

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	27
0 Introduction	29
0.1 Extended Example: A faster way to multiply	32
0.1.1 Beyond Karatsuba’s algorithm	36
0.2 Algorithms beyond arithmetic	38
0.3 On the importance of negative results.	39
0.4 Roadmap to the rest of this course	40
0.4.1 Dependencies between chapters	41
0.5 Exercises	42
0.6 Bibliographical notes	44
0.7 Further explorations	44
1 Mathematical Background	47
1.1 A mathematician’s apology	47
1.2 A quick overview of mathematical prerequisites	48
1.3 Reading mathematical texts	50
1.3.1 Example: Defining a one to one function	52
1.4 Basic discrete math objects	54
1.4.1 Sets	54
1.4.2 Sets in Python (optional)	55
1.4.3 Special sets	56
1.4.4 Functions	57
1.4.5 Graphs	60
1.4.6 Logic operators and quantifiers.	62
1.4.7 Quantifiers for summations and products	63
1.4.8 Parsing formulas: bound and free variables	63
1.4.9 Asymptotics and Big- <i>O</i> notation	65

1.4.10	Some “rules of thumb” for Big- O notation	67
1.5	Proofs	67
1.5.1	Proofs and programs	68
1.6	Extended example: graph connectivity	68
1.6.1	Mathematical induction	70
1.6.2	Proving the theorem by induction	71
1.6.3	Writing down the proof	73
1.7	Proof writing style	76
1.7.1	Patterns in proofs	76
1.8	Non-standard notation	79
1.9	Exercises	81
1.10	Bibliographical notes	83
2	Computation and Representation	85
2.1	Examples of binary representations	87
2.1.1	Representing natural numbers	87
2.1.2	Representing (potentially negative) integers	89
2.1.3	Representing rational numbers	90
2.2	Representing real numbers	92
2.2.1	Can we represent reals <i>exactly</i> ?	92
2.3	Beyond numbers	97
2.3.1	Finite representations	97
2.3.2	Prefix-free encoding	98
2.3.3	Making representations prefix-free	99
2.3.4	“Proof by Python” (optional)	100
2.3.5	Representing letters and text	103
2.3.6	Representing vectors, matrices, images	106
2.3.7	Representing graphs	106
2.3.8	Representing lists	107
2.3.9	Notation	107
2.4	Defining computational tasks	108
2.4.1	Distinguish functions from programs	110
2.4.2	Advanced note: beyond computing functions . . .	111
2.5	Exercises	113
2.6	Bibliographical notes	115
2.7	Further explorations	116
I	Finite computation	117
3	Defining computation	119
3.1	Defining computation	121
3.1.1	Boolean formulas with AND, OR, and NOT. . . .	123
3.1.2	The NAND function	127
3.2	Informally defining “basic operations” and “algorithms”	129

3.3	From NAND to infinity and beyond...	130
3.3.1	NAND Circuits	131
3.4	Physical implementations of computing devices.	136
3.4.1	Transistors and physical logic gates	136
3.4.2	NAND gates from transistors	140
3.5	Basing computing on other media (optional)	140
3.5.1	Biological computing	140
3.5.2	Cellular automata and the game of life	141
3.5.3	Neural networks	141
3.5.4	The marble computer	142
3.6	The NAND Programming language	144
3.6.1	NAND programs and NAND circuits	146
3.6.2	Circuits with other gate sets (optional)	148
3.7	Exercises	149
3.8	Biographical notes	150
3.9	Further explorations	150
4	Syntactic sugar, and computing every function	151
4.1	Some useful syntactic sugar	152
4.1.1	Constants	152
4.1.2	Functions / Macros	153
4.1.3	Example: Computing Majority via NAND's	153
4.1.4	Conditional statements	154
4.1.5	Bounded loops	155
4.1.6	Example: Adding two integers	155
4.2	Even more sugar (optional)	158
4.2.1	More indices	158
4.2.2	Non-Boolean variables, lists and integers	158
4.2.3	Storing integers	159
4.2.4	Example: Multiplying n bit numbers	159
4.3	Functions beyond arithmetic and LOOKUP	161
4.3.1	Constructing a NAND program for <i>LOOKUP</i>	161
4.4	Computing <i>every</i> function	163
4.4.1	Proof of NAND's Universality	164
4.4.2	Improving by a factor of n (optional)	166
4.4.3	The class $SIZE_{n,m}(T)$	167
4.5	Exercises	169
4.6	Bibliographical notes	170
4.7	Further explorations	170
5	Code as data, data as code	171
5.1	A NAND interpreter in NAND	172
5.1.1	Concrete representation for NAND programs	174
5.1.2	Representing a program as a string	175

5.1.3	A NAND interpreter in “pseudocode”	176
5.1.4	A NAND interpreter in Python	177
5.1.5	Constructing the NAND interpreter in NAND	178
5.2	A Python interpreter in NAND (discussion)	180
5.3	Counting programs, and lower bounds on the size of NAND programs	181
5.4	The physical extended Church-Turing thesis (discussion)	185
5.4.1	Attempts at refuting the PECTT	187
5.5	Exercises	191
5.6	Bibliographical notes	192
5.7	Further explorations	192
II	Uniform computation	195
6	Loops and infinity	197
6.1	The NAND++ Programming language	200
6.1.1	Enhanced NAND++ programs	200
6.1.2	Variables as arrays and well-formed programs	202
6.1.3	“Oblivious” / “Vanilla” NAND++	203
6.2	Computable functions	205
6.2.1	Infinite loops and partial functions	207
6.3	Equivalence of “vanilla” and “enhanced” NAND++	208
6.3.1	Simulating NAND++ programs by enhanced NAND++ programs.	209
6.3.2	Simulating enhanced NAND++ programs by NAND++ programs.	210
6.3.3	Well formed programs: The NAND++ style manual	214
6.4	Turing Machines	217
6.4.1	Turing machines as programming languages	223
6.4.2	Turing machines and NAND++ programs	224
6.5	Uniformity, and NAND vs NAND++ (discussion)	228
6.6	Exercises	230
6.7	Bibliographical notes	230
6.8	Further explorations	230
6.9	Acknowledgements	230
7	Equivalent models of computation	231
7.1	RAM machines and NAND«	231
7.1.1	Indexed access in NAND++	233
7.1.2	Two dimensional arrays in NAND++	235
7.1.3	All the rest	236
7.1.4	Turing equivalence (discussion)	236
7.2	The “Best of both worlds” paradigm (discussion)	237

7.2.1	Let's talk about abstractions.	238
7.3	Lambda calculus and functional programming languages	240
7.3.1	Formal description of the λ calculus.	243
7.3.2	Functions as first class objects	245
7.3.3	"Enhanced" lambda calculus	245
7.3.4	How basic is "basic"?	250
7.3.5	List processing	252
7.3.6	Recursion without recursion	252
7.4	Other models	256
7.4.1	Parallel algorithms and cloud computing	256
7.4.2	Game of life, tiling and cellular automata	256
7.4.3	Configurations of NAND++/Turing machines and one dimensional cellular automata	257
7.5	Turing completeness and equivalence, a formal definition (optional)	263
7.6	The Church-Turing Thesis (discussion)	264
7.7	Our models vs other texts	265
7.8	Exercises	266
7.9	Bibliographical notes	266
7.10	Further explorations	266
7.11	Acknowledgements	267
8	Universality and uncomputability	269
8.1	Universality: A NAND++ interpreter in NAND++	270
8.2	Is every function computable?	274
8.3	The Halting problem	276
8.3.1	Is the Halting problem really hard? (discussion)	278
8.3.2	Reductions	279
8.3.3	A direct proof of the uncomputability of <i>HALT</i> (optional)	281
8.4	Impossibility of general software verification	283
8.4.1	Rice's Theorem	286
8.4.2	Halting and Rice's Theorem for other Turing-complete models	290
8.4.3	Is software verification doomed? (discussion)	291
8.5	Exercises	293
8.6	Bibliographical notes	293
8.7	Further explorations	293
8.8	Acknowledgements	294
9	Restricted computational models	295
9.1	Turing completeness as a bug	295
9.2	Regular expressions	297

9.2.1	Efficient matching of regular expressions (advanced, optional)	301
9.2.2	Equivalence of DFA's and regular expressions (optional)	305
9.3	Limitations of regular expressions	306
9.4	Other semantic properties of regular expressions	311
9.5	Context free grammars	312
9.5.1	Context-free grammars as a computational model	315
9.5.2	The power of context free grammars	316
9.5.3	Limitations of context-free grammars (optional)	318
9.6	Semantic properties of context free languages	319
9.6.1	Uncomputability of context-free grammar equivalence (optional)	320
9.7	Summary of semantic properties for regular expressions and context-free grammars	323
9.8	Exercises	324
9.9	Bibliographical notes	324
9.10	Further explorations	324
9.11	Acknowledgements	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
10.7	Further explorations	340
10.8	Acknowledgements	340
III	Efficient algorithms	341
11	Efficient computation	343
11.1	Problems on graphs	344
11.1.1	Finding the shortest path in a graph	345
11.1.2	Finding the longest path in a graph	347
11.1.3	Finding the minimum cut in a graph	348
11.1.4	Finding the maximum cut in a graph	352
11.1.5	A note on convexity	352
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	356
11.3.1	Determinant of a matrix	356
11.3.2	The permanent (mod 2) problem	357
11.3.3	The permanent (mod 3) problem	358
11.3.4	Finding a zero-sum equilibrium	358
11.3.5	Finding a Nash equilibrium	359
11.3.6	Primality testing	359
11.3.7	Integer factoring	359
11.4	Our current knowledge	360
11.5	Lecture summary	361
11.6	Exercises	361
11.7	Bibliographical notes	362
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 \ll vs NAND++	368
12.3	Efficient universal machine: a NAND \ll interpreter in NAND \ll	371
12.4	Time hierarchy theorem	375
12.5	Unrolling the loop: Uniform vs non uniform computation	378
12.5.1	Algorithmic transformation of NAND++ to NAND and “Proof by Python” (optional)	380
12.5.2	The class $\mathbf{P}_{/poly}$	383
12.5.3	Simulating NAND with NAND++?	385
12.5.4	Uniform vs. Nonuniform computation: A recap	386
12.6	Extended Church-Turing Thesis	387
12.7	Exercises	388
12.8	Bibliographical notes	390
12.9	Further explorations	390
12.10	Acknowledgements	390
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	395
13.3	The independent set problem	397
13.4	Reducing Independent Set to Maximum Cut	400
13.5	Reducing 3SAT to Longest Path	401
13.6	Exercises	405
13.7	Bibliographical notes	405
13.8	Further explorations	405
13.9	Acknowledgements	405
14	NP, NP completeness, and the Cook-Levin Theorem	407
14.1	The class NP	407
14.1.1	Examples of NP functions	409
14.1.2	Basic facts about NP	411
14.2	From NP to 3SAT: The Cook-Levin Theorem	413
14.2.1	What does this mean?	414
14.2.2	The Cook-Levin Theorem: Proof outline	415
14.3	The <i>NANDSAT</i> Problem, and why it is NP hard.	417
14.4	The <i>3NAND</i> problem	419
14.5	From <i>3NAND</i> to <i>3SAT</i>	422
14.6	Wrapping up	423
14.7	Exercises	424
14.8	Bibliographical notes	424
14.9	Further explorations	425
14.10	Acknowledgements	425
15	What if P equals NP?	427
15.1	Search-to-decision reduction	428
15.2	Optimization	430
15.2.1	Example: Supervised learning	433
15.2.2	Example: Breaking cryptosystems	434
15.3	Finding mathematical proofs	435
15.4	Quantifier elimination (advanced)	436
15.4.1	Application: self improving algorithm for <i>3SAT</i>	439
15.5	Approximating counting problems (advanced, optional)	439
15.6	What does all of this imply?	440
15.7	Can $P \neq NP$ be neither true nor false?	442
15.8	Is $P = NP$ “in practice”?	443
15.9	What if $P \neq NP$?	444
15.10	Exercises	445
15.11	Bibliographical notes	446
15.12	Further explorations	446
15.13	Acknowledgements	446
16	Space bounded computation	447

16.1	Lecture summary	447
16.2	Exercises	447
16.3	Bibliographical notes	447
16.4	Further explorations	447
16.5	Acknowledgements	447
IV Randomized computation		449
17	Probability Theory 101	451
17.1	Random coins	452
17.1.1	Random variables	454
17.1.2	Distributions over strings	456
17.1.3	More general sample spaces.	457
17.2	Correlations and independence	457
17.2.1	Independent random variables	459
17.2.2	Collections of independent random variables.	461
17.3	Concentration	462
17.3.1	Chebyshev’s Inequality	464
17.3.2	The Chernoff bound	465
17.4	Lecture summary	466
17.5	Exercises	466
17.6	Bibliographical notes	468
17.7	Further explorations	468
17.8	Acknowledgements	468
18	Probabilistic computation	469
18.1	Finding approximately good maximum cuts.	470
18.1.1	Amplification	471
18.1.2	Two-sided amplification	473
18.1.3	What does this mean?	474
18.1.4	Solving SAT through randomization	475
18.1.5	Bipartite matching.	476
18.2	Exercises	480
18.3	Bibliographical notes	481
18.4	Further explorations	481
18.5	Acknowledgements	481
19	Modeling randomized computation	483
19.0.1	An alternative view: random coins as an “extra input”	485
19.0.2	Amplification	487
19.1	BPP and NP completeness	488
19.2	The power of randomization	490
19.2.1	Solving BPP in exponential time	490

19.2.2	Simulating randomized algorithms by circuits or straightline programs.	491
19.3	Derandomization	493
19.3.1	Pseudorandom generators	494
19.3.2	From existence to constructivity	496
19.3.3	Usefulness of pseudorandom generators	497
19.4	P = NP and BPP vs P	498
19.5	Non-constructive existence of pseudorandom genera- tors (advanced, optional)	502
19.6	Exercises	504
19.7	Bibliographical notes	504
19.8	Further explorations	504
19.9	Acknowledgements	504
V	Advanced topics	505
20	Cryptography	507
20.1	Classical cryptosystems	508
20.2	Defining encryption	509
20.3	Defining security of encryption	510
20.4	Perfect secrecy	511
20.4.1	Example: Perfect secrecy in the battlefield	513
20.4.2	Constructing perfectly secret encryption	513
20.5	Necessity of long keys	515
20.6	Computational secrecy	518
20.6.1	Stream ciphers or the “derandomized one-time pad”	519
20.7	Computational secrecy and NP	522
20.8	Public key cryptography	524
20.8.1	Public key encryption, trapdoor functions and pseudrandom generators	526
20.9	Other security notions	529
20.10	Magic	529
20.10.1	Zero knowledge proofs	529
20.10.2	Fully homomorphic encryption	530
20.10.3	Multiparty secure computation	530
20.11	Lecture summary	531
20.12	Exercises	531
20.13	Bibliographical notes	531
20.14	Further explorations	532
20.15	Acknowledgements	532
21	Proofs and algorithms	533
21.1	Lecture summary	533

21.2	Exercises	533
21.3	Bibliographical notes	533
21.4	Further explorations	533
21.5	Acknowledgements	534
22	Quantum computing	535
22.1	The double slit experiment	536
22.2	Quantum amplitudes	537
22.3	Bell’s Inequality	540
22.4	Quantum weirdness	541
22.5	Quantum computing and computation - an executive summary.	542
22.6	Quantum systems	544
22.6.1	Quantum amplitudes	545
22.6.2	Recap	546
22.7	Analysis of Bell’s Inequality (optional)	547
22.8	Quantum computation	548
22.8.1	Quantum circuits	549
22.8.2	QNAND programs (optional)	552
22.8.3	Uniform computation	553
22.9	Physically realizing quantum computation	554
22.10	Shor’s Algorithm: Hearing the shape of prime factors	555
22.10.1	Period finding	555
22.10.2	Shor’s Algorithm: A bird’s eye view	557
22.11	Quantum Fourier Transform (advanced, optional)	559
22.11.1	Quantum Fourier Transform over the Boolean Cube: Simon’s Algorithm	560
22.12	Exercises	563
22.13	Bibliographical notes	564
22.14	Further explorations	564
22.15	Acknowledgements	564
VI	Appendices	565
A	The NAND Programming Language	567
B	The NAND++ Programming Language	599
C	The Lambda Calculus	611

