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. 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.
2 Courses

There are five main goals, given in chronological 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 (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 (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.
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 also paradigm driven. Topics covered include set manipulation, graphs, basic text searching and NP-completeness. Goals 2 to 4 should be fulfilled 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 more detailed content for each of these courses, including the approximate minimal lecture time for each of them. They are suitable for three or four months per term, with 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 introduction course. The second course should be a second year course, at least, after the first course. At this same level, two related courses should exist:

- Discrete Mathematics. This course should cover CS mathematics, including sets, combinatorics, logic, recurrences, binary relations and functions, algebras, graph theory, matrices, probability, etc. See [15] for a nice approach 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. The complete layout of the courses is given in Figure 1.

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 pseudo-code. This drift possible implies a change of programming language or the use of other tools. One
An 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 [16].

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. 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 randomization.

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 [9], or enlightening ones as [26, 32]. Some of them provided animated versions, for example [9] 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 [26] follows consistently a single design paradigm: induction. On the other hand, to understand algorithm analysis, Rawlins book [32] does a wonderful job. We can divide textbooks by their orientation. Some are oriented to concepts (for example [5, 7, 40]) while others to analysis (for example [6, 31, 33]).

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 [17]). Some algorithms books are given at a lower level, and because of this are suitable for the ADS course [4, 35, 40]. There are also many data structures oriented books, for example [25].

For seminar type courses, specialized books, according to the context, should be chosen. To cover advanced algorithm analysis techniques we suggest [14, 39], especially the first. For specific topics we can mention: parallel algorithms [24], computational geometry [28, 29, 30, 34], text algorithms [11, 10], and genetic algorithms [12]. A good general reference book, which in this sense is unique, is [13]. Knuth's books are also useful [20, 21].

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 [8].

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. 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\(^1\).

\(^1\)Available via ftp from sbsdav.cs.uni-sb.de:/pub/LEDA/LEDA-2.0.1.tar.Z
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. In our proposal, we have 62 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.

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.

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. One difference is that we do not cover all the contents suggested for theory, because we think that this should be covered in the CS theory courses. 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. On the other hand we stress abstraction and design along the courses proposed, rather than just contents. Nevertheless, we cover most of the topics suggested in [1, 2].

Acknowledgement

We thank the helpful comments of Gonzalo Navarro.

A Introduction to Programming

We include here only the contents related to algorithms. This course should start developing algorithmic thinking. The basic contents are:

- Arrays in one and two dimensions. Sequential and binary search.
- First notions about design of efficient algorithms. Basic sorting algorithms: insertion and selection sort. Recursion and Quicksort.
- Dynamic data structures: linked lists, trees, graphs. Memory management algorithms.
- Abstract data types: stacks, queues, priority queues, search trees, hashing.

B Algorithms and Data Structures

The goals of this course are the study of algorithm design techniques, and how they are used to develop programs. At the end the student should know the main algorithms and data structures, including an informal analysis of their performance. The first part of the course includes programming techniques. The specific contents includes:

- First notion of analysis of algorithms. Survey of abstract data types: arrays, lists, stacks, queues, trees.
- The dictionary problem: skip lists, binary search trees, balanced search trees (AVL, B-trees), splay trees, digital trees, hashing.
- Sorting: lower bounds for comparison algorithms, Quicksort, Heapsort, Bucketsort.
- Graphs: representation, traversals, minimal spanning trees.

Lecture hours: 20. Closed lab hours: 10. Suggested Bibliography: [4, 26, 35].

C 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:

- Techniques for algorithm design: induction, divide and conquer, dynamic programming, greedy heuristics, randomization.
- Set manipulation and operations. Union-find problem.
- Graph algorithms: spanning trees, transitive closure, minimal path algorithms.
- Text searching: automata searching, classical string searching algorithms, regular expression algorithms.
- Notions of parallel and distributed algorithms.

Lecture hours: 30. Closed lab hours: 15. Suggested bibliography: [9, 26, 32]
D Advanced Data Structures and Algorithms

This is a survey of advanced topics. Each part could be a course on its own. The contents are:

- Advanced analysis tools.
- New data structures, including secondary memory structures.
- Parallel and distributed algorithms.
- Notions of cryptography, probabilistic algorithms, text algorithms and computational geometry.

Lecture hours: 45. Only a final project and open labs are needed. Suggested bibliography: [14, 36, 24, 30, 11, 10]

References


