Computational Fluency, Algorithms, and Mathematical Proficiency: One Mathematician's Perspective

Hyman Bass

In recent years, few aspects of mathematics education have been as much discussed and debated as the notions of computational fluency and algorithms. A National Research Council report, *Adding It Up: Helping Children Learn Mathematics* (Kilpatrick, Swafford, and Findell 2001), offers an image of what it means to have skill with mathematics, or *mathematical proficiency.* This concept is helpful for moving beyond these debates. Mathematical proficiency includes five components: conceptual understanding, procedural fluency, strategic competence, adaptive reasoning, and productive disposition (Kilpatrick, Swafford, and Findell 2001, p. 116). That these components are not separate but fundamentally intertwined is important to note. This article illustrates some of the ways in which the goal of computational fluency and an appreciation of mathematical algorithms are related to this larger concept of mathematical proficiency.

Computation, Fluency, and Algorithms

What do we mean by *computation*? Suppose a colleague asks you what 3.29×17.6 is. You could simply say, "It is 3.29 times 17.6." This statement is true, but clearly not what your colleague is asking for. Your colleague means to ask, "What is the result when you compute the expression 3.29×17.6?" To "compute" a value means to find a standard representation of the numerical value of a mathematical expression—in this case, the decimal representation.

The representation depends on the context and may need to be specified because the standard representation can take a variety of forms. For example, when computing $1\ 3/4 \div 1/2$, you may choose to represent the value as 7/2 (a fraction), 3 1/2 (a mixed number), or 3.5 (a decimal). Your choice will depend on how you want to use the result.

Computation is a particular form of mathematical problem solving. Typically, but not exclusively, it involves the performance of basic numerical operations ($+, -, \times, \div$) on some class of numbers. ("Computing" and "calculating" are used interchangeably here in their full mathematical sense. The fact that computing may engage significant problem-solving skills can easily be revealed when computing prime factorizations of greater numbers, areas and volumes, integrals, and so on.)

Considering what is meant by the term *fluency* is a little more complicated. According to *Principles and Standards for School Mathematics*, "Fluency refers to having efficient, accurate, and generalizable methods (algorithms) for computing that are

Hyman Bass, hybass@umich.edu, is a professor of mathematics and mathematics education at the University of Michigan. His mathematical research is in algebra. In mathematics education, he collaborates with Deborah Ball to investigate the mathematical knowledge that the teaching of mathematics entails.

This article owes many of its ideas, examples, and exposition to discussions with Deborah Ball, to whom the author expresses his indebtedness.

based on well-understood properties and number relationships" (NCTM 2000, p. 144). Susan Jo Russell (2000) provides a synopsis of the concept of computational fluency represented in *Principles and Standards for School Mathematics*. She cites the document as making central three characteristics: *efficiency* (not getting lost or bogged down), *accuracy* (being careful and keeping good records), and *flexibility*. Note that these terms are used to describe qualities of the *person* doing the computing, not properties of an algorithm, as I use the terms in this article.

Seeking a single, clearly described, generic solution method that works in all cases makes sense for mathematical problems that occur repeatedly, in more or less standard forms. Such a general solution method, for a general class of problems, is called an *algorithm*. An algorithm consists of a precisely specified sequence of steps that will lead to a complete solution for a certain class of computational problems. One familiar example is the traditional long-division algorithm. If we wanted to instruct a machine to solve such a class of problems, the algorithm would tell us how to program the steps that the machine should perform on any instance of the problem to get the desired answer. Indeed, an algorithm can be thought of as a conceptual version of a machine program.

The term "algorithm" sometimes provokes disdain among educators because of the oppressive ways in which traditional algorithms often are taught. In fact, algorithms are remarkable tools in mathematics and computer science. They have great practical and theoretical importance. They also are important in learning mathematics.

A Familiar Algorithm

Consider the following example of a familiar algorithm to solve multidigit subtraction computation problems:

$$\begin{array}{r} 36 \\ -\ 19 \\ \hline \end{array}$$

One could invent a variety of ways to perform this calculation. One way is the following:

1. Check whether the number in the ones place in the top number (in this case, 6) is greater than the number in the ones place in the bottom number (in this case, 9).
   (a) If yes, subtract the second of these numbers from the first and write the result below the line, in the ones column.
   (b) If no (as in this case), regroup the tens and ones in the top number so that you represent the number with one less ten in the tens place and ten more ones in the ones place, and record this regrouped representation of number. (In this case, rewrite 36 as 2 tens and 16 ones.) If the digit in the tens place is 0, regroup the hundreds, tens, and ones as follows: First, regroup the hundreds and tens (for example, 403 would be rewritten as 3 hundreds, 10 tens, and 3 ones). Then, with 10 tens available, regroup the tens and ones. Follow analogous procedures if the hundreds digit, and possibly other digits to the left, also is 0. Subtract the number in the ones place in the bottom number from the number now in the ones place in the top number (in this case, 16 – 9). Record the result below the line, in the ones column.
2. Move to the numbers in the tens place. Repeat the steps in number 1, but subtract in the tens column. If regrouping is necessary, regroup hundreds and tens as above, and so on.
3. Continue in a similar fashion until all places of the numbers have been checked, regrouped if necessary, and subtracted.
4. The answer appears in standard base-10 place-value notation below the line.

The result, with written notation of the performance of the steps, might look like this:

$$\begin{array}{r} {}^{2}\not{3}\,{}^{1}6 \\ -\ \ 1\ 9 \\ \hline 1\ 7 \end{array}$$

Writing out the steps of this algorithm in words, in detail, and with precision is elaborate and complex. Recorded with mathematical notation, as above, the algorithm is elegantly compact. This compactness, however, hides the meaning and complexity of the steps involved. That this sequence of steps works efficiently to subtract any two whole numbers, no matter how great, is remarkable.

Important Qualities of Algorithms

Algorithms have qualities that are important to evaluating the algorithms' usefulness. These include the following:

- *Accuracy* (or *reliability*). The algorithm should always produce a correct answer.
- *Generality.* The algorithm applies to all instances of the problem, or class.
- *Efficiency* (or *complexity*). This refers to whether the cost (the time, effort, difficulty, or resources)

of executing the algorithm is reasonably low compared to the input size of the problem.
- *Ease of accurate use* (versus *error proneness*). The algorithm can be used reasonably easily and does not lead to a high frequency of error in use.
- *Transparency* (versus *opacity*). What the steps of the algorithm mean mathematically, and why they advance us toward the problem solution, is clearly visible.

To qualify as an algorithm, a procedure must be characterized by accuracy and generality. Ease of accurate use, efficiency, and transparency also are desirable qualities, although they often are in competition with one another. An algorithm that will be used to program a machine must be efficient, to achieve computational speed, but does not have to be transparent. If humans will learn and use the algorithm, however, transparency and ease of use are important.

An Alternative Algorithm

Consider a different algorithm for the same class of multidigit subtraction problems. We can examine how it works for the same problem:

$$\begin{array}{r} 36 \\ -\ 19 \\ \hline 2\ \bar{3} \end{array} \rightarrow 17$$

How does this method work? Instead of requiring the user to check whether the top number, in a given place, is greater than the bottom number, this algorithm permits the user to subtract separately on each column, using integers (negative as well as positive) to keep track of the parts of the calculation. In written form, the steps are as follows:

1. Subtract the number in the ones place of the bottom number from the number in the ones place of the top number (in this case, 6 – 9). Write the result below the line, in the ones place (in this case, $\bar{3}$).
2. Move to the tens place, subtracting the bottom number from the top number and writing the result. Continue in the same way until all places of the number have been subtracted.
3. Add the results in each column from left to right, and record the answer in standard decimal notation. In this example, add $20 + \bar{3}$; the result is 17.

Looking at the structure of this algorithm is one way to understand why it works. We can write an analogous subtraction equation in algebraic form:

$$(3x + 6) - (x + 9) = (2x - 3)$$

This says that when you subtract the term $(x + 9)$ from $(3x + 6)$, you will always get $(2x - 3)$, no matter what x is. But consider the case when $x = 10$. Then this equation can be written as follows:

$$(30 + 6) - (10 + 9) = (20 - 3)$$

Notice that this is the problem on which we have been working: 36 – 19. This way of writing the problem shows that the answer is the result of subtracting 30 – 10 (which is 20) and 6 – 9 (which is $\bar{3}$). Then the answer (17) is produced from 20 – 3. Also important to note is that the (20 – 3) corresponds to the digits 2 and $\bar{3}$, and these numbers represent 20 and $\bar{3}$. Because we can think of place-value representation of a number as a "polynomial" in powers of ten, this way of writing out the problem can help show that the algorithm works for any subtraction computation.

Comparing Familiar and Alternative Algorithms

How do these two algorithms compare?

- *Accuracy* (or *reliability*): Both algorithms produce the correct answer every time.
- *Generality*: Both algorithms apply to all multidigit subtraction problems.
- *Efficiency* (or *complexity*): The second algorithm is possibly more efficient because the steps are exactly the same for each place. No conditional judgment is required as it is for the first, where the user must determine in each column whether regrouping, and possibly even multiple regrouping, is required.
- *Ease of accurate use* (versus *error proneness*): The second algorithm can be used reasonably easily and does not lead to a high frequency of error in use. The first often leads to errors, especially when the user makes mistakes in the notation—for example, forgetting to completely record the result of regrouping.
- *Transparency* (versus *opacity*): The steps of the second algorithm are more transparent because each step involves merely a straight calculation that can be easily seen. The first algorithm involves a clever use of decimal notation to rewrite numbers in non-standard form so that in each place numbers of sufficient magnitude exist to carry out the subtractions without use of negative numbers (for example, as in rewriting 36 to 20 + 16 before subtracting 9 in the ones column). The meaning of what is being done, however, is not notationally apparent.

A Third Method

We can consider a third approach to subtracting multidigit numbers, using the same problem:

$$\begin{array}{rr} & 36 \\ - & 19 \end{array} \rightarrow \begin{array}{rr} & 37 \\ - & 20 \end{array} \rightarrow \begin{array}{rr} & 37 \\ - & 20 \\ \hline & 17 \end{array}$$

This method involves changing the two numbers by equal amounts and transforming the calculation into an easier one to perform. In this case, 37 – 20 can be performed mentally. Adding 1 to each of the numbers preserves their difference.

The steps might be written as follows:

1. Inspect the problem and decide how to transform the two numbers so that their difference remains the same but the calculation becomes simple.
2. Add or subtract the same amount to both numbers, based on judgment made in step 1.
3. Perform the mental calculation and write the result.

How does this method compare, using our criteria?

- *Accuracy* (or *reliability*): This method—adding or subtracting like amounts to each number—will always preserve the difference.
- *Generality*: This method would work for all multidigit subtraction problems.
- *Efficiency* (or *complexity*): This method is sometimes, but not always, efficient. Sometimes it requires high levels of user judgment and moving a complex computation to the front of the problem in order to produce a simple calculation at the end.
- *Ease of accurate use* (versus *error proneness*): This method depends heavily on user judgment and is prone to error in some cases.
- *Transparency* (versus *opacity*): This method is fairly transparent. Adding or subtracting like amounts to each number clearly preserves the difference.

A significant difference exists between the third method and the first two. The first two are algorithms; they involve steps that can be programmed to produce the correct answer. Although the third method is accurate and works in general, the amount to add to or subtract from each number cannot be precisely programmed. Deciding what to add or subtract depends on individual user judgment. It is not even clear that there is always a good choice available. Consequently, although it is a valid method, it cannot be called an algorithm.

Any mathematical problem admits multiple approaches and solution methods, each with its own merits and trade-offs. Proficient problem solvers will understand how to deploy a variety of such methods, how they are related, and how to make a discriminating choice of which method to use in any problem instance. This applies, in particular, to choices of algorithms or other methods for computational problems. It may, and often does, happen that an optimal algorithm or method to use varies significantly across different regimes of the problem class. For example, the third method is a useful choice for problems such as the one we have been using but would be less helpful for others. Computational fluency entails bringing problem-solving skills and understanding to computational problems.

Student-Invented "Algorithms" and Mental Arithmetic

Russell (2000) has highlighted the central role that student-invented procedures and strategies for computation play. She illustrates this by analyzing how a computationally fluent student might approach three different subtraction problems (see **fig. 1**).

The problem-solving skills outlined in **figure 1** certainly are important and valuable. At the same time, note what is not present. First, these three approaches, as presented, fall short of being algorithms for subtraction of multidigit numbers. "Counting up" becomes an algorithm only when one specifies a definite and general procedure for the counting. One possibility, counting up by ones, would define an algorithm (it is accurate and general) but would not be practical overall. Doing the count more efficiently, as above, clearly is preferable but is idiosyncratic to this case; what general form this might take is unclear. Similarly, the procedures used in cases (b) and (c) depend on special features of the problem instance. Because they are inherently not general, they are not algorithms.

One of the attractive features of these strategies is that they can be executed mentally and with transparent steps. They permit us to eliminate much record-keeping by relying on short-term memory and meaningful mental models of the operations. This works to treat many of the concrete computations that arise in daily life, in and out of the classroom. Human short-term memory is finite, however, which imposes important limitations on the scale of computations that can be treated by such methods. One response to this is to emphasize mental computation and have recourse to technology when the regime of the problems

FIGURE 1

Possible student strategies for solving three different subtraction problems

(a) 673 – 484
One method is "counting up." The approach for solving (a) might be to break the problem into the following mentally tracked steps:

$$673 - 484 = (500 - 484) + (600 - 500) + (673 - 600)$$
$$= 16 + 100 + 73 = 189$$

(b) 673 – 473
In example (b), the identical units and tens digits permit the student to recognize immediately that the difference is 600 – 400 = 200.

(c) 673 – 492
Example (c) invites the flexible problem solver to use the observation that 492 + 8 = 500. This may be deployed in either of the (mentally tracked) forms:

$673 - 492 = 673 - 500 + 8 = 173 + 8 = 181$, or
$673 - 492 = (673 + 8) - (492 + 8) = 681 - 500 = 181$.

becomes too large for mental methods.

This response would be compelling if the only purpose of learning about computation was to carry out explicit computations; however, it is shortsighted. Learning, with understanding, about computation and algorithms to solve problems is a natural and fertile site for learning about the detailed nature of numbers and operations and their various models and representations. Generality is one of the most important and powerful characteristics of mathematics. Students should not only know how to solve concrete problems with ad hoc methods but also be able to conceptually describe and experience mathematical phenomena that extend beyond direct concrete experience. For example, they should be able to conceive and "feel" the nature and structure of computations that are too large or abstract to execute concretely. Instruction, even in the early grades, should afford opportunities to develop such sensibilities about mathematical generality.

Basic Skills, Understanding, and Technology

Some of the public debates about education reform have pitted "basic skills," which are often characterized as knowledge of "standard" algorithms for numerical operations, against conceptual understanding. Sensible people now recognize that this is a false dichotomy. Both forms of knowledge are essential and are basically intertwined (Kilpatrick, Swafford, and Findell 2001). Nonetheless, some people stubbornly persist in arguing about which should come first. Some even argue that learning traditional algorithms can impair children's development of computational fluency. The evidence that they often cite, however, essentially refers to methods of instruction that have emphasized mindless and rote learning of algorithms. No reason exists to assume that this teaching approach is inherent to learning about algorithms. Regrettably, this confusion has provoked a lot of needless and sometimes acrimonious debate. We must recognize the mathematical richness and usefulness of algorithms and find ways to help students develop appropriate mathematical proficiency in constructing and analyzing them.

Traditional algorithms have evolved over time for frequent daily users who want to do routine calculations, essentially mechanically. They tend to be cleverly efficient (minimizing the amount of space and writing used) but also opaque (the steps are not notationally expressive of their mathematical meaning). Therefore, if these algorithms are learned mechanically and by rote, the opaque knowledge, unsupported by sense making and understanding, often is fragile and error-prone, as many researchers have documented.

Another common refrain is that the presence of computational technology now renders obsolete the need to learn "basic skills" because machines will do the computing for people. This argument is misguided. Clearly, the availability of technology calls for significant changes to the emphasis and orientation of computational instruction. These changes do not mean simply teaching students how to use calculators and computer algebra systems.

The nature and ubiquity of calculators bring into the foreground the very concept of algorithms—their nature, their qualities (such as accuracy, gen-

erality, efficiency, ease of use, and transparency), and how they might be analyzed, compared, and invented. Viewed in this light, traditional, novel, and student-invented algorithms suddenly occupy the same interactive space. In this space, the primary aims of study are the analyses and comparisons of algorithms, not simply the numerical answer that any one of them might produce. In this kind of instructional milieu, learning a traditional algorithm means learning not only how to execute it with several examples but also how to explain the mathematical significance of its various steps and prove that it produces a correct answer. Students should develop this sort of understanding for each algorithm they study. This kind of instruction offers an image of developing computational fluency in which basic skill is both strongly present and inseparable from conceptual understanding. It also supports the fluent and discriminating use of technology and an appreciation of, for example, the effects of errors caused by rounding off.

Understanding algorithms is central to developing computational fluency. Teachers must combine a renewed appreciation of the contributions algorithms make to mathematical proficiency with a design of approaches to teaching and learning that can develop both understanding and skill. They also must appreciate what efficiency affords and respect what it takes to use compact methods sensibly, flexibly, and appropriately.

References

Kilpatrick, J., J. Swafford, and B. Findell. *Adding It Up: Helping Children Learn Mathematics*. Washington, D.C.: National Research Council, 2001.

National Council of Teachers of Mathematics (NCTM). *Principles and Standards for School Mathematics.* Reston, Va.: NCTM, 2000.

Russell, Susan Jo. "Developing Computational Fluency with Whole Numbers." *Teaching Children Mathematics* 7 (November 2000): 154–58. ▲