

Contents

Acknowledgments	2
Preface for instructors	11
Which “theory” course are we talking about?	12
The features that might make this book appealing .	13
What’s in and what’s out	14
Possible courses based on this book	15
Computer science as a liberal art	15
Part 0: Overview	17
1 Introduction: what can and cannot be computed?	18
1.1 Tractable problems	19
1.2 Intractable problems	20
1.3 Uncomputable problems	21
1.4 A more detailed overview of the book	22
1.5 Prerequisites for understanding this book	24
1.6 The goals of the book	25
1.7 Why study the theory of computation?	27
Exercises	28
Part I: Computability theory	30
2 What is a computer program?	31
2.1 Some Python program basics	32
2.2 SISO Python programs	35
Programs that call other functions and programs . .	37
2.3 ASCII characters and multiline strings	37
2.4 Some problematic programs	40

4

2.5	Formal definition of Python program	41
2.6	Decision programs and equivalent programs	44
2.7	Real-world programs vs SISO Python programs	44
	Exercises	46
3	Some impossible Python programs	50
3.1	Proof by contradiction	50
3.2	Programs that analyze other programs	52
	Programs that analyze themselves	52
3.3	The program <code>yesOnString.py</code>	53
3.4	The program <code>yesOnSelf.py</code>	56
3.5	The program <code>notYesOnSelf.py</code>	57
3.6	<code>yesOnString.py</code> can't exist, either	58
	A compact proof that <code>yesOnString.py</code> can't exist	59
3.7	Perfect bug-finding programs are impossible	62
3.8	We can still find bugs, but we can't do it perfectly	64
	Exercises	65
4	What is a computational problem?	69
4.1	Graphs, alphabets, strings, and languages	72
	Graphs	72
	Trees and rooted trees	74
	Alphabets	75
	Strings	75
	Languages	76
4.2	Defining computational problems	79
	Positive and negative instances	82
	Notation for computational problems	83
4.3	Categories of computational problems	83
	Converting between general and decision problems	87
	Complement of a decision problem	88
	Computational problems with two input strings	88
4.4	The formal definition of "solving" a problem	89
	Computable functions	91
4.5	Recognizing and deciding languages	91
	Recognizable languages	93
	Exercises	95
5	Turing machines: the simplest computers	101
5.1	Definition of Turing machine	102
	Halting and looping	108
	Accepters and transducers	108

	Abbreviated notation for state diagrams	109
	Creating your own Turing machines	111
5.2	Some nontrivial Turing machines	112
	The moreCsThanGs machine	112
	The countCs machine	112
	Important lessons from the countCs example	118
5.3	From single-tape Turing machines to multi-tape Turing machines	120
	Two-tape, single-head Turing machines	121
	Two-way infinite tapes	123
	Multi-tape, single-head Turing machines	124
	Two-tape, two-head Turing machines	124
5.4	From multi-tape Turing machines to Python programs and beyond	127
	Multi-tape Turing machine → random-access Turing machine	127
	Random-access Turing machine → real computer	129
	Modern computer → Python program	132
5.5	Going back the other way: simulating a Turing machine with Python	132
	A serious caveat: memory limitations and other technicalities	134
5.6	Classical computers can simulate quantum computers	135
5.7	All known computers are Turing-equivalent	136
	Exercises	136
6	Universal computer programs: programs that can do anything	141
6.1	Universal Python programs	142
6.2	Universal Turing machines	144
6.3	Universal computation in the real world	146
6.4	Programs that alter other programs	150
	Ignoring the input and performing a fixed calculation instead	152
6.5	Problems that are undecidable but recognizable	153
	Exercises	154
7	Reductions: how to prove a problem is hard	157
7.1	A reduction for easiness	157
7.2	A reduction for hardness	160
7.3	Formal definition of Turing reduction	162
7.4	Properties of Turing reductions	165

7.5	An abundance of uncomputable problems	166
	The variants of YESONSTRING	166
	The halting problem and its variants	170
	Uncomputable problems that aren't decision problems	173
7.6	Even more uncomputable problems	175
	The computational problem $COMPUTES_F$	177
	Rice's theorem	180
7.7	Uncomputable problems that aren't about programs	180
7.8	Not every question about programs is uncomputable	182
7.9	Proof techniques for uncomputability	183
	Technique 1: the reduction recipe	183
	Technique 2: reduction with explicit Python programs	185
	Technique 3: apply Rice's theorem	186
	Exercises	187
8	Nondeterminism: magic or reality?	192
8.1	Nondeterministic Python programs	194
8.2	Nondeterministic programs for non-decision problems	198
8.3	Computation trees	200
8.4	Nondeterminism doesn't change what is computable	204
8.5	Nondeterministic Turing machines	205
8.6	Formal definition of nondeterministic Turing machines	208
8.7	Models of nondeterminism	210
8.8	Unrecognizable problems	210
8.9	Why study nondeterminism?	211
	Exercises	212
9	Finite automata: computing with limited resources	216
9.1	Deterministic finite automata	216
9.2	Nondeterministic finite automata	220
	State diagrams for nfas	220
	Formal definition of an nfa	222
	How does an nfa accept a string?	223
	Sometimes nfas make things easier	223
9.3	Equivalence of nfas and dfas	224
	Nondeterminism can affect computability: the exam- ple of pdas	227
	Practicality of converted nfas	230
	Minimizing the size of dfas	230
9.4	Regular expressions	231
	Pure regular expressions	231
	Standard regular expressions	232

	Converting between regexes and finite automata . . .	233
9.5	Some languages aren't regular	237
	The non-regular language $GnTn$	238
	The key difference between Turing machines and finite automata	239
9.6	Many more non-regular languages	240
	The pumping lemma	242
9.7	Combining regular languages	245
	Exercises	246

Part II: Computational complexity theory 251

10	Complexity theory: when efficiency does matter	252
10.1	Complexity theory uses asymptotic running times	253
10.2	Big- O notation	255
	Dominant terms of functions	256
	A practical definition of big- O notation	259
	Superpolynomial and subexponential	260
	Other asymptotic notation	261
	Composition of polynomials is polynomial	261
	Counting things with big- O	262
10.3	The running time of a program	263
	Running time of a Turing machine	263
	Running time of a Python program	265
	The lack of rigor in Python running times	269
10.4	Fundamentals of determining time complexity	270
	A crucial distinction: the length of the input vs the numerical value of the input	270
	The complexity of arithmetic operations	272
	The complexity of factoring	276
	The complexity of sorting	278
10.5	For complexity, the computational model <i>does</i> matter . . .	279
	Simulation costs for common computational models	279
	Our standard computational model: Python programs	283
10.6	Complexity classes	284
	Exercises	287
11	Poly and Expo: the two most fundamental complexity classes	292
11.1	Definitions of Poly and Expo	292
	Poly and Expo compared to P, Exp and FP	294
11.2	Poly is a subset of Expo	294

11.3	A first look at the boundary between Poly and Expo	295
	ALL3SETS and ALLSUBSETS	296
	Traveling salespeople and shortest paths	297
	Multiplying and factoring	301
	Back to the boundary between Poly and Expo	301
	Primality testing is in Poly	303
11.4	Poly and Expo don't care about the computational model .	304
11.5	HaltEx: A decision problem in Expo but not Poly	305
11.6	Other problems that are outside Poly	311
11.7	Unreasonable encodings of the input affect complexity . . .	312
11.8	Why study Poly, really?	313
	Exercises	314
12	PolyCheck and NPoly: hard problems that are easy to verify	319
12.1	Verifiers	320
12.2	Polytime verifiers	324
12.3	The complexity class PolyCheck	326
	Some PolyCheck examples: PACKING, SUBSETSUM, and PARTITION	327
12.4	The complexity class NPoly	329
12.5	PolyCheck and NPoly are identical	330
	Every PolyCheck problem is in NPoly	331
	Every NPoly problem is in PolyCheck	332
12.6	The PolyCheck/NPoly sandwich	334
12.7	Nondeterminism <i>does</i> seem to change what is computable <i>efficiently</i>	336
12.8	The fine print about NPoly	338
	An alternative definition of NPoly	338
	NPoly compared to NP and FNP	339
	Exercises	341
13	Polynomial-time mapping reductions: proving X is as easy as Y	346
13.1	Definition of polytime mapping reductions	347
	Polyreducing to non-decision problems	349
13.2	The meaning of polynomial time mapping reductions	350
13.3	Proof techniques for polyreductions	351
13.4	Examples of polyreductions using Hamilton cycles	352
	A polyreduction from UHC to DHC	352
	A polyreduction from DHC to UHC	355
13.5	Three satisfiability problems: CircuitSAT, SAT, and 3-SAT	357
	CIRCUITSAT	358

SAT	360
3-SAT	362
13.6 Polyreductions between CircuitSAT, SAT, and 3-SAT	362
13.7 Polyequivalence and its consequences	368
Exercises	369
14 NP-completeness: most hard problems are equally hard	373
14.1 P versus NP	373
14.2 NP-completeness	375
Reformulations of P versus NP using NP-completeness	377
14.3 NP-hardness	378
14.4 Consequences of P=NP	380
14.5 CIRCUITSAT is a “hardest” NP problem	382
14.6 NP-completeness is widespread	389
14.7 Proof techniques for NP-completeness	390
14.8 The good news and bad news about NP-completeness	391
Problems in NPoly but probably not NP-hard	392
Some problems that are in P	392
Some NP-hard problems can be approximated efficiently	393
Some NP-hard problems can be solved efficiently for real-world inputs	393
Some NP-hard problems can be solved in pseudo-polynomial time	394
Exercises	394
Part III: Origins and applications	399
15 The original Turing machine	400
15.1 Turing’s definition of a “computing machine”	401
15.2 Machines can compute what humans can compute	408
15.3 The Church-Turing thesis: a law of nature?	412
The equivalence of digital computers	413
Church’s thesis: the equivalence of computer programs and algorithms	413
Turing’s thesis: the equivalence of computer programs and human brains	414
Church-Turing thesis: the equivalence of all computational processes	415
Exercises	416

16 You can't prove everything that's true	419
16.1 Mechanical proofs	420
Semantics and truth	424
Consistency and completeness	426
Decidability of logical systems	427
16.2 Arithmetic as a logical system	429
Converting the halting problem to a statement about integers	431
Recognizing provable statements about integers . . .	432
The consistency of Peano arithmetic	434
16.3 The undecidability of mathematics	434
16.4 The incompleteness of mathematics	436
16.5 What have we learned and why did we learn it?	440
Exercises	440
17 Karp's 21 problems	444
17.1 Karp's overview	444
17.2 Karp's definition of NP-completeness	447
17.3 The list of 21 NP-complete problems	449
17.4 Reductions between the 21 NP-complete problems	452
Polyreducing SAT to CLIQUE	454
Polyreducing CLIQUE to NODE COVER	456
Polyreducing DHC to UHC	457
Polyreducing SAT to 3-SAT	458
Polyreducing KNAPSACK to PARTITION	458
17.5 The rest of the paper: NP-hardness and more	460
Exercises	461
18 Conclusion: what will be computed?	463
18.1 The big ideas about what can be computed	464