Comparing the structure of algorithms: The case of long division and log division

Christof Weber

Comparing the structure of algorithms: The case of long division and log division

Christof Weber

School of Education, University of Applied Sciences Northwestern Switzerland, Muttenz, Switzerland; christof.weber@fhnw.ch

This paper is a theoretical contribution to the comprehension of logarithms by means of comparing algorithms. First an algorithmic approach to the logarithm (so-called log division) is introduced, which then is compared with the standard division algorithm (long division). By working out mathematical correspondences and differences between the two algorithms, this approach focuses on structural aspects of two mathematical operations and their relations. Comparing algorithms in classrooms is proposed as a way of algorithmic thinking that would go beyond rote memorization.

Keywords: Mathematical algorithms, logarithms, long division, comparison, algorithmic thinking.

Introduction

Algorithms have played a central role in mathematics since antiquity. Examples such as Heron's method (for computing square roots) and the Euclidean algorithm (for computing the greatest common divisor) are impressive evidence of this. Their development is still an integral part of mathematical research (Chabert, 1999; Ziegenbalg, 2016). Accordingly, mathematics education until the late Middle Ages was primarily computational education, that is, the teaching and practice of computational methods (Graumann, 2002; Ziegenbalg, 2016). Still today in school instruction of algorithms occupy an important place. In every case algorithms are "a finite sequence of explicitly defined, step-by-step computational procedures which ends in a clearly defined outcome" (Wu, 2011, p. 57). It is hardly surprising, then, that algorithms and also dealing with algorithms is sometimes regarded as one of the "central ideas" for mathematics education (Heymann, 2003).

As can be seen from the above examples, each algorithm always solves a whole class of structurally related computational problems. Another strength is that we do not have to worry about how it works or validate the result since it turns a problem into a routine task. This is a substantial relief for the user and effectively clears the way to tackle new, unresolved issues. The flip side of this liberating mindlessness is that users can also mechanically execute an algorithm without being able to justify it, to know the limits of its applicability, etc. Thus, learning algorithms takes place within a field of tension between mechanical memorizing and meaningfully acquiring, in short: *memorization by rote* vs. *conceptual understanding* (Baroody, 2003; Wu, 1999).

The dangers of rote learning have been empirically established in several studies in the case of standard algorithms for the four arithmetic operations (e.g. Kamii & Dominick, 1998). This could also be the reason why some authors contrast knowledge of standard algorithms with conceptual understanding (Baroody, 2003; Fan & Bokhove, 2014), or why the term "algorithmic thinking" is sometimes understood rather in a negative sense. Occasionally, a reduction or even a banishment of certain standard algorithms from the curriculum is advocated (for instance in Canada, see Fan & Bokhove, 2014, p. 482, or in North Rhine-Westphalia / Germany, see Krauthausen, 2018, p. 91).

Understanding algorithms in the classroom and in teacher education

However, not all authors go so far, but rather focus on the role of the teacher: "[Rote learning might take place] when the teacher does not possess a deep enough understanding of the underlying mathematics to explain it well. The problem of rote learning then lies with inadequate professional development and not with the algorithm" (Wu, 1999, p. 6). To defuse this problem, there are different suggestions for classroom instruction and teacher education. They can be divided into two strands:

1. A first strand of proposals recommends that students not be provided with ready-made algorithms and detailed directions for their use. Instead, they should be encouraged to *invent their own methods and individual strategies* for solving computational problems. In mathematics education at primary school level, this includes informal mental arithmetic as well as informal pencil-and-paper arithmetic. The point is, then, that students develop individual strategies in which they operate not with the individual digits of numbers but always with "numbers as wholes", that are decomposed in meaningful ways. In the case of the division, this could mean subtracting easy multiples of the divisor from the dividend until the dividend has been reduced to zero or the remainder is less than the divisor, then adding together the partial quotients to obtain the quotient ("partial quotient division", Kilpatrick, Swafford, & Findell, 2001, p. 221).
2. Another strand of suggestions recommends *comparing standard algorithms or individual strategies* in the classroom as a way to understand why they work. For example, Bass (2003) suggests that students in the classroom compare standard with alternative algorithms in terms of their generality or efficiency, while Simonsen and Teppo (1999) suggest this kind of comparison in teacher training programs. Empirical studies of Durkin, Star and Rittle-Johnson (2017) show in which way learners can benefit from comparing different worked-out strategies for solving the same linear equation. There is also a certain tradition in teacher education that elementary teacher trainees must compare standard algorithms such as long division in base-10 with the same algorithm in other number bases in order to understand better the connection between number bases and algorithms (Padberg & Büchter, 2015). The aim is to see the algorithm not only as a recipe and to perform it fluently, but also to see it in its relation to other algorithms by focusing on certain structural aspects.

The present article focuses on the second approach, the comparison of algorithms, in the context of the teaching and learning of logarithms. This calls forth in particular the following questions:

- *How can the logarithm be described algorithmically?*
- *What is the mathematical relation between the algorithmic description of logarithms and that of division? In what respect do the two algorithms correspond, and in what respect do they differ?*

Comparing log division with long division

Primary students learn that division of integers can be seen as partitive (fair-sharing) division as well as quotative (measurement) division (Greer, 1992). It is the measurement interpretation that allows the division to be conceived as an algorithm: in the long division algorithm, the divisor is repeatedly subtracted from the dividend until the remainder is as close to zero as possible (for a justification see Wu, 2011, pp. 110–122). Mathematically, this has to do with the fact that the divi-

sion is the inverse operation of multiplication and multiplication can be conceived as repeated addition (at least for whole numbers). Now, as logarithms are an inverse of powers, and powers are repeated multiplication, the logarithm can be treated algorithmically in a similar way—as a *repeated division* (Vos & Espedal, 2016; Weber, 2016). This allows logarithmic values to be calculated algorithmically. In keeping with the name of the standard division algorithm, "long division", I call the corresponding algorithm "log division". It is described now in order to then compare it with long division.

Log division: description of the algorithm

Repeated subtraction and repeated division can be naturally applied to calculate quotients and logarithms that come out even and yield integer answers: 8 : 2 equals 4 since $\underbrace{8-2-2-2-2}_{4\times}=0$, and in the same way, $\log_2 8$ equals 3 since $\underbrace{8:2:2:2}_{3\times}=1$. Relying on repeated division, I now give a step-by-step description of how the log division algorithm works (Goldberg, 2006; Weber, 2016). Suppose you have to compute $\log_{16}32$ (see Figure 1):

Step 1: Calculate how many times the argument (32) can be divided by the base (16) before the result is less than the base. In our case, 32 can be divided by 16 only one time (because dividing 32 by 16 two times, the quotient would be less than 1). You record this result of 1 above the bracket (noted as . in Figure 1).

Step 2: Next, you calculate the base raised to this result: $16^1 = 16$. You place this number beneath the argument of the logarithm and divide (instead of subtracting as in long division) the argument by this value ($32 : 16^1$). Note the result (2) as a remainder underneath.

Step 3: You raise the remainder to the 10-th power (recall that in the same step in long division you bring down a further digit from the dividend, in effect multiplying the original remainder by 10). Note the result as the new argument: $2^{10} = 1024$.

Step 4: If (as in our example) the new argument does not equal 1, go back to the Step 1, if it equals 1 (not 0 as in long division), the process has come to an end.

Conclusion: By cycling through the steps 1 to 4 until arriving at the argument 1, you obtain the single digits of the decimal expansion of the result. In our example, you reach the remainder of 1 after two additional cycles (producing 1, 2, 5 as the successive exponents of 16.) Stringing the numbers together, the decimal expansion of $\log_{16}32$ is found to be to be 1.25.

Figure 1 shows not only the computation of $\log_{16}32$ digit by digit, but also the corresponding logarithmic expressions that reveal what the algorithm actually "does." It also shows that the algorithm works not only for natural, but also for non-integer results. In order to emphasize the *mathematical analogy to division*, those who use the colon ":" notation for long division (for instance in countries where Romance languages or German are spoken) could use the non-standard tricolon "⁝" notation for log division, for instance $\log_{16}32 = 32 \vdots 16$ (for the historical background, see Weber, 2016).

Those who use the bracket notation "□" (for instance in English-speaking countries, Mexico or Japan) could use the notation "□ □" for log division, $\log_{16}32 = 16 \overline{)\,32}$ (Weber, 2019).

$\log_{16}32 = 32 : 16 = 1.25$ $\underline{: 16 \left(= 16^{\boxed{1}}\right)}$ 2 $2^{10} = 1024$ $\underline{: 256 \left(= 16^{\boxed{2}}\right)}$ 4 $4^{10} = 1\,048\,576$ $\underline{: 1\,048\,576 \left(= 16^{\boxed{5}}\right)}$ 1	$\log_{16}32 =$ $= \log_{16}(16^{\boxed{1}} \cdot 2) = \boxed{1} + \log_{16}(2)$ $= 1 + \frac{1}{10} \cdot \log_{16}(2^{10})$ $= 1 + \frac{1}{10} \cdot \log_{16}(1024)$ $= 1 + \frac{1}{10} \cdot \log_{16}(16^{\boxed{2}} \cdot 4) \quad = 1.\boxed{2} + \frac{1}{10} \cdot \log_{16}(4)$ $= 1.2 + \frac{1}{100} \cdot \log_{16}(4^{10})$ $= 1.2 + \frac{1}{100} \cdot \log_{16}(1\,048\,576)$ $= 1.2 + \frac{1}{100} \cdot \log_{16}(16^{\boxed{5}} \cdot 1) = 1.2\boxed{5}$

Figure 1: Log division algorithm (applied to $\log_{16}32$) – *to the left*. Stepwise explanation – *to the right*

Comparing the mathematical structure

In order to not only explain the algorithms by means of examples, an approach to a more thorough, mathematical justification is given now, in two steps: To start with, I explain why division is equivalent to repeated subtraction and why finding the logarithm is equivalent to repeated division, which allows us then to compare the respective algorithms with each other.

For this we first consider the following mathematical *equivalence relations*:

- For the division of a whole number *b* by a whole number *a* ($a \neq 0$) one has:

$$? = b : a \quad \longleftrightarrow \quad ? \times a = b \quad \longleftrightarrow \quad b - ? \times a = 0$$

- For the logarithm of a natural number *b* to a natural-number base *a* ($a \neq 1$) one has:

$$? = \log_a b \quad \longleftrightarrow \quad a^{?} = b \quad \longleftrightarrow \quad b : a^{?} = 1$$

From these relations follow several conclusions:

1. The quotient $b : a$ gives (in the context of whole numbers) "how many *a*'s are in *b*" as a *summand* (Wu, 2011, p. 103). On the other hand, the logarithm $\log_a b$ (or $b : a$, respectively) gives how many *a*'s are in *b* as a *factor*.
2. The two equivalence relations structurally resemble each other, since they are each based on a respective inverse operation. In this respect, they are *analogous* to each other. However, they operate at different hierarchy levels: while division and multiplication are second-level operations,

built upon the first-level operation of subtraction, logarithms and powers are third-level operations, built upon the second-level operation of division (see Figure 2).

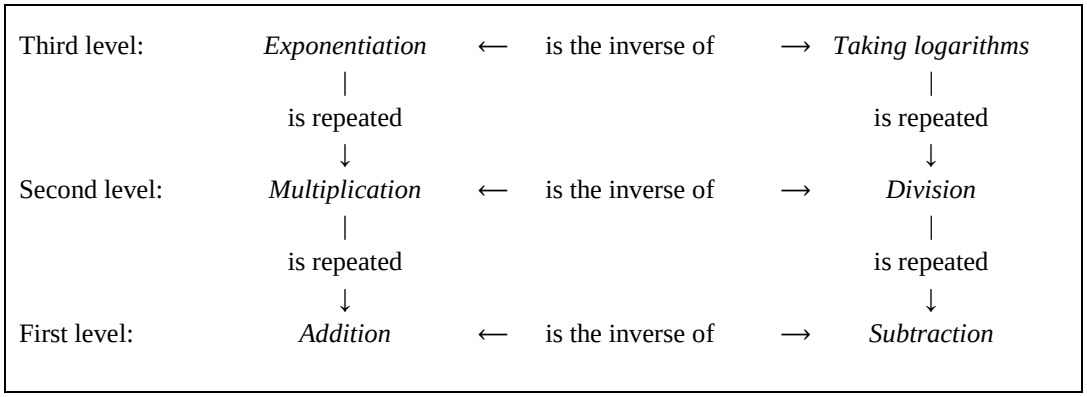

Figure 2: Hierarchy levels of arithmetic operations

3. According to conclusion 1 above, *algorithms* can be derived to calculate the value of the corresponding quotient or the corresponding logarithm. Thus, the algorithm for long division counts how many times the number *a* is subtracted from the number *b*. The log division, on the other hand, counts how many times the number *a* divides the number *b*. Because of conclusion 2, the two algorithms are also analogous to one another; in particular the individual calculation steps correspond:

   · Analogous to the fact that in the long division the remainders are multiplied by 10 (a second level operation), the remainders in the log division must be raised to the power 10 (see the corresponding third level operation) (see Step 3, above).

   · The algorithm of long division ends as soon as the repeated subtraction leads to the remainder 0 (the additive identity). Accordingly, log division terminates as soon as the repeated division leads to the "remainder" 1 (the multiplicative identity) (see Step 4). If this occurs, the result is an integer or a finite decimal expansion (see Figure 1), otherwise not.

In addition to this analogy and its consequences, there are also two significant *differences* between the two algorithms that should be kept in mind:

1. Numbers are usually encoded as decimal numbers, meaning they are decomposed as a series of digits, each weighted by a power of ten, and added together. As with all standard algorithms of arithmetic, the strength of the long division is that it does not divide the dividend as a whole, but essentially digit by digit (Wu, 2011, p. 56). That this can be done is due to the fact that division distributes over addition, $(b_1 + b_2) : a = b_1 : a + b_2 : a$. In order to make the division algorithm more accessible in the classroom, it is sometimes suggested to discuss the already-mentioned partial quotient method, which treats the dividend as a whole entity, from which easy multiples of the divisor are subtracted (see above).

Unfortunately, logarithms do not distributive over addition, $\log_a(b_1 + b_2) \neq \log_a b_1 + \log_a b_2$. Hence it is not possible to march along, calculating one digit of the logarithm for each digit of the argument as with division: While 32 = 30 + 2, log 32 ≠ log 30 + log 2. As a result, you have to pay attention from the beginning to all the digits of the number, similar to the partial quotients division (see Step 1).

2. The multiplication operation, for example $2 \times 4 = 8$, can be inverted in two ways: Either you are interested in the first factor $? \times 4 = 8$, or the second factor ($2 \times ? = 8$). While the first factor can be found via repeated subtraction (measurement interpretation), $8 \underbrace{- 4 - \ldots - 4}_{? \times} = 0$, the second question corresponds to the fair-sharing (partitioning interpretation), $8 \underbrace{- ? - ? - ? - ?}_{4 \times} = 0$, an equation that can only be immediately solved by guessing. Since however multiplication is commutative, the question of the second factor can always be handled as a question regarding the first factor: $2 \times ? = ? \times 2$. To sum up: *Since multiplication is a binary operation, finding its inverse can be understood in two ways, but due to its commutativity, both lead in the end to one and the same operation, division*. (Wu, 2011, pp. 99–100)

   In the same way the operation of exponentiation, for example, $2^3 = 8$, can be inverted in different ways, depending on whether you are interested in finding the exponent $2^? = 8$, or the base $?^3 = 8$. The answer to the first question can be calculated through repeated division step-by-step: $8 \underbrace{: 2 : \ldots : 2}_{? \times} = 1$, while the second question leads to the equation $8 \underbrace{: ? : \ldots : ?}_{3 \times} = 1$. In contrast to multiplication, the one form of inverse cannot be applied to find the other one, since exponentiation is not commutative. In other words: *On account of its non-commutativity* ($a^b \neq b^a$), *exponentiation has two distinct inverse operations: taking the logarithm,* $\log_2 8 = 3$, *as well as extracting the root,* $\sqrt[3]{8} = 2$. Accordingly, the method of repeated division can only be applied to solve inverse questions regarding the *exponent* of a power, but not those concerning its *base*.

Conclusion and questions

There is a certain tradition in mathematics education to place concepts of elementary mathematics into meaningful contexts in order to make them easier to learn, and to promote efficient and flexible application later on (e.g., for long division, see Pratt, Lupton & Richardson, 2015). For secondary schools, the situation is a bit different. So while it may be possible to occasionally identify meaningful contexts, with increasingly abstract content this is not necessarily desired. For concepts such as the logarithm, which can be conceptualized algorithmically, it therefore makes sense to compare them with other mathematical algorithms in order to make them accessible and to understand why they work (Bass, 2003). While I have described the algorithmic approach of "log division" previously, the correspondences and differences between the algorithms seem to have never been worked out before.

This paper also raises various questions. First of all, the question of the mathematical generalization of the log division from natural to rational bases and arguments remains open. The question of the

explanatory power of the log division is also not covered here, that is, in which situations our algorithmic conceptualization of logarithms can serve as a basis for solving tasks and for reasoning (for instance, one could argue that logarithms are not defined for negative arguments because a negative number divided repeatedly by the positive basis will never yield 1, see Weber, 2016). Another group of questions is empirical in nature and asks about the effects and pitfalls that an algorithmic approach combined with comparison would have in teacher education and in subsequent classroom instruction (for a classroom teaching experiment to introduce logarithms by log division, see Weber 2019). In particular, in which ways might comparing the structure of algorithms foster conceptual understanding?

Algorithmic and algebraic thinking

Last but not least, the results of this contribution might be used to clarify the construct "algorithmic thinking" in mathematics education: comparing algorithms might be a way of dealing with algorithms in the classroom beyond just learning to perform them. For instance, in order to bring out further aspects of the construct, the cognitive processes of students and teachers could be studied when they work on the new algorithm of log division by relating it to the familiar long division.

Teaching algorithms by comparing them, as outlined above, focuses on seeing structure and relationships between mathematical operations. While seeing structure in numerical expressions can be seen as algebraic thinking at primary level (Kaput, 2008), comparing algorithms might be seen as a specific form of algebraic reasoning at the secondary level (Kieran, 2018). Kaput's view that "the heart of algebraic reasoning is comprised of complex symbolization processes that serve purposeful generalization and reasoning with generalizations" (2008, p. 8), raises more questions related to the contents of this paper. For instance, at primary and secondary school, algorithms are never explained or justified in full generality, but by generic examples. To what extent is this characteristic similar to teaching early algebraic reasoning by generic arguments? Furthermore, any given algorithm does not solve just a single task, but a class of multiple problems. To what extent is this characteristic similar to algebraic reasoning, understood as "reasoning with generalizations"? Future research will have to clarify and answer these and several other questions.

References

Baroody, A. J. (2003). The development of adaptive expertise and flexibility: The integration of conceptual and procedural knowledge. In A. J. Baroody & A. Dowker (Eds.), *The development of arithmetic concepts and skills* (pp. 1–33). Mahwah, NJ: Erlbaum Associates.

Bass, H. (2003). Computational fluency, algorithms, and mathematical proficiency: One mathematician's perspective. *Teaching Children Mathematics*, *9*, 322–27.

Chabert, J.-L. (1999). *A history of algorithms*. Berlin, Germany: Springer.

Durkin, K., Star, J. R., & Rittle-Johnson, B. (2017). Using comparison of multiple strategies in the mathematics classroom: lessons learned and next steps. *ZDM Mathematics Education*, *49*, 585–597.

Fan, L., & Bokhove, C. (2014). Rethinking the role of algorithms in school mathematics: a conceptual model with focus on cognitive development. *ZDM Mathematics Education*, *46*, 481–492.

Goldberg, M. (2006). Computing logarithms digit-by-digit. *International Journal of Mathematical Education in Science and Technology*, *37*, 109–114.

Greer, B. (1992). Multiplication and division as models of situations. In D. A. Grouws (Ed.), *Handbook for research on mathematics teaching and learning* (pp. 276–295). New York: Macmillan.

Heymann, H. W. (2003). *Why teach mathematics? A focus on general education*. Dordrecht, Netherlands: Springer.

Kamii C., & Dominick, A. (1998). To teach or not to teach algorithms. *Journal of Mathematical Behavior*, 16, 51–61.

Kaput, J. (2008). What is algebra? What is algebraic reasoning? In J. Kaput, D. Carraher, & M. Blanton (Eds.), *Algebra in the early grades* (pp. 5–18). New York: Lawrence Erlbaum.

Kieran, C. (2018). Seeking, using, and expressing structure in numbers and numerical operations: A fundamental path to developing early algebraic thinking. In C. Kieran (Ed.), *Teaching and learning algebraic thinking with 5- to 12-year-olds* (pp. 79–105). Cham, Switzerland: Springer.

Kilpatrick, J., Swafford, J., & Findell, B. (2001). *Adding it up: Helping children learn mathematics*. Washington, DC: National Academy Press.

Krauthausen, G. (2018). *Einführung in die Mathematikdidaktik – Grundschule*. Berlin, Germany: Springer.

Padberg, F., & Büchter, A. (2015). *Einführung Mathematik Primarstufe – Arithmetik*. Berlin, Germany: Springer Spektrum.

Pratt, S., Lupton, T., & Richardson, K. (2015). Division quilts: A measurement model. *Teaching Children Mathematics*, 22, 103–109.

Simonsen, L., & Teppo, A. (1999). Using alternative algorithms with preservice teachers. *Teaching Children Mathematics*, *5*, 516–519.

Vos, P., & Espedal, B. (2016). Logarithms – a meaningful approach with repeated division. *Mathematics Teaching*, *251*, 30–33.

Weber, C. (2016). Making logarithms accessible — operational and structural basic models for logarithms. *Journal für Mathematik-Didaktik*, *37*(Suppl. 1), 69–98.

Weber, C. (2019). Making sense of logarithms as counting division. *The Mathematics Teacher*, *112*, 374–380.

Wu, H.-S. (1999). Basic skills versus conceptual understanding. *American Educator*, *23*, 1–7.

Wu, H.-S. (2011). *Understanding numbers in elementary school mathematics.* Providence, RI: American Mathematical Society.

Ziegenbalg, J., Ziegenbalg, O., & Ziegenbalg, B. (2016). *Algorithmen von Hammurapi bis Gödel*. Wiesbaden, Germany: Springer Spektrum.