

# Data Structure & Algorithms – Cấu Trúc Dữ Liệu & Giải Thuật

Nguyễn Quân Bá Hồng\*

Ngày 6 tháng 7 năm 2025

## Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:

URL: [https://nqbh.github.io/advanced\\_STEM/](https://nqbh.github.io/advanced_STEM/).

Latest version:

- *Data Structure & Algorithms – Cấu Trúc Dữ Liệu & Giải Thuật*.

PDF: URL: [https://github.com/NQBH/advanced\\_STEM\\_beyond/blob/main/data\\_structure\\_algorithm/NQBH\\_data\\_structure\\_algorithm.pdf](https://github.com/NQBH/advanced_STEM_beyond/blob/main/data_structure_algorithm/NQBH_data_structure_algorithm.pdf).

TeX: URL: [https://github.com/NQBH/advanced\\_STEM\\_beyond/blob/main/data\\_structure\\_algorithm/NQBH\\_data\\_structure\\_algorithm.tex](https://github.com/NQBH/advanced_STEM_beyond/blob/main/data_structure_algorithm/NQBH_data_structure_algorithm.tex).

- .

PDF: URL: [.pdf](#).

TeX: URL: [.tex](#).

## Mục lục

<b>1 Basic Data Structure – Cấu Trúc Dữ Liệu Cơ Bản</b>	<b>1</b>
<b>2 Algorithms – Thuật Giải/Thuật Toán</b>	<b>1</b>
2.1 [Cor+22]. Introduction to Algorithms. 4e	1
2.2 [Knu98]. DONALD ERWIN KNUTH. The Art of Computer Programming. Vol. 3: Sorting & Searching. 2e	10
2.3 [Knu11]. DONALD ERWIN KNUTH. The Art of Computer Programming. Vol. 4: Combinatorial Algorithms. Part 1. 1e	15
2.4 [Knu11]. DONALD ERWIN KNUTH. The Art of Computer Programming. Vol. 4: Combinatorial Algorithms. Part 2. 1e	15
<b>3 Wikipedia's</b>	<b>15</b>
<b>4 Miscellaneous</b>	<b>15</b>
<b>Tài liệu</b>	<b>15</b>

## 1 Basic Data Structure – Cấu Trúc Dữ Liệu Cơ Bản

## 2 Algorithms – Thuật Giải/Thuật Toán

### 2.1 [Cor+22]. Introduction to Algorithms. 4e

[630 Amazon ratings][9071 Goodreads ratings]

- Amazon review. “A comprehensive update of the leading algorithms text, with new material on matchings in bipartite graphs, online algorithms, ML, & other topics. Some books on algorithms are rigorous but incomplete; others cover masses of material but lack rigor. *Introduction to Algorithms* uniquely combines rigor & comprehensiveness. It covers a broad range of algorithms in depth, yet makes their design & analysis accessible to all levels of readers, with self-contained chapters & algorithms in pseudocode. Since the publication of 1e, *Introduction to Algorithms* has become the leading algorithms text in universities worldwide as well as the standard reference for professionals. This 4e has been updated throughout. New for 4e:
  - New chapters on matching in bipartite graphs, online algorithms, & ML
  - New material on topics including solving recurrence equations, hash tables, potential functions, & suffix arrays

---

\*A scientist- & creative artist wannabe, a mathematics & computer science lecturer of Department of Artificial Intelligence & Data Science (AIDS), School of Technology (SOT), UMT Trường Đại học Quản lý & Công nghệ TP.HCM, Hồ Chí Minh City, Việt Nam.  
E-mail: [nguyenquanbahong@gmail.com](mailto:nguyenquanbahong@gmail.com) & [hong.nguyenquanba@umt.edu.vn](mailto:hong.nguyenquanba@umt.edu.vn). Website: <https://nqbh.github.io/>. GitHub: <https://github.com/NQBH>.

- 140 new exercises & 22 new problems
- Reader feedback-informed improvements to odd problems
- Clearer, more personal, & gender-neutral writing style
- Color added to improve visual presentation
- Notes, bibliography, & index updated to reflect developments in the field
- Website with new supplementary material

This book is a comprehensive update of the leading algorithms text, covering a broad range of algorithms in depth, yet making their design & analysis accessible to all levels of readers, with self-contained chapters & algorithms in pseudocode.”

- “A data structure is a way to store & organize data in order to facilitate access & modifications.”
- “Machine learning can be thought of as a method for performing algorithmic tasks without explicitly designing an algorithm, but instead inferring patterns from data & thereby automatically learning a solution.”
- “The running time of an algorithm on a particular input is the number of instructions & data accesses executed.”

**About Author.** THOMAS H. CORMEN is Emeritus Professor of Computer Science at Dartmouth College. CHARLES E. LEISERSON is Edwin Sibley Webster Professor in Electrical Engineering & Computer Science at MIT. RONALD L. RIVEST is Institute Professor at MIT. CLIFFORD STEIN is Wai T. Chang Professor of Industrial Engineering & Operations Research, & of Computer Science at Columbia University.

- **Preface.** Not so long ago, anyone who had heard word “algorithm” has almost certainly a computer scientist or mathematician. With computers having become prevalent in our modern lives, however, term is no longer esoteric (bí truyền). If look around home, find algorithms running in most mundane (tầm thường) places: microwave oven, washing machine, &, of course, computer. Ask algorithms to make recommendations to you: what music might like or what route to take when driving. Our society, for better or for worse, asks algorithms to suggest sentences for convicted criminals. Even rely on algorithms to keep you alive, or at least not to kill you: control systems in car or in medical equipment [To understand many of ways in which algorithms influence our daily lives, see book by FRY [162]]. Word “algorithm” appears somewhere in news seemingly every day.

Therefore, it behooves you to understand algorithms not just as a student or practitioner of computer science, but as a citizen of world. Once understand algorithms, can educate others about what algorithms are, how they operate, & what their limitations are.

This book provides a comprehensive introduction to modern study of computer algorithms. It presents many algorithms & covers them in considerable depth, yet makes their design accessible to all levels of readers. All analyses are laid out, some simple, some more involved. Have tried to keep explanations clear without sacrificing depth of coverage or mathematical rigor.

Each chap presents an algorithm, a design technique, an application area, or a related topic. Algorithms are described in English & in a pseudocode designed to be readable by anyone who has done a little programming. Book contains 231 figures – many with multiple parts – illustrating how algorithms work. Since emphasize *efficiency* as a design criterion, include careful analyses of running times of algorithms.

Text is intended primarily for use in undergraduate or graduate courses in algorithms or data structures. Because it discusses engineering issues in algorithm design, as well as mathematical aspects, it is equally well suited for self-study by technical professionals.

In this, 4e, have once again updated entire book. Changes cover a broad spectrum, including new chaps & sects, color illustrations, & what hope you’ll find to be a more engaging writing style.

- **To teacher.** Have designed this book to be both versatile & complete. Should find it useful for a variety of courses, from an undergraduate course in data structures up through a graduate course in algorithms. Because have provided considerably more material than can fit in a typical 1-term course, can select material that best supports course wish to teach.

Should find it easy to organize your course around just chaps you need. Have made chaps relatively self-contained, so that you need not worry about an unexpected & unnecessary dependence of 1 chap on another. Whereas in an undergraduate course, might use only some secs from a chap, in a graduate course, might cover entire chap.

Have included 931 exercises & 162 problems. Each sect ends with exercises, & each chap ends with problems. Exercises are generally short questions that test basic mastery of material. Some are simple self-check thought exercises, but many are substantial & suitable as assigned homework. Problems include more elaborate case studies which often introduce new material. They often consist of several parts that lead student through steps required to arrive at a solution.

As with 3e of this book, have made publicly available solutions to some, but by no means all, of problems & exercises. Can find these solutions on our website, <http://mitpress.mit.edu/algorithms/>. Want to check this site to see whether it contains solution to an exercise or problem that you plan to assign. Since set of solutions that post might grow over time, recommend: check site each time teach course.

Have starred sects & exercises that are more suitable for graduate students than for undergraduates. A starred sect is not necessarily more difficult than an unstarred one, but it may require an understanding of more advanced mathematics. Likewise, starred exercises may require an advanced background or more than average creativity.

- **To student.** Hope: this textbook provides you with an enjoyable introduction to field of algorithms. Have attempted to make every algorithm accessible & interesting. To help you when encounter unfamiliar or difficult algorithms, describe each one in a step-by-step manner. Also provide careful explanations of mathematics needed to understand analysis of algorithms & supporting figures to help visualize what is going on.

Since this book is large, your class will probably cover only a portion of its material. Although hope: will find this book helpful to you as a course textbook now, have also tried to make it comprehensive enough to warrant space on your future professional bookshelf.

What are prerequisites for reading this book?

- \* Need some programming experience. In particular, should understand recursive procedures & simple data structures, e.g. arrays & linked lists (although Sect. 10.2 covers linked lists & a variant that you may find new).
- \* You should have some facility with mathematical proofs, & especially proofs by mathematical induction. A few portions of book rely on some knowledge of elementary calculus. Although this book uses mathematics throughout, Part I & Appendices A–D teach you all mathematical techniques you will need.

Website <http://mitpress.mit.edu/algorithms/>, links to solutions for some of problems & exercises. Feel free to check your solutions against ours.

- **To the professional.** Wide range of topics in this book makes it an excellent handbook on algorithms. Because each chap is relatively self-contained, can focus on topics most relevant to you.

Since most of algorithms discuss have great practical utility, address implementation concerns & other engineering issues. Often provide practical alternatives to few algorithms that are primarily of theoretical interest.

If wish to implement any of algorithms, should find translation of our pseudocode into your favorite programming language to be a fairly straightforward task. Have designed pseudocode to present each algorithm clearly & succinctly. Consequently, do not address error handling & other software-engineering issues that require specific assumptions about your programming environment. Attempt to present each algorithm simply & directly without allowing idiosyncrasies of a particular programming language to obscure its essence. If used to 0-origin arrays, might find our frequent practice of indexing arrays from 1 a minor stumbling block. Can always either subtract 1 from our indices or just overallocate array & leave position 0 unused. Understand: if using this book outside of a course, then might be unable to check your solutions to problems & exercises against solutions provided by an instructor. Website <http://mitpress.mit.edu/algorithms/>, links to solutions for some of problems & exercises so that can check work.

- **To colleagues.** Have supplied an extensive bibliography & pointers to current literature. Each chap ends with a set of chap notes that give historical details & references. Chap notes do not provide a complete reference to whole field of algorithms, however. Though it may be hard to believe for a book of this size, space constraints prevented us from including many interesting algorithms.

Despite myriad requests from students for solutions to problems & exercises, have adopted policy of not citing references for them, removing temptation for students to look up a solution rather than to discover it themselves.

- **Changes for 4e.** As said about changes for 2e & 3e, depending on how you look at it, book changed either not much or quite a bit. A quick look at table of contents shows: most of 3e chaps & sects appear in 4e. Removed 3 chaps & several sects, but have added 3 new chaps & several new sects apart from these new chaps.

Kept hybrid organization (tổ chức hỗn hợp) from 1st 3 editions. Rather than organizing chaps only by problem domains or only according to techniques, this book incorporates elements of both. It contains technique-based chaps on divide-&-conquer, dynamic programming, greedy algorithms, amortized analysis, augmenting data structures, NP-completeness, & approximation algorithms. But it also has entire parts on sorting, on data structures for dynamic sets, & on algorithms for graph problems. Find: although need to know how to apply techniques for designing & analyzing algorithms, problems seldom announce to you which techniques are most amenable to solving them.

Some of changes in 4e apply generally across book, & some are specific to particular chaps or sects. Here is a summary of most significant general changes:

- \* Added 140 new exercises & 22 new problems. Also improved many of old exercises & problems, often as result of reader feedback. (Thanks to all readers who made suggestions.)
- \* Have color! With designers from MIT Press, selected a limited palette, devised to convey information & to be pleasing to eye. (Delighted to display red-black trees in – get this – red & black!) To enhance readability, defined terms, pseudocode comments, & page numbers in index are in color.
- \* Pseudocode procedures appear on a tan background to make them easier to spot, & they do not necessarily appear on page of their 1st ref. When they don't, text directs you to relevant page. In same vein, nonlocal refs to numbers equations, theorems, lemmas, & corollaries include page number.
- \* Removed topics that were rarely taught. Dropped in their entirety chaps on Fibonacci heaps, van Emde Boas trees, & computational geometry. In addition, following material was excised (vật liệu đã được cắt bỏ): maximum-subarray problem, implementing pointers & objects, perfect hashing, randomly built binary search trees, matroids, push-relabel algorithms for maximum flow, iterative fast Fourier transform method, details of simplex algorithm for linear programming, & integer factorization. Can find all removed material on website <http://mitpress.mit.edu/algorithms/>.
- \* Reviewed entire book & rewrote sentences, paragraphs, & sects to make writing cleaner, more personal, & gender neutral. E.g., “traveling-salesman problem” in prev editions is now called “traveling-salesperson problem.” Believe: critically important for engineering & science, including our own field of CS, to be welcoming to everyone. (The 1 place that stumped

us is in Chap. 13, which requires a term for a parent's sibling. Because English language has no such gender-neutral term, regretfully stuck with "uncle".)

- \* Chap notes, bibliography, & index were updated, reflecting dramatic growth of field of algorithm since 3e.
- \* Corrected errors, posting most corrections on our website of 3e errata. Those what were reported while were in full swing preparing this edition were not posted, but were corrected in this edition. (Thanks again to all readers who helped us identify issues.)

Specific changes for 4e include following:

- \* Renamed Chap. 3 & added a sect giving an overview of asymptotic notation before delving into formal defs.
- \* Chap. 4 underwent substantial changes to improve its mathematical foundation & make it more robust & intuitive. Notion of an algorithmic recurrence was introduced, & topic of ignoring floors & ceilings in recurrences was addressed more rigorously. 2nd case of master theorem incorporates polylogarithmic factors, & a rigorous proof of a "continuous" version of master theorem is now provided. Also present powerful & general Akra-Bazzi method (without proof).
- \* Deterministic order-statistic algorithm in Chap. 9 is slightly different, & analyses of both randomized & deterministic order-statistic algorithms have been revamped.
- \* In addition to stacks & queues, Sect. 10.1 discusses ways to store arrays & matrices.
- \* Chap. 11 on hash tables includes a modern treatment of hash functions. Also emphasize linear probing as an efficient method for resolving collisions when underlying hardware implements caching to favor local searches.
- \* To replace sects on matroids in Chap. 15, converted a problem in 3e about offline caching into a full sect.
- \* Sect. 16.4 now contains a more intuitive explanation of potential functions to analyze table doubling & halving.
- \* Chap. 17 on augmenting data structures was relocated from Part III to Part V, reflecting our view: this techniques goes beyond basic material.
- \* Chap. 25 is a new chap about matchings in bipartite graphs. It presents algorithms to find a matching of maximum cardinality, to solve stable-marriage problem, & to find a maximum-weight matching (known as "assignment problem").
- \* Chap. 26, on task-parallel computing, has been updated with modern terminology, including name of chap.
- \* Chap. 27, which covers online algorithms, is another new chap. In an online algorithm, input arrives over time, rather than being available in its entirety at start of algorithm. Chap describes several examples of online algorithms, including determining how long to wait for an elevator before taking stairs, maintaining a linked list via move-to-front heuristic, & evaluating replacement policies for caches.
- \* In Chap. 29, removed detailed representation of simplex algorithm, as it was math heavy without really conveying many algorithmic ideas. Chap now focuses on key aspect of how to model problems as linear programs, along with essential duality property of linear programming.
- \* Sect. 32.5 adds to chap on string matching simple, yet powerful, structure of suffix arrays.
- \* Chap. 33, on ML, is 3rd new chap. Introduce several basic methods used in ML: clustering to group similar items together, weighted-majority algorithms, & gradient descent to find minimizer of a function.
- \* Sect. 34.5.6 summarizes strategies for polynomial-time reductions to show: problems are NP-hard.
- \* Proof of approximation algorithm for set-covering problem in Sect. 35.3 has been revised.
- Website. Can use website <http://mitpress.mit.edu/algorithms/>, to obtain supplementary information & to communicate with us. Website links to a list of known errors, material from 3e that is not included in 4e, solutions to selected exercises & problems, Python implementations of many of algorithms in this book, a list explaining corny professor jokes (of course), as well as other content, which may add to. Website also tells how to report errors or make suggestions.

## • I. FOUNDATIONS.

- Introduction. When design & analyze algorithms, need to be able to describe how they operate & how to design them. Also need some mathematical tools to show: your algorithms do right thing & do it efficiently. This part will get you started. Later parts of this book will build upon this base.
- 1. Chap. 1 provides an overview of algorithms & their place in modern computing systems. This chap defines what an algorithm is & lists some examples. It also makes a case for considering algorithms as a technology, alongside technologies e.g. fast hardware, graphical user interfaces, object-oriented systems, & networks.
- 2. Chap. 2, see 1st algorithms, which solve problem of sorting a sequence of  $n$  numbers. They are written in a pseudocode which, although not directly translatable to any conventional programming language, conveys structure of algorithm clearly enough that you should be able to implement it in language of your choice. Sorting algorithms examined are insertion sort, which uses an incremental approach, & merge sort, which uses a recursive technique known as "divide-&-conquer." Although time each requires increases with value of  $n$ , rate of increase differs between 2 algorithms. Determine these running times in Chap. 2 & develop a useful "asymptotic" notation to express them.
- 3. Chap. 3 precisely defines asymptotic notation. Use asymptotic notation to bound growth of functions – most often, functions that describe running time of algorithms – from above & below. Chap starts by informally defining most commonly used asymptotic notations & giving an example of how to apply them. It then formally defines 5 asymptotic notations & presents conventions for how to put them together. Rest of Chap. 3 is primarily a presentation of mathematical notation, more to ensure your use of notation matches that in this book than to teach new mathematical concepts.

4. Chap. 4 delves further into divide-&-conquer method introduced in Chap. 2. It provides 2 additional examples of divide-&-conquer algorithms for multiplying square matrices, including Strassen's surprising method. Chap. 4 contains methods for solving recurrences, which are useful for describing running times of recursive algorithms. In substitution method, guess an answer & prove it correct. Recursion trees provide 1 way to generate a guess. Chap. 4 also presents powerful technique of "master method", which can often use to solve recurrences that arise from divide-&-conquer algorithms. Although chap provides a proof of a foundational theorem on which master theorem depends, should feel free to employ master method without delving into proof. Chap. 4 concludes with some advanced topics.
  5. Chap. 5 introduces probabilistic analysis & randomized algorithms. Typically use probabilistic analysis to determine running time of an algorithm in cases in which, due to presence of an inherent probability distribution, running time may differ on different inputs of same size. In some cases, might assume: inputs conform to a known probability distribution, so that you are averaging running time over all possible inputs. In other cases, probability distribution comes not from inputs but from random choices made during course of algorithm. An algorithm whose behavior is determined not only by its input but by values produced by a random-number generator is a randomized algorithm. Can use randomized algorithms to enforce a probability distribution on inputs – thereby ensuring: no particular input always causes poor performance – or even to bound error rate of algorithms that are allowed to produce incorrect results on a limited basis.
  6. Appendices A–D contain other mathematical material that you will find helpful as read this book. Might have seen much of material in appendix chaps before having read this book (although specific defs & notational conventions use may differ in some cases from what you have seen in past), & so you should think of appendices as reference material. On other hand, probably have not already seen most of material in Part I. All chaps in Part I & appendices are written with a tutorial flavor.
- 1. Role of Algorithms in Computing. What are algorithms? Why is study of algorithms worthwhile? What is role of algorithms relative to other technologies used in computers? This chap will answer these questions.
    - \* 1.1. Algorithms. Informally, an *algorithm* is any well-defined computational procedure that takes some value, or set of values, as input & produces some value, or set of values, as output in a finite amount of time. An algorithm is thus a sequence of computational steps that transform input into output.  
p. 28
  - 2. Getting Started.
  - 3. Characterizing Running Times.
  - 4. Divide-&-Conquer.
  - 5. Probabilistic Analysis & Randomized Algorithms.
- II. SORTING & ORDER STATISTICS.
    - Introduction.
    - 6. Heapsort.
    - 7. Quicksort.
    - 8. Sorting in Linear Time.
    - 9. Medians & Other Statistics.
  - III. DATA STRUCTURES.
    - Introduction.
    - 10. Elementary Data Structures.
    - 11. Hash Tables.
    - 12. Binary Search Trees.
    - 13. Red-Black Trees.
  - IV. ADVANCED DESIGN & ANALYSIS TECHNIQUES.
    - Introduction. This part covers 3 important techniques used in designing & analyzing efficient algorithms: dynamic programming (Chap. 14), greedy algorithms (Chap. 15), & amortized analysis (Chap. 16). Earlier parts have presented other widely applicable techniques, e.g. divide-&-conquer, randomization, & how to solve recurrences. Techniques in this part are somewhat more sophisticated, but will be able to use them solve many computational problems. Themes introduced in this part will recur later in this book.

Dynamic programming typically applies to optimization problems in which make a set of choices in order to arrive at an optimal solution, each choice generates subproblems of same form as original problem, & same subproblems arise repeatedly. Key strategy: store solution to each such subproblem rather than recompute it. Chap. 14 shows how this simple idea can sometimes transform exponential-time algorithms into polynomial-time algorithms.

Like dynamic-programming algorithms, greedy algorithms typically apply to optimization problems in which you make a set of choices in order to arrive at an optimal solution. Idea of a greedy algorithm: make each choice in a locally optimal

manner, resulting in a faster algorithm than you get with dynamic programming. Chap. 15 will help you determine when greedy approach works.

Technique of amortized analysis (phân tích khấu hao) applies to certain algorithms that perform a sequence of similar operations. Instead of bounding cost of sequence of operations by bounding actual cost of each operation separately, an amortized analysis provides a worst-case bound on actual cost of entire sequence. 1 advantage of this approach: although some operations might be expensive, many others might be cheap. Can use amortized analysis when designing algorithms, since design of an algorithm & analysis of running time are often closely intertwined. Chap. 16 introduces 3 ways to perform an amortized analysis of an algorithm.

- 14. Dynamic Programming. Dynamic programming, like divide-&-conquer method, solves problems by combining solutions to subproblems. (“Programming” in this context refers to a tabular method, not to writing computer code.) As saw in Chaps. 2 & 4, divide-&-conquer algorithms partition problem into disjoint subproblems, solve subproblems recursively, & then combine their solutions to solve original problem. In contrast, dynamic programming applies when subproblems overlap – i.e., when subproblems share subsubproblems. In this context, a divide-&-conquer algorithm does more work than necessary, repeatedly solving common subsubproblems. A dynamic-programming algorithm solves each subsubproblem just once & then saves its answer in a table, thereby avoiding work of recomputing answer every time it solves each subsubproblem.

Dynamic programming typically applies to *optimization problems*. Such problems can have many possible solutions. Each solution has a value, & want to find a solution with optimal (minimum or maximum) value. Call such a solution *an* optimal solution to problem, as opposed to *the* optimal solution, since there may be several solutions that achieve optimal value.

To develop a dynamic-programming algorithm, follow a sequence of 4 steps:

1. Characterize structure of an optimal solution.
2. Recursively define value of an optimal solution.
3. Compute value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

Steps 1–3 form basis of a dynamic-programming solution to a problem. If need only value of an optimal solution, & not solution itself, then can omit step 4. When do perform step 4, it often pays to maintain additional information during step 3 so that can easily construct an optimal solution.

Sects that follow use dynamic-programming method to solve some optimization problems. Sects. 14.1 examines problem of cutting a rod into rods of smaller length in a way that maximizes their total value. Sect. 14.2 shows how to multiply a chain of matrices while performing fewest total scalar multiplications. Given these examples of dynamic programming, Sect. 14.3 discusses 2 key characteristics that a problem must have for dynamic programming to be a viable solution technique. Sect. 14.4 then shows how to find longest common subsequence of 2 sequences via dynamic programming. Finally, Sect. 14.5 uses dynamic programming to construct binary search trees that are optimal, given a known distribution of keys to be looked up.

\* 14.1. Rod cutting. 1st example uses dynamic programming to solve a simple problem in deciding where to cut steel rods. Serling Enterprises buys long steel rods & cuts them into shorter rods, which it then sells. Each cut is free. Management of Serling Enterprises wants to know best way to cut up rods.

Serling Enterprises has a table giving, for  $i = 1, 2, \dots$ , price  $p_i$  in dollars that they charge for a rod of length  $i$  inches. Length of each rod in inches is always an integer. Fig. 14.1: A sample price table for rods. Each rod of length  $i$  inches earns company  $p_i$  dollars of revenue. gives a sample price table.

*Rod-cutting problem* is following. Given a rod of length  $n$  inches & a table of prices  $p_i$  for  $i = 1, \dots, n$ , determine maximum revenue  $r_n$  obtainable by cutting up rod & selling pieces. If price  $p_n$  for a rod of length  $n$  is large enough, an optimal solution might require no cutting at all.

Consider case when  $n = 4$ . Fig. 14.2: 8 possible ways of cutting a rod of length 4. Above each piece is value of that piece, according to sample price chart of Fig. 14.1. Optimal strategy is part (c) – cutting rod into 2 pieces of length 2 – which has total value 10. shows all ways to cut up a rod of 4 inches in length, including way with no cuts at all. Cutting a 4-inch rod into 2 2-inch pieces produces revenue  $p_2 + p_2 = 5 + 5 = 10$ , which is optimal.

Serling Enterprises can cut up a rod of length  $n$  in  $2^{n-1}$  different ways, since they have an independent option of cutting, or not cutting, at distance  $i$  inches from left end, for  $i = 1, \dots, n-1$  [If pieces are required to be cut in order of monotonically increasing size, there are fewer ways to consider. For  $n = 4$ , only 5 such ways are possible: Number of

ways is called *partition function*, which is approximately equal to  $\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{4n\sqrt{3}}$ . This quantity is  $< 2^{n-1}$ , but still much greater than any polynomial in  $n$ . Won’t pursue this line of inquiry further, however.] Denote a decomposition into pieces using ordinary additive notation, so that  $7 = 2 + 2 + 3$  indicates: a rod of length 7 is cut into 3 pieces – 2 of length 2 & 1 of length 3. If an optimal solution cuts rod into  $k$  pieces, for some  $1 \leq k \leq n$ , then an optimal decomposition  $n = \sum_{j=1}^k i_j$  of rod into pieces of lengths  $i_1, \dots, i_k$  provides maximum corresponding revenue  $r_n = \sum_{j=1}^k p_{i_j}$ .

For sample problem in Fig. 14.1, can determine optimal revenue figures  $r_i$ , for  $i = 1, \dots, 10$ , by inspection, with corresponding optimal decompositions [value of  $r_1, \dots, r_{10}$ ].

More generally, can express values  $r_n$  for  $n \geq 1$  in terms of optimal revenues from shorter rods: (14.1)

$$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1\}.$$

1st argument  $p_n$  corresponds to making no cuts at all & selling rod of length  $n$  as is. Other  $n-1$  arguments to max correspond to maximum revenue obtained by making an initial cut of rod into 2 pieces of size  $i, n-i$ , for each  $i = 1, \dots, n-1$ ,



& then optimally cutting up those pieces further, obtaining revenues  $r_i, r_{n-i}$  from those 2 pieces. Since don't know ahead of time which value of  $i$  optimizes revenue, have to consider all possible values for  $i$  & pick the one that maximizes revenue. Also have option of picking no  $i$  at all if greatest revenue comes from selling rod uncut.

To solve original problem of size  $n$ , solve smaller problems of same type. Once make 1st cut, 2 resulting pieces form independent instances of rod-cutting problem. Overall optimal solution incorporates optimal solutions to 2 resulting subproblems, maximizing revenue from each of those 2 pieces. Say: rod-cutting problem exhibits *optimal substructure*: optimal solutions to a problem incorporate optimal solutions to related subproblems, which you may solve independently. In a related, but slightly simpler, way to arrange a recursive structure for rod-cutting problem, view a decomposition as consisting of a 1st piece of length  $i$  cut off left-hand end, & then a right-hand remainder of length  $n-i$ . Only remainder, & not 1st piece, may be further divided. Think of every decomposition of a length- $n$  rod in this way: as a 1st piece followed by some decomposition of remainder. Then can express solution with no cuts at all by saying: 1st piece has size  $i = n$  & revenue  $p_n$  & remainder has size 0 with corresponding revenue  $r_0 = 0$ . Thus obtain following simpler version of equation (14.1): (14.2)

$$r_n = \max\{p_i + r_{n-i} : 1 \leq i \leq n\}.$$

In this formulation, an optimal solution embodies solution to only 1 related subproblem – remainder – rather than 2.

• **Recursive top-down implementation.** CUT-ROD procedure implements computation implicit in (14.2) in a straightforward, top-down, recursive manner. It takes as input an array  $p[1 : n]$  of prices & an integer  $n$ , & it returns maximum revenue possible for a rod of length  $n$ . For length  $n = 0$ , no revenue is possible, & so CUT-ROD returns 0 in line 2. Line 3 initializes maximum revenue  $q$  to  $-\infty$ , so that for loop in lines 4–5 correctly computes  $q = \max\{p_i + \text{CUT-ROD}(p, n-i) : 1 \leq i \leq n\}$ . Line 6 then returns this value. A simple induction on  $n$  proves: this answer = desired answer  $r_n$  using (14.2). [CUT-ROD( $p, n$ ) algorithm].

If code up CUT-ROD in favorite programming language & run it on your computer, find: once input size becomes moderately large, your program takes a long time to run. For  $n = 40$ , your program may take several minutes & possibly  $> 1$  hour. For large values of  $n$ , also discover: each time increase  $n$  by 1, your program's running time approximately doubles.

Why is CUT-ROD so inefficient? Problem: CUT-ROD calls itself recursively over & over again with same parameter values, i.e., it solves same subproblems repeatedly. Fig. 14.3: Recursion tree showing recursive calls resulting from a call CUT-ROD( $p, n$ ) for  $n = 4$ . Each node label gives size  $n$  of corresponding subproblem, so that an edge from a parent with label  $s$  to a child with label  $t$  corresponds to cutting off an initial piece of size  $s - t$  & leaving a remaining subproblem of size  $t$ . A path from root to a leaf corresponds to 1 of  $2^{n-1}$  ways of cutting up a rod of length  $n$ . In general, this recursion tree has  $2^n$  nodes &  $2^{n-1}$  leaves. shows a recursion tree demonstrating what happens for  $n = 4$ : CUT-ROD( $p, n$ ) calls CUT-ROD( $p, n-i$ ) for  $i = 1, \dots, n$ . Equivalently, CUT-ROD( $p, n$ ) calls CUT-ROD( $p, j$ ) for each  $j = 0, 1, \dots, n-1$ . When this process unfolds recursively, amount of work done, as a function of  $n$ , grows explosively.

To analyze running time of CUT-ROD, let  $T(n)$  denote total number of calls made to CUT-ROD( $p, n$ ) for a particular value of  $n$ . This expression equals number of nodes in a subtree whose root is labeled  $n$  in recursion tree. Count includes initial call at its root. Thus,  $T(0) = 1$  & (14.3)

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j).$$

Initial 1 is for call at root, & term  $T(j)$  counts number of calls (including recursive calls) due to call CUT-ROD( $p, n-i$ ), where  $j = n-i$ . Exercise 14.1-1:  $T(n) = 2^n$  & so running time of CUT-ROD is exponential in  $n$ .

In retrospect, this exponential running time is not so surprising. CUT-ROD explicitly considers all possible ways of cutting up a rod of length  $n$ . How many ways are there? A rod of length  $n$  has  $n-1$  potential locations to cut. Each possible way to cut up rod makes a cut at some subset of these  $n-1$  locations, including empty set, which makes for no cuts. Viewing each cut location as a distinct member of a set of  $n-1$  elements, can see: there are  $2^{n-1}$  subsets. Each leaf in recursion tree of Fig. 14.3 corresponds to 1 possible way to cut up rod. Hence, recursion tree has  $2^{n-1}$  leaves. Labels on simple path from root to a leaf give sizes of each remaining right-hand piece before making each cut. I.e., labels give corresponding cut points, measured from right-hand end of rod.

• **Using dynamic programming for optimal rod cutting.**

- 15. Greedy Algorithms.
- 16. Amortized Analysis.
- V. ADVANCED DATA STRUCTURES.
  - Introduction.
  - 17. Augmenting Data Structures.
  - 18. B-Trees.
  - 19. Data Structures for Disjoint Sets.
- VI. GRAPH ALGORITHMS.

- **Introduction.** Graphs problems pervade CS, & algorithms for working with them are fundamental to the field. Hundreds of interesting computational problems are couched in terms of graphs. This part touches on a few of the more significant ones.
  - Các vấn đề về đồ thị tràn ngập trong CS, & các thuật toán để làm việc với chúng là nền tảng cho lĩnh vực này. Hàng trăm vấn đề tính toán thú vị được trình bày dưới dạng đồ thị. Phần này đề cập đến 1 số vấn đề quan trọng hơn.
  - Chap. 20 shows how to represent a graph in a computer & then discusses algorithms based on searching a graph using either BFS or DFS. Chap gives 2 applications of DFS: topologically sorting a directed acyclic graph & decomposing a directed graph into its strongly connected components.
  - Chương 20 trình bày cách biểu diễn đồ thị trong máy tính & sau đó thảo luận về các thuật toán dựa trên việc tìm kiếm đồ thị bằng BFS hoặc DFS. Chương này đưa ra 2 ứng dụng của DFS: sắp xếp tô pô 1 đồ thị có hướng không có chu trình & phân tích 1 đồ thị có hướng thành các thành phần liên thông mạnh của nó.
  - Chap. 21 describes how to compute a minimum-weight spanning tree of a graph: least-weight way of connecting all of vertices together when each edge has an associated weight. Algorithms for computing minimum spanning trees serve as good examples of greedy algorithms (Chap. 15).
  - Chương 21 mô tả cách tính cây khung có trọng số tối thiểu của đồ thị: cách có trọng số tối thiểu để kết nối tất cả các đỉnh với nhau khi mỗi cạnh có trọng số liên quan. Các thuật toán để tính cây khung tối thiểu đóng vai trò là ví dụ tốt về thuật toán tham lam (Chương 15).
  - Chaps. 22–23 consider how to compute shortest paths between vertices when each edge has an associated length or “weight”. Chap. 22 shows how to find shortest paths from a given source vertex to all other vertices, & Chap. 23 examines methods to compute shortest paths between every pair of vertices.
  - Chương 22–23 xem xét cách tính đường đi ngắn nhất giữa các đỉnh khi mỗi cạnh có chiều dài hoặc “trọng số” liên quan. Chương 22 cho thấy cách tìm đường đi ngắn nhất từ 1 đỉnh nguồn cho trước đến tất cả các đỉnh khác, & Chương 23 kiểm tra các phương pháp tính đường đi ngắn nhất giữa mọi cặp đỉnh.
  - Chap. 24 shows how to compute a maximum flow of material in a flow network, which is a directed graph having a specified source vertex of material, a specified sink vertex, & specified capacities for amount of material that can traverse each directed edge. This general problem arises in many forms, & a good algorithm for computing maximum flows can help solve a variety of related problems efficiently.
  - Chương 24 trình bày cách tính toán lưu lượng vật liệu cực đại trong mạng lưu lượng, là đồ thị có hướng có đỉnh nguồn vật liệu xác định, đỉnh chìm xác định, & dung lượng xác định cho lượng vật liệu có thể đi qua mỗi cạnh có hướng. Vấn đề chung này phát sinh ở nhiều dạng, & 1 thuật toán tốt để tính lưu lượng cực đại có thể giúp giải quyết hiệu quả nhiều vấn đề liên quan.
  - Chap. 25 explores matchings in bipartite graphs: methods for pairing up vertices that are partitioned into 2 sets by selecting edges that go between sets. Bipartite-matching problems model several situations that arise in real world. Chap examines how to find a matching of maximum cardinality; “stable-marriage problem”, which has highly practical application of matching medical residents to hospitals; & assignment problems, which maximize total weight of a bipartite matching.
  - Chương 25 khám phá các phép ghép trong đồ thị hai phần: các phương pháp ghép các đỉnh được phân vùng thành 2 tập bằng cách chọn các cạnh nằm giữa các tập. Các bài toán ghép hai phần mô hình hóa 1 số tình huống phát sinh trong thế giới thực. Chương này xem xét cách tìm phép ghép có số lượng tối đa; “bài toán hôn nhân ổn định”, có ứng dụng thực tế cao trong việc ghép các bác sĩ nội trú với các bệnh viện; & các bài toán gán, giúp tối đa hóa tổng trọng số của phép ghép 2 phần.
- When characterize running time of a graph algorithm on a given graph  $G = (V, E)$ , usually measure size of input in terms of number of vertices  $|V|$  & number of edges  $|E|$  of graph. I.e., denote size of input with 2 parameters, not just 1. Adopt a common notational convention for these parameters. Inside asymptotic notation (e.g.  $O$ -notation or  $\Theta$ -notation), & *only* inside such notation, symbol  $V$  denotes  $|V|$  & symbol  $E$  denotes  $|E|$ . E.g., might say, “algorithm runs in  $O(VE)$  time”, i.e., algorithm runs in  $O(|V||E|)$  time. This convention makes running-time formulas easier to read, without risk of ambiguity.
  - Khi mô tả thời gian chạy của 1 thuật toán đồ thị trên 1 đồ thị cho trước  $G = (V, E)$ , thường đo kích thước của đầu vào theo số đỉnh  $|V|$  & số cạnh  $|E|$  của đồ thị. Tức là, biểu thị kích thước của đầu vào bằng 2 tham số, không chỉ 1. Áp dụng 1 quy ước ký hiệu chung cho các tham số này. Bên trong ký hiệu tiệm cận (ví dụ: ký hiệu  $O$  hoặc ký hiệu  $\Theta$ ), & *only* bên trong ký hiệu như vậy, ký hiệu  $V$  biểu thị  $|V|$  & ký hiệu  $E$  biểu thị  $|E|$ . Ví dụ, có thể nói, “thuật toán chạy trong thời gian  $O(VE)$ ”, tức là thuật toán chạy trong thời gian  $O(|V||E|)$ . Quy ước này giúp các công thức thời gian chạy dễ đọc hơn, không có nguy cơ mơ hồ.
- Another convention adopted appears in pseudocode. Denote vertex set of a graph  $G$  by  $G.V$  & its edge set by  $G.E$ . I.e., pseudocode views vertex & edge sets as attributes of a graph.
  - Một quy ước khác được áp dụng xuất hiện trong mã giả. Ký hiệu tập đỉnh của đồ thị  $G$  bởi  $G.V$  & tập cạnh của nó bởi  $G.E$ . I.e., mã giả xem tập đỉnh & cạnh là các thuộc tính của đồ thị.

- **20. Elementary Graph Algorithms.** This chap presents methods for representing a graph & for searching a graph. Searching a graph means systematically following edges of graph so as to visit vertices of graph. A graph-searching algorithm can discover much about structure of a graph. Many algorithms begin by searching their input graph to obtain this structural information. Several other graph algorithms elaborate on basic graph searching. Techniques for searching a graph lie at heart of field of graph algorithms.
  - Chương này trình bày các phương pháp để biểu diễn 1 đồ thị & để tìm kiếm 1 đồ thị. Tìm kiếm 1 đồ thị có nghĩa là theo dõi 1 cách có hệ thống các cạnh của đồ thị để thăm các đỉnh của đồ thị. Một thuật toán tìm kiếm đồ thị có thể khám phá



nhiều về cấu trúc của 1 đồ thị. Nhiều thuật toán bắt đầu bằng cách tìm kiếm đồ thị đầu vào của chúng để có được thông tin cấu trúc này. Một số thuật toán đồ thị khác giải thích chi tiết về tìm kiếm đồ thị cơ bản. Các kỹ thuật để tìm kiếm 1 đồ thị nằm ở trung tâm của lĩnh vực thuật toán đồ thị.

Sect. 20.1 discusses 2 most common computational representations of graphs: as adjacency lists & as adjacency matrices. Sect. 20.2 presents a simple graph-searching algorithm called BFS & shows how to create a breadth-1st tree. Sect. 20.3 presents depth-1st search & proves some standard results about order in which DFS visits vertices. Sect. 20.4 provides our 1st real application of DFS: topological sorting a directed acyclic graph. A 2nd application of DFS, finding strongly connected components of a directed graph, is topic of Sect. 20.5.

– Mục 20.1 thảo luận về 2 biểu diễn tính toán phổ biến nhất của đồ thị: dưới dạng danh sách kề & dưới dạng ma trận kề. Mục 20.2 trình bày 1 thuật toán tìm kiếm đồ thị đơn giản có tên là BFS & cho thấy cách tạo cây theo chiều rộng 1. Mục 20.3 trình bày tìm kiếm theo chiều sâu 1 & chứng minh 1 số kết quả chuẩn về thứ tự mà DFS duyệt các đỉnh. Mục 20.4 cung cấp ứng dụng thực tế đầu tiên của chúng tôi về DFS: sắp xếp tô pô cho đồ thị có hướng phi chu trình. Ứng dụng thứ 2 của DFS, tìm các thành phần liên thông mạnh của đồ thị có hướng, là chủ đề của Mục 20.5.

- 21. Minimum Spanning Trees. Can choose between 2 standard ways to represent a graph  $G = (V, E)$ : as a collection of adjacency lists or as an adjacency matrix. Either way applies to both directed & undirected graphs. Because adjacency-list representation provides a compact way to represent *sparse* graphs – those for which  $|E|$  is much less than  $|V|^2$  – it is usually method of choice. Most of graph algorithms presented in this book assume: an input graph is represented in adjacency-list form. Might prefer an adjacency-matrix representation, however, when graph is *dense* –  $|E|$  is close to  $|V|^2$  – or when you need to be able to tell quickly whether there is an edge connecting 2 given vertices. E.g., 2 of all-pairs shortest-paths algorithms presented in Chap. 23 assume: their input graphs are represented by adjacency matrices.

– Cây khung nhỏ nhất. Có thể chọn giữa 2 cách chuẩn để biểu diễn đồ thị  $G = (V, E)$ : dưới dạng tập hợp các danh sách kề hoặc dưới dạng ma trận kề. Cả hai cách đều áp dụng cho cả đồ thị có hướng & vô hướng. Bởi vì biểu diễn danh sách kề cung cấp 1 cách gọn nhẹ để biểu diễn đồ thị *thưa* – đồ thị mà  $|E|$  nhỏ hơn nhiều so với  $|V|^2$  – nên đây thường là phương pháp được lựa chọn. Hầu hết các thuật toán đồ thị được trình bày trong cuốn sách này đều giả định: đồ thị đầu vào được biểu diễn dưới dạng danh sách kề. Tuy nhiên, có thể thích biểu diễn ma trận kề hơn khi đồ thị *dày đặc* –  $|E|$  gần với  $|V|^2$  – hoặc khi bạn cần có khả năng nhanh chóng biết liệu có cạnh nào kết nối 2 đỉnh đã cho hay không. Ví dụ: 2 trong số các thuật toán đường đi ngắn nhất mọi cặp được trình bày trong Chương 23 giả định: đồ thị đầu vào của chúng được biểu diễn bằng ma trận kề.

p. 720+++

- 22. Single-Source Shortest Paths.
- 23. All-Pairs Shortest Paths.
- 24. Maximum Flow.
- 25. Matching in Bipartite Graphs.

## • VII. SELECTED TOPICS.

- Introduction.
- 26. Parallel Algorithms.
- 27. Online Algorithms.
- 28. Matrix Operations.
- 29. Linear Programming.
- 30. Polynomials & FFT.
- 31. Number-Theoretic Algorithms.
- 32. String Matching.
- 33. Machine-Learning Algorithms.
- 34. NP-Completeness.
- 35. Approximation Algorithms.

## • VIII. APPENDIX: MATHEMATICAL BACKGROUND.

- Introduction.
- A. Summations.
- B. Sets, etc.
- C. Counting & Probability.
- D. Matrices.

## 2.2 [Knu98]. DONALD ERWIN KNUTH. The Art of Computer Programming. Vol. 3: Sorting & Searching. 2e

- Preface. “Cookery is become an art, a noble science, cooks are gentlemen.” – TITUS LIVIUS, *Ab Urbe Condita* XXXIX.vi (Robert Burton, *Anatomy of Melancholy* – Giải phẫu của sự u sầu 1.2.2.2)

This book forms a natural sequel to material on information structures in Chap. 2 of Vol. 1, because it adds concept of linearly ordered data to other basic structural ideas.

– Cuốn sách này là phần tiếp theo tự nhiên của tài liệu về cấu trúc thông tin trong Chương 2 của Tập 1, vì nó bổ sung khái niệm dữ liệu được sắp xếp tuyến tính vào các ý tưởng cấu trúc cơ bản khác.

Title “Sorting & Searching” may sound as if this book is only for those systems programmers who are concerned with preparation of general-purpose sorting routines or applications to information retrieval. But in fact area of sorting & searching provides an ideal framework for discussing a wide variety of important general issues:

- How are good algorithms discovered?
- How can efficiency of algorithms be analyzed mathematically?
- How can a person choose rationally between different algorithms for same task?
- In what senses can algorithms be proved “best possible”?
- How does theory of computing interact with practical considerations?
- How can external memories like tapes, drums, or disks be used efficiently with large database?

Indeed, KNUTH believe: virtually *every* important aspect of programming arises somewhere in context of sorting or searching!

– Tiêu đề “Sắp xếp & Tìm kiếm” có vẻ như cuốn sách này chỉ dành cho những lập trình viên hệ thống quan tâm đến việc chuẩn bị các quy trình sắp xếp mục đích chung hoặc các ứng dụng để truy xuất thông tin. Nhưng trên thực tế, lĩnh vực sắp xếp & tìm kiếm cung cấp 1 khuôn khổ lý tưởng để thảo luận về nhiều vấn đề chung quan trọng:

- Các thuật toán tốt được phát hiện như thế nào?
- Hiệu quả của các thuật toán có thể được phân tích về mặt toán học như thế nào?
- Làm thế nào 1 người có thể lựa chọn hợp lý giữa các thuật toán khác nhau cho cùng 1 nhiệm vụ?
- Theo nghĩa nào thì các thuật toán có thể được chứng minh là “tốt nhất có thể”?
- Lý thuyết điện toán tương tác với các cân nhắc thực tế như thế nào?
- Làm thế nào để sử dụng hiệu quả các bộ nhớ ngoài như băng, trống hoặc đĩa với cơ sở dữ liệu lớn?

Thật vậy, KNUTH tin rằng: hầu như *mọi* khía cạnh quan trọng của lập trình đều phát sinh ở đâu đó trong bối cảnh sắp xếp hoặc tìm kiếm!

This volume comprises Chaps. 5–6 of complete series. Chap. 5 is concerned with sorting into order; this is a large subset that has been divided chiefly into 2 parts, internal sorting & external sorting. There also are supplementary sects, which develop auxiliary theories about permutations (Sect. 5.1) & about optimum techniques for sorting (Sect. 5.3). Chap. 6 deals with problem of searching for specified items in tables or files; this is subdivided into methods that search sequentially, or by comparison of keys, or by digital properties, or by hashing, & then more difficult problem of secondary key retrieval is considered. There is a surprising amount of interplay between both chaps, with strong analogies typing topics together. 2 important varieties of information structures are also discussed, in addition to those considered in Chap. 2, namely priority queues (Sect. 5.2.3) & linear lists represented as balanced trees (Sect. 6.2.3).

– Tập này bao gồm các Chương 5-6 của bộ sách hoàn chỉnh. Chương 5 đề cập đến việc sắp xếp theo thứ tự; đây là 1 tập hợp con lớn được chia chủ yếu thành 2 phần, sắp xếp nội bộ & sắp xếp ngoài. Ngoài ra còn có các giáo phái bổ sung, phát triển các lý thuyết phụ trợ về hoán vị (Mục 5.1) & về các kỹ thuật tối ưu để sắp xếp (Mục 5.3). Chương 6 đề cập đến vấn đề tìm kiếm các mục đã chỉ định trong các bảng hoặc tệp; điều này được chia thành các phương pháp tìm kiếm theo trình tự, hoặc bằng cách so sánh các khóa, hoặc bằng các thuộc tính kỹ thuật số, hoặc bằng cách băm, & sau đó là vấn đề khó hơn về việc truy xuất khóa thứ cấp được xem xét. Có 1 lượng tương tác đáng ngạc nhiên giữa cả hai chương, với các phép loại suy mạnh mẽ khi gõ các chủ đề lại với nhau. 2 loại cấu trúc thông tin quan trọng cũng được thảo luận, ngoài các loại được xem xét trong Chương 2, cụ thể là hàng đợi ưu tiên (Mục 5.2.3) & danh sách tuyến tính được biểu diễn dưới dạng cây cân bằng (Mục 6.2.3).

Like Vols. 1–2, this book includes a lot of material that does not appear in other publications. Many people have kindly written to me about their ideas, or spoken to me about them, & I hope I have not distorted material too badly when I have presented it in my own words.

– Giống như Tập 1-2, cuốn sách này bao gồm nhiều tài liệu không xuất hiện trong các ấn phẩm khác. Nhiều người đã tử tế viết thư cho tôi về ý tưởng của họ, hoặc nói chuyện với tôi về chúng, & Tôi hy vọng tôi không bóp méo tài liệu quá tệ khi tôi trình bày theo lời của riêng tôi.

I have not had time to search patent literature systematically; indeed, I decry current tendency to seek patents on algorithms (Sect. 5.4.5). If somebody sends me a copy of a relevant patent not presently cited in this book, I will dutifully refer to it

in future editions. However, I want to encourage people to continue centuries-old mathematical tradition of putting newly discovered algorithms into public domain. There are better ways to earn a living than to prevent other people from making use of one's contributions to CS.

– Tôi không có thời gian để tìm kiếm tài liệu về bằng sáng chế 1 cách có hệ thống; thực ra, tôi lên án xu hướng hiện nay là tìm kiếm bằng sáng chế về thuật toán (Mục 5.4.5). Nếu ai đó gửi cho tôi 1 bản sao của bằng sáng chế có liên quan hiện không được trích dẫn trong cuốn sách này, tôi sẽ tận tụy tham khảo nó trong các phiên bản sau. Tuy nhiên, tôi muốn khuyến khích mọi người tiếp tục truyền thống toán học đã có từ nhiều thế kỷ là đưa các thuật toán mới được phát hiện vào phạm vi công cộng. Có nhiều cách tốt hơn để kiếm sống hơn là ngăn cản người khác sử dụng những đóng góp của mình cho khoa học máy tính.

Before I retired from teaching, I used this book as a text for a student's 2nd course in data structures, at junior-to-graduate level, omitting most of mathematical material. Also used mathematical portions of this book as basis for graduate-level courses in analysis of algorithms, emphasizing especially Sects. 5.1, 5.2.2, 6.3, & 6.4. A graduate-level course on concrete computational complexity could also be based on Sects. 5.3, & 5.4.4, together with Sects. 4.3.3, 4.6.3, & 4.6.4 of Volume 2.

– Trước khi nghỉ hưu, tôi đã sử dụng cuốn sách này làm giáo trình cho khóa học thứ 2 của sinh viên về cấu trúc dữ liệu, ở trình độ từ cơ sở đến sau đại học, bỏ qua hầu hết các tài liệu toán học. Tôi cũng sử dụng các phần toán học của cuốn sách này làm cơ sở cho các khóa học sau đại học về phân tích thuật toán, đặc biệt nhấn mạnh vào Mục 5.1, 5.2.2, 6.3, & 6.4. Một khóa học sau đại học về độ phức tạp tính toán cụ thể cũng có thể dựa trên Mục 5.3, & 5.4.4, cùng với Mục 4.3.3, 4.6.3, & 4.6.4 của Tập 2.

For most part this book is self-contained, except for occasional discussions relating to MIX computer explained in Vol. 1. Appendix B contains a summary of mathematical notations used, some of which are a little different from those found in traditional mathematics books.

– Phần lớn cuốn sách này là nội dung độc lập, ngoại trừ 1 số cuộc thảo luận thỉnh thoảng liên quan đến máy tính MIX được giải thích trong Tập 1. Phụ lục B tóm tắt các ký hiệu toán học được sử dụng, 1 số trong đó hơi khác so với những ký hiệu có trong các sách toán học truyền thống.

- **Preface to 2e.** This new edition matches 3es of Vols. 1–2, in which I have been able to celebrate completion of  $\text{T}_{\text{E}}\text{X}$  & METAFONT by applying those systems to publications they were designed for.

Conversion to electronic format has given me opportunity to go over every word of text & every punctuation mark. Have tried to retain youthful exuberance of original sentences while perhaps adding some more mature judgment. Dozens of new exercises have been added; dozens of old exercises have been given new & improved answers. Changes appear everywhere, but most significantly in Sects. 5.1.4 (about permutations & tableaux), 5.3 (about optimum sorting), 5.4.9 (about disk sorting), 6.2.2 (about entropy), 6.4 (about universal hashing), & 6.5 (about multidimensional trees & tries).

– Việc chuyển đổi sang định dạng điện tử đã cho tôi cơ hội xem xét lại từng từ trong văn bản & từng dấu câu. Đã cố gắng giữ lại sự phấn khích trẻ trung của các câu gốc trong khi có lẽ thêm 1 số phán đoán chín chắn hơn. Hàng chục bài tập mới đã được thêm vào; hàng chục bài tập cũ đã được đưa ra & câu trả lời mới được cải thiện. Những thay đổi xuất hiện ở khắp mọi nơi, nhưng đáng kể nhất là ở các Mục. 5.1.4 (về hoán vị & bảng), 5.3 (về sắp xếp tối ưu), 5.4.9 (về sắp xếp đĩa), 6.2.2 (về entropy), 6.4 (về băm phổ quát), & 6.5 (về cây đa chiều & thử).

*The Art of Computer Programming* is, however, still a work in progress. Researching on sorting & searching continues to grow at a phenomenal rate. Therefore some parts of this book are headed by an “under construction” icon, to apologize for fact: material is not up-to-date. E.g., if I were teaching an undergraduate class on data structures today, I would surely discuss randomized structures e.g. treaps at some length; but at present, only able to cite principal papers on subject, & to announce plans for a future Sect. 6.2.5. My files are bursting with important material that I plan to include in final, glorious, 3e of Vol. 3, perhaps 17 years from now. But I must finish Vols. 4–5 1st, & I do not want to delay their publication any more than absolutely necessary.

– Tuy nhiên, *Nghệ thuật lập trình máy tính* vẫn đang trong quá trình hoàn thiện. Nghiên cứu về sắp xếp & tìm kiếm tiếp tục phát triển với tốc độ phi thường. Do đó, 1 số phần của cuốn sách này được đánh dấu bằng biểu tượng “đang xây dựng”, để xin lỗi vì thực tế: tài liệu không được cập nhật. Ví dụ, nếu tôi đang giảng dạy 1 lớp đại học về cấu trúc dữ liệu ngày nay, tôi chắc chắn sẽ thảo luận về các cấu trúc ngẫu nhiên, ví dụ như treaps ở 1 số phần; nhưng hiện tại, chỉ có thể trích dẫn các bài báo chính về chủ đề này, & để công bố kế hoạch cho Phần 6.2.5 trong tương lai. Các tệp của tôi đang tràn ngập tài liệu quan trọng mà tôi dự định đưa vào bản cuối cùng, tuyệt vời, 3e của Tập 3, có lẽ là 17 năm nữa. Nhưng tôi phải hoàn thành Tập 4-5 trước, & Tôi không muốn trì hoãn việc xuất bản chúng lâu hơn mức cần thiết.

“There are certain common Privileges of a Writer, the Benefit whereof, I hope, there will be no Reason to doubt; Particularly, that where I am not understood, it shall be concluded, that something very useful & profound is coucht underneath.” – JONATHAN SWIFT, *Tale of a Tub*, Preface (1704) – Có 1 số Quyền lợi chung của 1 Nhà văn, mà tôi hy vọng rằng sẽ không có Lý do gì để nghi ngờ; Đặc biệt, khi tôi không được hiểu, có thể kết luận rằng có điều gì đó rất hữu ích & sâu sắc ẩn chứa bên dưới.

- **Notes on Exercises.** Exercises in this set of books have been designed for self-study as well as for classroom study. If is difficult, if not impossible, for anyone to learn a subject purely by reading about it, without applying information to specific problems & thereby being encouraged to think about what has been read. Furthermore, we all learn best things that we have discovered

for ourselves. Therefore exercises form a major part of this work; a definite attempt has been made to keep them as informative as possible & to select problems that are enjoyable as well as instructive.

– Ghi chú về Bài tập. Các bài tập trong bộ sách này được thiết kế để tự học cũng như để học trên lớp. Thật khó, nếu không muốn nói là không thể, đối với bất kỳ ai muốn học 1 môn học chỉ bằng cách đọc về môn học đó, mà không áp dụng thông tin vào các bài toán cụ thể & do đó được khuyến khích suy nghĩ về những gì đã đọc. Hơn nữa, tất cả chúng ta đều học được những điều tốt nhất mà chúng ta tự khám phá ra. Do đó, các bài tập đóng vai trò chính trong công việc này; chúng tôi đã nỗ lực hết sức để giữ cho chúng mang tính thông tin nhất có thể & để chọn các bài toán vừa thú vị vừa mang tính hướng dẫn.

## • Chap. 5: Sorting.

- “There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in introduction of a new order of things.” – NICCOLÒ MACHIAVELLI, *The Prince* (1513)
  - Không có gì khó khăn hơn để thực hiện, nguy hiểm hơn để tiến hành, hoặc không chắc chắn hơn về thành công, hơn là việc dẫn đầu trong việc giới thiệu 1 trật tự mới.
- “But you can’t look up all those license numbers in time,” Drake objected. “We don’t have to, Paul. We merely arrange a list & look for duplications.” – PERRY MASON, in *he Case of the Angry Mourner* (1951)
  - Nhưng bạn không thể tra cứu tất cả các số giấy phép đó kịp thời,” Drake phản đối. “Chúng tôi không cần phải làm vậy, Paul. Chúng tôi chỉ sắp xếp 1 danh sách & tìm kiếm sự trùng lặp.
- ““Treesort” Computer – With this new ‘computer-approach’ to nature study you can quickly identify over 260 different trees of U.S., Alaska, & Canada, even palms, desert trees, & other exotics. To sort, you simply insert the needle.” – EDMUND SCIENTIFIC COMPANY, *Catalog* (1964)
  - Máy tính “Treesort” – Với ‘cách tiếp cận máy tính’ mới này đối với nghiên cứu thiên nhiên, bạn có thể nhanh chóng xác định hơn 260 loại cây khác nhau của Hoa Kỳ, Alaska, & Canada, thậm chí cả cây cọ, cây sa mạc, & các loại cây ngoại lai khác. Để phân loại, bạn chỉ cần chèn kim.

In this chap, study a topic that arises frequently in programming: rearrangement of items into ascending or descending order. Imagine how hard it would be to us a dictionary if its words were not alphabetized! In a similar way, order in which items are stored in computer memory often has a profound influence on speed & simplicity of algorithms that manipulate those items.

– Trong chương này, hãy nghiên cứu 1 chủ đề thường gặp trong lập trình: sắp xếp lại các mục theo thứ tự tăng dần hoặc giảm dần. Hãy tưởng tượng xem sẽ khó khăn như thế nào đối với chúng ta khi sử dụng từ điển nếu các từ trong đó không được sắp xếp theo thứ tự bảng chữ cái! Tương tự như vậy, thứ tự các mục được lưu trữ trong bộ nhớ máy tính thường có ảnh hưởng sâu sắc đến tốc độ & tính đơn giản của các thuật toán xử lý các mục đó.

Although dictionaries of English language define “sorting” as process of separating or arranging things according to class or kind, computer programmers traditionally use word in much more special sense of marshaling things into ascending or descending order. Process should perhaps be called *ordering*, not sorting; but anyone who tries to call it “ordering” is soon led into confusion because of many different meanings attached to that word. Consider following sentence, e.g.: “Since only 2 of our tape drives were in working order, I was ordered to order more tape units in short order, in order to order data several orders of magnitude faster.” Mathematical terminology abounds with still more senses of order (order of a group, order of a permutation, order of a branch point, relations of orders, etc., etc.). Thus find word “order” can lead to chaos.

– Mặc dù các từ điển tiếng Anh định nghĩa “sắp xếp” là quá trình tách hoặc sắp xếp mọi thứ theo lớp hoặc loại, các lập trình viên máy tính theo truyền thống sử dụng từ này theo nghĩa đặc biệt hơn nhiều là sắp xếp mọi thứ theo thứ tự tăng dần hoặc giảm dần. Quá trình có lẽ nên được gọi là sắp xếp, không phải sắp xếp; nhưng bất kỳ ai cố gắng gọi nó là “sắp xếp” sẽ sớm bị nhầm lẫn vì có nhiều nghĩa khác nhau gắn liền với từ đó. Hãy xem xét câu sau, ví dụ: “Vì chỉ có 2 ổ băng của chúng tôi hoạt động bình thường, nên tôi được lệnh đặt hàng thêm các đơn vị băng trong thời gian ngắn, để sắp xếp dữ liệu nhanh hơn nhiều cấp độ.” Thuật ngữ toán học có rất nhiều nghĩa về thứ tự (thứ tự của 1 nhóm, thứ tự của 1 hoán vị, thứ tự của 1 điểm nhánh, mối quan hệ của các thứ tự, v.v., v.v.). Do đó, tìm từ “trật tự” có thể dẫn đến hỗn loạn.

Some people have suggested that “sequencing” would be best name for process of sorting into order; but this word often seems to lack right connotation, especially when equal elements are present, & it occasionally conflicts with other terminology. It is quite true that “sorting” is itself an overused word (“I was sort of out of sorts after sorting that sort of data”), but it has become firmly established in computing parlance. Therefore we shall use word “sorting” chiefly in strict sense of sorting into order, without further apologies.

– 1 số người đã gợi ý rằng “sequencing” sẽ là tên gọi tốt nhất cho quá trình sắp xếp theo thứ tự; nhưng từ này thường có vẻ thiếu đúng nghĩa, đặc biệt là khi có các phần tử bằng nhau, & đôi khi nó xung đột với các thuật ngữ khác. Đúng là “sorting” tự nó là 1 từ bị lạm dụng (“Tôi đã hơi lạc lõng sau khi sắp xếp loại dữ liệu đó”), nhưng nó đã trở nên vững chắc trong thuật ngữ máy tính. Do đó, chúng ta sẽ sử dụng từ “sorting” chủ yếu theo nghĩa chặt chẽ là sắp xếp theo thứ tự, mà không cần xin lỗi thêm.

Some of most important applications of sorting are:

1. *Solving “togetherness” problem*, in which all items with same identification are brought together. Suppose: we have 10000 items in arbitrary order, many of which have equal values; & suppose we want to rearrange data so that all items with equal values appear in consecutive positions. This is essentially the problem of sorting in older sense of word; & it can be

solved easily by sorting file in new sense of word, so that values are in ascending order,  $v_1 \leq v_2 \leq \dots \leq v_{10000}$ . Efficiency achievable in this procedure explains why original meaning of “sorting” has changed.

- *Giải quyết vấn đề “cùng nhau”*, trong đó tất cả các mục có cùng định danh được đưa lại với nhau. Giả sử: chúng ta có 10000 mục theo thứ tự tùy ý, nhiều mục trong số đó có giá trị bằng nhau; & giả sử chúng ta muốn sắp xếp lại dữ liệu sao cho tất cả các mục có giá trị bằng nhau xuất hiện ở các vị trí liên tiếp. Về cơ bản, đây là vấn đề sắp xếp theo nghĩa cũ của từ; & có thể giải quyết dễ dàng bằng cách sắp xếp tệp theo nghĩa mới của từ, sao cho các giá trị theo thứ tự tăng dần,  $v_1 \leq v_2 \leq \dots \leq v_{10000}$ . Hiệu quả đạt được trong quy trình này giải thích tại sao nghĩa gốc của “sắp xếp” đã thay đổi.
- 2. *Matching items in  $\geq 2$  files*. If several files have been sorted into same order, possible to find all of matching entries in 1 sequential pass through them, without backing up. This is principle that PERRY MASON used to help solve a murder case. We can usually process a list of information most quickly by traversing it in sequence from beginning to end, instead of skipping around at random in list, unless entire list is small enough to fit in a high-speed random-access memory. Sorting makes it possible to use sequential accessing on large files, as a feasible substitute for direct addressing.
  - *Các mục khớp trong các tệp  $\geq 2$* . Nếu 1 số tệp đã được sắp xếp theo cùng 1 thứ tự, có thể tìm thấy tất cả các mục khớp trong 1 lần duyệt tuần tự qua chúng, mà không cần sao lưu. Đây là nguyên tắc mà PERRY MASON đã sử dụng để giúp giải quyết 1 vụ án mạng. Chúng ta thường có thể xử lý danh sách thông tin nhanh nhất bằng cách duyệt theo trình tự từ đầu đến cuối, thay vì bỏ qua ngẫu nhiên trong danh sách, trừ khi toàn bộ danh sách đủ nhỏ để vừa với bộ nhớ truy cập ngẫu nhiên tốc độ cao. Sắp xếp giúp có thể sử dụng truy cập tuần tự trên các tệp lớn, như 1 giải pháp thay thế khả thi cho việc định địa chỉ trực tiếp.
- 3. *Searching for information by key values*. Sorting is also an aid to searching, hence it helps us make computer output more suitable for human consumption. In fact, a listing that has been sorted into alphabetic order often looks quite authoritative even when associated numerical information has been incorrectly computed.
  - *Tìm kiếm thông tin theo giá trị khóa*. Sắp xếp cũng là 1 công cụ hỗ trợ tìm kiếm, do đó giúp chúng ta làm cho đầu ra máy tính phù hợp hơn với nhu cầu sử dụng của con người. Trên thực tế, 1 danh sách được sắp xếp theo thứ tự chữ cái thường trông khá có thẩm quyền ngay cả khi thông tin số liên quan đã được tính toán không chính xác.

Although sorting has traditionally been used mostly for business data processing, it is actually a basic tool that every programmer should keep in mind for use in a wide variety of situations. We have discussed its use for simplifying algebraic formulas, in exercise 2.3.2–17. Exercises below illustrate diversity of typical applications.

– Mặc dù sắp xếp theo truyền thống chủ yếu được sử dụng để xử lý dữ liệu kinh doanh, nhưng thực tế đây là 1 công cụ cơ bản mà mọi lập trình viên nên ghi nhớ để sử dụng trong nhiều tình huống khác nhau. Chúng tôi đã thảo luận về việc sử dụng nó để đơn giản hóa các công thức đại số trong bài tập 2.3.2–17. Các bài tập dưới đây minh họa sự đa dạng của các ứng dụng điển hình.

1 of 1st large-scale software systems to demonstrate versatility of sorting was LARC Scientific Compiler developed by J. ERDWINN, D. E. FERGUSON, & their associates at Computer Sciences Corporation in 1960. This optimizing compiler for an extended FORTRAN language made heavy use of sorting so that various compilation algorithms were presented with relevant parts of source program in a convenient sequence. 1st pass was a lexical scan that divided FORTRAN source code into individual tokens, each representing an identifier or a constant or an operator, etc. Each token was assigned several sequence numbers; when sorted on name & an appropriate sequence number, all uses of a given identifier were brought together. “Defining entries” by which a user would specify whether an identifier stood for a function name, a parameter, or a dimensioned variable were given low sequence numbers, so that they would appear 1st among tokens having a given identifier; this made it easy to check for conflicting usage & to allocate storage w.r.t. EQUIVALENCE declarations. Information thus gathered about each identifier was now attached to each token; in this way no “symbol table” of identifiers needed to be maintained in high-speed memory. Updated tokens were then sorted on another sequence number, which essentially brought source program back into its original order except that numbering scheme was cleverly designed to put arithmetic expressions into a more convenient “Polish prefix” form. Sorting was also used in later phases of compilation, to facilitate loop optimization, to merge error messages into listing, etc. In short, compiler was designed so that virtually all processing could be done sequentially from files that were stored in an auxiliary drum memory, since appropriate sequence numbers were attached to data in such a way that it could be sorted into various convenient arrangements.

– 1 trong những hệ thống phần mềm quy mô lớn đầu tiên chứng minh tính linh hoạt của việc sắp xếp là LARC Scientific Compiler do J. ERDWINN, D. E. FERGUSON, & các cộng sự của họ tại Computer Sciences Corporation phát triển vào năm 1960. Trình biên dịch tối ưu hóa này cho ngôn ngữ FORTRAN mở rộng đã sử dụng rất nhiều sắp xếp để các thuật toán biên dịch khác nhau được trình bày với các phần có liên quan của chương trình nguồn theo trình tự thuận tiện. Lần quét đầu tiên là quét từ vệt chia mã nguồn FORTRAN thành các mã thông báo riêng lẻ, mỗi mã thông báo đại diện cho 1 mã định danh hoặc 1 hằng số hoặc 1 toán tử, v.v. Mỗi mã thông báo được gán 1 số số thứ tự; khi được sắp xếp theo tên & 1 số thứ tự thích hợp, tất cả các lần sử dụng của 1 mã định danh nhất định sẽ được đưa lại với nhau. “Các mục nhập định nghĩa” mà người dùng sẽ chỉ định xem 1 mã định danh có đại diện cho tên hàm, tham số hay biến có kích thước được gán các số thứ tự thấp để chúng xuất hiện đầu tiên trong số các mã thông báo có mã định danh nhất định; điều này giúp dễ dàng kiểm tra việc sử dụng xung đột & để phân bổ dung lượng lưu trữ cho các khai báo EQUIVALENCE. Thông tin thu thập được về mỗi mã định danh như vậy giờ đây được đính kèm vào mỗi mã thông báo; theo cách này, không cần phải duy trì “bảng ký hiệu” của các mã định danh trong bộ nhớ tốc độ cao. Các mã thông báo được cập nhật sau đó được sắp xếp theo 1 số thứ tự khác, về cơ bản là đưa chương trình nguồn trở lại thứ tự ban đầu của nó ngoại trừ lược đồ đánh số được thiết kế khéo léo để đưa các biểu thức số học vào dạng “tiền tố Ba Lan” thuận tiện hơn. Sắp xếp cũng được sử dụng trong các giai đoạn biên dịch sau này, để

tạo điều kiện tối ưu hóa vòng lặp, để hợp nhất các thông báo lỗi vào danh sách, v.v. Tóm lại, trình biên dịch được thiết kế sao cho hầu như mọi quá trình xử lý đều có thể được thực hiện tuần tự từ các tệp được lưu trữ trong bộ nhớ trống phụ, vì các số thứ tự thích hợp được đính kèm vào dữ liệu theo cách có thể sắp xếp dữ liệu thành nhiều cách sắp xếp thuận tiện khác nhau.

Computer manufacturers of 1960s estimated  $> 25\%$  of running time on their computers was spent on sorting, when all customers were taken into account. In fact, there were many installations in which task of sorting was responsible for  $> \frac{1}{2}$  of computing time. From these statistics we may conclude that either (i) there are many important applications of sorting, or (ii) many people sort when they shouldn't, or (iii) inefficient sorting algorithms have been in common use. Real truth probably involves all 3 of these possibilities, but in any event we can see: sorting is worthy of serious study, as a practical matter.

– Các nhà sản xuất máy tính của những năm 1960 ước tính  $> 25\%$  thời gian chạy trên máy tính của họ được dành cho việc sắp xếp, khi tất cả khách hàng đều được tính đến. Trên thực tế, có nhiều cài đặt trong đó nhiệm vụ sắp xếp chịu trách nhiệm cho  $> \frac{1}{2}$  thời gian tính toán. Từ các số liệu thống kê này, chúng ta có thể kết luận rằng (i) có nhiều ứng dụng quan trọng của việc sắp xếp, hoặc (ii) nhiều người sắp xếp khi họ không nên, hoặc (iii) các thuật toán sắp xếp không hiệu quả đã được sử dụng phổ biến. Sự thật thực sự có thể bao gồm cả 3 khả năng này, nhưng trong mọi trường hợp, chúng ta có thể thấy: sắp xếp đáng được nghiên cứu nghiêm túc, như 1 vấn đề thực tế.

Even if sorting were almost useless, there would be plenty of rewarding reasons for studying it anyway! Ingenious algorithms that have been discovered show: sorting is an extremely interesting topic to explore in its own right. Many fascinating unsolved problems remain in this area, as well as quite a few solved ones.

– Ngay cả khi sắp xếp gần như vô dụng, vẫn có rất nhiều lý do đáng để nghiên cứu nó! Các thuật toán khéo léo đã được phát hiện cho thấy: sắp xếp là 1 chủ đề cực kỳ thú vị để khám phá theo cách riêng của nó. Nhiều vấn đề chưa được giải quyết hấp dẫn vẫn còn trong lĩnh vực này, cũng như khá nhiều vấn đề đã được giải quyết.

From a broader perspectives we will find also: sorting algorithms make a valuable *case study* of how to attack computer programming problems in general. Many important principles of data structure manipulation will be illustrated in this chap. We will be examining evolution of various sorting techniques in an attempt to indicate how ideas were discovered in 1st place. By extrapolating this case study, can learn a good deal about strategies that help us design good algorithms for other computer problems.

– Từ góc nhìn rộng hơn, chúng ta cũng sẽ thấy: các thuật toán sắp xếp tạo ra 1 *case study* có giá trị về cách giải quyết các vấn đề lập trình máy tính nói chung. Nhiều nguyên tắc quan trọng về thao tác cấu trúc dữ liệu sẽ được minh họa trong chương này. Chúng ta sẽ xem xét sự tiến hóa của nhiều kỹ thuật sắp xếp khác nhau nhằm chỉ ra cách các ý tưởng được phát hiện ngay từ đầu. Bằng cách suy rộng nghiên cứu trường hợp này, có thể học được nhiều điều về các chiến lược giúp chúng ta thiết kế các thuật toán tốt cho các vấn đề máy tính khác.

Sorting techniques also provide excellent illustrations of general ideas involved in *analysis of algorithms* – ideas used to determine performance characteristics of algorithms so that an intelligent choice can be made between competing methods. Readers who are mathematically inclined will find quite a few instructive techniques in this chap for estimating speed of computer algorithms & for solving complicated recurrence relations. On other hand, material has been arranged so that readers without a mathematical bent can safely skip over these calculations.

– Các kỹ thuật sắp xếp cũng cung cấp những minh họa tuyệt vời về các ý tưởng chung liên quan đến *phân tích thuật toán* – các ý tưởng được sử dụng để xác định các đặc điểm hiệu suất của thuật toán để có thể đưa ra lựa chọn thông minh giữa các phương pháp cạnh tranh. Những độc giả có khuynh hướng toán học sẽ tìm thấy khá nhiều kỹ thuật hướng dẫn trong chương này để ước tính tốc độ của các thuật toán máy tính & để giải các mối quan hệ đệ quy phức tạp. Mặt khác, tài liệu đã được sắp xếp để những độc giả không có khuynh hướng toán học có thể bỏ qua các phép tính này 1 cách an toàn.

p. 5+++

- 5.1. Combinatorial Properties of Permutations.
- 5.2. Internal Sorting.
- 5.3. Optimum Sorting.
- 5.4. External Sorting.
- 5.5. Summary, History, & Bibliography.
- Chap. 6: Searching.
  - 6.1. Sequential Searching.
  - 6.2. Searching by Comparison of Keys.
  - 6.3. Digital Searching.
  - 6.4. Hashing.
  - 6.5. Retrieval on Secondary Keys.
- Appendix A: Tables of Numerical Quantities.
  - 1. Fundamental Constants (decimal).
  - 2. Fundamental Constants (octal).

- 3. Harmonic Numbers, Bernoulli Numbers, Fibonacci Numbers.

- Appendix B: Index to Notations.

- Appendix C: Index to Algorithms & Theorems.

**2.3** [Knu11]. DONALD ERWIN KNUTH. **The Art of Computer Programming. Vol. 4: Combinatorial Algorithms. Part 1. 1e**

**2.4** [Knu11]. DONALD ERWIN KNUTH. **The Art of Computer Programming. Vol. 4: Combinatorial Algorithms. Part 2. 1e**

### **3 Wikipedia's**

### **4 Miscellaneous**

### **Tài liệu**

[Cor+22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 4th edition. MIT Press, 2022, p. 1312.

[Knu11] Donald Ervin Knuth. *The Art of Computer Programming. Volume 4A: Combinatorial Algorithms. Part 1*. 2nd edition. Addison-Wesley Professional, 2011, p. 912.

[Knu98] Donald Ervin Knuth. *The Art of Computer Programming. Volume 3: Sorting & Searching*. 2nd edition. Addison-Wesley Professional, 1998, p. 800.