

Contents (detailed)

- Preface** **19**
 - 0.1 To the student 21
 - 0.1.1 Is the effort worth it? 22
 - 0.2 To potential instructors 22
 - 0.3 Acknowledgements 25

- Preliminaries** **27**

- 0 Introduction** **29**
 - 0.1 Extended Example: A faster way to multiply 32
 - 0.1.1 Beyond Karatsuba’s algorithm 35
 - 0.2 Algorithms beyond arithmetic 37
 - 0.3 On the importance of negative results. 38
 - 0.4 Roadmap to the rest of this book 39
 - 0.4.1 Dependencies between chapters 40
 - 0.5 Exercises 41
 - 0.6 Bibliographical notes 43

- 1 Mathematical Background** **45**
 - 1.1 A mathematician’s apology 45
 - 1.2 This chapter: a reader’s manual 46
 - 1.3 A quick overview of mathematical prerequisites 46
 - 1.4 Reading mathematical texts 47
 - 1.4.1 Definitions 48
 - 1.4.2 Assertions: Theorems, lemmas, claims 50
 - 1.4.3 Proofs 50
 - 1.4.4 Example: Defining a one to one function 51
 - 1.5 Basic discrete math objects 52
 - 1.5.1 Sets 52
 - 1.5.2 Sets in Python (optional) 53
 - 1.5.3 Special sets 54
 - 1.5.4 Functions 55
 - 1.5.5 Graphs 58
 - 1.5.6 Logic operators and quantifiers. 60

1.5.7	Quantifiers for summations and products	61
1.5.8	Parsing formulas: bound and free variables	61
1.5.9	Asymptotics and Big- <i>O</i> notation	63
1.5.10	Some “rules of thumb” for Big- <i>O</i> notation	65
1.6	Proofs	65
1.6.1	Proofs and programs	66
1.6.2	Proof writing style	66
1.6.3	Patterns in proofs	67
1.7	Extended example: Topological Sorting	69
1.7.1	Mathematical induction	71
1.7.2	Proving the result by induction	72
1.7.3	Minimality and uniqueness	74
1.8	Notation and conventions	76
1.8.1	Conventions	77
1.9	Exercises	79
1.10	Bibliographical notes	81
2	Computation and Representation	83
2.1	Defining representation	84
2.1.1	Representing natural numbers.	85
2.1.2	Implementing the representation in python:	87
2.1.3	Meaning of representation	88
2.2	Representing more objects	89
2.2.1	Representing (potentially negative) integers	89
2.2.2	Representing rational numbers	90
2.3	Representing real numbers	91
2.3.1	Can we represent reals <i>exactly</i> ?	92
2.4	Beyond numbers	96
2.4.1	Finite representations	96
2.4.2	Prefix-free encoding	97
2.4.3	Making representations prefix-free	99
2.4.4	“Proof by Python” (optional)	100
2.4.5	Representing letters and text	103
2.4.6	Representing vectors, matrices, images	106
2.4.7	Representing graphs	106
2.4.8	Representing lists	107
2.4.9	Notation	107
2.5	Defining computational tasks	108
2.5.1	Distinguish functions from programs!	110
2.6	Exercises	112
2.7	Bibliographical notes	116

I	Finite computation	119
3	Defining computation	121
3.1	Defining computation	121
3.2	Boolean formulas with AND, OR, and NOT.	123
3.2.1	Extended example: Computing XOR from AND,OR,NOT.	125
3.2.2	Informally defining “basic operations” and “algorithms”	127
3.3	Physical implementations of computing devices.	129
3.3.1	Transistors	129
3.3.2	Logical gates from transistors	130
3.3.3	Biological computing	130
3.3.4	Cellular automata and the game of life	130
3.3.5	Neural networks	131
3.3.6	The marble computer	131
3.4	Boolean Circuits	132
3.5	Equivalence of circuits and straight-line programs	133
3.5.1	“Proof by Python”	136
3.6	The NAND function	138
3.6.1	NAND Circuits	139
3.6.2	More examples of NAND circuits (optional)	141
3.6.3	The NAND-CIRC Programming language	143
3.7	Equivalence of all these models	146
3.7.1	Circuits with other gate sets (optional)	147
3.8	Exercises	148
3.9	Biographical notes	149
4	Syntactic sugar, and computing every function	151
4.1	Some examples syntactic sugar	152
4.1.1	Constants	152
4.1.2	Functions / Macros	152
4.1.3	Example: Computing Majority via NAND’s	153
4.1.4	Conditional statements	153
4.2	Extended example: Addition and Multiplicatoin (op- tional)	155
4.3	The LOOKUP function	156
4.3.1	Constructing a NAND-CIRC program for LOOKUP	157
4.4	Computing <i>every</i> function	159
4.4.1	Proof of NAND’s Universality	160
4.4.2	Improving by a factor of n (optional)	162
4.5	The class $SIZE_{n,m}(T)$	163
4.6	Exercises	165

4.7	Bibliographical notes	167
5	Code as data, data as code	169
5.1	A NAND interpreter in NAND	170
5.1.1	Efficient universal programs	171
5.1.2	Concrete representation for NAND-CIRC programs	172
5.1.3	A NAND interpreter in “pseudocode”	173
5.1.4	A NAND interpreter in Python	175
5.1.5	Constructing the NAND interpreter in NAND	176
5.2	A Python interpreter in NAND (discussion)	178
5.3	Counting programs, and lower bounds on the size of NAND-CIRC programs	179
5.4	Size hierarchy theorem (advanced, optional)	182
5.5	The physical extended Church-Turing thesis (discussion)	183
5.5.1	Attempts at refuting the PECTT	186
5.6	Finite computation recap	191
5.7	Exercises	191
5.8	Bibliographical notes	193
II	Uniform computation	195
6	Loops and infinity	197
6.1	Turing Machines	199
6.1.1	Example: A Turing machine for palindromes	200
6.1.2	Turing machines: a formal definition	202
6.1.3	Computable functions	204
6.1.4	Infinite loops and partial functions	205
6.2	Turing machines as programming languages	206
6.2.1	The NAND-TM Programming language	207
6.2.2	Sneak peak: NAND-TM vs Turing machines	209
6.2.3	Examples	210
6.3	Equivalence of Turing machines and NAND-TM programs	212
6.3.1	Specification vs implementation (again)	215
6.4	NAND-TM syntactic sugar	215
6.4.1	Well formed programs: The NAND-TM style manual	217
6.5	Uniformity, and NAND vs NAND-TM (discussion)	221
6.6	Exercises	222
6.7	Bibliographical notes	223
7	Equivalent models of computation	225

7.1	RAM machines and NAND-RAM	225
7.2	The gory details (optional)	228
7.2.1	Indexed access in NAND-TM	228
7.2.2	Two dimensional arrays in NAND-TM	230
7.2.3	All the rest	230
7.3	Turing equivalence (discussion)	230
7.3.1	The “Best of both worlds” paradigm	232
7.3.2	Let’s talk about abstractions.	232
7.3.3	Definition of “Algorithm”	234
7.4	Lambda calculus and functional programming lan- guages	235
7.4.1	Applying functions to functions	236
7.4.2	Obtaining multi-argument functions via Currying	236
7.4.3	Formal description of the λ calculus.	237
7.4.4	Functions as first class objects	239
7.5	The “Enhanced” λ calculus	239
7.5.1	Enhanced λ expressions	242
7.5.2	Enhanced λ calculus is Turing-complete	243
7.6	The pure λ calculus	245
7.6.1	List processing	247
7.6.2	The Y combinator, or recursion without recursion	248
7.6.3	Infinite loops in the λ calculus	251
7.7	More Turing-complete computational models	251
7.7.1	Parallel algorithms and cloud computing	251
7.7.2	Game of life, tiling and cellular automata	252
7.7.3	Configurations of Turing machines and one dimensional cellular automata	252
7.7.4	Turing completeness and equivalence, a formal definition (optional)	256
7.8	The Church-Turing Thesis (discussion)	257
7.8.1	Different models of computation	258
7.9	Exercises	259
7.10	Bibliographical notes	259
8	Universality and uncomputability	261
8.1	Universality or a self-circular evaluator	262
8.2	Is every function computable?	267
8.3	The Halting problem	269
8.3.1	Is the Halting problem really hard? (discussion)	271
8.3.2	A direct proof of the uncomputability of <i>HALT</i> (optional)	272
8.4	Reductions	274
8.4.1	Example: Halting on the zero problem	276

8.5	Rice's Theorem and the impossibility of general software verification	278
8.5.1	Rice's Theorem	280
8.5.2	Halting and Rice's Theorem for other Turing-complete models	284
8.5.3	Is software verification doomed? (discussion)	285
8.6	Exercises	287
8.7	Bibliographical notes	289
9	Restricted computational models	291
9.1	Turing completeness as a bug	291
9.2	Regular expressions	293
9.3	Deterministic finite automata, and efficient matching of regular expressions (optional)	298
9.3.1	Matching regular expressions using constant memory	301
9.3.2	Deterministic Finite Automata	302
9.3.3	Regular functions are closed under complement	305
9.4	Limitations of regular expressions	305
9.5	Other semantic properties of regular expressions	309
9.6	Context free grammars	311
9.6.1	Context-free grammars as a computational model	313
9.6.2	The power of context free grammars	315
9.6.3	Limitations of context-free grammars (optional)	317
9.7	Semantic properties of context free languages	318
9.7.1	Uncomputability of context-free grammar equivalence (optional)	319
9.8	Summary of semantic properties for regular expressions and context-free grammars	322
9.9	Exercises	323
9.10	Bibliographical notes	324
10	Is every theorem provable?	325
10.1	Hilbert's Program and Gödel's Incompleteness Theorem	326
10.2	Quantified integer statements	330
10.3	Diophantine equations and the MRDP Theorem	332
10.4	Hardness of quantified integer statements	333
10.4.1	Step 1: Quantified mixed statements and computation histories	334
10.4.2	Step 2: Reducing mixed statements to integer statements	337
10.5	Exercises	339
10.6	Bibliographical notes	340

III Efficient algorithms	343
11 Efficient computation	345
11.1 Problems on graphs	346
11.1.1 Finding the shortest path in a graph	347
11.1.2 Finding the longest path in a graph	349
11.1.3 Finding the minimum cut in a graph	350
11.1.4 Finding the maximum cut in a graph	353
11.1.5 A note on convexity	353
11.2 Beyond graphs	354
11.2.1 The 2SAT problem	354
11.2.2 The 3SAT problem	355
11.2.3 Solving linear equations	355
11.2.4 Solving quadratic equations	356
11.3 More advanced examples	357
11.3.1 Determinant of a matrix	357
11.3.2 The permanent (mod 2) problem	358
11.3.3 The permanent (mod 3) problem	358
11.3.4 Finding a zero-sum equilibrium	359
11.3.5 Finding a Nash equilibrium	359
11.3.6 Primal testing	359
11.3.7 Integer factoring	360
11.4 Our current knowledge	360
11.5 Lecture summary	361
11.6 Exercises	361
11.7 Bibliographical notes	361
11.8 Further explorations	362
11.9 Acknowledgements	362
12 Modeling running time	363
12.1 Formally defining running time	364
12.1.1 Nice time bounds	365
12.1.2 Non-boolean and partial functions (optional)	367
12.2 Efficient simulation of RAM machines: NAND-RAM vs NAND-TM	368
12.3 Efficient universal machine: a NAND-RAM inter- preter in NAND-RAM	371
12.4 Time hierarchy theorem	375
12.5 Unrolling the loop: Uniform vs non uniform computa- tion	377
12.5.1 Algorithmic transformation of NAND-TM to NAND and “Proof by Python” (optional)	380
12.5.2 The class \mathbf{P}_{poly}	383
12.5.3 Simulating NAND with NAND-TM?	384

12.5.4	Uniform vs. Nonuniform computation: A recap	385
12.6	Extended Church-Turing Thesis	386
12.7	Exercises	388
12.8	Bibliographical notes	389
12.9	Further explorations	389
12.10	Acknowledgements	389
13	Polynomial-time reductions	391
13.0.1	Decision problems	392
13.1	Reductions	392
13.2	Some example reductions	394
13.2.1	Reducing 3SAT to quadratic equations	394
13.3	The independent set problem	397
13.4	Reducing Independent Set to Maximum Cut	399
13.5	Reducing 3SAT to Longest Path	401
13.6	Exercises	402
13.7	Bibliographical notes	403
13.8	Further explorations	403
13.9	Acknowledgements	403
14	NP, NP completeness, and the Cook-Levin Theorem	405
14.1	The class NP	405
14.1.1	Examples of NP functions	407
14.1.2	Basic facts about NP	409
14.2	From NP to 3SAT: The Cook-Levin Theorem	411
14.2.1	What does this mean?	412
14.2.2	The Cook-Levin Theorem: Proof outline	413
14.3	The <i>NANDSAT</i> Problem, and why it is NP hard.	414
14.4	The <i>3NAND</i> problem	416
14.5	From <i>3NAND</i> to <i>3SAT</i>	418
14.6	Wrapping up	419
14.7	Exercises	419
14.8	Bibliographical notes	420
14.9	Further explorations	420
14.10	Acknowledgements	420
15	What if P equals NP?	421
15.1	Search-to-decision reduction	422
15.2	Optimization	424
15.2.1	Example: Supervised learning	428
15.2.2	Example: Breaking cryptosystems	428
15.3	Finding mathematical proofs	429
15.4	Quantifier elimination (advanced)	430
15.4.1	Application: self improving algorithm for <i>3SAT</i>	433
15.5	Approximating counting problems (advanced, optional)	433

15.6	What does all of this imply?	435
15.7	Can $P \neq NP$ be neither true nor false?	437
15.8	Is $P = NP$ “in practice”?	438
15.9	What if $P \neq NP$?	439
15.10	Exercises	440
15.11	Bibliographical notes	441
15.12	Further explorations	441
15.13	Acknowledgements	441
16	Space bounded computation	443
16.1	Lecture summary	443
16.2	Exercises	443
16.3	Bibliographical notes	443
16.4	Further explorations	443
16.5	Acknowledgements	443
IV	Randomized computation	445
17	Probability Theory 101	447
17.1	Random coins	447
17.1.1	Random variables	450
17.1.2	Distributions over strings	451
17.1.3	More general sample spaces.	452
17.2	Correlations and independence	452
17.2.1	Independent random variables	454
17.2.2	Collections of independent random variables.	455
17.3	Concentration	456
17.3.1	Chebyshev’s Inequality	457
17.3.2	The Chernoff bound	458
17.4	Lecture summary	459
17.5	Exercises	459
17.6	Bibliographical notes	462
18	Probabilistic computation	463
18.1	Finding approximately good maximum cuts.	464
18.1.1	Amplification	465
18.1.2	Two-sided amplification	467
18.1.3	What does this mean?	467
18.1.4	Solving SAT through randomization	468
18.1.5	Bipartite matching.	470
18.2	Exercises	473
18.3	Bibliographical notes	473
18.4	Further explorations	474
18.5	Acknowledgements	474

19 Modeling randomized computation	475
19.0.1 An alternative view: random coins as an “extra input”	476
19.0.2 Amplification	478
19.1 BPP and NP completeness	479
19.2 The power of randomization	481
19.2.1 Solving BPP in exponential time	481
19.2.2 Simulating randomized algorithms by circuits or straight-line programs.	481
19.3 Derandomization	483
19.3.1 Pseudorandom generators	484
19.3.2 From existence to constructivity	486
19.3.3 Usefulness of pseudorandom generators	487
19.4 P = NP and BPP vs P	488
19.5 Non-constructive existence of pseudorandom generators (advanced, optional)	491
19.6 Exercises	494
19.7 Bibliographical notes	494
19.8 Further explorations	494
19.9 Acknowledgements	494
V Advanced topics	495
20 Cryptography	497
20.1 Classical cryptosystems	498
20.2 Defining encryption	500
20.3 Defining security of encryption	501
20.4 Perfect secrecy	503
20.4.1 Example: Perfect secrecy in the battlefield	504
20.4.2 Constructing perfectly secret encryption	505
20.5 Necessity of long keys	506
20.6 Computational secrecy	508
20.6.1 Stream ciphers or the “derandomized one-time pad”	510
20.7 Computational secrecy and NP	512
20.8 Public key cryptography	514
20.8.1 Defining public key encryption	516
20.8.2 Diffie-Hellman key exchange	517
20.9 Other security notions	518
20.10 Magic	519
20.10.1 Zero knowledge proofs	519
20.10.2 Fully homomorphic encryption	519
20.10.3 Multiparty secure computation	520

20.11	Exercises	521
20.12	Bibliographical notes	521
20.13	Further explorations	522
20.14	Acknowledgements	522
21	Proofs and algorithms	523
21.1	Lecture summary	523
21.2	Exercises	523
21.3	Bibliographical notes	523
21.4	Further explorations	523
21.5	Acknowledgements	524
22	Quantum computing	525
22.1	The double slit experiment	526
22.2	Quantum amplitudes	526
22.3	Bell's Inequality	529
22.4	Quantum weirdness	530
22.5	Quantum computing and computation - an executive summary.	531
22.6	Quantum systems	533
22.6.1	Quantum amplitudes	535
22.6.2	Recap	536
22.7	Analysis of Bell's Inequality (optional)	536
22.8	Quantum computation	538
22.8.1	Quantum circuits	539
22.8.2	Q NAND-CIRC programs (optional)	542
22.8.3	Uniform computation	542
22.9	Physically realizing quantum computation	543
22.10	Shor's Algorithm: Hearing the shape of prime factors	544
22.10.1	Period finding	545
22.10.2	Shor's Algorithm: A bird's eye view	545
22.11	Quantum Fourier Transform (advanced, optional)	548
22.11.1	Quantum Fourier Transform over the Boolean Cube: Simon's Algorithm	549
22.11.2	From Fourier to Period finding: Simon's Algo- rithm (advanced, optional)	551
22.11.3	From Simon to Shor (advanced, optional)	551
22.12	Exercises	553
22.13	Bibliographical notes	553
22.14	Further explorations	554
22.15	Acknowledgements	554
VI	Appendices	555

