THE FIRST COURSE IN COMPUTER SCIENCE NEEDS A MATHEMATICS COREQUISITE

First-year computer science students need to know and use a considerable amount of mathematics. A corequired course in discrete mathematics is a good solution.

ANTHONY RALSTON

What role should mathematics play in an undergraduate computer science curriculum? After years of debate there now seems to be this consensus among academic computer scientists:

1. Undergraduate computer science programs need more mathematical content, which needs to be more closely integrated with computer science courses than is implied by Curriculum 78 [1].
2. Traditional undergraduate mathematics (i.e., the calculus sequence) is not generally supportive of the computer science curriculum. However, many areas in discrete mathematics are supportive (see, for example, [4]).

The consensus does not necessarily extend to a belief that the very first college course in computing needs mathematics support. It is the thesis of this paper that such a first course does need mathematics support. My evidence for this is based on recent experience teaching the one-year Introduction to Computer Science course at the State University of New York at Buffalo.

Exactly what is a "first course in computer science"? While American colleges and universities present a wide range of material under this heading, two fairly distinct models emerge:

Course 1
This is a one-semester course with the essential purpose of teaching the student to program in a high-level language. This course may include some content related to programming methodology (e.g., structured programming) but seldom includes any other computer science topics. The aim of this course is simply to make the student reasonably proficient in programming in the one language.

© 1984 ACM 0001-0782/84/1000-1002 75¢

Course 2
This is a one- or two-semester course in which the student is taught to program in one or more high-level languages, and in which the student generally does as much or more actual programming each semester than in Course 1, but where learning languages and programming in them are not the main purpose of the course. Rather, the programming languages are a vehicle for introducing the student to the discipline of computer science; the emphasis is on algorithms (and their analysis and even verification), programming methodology, data structures (particularly in the two-semester sequence), and so on. The descriptions of CS1 and CS2 in Curriculum 78 are very much in the spirit of this model, although they specify no mathematics prerequisite or corequisite. The syllabus of the new Advanced Placement Computer Science course for high schools is also very much in this spirit [6].

Whether a particular college or university uses one or the other of these models depends more on the objectives of the course than on the type or quality of the school. If the first course is intended to give students a skill they will need in their undergraduate careers, it is likely to resemble Course 1; if it is intended to be analogous to first courses in other disciplines, it is likely to resemble Course 2. With only a modicum of bias, it can be said that the former motivation smacks of the old slide rule course that engineers used to take. In any case, Course 1 tends to give all students, including potential computer science majors, the unfortunate impression that Computer Science = Programming.

Since my purpose here is not really polemical, let me emphasize my clear preference for Course 2 and note that it is about time computer scientists viewed the introductory sequence in their discipline the way sci-

entists in other disciplines do: as an overall introduction to the field and not just as vocational training in a particular aspect of it.

The one-year introductory computer science sequence at SUNY at Buffalo approximates Course 2. The language of instruction for both semesters is Pascal. The first semester covers the main features of Pascal and introduces the student to algorithms and programming methodology. The second semester covers the rest of Pascal and emphasizes data structures and particular application areas such as sorting. (Typically, separate textbooks are used for the Pascal and computer science portions of the courses.)

MATHEMATICS NEEDED TO SUPPORT THE FIRST COURSE

The list on page 1004 presents the mathematics topics that I would like students to know by the time they are needed in a one-year computer science sequence, matched with related topics in computer science for which the mathematics would be useful. Different instructors will cover different topics in computer science; nevertheless, the range of topics included in the list should correspond fairly well to the material that would be presented in a Course 2 situation. I compiled this list by going back over my lecture notes for the first-year course and listing the mathematics topics I had discussed or at least mentioned. The list of topics, then, is not a syllabus for a discrete mathematics course, although it could serve as the basis for one. Indeed, the list bears some resemblance to a syllabus for a typical discrete mathematics course for sophomore or junior computer science students. There is, however, less of an abstract algebraic flavor than is usually found in such courses.

It is important that the level of presentation to freshmen be substantially lower than is common in discrete mathematics courses for computer scientists. Roughly two-thirds of the topics listed are justified by their importance in the analysis of simple, basic algorithms. (Readers might be interested to compare this list of mathematics topics with Knuth's from Chapter 1 of [2]). For example, when discussing the Tower of Hanoi problem as an example of a simple recursive algorithm, it is desirable also to do an analysis of the number of moves as a function of the height; similarly, in discussing Quicksort and bubble sort, one should demonstrate why one is a good algorithm and the other a poor algorithm.

The mathematics required for the first of these examples is trivial; that required for the second is not. Indeed, the full analysis of Quicksort requires a knowledge of the limiting behavior of the harmonic numbers, a topic beyond the reach of most college freshmen. But even if a rigorous treatment of the relevant mathematics is beyond the student's current mathematical understanding, this material can be presented intuitively.

The mathematics topics listed that are not directly needed for the analysis of algorithms have to do either with notation (e.g., summation notation) or with basic definitions (e.g., the absolute value function). However good their high-school mathematics preparation may have been, students entering college seldom really know most of this material, and need reinforcement. Thus, a mathematics course in which they learn (or relearn) this material in the context of a full discrete mathematics syllabus gives students valuable knowledge for the first course in computer science. Computer science instructors should—must—be able to use such elementary mathematics without worrying about how many students have ever seen it before. At times, then, instructors will have to review the mathematics in the computer science course; this is a better alternative, however, than having to introduce it.

It is about time computer scientists viewed the introductory sequence in their discipline as an overall introduction to the field and not just as vocational training in a particular aspect of it.

If students are not familiar with the mathematics listed, there are at least three available alternatives, none of which is very satisfactory. The first is to interpolate the mathematics as part of the computer science course. This has at least one and possibly both of the following drawbacks: The time available for the computer science subject matter is reduced, and the treatment accorded the mathematics is usually cursory. The almost inevitable result is an insufficient understanding of the mathematics.

One possible counterargument to this alternative is that students will be more motivated to learn mathematics in a computer science course than they would be in a separate course. My feeling, however, is that students see mathematics introduced in a computer science course as an unwelcome intrusion on or diversion from the main subject matter. Moreover, if students see mathematics learned in a separate course applied in the first computer science course, they are more likely to believe in the importance of mathematics in computer science than they would be from anything an instructor might tell them in the computer science course.

A second alternative to a separate mathematics course is handwaving, that is, providing brief intuitive discussions of relevant mathematical information during the presentation of computer science material. This solution will not detract excessively from the main subject matter, but it is likely to convince students that the mathematical material is of minor importance and to confuse at least as many students as it enlightens.

A third—and extreme—alternative is to avoid mathematics altogether. In the context of this discussion, however, this is really no solution at all: To exclude mathematics is essentially to transform Course 2 into Course 1.

MATHEMATICS NEEDED TO SUPPORT THE FIRST COURSE IN COMPUTER SCIENCE

Mathematics Topic	Computer Science Applications
ELEMENTARY MATHEMATICS	
Summation (and product) notation	Useful in many applications, but especially in the analysis of elementary algorithms
Subscripts	Useful wherever summation notation is needed; with arrays
Specific simple functions, esp. absolute value, truncation, trigonometric	Useful in many applications, specifically in relation to the intrinsic functions in any language
Logarithms	In the analysis of most algorithms (e.g., sorting) and in many order-of-magnitude arguments
Prime numbers	A standard computer science algorithm and program
Greatest common divisor	The Euclidean algorithm is another standard computer science application
Floor and ceiling functions	Pervasive in computer science
GENERAL MATHEMATICAL IDEAS	
Functions	In the idea of a function as a type of subprogram; more generally, as a correspondence or mapping in any analysis of algorithms
Sets (and operations on sets)	In the idea of a set data type, but also in many arguments dealing with algorithms and their analysis
ALGEBRA	
Matrix algebra	In many uses of arrays, for example, in the computation of the path matrix of a graph
Polish notation	In any discussion of the compilation of arithmetic expressions, as well as in more general discussions of infix, postfix, and prefix notation
Congruences	In an elementary discussion of random number generators; notation also useful in GCD discussions
SUMMATION AND LIMITS	
The elementary summation calculus	Summation of the first n integers arises in many contexts
Order notation (and thus limits of sequences)	Essentially all algorithm analysis requires the use and understanding of $O(f(n))$
Harmonic numbers	In various areas, for instance, in the analysis of Quicksort
NUMBERS AND NUMBER SYSTEMS	
Positional notation	This is obviously a high school (if not an elementary school) topic, but many college students don't really understand it; any discussion of computer arithmetic requires an understanding of this topic
Nondecimal bases and base conversion	Obviously, binary and other bases are essential to an understanding of computer science; base conversion algorithms can help a student to understand this topic
LOGIC AND BOOLEAN ALGEBRA	
Boolean operators and expressions	The obvious use is with Boolean expressions in programming languages, but these are also useful in elementary discussions of logic design
Basic mathematical logic	Useful in a variety of contexts and necessary in any discussion of algorithm verification
PROBABILITY	
Basic concepts of sample spaces and laws of probability	Almost all analysis of algorithms depends on assumptions of equally probable data sets
COMBINATORICS	
Permutations, combinations, and counting	Needed in the analysis of most algorithms and especially in many important algorithms (e.g., the traveling salesperson problem)
Binominal coefficients and theorem	Very common in discrete mathematical analysis
GRAPH THEORY	
Basic concepts	Useful in discussing graphs as data structures and, more specifically, in networking problems
Trees	A knowledge of the properties of trees as mathematical objects will support an understanding of their applications in computer science
DIFFERENCE EQUATIONS AND RECURRENCE RELATIONS	
Solution of simple difference equations	Useful in such problems as the Tower of Hanoi and Quicksort
Generating functions	A necessary tool in the analysis of algorithms

My conclusion is that the best solution is a one-year course in discrete mathematics pitched at an intellectual level equivalent to that of an introductory calculus course. This course should be a prerequisite or a corequisite to the first computer science course. The two-semester mathematics sequence should cover most, but probably not all, of the topics presented on page 1004.

Since there is as yet so little experience with such courses, we can expect their content and organization to evolve as instructors gain experience. For computer science students, this course can—and probably should—precede the college calculus sequence (see [3] and [5]).

There is no reason why discrete mathematics should not be taught by mathematics departments or why it should be geared for computer science students only. Mathematics majors, social science majors, and even physical science and engineering majors could all benefit from a familiarity with discrete mathematics. In establishing a syllabus for such a course, instructors should consider whether a given discrete mathematics topic is useful in a variety of applications or whether it is of interest almost exclusively to computer science students (in which case it should probably be excluded: Polish notation and much of the material on trees would fall into this category, but basic graph theory would not).

This does not imply that no new mathematics should be introduced in a first course in computer science. There should be at most a small amount, which can be more easily absorbed by students who have taken or are taking a discrete mathematics course. Similarly, physical scientists and engineers are going to have to rethink the demands they have been making on mathematics departments. The first two years of college mathematics have been geared for physics, chemistry, and engineering students. Mathematics departments were expected, in the first two years, to teach virtually all the mathematics that students in these fields would need. But with the inevitable squeeze of three or four terms of calculus into as little as two as discrete mathematics becomes a standard component of the first two years of college mathematics, material that is of general relevance to many students must take precedence over material that is only relevant to specific disciplines in the calculus sequence. Faculty in the physical sciences and engineering will have to include the mathematics specifically of interest just to them in their own courses.

The title of this paper stipulates a mathematics corequisite rather than a prerequisite. This is because a well-structured and well-taught discrete mathematics course will emphasize an algorithmic approach. This, in turn, means that students will benefit considerably from being able to implement their algorithms on a computer and will as a result be able to study them more intensively. This is a rare instance of two courses serving to reinforce each other.

A critic of a draft of this article argued that the discrete mathematics course should be a prerequisite for the *second* computer science course because it is so difficult for a student to fit two corequired courses into a schedule. Fair enough; but students frequently do take both a mathematics and a computer science course in both semesters of their freshman year.

Introducing more mathematics-related topics into the first course in computer science not only will enhance the direct value of that course, it will give students education of lasting value.

All of us who teach computer science to college students pay lip service to the idea that we should *educate* those students and not just prepare them for high-paying jobs. We know that our discipline is changing so rapidly that little of what we teach today will be truly relevant even ten years from now. One way to act on this knowledge is to emphasize, as far as we are prescient enough to do so, the topics that will serve our students in the widest variety of possible futures. A sound grounding in mathematics is a time-honored way of providing the education that encourages professional growth, rather than the training that inhibits it. Introducing more mathematics-related topics into the first course in computer science not only will enhance the direct value of that course, it will give students education of lasting value.

Acknowledgment. I would like to thank Stephen Garland, Stephen B. Maurer, Edwin D. Reilly, Jr., Mary Shaw, and Aaron Tenenbaum for valuable comments on and criticism of a draft of this paper.

REFERENCES

1. Curriculum Committee on Computer Science. Curriculum 78—Recommendations for the undergraduate program in computer science. *Commun. ACM 22*, 3 (Mar. 1979), 147–166.
2. Knuth, D.E. *The Art of Computer Programming.* Vol. 1, 2nd ed. Addison-Wesley, Reading, Mass., 1973.
3. Ralston, A. Computer science, mathematics and the undergraduate curricula in both. *Am. Math. Monthly 88*, 7 (1981), 455–472.
4. Ralston, A., and Shaw, M. Curriculum 78—Is computer science really that unmathematical? *Commun. ACM 23*, 2 (Feb. 1981), 67–70.
5. Ralston, A., and Young, G.S., Eds. *The Future of College Mathematics.* Springer-Verlag, New York, 1983.
6. The College Board. *Advanced Placement Course Description: Computer Science.* The College Board, Princeton, N.J., 1983.

CR Categories and Subject Descriptors: G.2.0 [**Discrete Mathematics**]: General; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*
General Terms: None
Additional Key Words and Phrases: First course in computer science

Author's Present Address: Anthony Ralston, Dept. of Computer Science, SUNY at Buffalo, 226 Bell Hall, Buffalo, NY 14260.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.