## CONTENTS


Acknowledgments ..... xiii
Preface for instructors ..... xV
The inspiration of GEB ..... xv
Which "theory" course are we talking about? ..... xV
The features that might make this book appealing ..... xvi
What's in and what's out ..... xvii
Possible courses based on this book ..... xvii
Computer science as a liberal art ..... xviii
OVERVIEW ..... 1
1 INTRODUCTION: WHAT CAN AND CANNOT BE COMPUTED? ..... 3
1.1 Tractable problems ..... 4
1.2 Intractable problems ..... 5
1.3 Uncomputable problems ..... 6
1.4 A more detailed overview of the book ..... 6
Overview of part I: Computability theory ..... 6
Overview of part II: Complexity theory ..... 7
Overview of part III: Origins and applications ..... 8
1.5 Prerequisites for understanding this book ..... 8
1.6 The goals of the book ..... 9
The fundamental goal: What can be computed? ..... 9
Secondary goal 1: A practical approach ..... 9
Secondary goal 2: Some historical insight ..... 10
1.7 Why study the theory of computation? ..... 10
Reason 1: The theory of computation is useful ..... 10
Reason 2: The theory of computation is beautiful and important ..... 11
Exercises ..... 11
Part I: COMPUTABILITY THEORY ..... 13
2 WHAT IS A COMPUTER PROGRAM? ..... 15
2.1 Some Python program basics ..... 15
Editing and rerunning a Python program ..... 17
Running a Python program on input from a file ..... 17
Running more complex experiments on Python programs ..... 18
2.2 SISO Python programs ..... 18
Programs that call other functions and programs ..... 20
2.3 ASCII characters and multiline strings ..... 21
2.4 Some problematic programs ..... 22
2.5 Formal definition of Python program ..... 23
2.6 Decision programs and equivalent programs ..... 25
2.7 Real-world programs versus SISO Python programs ..... 26
Exercises ..... 27
3 SOME IMPOSSIBLE PYTHON PROGRAMS ..... 30
3.1 Proof by contradiction ..... 30
3.2 Programs that analyze other programs ..... 31
Programs that analyze themselves ..... 33
3.3 The program yesonstring.py ..... 33
3.4 The program yesOnSelf.py ..... 34
3.5 The program notYesOnSelf.py ..... 36
3.6 yesonstring.py can't exist either ..... 37
A compact proof that yesOnString.py can't exist ..... 37
3.7 Perfect bug-finding programs are impossible ..... 39
3.8 We can still find bugs, but we can't do it perfectly ..... 41
Exercises ..... 42
4 WHAT IS A COMPUTATIONAL PROBLEM? ..... 45
4.1 Graphs, alphabets, strings, and languages ..... 46
Graphs ..... 46
Trees and rooted trees ..... 49
Alphabets ..... 49
Strings ..... 50
Languages ..... 51
4.2 Defining computational problems ..... 53
Positive and negative instances ..... 55
Notation for computational problems ..... 56
4.3 Categories of computational problems ..... 57
Search problems ..... 57
Optimization problems ..... 57
Threshold problems ..... 57
Function problems ..... 58
Decision problems ..... 58
Converting between general and decision problems ..... 59
Complement of a decision problem ..... 60
Computational problems with two input strings ..... 61
4.4 The formal definition of "solving" a problem ..... 62
Computable functions ..... 63
4.5 Recognizing and deciding languages ..... 63
Recognizable languages ..... 65
Recursive and recursively enumerable languages ..... 66
Exercises ..... 66
5 TURING MACHINES: THE SIMPLEST COMPUTERS ..... 71
5.1 Definition of a Turing machine ..... 72
Halting and looping ..... 76
Accepters and transducers ..... 77
Abbreviated notation for state diagrams ..... 78
Creating your own Turing machines ..... 78
5.2 Some nontrivial Turing machines ..... 79
The morecsThanGs machine ..... 80
The countCs machine ..... 81
Important lessons from the countCs example ..... 85
5.3 From single-tape Turing machines to multi-tape Turing machines ..... 86
Two-tape, single-head Turing machines ..... 87
Two-way infinite tapes ..... 88
Multi-tape, single-head Turing machines ..... 89
Two-tape, two-head Turing machines ..... 90
5.4 From multi-tape Turing machines to Python programs and beyond ..... 91
Multi-tape Turing machine $\rightarrow$ random-access Turing machine ..... 92
Random-access Turing machine $\rightarrow$ real computer ..... 92
Modern computer $\rightarrow$ Python program ..... 95
5.5 Going back the other way: Simulating a Turing machine with Python ..... 95
A serious caveat: Memory limitations and other technicalities ..... 97
5.6 Classical computers can simulate quantum computers ..... 98
5.7 All known computers are Turing equivalent ..... 98
Exercises ..... 99
6 UNIVERSAL COMPUTER PROGRAMS: PROGRAMS THAT CAN DO ANYTHING ..... 103
6.1 Universal Python programs ..... 104
6.2 Universal Turing machines ..... 105
6.3 Universal computation in the real world ..... 107
6.4 Programs that alter other programs ..... 110
Ignoring the input and performing a fixed calculation instead ..... 112
6.5 Problems that are undecidable but recognizable ..... 113
Exercises ..... 114
7 REDUCTIONS: HOW TO PROVE A PROBLEM IS HARD ..... 116
7.1 A reduction for easiness ..... 116
7.2 A reduction for hardness ..... 118
7.3 Formal definition of Turing reduction ..... 120
Why "Turing" reduction? ..... 120
Oracle programs ..... 121
Why is $\leq$ т used to denote a Turing reduction? ..... 121
Beware the true meaning of "reduction" ..... 121
7.4 Properties of Turing reductions ..... 122
7.5 An abundance of uncomputable problems ..... 123
The variants of YesOnSTRING ..... 123
The halting problem and its variants ..... 126
Uncomputable problems that aren't decision problems ..... 128
7.6 Even more uncomputable problems ..... 130
The computational problem COMPUTES $F_{F}$ ..... 132
Rice's theorem ..... 134
7.7 Uncomputable problems that aren't about programs ..... 134
7.8 Not every question about programs is uncomputable ..... 135
7.9 Proof techniques for uncomputability ..... 136
Technique 1: The reduction recipe ..... 137
Technique 2: Reduction with explicit Python programs ..... 138
Technique 3: Apply Rice's theorem ..... 139
Exercises ..... 140
8 NONDETERMINISM: MAGIC OR REALITY? ..... 143
8.1 Nondeterministic Python programs ..... 144
8.2 Nondeterministic programs for nondecision problems ..... 148
8.3 Computation trees ..... 149
8.4 Nondeterminism doesn't change what is computable ..... 153
8.5 Nondeterministic Turing machines ..... 154
8.6 Formal definition of nondeterministic Turing machines ..... 156
8.7 Models of nondeterminism ..... 158
8.8 Unrecognizable problems ..... 158
8.9 Why study nondeterminism? ..... 159
Exercises ..... 160
9 FINITE AUTOMATA: COMPUTING WITH LIMITED RESOURCES ..... 164
9.1 Deterministic finite automata ..... 164
9.2 Nondeterministic finite automata ..... 167
State diagrams for nfas ..... 168
Formal definition of an nfa ..... 169
How does an nfa accept a string? ..... 170
Sometimes nfas make things easier ..... 170
9.3 Equivalence of $n f a s$ and dfas ..... 170
Nondeterminism can affect computability: The example of pdas ..... 173
Practicality of converted nfas ..... 174
Minimizing the size of dfas ..... 175
9.4 Regular expressions ..... 175
Pure regular expressions ..... 176
Standard regular expressions ..... 177
Converting between regexes and finite automata ..... 178
9.5 Some languages aren't regular ..... 181
The nonregular language GnTn ..... 181
The key difference between Turing machines and finite automata ..... 183
9.6 Many more nonregular languages ..... 183
The pumping lemma ..... 185
9.7 Combining regular languages ..... 187
Exercises ..... 188
Part II: COMPUTATIONAL COMPLEXITY THEORY ..... 193
10 COMPLEXITYTHEORY: WHEN EFFICIENCY DOES MATER ..... 195
10.1 Complexity theory uses asymptotic running times ..... 195
10.2 Big-O notation ..... 197
Dominant terms of functions ..... 199
A practical definition of big-O notation ..... 201
Superpolynomial and subexponential ..... 202
Other asymptotic notation ..... 202
Composition of polynomials is polynomial ..... 203
Counting things with big-O ..... 203
10.3 The running time of a program ..... 204
Running time of a Turing machine ..... 204
Running time of a Python program ..... 206
The lack of rigor in Python running times ..... 209
10.4 Fundamentals of determining time complexity ..... 210
A crucial distinction: The length of the input versus the numerical value of the input ..... 210
The complexity of arithmetic operations ..... 212
Beware of constant-time arithmetic operations ..... 214
The complexity of factoring ..... 215
The importance of the hardness of factoring ..... 216
The complexity of sorting ..... 217
10.5 For complexity, the computational model does matter ..... 217
Simulation costs for common computational models ..... 217
Multi-tape simulation has quadratic cost ..... 218
Random-access simulation has cubic cost ..... 219
Universal simulation has logarithmic cost ..... 219
Real computers cost only a constant factor ..... 220
Python programs cost the same as real computers ..... 220
Python programs can simulate random-access Turing machines efficiently ..... 220
Quantum simulation may have exponential cost ..... 220
All classical computational models differ by only polynomial factors ..... 221
Our standard computational model: Python programs ..... 221
10.6 Complexity classes ..... 221
Exercises ..... 224
11 Poly and Expo: The wo most fundamental complexir classes ..... 228
11.1 Definitions of Poly and Expo ..... 228
Poly and Expo compared to P, Exp, and FP ..... 229
11.2 Poly is a subset of Expo ..... 230
11.3 A first look at the boundary between Poly and Expo ..... 231
All3SETS and AllSubsets ..... 231
Traveling salespeople and shortest paths ..... 232
Multiplying and factoring ..... 235
Back to the boundary between Poly and Expo ..... 235
Primality testing is in Poly ..... 237
11.4 Poly and Expo don't care about the computational model ..... 238
11.5 HaltEx: A decision problem in Expo but not Poly ..... 238
11.6 Other problems that are outside Poly ..... 243
11.7 Unreasonable encodings of the input affect complexity ..... 244
11.8 Why study Poly, really? ..... 245
Exercises ..... 246
12 PolyCheck AND NPoly: HARD PROBLEMS THAT ARE EASY TO VERIFY ..... 250
12.1 Verifiers ..... 250
Why "unsure"? ..... 253
12.2 Polytime verifiers ..... 254
Bounding the length of proposed solutions and hints ..... 255
Verifying negative instances in exponential time ..... 255
Solving arbitrary instances in exponential time ..... 255
12.3 The complexity class PolyCheck ..... 256
Some PolyCheck examples: Packing, SubsetSum, and Partition ..... 256
The haystack analogy for PolyCheck ..... 257
12.4 The complexity class NPoly ..... 258
12.5 PolyCheck and NPoly are identical ..... 259
Every PolyCheck problem is in NPoly ..... 259
Every NPoly problem is in PolyCheck ..... 260
12.6 The PolyCheck/NPoly sandwich ..... 263
12.7 Nondeterminism does seem to change what is computable efficiently ..... 264
12.8 The fine print about NPoly ..... 265
An alternative definition of NPoly ..... 265
NPoly compared to NP and FNP ..... 266
Exercises ..... 268
13 POLYNOMIAL-TIME MAPPING REDUCTIONS: PROVING XIS AS EASY AS Y ..... 272
13.1 Definition of polytime mapping reductions ..... 272
Polyreducing to nondecision problems ..... 275
13.2 The meaning of polynomial-time mapping reductions ..... 275
13.3 Proof techniques for polyreductions ..... 276
13.4 Examples of polyreductions using Hamilton cycles ..... 277
A polyreduction from UHC to DHC ..... 278
A polyreduction from DHC to UHC ..... 279
13.5 Three satisfiability problems: CIRCUITSAT, SAT, and 3-SAT ..... 281
Why do we study satisfiability problems? ..... 281
Circuitsat ..... 281
SAT ..... 282
Conjunctive normal form ..... 284
ASCII representation of Boolean formulas ..... 284
3-SAT ..... 285
13.6 Polyreductions between Circuitsat, SAT, and 3-SAT ..... 285
The Tseytin transformation ..... 285
13.7 Polyequivalence and its consequences ..... 290
Exercises ..... 291
14 NP-COMPLETENESS: MOST HARD PROBLEMS ARE EQUALLY HARD ..... 294
14.1 $P$ versus NP ..... 294
14.2 NP-completeness ..... 296
Reformulations of P versus NP using NP-completeness ..... 298
14.3 NP-hardness ..... 298
14.4 Consequences of $\mathrm{P}=\mathrm{NP}$ ..... 301
14.5 CIRCUITSAT is a "hardest" NP problem ..... 302
14.6 NP-completeness is widespread ..... 306
14.7 Proof techniques for NP-completeness ..... 308
14.8 The good news and bad news about NP-completeness ..... 309
Problems in NPoly but probably not NP-hard ..... 309
Some problems that are in $P$ ..... 309
Some NP-hard problems can be approximated efficiently ..... 310
Some NP-hard problems can be solved efficiently for real-world inputs ..... 310
Some NP-hard problems can be solved in pseudo-polynomial time ..... 310
Exercises ..... 311
Part III: ORIGINS AND APPLICATIONS ..... 315
15 THE ORIGINAL TURING MACHINE ..... 317
15.1 Turing's definition of a "computing machine" ..... 318
15.2 Machines can compute what humans can compute ..... 324
15.3 The Church-Turing thesis: A law of nature? ..... 327
The equivalence of digital computers ..... 327
Church's thesis: The equivalence of computer programs and algorithms ..... 328
Turing's thesis: The equivalence of computer programs and human brains ..... 329
Church-Turing thesis: The equivalence of all computational processes ..... 329
Exercises ..... 330
16 YOU CAN'T PROVE EVERYTHING THAT'S TRUE ..... 332
The history of computer proofs ..... 332
16.1 Mechanical proofs ..... 333
Semantics and truth ..... 336
Consistency and completeness ..... 338
Decidability of logical systems ..... 339
16.2 Arithmetic as a logical system ..... 340
Converting the halting problem to a statement about integers ..... 341
Recognizing provable statements about integers ..... 343
The consistency of Peano arithmetic ..... 344
16.3 The undecidability of mathematics ..... 345
16.4 The incompleteness of mathematics ..... 346
16.5 What have we learned and why did we learn it? ..... 349
Exercises ..... 350
17 KARP'S 21 PROBLEMS ..... 353
17.1 Karp's overview ..... 353
17.2 Karp's definition of NP-completeness ..... 355
17.3 The list of 21 NP-complete problems ..... 357
17.4 Reductions between the 21 NP-complete problems ..... 359
Polyreducing Sat to Clique ..... 361
Polyreducing Clique to Node Cover ..... 363
Polyreducing DHC to UHC ..... 364
Polyreducing Sat to 3-SAT ..... 365
Polyreducing Knapsack to Partition ..... 365
17.5 The rest of the paper: NP-hardness and more ..... 367
Exercises ..... 367
18 CONCLUSION: WHAT WILL BE COMPUTED? ..... 370
18.1 The big ideas about what can be computed ..... 370
Bibliography ..... 373
Index ..... 375

