## ART

\_ \_ . \_ . .

# ALGORITHM ANALYSIS

FROM FOUNDATIONS TO PRACTICE

 $\Theta(\log n)$ 

<sup>Ω(n²)</sup> Mahdi

LIVING FIRST EDITION

# Ahlaly

## ALGORITHMS • ABSTRACTION • ANALYSIS • ART

"From ancient counting stones to quantum algorithms every data structure tells the story of human ingenuity."

## LIVING FIRST EDITION

Updated October 18, 2025

© 2025 Mahdi

CREATIVE COMMONS • OPEN SOURCE

## LICENSE & DISTRIBUTION

## THE ART OF ALGORITHMIC ANALYSIS: ALGORITHMIC COST ANALYSIS AND ASYMPTOTIC REASONING

A Living Architecture of Computing

The Art of Algorithmic Analysis is released under the Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).

## FORMAL LICENSE TERMS

## Copyright © 2025 Mahdi

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

License URL: https://creativecommons.org/licenses/by-sa/4.0/

## You are free to:

- **Share** copy and redistribute the material in any medium or format for any purpose, even commercially.
- **Adapt** remix, transform, and build upon the material for any purpose, even commercially.

## Under the following terms:

- Attribution You must give appropriate credit to Mahdi, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- **No additional restrictions** You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

### **DISTRIBUTION & SOURCE ACCESS**

**Repository:** The complete source code (LaTeX, diagrams, examples) is available at:

https://github.com/m-mdy-m/algorithms-data-structures/tree/main/books/books

## **Preferred Citation Format:**

Mahdi. (2025). The Art of Algorithmic Analysis. Retrieved from

https://github.com/m-mdy-m/algorithms-data-structures/tree/main/books/books

**Version Control:** This is a living document. Check the repository for the most current version and revision history.

### **WARRANTIES & DISCLAIMERS**

**No Warranty:** This work is provided "AS IS" without warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

**Limitation of Liability:** In no event shall Mahdi be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising from the use of this work.

**Educational Purpose:** This work is intended for educational and research purposes. Practical implementation of algorithms and techniques should be thoroughly tested and validated for production use.

## TECHNICAL SPECIFICATIONS

Typeset with: LATEX using Charter and Palatino font families

**Graphics:** TikZ and custom illustrations

Standards: Follows academic publishing conventions

**Encoding:** UTF-8 with full Unicode support

Format: Available in PDF, and LaTeX source formats

License last updated: October 18, 2025

For questions about licensing, contact: bitsgenix@gmail.com

## **Contents**

H	itie P	age				
C	onte	nts		iii		
Р	refac	e		vii		
Α	ckno	wledg	jmentsx	xii		
I	Fo	unda	ntions	1		
1	Mathematical and Algorithmic Prerequisites					
	1.1	Discre	ete Mathematics: The Language of Algorithms	4		
		1.1.1	Sets, Functions, and Relations	5		
		1.1.2	Combinatorics: Permutations, Combinations, and Binomial Coefficients .	5		
		1.1.3	Graph Theory Basics	5		
		1.1.4	Proof Techniques: Induction, Contradiction, and Contrapositive	5		
	1.2	Eleme	entary Probability Theory	5		
		1.2.1	Sample Spaces, Events, and Probability Measures	5		
		1.2.2	Random Variables and Expectations	5		
		1.2.3	Basic Distributions: Uniform, Bernoulli, Geometric, Binomial	5		
		1.2.4	Linearity of Expectation	5		
		1.2.5	Conditional Probability and Independence	5		
		1.2.6	Variance and Standard Deviation	5		
		1.2.7	Moment Generating Functions (Brief Introduction)	5		
	1.3	Mathe	ematical Analysis	5		
		1.3.1	Limits, Continuity, and Asymptotic Behavior	5		
		1.3.2	Sequences and Series	5		
		1.3.3	Summations and Closed Forms	5		
		1.3.4	Integration and Differentiation (Brief Review)	5		
		1.3.5	Taylor Series and Asymptotic Expansions	5		
		1.3.6	Stirling's Approximation	5		
	1.4	Linear	Algebra (Brief Overview)	5		
		1.4.1	Vectors, Matrices, and Linear Transformations	5		
		1.4.2	Eigenvalues and Eigenvectors	5		
		1.4.3	Applications to Markov Chains and Graph Algorithms	5		

		1.4.4	Matrix Operations and Complexity	5
	1.5	Numbe	er Theory Essentials	5
		1.5.1	Divisibility and Modular Arithmetic	5
		1.5.2	Prime Numbers and Factorization	5
		1.5.3	Greatest Common Divisor and Euclidean Algorithm	5
		1.5.4	Applications to Cryptography and Hashing	5
	1.6	Additio	nal Mathematical Tools	5
		1.6.1	Logarithms and Exponentials	5
		1.6.2	Floor and Ceiling Functions	5
		1.6.3	Asymptotic Notation (Informal Preview)	5
	1.7	Self-As	ssessment Exercises	5
		1.7.1	Discrete Mathematics Problems	5
		1.7.2	Probability Problems	5
		1.7.3	Analysis Problems	5
		1.7.4	Linear Algebra Problems	5
		1.7.5	Number Theory Problems	5
	1.8	Furthe	r Reading and Resources	5
		1.8.1	Recommended Textbooks for Each Topic	5
		1.8.2	Online Resources and Video Lectures	5
		1.8.3	Practice Problem Collections	5
ı	Fo	unds	ations of Algorithmic Analysis	6
2	Intr	oducti	on to Algorithm Analysis	7
	2.1	What Is	s Algorithm Analysis?	7
		2.1.1	Correctness vs. Efficiency	7
		2.1.2	Resource Measures: Time, Space, Energy, I/O	7
		2.1.3	The Need for Mathematical Models	7
	2.2	The RA	AM Model of Computation	7
		2.2.1	Basic Operations and Unit-Cost Assumption	7
		2.2.2	Memory Access Model	7
		2.2.3	Limitations and Extensions of the RAM Model	7
	2.3	Measu	ring Algorithm Performance	7
		2.3.1	Input Size and Problem Instances	7
		2.3.2	Counting Basic Operations	7

		2.3.3	Exact vs. Asymptotic Analysis	7
	2.4	Overvi	ew of Complexity Classes	7
		2.4.1	P, NP, NP-Complete, and NP-Hard (Brief Introduction)	7
		2.4.2	Why We Focus on Polynomial-Time Algorithms	7
3	Asy	mptot	tic Notation	8
	3.1	The Ne	eed for Asymptotic Analysis	9
		3.1.1	Why Exact Counts Are Often Impractical	9
		3.1.2	Growth Rates and Scalability	9
	3.2	Big-O	Notation (O)	9
		3.2.1	Formal Definition	9
		3.2.2	Intuition: Upper Bounds	9
		3.2.3	Common Functions and Their Growth Rates	9
		3.2.4	Examples and Non-Examples	9
		3.2.5	Properties of Big-O	9
	3.3	Big-Or	mega Notation ( $\Omega$ )	9
		3.3.1	Formal Definition	9
		3.3.2	Intuition: Lower Bounds	9
		3.3.3	Examples and Applications	9
		3.3.4	Relationship Between $O$ and $\Omega$	9
	3.4	Big-Th	eta Notation ( $\Theta$ )	9
		3.4.1	Formal Definition	9
		3.4.2	Intuition: Tight Bounds	9
		3.4.3	When to Use $\Theta$ vs. $O$	9
		3.4.4	Examples of Tight Bounds	9
	3.5	Little-o	and Little-omega Notation $(o,\omega)$	9
		3.5.1	Formal Definitions	9
		3.5.2	Strict Asymptotic Bounds	9
		3.5.3	Applications in Analysis	9
	3.6	Comm	on Misconceptions and Pitfalls	9
		3.6.1	Confusing $O$ with $\Theta$	9
		3.6.2	Ignoring Constants in Practice	9
		3.6.3	Misapplying Asymptotic Notation to Small Inputs	9
	3.7	Compa	aring Functions	9

		3.7.1	L'Hôpital's Rule for Limits	9
		3.7.2	Logarithmic vs. Polynomial vs. Exponential Growth	9
		3.7.3	Hierarchy of Common Complexity Classes	9
	3.8	Exerci	ses	9
4	Red	urren	ce Relations and Their Solutions	10
	4.1	Introdu	uction to Recurrence Relations	11
		4.1.1	What Are Recurrences?	11
		4.1.2	Why They Arise in Algorithm Analysis	11
		4.1.3	Examples from Divide-and-Conquer Algorithms	11
	4.2	The Su	ubstitution Method	11
		4.2.1	Guessing the Solution	11
		4.2.2	Proving by Induction	11
		4.2.3	Examples: Mergesort, Binary Search	11
		4.2.4	Strengthening the Inductive Hypothesis	11
	4.3	The Re	ecursion-Tree Method	11
		4.3.1	Visualizing the Recurrence	11
		4.3.2	Summing Over Levels	11
		4.3.3	Examples and Illustrations	11
		4.3.4	Limitations and When to Use	11
	4.4	The M	aster Theorem	11
		4.4.1	Statement of the Master Theorem (Standard Form)	11
		4.4.2	Three Cases and Their Intuition	11
		4.4.3	Proof Sketch (Via Recursion Trees)	11
		4.4.4	Examples: $T(n) = aT(n/b) + f(n) \dots \dots \dots \dots \dots$	11
		4.4.5	Regularity Condition and Edge Cases	11
		4.4.6	Extended Master Theorem (Akra-Bazzi)	11
	4.5	The Al	kra-Bazzi Method	11
		4.5.1	Motivation: Unequal Subproblem Sizes	11
		4.5.2	Statement and Conditions	11
		4.5.3	Examples and Applications	11
		4.5.4	Proof Overview (Advanced)	11
	4.6	Linear	Recurrences with Constant Coefficients	11
		4.6.1	Homogeneous Linear Recurrences	11

		4.6.2	Characteristic Equations	11
		4.6.3	Solving Fibonacci-Type Recurrences	11
		4.6.4	Non-Homogeneous Recurrences and Particular Solutions	11
	4.7	Genera	ating Functions	11
		4.7.1	Introduction to Generating Functions	11
		4.7.2	Solving Recurrences with Generating Functions	11
		4.7.3	Examples: Catalan Numbers, Stirling Numbers	11
	4.8	Advand	ced Topics	11
		4.8.1	Full History Recurrences	11
		4.8.2	Recurrences with Variable Coefficients	11
		4.8.3	Probabilistic Recurrences (Preview)	11
	4.9	Exercis	ses	11
5	Bes	t-Case	e, Worst-Case, and Average-Case Analysis	12
	5.1	Definin	ng Input Classes	13
		5.1.1	What Constitutes an "Input"?	13
		5.1.2	Problem Instances and Instance Distributions	13
	5.2	Best-C	ase Analysis	13
		5.2.1	Definition and Purpose	13
		5.2.2	Examples: Insertion Sort, Linear Search	13
		5.2.3	When Best-Case Matters (and When It Doesn't)	13
	5.3	Worst-	Case Analysis	13
		5.3.1	Definition and Motivation	13
		5.3.2	Guarantees and Robustness	13
		5.3.3	Examples: Quicksort, Searching in Unsorted Arrays	13
		5.3.4	Lower Bounds and Optimality	13
	5.4	Averag	je-Case Analysis	13
		5.4.1	Definition: Expected Running Time	13
		5.4.2	Assumptions About Input Distributions	13
		5.4.3	Probabilistic Models: Uniform, Gaussian, etc	13
		5.4.4	Examples: Quicksort, Hashing, Skip Lists	13
	5.5	Probab	pilistic Analysis vs. Randomized Algorithms	13
		5.5.1	Distinction Between the Two Concepts	13
		5.5.2	Randomized Quicksort: Expected $O(n \log n)$	13

		5.5.3	Las Vegas vs. Monte Carlo Algorithms	13
	5.6	Smoot	hed Analysis	13
		5.6.1	Motivation: Beyond Worst-Case Pessimism	13
		5.6.2	Introduction to Smoothed Analysis	13
		5.6.3	Case Study: Simplex Algorithm	13
	5.7	Exercis	ses	13
6	Pro	babilis	stic Analysis of Algorithms	14
	6.1	Founda	ations of Probabilistic Analysis	15
		6.1.1	Random Variables in Algorithm Analysis	15
		6.1.2	Indicator Random Variables	15
		6.1.3	Linearity of Expectation	15
	6.2	Expect	ted Running Time	15
		6.2.1	Formal Definition	15
		6.2.2	Computing Expectations via Indicator Variables	15
		6.2.3	Examples: Hiring Problem, Randomized Quicksort	15
	6.3	Probab	pilistic Bounds	15
		6.3.1	Markov's Inequality	15
		6.3.2	Chebyshev's Inequality	15
		6.3.3	Chernoff Bounds	15
		6.3.4	Applications to Load Balancing and Hashing	15
	6.4	Rando	mized Algorithms	15
		6.4.1	Randomized Quicksort (Detailed Analysis)	15
		6.4.2	Randomized Selection (Quickselect)	15
		6.4.3	Hashing and Universal Hash Functions	15
		6.4.4	Bloom Filters and Probabilistic Data Structures	15
	6.5	Analys	is of Randomized Data Structures	15
		6.5.1	Skip Lists	15
		6.5.2	Treaps	15
		6.5.3	Hash Tables with Chaining and Open Addressing	15
	6.6	High-P	Probability Results	15
		6.6.1	What Does "With High Probability" Mean?	15
		6.6.2	Concentration Inequalities	15
		6.6.3	Union Bound and Probabilistic Method	15

6.7	Exerci	ses	5
A	dvan	ced Analysis Techniques 1	6
Am	ortize	d Analysis	7
7.1	Introdu	uction to Amortized Analysis	8
	7.1.1	Motivation: Why Average Per-Operation Cost?	8
	7.1.2	Amortized vs. Average-Case Analysis	8
	7.1.3	When to Use Amortized Analysis	8
7.2	Aggre	gate Analysis	8
	7.2.1	Definition and Methodology	8
	7.2.2	Example: Dynamic Array (Vector) Resizing	8
	7.2.3	Example: Binary Counter Increment	8
	7.2.4	Example: Stack with Multipop	8
7.3	The A	ccounting Method	8
	7.3.1	Conceptual Framework: Credits and Debits	8
	7.3.2	Defining Amortized Costs	8
	7.3.3	Example: Dynamic Array via Accounting	8
	7.3.4	Example: Splay Trees (Introduction)	8
	7.3.5	Ensuring Non-Negative Credit Balance	8
7.4	The Po	otential Method	8
	7.4.1	Potential Functions: Definition and Intuition	8
	7.4.2	Relating Amortized Cost to Actual Cost	8
	7.4.3	Designing Good Potential Functions	8
	7.4.4	Example: Dynamic Array via Potential Method	8
	7.4.5	Example: Binary Counter via Potential Method	8
	7.4.6	Example: Fibonacci Heaps (Overview)	8
7.5	Compa	aring the Three Methods	8
	7.5.1	Strengths and Weaknesses	8
	7.5.2	When to Choose Which Method	8
	7.5.3	Equivalence of Methods (Informal Discussion)	8
7.6	Advan	ced Applications	8
	7.6.1	Splay Trees: Full Analysis	8
	7.6.2	Fibonacci Heaps	8
	7.6.3	Disjoint-Set Union (Union-Find)	8
	7.1 7.2 7.3	Advan  Amortize  7.1 Introdu 7.1.1 7.1.2 7.1.3  7.2 Aggree 7.2.1 7.2.2 7.2.3 7.2.4  7.3 The Ad 7.3.1 7.3.2 7.3.3 7.3.4 7.3.5  7.4 The Po 7.4.1 7.4.2 7.4.3 7.4.4 7.4.5 7.4.6  7.5 Compa 7.5.1 7.5.2 7.5.3  7.6 Advan 7.6.1 7.6.2	Advanced Analysis Techniques  Amortized Analysis  7.1 Introduction to Amortized Analysis  7.1.1 Motivation: Why Average Per-Operation Cost?  7.1.2 Amortized vs. Average-Case Analysis  7.1.3 When to Use Amortized Analysis  7.2 Aggregate Analysis  7.2.1 Definition and Methodology  7.2.2 Example: Dynamic Array (Vector) Resizing  7.2.3 Example: Binary Counter Increment  7.2.4 Example: Stack with Multipop  7.3 The Accounting Method  7.3.1 Conceptual Framework: Credits and Debits  7.3.2 Defining Amortized Costs  7.3.3 Example: Splay Trees (Introduction)  7.3.4 Example: Splay Trees (Introduction)  7.3.5 Ensuring Non-Negative Credit Balance  7.4 The Potential Method  7.4.1 Potential Functions: Definition and Intuition  7.4.2 Relating Amortized Cost to Actual Cost  7.4.3 Designing Good Potential Functions  7.4.4 Example: Dynamic Array via Potential Method  7.4.5 Example: Binary Counter via Potential Method  7.4.6 Example: Fibonacci Heaps (Overview)  7.5.1 Strengths and Weaknesses  7.5.2 When to Choose Which Method  7.5.3 Equivalence of Methods (Informal Discussion)  7.6 Advanced Applications  7.6.1 Splay Trees: Full Analysis  7.6.2 Fibonacci Heaps

	7.7	Exercis	ses	18
8	Spa	ce Co	mplexity Analysis	19
	8.1	Introdu	uction to Space Complexity	20
		8.1.1	Why Space Matters	20
		8.1.2	Types of Memory: Stack, Heap, Static	20
		8.1.3	In-Place vs. Out-of-Place Algorithms	20
	8.2	Measu	ring Space Usage	20
		8.2.1	Auxiliary Space vs. Total Space	20
		8.2.2	Recursive Call Stack Depth	20
		8.2.3	Implicit vs. Explicit Data Structures	20
	8.3	Examp	oles of Space Complexity Analysis	20
		8.3.1	Iterative Algorithms: Loops and Arrays	20
		8.3.2	Recursive Algorithms: Mergesort, Quicksort	20
		8.3.3	Dynamic Programming: Memoization vs. Tabulation	20
		8.3.4	Graph Algorithms: BFS, DFS, Shortest Paths	20
	8.4	Space	-Time Tradeoffs	20
		8.4.1	Caching and Memoization	20
		8.4.2	Lookup Tables and Precomputation	20
		8.4.3	Compression and Succinct Data Structures	20
	8.5	Stream	ning and Online Algorithms	20
		8.5.1	Sublinear Space Algorithms	20
		8.5.2	Sketching and Sampling Techniques	20
		8.5.3	Examples: Distinct Elements, Heavy Hitters	20
	8.6	Space	Complexity Classes	20
		8.6.1	L, NL, PSPACE (Brief Overview)	20
		8.6.2	Savitch's Theorem	20
	8.7	Exercis	ses	20
9	Cac	he-Aw	vare and I/O Complexity	21
	9.1		uction to the Memory Hierarchy	22
		9.1.1	Registers, Cache (L1, L2, L3), RAM, Disk	22
		9.1.2	Latency and Bandwidth Characteristics	22
		9.1.3	Why Algorithm Design Must Consider Memory	22
	9.2	The Ex	kternal Memory Model (I/O Model)	22

	9.2.1	Parameters: $N$ (data size), $M$ (memory size), $B$ (block size)	22
	9.2.2	I/O Complexity: Counting Block Transfers	22
	9.2.3	Comparison with RAM Model	22
9.3	3 I/O-Ef	ficient Algorithms	22
	9.3.1	Scanning and Sorting	22
	9.3.2	Matrix Operations	22
	9.3.3	Graph Algorithms	22
9.4	Cache	e-Oblivious Algorithms	22
	9.4.1	Motivation: Optimal Without Knowing $M$ and $B$	22
	9.4.2	Cache-Oblivious Sorting (Funnelsort)	22
	9.4.3	Cache-Oblivious Matrix Multiplication	22
	9.4.4	Cache-Oblivious B-Trees (van Emde Boas Layout)	22
9.5	Cache	-Aware Analysis	22
	9.5.1	Modeling Cache Behavior	22
	9.5.2	Locality of Reference: Temporal and Spatial	22
	9.5.3	Blocking and Tiling Techniques	22
9.6	Real-V	Norld Considerations	22
	9.6.1	Multi-Level Caches	22
	9.6.2	Cache Replacement Policies (LRU, LFU, etc.)	22
	9.6.3	Prefetching and Speculative Execution	22
	9.6.4	False Sharing and Cache Line Effects	22
9.7	Case	Studies	22
	9.7.1	Database Query Processing	22
	9.7.2	External Memory Sorting in Practice	22
	9.7.3	Scientific Computing and Large-Scale Simulations	22
9.8	B Exerci	ses	22
10 Ca	ache-Av	vare Scheduling and Analysis for Multicores	23
10	.1 Introdu	uction to Multicore and Parallel Computing	24
	10.1.1	Shared vs. Distributed Memory	24
	10.1.2	Parallel Models: PRAM, Fork-Join, Work-Stealing	24
	10.1.3	Performance Metrics: Work, Span, Parallelism	24
10	.2 Cache	Coherence and Consistency	24
	10.2.1	MESI and MOESI Protocols	24

		10.2.2	False Sharing in Multicore Systems	24
		10.2.3	Impact on Algorithm Design	24
	10.3	Cache-	Aware Parallel Algorithms	24
		10.3.1	Parallel Sorting with Cache Awareness	24
		10.3.2	Parallel Matrix Multiplication (Strassen, Coppersmith-Winograd)	24
		10.3.3	Load Balancing and Task Granularity	24
	10.4	Real-Ti	ime and Embedded Systems	24
		10.4.1	WCET Analysis in Cache-Aware Contexts	24
		10.4.2	Predictability vs. Average-Case Performance	24
		10.4.3	Cache Partitioning and Locking	24
	10.5	Schedu	uling Strategies	24
		10.5.1	Static vs. Dynamic Scheduling	24
		10.5.2	Work-Stealing Algorithms	24
		10.5.3	Affinity Scheduling for Cache Locality	24
	10.6	Analysi	is Techniques	24
		10.6.1	DAG-Based Analysis (Cilk Model)	24
		10.6.2	Brent's Theorem and Greedy Scheduling	24
		10.6.3	Cache Miss Analysis in Parallel Programs	24
	10.7	Case S	Studies from Research	24
		10.7.1	ECRTS 2007: Cache-Aware Real-Time Scheduling	24
		10.7.2	Cache-Aware Scheduling for Multicores (Embedded Systems)	24
		10.7.3	VLDB 2019: Concurrent Hash Tables and Cache Performance	24
	10.8	Exercis	ses	24
V	1	ower	Bounds and Optimality	25
•				
1			unds for Comparison-Based Algorithms	
	11.1		on Trees	27
		11.1.1	Modeling Algorithms as Decision Trees	27
		11.1.2	Height of Decision Trees and Worst-Case Complexity	27
	11.2	Sorting	Lower Bound	27
		11.2.1	Information-Theoretic Argument	27
		11.2.2	$\Omega(n \log n)$ Lower Bound for Comparison Sorting	27
		11.2.3	Implications and Optimal Algorithms	27
	11.3	Selection	on and Searching Lower Bounds	27

		11.3.1	Finding the Minimum: $\Omega(n)$	27
		11.3.2	Finding Median: Adversary Arguments	27
		11.3.3	Searching in Sorted Arrays: $\Omega(\log n)$	27
	11.4	Advers	ary Arguments	27
		11.4.1	General Framework	27
		11.4.2	Examples: Merging, Element Uniqueness	27
	11.5	Exercis	ses	27
12	Alge	ebraic	and Non-Comparison Lower Bounds	28
	12.1	Algebra	aic Decision Trees	28
		12.1.1	Extending Beyond Comparisons	28
		12.1.2	Element Distinctness Lower Bound	28
	12.2	Comm	unication Complexity	28
		12.2.1	Models and Definitions	28
		12.2.2	Applications to Data Structures	28
	12.3	Cell-Pr	obe Model	28
		12.3.1	Lower Bounds for Data Structures	28
		12.3.2	Dynamic vs. Static Data Structures	28
	12.4	Exercis	ses	28
V	Sr	ecia	lized Topics and Applications	29
13	Ana		• • • • • • • • • • • • • • • • • • • •	
. •		lvsis (	of Specific Algorithm Paradigms	30
		Divide-	and-Conquer Algorithms	31
		Divide- 13.1.1	and-Conquer Algorithms	<b>31</b>
		Divide-	and-Conquer Algorithms	31
	13.1	Divide- 13.1.1 13.1.2 13.1.3	and-Conquer Algorithms	31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3	and-Conquer Algorithms	31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy	and-Conquer Algorithms	31 31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy 13.2.1	and-Conquer Algorithms	31 31 31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy 13.2.1 13.2.2 13.2.3	and-Conquer Algorithms	31 31 31 31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy 13.2.1 13.2.2 13.2.3	and-Conquer Algorithms	31 31 31 31 31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy 13.2.1 13.2.2 13.2.3 Dynam	and-Conquer Algorithms  General Framework and Recurrence Relations  Examples: Mergesort, Quicksort, Strassen's Algorithm  Optimality and Lower Bounds  Algorithms  Correctness via Exchange Arguments  Matroid Theory (Brief Introduction)  Examples: Huffman Coding, Kruskal's MST	31 31 31 31 31 31 31
	13.1	Divide- 13.1.1 13.1.2 13.1.3 Greedy 13.2.1 13.2.2 13.2.3 Dynam 13.3.1	and-Conquer Algorithms  General Framework and Recurrence Relations  Examples: Mergesort, Quicksort, Strassen's Algorithm  Optimality and Lower Bounds  Algorithms  Correctness via Exchange Arguments  Matroid Theory (Brief Introduction)  Examples: Huffman Coding, Kruskal's MST  ic Programming  Optimal Substructure and Overlapping Subproblems	31 31 31 31 31 31 31 31

	13.4	Backtra	acking and Branch-and-Bound	31
		13.4.1	Pruning the Search Space	31
		13.4.2	Worst-Case Exponential, Average-Case Better	31
		13.4.3	Examples: N-Queens, Traveling Salesman	31
	13.5	Exercis	ses	31
14	Onli	ine Alç	gorithms and Competitive Analysis	32
	14.1	Introdu	ction to Online Algorithms	32
		14.1.1	Online vs. Offline Problems	32
		14.1.2	Competitive Ratio	32
	14.2	Examp	oles of Online Problems	32
		14.2.1	Paging and Caching (LRU, FIFO, LFU)	32
		14.2.2	Load Balancing	32
		14.2.3	Online Scheduling	32
	14.3	Compe	etitive Analysis Techniques	32
		14.3.1	Deterministic vs. Randomized Algorithms	32
		14.3.2	Lower Bounds via Adversary Arguments	32
	14.4	Exercis	ses	32
15	App	roxim	ation Algorithms	33
	15.1	Introdu	ection to Approximation	33
		15.1.1	NP-Hardness and Intractability	33
		15.1.2	Approximation Ratios	33
	15.2	Examp	oles of Approximation Algorithms	33
		15.2.1	Vertex Cover (2-Approximation)	33
		15.2.2	Set Cover (Greedy, $\log n$ -Approximation)	33
		15.2.3	Traveling Salesman Problem (Metric TSP)	33
	15.3	Analys	is Techniques	33
		15.3.1	Bounding Optimal Solutions	33
		15.3.2	Linear Programming Relaxations	33
	15.4	Exercis	ses	33
16	Para	amete	rized Complexity	34
			oction to Parameterized Algorithms	34
		16.1.1	Fixed-Parameter Tractability (FPT)	34
		16.1.2	Kernelization	34

	16.2	Examp	les and Analysis	34
		16.2.1	Vertex Cover Parameterized by Solution Size	34
		16.2.2	Treewidth and Graph Algorithms	34
	16.3	W-Hier	archy and Hardness	34
		16.3.1	W[1], W[2], and Beyond	34
	16.4	Exercis	ses	34
VI	Р	ractio	cal Considerations and Case Studies	35
17	Fron	n The	ory to Practice	36
				36
		17.1.1	When $O(n \log n)$ Beats $O(n)$ in Practice	36
		17.1.2	Empirical Performance Measurements	36
	17.2	Algorith	nm Engineering	36
		17.2.1	Profiling and Benchmarking	36
		17.2.2	Tuning for Specific Hardware	36
		17.2.3	Libraries and Implementations (STL, Boost, etc.)	36
	17.3	Paralle	I and Distributed Algorithm Analysis	36
		17.3.1	Scalability and Speedup	36
		17.3.2	Amdahl's Law and Gustafson's Law	36
	17.4	Energy	Efficiency	36
		17.4.1	Energy as a Resource	36
		17.4.2	Green Computing and Mobile Devices	36
	17.5	Exercis	ses	36
18	Cas	e Stuc	lies	37
	18.1	Sorting	Algorithms in Practice	37
		18.1.1	Timsort, Introsort, Radix Sort	37
		18.1.2	Comparison of Theoretical vs. Empirical Performance	37
	18.2	Graph	Algorithms in Large-Scale Systems	37
		18.2.1	Web Graphs and PageRank	37
		18.2.2	Social Network Analysis	37
	18.3	Machin	e Learning and Data Science	37
		18.3.1	Complexity of Training Algorithms	37
		18.3.2	SGD, AdaGrad, Adam: Time and Space Analysis	37

	18.4	Database Systems	37
		18.4.1 Query Optimization	37
		18.4.2 Indexing Structures (B-Trees, LSM-Trees)	37
	18.5	Exercises	37
Α	Mat	hematical Background	38
	A.1	Summation Formulas	38
	A.2	Logarithms and Exponentials	38
	A.3	Recurrence Relations (Quick Reference)	38
	A.4	Probability Distributions	38
	A.5	Matrix Operations	38
В	Pse	udocode Conventions	39
	B.1	Notation and Style	39
	B.2	Common Data Structures	39
С	Solu	utions to Selected Exercises	40
D	Glos	ssary of Terms	41
Ε	Inde	ex of Algorithms	42
F	Ann	otated Bibliography	43
	F.1	Foundational Texts	43
	F.2	Research Papers by Topic	43
	F3	Online Courses and Resources	43

## **Preface**

 $E^{\scriptscriptstyle ext{VERY RIGOROUS JOURNEY begins with a question.}}$  For this book, that question was deceptively simple: How do we truly measure the cost of computation?

## **Genesis of This Work**

During my studies in computer science, I encountered a persistent frustration: algorithmic analysis was presented fragmentedly across courses and textbooks. Asymptotic notation appeared in one context, recurrence relations in another, amortized analysis as an advanced topic buried in data structures courses. Each technique existed in isolation, its connection to broader analytical frameworks obscured.

I wanted something different—a unified treatment that explains not just *how* to analyze algorithms, but *why* these particular analytical methods emerged, *how* they relate to one another, and *when* each provides the deepest insight. Unable to find such a resource, I decided to create it.

This book represents that effort: a comprehensive synthesis of algorithmic analysis techniques, built from extensive research across the literature of computer science, applied mathematics, and complexity theory.

## **A Living Document**

This work is fundamentally different from traditional textbooks in one crucial respect: it is alive and evolving.

As I continue researching algorithmic analysis—discovering new connections, understanding techniques more deeply, encountering novel applications—this book grows and improves. Each week brings refinements: clearer explanations, additional examples, connections I hadn't previously recognized, corrections to subtle errors. The book you're reading today is more complete than the version that existed last month. The version that will exist next month will be more refined than what you see now. This living nature means:

• **Continuous Improvement**: Sections evolve as my understanding deepens through ongoing research

- **Integration of New Research**: Recent developments in algorithmic analysis are incorporated as they emerge
- Community Feedback: Reader corrections, suggestions, and insights strengthen the work
- Transparency About Limitations: I openly acknowledge where current understanding is incomplete

Traditional textbooks freeze knowledge at publication time. This book remains fluid, growing alongside both my research and the field itself.

## Foundation on Giants' Shoulders

While this book represents my synthesis and presentation, the knowledge it contains stands on the foundational work of brilliant computer scientists and mathematicians:

## What Makes This Book Distinctive

Several characteristics distinguish this treatment from existing resources:

**Analysis-First Perspective** Most algorithms texts treat analysis as a supporting tool for understanding algorithms. This book inverts that relationship: analysis techniques are the primary focus, with algorithms serving as examples to illustrate analytical methods.

**Comprehensive Coverage** From fundamental asymptotic notation to research-level topics like cache-oblivious algorithms and parameterized complexity, the book spans the full spectrum of analytical techniques.

**Unified Framework** Rather than presenting techniques in isolation, the book constantly highlights connections—how methods relate, when one technique is preferred over another, why certain problems require specific analytical approaches.

**Progressive Development** Concepts build systematically. Each chapter assumes only material from preceding chapters, allowing readers to develop understanding incrementally without gaps.

**Mathematical Rigor with Intuition** Every formal development begins with intuitive motivation. Proofs serve understanding, revealing *why* results hold, not merely verifying *that* they hold.

**Living Evolution** Perhaps most importantly: this book acknowledges its own incompleteness and commits to continuous improvement through ongoing research and community feedback.

## Who Should Read This Book

This book serves multiple audiences:

**Undergraduate students** who have completed introductory algorithms and want deeper analytical understanding. You should be comfortable with basic programming, discrete mathematics, and elementary proofs.

**Graduate students** needing advanced analysis techniques for research. This book provides the analytical toolkit for reading algorithms research and analyzing novel algorithms.

**Practitioners** seeking principled frameworks for algorithm selection and performance prediction. The analytical perspective here complements practical engineering experience.

**Self-learners** with intellectual curiosity about algorithmic efficiency. If you've wondered *why* certain algorithms are considered efficient, this book provides rigorous answers.

Whatever your background, you should bring mathematical maturity—comfort with abstraction, formal definitions, and logical reasoning. Specific prerequisites (discrete mathematics, probability, calculus) are reviewed in Part I, but prior exposure helps.

## **Structure Overview**

The book organizes into six major parts:

- 1. **Foundations** Establishing context, prerequisites, and reading strategies
- 2. **Foundations of Algorithmic Analysis** Core techniques: asymptotic notation, recurrences, best/worst/average case, probabilistic analysis
- 3. **Advanced Analysis Techniques** Amortized analysis, space complexity, memory hierarchy effects, parallel analysis
- 4. **Lower Bounds and Optimality** Understanding fundamental limits on algorithmic efficiency

- 5. **Specialized Topics and Applications** Analysis techniques for specific algorithm paradigms
- 6. **Practical Considerations and Case Studies** Bridging theory to real-world performance

Each part builds on preceding material, developing increasingly sophisticated analytical capabilities.

## A Note on Rigor

This book takes mathematical rigor seriously—not as pedantic formalism, but as the discipline enabling precise reasoning about complex systems.

Informal intuition is valuable but insufficient. Rigorous analysis distinguishes what intuition conflates: logarithmic from linear growth, amortized from average cost, theoretical complexity from practical performance. Mathematical precision is not obstacle but tool—enabling reliable reasoning, clear communication, and principled decision-making.

That said, rigor serves understanding. Every formal development begins with intuition. Proofs reveal insights, not just verify results. If formalism feels overwhelming initially, prioritize key ideas on first reading, returning later for proof details.

## **Final Thoughts**

Algorithmic analysis is often presented as necessary prerequisite—technical machinery required before "real" algorithms work begins. This perspective misses something fundamental.

Analysis is not merely evaluation. It is a framework for *thinking* about computation—revealing patterns in costs, exposing hidden problem structure, developing intuition about what solutions might be possible.

Mastering these techniques changes how you approach problems. You develop analytical lenses that transform how you perceive computational challenges. This cognitive shift is the ultimate goal.

The journey ahead is demanding. You will encounter abstract mathematics, work through detailed proofs, solve challenging exercises. But the reward—deep, rigorous understanding of computational cost—justifies the effort.

And remember: this book is alive. As research continues and understanding deepens, the work improves. Your engagement—through questions, corrections, and insights—contributes to that improvement.

Welcome to The Art of Algorithmic Analysis. Let's begin.

*Mahdi* 2025-2026

## Acknowledgments

I would like to express my gratitude to everyone who supported me during the creation of this book. Special thanks to the open-source community for their invaluable resources and to all those who reviewed early drafts and provided feedback.

## Part I Foundations

HE FOUNDATION of algorithmic analysis rests upon a bedrock of mathematical rigor and computational intuition. Before embarking on the journey of analyzing algorithms asymptotically, understanding recurrences, or diving into amortized analysis, we must first ensure our mathematical toolkit is complete and our understanding of fundamental concepts is solid.

This part serves as both a review and a deep dive into the essential mathematical and algorithmic prerequisites that underpin everything that follows. Whether you are a student encountering these topics for the first time or a practitioner seeking to refresh your knowledge, this foundation will prepare you for the sophisticated analysis techniques explored in subsequent parts.

"In mathematics, you don't understand things. You just get used to them."

— JOHN VON NEUMANN

First Edition • 2025

## Mathematical and Algorithmic Prerequisites

HIS CHAPTER establishes the mathematical foundation required for rigorous algorithm analysis. We cover discrete mathematics, probability theory, mathematical analysis, linear algebra, and number theory—each selected for its direct relevance to algorithmic reasoning.

## Why This Chapter Matters:

- **Discrete Mathematics** provides the language of algorithms: sets, functions, graphs, and proof techniques
- Probability Theory enables average-case and randomized algorithm analysis
- Mathematical Analysis gives us tools for asymptotic reasoning and limits
- Linear Algebra underpins graph algorithms, Markov chains, and numerical methods
- Number Theory appears in cryptography, hashing, and randomization

**How to Use This Chapter:** If you're already comfortable with a topic, skim it for notation and conventions. Each section includes self-assessment problems to verify your understanding.

## 1.1 Discrete Mathematics: The Language of Algorithms

Discrete mathematics forms the grammatical structure of algorithm design. Unlike continuous mathematics, we work with countable, distinct objects: integers, graphs, logical propositions. This section covers the foundational concepts that appear repeatedly throughout algorithm analysis.

First Edition • 2025

## 1.1.1 Sets, Functions, and Relations

**Set Theory Basics** 

**Operations on Sets** 

Functions: Injective, Surjective, Bijective

**Relations and Equivalence Classes** 

**Order Relations and Partial Orders** 

## 1.1.2 Combinatorics: Permutations, Combinations, and Binomial Coefficients

The Fundamental Counting Principles

Permutations with and without Repetition

Combinations and the Binomial Theorem

Pigeonhole Principle and Applications

**Inclusion-Exclusion Principle** 

Combinatorial Identities Useful in Analysis

## 1.1.3 Graph Theory Basics

**Graph Definitions and Representations** 

Paths, Cycles, and Connectivity

**Trees and Their Properties** 

Directed Graphs and DAGs

Graph Algorithms Preview (BFS, DFS)

## 1.1.4 Proof Techniques: Induction, Contradiction, and Contrapositive

**Direct Proof Structure** 

**Proof by Contradiction** 

**Proof by Contrapositive** 

Mathematical Induction: Weak Form

Strong Induction and Well-Ordering Principle

Structural Induction for Recursive Definitions

Common Proof Pitfalls and How to Avoid Them

## 1.2 Elementary Probability Theory

## Part II

Foundations of Algorithmic Analysis

## **Introduction to Algorithm Analysis**

2.1	What Is Algorithm Analysis?		
2.1.1	Correctness vs. Efficiency		
2.1.2	Resource Measures: Time, Space, Energy, I/O		
2.1.3	The Need for Mathematical Models		
2.2	The RAM Model of Computation		
2.2.1	Basic Operations and Unit-Cost Assumption		
2.2.2	Memory Access Model		
2.2.3	Limitations and Extensions of the RAM Model		
2.3	Measuring Algorithm Performance		
2.3.1	Input Size and Problem Instances		
2.3.2	Counting Basic Operations		
2.3.3	Exact vs. Asymptotic Analysis		
2.4	Overview of Complexity Classes		
2.4.1	P. NP. NP-Complete, and NP-Hard (Brief Introduct		

2.4.2 Why We Focus on Polynomial-Time Algorithms

## **Asymptotic Notation**

3.1	The Need	for Asym	ptotic Analysis

- 3.1.1 Why Exact Counts Are Often Impractical
- 3.1.2 Growth Rates and Scalability
- 3.2 Big-O Notation (O)
- 3.2.1 Formal Definition
- 3.2.2 Intuition: Upper Bounds
- 3.2.3 Common Functions and Their Growth Rates
- 3.2.4 Examples and Non-Examples
- 3.2.5 Properties of Big-O

**Transitivity** 

Addition and Multiplication Rules

Reflexivity and Asymmetry

## 3.3 Big-Omega Notation ( $\Omega$ )

- 3.3.1 Formal Definition
- 3.3.2 Intuition: Lower Bounds
- 3.3.3 Examples and Applications
- 3.3.4 Relationship Between O and  $\Omega$

## 3.4 Big-Theta Notation (⊕)

## **Recurrence Relations and Their Solutions**

4.1	Introduction	to Recurrence	Relations
<b>T.</b> T	IIIIIUuucuuli	to recurrence	inclauons

- 4.1.1 What Are Recurrences?
- 4.1.2 Why They Arise in Algorithm Analysis
- 4.1.3 Examples from Divide-and-Conquer Algorithms

## 4.2 The Substitution Method

- 4.2.1 Guessing the Solution
- 4.2.2 Proving by Induction
- 4.2.3 Examples: Mergesort, Binary Search
- 4.2.4 Strengthening the Inductive Hypothesis

## 4.3 The Recursion-Tree Method

- 4.3.1 Visualizing the Recurrence
- 4.3.2 Summing Over Levels
- 4.3.3 Examples and Illustrations
- 4.3.4 Limitations and When to Use

## 4.4 The Master Theorem

4.4.2

4.4.1 Statement of the Master Theorem (Standard Form)

Three Cases and Their Intuition

## Best-Case, Worst-Case, and Average-Case Analysis

5.1 Defining Input Classes
----------------------------

- 5.1.1 What Constitutes an "Input"?
- 5.1.2 Problem Instances and Instance Distributions
- 5.2 Best-Case Analysis
- 5.2.1 Definition and Purpose
- 5.2.2 Examples: Insertion Sort, Linear Search
- 5.2.3 When Best-Case Matters (and When It Doesn't)
- 5.3 Worst-Case Analysis
- 5.3.1 Definition and Motivation
- 5.3.2 Guarantees and Robustness
- 5.3.3 Examples: Quicksort, Searching in Unsorted Arrays
- 5.3.4 Lower Bounds and Optimality
- 5.4 Average-Case Analysis
- 5.4.1 Definition: Expected Running Time
- 5.4.2 Assumptions About Input Distributions
- 5.4.3 Probabilistic Models: Uniform, Gaussian, etc.
- 5.4.4 Examples: Quicksort, Hashing, Skip Lists

#### **Probabilistic Analysis of Algorithms**

6.1	Foundations of Probabilistic Analysis
6.1.1	Random Variables in Algorithm Analysis
6.1.2	Indicator Random Variables
6.1.3	Linearity of Expectation
6.2	<b>Expected Running Time</b>
6.2.1	Formal Definition
6.2.2	Computing Expectations via Indicator Variables
6.2.3	Examples: Hiring Problem, Randomized Quicksort
6.3	Probabilistic Bounds
	Probabilistic Bounds  Markov's Inequality
6.3.1	
6.3.1 6.3.2	Markov's Inequality
6.3.1 6.3.2 6.3.3	Markov's Inequality Chebyshev's Inequality
6.3.1 6.3.2 6.3.3 6.3.4	Markov's Inequality Chebyshev's Inequality Chernoff Bounds
6.3.1 6.3.2 6.3.3 6.3.4 6.4	Markov's Inequality Chebyshev's Inequality Chernoff Bounds Applications to Load Balancing and Hashing

6.4.2 Randomized Selection (Quickselect)

6:4:3 · 202 Hashing and Universal Hash Functions

**Bloom Filters and Probabilistic Data Structures** 

# Part III Advanced Analysis Techniques

#### **Amortized Analysis**

7.1	Introduction	to An	nortized	Analy	/sis

- 7.1.1 Motivation: Why Average Per-Operation Cost?
- 7.1.2 Amortized vs. Average-Case Analysis
- 7.1.3 When to Use Amortized Analysis
- 7.2 Aggregate Analysis
- 7.2.1 Definition and Methodology
- 7.2.2 Example: Dynamic Array (Vector) Resizing
- 7.2.3 Example: Binary Counter Increment
- 7.2.4 Example: Stack with Multipop

#### 7.3 The Accounting Method

- 7.3.1 Conceptual Framework: Credits and Debits
- 7.3.2 Defining Amortized Costs
- 7.3.3 Example: Dynamic Array via Accounting
- 7.3.4 Example: Splay Trees (Introduction)
- 7.3.5 Ensuring Non-Negative Credit Balance

#### 7.4 The Potential Method

**20**|43

#### **Chapter 8**

## **Space Complexity Analysis**

8.1	Introduction to Space Complexity
8.1.1	Why Space Matters
8.1.2	Types of Memory: Stack, Heap, Static
8.1.3	In-Place vs. Out-of-Place Algorithms
8.2	Measuring Space Usage
8.2.1	Auxiliary Space vs. Total Space
8.2.2	Recursive Call Stack Depth
8.2.3	Implicit vs. Explicit Data Structures
8.3	<b>Examples of Space Complexity Analysis</b>
8.3.1	Iterative Algorithms: Loops and Arrays
8.3.2	Recursive Algorithms: Mergesort, Quicksort
8.3.3	Dynamic Programming: Memoization vs. Tabulation
8.3.4	Graph Algorithms: BFS, DFS, Shortest Paths
8.4	Space-Time Tradeoffs
8.4.1	Caching and Memoization
8.4.2	Lookup Tables and Precomputation
<b>8</b> 51 <b>4</b> 01130	202Compression and Succinct Data Structures

**Streaming and Online Algorithms** 

#### Cache-Aware and I/O Complexity

9.1 Introduction to the Memory Hierarchy
--

- 9.1.1 Registers, Cache (L1, L2, L3), RAM, Disk
- 9.1.2 Latency and Bandwidth Characteristics
- 9.1.3 Why Algorithm Design Must Consider Memory
- 9.2 The External Memory Model (I/O Model)
- 9.2.1 Parameters: N (data size), M (memory size), B (block size)
- 9.2.2 I/O Complexity: Counting Block Transfers
- 9.2.3 Comparison with RAM Model

#### 9.3 I/O-Efficient Algorithms

#### 9.3.1 Scanning and Sorting

**External Merge Sort** 

I/O Complexity:  $O((N/B)\log_{M/B}(N/B))$ 

#### 9.3.2 Matrix Operations

**Matrix Transposition** 

**Matrix Multiplication** 

#### 9.3.3 Graph Algorithms

I/O-Efficient BFS and DFS

**Minimum Spanning Tree** 

**22**|43

#### 9.4 Cache-Oblivious Algorithms

# Cache-Aware Scheduling and Analysis for Multicores

10.1	Introduction to Multicore and Parallel Compu	iting
10.1.1	Shared vs. Distributed Memory	
10.1.2	Parallel Models: PRAM, Fork-Join, Work-Stealing	
10.1.3	Performance Metrics: Work, Span, Parallelism	
10.2	Cache Coherence and Consistency	
10.2.1	MESI and MOESI Protocols	
10.2.2	False Sharing in Multicore Systems	
10.2.3	Impact on Algorithm Design	
10.3	Cache-Aware Parallel Algorithms	
10.3.1	Parallel Sorting with Cache Awareness	
10.3.2	Parallel Matrix Multiplication (Strassen, Coppers Winograd)	mith-
10.3.3	Load Balancing and Task Granularity	
10.4	Real-Time and Embedded Systems	
10.4.1	WCET Analysis in Cache-Aware Contexts	
<b>1:0:4:2</b> :0	Predictability vs. Average-Case Performance	<b>24</b>  43

10.4.3 Cache Partitioning and Locking

# Part IV Lower Bounds and Optimality

# Lower Bounds for Comparison-Based Algorithms

444		• •	
11.1	1)6	C1C1AT	ı Trees
T T • T	$\mathcal{L}_{\mathcal{L}}$	CISIUI	1 11663

- 11.1.1 Modeling Algorithms as Decision Trees
- 11.1.2 Height of Decision Trees and Worst-Case Complexity

#### 11.2 Sorting Lower Bound

- 11.2.1 Information-Theoretic Argument
- 11.2.2  $\Omega(n \log n)$  Lower Bound for Comparison Sorting
- 11.2.3 Implications and Optimal Algorithms

#### 11.3 Selection and Searching Lower Bounds

- **11.3.1** Finding the Minimum:  $\Omega(n)$
- 11.3.2 Finding Median: Adversary Arguments
- **11.3.3** Searching in Sorted Arrays:  $\Omega(\log n)$

#### 11.4 Adversary Arguments

- 11.4.1 General Framework
- 11.4.2 Examples: Merging, Element Uniqueness

#### 11.5 Exercises

# Algebraic and Non-Comparison Lower Bounds

12.1	Algebraic Decision Trees
12.1.1	<b>Extending Beyond Comparisons</b>
12.1.2	<b>Element Distinctness Lower Bound</b>
12.2	<b>Communication Complexity</b>
12.2.1	Models and Definitions
12.2.2	Applications to Data Structures
12.3	Cell-Probe Model
12.3.1	Lower Bounds for Data Structures
12.3.2	Dynamic vs. Static Data Structures
12 4	Exercises

### Part V

**Specialized Topics and Applications** 

# **Analysis of Specific Algorithm Paradigms**

13.1	Divide-and-Conquer Algorithms
13.1.1	General Framework and Recurrence Relations
13.1.2	Examples: Mergesort, Quicksort, Strassen's Algorithm
13.1.3	Optimality and Lower Bounds
13.2	Greedy Algorithms
13.2.1	Correctness via Exchange Arguments
13.2.2	Matroid Theory (Brief Introduction)
13.2.3	Examples: Huffman Coding, Kruskal's MST
13.3	Dynamic Programming
13.3.1	Optimal Substructure and Overlapping Subproblems
13.3.2	Memoization vs. Tabulation
13.3.3	Time and Space Complexity Analysis
13.3.4	Examples: Knapsack, Edit Distance, Matrix Chain Multiplication
13.4	Backtracking and Branch-and-Bound

# Online Algorithms and Competitive Analysis

14.1	Introduction to Online Algorithms
14.1.1	Online vs. Offline Problems
14.1.2	Competitive Ratio
14.2	<b>Examples of Online Problems</b>
14.2.1	Paging and Caching (LRU, FIFO, LFU)
14.2.2	Load Balancing
14.2.3	Online Scheduling
14.3	<b>Competitive Analysis Techniques</b>
14.3.1	Deterministic vs. Randomized Algorithms
14.3.2	Lower Bounds via Adversary Arguments
14.4	Exercises

## **Approximation Algorithms**

15.1	Introduction to Approximation
15.1.1	NP-Hardness and Intractability
15.1.2	Approximation Ratios
15.2	<b>Examples of Approximation Algorithms</b>
15.2.1	Vertex Cover (2-Approximation)
15.2.2	<b>Set Cover (Greedy,</b> $\log n$ <b>-Approximation)</b>
15.2.3	Traveling Salesman Problem (Metric TSP)
15.3	Analysis Techniques
15.3.1	<b>Bounding Optimal Solutions</b>
15.3.2	Linear Programming Relaxations
15.4	Exercises

#### **Parameterized Complexity**

16.1	Introduction to Parameterized Algorithms
16.1.1	Fixed-Parameter Tractability (FPT)

- 16.1.2 Kernelization
- 16.2 Examples and Analysis
- 16.2.1 Vertex Cover Parameterized by Solution Size
- 16.2.2 Treewidth and Graph Algorithms
- 16.3 W-Hierarchy and Hardness
- 16.3.1 W[1], W[2], and Beyond
- 16.4 Exercises

#### Part VI

# **Practical Considerations and Case Studies**

## **From Theory to Practice**

17.1	Hidden Constants and Lower-Order Terms
17.1.1	When $O(n \log n)$ Beats $O(n)$ in Practice
17.1.2	<b>Empirical Performance Measurements</b>
17.2	Algorithm Engineering
17.2.1	Profiling and Benchmarking
17.2.2	Tuning for Specific Hardware
17.2.3	Libraries and Implementations (STL, Boost, etc.)
17.3	Parallel and Distributed Algorithm Analysis
	Parallel and Distributed Algorithm Analysis Scalability and Speedup
17.3.1	
17.3.1 17.3.2	Scalability and Speedup
17.3.1 17.3.2 17.4	Scalability and Speedup  Amdahl's Law and Gustafson's Law
17.3.1 17.3.2 17.4 17.4.1	Scalability and Speedup  Amdahl's Law and Gustafson's Law  Energy Efficiency

#### **Case Studies**

18.1	Sorting Algorithms in Practice
18.1.1	Timsort, Introsort, Radix Sort
18.1.2	Comparison of Theoretical vs. Empirical Performance
18.2	<b>Graph Algorithms in Large-Scale Systems</b>
18.2.1	Web Graphs and PageRank
18.2.2	Social Network Analysis
18.3	Machine Learning and Data Science
18.3.1	Complexity of Training Algorithms
18.3.2	SGD, AdaGrad, Adam: Time and Space Analysis
18.4	Database Systems
18.4.1	Query Optimization
18.4.2	Indexing Structures (B-Trees, LSM-Trees)
18.5	Exercises

#### Appendix A

#### **Mathematical Background**

- A.1 Summation Formulas
- A.2 Logarithms and Exponentials
- A.3 Recurrence Relations (Quick Reference)
- A.4 Probability Distributions
- A.5 Matrix Operations

### Appendix B

#### **Pseudocode Conventions**

- **B.1** Notation and Style
- **B.2** Common Data Structures

# Appendix C Solutions to Selected Exercises

# Appendix D Glossary of Terms

# Appendix E Index of Algorithms

### Appendix F

### **Annotated Bibliography**

- F.1 Foundational Texts
- F.2 Research Papers by Topic
- F.3 Online Courses and Resources