP = coNP ?	55
2.7.5	Is there anything between NP and NP -complete?	55
2.7.6	Coping with NP hardness.	56
2.7.7	Finer explorations of time complexity	56
	Chapter notes and history	57
	Exercises	58
3	Diagonalization	61
3.1	Time Hierarchy Theorem	62
3.2	Nondeterministic Time Hierarchy Theorem	62
3.3	Ladner's Theorem: Existence of NP -intermediate problems	64
3.4	Oracle machines and the limits of diagonalization	65
3.4.1	Logical independence versus relativization	67
	Chapter notes and history	68
	Exercises	69
4	Space complexity	71
4.1	Definition of space bounded computation	71
4.1.1	Configuration graphs.	72
4.1.2	Some space complexity classes.	74
4.1.3	Space Hierarchy Theorem	75
4.2	PSPACE completeness	75
4.2.1	Savitch's theorem	77
4.2.2	The essence of PSPACE : optimum strategies for game-playing	78
4.3	NL completeness	79
4.3.1	Certificate definition of NL : read-once certificates	81
4.3.2	NL = coNL	81
	Chapter notes and history	83
	Exercises	83
5	The Polynomial Hierarchy and Alternations	85
5.1	The class Σ_2^P	85
5.2	The polynomial hierarchy.	86
5.2.1	Properties of the polynomial hierarchy.	87
5.2.2	Complete problems for levels of PH	87
5.3	Alternating Turing machines	88
5.3.1	Unlimited number of alternations	89
5.4	Time versus alternations: time-space tradeoffs for SAT	90
5.5	Defining the hierarchy via oracle machines.	91
	Chapter notes and history	92
	Exercises	92
6	Boolean Circuits	95
6.1	Boolean circuits and \mathbf{P}_{poly}	96
6.1.1	Relation between \mathbf{P}_{poly} and P	98
6.1.2	Circuit Satisfiability and an alternative proof of the Cook-Levin Theorem	98
6.2	Uniformly generated circuits	99

6.2.1	Logspace-uniform families	100
6.3	Turing machines that take advice	100
6.4	\mathbf{P}_{poly} and \mathbf{NP}	101
6.5	Circuit lower bounds	102
6.6	Non-uniform hierarchy theorem	103
6.7	Finer gradations among circuit classes	104
6.7.1	The classes \mathbf{NC} and \mathbf{AC}	105
6.7.2	\mathbf{P} -completeness	105
6.8	Circuits of exponential size	106
	Chapter notes and history	107
	Exercises	107
7	Randomized Computation	109
7.1	Probabilistic Turing machines	110
7.2	Some examples of PTMs	112
7.2.1	Finding a median	112
7.2.2	Probabilistic Primality Testing	113
7.2.3	Polynomial identity testing	114
7.2.4	Testing for perfect matching in a bipartite graph.	115
7.3	One-sided and “zero-sided” error: \mathbf{RP} , \mathbf{coRP} , \mathbf{ZPP}	115
7.4	The robustness of our definitions	116
7.4.1	Role of precise constants: error reduction.	116
7.4.2	Expected running time versus worst-case running time.	117
7.4.3	Allowing more general random choices than a fair random coin.	117
7.5	Relationship between \mathbf{BPP} and other classes	119
7.5.1	$\mathbf{BPP} \subseteq \mathbf{P}_{\text{poly}}$	119
7.5.2	\mathbf{BPP} is in \mathbf{PH}	120
7.5.3	Hierarchy theorems and complete problems?	121
7.6	Randomized reductions	122
7.7	Randomized space-bounded computation	122
	Chapter notes and history	123
	Exercises	124
8	Interactive proofs	127
8.1	Interactive proofs: some variations	127
8.1.1	Warmup: Interactive proofs with deterministic verifier and prover	128
8.1.2	The class \mathbf{IP} : probabilistic verifier	129
8.1.3	Interactive proof for graph non-isomorphism	131
8.2	Public coins and \mathbf{AM}	133
8.2.1	Simulating private coins	133
8.2.2	Set Lower Bound Protocol.	134
8.2.3	Sketch of proof of Theorem 8.12	138
8.2.4	Can GI be \mathbf{NP} -complete?	138
8.3	$\mathbf{IP} = \mathbf{PSPACE}$	139
8.3.1	Arithmetization	140
8.3.2	Interactive protocol for $\#\text{SAT}_D$	140
8.3.3	Protocol for TQBF: proof of Theorem 8.19	142
8.4	The power of the prover	143
8.5	Multiprover interactive proofs (\mathbf{MIP})	144
8.6	Program Checking	145
8.6.1	Languages that have checkers	146
8.6.2	Random Self Reducibility and the Permanent	146
8.7	Interactive proof for the Permanent	147
8.7.1	The protocol	148
	Chapter notes and history	149
	Exercises	150

9	Cryptography	151
9.1	Perfect secrecy and its limitations	152
9.2	Computational security, one-way functions, and pseudorandom generators	153
9.2.1	One way functions: definition and some examples	155
9.2.2	Encryption from one-way functions	156
9.2.3	Pseudorandom generators	157
9.3	Pseudorandom generators from one-way permutations	158
9.3.1	Unpredictability implies pseudorandomness	158
9.3.2	Proof of Lemma 9.10: The Goldreich-Levin Theorem	160
9.4	Zero knowledge	163
9.5	Some applications	165
9.5.1	Pseudorandom functions	166
9.5.2	Derandomization	167
9.5.3	Tossing coins over the phone and bit commitment	168
9.5.4	Secure multiparty computations	168
9.5.5	Lower bounds for machine learning	169
	Chapter notes and history	169
	Exercises	171
10	Quantum Computation	175
10.1	Quantum weirdness: the 2-slit experiment	176
10.2	Quantum superposition and qubits	178
10.2.1	EPR paradox	179
10.3	Definition of quantum computation and BQP	182
10.3.1	Some necessary linear algebra	182
10.3.2	The quantum register and its state vector	183
10.3.3	Quantum operations	183
10.3.4	Some examples of quantum operations	183
10.3.5	Quantum computation and BQP	185
10.3.6	Quantum circuits	187
10.3.7	Classical computation as a subcase of quantum computation	187
10.3.8	Universal operations	187
10.4	Grover's search algorithm.	188
10.5	Simon's Algorithm	192
10.5.1	Proof of Theorem 10.14	192
10.6	Shor's algorithm: integer factorization using quantum computers	193
10.6.1	The Fourier transform over \mathbb{Z}_M	194
10.6.2	Quantum Fourier Transform over \mathbb{Z}_M	195
10.6.3	Shor's Order-Finding Algorithm.	196
10.6.4	Reducing factoring to order finding.	199
10.6.5	Rational approximation of real numbers	200
10.7	BQP and classical complexity classes	200
10.7.1	Quantum analogs of NP and AM	201
	Chapter notes and history	202
	Exercises	203
11	PCP Theorem and Hardness of Approximation: An introduction	205
11.1	Motivation: approximate solutions to NP -hard optimization problems	206
11.2	Two views of the PCP Theorem.	207
11.2.1	PCP Theorem and locally testable proofs	207
11.2.2	PCP and Hardness of Approximation	210
11.3	Equivalence of the two views	211
11.3.1	Equivalence of theorems 11.5 and 11.9.	212
11.3.2	Review of the two views of the PCP Theorem	213
11.4	Hardness of approximation for vertex cover and independent set	213
11.5	NP \subseteq PCP (poly(n), 1): PCP from the Walsh-Hadamard code	215
11.5.1	Tool: Linearity Testing and the Walsh-Hadamard Code	215

11.5.2 Proof of Theorem 11.19	217
Chapter notes and history	219
Exercises	220
II Lower bounds for Concrete Computational Models	221
12 Decision Trees	223
12.1 Decision trees and decision tree complexity	223
12.2 Certificate Complexity	225
12.3 Randomized Decision Trees	227
12.4 Some techniques for decision tree lower bounds	228
12.5 Lower bounds on Randomized Complexity	228
12.5.1 Sensitivity	228
12.5.2 The degree method	230
Chapter notes and history	231
Exercises	231
13 Communication Complexity	233
13.1 Definition of two-party communication complexity.	233
13.2 Lower bound methods	234
13.2.1 The fooling set method	235
13.2.2 The tiling method	235
13.2.3 The rank method	237
13.2.4 The discrepancy method	237
13.2.5 A technique for upper bounding the discrepancy	238
13.2.6 Comparison of the lower bound methods	239
13.3 Multiparty communication complexity	239
13.4 Overview of other communication models	242
Chapter notes and history	243
Exercises	244
14 Circuit lower bounds	247
14.1 \mathbf{AC}^0 and Håstad’s Switching Lemma	247
14.1.1 Håstad’s switching lemma	248
14.1.2 Proof of the switching lemma (Lemma 14.2)	249
14.2 Circuits With “Counters”: \mathbf{ACC}	251
14.3 Lower bounds for monotone circuits	253
14.3.1 Proving Theorem 14.7	254
14.4 Circuit complexity: The frontier	256
14.4.1 Circuit lower bounds using diagonalization	256
14.4.2 Status of \mathbf{ACC} versus \mathbf{P}	257
14.4.3 Linear Circuits With Logarithmic Depth	257
14.4.4 Branching Programs	258
14.5 Approaches using communication complexity	259
14.5.1 Connection to \mathbf{ACC}_0 Circuits	259
14.5.2 Connection to Linear Size Logarithmic Depth Circuits	260
14.5.3 Connection to branching programs	260
14.5.4 Karchmer-Wigderson communication games and depth lower bounds	260
Chapter notes and history	262
Exercises	263

15 Proof complexity	265
15.1 Some examples	265
15.2 Propositional calculus and resolution	266
15.2.1 Lower bounds using the bottleneck method	267
15.2.2 Interpolation theorems and exponential lower bounds for resolution	268
15.3 Other proof systems: a tour d’horizon	270
15.4 Metamathematical musings	271
Chapter notes and history	272
Exercises	273
16 Algebraic computation models	275
16.1 Algebraic straight-line programs and algebraic circuits	276
16.1.1 Algebraic straight line programs	276
16.1.2 Examples	277
16.1.3 Algebraic circuits	278
16.1.4 Analogs of P , NP for algebraic circuits	279
16.2 Algebraic Computation Trees	281
16.2.1 The topological method for lower bounds	284
16.3 The Blum-Shub-Smale Model	286
16.3.1 Complexity Classes over the Complex Numbers	287
16.3.2 Complete problems and Hilbert’s Nullstellensatz	288
16.3.3 Decidability Questions: Mandelbrot Set	288
Chapter notes and history	289
Exercises	290
III Advanced topics	293
17 Complexity of counting	295
17.1 Examples of Counting Problems	295
17.1.1 Counting problems and probability estimation	296
17.1.2 Counting can be harder than decision	297
17.2 The class #P	297
17.2.1 The class PP : decision-problem analog for #P .	298
17.3 #P completeness.	299
17.3.1 Permanent and Valiant’s Theorem	299
17.3.2 Approximate solutions to #P problems	304
17.4 Toda’s Theorem: PH \subseteq P ^{#SAT}	305
17.4.1 A detour: Boolean satisfiability with unique solutions	306
17.4.2 Properties of \oplus and proof of Lemma 17.17 for NP , coNP	307
17.4.3 Proof of Lemma 17.17; general case	308
17.4.4 Step 2: Making the reduction deterministic	309
17.5 Open Problems	310
Chapter notes and history	310
Exercises	311
18 Average Case Complexity: Levin’s Theory	313
18.1 Distributional Problems and distP	314
18.2 Formalization of “real-life distributions”	316
18.3 distNP and its complete problems	317
18.3.1 A complete problem for distNP	318
18.3.2 P -samplable distributions	319
18.4 Philosophical and practical implications	320
Chapter notes and history	321
Exercises	321

19	Hardness Amplification and Error Correcting Codes	323
19.1	Mild to strong hardness: Yao's XOR Lemma.	324
19.1.1	Proof of Yao's XOR Lemma using Impagliazzo's Hardcore Lemma.	326
19.1.2	Proof of Impagliazzo's Lemma	327
19.2	Tool: Error correcting codes	329
19.2.1	Explicit codes	330
19.2.2	Walsh-Hadamard Code.	331
19.2.3	Reed-Solomon Code	331
19.2.4	Reed-Muller Codes.	332
19.2.5	Concatenated codes	332
19.3	Efficient decoding.	333
19.3.1	Decoding Reed-Solomon	333
19.3.2	Decoding concatenated codes.	334
19.4	Local decoding and hardness amplification	335
19.4.1	Local decoder for Walsh-Hadamard.	336
19.4.2	Local decoder for Reed-Muller	337
19.4.3	Local decoding of concatenated codes.	338
19.4.4	Putting it all together.	339
19.5	List decoding	340
19.5.1	List decoding the Reed-Solomon code	341
19.6	Local list decoding: getting to $\mathbf{BPP} = \mathbf{P}$	342
19.6.1	Local list decoding of the Walsh-Hadamard code.	342
19.6.2	Local list decoding of the Reed-Muller code	342
19.6.3	Local list decoding of concatenated codes.	344
19.6.4	Putting it all together.	344
	Chapter notes and history	345
	Exercises	346
20	Derandomization	349
20.1	Pseudorandom Generators and Derandomization	350
20.1.1	Derandomization using pseudorandom generators	351
20.1.2	Hardness and Derandomization	352
20.2	Proof of Theorem 20.6: Nisan-Wigderson Construction	353
20.2.1	Two toy examples	354
20.2.2	The NW Construction	355
20.3	Derandomization under uniform assumptions	358
20.4	Derandomization requires circuit lower bounds	359
	Chapter notes and history	362
	Exercises	363
21	Pseudorandom constructions: expanders and extractors	365
21.1	Random walks and eigenvalues	366
21.1.1	Distributions as vectors and the parameter $\lambda(G)$	366
21.1.2	Analysis of the randomized algorithm for undirected connectivity.	369
21.2	Expander graphs.	369
21.2.1	The Algebraic Definition	370
21.2.2	Combinatorial expansion and existence of expanders.	370
21.2.3	Algebraic expansion implies combinatorial expansion.	371
21.2.4	Combinatorial Expansion Implies Algebraic Expansion	372
21.2.5	Error reduction using expanders.	373
21.3	Explicit construction of expander graphs	375
21.3.1	Rotation maps.	375
21.3.2	The matrix/path product	376
21.3.3	The tensor product	376
21.3.4	The replacement product	377
21.3.5	The actual construction.	379
21.4	Deterministic logspace algorithm for undirected connectivity.	380

21.4.1	The logspace algorithm for connectivity (proof of Theorem 21.21)	381
21.5	Weak Random Sources and Extractors	382
21.5.1	Min Entropy	382
21.5.2	Statistical distance	383
21.5.3	Definition of randomness extractors	383
21.5.4	Existence proof for extractors.	384
21.5.5	Extractors based on hash functions	385
21.5.6	Extractors based on random walks on expanders	385
21.5.7	Extractors from pseudorandom generators	386
21.6	Pseudorandom generators for space bounded computation	388
	Chapter notes and history	391
	Exercises	393
22	Proofs of PCP Theorems and the Fourier Transform Technique	397
22.1	Constraint satisfaction problems with non-binary alphabet	397
22.2	Proof of the PCP Theorem	398
22.2.1	Proof outline for the PCP Theorem.	398
22.2.2	Dinur's Gap Amplification: Proof of Lemma 22.5	399
22.2.3	Expanders, walks, and hardness of approximating INDSET	400
22.2.4	Dinur's Gap-amplification	401
22.2.5	Alphabet Reduction: Proof of Lemma 22.6	405
22.3	Hardness of 2CSP_W : Tradeoff between gap and alphabet size	407
22.3.1	Idea of Raz's proof: Parallel Repetition	408
22.4	Håstad's 3-bit PCP Theorem and hardness of MAX-3SAT	408
22.4.1	Hardness of approximating MAX-3SAT	409
22.5	Tool: the Fourier transform technique	410
22.5.1	Fourier transform over $\text{GF}(2)^n$	410
22.5.2	The connection to PCPs: High level view	411
22.5.3	Analysis of the linearity test over $\text{GF}(2)$	411
22.6	Coordinate functions, Long code and its testing	412
22.7	Proof of Theorem 22.16	415
22.8	Hardness of approximating SET-COVER	418
22.9	Other PCP Theorems: A Survey	420
22.9.1	PCP's with sub-constant soundness parameter	420
22.9.2	Amortized query complexity	420
22.9.3	2-bit tests and powerful fourier analysis	421
22.9.4	Unique games and threshold results	421
22.9.5	Connection to Isoperimetry and Metric Space Embeddings	422
	Chapter notes and history	422
	Exercises	423
22.A	Transforming $q\text{CSP}$ instances into "nice" instances.	426
23	Why are circuit lower bounds so difficult?	429
23.1	Definition of natural proofs	429
23.2	What's so natural about natural proofs?	430
23.2.1	Why constructiveness?	431
23.2.2	Why largeness?	432
23.2.3	Natural proofs from complexity measures	432
23.3	Proof of Theorem 23.1	433
23.4	An "unnatural" lower bound	434
23.5	A philosophical view	435
	Chapter notes and history	436
	Exercises	436

A Mathematical Background.	439
A.1 Sets, Functions, Pairs, Strings, Graphs, Logic	439
A.2 Probability theory	441
A.2.1 Random variables and expectations.	441
A.2.2 The averaging argument	442
A.2.3 Conditional probability and independence	443
A.2.4 Deviation upper bounds	444
A.2.5 Some other inequalities.	445
A.2.6 Statistical distance	446
A.3 Number theory and groups	447
A.3.1 Groups.	448
A.3.2 Finite groups	448
A.3.3 The Chinese Remainder Theorem	449
A.4 Finite fields	450
A.4.1 Non-prime fields.	450
A.5 Basic facts from linear algebra	451
A.5.1 Inner product	452
A.5.2 Dot product	453
A.5.3 Eigenvectors and eigenvalues	453
A.5.4 Norms	454
A.5.5 Metric spaces	455
A.6 Polynomials	455
 Hints for selected exercises	 457
 Main Theorems and Definitions	 467
 Bibliography	 471

Introduction

“As long as a branch of science offers an abundance of problems, so long it is alive; a lack of problems foreshadows extinction or the cessation of independent development.”

David Hilbert, 1900

“The subject of my talk is perhaps most directly indicated by simply asking two questions: first, is it harder to multiply than to add? and second, why?...I (would like to) show that there is no algorithm for multiplication computationally as simple as that for addition, and this proves something of a stumbling block.”

Alan Cobham, 1964

The notion of *computation* has existed in some form for thousands of years, in contexts as varied as routine account-keeping and astronomy. Here are three examples of tasks that we may wish to solve using computation:

- Given two integer numbers, compute their product.
- Given a set of n linear equations over n variables, find a solution if it exists.
- Given a list of acquaintances and a list of all pairs among them who do not get along, find the largest set of acquaintances you can invite to a dinner party such that every two invitees get along with one another.

Throughout history people had a notion of a process of producing an output from a set of inputs in a finite number of steps, and thought of “computation” as “a person writing numbers on a scratch pad following certain rules.”

One of the important scientific advances in the first half of the 20th century was that the notion of “computation” received a much more precise definition. From this definition it quickly became clear that computation can happen in diverse physical and mathematical systems —Turing machines, lambda calculus, cellular automata, pointer machines, bouncing billiards balls, Conway’s *Game of life*, etc.. Surprisingly, all these forms of computation are equivalent —in the sense that each model is capable of implementing all computations that we can conceive of on any other model (see Chapter 1). This realization quickly led to the invention of the standard *universal electronic computer*, a piece of hardware that is capable of executing all possible programs. The computer’s rapid adoption in society in the subsequent decades brought computation into every aspect of modern life, and made computational issues important in design, planning, engineering, scientific discovery, and many other human endeavors. Computer *algorithms*, which are methods of solving computational problems, became ubiquitous.

But computation is not “merely” a practical tool. It is also a major scientific concept. Generalizing from physical models such as cellular automata, scientists now view many natural phenomena as akin to computational processes. The understanding of reproduction in living things was triggered by the discovery of self-reproduction in computational machines. (In fact, a book by the physicist Schrödinger [Sch44] predicted the existence of a DNA-like substance in cells before Watson and Crick discovered it, and was credited by Crick as an inspiration for that research.) Today, computational models underlie many research areas in biology and neuroscience. Several physics theories such as QED give a description of

nature that is very reminiscent of computation, motivating some scientists to even suggest that the entire universe may be viewed as a giant computer (see Lloyd [Llo06]). In an interesting twist, such physical theories have been used in the past decade to design a model for *quantum computation*; see Chapter 10.

Computability versus complexity. After their success in defining computation, researchers focused on understanding what problems are *computable*. They showed that several interesting tasks are *inherently uncomputable*: no computer can solve them without going into infinite loops (i.e., never halting) on certain inputs. Though a beautiful topic, computability will not be our focus in this book. We discuss it briefly in Chapter 1 and refer the reader to standard texts [Sip96, HMU01, Koz97, Rog87] for more details. Instead, we focus on *computational complexity theory*, which focuses on issues of *computational efficiency*—quantifying the amount of computational resources required to solve a given task. Below, we describe at an informal level how one can quantify “efficiency,” and after that discuss some of the issues that arise in connection with its study.

Quantifying computational efficiency

To explain what we mean by *computational efficiency*, we use the three examples of computational tasks we mentioned above. We start with the task of multiplying two integers. Consider two different methods (or *algorithms*) to perform this task. The first is *repeated addition*: to compute $a \cdot b$, just add a to itself $b - 1$ times. The other is the *grade-school algorithm* illustrated in Figure 1. Though the repeated addition algorithm is perhaps simpler than the grade-school algorithm, we somehow feel that the latter is *better*. Indeed, it is much more efficient. For example, multiplying 577 by 423 using repeated addition requires 422 additions, whereas doing it with the grade-school algorithm 3 multiplications of a number by a single digit and 3 additions.

$$\begin{array}{r}
 5 7 7 \\
 4 2 3 \\
 \hline
 1 7 3 1 \\
 1 1 5 4 \\
 2 3 0 8 \\
 \hline
 2 4 4 0 7 1
 \end{array}$$

Figure 1 Grade-school algorithm for multiplication. Illustrated for computing $577 \cdot 423$.

We will quantify the efficiency of an algorithm by studying how its number of *basic operations* scales as we increase the *size* of the input. For this discussion, let the *basic operations* be addition and multiplication of single digits. (In other settings, we may wish to throw in division as a basic operation.) The *size* of the input is the number of digits in the numbers. The number of basic operations used to multiply two n -digit numbers (i.e., numbers between 10^{n-1} and 10^n) is at most $2n^2$ for the grade-school algorithm and at least $n10^{n-1}$ for repeated addition. Phrased this way, the huge difference between the two algorithms is apparent: even for 11-digit numbers, a pocket calculator running the grade-school algorithm would beat the best current supercomputer running the repeated addition algorithm. For slightly larger numbers even a fifth grader with pen and paper would outperform a supercomputer. We see that *the efficiency of an algorithm is to a considerable extent much more important than the technology used to execute it*.

Surprisingly enough, there is an even faster algorithm for multiplication that uses the *Fast Fourier Transform*. It was only discovered some 40 years ago and multiplies two n -digit numbers using $cn \log n \log \log n$ operations where c is some absolute constant independent of n ; see Chapter 16. We call such an algorithm an $O(n \log n \log \log n)$ -step algorithm: see

our notational conventions below. As n grows, this number of operations is significantly smaller than n^2 .

For the task of solving linear equations, the classic *Gaussian elimination* algorithm (named after Gauss but already known in some form to Chinese mathematicians of the first century) uses $O(n^3)$ basic arithmetic operations to solve n equations over n variables. In the late 1960's, Strassen found a more efficient algorithm that uses roughly $O(n^{2.81})$ operations, and the best current algorithm takes $O(n^{2.376})$ operations (see Chapter 16).

The *dinner party* task also has an interesting story. As in the case of multiplication, there is an obvious and simple inefficient algorithm: try all possible subsets of the n people from the largest to the smallest, and stop when you find a subset that does not include any pair of guests who don't get along. This algorithm can take as much time as the number of subsets of a group of n people, which is 2^n . This is highly unpractical—an organizer of, say, a 70-person party, would need to plan it at least a thousand years in advance, even if she has a supercomputer at her disposal. Surprisingly, we still do not know of a significantly better algorithm for this task. In fact, as we will see in Chapter 2, we have reasons to suspect that no efficient algorithm *exists*, since this task turns out to be equivalent to the *independent set* computational problem, which, together with thousands of other important problems, is **NP**-complete. The famous “**P** versus **NP**” question (Chapter 2) asks whether or not any of these problems has an efficient algorithm.

Proving nonexistence of efficient algorithms

We have seen that sometimes computational tasks turn out to have nonintuitive algorithms that are more efficient than algorithms used for thousands of years. It would therefore be really interesting to prove for some computational tasks that the current algorithm is the *best*—in other words, no better algorithms exist. For instance, we could try to prove that the $O(n \log n \log \log n)$ -step algorithm for multiplication cannot be improved upon (thus implying that multiplication is inherently more difficult than addition, which does have an $O(n)$ -step algorithm). Or, we could try to prove that there is no algorithm for the dinner party task that takes fewer than $2^{n/10}$ steps. Trying to prove such results is a central goal of complexity theory.

How can we ever prove such a nonexistence result? There are infinitely many possible algorithms! So we have to *mathematically prove* that each one of them is less efficient than the known algorithm. This may be possible to do, because computation is a mathematically precise notion. In fact, this kind of result (if proved) would fit into a long tradition of *impossibility results* in mathematics, such as the independence of Euclid's parallel postulate from the other basic axioms of geometry, or the impossibility of trisecting an arbitrary angle using a compass and straightedge. Such results count among the most interesting, fruitful, and surprising results in mathematics.

In complexity theory, we are still only rarely able to prove such nonexistence of algorithms. We do have important nonexistence results in some concrete computational models that are not as powerful as general computers, which are described in Part II of the book. Since we are still missing good results for general computers, one important source of progress in complexity theory is our stunning success in *interrelating* different complexity questions, and the rest of the book is filled with examples of these.

Some interesting questions about computational efficiency

Now we give an overview of some important issues regarding computational complexity, all of which will be treated in greater detail in later chapters. An overview of mathematical background is given in Appendix A.

1. Computational tasks in a variety of disciplines such as the life sciences, social sciences and operations research involve searching for a solution across a vast space of possibilities (for example, the aforementioned tasks of solving linear equations and finding a maximal set of invitees to a dinner party). This is sometimes called *exhaustive search*, since the search *exhausts* all possibilities. Can this exhaustive search be replaced by a more *efficient* search algorithm?

As we will see in Chapter 2, this is essentially the famous “**P** vs. **NP**” question, considered *the* central problem of complexity theory. Many interesting search problems are **NP**-complete, which means that if the famous conjecture $\mathbf{P} \neq \mathbf{NP}$ is true, then these problems do not have efficient algorithms; they are *inherently intractable*.

2. Can algorithms use randomness (i.e., coin tossing) to speed up computation?

Chapter 7 introduces randomized computation and describes efficient *probabilistic algorithms* for certain tasks. But Chapters 19 and 20 show a surprising recent result giving strong evidence that randomness does *not* help speed up computation too much, in the sense that any probabilistic algorithm can be replaced with a *deterministic* algorithm (tossing no coins) that is almost as efficient.

3. Can hard problems become easier to solve if we allow the algorithms to err on a small number of inputs, or to only compute an *approximate* solution?

Average-case complexity and *approximation algorithms* are studied in Chapters 11, 18, 19, and 22. These chapters also show fascinating connections between these questions, the power of randomness, different notions of mathematical proofs, and the theory of error correcting codes.

4. Can we derive any practical benefit from computationally hard problems? For example, can we use them to construct cryptographic protocols that are *unbreakable* (at least by any plausible adversary)?

As described in Chapter 9, the security of digital cryptography is intimately related to the **P** vs. **NP** question (see Chapter 2) and average-case complexity (see Chapters 18).

5. Can we use the counterintuitive quantum mechanical properties of matter to build faster computers?

Chapter 10 describes the fascinating notion of *quantum computers* that use quantum mechanics to speed up certain computations. Peter Shor has shown that, if ever built, quantum computers will be able to factor integers efficiently (thus breaking many current cryptosystems). However, currently there are many daunting obstacles to actually building such computers,

6. Do we need people to prove mathematical theorems, or can we generate mathematical proofs automatically? Can we check a mathematical proof without reading it completely? Do interactive proofs, involving a dialog between prover and verifier, have more power than standard “static” mathematical proofs?

The notion of *proof*, central to mathematics, turns out to be central to computational complexity as well, and complexity has shed new light on the meaning of mathematical proofs. Whether mathematical proofs can be generated automatically turns out to depend on the **P** vs. **NP** question (see Chapter 2). Chapter 11 describes *probabilistically checkable proofs*. These are surprisingly robust mathematical proofs that can be checked by only reading them in very few probabilistically chosen locations, in contrast to the traditional proofs that require line-by-line verification. Along similar lines we introduce the notion of *interactive proofs* in Chapter 8 and use them to derive some surprising results. Finally, *proof complexity*, a subfield of complexity studying the minimal proof length of various statements, is studied in Chapter 15.

At roughly 40 years of age, Complexity theory is still an infant science and many important results are less than 20 years old. We have few complete answers for any of these questions. In a surprising twist, computational complexity has also been used to prove some

metamathematical theorems: they provide evidence of the difficulty of resolving some of the questions of . . . computational complexity; see Chapter 23.

We conclude with another quote from Hilbert's 1900 lecture:

Proofs of impossibility were effected by the ancients ... [and] in later mathematics, the question as to the impossibility of certain solutions plays a preminent part. ...

In other sciences also one meets old problems which have been settled in a manner most satisfactory and most useful to science by the proof of their impossibility. ... After seeking in vain for the construction of a perpetual motion machine, the relations were investigated which must subsist between the forces of nature if such a machine is to be impossible; and this inverted question led to the discovery of the law of the conservation of energy. ...

It is probably this important fact along with other philosophical reasons that gives rise to conviction ... that every definite mathematical problem must necessarily be susceptible to an exact settlement, either in the form of an actual answer to the question asked, or by the proof of the impossibility of its solution and therewith the necessary failure of all attempts. ... This conviction... is a powerful incentive to the worker. We hear within us the perpetual call: There is the problem. Seek its solution. You can find it by pure reason, for in mathematics there is no ignorance.

Main Theorems and Definitions

Definition 1.3: Computing a function and running time	19
Theorem 1.9: Efficient Universal Turing machine	23
Definition 1.13: The class \mathbf{P}	26
Definition 2.1: The class \mathbf{NP}	38
Definition 2.7: Reductions, \mathbf{NP} -hardness and \mathbf{NP} -completeness	41
Theorem 2.10: Cook-Levin Theorem [Coo71, Lev73]	43
Theorem 3.1: Time Hierarchy Theorem [HS65]	62
Theorem 3.2: Non-deterministic Time Hierarchy Theorem [Coo72]	62
Theorem 3.3: “ \mathbf{NP} intermediate” languages [Lad75]	64
Theorem 3.7: Baker, Gill, Solovay [BGS75]	66
Definition 4.1: Space-bounded computation	71
Theorem 4.8: Space Hierarchy Theorem [SHL65]	75
Theorem 4.13: TQBF is \mathbf{PSPACE} -complete [SM73]	76
Theorem 4.14: Savitch’s Theorem [Sav70]: $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{SPACE}(S(n)^2)$	77
Definition 4.16: logspace reduction and \mathbf{NL} -completeness	79
Theorem 4.18: PATH is \mathbf{NL} -complete	80
Theorem 4.20: Immerman-Szelepcsényi Theorem [Imm88, Sze87]: $\mathbf{NL} = \mathbf{coNL}$	82
Definition 5.3: Polynomial Hierarchy	87
Definition 5.7: Alternating time	89
Theorem 5.11: Time/Space tradeoff for SAT [For97a, FLvMV00]	90
Definition 6.1: Boolean circuits	96
Definition 6.5: The class $\mathbf{P}_{/poly}$	97
Theorem 6.6: $\mathbf{P} \subseteq \mathbf{P}_{/poly}$	98
Theorem 6.18: Polynomial-time TM’s with advice decide $\mathbf{P}_{/poly}$	101
Theorem 6.19: Karp-Lipton Theorem [KL80]	101
Theorem 6.20: Meyer’s Theorem [KL80]	102
Theorem 6.21: Existence of hard functions [Sha49a]	102
Theorem 6.22: Non-uniform hierarchy theorem	103

Theorem 6.27: NC and parallel algorithms.....	105
Definition 7.2: The classes BPTIME and BPP	111
Theorem 7.10: Error reduction for BPP	116
Theorem 7.14: $\mathbf{BPP} \subseteq \mathbf{P}_{/\text{poly}}$ [Adl78].....	119
Theorem 7.15: Sipser-Gács Theorem: $\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$	120
Definition 7.18: The classes BPL and RL	122
Definition 8.6: Probabilistic verifiers and the class IP	130
Theorem 8.12: Goldwasser-Sipser [GS87]: $\mathbf{IP}[k] \subseteq \mathbf{AM}[k+2]$	134
Definition 8.14: Pairwise independent hash functions.....	135
Theorem 8.19: IP in PSPACE [LFKN90, Sha90].....	139
Definition 9.3: Negligible functions.....	154
Definition 9.4: One way functions.....	155
Theorem 9.6: Encryption from one-way function.....	156
Definition 9.8: Secure pseudorandom generators.....	158
Theorem 9.9: Pseudorandom generators from one-way functions [HILL99].....	158
Theorem 9.11: Unpredictability implies pseudorandomness [Yao82a].....	159
Theorem 9.12: Goldreich-Levin Theorem [GL89].....	160
Definition 9.14: Zero knowledge proofs.....	164
Definition 10.6: quantum operation.....	183
Definition 10.9: Quantum Computation and the class BQP	186
Theorem 10.12: Universal basis for quantum operations [Deu89, Kit97].....	188
Theorem 10.13: Grover's Algorithm [Gro96].....	188
Theorem 10.14: Simon's Algorithm [Sim94].....	192
Theorem 10.15: Shor's Algorithm: Factoring in BQP [Sho97].....	194
Lemma 10.17: Quantum Fourier Transform [BV93].....	196
Definition 11.1: Approximation of MAX-3SAT.....	206
Definition 11.4: PCP verifier.....	209
Theorem 11.5: The PCP Theorem [AS92, ALM ⁺ 92].....	209
Theorem 11.9: PCP Theorem: Hardness of Approximation view.....	210
Definition 11.11: Constraint satisfaction problems (CSP).....	211
Theorem 11.19: Exponential-sized PCP system for NP [ALM ⁺ 92].....	215
Definition 12.1: Decision tree complexity.....	224
Definition 12.3: Certificate complexity.....	226
Definition 12.6: Randomized decision trees.....	227
Definition 13.1: Two party communication complexity.....	234

Theorem 13.4: Equality has linear communication complexity	235
Theorem 13.8: Tiling and communication complexity [AUY83]	236
Theorem 13.24: Lower bound for generalized inner product	241
Theorem 14.1: $\bigoplus \notin \mathbf{AC}^0$ [FSS81, Ajt83]	248
Lemma 14.2: Håstad's switching lemma [Hås86]	248
Theorem 14.4: Razborov-Smolensky [Raz87, Smo87]: $MOD_p \notin \mathbf{ACC}^0(q)$	251
Theorem 14.7: Monotone-circuit lower bound for CLIQUE [Raz85a, And85, AB87]	253
Theorem 15.3: Classical Interpolation Theorem	268
Theorem 15.4: Feasible Interpolation Theorem	269
Theorem 15.5: Exponential resolution lower bound	270
Definition 16.2: Algebraic straight-line program over \mathbb{F}	276
Definition 16.7: $\mathbf{AlgP}_{/\text{poly}}$, $\mathbf{AlgNP}_{/\text{poly}}$	279
Theorem 16.12: Completeness of determinant and permanent [Val79a]	281
Definition 16.15: Algebraic Computation Tree over \mathbb{R}	283
Definition 16.16: algebraic computation tree complexity	283
Theorem 16.19: Topological lower bound on algebraic tree complexity [BO83]	284
Definition 17.5: $\#\mathbf{P}$	298
Theorem 17.11: Valiant's Theorem [Val79b]: perm is $\#\mathbf{P}$ -complete	300
Theorem 17.14: Toda's Theorem [Tod91]: $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{SAT}}$	305
Definition 17.16: \bigoplus quantifier and $\bigoplus \mathbf{SAT}$	305
Lemma 17.17: Randomized reduction from \mathbf{PH} to $\bigoplus \mathbf{SAT}$	305
Theorem 17.18: Valiant-Vazirani Theorem [VV86]	306
Definition 18.1: Distributional problem	314
Definition 18.4: Polynomial on average and \mathbf{distP}	316
Definition 18.5: The class \mathbf{distNP}	317
Definition 18.6: Average-case reduction	317
Theorem 18.8: Existence of a \mathbf{distNP} -complete problem [Lev86]	318
Definition 19.1: Average-case and worst-case hardness	325
Theorem 19.2: Yao's XOR Lemma [Yao82a]	325
Definition 19.5: Error Correcting Codes	329
Theorem 19.15: Unique decoding for Reed-Solomon [BW86]	333
Definition 19.16: Local decoder	335
Theorem 19.17: Hardness amplification from local decoding	336
Theorem 19.21: Worst-case hardness to mild hardness	339
Theorem 19.24: List decoding for the Reed-Solomon code [Sud96]	341

Theorem 19.27: Worst-case hardness to strong hardness	344
Definition 20.2: Pseudorandom generators	350
Theorem 20.6: PRGs from average-case hardness	353
Theorem 20.7: Derandomization under worst-case assumptions	353
Definition 20.12: NW Generator	356
Theorem 20.16: Uniform derandomization [IW98]	358
Theorem 20.17: Derandomization implies lower bounds [KI03]	360
Definition 21.2: The parameter $\lambda(G)$	367
Definition 21.6: (n, d, λ) -expander graphs	370
Definition 21.7: Combinatorial (edge) expansion	370
Theorem 21.9: Combinatorial vs. algebraic expansion	371
Theorem 21.12: Expander walks	374
Theorem 21.15: Expander Chernoff Bound	375
Theorem 21.19: Explicit construction of expanders	379
Theorem 21.21: Reingold's Theorem: $\text{UPATH} \in \mathbf{L}$	380
Definition 21.24: Randomness extractors	384
Theorem 21.28: Constructive version of Theorem 20.7	387
Theorem 21.31: Nisan's pseudorandom generator [Nis90]	388
Lemma 22.4: PCP Main Lemma	398
Lemma 22.5: Gap Amplification [Din06]	399
Lemma 22.6: Alphabet Reduction	399
Theorem 22.16: Håstad's 3-bit PCP [Hås97]	409
Theorem 23.1: Natural proofs [RR94]	430
Theorem 23.8: PromiseMA $\not\subseteq$ SIZE (n^c) [San07]	435