Teaching Algorithms

Ricardo A. Baeza-Yates

Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile
E-mail: rbaeza@dcc.uchile.cl

Abstract

In this paper we propose and discuss how to teach algorithms, including contents, methodologies, textbooks, and computer labs. We use the ACM/IEEE curricula as a starting point and compare our proposal to theirs. We raise several issues, but we do not provide definite answers. Our main proposal is a paradigm driven methodology for the main algorithmic course, as well as some paradigms and problems not usually covered. An ultimate teaching algorithm is still an open problem.

1 Introduction

Algorithms and data structures (just algorithms in the rest of the paper) are at the heart of computer science, being the building blocks of any software. At the same time, learning how to program goes side by side with algorithms. In this paper we address the issue of teaching algorithms, covering the different courses involved, their contents, their bibliography, etc. Although no specific textbook is recommended, some of them are compared and many others are referenced.

First computer science years are usually taught dividing contents by topics or type of problems. In algorithms this is also true, but a shift to contents based on paradigms or techniques seems to be more useful for future algorithmic challenges. This allows better design and also raises the importance of the analysis of algorithms.

We also address briefly the interaction between programming and algorithms. Programming paradigms are naturally tied to design techniques, and introductory computer science courses, are also usually introductory to algorithms. At the beginning, algorithms just need basic programming techniques. Later, when problem complexity and analysis become important, algorithms require discrete mathematics and theoretical foundations as automata theory and formal languages. At this stage, new software tools should be considered, such as algorithm animation or ad-hoc library classes.

Algorithms and data structures is one of the nine subject areas of the ACM/IEEE 1991 Computing Curricula [2]. We have used this proposal and [1] as a starting point and we compare our proposal to theirs. The ideas and suggestions given in this paper might be biased by our own experience in teaching algorithms, the curricula of our university and the fact that we do research on algorithms. A preliminary version of this work was presented as an invited paper in the IV Iberoamerican Congress on Computer Science Education [7].

2 Courses

There are five main goals, given in cronological order:

1. Knowledge of basic data structures and their associated algorithms.
2. Understanding of basic algorithmic design techniques.
3. Ability to apply mathematical tools to analyze algorithms.
4. Understanding of problem complexity classes.
5. Knowledge of advanced data structures, design and analysis techniques, and novel algorithmic areas.

Although all of these goals are important, the first two are completely necessary.

The approximate contents associated to the goals above, giving between parentheses the corresponding knowledge units (this subject is denoted by AL) of the ACM/IEEE curricula, are:

1. Data structures: arrays, queues, linked lists, search trees, graphs (AL1, AL2).
2. Searching and sorting algorithms (AL6).
3. Design techniques: induction, divide and conquer, dynamic programming, recursion, heuristics, randomization, use of finiteness[1] (AL3, AL8).
4. Analysis: recurrences, order notation, basic analysis tools, lower bounds (AL4).
5. Problem complexity: NP-complete problems and basic time and space complexity classes (AL5).
6. Advanced topics: parallel and distributed algorithms (AL9), probabilistic algorithms, approximation algorithms, multidimensional searching, complex analysis techniques, cryptography, computational geometry, string and computational biology algorithms, genetic algorithms.

The contents above are naturally divided in at least four courses, as follows:

IP Introduction to programming. This is the initial computer science course, and may include more than one semester. The content of this course is mainly programming techniques and learning a first programming language. However, that usually implies learning basic data structures (array, queues and lists), searching algorithms (sequential and binary search), and sorting algorithms (Quicksort, selection, insertion). This course mainly fulfills goal 1.

ADS Algorithms and data structures. The content should include all basic data structures in a formal way, by using abstract data types. Their use on efficient searching and sorting algorithms, with a first simple and approximate performance analysis. Concepts as program verification should be included here. Usually the course is divided by topics or problems. We say that is *topic* based. Goal 1 should be completed with this course as well as a first step towards goal 2.

[1] This is a personal paradigm for problems which makes use of the fact that the input is finite in some sense, for example bucketsort, digital trees, and Boyer-Moore string searching.

DAA Design and analysis of algorithms. This course should be the main formal course on algorithms, covering design paradigms, analysis techniques and problem complexity. Although traditionally this course is topic based, it should be *paradigm* driven. Topics covered include set manipulation, graphs, basic text searching and NP-completeness. Goals 2 to 4 should be fullfilled in this course.

SEM Advanced algorithms and data structures. This is either a survey course reserved for advanced topics and basically matches goal 5. The content should include some of the possible topics of goal 5 and could also be research oriented. Another possibility is to have a series of courses, where each topic can be covered in detail. More than one course is really needed to completely cover all the material.

The first three courses should be compulsory in any curricula. The last one must be optional or at a graduate level. In the appendix we propose a detailed content for the main algorithmic course using a paradigm driven approach. This is just a proposal as I have neever actually taught a course completely in this way. The course is suitable for three or four months per term, whith three hours of formal lectures and one or two hours of closed labs per week.

3 Discussion

3.1 Prerequisites

The first course could be used not only for a CS curricula, but also for engineering or other fields needing a CS introductory course. The second course should be a second year course, at least, after the first course. At the same level, two related courses should exist:

- Discrete Mathematics. This course should cover CS oriented mathematics, including sets, combinatorics, logic, recurrences, binary relations and functions, algebras, graph theory, matrices, probability, etc. See [20, 19] for nice approaches to this subject.

- Foundations of CS. This course should cover automata theory, formal languages, problem complexity, decidability. A first glance to NP-completeness could be given here, leaving the algorithmic side of it to the DAA course.

These two courses are compulsory requirements for the DAA course. This third course should be taught at least at a third year level, because a good degree of maturity is needed. More advanced courses could have specific prerequisites. For example, a course on parallel or distributed algorithms may need some notions of computer architecture or operating systems.

3.2 Teaching Methodology

Along the two first courses, the emphasis should drift from teaching programming techniques to teaching algorithmic techniques. In other words, from low level constructs to high level pseudocode. This drift possible implies a change of programming language or the use of other tools. One immediate way to achieve this is to advance from a typical programming language to an object oriented language (this is discussed later). An interesting reference on how to teach is [21].

Traditionally, covering the contents of the courses given is guided by data structures (e.g., trees, graphs) or problems (e.g., searching, sorting). Although this is natural for introductory courses, it is not the best way to cover goals 2 and 3 (usually analytical techniques are related to design techniques). For that reason, a better alternative is to divide the contents by the different paradigms for algorithm design. Most textbooks have a section which outlines these paradigms, but not too many of them use this approach. We think that a paradigm driven material for the DAA course could be more useful to really grasp the essence of algorithmic design. For this task, good toy problems are essential [6]. In this respect, there are several recent results that should be included. Between them, we can mention on-line algorithms to understand optimality trade-offs and skip lists or perfect hashing as simple examples of ramdomization.

A last issue is computer laboratories. Closed labs should be approximately one third of the total lecture time of the course. On the other hand, open labs (assignments) should also be compulsory, requiring at least another one third of the course. Both, conceptual and experimental assignments are needed. The first to reinforce the course contents itself. The later are useful to compare algorithms and to emphasize implementation aspects. For example, hybrid algorithms that for small inputs use simple techniques or algorithms that depend on the input distribution. Another kind of computer based assignments is related to complementary software tools that can help the student and also the teacher. Among them we have to mention algorithm animation tools (e.g. Balsa) to ease understanding; and symbolic algebra systems (e.g. Maple or Mathematica) to ease the analysis of algorithms. We refer the reader to [1] for other issues regarding computer labs.

3.3 Textbooks

Most algorithm textbooks are designed for the DAA course, and there are plenty of them. Most of them are very reasonable, so it is not easy to choose just one. We can distinguish classic textbooks as [3], comprehensive ones as [11], or enlightening ones as [32, 42]. Some of them provide animated versions, for example [11] is available in CD-ROM, but only for MacIntosh. An extensive list of textbooks is given in the bibliography. We can use classic or comprehensive books to cover contents. But we suggest new books to capture the beauty of algorithms. For example, Manber's book [32] follows consistently a single design paradigm: induction. On the other hand, to understand algorithm analysis, Rawlins book [42] does a wonderful job. We can divide classical textbooks by their orientation. Some are oriented to concepts (for example [5, 9, 54]) while others to analysis (for example [8, 40, 43]). Brassard and Bratley's textbook [9], to the best of our knowledge, is the only one that attempts a paradigm driven approach. The same does Parberry's book [37]. However, this is a book just on problems.

Textbooks for the IP course are usually introductory programming books, and there are many of them. For this reason, and because they depend on the choice of a first programming language, we do not discuss them here. Most of them are not algorithmic oriented (one exception is [22]). Some algorithms books are given at a lower level, and because of this are suitable for the ADS course [4, 45, 54]. There are also many data structures oriented books, for example [31].

For seminar type courses, specialized books, according to the context, should be chosen. To cover advanced algorithm analysis techniques we suggest [19, 53, 13], especially the first. For specific topics we can mention: parallel algorithms [29, 41, 30, 14, 48, 16], distributed algorithms [51], computational geometry [34, 36, 38, 44], text algorithms [15, 12], randomized algorithms [35], cryptography [50, 47], and genetic algorithms [17]. A good general reference book, which in this sense is unique, is [18]. Knuth's books are also useful [25, 26].

3.4 Computer Requirements

The first two courses need basic computer resources. They could range from personal computers to workstations. The DAA course needs better laboratories, if possible. For example, practical classes could be done with audiovisual techniques with the help of the algorithm animation tools already mentioned. An specific experience using Balsa is described in [10]. Other useful visual tool is a visual debugger. For a complete discussion of software visualization we suggest [39].

A difficult issue is the choice of a programming language. We will not discuss what should be the first programming language, as there is no consensus today about this (typical recent choices are Modula, Scheme or Turing). However, we think that an imperative language is better for the classical design techniques. One reason is that most of the developing of algorithms, and subsequently many techniques, have been done with imperative languages. Another reason is that these languages allows to be closer to the real machine, being easier to model and understand the time or space used by an algorithm (for example, this is difficult to do with functional languages).

The second course can move to a system programming language, being C or C++ typical choices (this is just a practical decision as the right language can be a metter of religion!). We strongly suggest to use an object oriented language for the DAA course, because it eases the implementation of abstract data types. The best choice might be C++, which can be complemented with public domain library classes that include all the basic data types and algorithms. Among those libraries we can recommend LEDA[2].

4 Comparison with the ACM/IEEE Curricula

In the ACM/IEEE Curricula, the subject is divided in nine knowledge units for approximately 47 lecture hours. An important difference in our proposal is that we do not include unit AL7 (Computability and Undecidability), which we think should be taught in the Foundations of CS course together with NP-complete problems, leaving the problem of how to solve hard problems to the DAA course. Note that we could say that the Foundations of CS course is part of the area of algorithms, but we think it is much more. This course is the conceptual base not only for algorithms, but also for other areas. Another difference is that only parallel and distributed algorithms are included in the advanced part, as some of the other topics could be considered as problem specialized algorithms. However, concepts as probabilistic or approximation algorithms are generic, and also some areas like computational geometry are now important on their own. Nevertheless, we cover most of the topics suggested in [1, 2].

In our proposal, we have 80 hours of formal lectures. This is because some of the material is repeated on more than one course, but the emphasis in each case is different: first is descriptive, then is design oriented, and later focused on analysis and efficiency. For example, the data structures and algorithm analysis course proposed by ACM/IEEE is very similar to our DAA proposal in contents, but different in the methodology if we use a paradigm driven approach.

[2] Available via ftp from `sbsvax.cs.uni-sb.de:/pub/LEDA/LEDA-2.0.1.tar.Z`

In [1] each subject is divided in three areas: theory, abstraction and design. Algorithms are perhaps the best discipline to show the difference and importance of these three areas. Also, twelve recurring concepts are mentioned. From those, the most important ones in this case are conceptual and formal models, efficiency and trade-offs. On the other hand we also stress abstraction and design along the courses proposed, rather than just contents.

Acknowledgement

We thank the helpful comments of Gonzalo Navarro and Nivio Ziviani.

A Design and Analysis of Algorithms

The goals of this course is to design and analyze the efficiency of several algorithms, mainly for non-numerical problems. This course can be improved by the use of algorithm animation software and algorithmic library classes. The contents are:

- Algorithm complexity: time and space. Worst and average case. Machine models. Order notation. Techniques to solve recurrences, generating functions.
- Techniques for algorithm design: induction, divide and conquer, dynamic programming, greedy heuristics, randomization, use of finiteness.
- Sorting and selection: mergesort, quicksort, bucketsort. Selecting an element in linear time. Priority queues.
- Searching: interpolation search, optimal search trees, digital trees (tries and Patricia trees), hashing. Secondary memory: B-trees and linear hashing.
- Graph algorithms: spanning trees, transitive closure, minimal path algorithms.
- Text searching: classical string searching algorithms, automata searching.
- NP-Complete problems: non-deterministic Turing machines. Classes P and NP. The SAT problem. Other NP-complete problems. Simple heuristic approximation algorithms.
- Notions of parallel and distributed algorithms.

The contents above can be arranged to be paradigm driven as follows (partly based in [37]), using an optional miscelaneous final session for problems not included here:

- Algorithm complexity: time and space. Worst and average case. Machine models. Order notation. Techniques to solve recurrences, generating functions. Lower bound techniques.
- Induction: simple examples. Other paradigms as particular cases of induction.

- Divide and conquer: quicksort, mergesort, selection algorithms, Strassen's algorithm, search trees and other recursive data structures.
- Dynamic programming: simple examples, optimal binary search trees, Floyd's algorithm, longest common subsequence.
- Randomization. Pseudo-randomizing the input: hashing. Randomized algorithms: skip lists, perfect hashing. Random output: probabilistic algorithms.
- Use of finiteness: interpolation search, bucketsort, digital trees, Boyer-Moore type string searching algorithms.
- Heuristics: splay trees. Greedy algorithms: Dijkstra's algorithm, min-cost spanning trees. Solving NP-complete problems: exhaustive search, approximation algorithms.

Lecture hours: 40. Closed lab hours: 20. Suggested bibliography: [11, 32, 42, 37]

References

[1] ACM Task Force on the Core of Computer Science, "Computing as a Discipline", *Communications of the ACM* 32, 1989, 9–23.

[2] ACM/IEEE Joint Curriculum Task Force, "Summary of the Computing Curricula 1991", *Communications of the ACM* 34, 1991, 68–84.

[3] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

[4] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "Data Structures and Algorithms", Addison-Wesley, 1984.

[5] S. Baase, "Computer Algorithms: Introduction to Design and Analysis", Addison-Wesley 1988.

[6] R. Baeza-Yates, "Searching: An Algorithmic Tour", Technical Report, Dept. of Computer Science, Univ. of Chile, 1995.

[7] R. Baeza-Yates, "Teaching Algorithms", IV Iberoamerican Congress on Computer Science Education, Canela, Brazil, July 1995.

[8] L. Banachowski, A. Kreczmar and W. Rytter, "Analysis of Algorithms and Data Structures", Addison-Wesley, 1991.

[9] G. Brassard, and P. Bratley, "Algorithmics: Theory and Practice", Prentice-Hall, 1988. Second edition, 1995.

[10] M. Brown and R. Sedgewick, "Progress report: Brown university instructional computing laboratory", *ACM SIGCSE Bulletin*, 16, 1984, 91–101.

[11] T. Cormen, C. Leiserson and R. Rivest, "Introduction to Algorithms", MIT Press, 1991.

[12] M. Crochemore and W. Rytter, "Text Algorithms", Oxford Press, 1994.

[13] J.S. Vitter and P. Flajolet", "Average-case analysis of Algorithms and Data Structures", Handbook of Theoretical Computer Science (volume A), Elsevier and MIT Press, 1990, 431–524.

[14] T.J. Fountain, "Parallel Computing: Principles and Practice", Cambridge Univ. Press, 1995.

[15] W. Frakes and R. Baeza-Yates, editors. "Information Retrieval: Data Structures and Algorithms", Prentice-Hall, 1992.

[16] A. Gibbons and W. Rytter, "Efficient Parallel Algorithms", Cambridge University Press, 1988.

[17] D. Goldberg, "Genetic Algorithms", Addison-Wesley, 1989.

[18] G. Gonnet and R. Baeza-Yates, "Handbook of Algorithms and Data Structures", Addison-Wesley, 2ed, 1991.

[19] R.L. Graham, D.E. Knuth and O.Patashnik, "Concrete Mathematics: A Foundation for Computer Science", Addison-Wesley, 1988.

[20] D. Gries and F.B. Schneider, "A Logical Approach to Discrete Math", Springer-Verlag, 1993.

[21] D. Gries, "Teaching Calculation and Discrimination: A More Effective Curriculum", *Communications of the ACM* 34, 1991, 45–55.

[22] D. Harel, "Algorithmics: The spirit of computing", Addison-Wesley 1987.

[23] E. Horowitz and S. Sahni, "Fundamental of Computer Algorithms", Computer Science Press, 1982.

[24] T.C. Hu, "Combinatorial Algorithms", Addison-Wesley, 1982.

[25] D.E. Knuth, "The Art of Computer Programming", vol. 1, "Fundamental Algorithms", Addison-Wesley, 2nd ed., 1973.

[26] D.E. Knuth, "The Art of Computer Programming", Vol. 3, "Sorting and Searching", Addison-Wesley, 1973.

[27] D.C. Kozen, "The Design and Analysis of Algorithms", Springer-Verlag, 1992.

[28] L. Kronsjo, "Algorithms: Their complexity and efficiency", John Wiley, 1979.

[29] J. JáJá, "An Introduction to Parallel Algorithms", Addison-Wesley, 1992.

[30] F.T. Leighton, "Intr. to Parallel Algorithms and Architectures", Morgan Kaufmann, 1992.

[31] H.R. Lewis and L. Denenberg, "Data Structures and Their Algorithms", Harper Collins Publishers, 1991.

[32] U. Manber, "Introduction to Algorithms: A Creative approach", Addison-Wesley, 1989.

[33] K. Mehlhorn, "Data Structures and Algorithms, vol. I: Sorting and Searching", Springer-Verlag, 1984.

[34] K. Mehlhorn, "Data Structures and Algorithms, vol. III: Multidimensional Searching and Computational Geometry", Springer-Verlag, 1984.

[35] R. Motwani and P. Raghavan, "Randomized Algorithms", Cambridge University Press, 1995.

[36] J. Nievergelt and K. Hinrichs, "Algorithms and Data Structures with Applications to Graphics and Geometry", Prentice-Hall, 1993.

[37] I. Parberry, "Problems on Algorithms", Prentice-Hall, 1995.

[38] F.P. Preparata and M.I. Shamos. "Computational Geometry: An Introduction", Springer Verlag, 1985.

[39] Blaine A. Price, Ronald M. Beacker and Ian S. Small, A Principled Taxonomy of Software Visualization, *Journal of Visual Languages and Computing* 4, 211–266, 1993.

[40] P. Purdom and C. Brown, "The Analysis of Algorithms", Holt Reinhart and Winston, 1985.

[41] M. J. Quinn, "Parallel Computing: Theory and Practice", McGraw-Hill, 1994.

[42] G. Rawlins, "Compared to What? An Introduction to Analysis of Algorithms", Computer Science Press, 1991.

[43] E.M. Reingold, J. Nievergelt and N. Deo, "Combinatorial Algorithms: Theory and Practice", Prentice-Hall, 1977.

[44] J. Rourke, "Computational Geometry in C", Cambridge University Press, 1994.

[45] R. Sedgewick, "Algorithms", Addison Wesley, 1987. Versions for other languages, including C++ are also available.

[46] B. Salzberg, "File Structures: An Analytic Approach", Prentice-Hall, 1988.

[47] D. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd edition, John Wiley, 1995.

[48] D. Skillicorn, "The Foundations of Parallel Computing", Cambridge Univ. Press, 1995.

[49] S. Smith, "Design and Analysis of Algorithms", PWS-Kent, 1989.

[50] D.R. Stinson, "Cryptography Theory and Practice", Chemicla Rubber Corporation Press, 1995.

[51] G. Tel, "Introduction to Distributed Algorithms", Cambridge Univ. Press, 1995.

[52] M. Weiss, "Data Structures and Algorithm Analysis", Benjamin-Cummings, 1992.

[53] H. Wilf, "Algorithms and Complexity", Prentice-Hall, 1986.

[54] N. Wirth, "Algorithms and Data Structures", Prentice-Hall, 1986.