

# Contents (detailed)

<b>Preface</b>	<b>19</b>
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
<b>Preliminaries</b>	<b>25</b>
<b>0 Introduction</b>	<b>27</b>
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
<b>1 Mathematical Background</b>	<b>45</b>
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 thumb” 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 . . . . .	77
1.9	Exercises . . . . .	79
1.10	Bibliographical notes . . . . .	81
<b>2</b>	<b>Computation and Representation</b>	<b>83</b>
2.1	Examples of binary representations . . . . .	85
2.1.1	Representing natural numbers . . . . .	85
2.1.2	Representing (potentially negative) integers . . . .	87
2.1.3	Representing rational numbers . . . . .	88
2.2	Representing real numbers . . . . .	90
2.2.1	Can we represent reals <i>exactly</i> ? . . . . .	90
2.3	Beyond numbers . . . . .	94
2.3.1	Finite representations . . . . .	95
2.3.2	Prefix free encoding . . . . .	96
2.3.3	Making representations prefix free . . . . .	97
2.3.4	“Proof by Python” (optional) . . . . .	98
2.3.5	Representing letters and text . . . . .	100
2.3.6	Representing vectors, matrices, images . . . . .	103
2.3.7	Representing graphs . . . . .	104
2.3.8	Representing lists . . . . .	105
2.3.9	Notation . . . . .	105
2.4	Defining computational tasks . . . . .	106
2.4.1	Distinguish functions from programs . . . . .	108
2.4.2	Advanced note: beyond computing functions . . .	109
2.5	Exercises . . . . .	110
2.6	Bibliographical notes . . . . .	113
2.7	Further explorations . . . . .	113
<b>I</b>	<b>Finite computation</b>	<b>115</b>
<b>3</b>	<b>Defining computation</b>	<b>117</b>
3.1	Defining computation . . . . .	119
3.1.1	Defining “elementary operations” . . . . .	121
3.1.2	The NAND function . . . . .	124
3.2	Informally defining “basic operations” and “algorithms”	126

3.3	From NAND to infinity and beyond.. . . . .	127
3.3.1	NAND Circuits . . . . .	127
3.4	Physical implementations of computing devices. . . . .	133
3.4.1	Transistors and physical logic gates . . . . .	133
3.4.2	NAND gates from transistors . . . . .	137
3.5	Basing computing on other media (optional) . . . . .	137
3.5.1	Biological computing . . . . .	137
3.5.2	Cellular automata and the game of life . . . . .	138
3.5.3	Neural networks . . . . .	138
3.5.4	The marble computer . . . . .	139
3.6	The NAND Programming language . . . . .	141
3.6.1	NAND programs and NAND circuits . . . . .	143
3.7	Exercises . . . . .	146
3.8	Biographical notes . . . . .	146
3.9	Further explorations . . . . .	146
<b>4</b>	<b>Syntactic sugar, and computing every function</b>	<b>149</b>
4.1	Some useful syntactic sugar . . . . .	150
4.1.1	Constants . . . . .	150
4.1.2	Functions / Macros . . . . .	151
4.1.3	Example: Computing Majority via NAND's . . . . .	151
4.1.4	Conditional statements . . . . .	152
4.1.5	Bounded loops . . . . .	153
4.1.6	Example: Adding two integers . . . . .	153
4.2	Even more sugar (optional) . . . . .	156
4.2.1	More indices . . . . .	156
4.2.2	Non-Boolean variables, lists and integers . . . . .	156
4.2.3	Storing integers . . . . .	157
4.2.4	Example: Multiplying $n$ bit numbers . . . . .	157
4.3	Functions beyond arithmetic and LOOKUP . . . . .	159
4.3.1	Constructing a NAND program for <i>LOOKUP</i> . . . . .	159
4.4	Computing <i>every</i> function . . . . .	161
4.4.1	Proof of NAND's Universality . . . . .	162
4.4.2	Improving by a factor of $n$ (optional) . . . . .	164
4.4.3	The class $SIZE_{n,m}(T)$ . . . . .	165
4.5	Exercises . . . . .	167
4.6	Bibliographical notes . . . . .	168
4.7	Further explorations . . . . .	168
<b>5</b>	<b>Code as data, data as code</b>	<b>169</b>
5.1	A NAND interpreter in NAND . . . . .	170
5.1.1	Concrete representation for NAND programs . . . . .	172
5.1.2	Representing a program as a string . . . . .	173
5.1.3	A NAND interpreter in "pseudocode" . . . . .	174

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 programs . . . . .	179
5.4	The physical extended Church-Turing thesis (discussion)	183
5.4.1	Attempts at refuting the PECTT . . . . .	185
5.5	Exercises . . . . .	189
5.6	Bibliographical notes . . . . .	190
5.7	Further explorations . . . . .	190
<b>II Uniform computation</b>		<b>193</b>
<b>6</b>	<b>Loops and infinity</b>	<b>195</b>
6.1	The NAND++ Programming language . . . . .	197
6.1.1	Enhanced NAND++ programs . . . . .	198
6.1.2	“Oblivious” / “Vanilla” NAND++ . . . . .	200
6.2	Computable functions . . . . .	202
6.3	Equivalence of “vanilla” and “enhanced” NAND++ . . . . .	204
6.3.1	Simulating NAND++ programs by enhanced NAND++ programs. . . . .	205
6.3.2	Simulating enhanced NAND++ programs by NAND++ programs. . . . .	206
6.3.3	Another application of GOTO: well formed programs . . . . .	210
6.4	Turing Machines . . . . .	213
6.5	Uniformity, and NAND vs NAND++ (discussion) . . . . .	220
6.6	Exercises . . . . .	222
6.7	Bibliographical notes . . . . .	222
6.8	Further explorations . . . . .	223
6.9	Acknowledgements . . . . .	223
<b>7</b>	<b>Equivalent models of computation</b>	<b>225</b>
7.1	RAM machines and NAND $\llcorner$ . . . . .	225
7.1.1	Indexed access in NAND++ . . . . .	227
7.1.2	Two dimensional arrays in NAND++ . . . . .	229
7.1.3	All the rest . . . . .	230
7.1.4	Turing equivalence (discussion) . . . . .	230
7.2	The “Best of both worlds” paradigm (discussion) . . . . .	231
7.2.1	Let’s talk about abstractions. . . . .	232
7.3	Lambda calculus and functional programming lan- guages . . . . .	234
7.3.1	Formal description of the $\lambda$ calculus. . . . .	237
7.3.2	Functions as first class objects . . . . .	239

7.3.3	“Enhanced” lambda calculus . . . . .	239
7.3.4	How basic is “basic”? . . . . .	244
7.3.5	List processing . . . . .	246
7.3.6	Recursion without recursion . . . . .	246
7.4	Other models . . . . .	250
7.4.1	Parallel algorithms and cloud computing . . . . .	250
7.4.2	Game of life, tiling and cellular automata . . . . .	250
7.4.3	Configurations of NAND++/Turing machines and one dimensional cellular automata . . . . .	251
7.5	Turing completeness and equivalence, a formal defini- tion (optional) . . . . .	255
7.6	The Church-Turing Thesis (discussion) . . . . .	256
7.7	Our models vs other texts . . . . .	257
7.8	Exercises . . . . .	257
7.9	Bibliographical notes . . . . .	258
7.10	Further explorations . . . . .	258
7.11	Acknowledgements . . . . .	258
<b>8</b>	<b>Universality and uncomputability</b>	<b>259</b>
8.1	Universality: A NAND++ interpreter in NAND++ . . . . .	260
8.2	Is every function computable? . . . . .	263
8.3	The Halting problem . . . . .	265
8.3.1	Is the Halting problem really hard? (discussion) . . . . .	267
8.3.2	Reductions . . . . .	268
8.3.3	A direct proof of the uncomputability of <i>HALT</i> (optional) . . . . .	269
8.4	Impossibility of general software verification . . . . .	272
8.4.1	Rice’s Theorem . . . . .	274
8.4.2	Halting and Rice’s Therem for other Turing- complete models . . . . .	277
8.4.3	Is software verification doomed? (discussion) . . . . .	279
8.5	Exercises . . . . .	280
8.6	Bibliographical notes . . . . .	281
8.7	Further explorations . . . . .	281
8.8	Acknowledgements . . . . .	281
<b>9</b>	<b>Restricted computational models</b>	<b>283</b>
9.1	Turing completeness as a bug . . . . .	283
9.2	Regular expressions . . . . .	285
9.2.1	Efficient matching of regular expressions (ad- vanced, optional) . . . . .	288
9.3	Limitations of regular expressions . . . . .	291
9.4	Other semantic properties of regular expressions . . . . .	295
9.5	Context free grammars . . . . .	297

9.5.1	Context-free grammars as a computational model	299
9.5.2	The power of context free grammars . . . . .	301
9.5.3	Limitations of context-free grammars (optional) .	302
9.6	Semantic properties of context free languages . . . . .	304
9.6.1	Uncomputability of context-free grammar equivalence (optional) . . . . .	305
9.7	Summary of semantic properties for regular expres- sions and context-free grammars . . . . .	306
9.8	Exercises . . . . .	308
9.9	Bibliographical notes . . . . .	308
9.10	Further explorations . . . . .	308
9.11	Acknowledgements . . . . .	308
<b>10</b>	<b>Is every theorem provable?</b>	<b>309</b>
10.1	Hilbert's Program and Gödel's Incompleteness Theorem	310
10.2	Quantified integer statements . . . . .	314
10.3	Diophantine equations and the MRDP Theorem . . . . .	316
10.4	Hardness of quantified integer statements . . . . .	317
10.4.1	Step 1: Quantified mixed statements and com- putation histories . . . . .	318
10.4.2	Step 2: Reducing mixed statements to integer statements . . . . .	321
10.5	Exercises . . . . .	322
10.6	Bibliographical notes . . . . .	323
10.7	Further explorations . . . . .	323
10.8	Acknowledgements . . . . .	323
<b>III</b>	<b>Efficient algorithms</b>	<b>325</b>
<b>11</b>	<b>Efficient computation</b>	<b>327</b>
11.1	Problems on graphs . . . . .	328
11.1.1	Finding the shortest path in a graph . . . . .	329
11.1.2	Finding the longest path in a graph . . . . .	330
11.1.3	Finding the minimum cut in a graph . . . . .	331
11.1.4	Finding the maximum cut in a graph . . . . .	334
11.1.5	A note on convexity . . . . .	334
11.2	Beyond graphs . . . . .	336
11.2.1	The 2SAT problem . . . . .	336
11.2.2	The 3SAT problem . . . . .	337
11.2.3	Solving linear equations . . . . .	337
11.2.4	Solving quadratic equations . . . . .	338
11.3	More advanced examples . . . . .	338
11.3.1	The permanent (mod 2) problem . . . . .	339
11.3.2	The permanent (mod 3) problem . . . . .	339

11.3.3	Finding a zero-sum equilibrium . . . . .	339
11.3.4	Finding a Nash equilibrium . . . . .	340
11.3.5	Primal testing . . . . .	340
11.3.6	Integer factoring . . . . .	341
11.4	Our current knowledge . . . . .	341
11.5	Lecture summary . . . . .	342
11.6	Exercises . . . . .	342
11.7	Bibliographical notes . . . . .	343
11.8	Further explorations . . . . .	343
11.9	Acknowledgements . . . . .	343
<b>12</b>	<b>Modeling running time</b> . . . . .	<b>345</b>
12.1	NAND« vs NAND++ . . . . .	348
12.2	Efficient universal machine: a NAND« interpreter in NAND« . . . . .	350
12.3	Time hierarchy theorem . . . . .	352
12.4	Uniform vs non uniform computation . . . . .	354
12.4.1	The class $P_{poly}$ . . . . .	357
12.4.2	Simulating NAND with NAND++? . . . . .	358
12.4.3	Uniform vs. Nonuniform computation: A recap . . . . .	359
12.5	Extended Church-Turing Thesis . . . . .	360
12.6	Lecture summary . . . . .	361
12.7	Exercises . . . . .	362
12.8	Bibliographical notes . . . . .	362
12.9	Further explorations . . . . .	362
12.10	Acknowledgements . . . . .	362
<b>13</b>	<b>Polynomial-time reductions</b> . . . . .	<b>363</b>
13.0.1	Decision problems . . . . .	364
13.1	Reductions . . . . .	364
13.2	Some example reductions . . . . .	365
13.2.1	Reducing 3SAT to quadratic equations . . . . .	366
13.3	The independent set problem . . . . .	368
13.4	Reducing Independent Set to Maximum Cut . . . . .	370
13.5	Reducing 3SAT to Longest Path . . . . .	372
13.6	Exercises . . . . .	374
13.7	Bibliographical notes . . . . .	374
13.8	Further explorations . . . . .	374
13.9	Acknowledgements . . . . .	374
<b>14</b>	<b>NP, NP completeness, and the Cook-Levin Theorem</b> . . . . .	<b>375</b>
14.1	The class NP . . . . .	375
14.1.1	Examples: . . . . .	377
14.1.2	From NP to 3SAT . . . . .	377
14.1.3	What does this mean? . . . . .	378

14.2	The Cook-Levin Theorem . . . . .	379
14.2.1	The <i>NANDSAT</i> Problem, and why it is <b>NP</b> hard. . . . .	381
14.2.2	The <i>3NAND</i> problem . . . . .	383
14.2.3	From <i>3NAND</i> to <i>3SAT</i> . . . . .	384
14.2.4	Wrapping up . . . . .	386
14.3	Exercises . . . . .	386
14.4	Bibliographical notes . . . . .	387
14.5	Further explorations . . . . .	387
14.6	Acknowledgements . . . . .	387
<b>15</b>	<b>What if <math>P</math> equals <math>NP</math>?</b> . . . . .	<b>389</b>
15.1	Search-to-decision reduction . . . . .	390
15.2	Optimization . . . . .	392
15.2.1	Example: Supervised learning . . . . .	394
15.2.2	Example: Breaking cryptosystems . . . . .	395
15.3	Finding mathematical proofs . . . . .	395
15.4	Quantifier elimination . . . . .	397
15.4.1	Approximating counting problems . . . . .	398
15.5	What does all of this imply? . . . . .	398
15.6	Can $P \neq NP$ be neither true nor false? . . . . .	400
15.7	Is $P = NP$ “in practice”? . . . . .	401
15.8	What if $P \neq NP$ ? . . . . .	402
15.9	Exercises . . . . .	404
15.10	Bibliographical notes . . . . .	404
15.11	Further explorations . . . . .	404
15.12	Acknowledgements . . . . .	404
<b>16</b>	<b>Space bounded computation</b> . . . . .	<b>405</b>
16.1	Lecture summary . . . . .	405
16.2	Exercises . . . . .	405
16.3	Bibliographical notes . . . . .	405
16.4	Further explorations . . . . .	405
16.5	Acknowledgements . . . . .	405
<b>IV</b>	<b>Randomized computation</b> . . . . .	<b>407</b>
<b>17</b>	<b>Probability Theory 101</b> . . . . .	<b>409</b>
17.1	Random coins . . . . .	410
17.1.1	Random variables . . . . .	412
17.1.2	Distributions over strings . . . . .	414
17.1.3	More general sample spaces. . . . .	415
17.2	Correlations and independence . . . . .	415
17.2.1	Independent random variables . . . . .	417
17.2.2	Collections of independent random variables. . . . .	419



17.3	Concentration . . . . .	420
17.3.1	Chebyshev's Inequality . . . . .	422
17.3.2	The Chernoff bound . . . . .	423
17.4	Lecture summary . . . . .	424
17.5	Exercises . . . . .	424
17.6	Bibliographical notes . . . . .	426
17.7	Further explorations . . . . .	426
17.8	Acknowledgements . . . . .	426
<b>18</b>	<b>Probabilistic computation</b>	<b>427</b>
18.1	Finding approximately good maximum cuts. . . . .	428
18.1.1	Amplification . . . . .	429
18.1.2	Two-sided amplification . . . . .	430
18.1.3	What does this mean? . . . . .	430
18.1.4	Solving SAT through randomization . . . . .	431
18.1.5	Bipartite matching. . . . .	433
18.2	Lecture summary . . . . .	436
18.3	Exercises . . . . .	437
18.4	Bibliographical notes . . . . .	437
18.5	Further explorations . . . . .	438
18.6	Acknowledgements . . . . .	438
<b>19</b>	<b>Modeling randomized computation</b>	<b>439</b>
19.0.1	Random coins as an "extra input" . . . . .	441
19.0.2	Amplification . . . . .	443
19.1	<b>BPP</b> and <b>NP</b> completeness . . . . .	443
19.2	The power of randomization . . . . .	445
19.2.1	Solving <b>BPP</b> in exponential time . . . . .	445
19.2.2	Simulating randomized algorithms by circuits or straightline programs. . . . .	446
19.3	Derandomization . . . . .	448
19.3.1	Pseudorandom generators . . . . .	449
19.3.2	From existence to constructivity . . . . .	451
19.3.3	Usefulness of pseudorandom generators . . . . .	453
19.4	<b>P = NP</b> and <b>BPP</b> vs <b>P</b> . . . . .	454
19.5	Non-constructive existence of pseudorandom generators . . . . .	454
19.6	Lecture summary . . . . .	456
19.7	Exercises . . . . .	456
19.8	Bibliographical notes . . . . .	456
19.9	Further explorations . . . . .	456
19.10	Acknowledgements . . . . .	456

<b>V</b>	<b>Advanced topics</b>	<b>457</b>
<b>20</b>	<b>Cryptography</b>	<b>459</b>
20.1	Classical cryptosystems . . . . .	460
20.2	Defining encryption . . . . .	461
20.3	Defining security of encryption . . . . .	462
20.4	Perfect secrecy . . . . .	463
20.4.1	Example: Perfect secrecy in the battlefield . . . . .	465
20.4.2	Constructing perfectly secret encryption . . . . .	465
20.5	Necessity of long keys . . . . .	467
20.6	Computational secrecy . . . . .	470
20.6.1	Stream ciphers or the “derandomized one-time pad” . . . . .	471
20.7	Computational secrecy and NP . . . . .	474
20.8	Public key cryptography . . . . .	476
20.8.1	Public key encryption, trapdoor functions and pseudorandom generators . . . . .	478
20.9	Other security notions . . . . .	481
20.10	Magic . . . . .	481
20.10.1	Zero knowledge proofs . . . . .	481
20.10.2	Fully homomorphic encryption . . . . .	482
20.10.3	Multiparty secure computation . . . . .	482
20.11	Lecture summary . . . . .	483
20.12	Exercises . . . . .	483
20.13	Bibliographical notes . . . . .	483
20.14	Further explorations . . . . .	484
20.15	Acknowledgements . . . . .	484
<b>21</b>	<b>Proofs and algorithms</b>	<b>485</b>
21.1	Lecture summary . . . . .	485
21.2	Exercises . . . . .	485
21.3	Bibliographical notes . . . . .	485
21.4	Further explorations . . . . .	485
21.5	Acknowledgements . . . . .	486
<b>22</b>	<b>Quantum computing</b>	<b>487</b>
22.1	The double slit experiment . . . . .	488
22.2	Quantum amplitudes . . . . .	489
22.3	Bell’s Inequality . . . . .	492
22.4	Quantum weirdness . . . . .	493
22.5	Quantum computing and computation - an executive summary. . . . .	494
22.6	Quantum systems . . . . .	496
22.6.1	Quantum amplitudes . . . . .	497
22.6.2	Recap . . . . .	498

22.7	Analysis of Bell's Inequality (optional)	499
22.8	Quantum computation	500
22.8.1	Quantum circuits	501
22.8.2	QNAND programs (optional)	504
22.8.3	Uniform computation	505
22.9	Physically realizing quantum computation	506
22.10	Shor's Algorithm: Hearing the shape of prime factors	507
22.10.1	Period finding	507
22.10.2	Shor's Algorithm: A bird's eye view	509
22.11	Quantum Fourier Transform (advanced, optional)	511
22.11.1	Quantum Fourier Transform over the Boolean Cube: Simon's Algorithm	512
22.12	Exercises	515
22.13	Bibliographical notes	516
22.14	Further explorations	516
22.15	Acknowledgements	516
<b>VI</b>	<b>Appendices</b>	<b>517</b>
<b>A</b>	<b>The NAND Programming Language</b>	<b>519</b>
<b>B</b>	<b>The NAND++ Programming Language</b>	<b>551</b>
<b>C</b>	<b>The Lambda Calculus</b>	<b>563</b>

