

Contents (detailed)

Preface	19
0.1 To the student	21
0.1.1 Is the effort worth it?	21
0.2 To potential instructors	22
0.3 Acknowledgements	24
Preliminaries	25
0 Introduction	27
0.1 Extended Example: A faster way to multiply	30
0.1.1 Beyond Karatsuba’s algorithm	34
0.2 Algorithms beyond arithmetic	36
0.3 On the importance of negative results.	37
0.4 Roadmap to the rest of this course	38
0.4.1 Dependencies between chapters	39
0.5 Exercises	40
0.6 Bibliographical notes	42
0.7 Further explorations	42
1 Mathematical Background	45
1.1 A mathematician’s apology	45
1.2 A quick overview of mathematical prerequisites	46
1.3 Reading mathematical texts	48
1.3.1 Example: Defining a one to one function	50
1.4 Basic discrete math objects	52
1.4.1 Sets	52
1.4.2 Sets in Python (optional)	53
1.4.3 Special sets	54
1.4.4 Functions	55
1.4.5 Graphs	58
1.4.6 Logic operators and quantifiers.	60
1.4.7 Quantifiers for summations and products	61
1.4.8 Parsing formulas: bound and free variables	61
1.4.9 Asymptotics and Big- <i>O</i> notation	63

1.4.10	Some “rules of thumbs” for Big- O notation	65
1.5	Proofs	65
1.5.1	Proofs and programs	66
1.6	Extended example: graph connectivity	66
1.6.1	Mathematical induction	68
1.6.2	Proving the theorem by induction	69
1.6.3	Writing down the proof	71
1.7	Proof writing style	74
1.7.1	Patterns in proofs	74
1.8	Non-standard notation	76
1.9	Exercises	78
1.10	Bibliographical notes	80
2	Computation and Representation	81
2.1	Examples of binary representations	83
2.1.1	Representing natural numbers	83
2.1.2	Representing (potentially negative) integers	85
2.1.3	Representing rational numbers	86
2.2	Representing real numbers	88
2.2.1	Can we represent reals <i>exactly</i> ?	88
2.3	Beyond numbers	92
2.3.1	Finite representations	93
2.3.2	Prefix free encoding	94
2.3.3	Making representations prefix free	95
2.3.4	“Proof by Python” (optional)	96
2.3.5	Representing letters and text	98
2.3.6	Representing vectors, matrices, images	101
2.3.7	Representing graphs	102
2.3.8	Representing lists	103
2.3.9	Notation	103
2.4	Defining computational tasks	104
2.4.1	Distinguish functions from programs	106
2.4.2	Advanced note: beyond computing functions . . .	107
2.5	Exercises	108
2.6	Bibliographical notes	111
2.7	Further explorations	111
I	Finite computation	113
3	Defining computation	115
3.1	Defining computation	117
3.1.1	Defining “elementary operations”	119
3.1.2	The NAND function	122
3.2	Informally defining “basic operations” and “algorithms”	124

3.3	From NAND to infinity and beyond..	126
3.3.1	NAND Circuits	126
3.4	Physical implementations of computing devices.	131
3.4.1	Transistors and physical logic gates	131
3.4.2	NAND gates from transistors	135
3.5	Basing computing on other media (optional)	135
3.5.1	Biological computing	135
3.5.2	Cellular automata and the game of life	136
3.5.3	Neural networks	136
3.5.4	The marble computer	137
3.6	The NAND Programming language	137
3.6.1	NAND programs and NAND circuits	139
3.7	Exercises	142
3.8	Biographical notes	143
3.9	Further explorations	143
4	Syntactic sugar, and computing every function	145
4.1	Some useful syntactic sugar	146
4.1.1	Constants	146
4.1.2	Functions / Macros	147
4.1.3	Example: Computing Majority via NAND's	147
4.1.4	Conditional statements	148
4.1.5	Bounded loops	149
4.1.6	Example: Adding two integers	149
4.2	Even more sugar (optional)	152
4.2.1	More indices	152
4.2.2	Non-Boolean variables, lists and integers	152
4.2.3	Storing integers	153
4.2.4	Example: Multiplying n bit numbers	153
4.3	Functions beyond arithmetic and LOOKUP	155
4.3.1	Constructing a NAND program for <i>LOOKUP</i>	155
4.4	Computing <i>every</i> function	157
4.4.1	Proof of NAND's Universality	158
4.4.2	Improving by a factor of n (optional)	160
4.4.3	The class $SIZE_{n,m}(T)$	161
4.5	Exercises	163
4.6	Bibliographical notes	164
4.7	Further explorations	164
5	Code as data, data as code	165
5.1	A NAND interpreter in NAND	166
5.1.1	Concrete representation for NAND programs	168
5.1.2	Representing a program as a string	169
5.1.3	A NAND interpreter in "pseudocode"	170

5.1.4	A NAND interpreter in Python	171
5.1.5	Constructing the NAND interpreter in NAND	172
5.2	A Python interpreter in NAND (discussion)	174
5.3	Counting programs, and lower bounds on the size of NAND programs	175
5.4	The physical extended Church-Turing thesis (discussion)	179
5.4.1	Attempts at refuting the PECTT	181
5.5	Exercises	185
5.6	Bibliographical notes	186
5.7	Further explorations	186
II Uniform computation		189
6	Loops and infinity	191
6.1	The NAND++ Programming language	193
6.1.1	Enhanced NAND++ programs	194
6.1.2	“Oblivious” / “Vanilla” NAND++	196
6.2	Computable functions	198
6.3	Equivalence of “vanilla” and “enhanced” NAND++	200
6.3.1	Simulating NAND++ programs by enhanced NAND++ programs.	201
6.3.2	Simulating enhanced NAND++ programs by NAND++ programs.	202
6.3.3	Another application of GOTO: well formed programs	206
6.4	Turing Machines	209
6.5	Uniformity, and NAND vs NAND++ (discussion)	216
6.6	Exercises	218
6.7	Bibliographical notes	218
6.8	Further explorations	219
6.9	Acknowledgements	219
7	Equivalent models of computation	221
7.1	RAM machines and NAND \llcorner	221
7.1.1	Indexed access in NAND++	223
7.1.2	Two dimensional arrays in NAND++	225
7.1.3	All the rest	226
7.1.4	Turing equivalence (discussion)	226
7.2	The “Best of both worlds” paradigm (discussion)	227
7.2.1	Let’s talk about abstractions.	228
7.3	Lambda calculus and functional programming lan- guages	230
7.3.1	Formal description of the λ calculus.	233
7.3.2	Functions as first class objects	235

7.3.3	“Enhanced” lambda calculus	235
7.3.4	How basic is “basic”?	240
7.3.5	List processing	242
7.3.6	Recursion without recursion	242
7.4	Other models	246
7.4.1	Parallel algorithms and cloud computing	246
7.4.2	Game of life, tiling and cellular automata	246
7.4.3	Configurations of NAND++/Turing machines and one dimensional cellular automata	247
7.5	Turing completeness and equivalence, a formal defini- tion (optional)	251
7.6	The Church-Turing Thesis (discussion)	252
7.7	Our models vs other texts	253
7.8	Exercises	253
7.9	Bibliographical notes	254
7.10	Further explorations	254
7.11	Acknowledgements	254
8	Universality and uncomputability	255
8.1	Universality: A NAND++ interpreter in NAND++	256
8.2	Is every function computable?	259
8.3	The Halting problem	261
8.3.1	Is the Halting problem really hard? (discussion)	263
8.3.2	Reductions	264
8.3.3	A direct proof of the uncomputability of <i>HALT</i> (optional)	265
8.4	Impossibility of general software verification	268
8.4.1	Rice’s Theorem	270
8.4.2	Halting and Rice’s Theorem for other Turing- complete models	273
8.4.3	Is software verification doomed? (discussion)	275
8.5	Exercises	276
8.6	Bibliographical notes	277
8.7	Further explorations	277
8.8	Acknowledgements	277
9	Restricted computational models	279
9.1	Turing completeness as a bug	279
9.2	Regular expressions	281
9.2.1	Efficient matching of regular expressions (ad- vanced, optional)	284
9.3	Limitations of regular expressions	287
9.4	Other semantic properties of regular expressions	291
9.5	Context free grammars	293

9.5.1	Context-free grammars as a computational model	295
9.5.2	The power of context free grammars	297
9.5.3	Limitations of context-free grammars (optional) .	298
9.6	Semantic properties of context free languages	300
9.6.1	Uncomputability of context-free grammar equivalence (optional)	301
9.7	Summary of semantic properties for regular expres- sions and context-free grammars	302
9.8	Exercises	304
9.9	Bibliographical notes	304
9.10	Further explorations	304
9.11	Acknowledgements	304
10	Is every theorem provable?	305
10.1	Hilbert's Program and Gödel's Incompleteness Theorem	306
10.2	Quantified integer statements	310
10.3	Diophantine equations and the MRDP Theorem	312
10.4	Hardness of quantified integer statements	313
10.4.1	Step 1: Quantified mixed statements and com- putation histories	314
10.4.2	Step 2: Reducing mixed statements to integer statements	317
10.5	Exercises	318
10.6	Bibliographical notes	319
10.7	Further explorations	319
10.8	Acknowledgements	319
III	Efficient algorithms	321
11	Efficient computation	323
11.1	Problems on graphs	324
11.1.1	Finding the shortest path in a graph	325
11.1.2	Finding the longest path in a graph	326
11.1.3	Finding the minimum cut in a graph	327
11.1.4	Finding the maximum cut in a graph	330
11.1.5	A note on convexity	330
11.2	Beyond graphs	332
11.2.1	The 2SAT problem	332
11.2.2	The 3SAT problem	333
11.2.3	Solving linear equations	333
11.2.4	Solving quadratic equations	334
11.3	More advanced examples	334
11.3.1	The permanent (mod 2) problem	335
11.3.2	The permanent (mod 3) problem	335

11.3.3	Finding a zero-sum equilibrium	335
11.3.4	Finding a Nash equilibrium	336
11.3.5	Primality testing	336
11.3.6	Integer factoring	337
11.4	Our current knowledge	337
11.5	Lecture summary	338
11.6	Exercises	338
11.7	Bibliographical notes	339
11.8	Further explorations	339
11.9	Acknowledgements	339
12	Modeling running time	341
12.1	NAND \ll vs NAND++	344
12.2	Efficient universal machine: a NAND \ll interpreter in NAND \ll	346
12.3	Time hierarchy theorem	348
12.4	Uniform vs non uniform computation	350
12.4.1	The class P_{poly}	353
12.4.2	Simulating NAND with NAND++?	354
12.4.3	Uniform vs. Nonuniform computation: A recap	355
12.5	Extended Church-Turing Thesis	356
12.6	Lecture summary	357
12.7	Exercises	358
12.8	Bibliographical notes	358
12.9	Further explorations	358
12.10	Acknowledgements	358
13	Polynomial-time reductions	359
13.0.1	Decision problems	360
13.1	Reductions	360
13.2	Some example reductions	361
13.2.1	Reducing 3SAT to quadratic equations	362
13.3	The independent set problem	364
13.4	Reducing Independent Set to Maximum Cut	366
13.5	Reducing 3SAT to Longest Path	368
13.6	Exercises	370
13.7	Bibliographical notes	370
13.8	Further explorations	370
13.9	Acknowledgements	370
14	NP, NP completeness, and the Cook-Levin Theorem	371
14.1	The class NP	371
14.1.1	Examples:	373
14.1.2	From NP to 3SAT	373
14.1.3	What does this mean?	374

14.2	The Cook-Levin Theorem	375
14.2.1	The <i>NANDSAT</i> Problem, and why it is NP hard.	377
14.2.2	The <i>3NAND</i> problem	379
14.2.3	From <i>3NAND</i> to <i>3SAT</i>	380
14.2.4	Wrapping up	382
14.3	Exercises	382
14.4	Bibliographical notes	383
14.5	Further explorations	383
14.6	Acknowledgements	383
15	What if P equals NP?	385
15.1	Search-to-decision reduction	386
15.2	Optimization	388
15.2.1	Example: Supervised learning	390
15.2.2	Example: Breaking cryptosystems	391
15.3	Finding mathematical proofs	391
15.4	Quantifier elimination	393
15.4.1	Approximating counting problems	394
15.5	What does all of this imply?	394
15.6	Can $P \neq NP$ be neither true nor false?	396
15.7	Is $P = NP$ “in practice”?	397
15.8	What if $P \neq NP$?	398
15.9	Exercises	400
15.10	Bibliographical notes	400
15.11	Further explorations	400
15.12	Acknowledgements	400
16	Space bounded computation	401
16.1	Lecture summary	401
16.2	Exercises	401
16.3	Bibliographical notes	401
16.4	Further explorations	401
16.5	Acknowledgements	401
IV	Randomized computation	403
17	Probability Theory 101	405
17.1	Random coins	406
17.1.1	Random variables	408
17.1.2	Distributions over strings	410
17.1.3	More general sample spaces.	411
17.2	Correlations and independence	411
17.2.1	Independent random variables	413
17.2.2	Collections of independent random variables.	415

17.3	Concentration	416
17.3.1	Chebyshev's Inequality	418
17.3.2	The Chernoff bound	419
17.4	Lecture summary	420
17.5	Exercises	420
17.6	Bibliographical notes	422
17.7	Further explorations	422
17.8	Acknowledgements	422
18	Probabilistic computation	423
18.1	Finding approximately good maximum cuts.	424
18.1.1	Amplification	425
18.1.2	Two-sided amplification	426
18.1.3	What does this mean?	426
18.1.4	Solving SAT through randomization	427
18.1.5	Bipartite matching.	429
18.2	Lecture summary	432
18.3	Exercises	433
18.4	Bibliographical notes	433
18.5	Further explorations	434
18.6	Acknowledgements	434
19	Modeling randomized computation	435
19.0.1	Random coins as an "extra input"	437
19.0.2	Amplification	439
19.1	BPP and NP completeness	439
19.2	The power of randomization	441
19.2.1	Solving BPP in exponential time	441
19.2.2	Simulating randomized algorithms by circuits or straightline programs.	442
19.3	Derandomization	444
19.3.1	Pseudorandom generators	445
19.3.2	From existence to constructivity	447
19.3.3	Usefulness of pseudorandom generators	449
19.4	P = NP and BPP vs P	450
19.5	Non-constructive existence of pseudorandom generators	450
19.6	Lecture summary	452
19.7	Exercises	452
19.8	Bibliographical notes	452
19.9	Further explorations	452
19.10	Acknowledgements	452

V	Advanced topics	453
20	Cryptography	455
20.1	Classical cryptosystems	456
20.2	Defining encryption	457
20.3	Defining security of encryption	458
20.4	Perfect secrecy	459
20.4.1	Example: Perfect secrecy in the battlefield	461
20.4.2	Constructing perfectly secret encryption	461
20.5	Necessity of long keys	463
20.6	Computational secrecy	466
20.6.1	Stream ciphers or the “derandomized one-time pad”	467
20.7	Computational secrecy and NP	470
20.8	Public key cryptography	472
20.8.1	Public key encryption, trapdoor functions and pseudorandom generators	474
20.9	Other security notions	477
20.10	Magic	477
20.10.1	Zero knowledge proofs	477
20.10.2	Fully homomorphic encryption	478
20.10.3	Multiparty secure computation	478
20.11	Lecture summary	479
20.12	Exercises	479
20.13	Bibliographical notes	479
20.14	Further explorations	480
20.15	Acknowledgements	480
21	Proofs and algorithms	481
21.1	Lecture summary	481
21.2	Exercises	481
21.3	Bibliographical notes	481
21.4	Further explorations	481
21.5	Acknowledgements	482
22	Quantum computing	483
22.1	The double slit experiment	484
22.2	Quantum amplitudes	485
22.3	Bell’s Inequality	488
22.4	Quantum weirdness	489
22.5	Quantum computing and computation - an executive summary.	490
22.6	Quantum systems	492
22.6.1	Quantum amplitudes	493
22.6.2	Recap	494

22.7	Analysis of Bell's Inequality (optional)	495
22.8	Quantum computation	496
22.8.1	Quantum circuits	497
22.8.2	QNAND programs (optional)	500
22.8.3	Uniform computation	501
22.9	Physically realizing quantum computation	502
22.10	Shor's Algorithm: Hearing the shape of prime factors	503
22.10.1	Period finding	503
22.10.2	Shor's Algorithm: A bird's eye view	505
22.11	Quantum Fourier Transform (advanced, optional)	507
22.11.1	Quantum Fourier Transform over the Boolean Cube: Simon's Algorithm	508
22.12	Exercises	511
22.13	Bibliographical notes	512
22.14	Further explorations	512
22.15	Acknowledgements	512
VI	Appendices	513
A	The NAND Programming Language	515
B	The NAND++ Programming Language	547
C	The Lambda Calculus	559

