LEARNING ALGORITHMS

THROUGH

PROGRAMMING AND PUZZLE SOLVING

I

O H

L R T M

A G S

by Alexander Kulikov and Pavel PevznerWelcome!

Thank you for joining us! This book powers our popular Data Structures

and Algorithms online specialization on Coursera 1 and online MicroMas-

ters program at edX 2 . We encourage you to sign up for a session and learn

this material while interacting with thousands of other talented students

from around the world. As you explore this book, you will ﬁnd a number

of active learning components that help you study the material at your

own pace.

1. PROGRAMMING CHALLENGES ask you to implement the algo-

rithms that you will encounter in one of programming languages

that we support: C, C++, Java, JavaScript, Python, Scala, C#,

Haskell, Ruby, and Rust (the last four programming languages are

supported by Coursera only). These code challenges are embedded

in our Coursera and edX online courses.

2. ALGORITHMIC PUZZLES provide you with a fun way to “invent”

the key algorithmic ideas on your own! Even if you fail to solve some

puzzles, the time will not be lost as you will better appreciate the

beauty and power of algorithms. These puzzles are also embedded

in our Coursera and edX online courses.

3. EXERCISE BREAKS oﬀer “just in time” assessments testing your

understanding of a topic before moving to the next one.

4. STOP and THINK questions invite you to slow down and contem-

plate the current material before continuing to the next topic.

1 www.coursera.org/specializations/data-structures-algorithms

2 www.edx.org/micromasters/ucsandiegox-algorithms-and-data-structuresLearning Algorithms Through Programming

and Puzzle Solving

Alexander S. Kulikov and Pavel Pevzner

Active Learning Technologies

©2018Copyright © 2018 by Alexander S. Kulikov and Pavel Pevzner. All rights

reserved.

This book or any portion thereof may not be reproduced or used in any

manner whatsoever without the express written permission of the pub-

lisher except for the use of brief quotations in a book review.

ISBN: 978-0-9996762-0-2

Active Learning Technologies

Address:

3520 Lebon Drive

Suite 5208

San Diego, CA 92122, USATo my parents. — A.K.

To my family. — P.P.Contents

About This Book ix

Programming Challenges and Algorithmic Puzzles xii

What Lies Ahead xv

Meet the Authors xvi

Meet Our Online Co-Instructors xvii

Acknowledgments xviii

1 Algorithms and Complexity 1

1.1 What Is an Algorithm? . . . . . . . . . . . . . . . . . . . . . 1

1.2 Pseudocode . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.3 Problem Versus Problem Instance . . . . . . . . . . . . . . . 1

1.4 Correct Versus Incorrect Algorithms . . . . . . . . . . . . . 3

1.5 Fast Versus Slow Algorithms . . . . . . . . . . . . . . . . . . 4

1.6 Big-O Notation . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Algorithm Design Techniques 7

2.1 Exhaustive Search Algorithms . . . . . . . . . . . . . . . . . 7

2.2 Branch-and-Bound Algorithms . . . . . . . . . . . . . . . . 8

2.3 Greedy Algorithms . . . . . . . . . . . . . . . . . . . . . . . 8

2.4 Dynamic Programming Algorithms . . . . . . . . . . . . . . 8

2.5 Recursive Algorithms . . . . . . . . . . . . . . . . . . . . . . 12

2.6 Divide-and-Conquer Algorithms . . . . . . . . . . . . . . . 18

2.7 Randomized Algorithms . . . . . . . . . . . . . . . . . . . . 20

3 Programming Challenges 25

3.1 Sum of Two Digits . . . . . . . . . . . . . . . . . . . . . . . . 26

3.2 Maximum Pairwise Product . . . . . . . . . . . . . . . . . . 29

3.2.1 Naive Algorithm . . . . . . . . . . . . . . . . . . . . . 30

3.2.2 Fast Algorithm . . . . . . . . . . . . . . . . . . . . . . 34

3.2.3 Testing and Debugging . . . . . . . . . . . . . . . . . 34

vvi Contents

3.2.4 Can You Tell Me What Error Have I Made? . . . . . . 36

3.2.5 Stress Testing . . . . . . . . . . . . . . . . . . . . . . 37

3.2.6 Even Faster Algorithm . . . . . . . . . . . . . . . . . 41

3.2.7 A More Compact Algorithm . . . . . . . . . . . . . . 42

3.3 Solving a Programming Challenge in Five Easy Steps . . . . 42

3.3.1 Reading Problem Statement . . . . . . . . . . . . . . 42

3.3.2 Designing an Algorithm . . . . . . . . . . . . . . . . 43

3.3.3 Implementing an Algorithm . . . . . . . . . . . . . . 43

3.3.4 Testing and Debugging . . . . . . . . . . . . . . . . . 44

3.3.5 Submitting to the Grading System . . . . . . . . . . 45

3.4 Good Programming Practices . . . . . . . . . . . . . . . . . 45

4 Algorithmic Warm Up 53

4.1 Fibonacci Number . . . . . . . . . . . . . . . . . . . . . . . . 54

4.2 Last Digit of Fibonacci Number . . . . . . . . . . . . . . . . 56

4.3 Greatest Common Divisor . . . . . . . . . . . . . . . . . . . 58

4.4 Least Common Multiple . . . . . . . . . . . . . . . . . . . . 59

4.5 Fibonacci Number Again . . . . . . . . . . . . . . . . . . . . 60

4.6 Last Digit of the Sum of Fibonacci Numbers . . . . . . . . . 62

4.7 Last Digit of the Sum of Fibonacci Numbers Again . . . . . 63

5 Greedy Algorithms 65

5.1 Money Change . . . . . . . . . . . . . . . . . . . . . . . . . . 66

5.2 Maximum Value of the Loot . . . . . . . . . . . . . . . . . . 69

5.3 Maximum Advertisement Revenue . . . . . . . . . . . . . . 71

5.4 Collecting Signatures . . . . . . . . . . . . . . . . . . . . . . 73

5.5 Maximum Number of Prizes . . . . . . . . . . . . . . . . . . 75

5.6 Maximum Salary . . . . . . . . . . . . . . . . . . . . . . . . . 77

6 Divide-and-Conquer 81

6.1 Binary Search . . . . . . . . . . . . . . . . . . . . . . . . . . 82

6.2 Majority Element . . . . . . . . . . . . . . . . . . . . . . . . 85

6.3 Improving QuickSort . . . . . . . . . . . . . . . . . . . . . . 87

6.4 Number of Inversions . . . . . . . . . . . . . . . . . . . . . . 88

6.5 Organizing a Lottery . . . . . . . . . . . . . . . . . . . . . . 90

6.6 Closest Points . . . . . . . . . . . . . . . . . . . . . . . . . . 92Contents vii

7 Dynamic Programming 97

7.1 Money Change Again . . . . . . . . . . . . . . . . . . . . . . 98

7.2 Primitive Calculator . . . . . . . . . . . . . . . . . . . . . . . 99

7.3 Edit Distance . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

7.4 Longest Common Subsequence of Two Sequences . . . . . . 103

7.5 Longest Common Subsequence of Three Sequences . . . . . 105

7.6 Maximum Amount of Gold . . . . . . . . . . . . . . . . . . . 107

7.7 Partitioning Souvenirs . . . . . . . . . . . . . . . . . . . . . 109

7.8 Maximum Value of an Arithmetic Expression . . . . . . . . 111

Appendix 113

Compiler Flags . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

Frequently Asked Questions . . . . . . . . . . . . . . . . . . . . . 114viii ContentsAbout This Book

I ﬁnd that I don’t understand things unless I try to program them.

—Donald E. Knuth, The Art of Computer Programming, Volume 4

There are many excellent books on Algorithms — why in the world we

would write another one???

Because we feel that while these books excel in introducing algorith-

mic ideas, they have not yet succeeded in teaching you how to implement

algorithms, the crucial computer science skill.

Our goal is to develop an Intelligent Tutoring System for learning algo-

rithms through programming that can compete with the best professors in

a traditional classroom. This MOOC book is the ﬁrst step towards this goal

written speciﬁcally for our Massive Open Online Courses (MOOCs) form-

ing a specialization “Algorithms and Data Structures” on Coursera plat-

form 3 and a microMasters program on edX platform 4 . Since the launch

of our MOOCs in 2016, hundreds of thousand students enrolled in this

specialization and tried to solve more than hundred algorithmic program-

ming challenges to pass it. And some of them even got oﬀers from small

companies like Google after completing our specialization!

In the last few years, some professors expressed concerns about the

pedagogical quality of MOOCs and even called them the “junk food of ed-

ucation.” In contrast, we are among the growing group of professors who

believe that traditional classes, that pack hundreds of students in a single

classroom, represent junk food of education. In a large classroom, once

a student takes a wrong turn, there are limited opportunities to ask a ques-

tion, resulting in a learning breakdown, or the inability to progress further

without individual guidance. Furthermore, the majority of time a student

invests in an Algorithms course is spent completing assignments outside

the classroom. That is why we stopped giving lectures in our oﬄine classes

(and we haven’t got ﬁred yet :-). Instead, we give ﬂipped classes where stu-

dents watch our recorded lectures, solve algorithmic puzzles, complete

programming challenges using our automated homework checking sys-

tem before the class, and come to class prepared to discuss their learning

3 www.coursera.org/specializations/data-structures-algorithms

4 www.edx.org/micromasters/ucsandiegox-algorithms-and-data-structures

ixx About This Book

breakdowns with us.

When a student suﬀers a learning breakdown, that student needs im-

mediate help in order to proceed. Traditional textbooks do not provide

such help, but our automated grading system described in this MOOC

book does! Algorithms is a unique discipline in that students’ ability to

program provides the opportunity to automatically check their knowl-

edge through coding challenges. These coding challenges are far superior

to traditional quizzes that barely check whether a student fell asleep. In-

deed, to implement a complex algorithm, the student must possess a deep

understanding of its underlying algorithmic ideas.

We believe that a large portion of grading in thousands of Algorithms

courses taught at various universities each year can be consolidated into

a single automated system available at all universities. It did not escape

our attention that many professors teaching algorithms have implemented

their own custom-made systems for grading student programs, an illus-

tration of academic ineﬃciency and lack of cooperation between various

instructors. Our goal is to build a repository of algorithmic programming

challenges, thus allowing professors to focus on teaching. We have already

invested thousands of hours into building such a system and thousands

students in our MOOCs tested it. Below we brieﬂy describe how it works.

When you face a programming challenge, your goal is to implement

a fast and memory-eﬃcient algorithm for its solution. Solving program-

ming challenges will help you better understand various algorithms and

may even land you a job since many high-tech companies ask applicants

to solve programming challenges during the interviews. Your implemen-

tation will be checked automatically against many carefully selected tests

to verify that it always produces a correct answer and ﬁts into the time

and memory constrains. Our system will teach you to write programs that

work correctly on all of our test datasets rather than on some of them. This

is an important skill since failing to thoroughly test your programs leads

to undetected bugs that frustrate your boss, your colleagues, and, most

importantly, users of your programs.

You maybe wondering why it took thousands of hours to develop such

a system. First, we had to build a Compendium of Learning Breakdowns

for each programming challenge, 10–15 most frequent errors that stu-

dents make while solving it. Afterwards, we had to develop test cases

for each learning breakdown in each programming challenge, over 20000

test cases for just 100 programming challenges in our specialization.About This Book xi

We encourage you to sign up for our Algorithms and Data Structures

specialization on Coursera or MicroMasters program on edX and start in-

teracting with thousands of talented students from around the world who

are learning algorithms. Thank you for joining us!xii Programming Challenges and Algorithmic Puzzles

Programming Challenges and Algorithmic

Puzzles

This edition introduces basic algorithmic techniques using 29 program-

ming challenges represented as icons below:

SumofTwoDigits Maximum FibonacciNumber LastDigit Greatest

PairwiseProduct ofFibonacci CommonDivisor

5 6 2 7 4 Number 10

5 30103520 3 2

2+3 = 5 6 30 124224 5 F 170 =150804340016

2 1012 7 4 1 2 807970735635

1 273952047185 6

7 354214 28

4 2024 8 28

LeastCom- Fibonacci Sumof PartialSumof MoneyChange

monMultiple NumberAgain FibonacciNumbers FibonacciNumbers

0

3 1 0 1

6 13 1 1+1+2+3+5+8=20 2+3+5+8+13=31 ¢1 ¢5 ¢10

15 30 2 2 8 F n mod3 1 1

5 10 5 3 2 2

2

0

MaximumValue Maximum Collecting Maximum MaximumSalary

oftheLoot Advertisement Signatures NumberofPrizes

Revenue Resume

clicks prices 8

10 2 1 2 5

20 3

30 5

BinarySearch MajorityElement Improving Numberof Organizing

QuickSort Inversion aLottery

1 3 7 8 9 1215

1 3 7 8 9 1215 3 2 5 9 4

1 3 7 8 9 1215 1 0 2 1

ClosestPoints Money PrimitiveCalculator EditDistance LongestCommon

ChangeAgain Subsequenceof

short TwoSequences

¢1 1 hort

+1 7 2 9 3 1 5 9 4

¢3 port

×2 2 8 1 3 9 7

ports

×3

¢4

LongestCommon Maximum Partitioning MaximumValue

Subsequenceof AmountofGold Souvenirs ofanArith-

ThreeSequences meticExpression

8 3 2 1 7 3 3 6 4 1 9 6 9 1 ((8 − 5) × 3)=9

8 2 1 3 8 10 7 (8 − (5 × 3))= −7

6 8 3 1 4 7Programming Challenges and Algorithmic Puzzles xiii

You are also welcome to solve the following algorithmic puzzles avail-

able at http://dm.compsciclub.ru/app/list:

Book Sorting. Rearrange books on the shelf (in the increasing

7 9 4 5 3 1 6 2 8 order of heights) using minimum number of swaps.

Map Coloring. Use minimum number of colors such that

neighboring countries are assigned diﬀerent colors and each

country is assigned a single color.

Eight Queens. Place eight queens on the chessboard such that

no two queens attack each other (a queen can move horizon-

tally, vertically, or diagonally).

Clique Finding. Find the largest group of mutual friends

(each pair of friends is represented by an edge).

Hanoi Towers. Move all disks from one peg to another using

a minimum number of moves. In a single move, you can move

a top disk from one peg to any other peg provided that you

don’t place a larger disk on the top of a smaller disk.

Icosian Game. Find a cycle visiting each node exactly once.

Guarini Puzzle. Exchange the places of the white knights and

the black knights. Two knights are not allowed to occupy the

same cell of the chess board.

Room Assignment. Place each student in one of her/his

preferable rooms in a dormitory so that each room is occupied

by a single student (preferable rooms are shown by edges).xiv Programming Challenges and Algorithmic Puzzles

Tree Construction. Remove the minimum number of edges

from the graph to make it acyclic.

? ? ? ? Number of Paths. Find out how many paths are there to get

? ? ? ? from the bottom left circle to any other circle and place this

1 2 ? ?

0 1 ? ? number inside the corresponding circle.

Black and White Squares. Use the minimum number of

questions “What is the color of this square?” to ﬁnd two

neighboring squares of diﬀerent colors. The leftmost square

is white, the rightmost square is black, but the colors of all

other squares are unknown.

Twenty One Questions Game. Find an unknown integer 1 ≤

21 x ≤ N by asking the minimum number of questions “Is x = y?”

questions (for any 1 ≤ y ≤ N ). Your opponent will reply either “Yes”, or

“x < y”, or “x > y.”

Antique Calculator. Find the minimum number of opera-

+1 1 tions needed to get a positive integer n from the integer 1 us-

×2 ing only three operations: add 1, multiply by 2, or multiply

×3 by 3.

Subway Lines. You are planning a subway system where the

subway lines should not cross. Can you connect each pair of

the ﬁve stations except for a single pair?

Two Rocks Game. There are two piles of ten rocks. In each

turn, you and your opponent may either take one rock from

a single pile, or one rock from both piles. Your opponent

moves ﬁrst and the player that takes the last rock wins the

game. Design a winning strategy.

Three Rocks Game. There are two piles of ten rocks. In each

turn, you and your opponent may take up to three rocks. Your

opponent moves ﬁrst and the player that takes the last rock

wins the game. Design a winning strategy.What Lies Ahead xv

What Lies Ahead

Watch for our future editions that will cover the following topics.

Data Structures

Arrays and Lists

Priority Queues

Disjoint Sets

Hash Tables

Binary Search Trees

Algorithms on Graphs

Graphs Decomposition

Shortest Paths in Graphs

Minimum Spanning Trees

Shortest Paths in Real Life

Algorithms on Strings

Pattern Matching

Suﬃx Trees

Suﬃx Arrays

Burrows–Wheeler Transform

Advanced Algorithms and Complexity

Flows in Networks

Linear Programmings

NP-complete Problems

Coping with NP-completeness

Streaming Algorithmsxvi Meet the Authors

Meet the Authors

Alexander S. Kulikov is a senior research fellow

at Steklov Mathematical Institute of the Russian

Academy of Sciences, Saint Petersburg, Russia and

a lecturer at the Department of Computer Science and

Engineering at University of California, San Diego,

USA. He also directs the Computer Science Center in

Saint Petersburg that provides free advanced computer

science courses complementing the standard univer-

sity curricula. Alexander holds a Ph. D. from Steklov

Mathematical Institute. His research interests include

algorithms and complexity theory. He co-authored

online courses “Data Structures and Algorithms” and

“Introduction to Discrete Mathematics for Computer

Science” that are available at Coursera and edX.

Pavel Pevzner is Ronald R. Taylor Professor of Com-

puter Science at the University of California, San

Diego. He holds a Ph. D. from Moscow Institute

of Physics and Technology, Russia and an Honorary

Degree from Simon Fraser University. He is a Howard

Hughes Medical Institute Professor (2006), an As-

sociation for Computing Machinery Fellow (2010),

an International Society for Computational Biology

Fellow (2012), and a Member of the the Academia

Europaea (2016). He has authored the textbooks

Computational Molecular Biology: An Algorithmic

Approach (2000), An Introduction to Bioinformatics

Algorithms (2004) (jointly with Neil Jones), and Bioin-

formatics Algorithms: An Active Learning Approach

(2014) (jointly with Phillip Compeau). He co-authored

online courses “Data Structures and Algorithms”,

“Bioinformatics”, and “Analyze Your Genome!” that

are available at Coursera and edX.Meet Our Online Co-Instructors xvii

Meet Our Online Co-Instructors

Daniel Kane is an associate professor at the University

of California, San Diego with a joint appointment

between the Department of Computer Science and

Engineering and the Department of Mathematics. He

has diverse interests in mathematics and theoretical

computer science, though most of his work ﬁts into the

broad categories of number theory, complexity theory,

or combinatorics.

Michael Levin is an Associate Professor at the

Computer Science Department of Higher School of

Economics, Moscow, Russia and the Chief Data Sci-

entist at the Yandex.Market, Moscow, Russia. He also

teaches Algorithms and Data Structures at the Yandex

School of Data Analysis.

Neil Rhodes is a lecturer in the Computer Science and

Engineering department at the University of Califor-

nia, San Diego and formerly a staﬀ software engineer

at Google. Neil holds a B.A. and M.S. in Computer

Science from UCSD. He left the Ph.D. program at

UCSD to start a company, Palomar Software, and spent

ﬁfteen years writing software, books on software de-

velopment, and designing and teaching programming

courses for Apple and Palm. He’s taught Algorithms,

Machine Learning, Operating Systems, Discrete Math-

ematics, Automata and Computability Theory, and

Software Engineering at UCSD and Harvey Mudd

College in Claremont, California.xviii Acknowledgments

Acknowledgments

This book was greatly improved by the eﬀorts of a large number of indi-

viduals, to whom we owe a debt of gratitude.

Our co-instructors and partners in crime Daniel Kane, Michael Levin,

and Neil Rhodes invested countless hours in the development of our on-

line courses at Coursera and edX platforms.

Hundreds of thousands of our online students provided valuable feed-

back that led to many improvements in our MOOCs and this MOOC book.

In particular, we are grateful to the mentors of the Algorithmic Toolbox

class at Coursera: Ayoub Falah, Denys Diachenko, Kishaan Jeeveswaran,

Irina Pinjaeva, Fernando Gonzales Vigil Richter, and Gabrio Secco.

We thank our colleagues who helped us with preparing programming

challenges: Maxim Akhmedov, Roman Andreev, Gleb Evstropov, Niko-

lai Karpov, Sergey Poromov, Sergey Kopeliovich, Ilya Kornakov, Gennady

Korotkevich, Paul Melnichuk, and Alexander Tiunov.

We are grateful to Anton Konev for leading the development of inter-

active puzzles as well as Anton Belyaev and Kirill Banaru for help with

some of the puzzles.

We thank Alexey Kladov and Alexander Smal for help with the “Good

Programming Practices” section of the book.

Randall Christopher brought to life our idea for the textbook cover.

Finally, our families helped us preserve our sanity when we were work-

ing on this MOOC book.

A. K. and P. P.

Saint Petersburg and San Diego

December 2017Chapter 1: Algorithms and Complexity

This book presents programming challenges that will teach you how to

design and implement algorithms. Solving a programming challenge is

one of the best ways to understand an algorithm’s design as well as to

identify its potential weaknesses and ﬁx them.

1.1 What Is an Algorithm?

Roughly speaking, an algorithm is a sequence of instructions that one

must perform in order to solve a well-formulated problem. We will spec-

ify problems in terms of their inputs and their outputs, and the algorithm

will be the method of translating the inputs into the outputs. A well-

formulated problem is unambiguous and precise, leaving no room for mis-

interpretation.

After you designed an algorithm, two important questions to ask are:

“Does it work correctly?” and “How much time will it take?” Certainly

you would not be satisﬁed with an algorithm that only returned correct

results half the time, or took 1000 years to arrive at an answer.

1.2 Pseudocode

To understand how an algorithm works, we need some way of listing the

steps that the algorithm takes, while being neither too vague nor too for-

mal. We will use pseudocode, a language computer scientists often use

to describe algorithms. Pseudocode ignores many of the details that are

required in a programming language, yet it is more precise and less am-

biguous than, say, a recipe in a cookbook.

1.3 Problem Versus Problem Instance

A problem describes a class of computational tasks. A problem instance is

one particular input from that class. To illustrate the diﬀerence between

a problem and an instance of a problem, consider the following example.

You ﬁnd yourself in a bookstore buying a book for $4.23 which you pay

12 Chapter 1. Algorithms and Complexity

for with a $5 bill. You would be due 77 cents in change, and the cashier

now makes a decision as to exactly how you get it. You would be annoyed

at a ﬁstful of 77 pennies or 15 nickels and 2 pennies, which raises the

question of how to make change in the least annoying way. Most cashiers

try to minimize the number of coins returned for a particular quantity of

change. The example of 77 cents represents an instance of the Change

Problem, which we describe below.

The example of 77 cents represents an instance of the Change Problem

that assumes that there are d denominations represented by an array

c = (c 1 ,c 2 ,...,c d ). For simplicity, we assume that the denominations are

given in decreasing order of value. For example, c = (25,10,5,1) for

United States denominations.

Change Problem

Convert some amount of money into given denominations, using the smallest

possible number of coins.

Input: An integer money and an array of d denominations c =

(c 1 ,c 2 ,...,c d ), in decreasing order of value (c 1 > c 2 > ··· > c d ).

Output: A list of d integers i 1 ,i 2 ,...,i d such that c 1 · i 1 + c 2 · i 2 +

··· + c d · i d = money, and i 1 + i 2 + ··· + i d is as small as possible.

The algorithm that is used by cashiers all over the world to solve this

problem is simple:

Change