

Algorithms and Data Structures

Lucien Sina

9.12.2024

Contents

1	Introduction	5
2	Algorithms and Their Analysis	7
2.1	Abstraction Levels of Problem Descriptions	7
2.1.1	Example: Membership Check in a Set	7
2.2	Assessment of Algorithm Quality	8
2.2.1	Correctness	9
2.2.2	Understandability	9
2.2.3	Ease of Implementation	9
2.2.4	Efficiency (Runtime and Memory)	9
2.2.5	Trade-offs Between Criteria	9
2.3	Runtime and Memory of Algorithms	9
2.3.1	Measurement of Runtime	9
2.4	Constructing a Runtime Function That Makes Sense	11
2.4.1	Introduction	11
2.4.2	Abstraction of Runtime	11
2.4.3	Case Study: Searching in an Array	12
2.4.4	Using O-Notation	13
2.4.5	Exercises and Solutions	13
2.4.6	Final Word	15
2.5	Calculation Rules and Runtimes of Basic Control Structures	15
2.5.1	Elementary Operations and Sequences	15
2.5.2	Runtime of Some Basic Control Structures	16
2.6	Improving an Algorithm	17
2.6.1	Runtime Analysis	17
2.6.2	Improved Algorithm with Early Termination	18
2.6.3	Runtime Analysis	18
2.6.4	Optimized Algorithm with Binary Search	18
2.6.5	Runtime Analysis	19
2.6.6	Demonstration	19
2.7	General O-Notation	19
2.7.1	Definitions	20
2.7.2	Relations Between Functions	20
2.7.3	Example: Comparing $\log n$ and \sqrt{n}	21

2.7.4	Example: Runtime of <code>contains</code> and <code>contains₂</code>	22
3	Data Structures, Algebras, Abstract Data Types	25
3.1	Contexts	25
3.2	Abstraction Levels	25
3.2.1	Algebra as Data Type	25
3.2.2	Next Steps: Formalizing Data Types and Algebras	26
3.3	Algebras and Data Types	26
3.3.1	Example Signature	26
3.3.2	Two Approaches for Specification	27
3.4	Exercise: Algebra for a Counter	29
3.5	Polymorphic and Monomorphic Data Types	30
4	Basic Concepts	33
4.1	Algorithm	33
4.1.1	Key Concepts	34
4.2	Definitions of Algebra, ADT, and Model	35
4.3	Exercises and Solutions for Chapter 1: Basic Concepts	35
4.3.1	Exercise 1.1: Algorithm	35
4.3.2	Exercise 1.2: Signature, Algebra, and ADT	36
4.3.3	Exercise 1.3: Data Types and Data Structures	36
4.3.4	Exercise 1.4: Monomorphic and Polymorphic ADTs	37
4.3.5	Exercise 1.5: Implementation of Data Structures	37
5	Programming and Data Structures	39
5.1	Construction of Data Structures	39
5.2	Programming Languages	40
5.3	Data Types in Java	40
5.3.1	Primitive Data Types	40
5.4	Primitive Data Types in Java	41
5.5	Arrays	42
5.5.1	Example	42
5.5.2	Value Range of an Array Type	42
5.5.3	Address Calculation	43
5.5.4	Efficiency of Array Access	43
5.5.5	Arrays as Representational Tools	43
5.5.6	Example: Iterating Through an Array	43
5.5.7	Specific Array Ranges	43
5.5.8	Java-Specific Behavior	43
5.6	Classes	44
5.6.1	Structure of a Class Definition	44
5.6.2	Methods in Classes	45
5.6.3	Realizing Algebra or Abstract Data Types	45
5.7	Records	45
5.7.1	Definition of Records	45
5.7.2	Examples of Record Definitions	46

5.7.3	Operations on Records	46
5.7.4	Value Range of a Record Type	47
5.7.5	Representation and Address Calculation	47
5.7.6	Constant Time Access	47
6	Dynamic Data Structures	49
6.1	Pointer Types	49
6.1.1	Definition of Pointer Types	49
6.1.2	Pointer Variable Declaration and Usage	49
6.1.3	Graphical Representation of Pointers	50
6.1.4	Importance of Pointers in Dynamic Data Structures	50
6.2	Dereferencing and Garbage Collection	50
6.3	Reference Types in Java	52
6.3.1	Assignments	53
6.4	Method Calls in Java	54
7	Additional data structures	57
7.1	Enumeration Types	57
7.2	Subrange Types	59
7.3	Sets	60
8	Elementary Data Types	63
8.1	Lists	63
8.1.1	Lists with First, Rest, Append and Concat	63
8.2	Lists with Explicit Positions	64
9	List Implementations	69
9.1	Double-Linked List	69
9.2	Implementation of Double-Linked List Operations	70
9.2.1	Creating an Empty List	70
9.2.2	Accessing the First Position	71
9.2.3	Inserting Elements	71
9.2.4	Concatenating Lists	71
9.2.5	Finding Elements	72
9.2.6	Deleting Elements	73
9.3	Simple Linked List	73
9.4	Linked Lists in Arrays	76
10	Stacks	79
10.1	Stack Specification	79
10.2	Stack Implementation	80
11	Queues	83
11.1	Queue Operations and Extensions	83
11.2	Algebraic Specification of Queues	83
11.3	Queue Signature	84
11.3.1	Diagram Representation	84

11.4 Applications of Queues	84
11.5 Java Queue Implementation	84
11.5.1 Queue Implementation Using a Cyclic Array	85
11.5.2 Queue Implementation Using Two Stacks	86
11.5.3 Comparison of the Two Implementations	87
12 Mappings	89
12.1 Characteristics of Mappings	89
12.2 Finite and Scalar Domain Mappings	89
12.3 Large or Sparse Domains	90
12.4 Summary of Mapping Implementations	91
13 Binary Trees	93
13.1 Definition	93
13.1.1 Binary Trees	94
13.1.2 Leaves and Subtrees	94
13.1.3 Nodes and Their Classification	95
13.1.4 Paths, Ancestors, and Descendants	95
13.1.5 Subtrees and Path Lengths	96
13.1.6 Height and Depth	96
13.2 Characteristics of Binary Trees	97
13.2.1 Maximum Height of a Binary Tree	97
13.2.2 Minimal Height of a Binary Tree	98
13.2.3 Exact Minimal Height of a Binary Tree	98
13.2.4 Relationship Between Leaves and Inner Nodes	99
13.2.5 Summary of Characteristics	100
13.3 Algebra Tree	100
13.3.1 Sorts	100
13.3.2 Operations	100
13.3.3 Set Definition	100
13.3.4 Function Definitions	100
13.3.5 Recursive Structures	101
13.3.6 Tree Traversals	101
13.3.7 Exercise: Computing Tree Height	101
13.4 Binary Tree Java Implementations	102
13.4.1 With Pointers	102
13.4.2 With Array Embedding	103
13.4.3 Complexity Comparison	104
14 General Trees	107
14.1 Definition	107
14.1.1 Illustration	107
14.2 General Tree Implementations	109
14.2.1 Implementation with Arrays	109
14.2.2 Implementation with Binary Trees	109

15 Exercises	111
15.1 Exercise 1: Algebra for a Cyclical List (Ring)	111
15.2 Exercise 2: Java Implementation of List ₂ Operations	112
15.3 Exercise 3: Representing and Differentiating Polynomials with List ₁	114
15.4 Exercise 4: Polynomial Addition Using a Linked List	114
15.5 Exercise 5: Implementing Stack and Queue Using List ₂	116
15.6 Exercise 6: Extending the Algebraic Specification of Trees	118
15.7 Exercise 7: Ancestor Check Using Tree Traversals	118
15.8 Exercise 8: Non-Recursive Tree Traversal Using Stacks	119
16 Final Words	121
16.1 Recommended Books by Lucien Sina	121
16.1.1 Logic	121
16.1.2 Programming	121
16.2 Literature	122

© 2025 by Lucien Sina. All rights reserved.

ISBN: 9789403830537

Printed by Bookmundo

Preface

Efficient algorithms and data structures lie at the heart of computer science, forming the foundation of solving complex problems in programming and beyond. The ability to design, analyze, and implement algorithms is essential for tackling the diverse challenges faced by programmers. This book aims to equip readers with the tools and insights necessary to navigate this fascinating field.

To address the core problem areas in computer science, programmers must learn to create new algorithms by skillfully combining existing ones. Equally important is the ability to analyze these algorithms in terms of runtime efficiency and memory usage. Data structures, which organize information to enable efficient algorithmic operations, are another crucial focus of this book.

A clear distinction will be made between **data types**—the abstract, specification-oriented view of organizing data—and **data structures**, which represent the concrete implementations of these data types. Readers will discover that a single data type may have multiple implementations, each with its own trade-offs. By exploring these distinctions, this book will help develop a deep understanding of how to select and design efficient solutions for different scenarios.

To fully benefit from this book, a basic proficiency in programming is recommended. Familiarity with a language like Java will be particularly useful. For readers seeking foundational programming skills, I encourage consulting my previous book on programming, which introduces the essential concepts required for effective coding. Additionally, while some mathematical skills can enhance your understanding, I have made a conscious effort to develop all necessary mathematical concepts as part of this book.

To reinforce learning, the book provides numerous exercises, complete with solutions. These exercises are designed to solidify the theoretical concepts and give hands-on experience with practical implementations.

Whether you are a student, a programmer, or simply curious about algorithms and data structures, this book is structured to guide you through the subject step by step. I hope it serves as a valuable resource in your journey into this essential area of computer science.

Chapter 1

Introduction

Algorithms operate on data structures, and data structures contain algorithms as components. Consequently, algorithms and data structures are inseparably linked. In this book, we aim to classify algorithms and data structures within the context of related concepts such as functions, procedures, abstract data types, data types, algebras, types, classes, and modules.

At the most abstract level, we employ mathematics to formalize the specifications of algorithms and data structures. An algorithm realizes a function, which serves as a specification for that algorithm. Likewise, an algorithm itself is a specification of a procedure, function, or method that must eventually be implemented.

It is important to note that algorithms are typically not presented as computer programs. Instead, they are described on a higher, more human-readable level to facilitate communication. However, a computer program *can* serve as one way to describe an algorithm. In other words, a computer program **is** an algorithm, but the description of an algorithm is usually not limited to being a computer program.

In the first chapter, we will focus on the analysis of algorithms, specifically examining their runtime and memory usage.

On the side of data structures, we begin with abstract concepts such as **abstract data types** and **algebras**, which define concrete data types. A data structure is an implementation of an algebra or an abstract data type at the algorithmic level. Data structures themselves can then be implemented in programming languages. At the programming level, we encounter key concepts such as data types, classes, and modules.

This structured approach allows us to bridge the gap between high-level abstract concepts and their concrete implementations in programming, providing a comprehensive understanding of algorithms and data structures.