Exemplifying algorithmic thinking in mathematics education

Elise Lockwood, Anna DeJarnette, Matt Thomas, Knut Mørken

Exemplifying algorithmic thinking in mathematics education

Elise Lockwood[1], Anna DeJarnette[2], Matt Thomas[3], and Knut Mørken[4]

[1]Oregon State University, United States; Elise.Lockwood@oregonstate.edu

[2]University of Cincinnati, United States; dejarnaa@ucmail.uc.edu

[3]Ithaca College, United States; mthomas7@ithaca.edu

[4]University of Oslo, Norway; knutm@math.uio.no

In this paper, we consider the construct of algorithmic thinking in mathematics education. We are convinced that algorithmic thinking is an invaluable way of thinking for students to develop, and so we are motivated to promote theoretical discussions in the field about the nature and utility of algorithmic thinking. We present three examples of algorithmic thinking – a mathematical example, an example from a mathematician interview, and an example from an undergraduate student interview. We then briefly review some relevant literature related to algorithmic thinking, and we conclude with some avenues for future research. Our goal is to further conversations about and refinements of characterizations of this important topic.

Keywords: Algorithmic thinking, Computing, Mathematicians, Undergraduate students.

Introduction and Motivation.

The phrase "algorithmic thinking" (AT) has appeared intermittently in mathematics education literature for the past several decades (e.g., Abramovich, 2015; Knuth, 1985; Schwank, 1993; Stephens, 2018), and it seems to be gaining a resurgence of interest with corresponding attention in computation and computational thinking in mathematics settings. In this article, we explore the construct of algorithmic thinking in mathematics education research. We are motivated by observations from our research with students and mathematicians, as well as in our own mathematical experiences. We have noticed a certain kind of algorithmic approach to problems, particularly within computational, machine-based settings, which reflect the presence of algorithmic thinking. These approaches, and the reasoning that underlies them, seem to be useful and valuable, and thus they represent a phenomenon that we want to better understand. In addition, we are motivated by the increasing presence of the term AT in mathematics and CS education literature, and we want to facilitate consistency and coherence for how it might be used in mathematics education.

We have two goals in this paper. First, we aim to exemplify AT within computational settings. We provide examples of such approaches in three ways: in mathematical examples, in interview data with mathematicians, and in data with undergraduate students. These examples allow us to illustrate what we mean by algorithmic approaches, hopefully facilitating better communication and discussion about the construct of AT. Second, given the existence of such algorithmic approaches and their reflection of AT, we briefly situate these ideas within math education literature and suggest ways to explore AT in future work. Ultimately, we aim to establish a shared understanding of algorithmic thinking, which could contribute to broader interests within the mathematics

education community about the nature of thinking and problem solving in an increasingly computational world.

Broadly, an algorithm can be characterized as a set of steps to accomplish a given task. This broad definition allows for the term algorithm to include non-mathematical tasks like making a cup of coffee, as well as encompassing processes like the bisection method presented below. Rasmussen, Zandieh, King, and Teppo (2005) offered a definition of "algorithm" that serves as a useful starting point and highlights one way to characterize a distinction between an algorithm and a procedure in mathematics. Rasmussen et al. noted, "we use the term 'procedure' to indicate steps used to solve a particular task, and the term 'algorithm' as a reference for a generalized procedure that is effective across a wide range of tasks" (2005, p. 63). Implicit in this characterization is that there is often an underlying, generalizable approach of way of thinking on which an algorithm is based. Gravemeijer and van Galen (2003) made an analogy between mathematical algorithms and facts, using the example of the formula for area of a triangle (p. 115). To *know* a formula such as $A = \frac{1}{2} b h$ implies that one can apply that formula to calculate area across a range of examples. This is consistent with the notion of a generalized procedure, and this is the general perspective of algorithms that we take in our work. Further, we acknowledge that any process could technically be considered algorithmic if it involves steps of any kind, but we are interested in approaches that foreground the *development* of an algorithm as opposed to only the performance or implementation of algorithms.

The phenomenon that we discuss in this paper is the ability to develop, explain, and iterate the steps of an algorithm in a mathematical context. Our examples highlight algorithmic approaches within a computational setting because such a setting tends to foreground the value of such approaches.

Examples of Algorithmic Approaches.

We offer examples of algebraic approaches in three contexts: a mathematical example, an example from an interview with a mathematician, and an example from an interview with an undergraduate student. In each of these cases we explain what makes the approach algorithmic, and we also discuss potential affordances of such an approach. Our aim is to demonstrate what we mean by algorithmic approaches, and also to make the case that such approaches arise in a variety of settings and situations.

An algorithmic approach in a mathematical example – the bisection method.

We begin with a mathematical example. We focus on an example of solving equations, which is a classical and central topic in mathematics. The standard approach to solving $f(x) = 0$ is to apply a sequence of algebraic operations to both sides of the equation, with the ultimate goal of isolating $\boldsymbol{x}$ on one side. This works well for certain classes of equations like polynomials of degree at most four and some equations involving trigonometric, exponential, and logarithmic functions. In more general situations it is usually not possible to find an explicit formula for the solution. However, the Intermediate Value Theorem tells us that a solution does exist even in very general situations: "Let f be a continuous function defined on the real interval $[a,b]$. If $f(a)$ and $f(b)$ have opposite signs, then f must have a zero at some real number c in (a,b)."

The Intermediate Value Theorem says nothing about where in $[a, b]$ the zero is, but it can help us to develop a strategy for computing approximations to the zero. Perhaps the most obvious approach is systematic guessing, and the simplest guess is the midpoint $m = (a + b)/2$. If we compute $f(m)$ and determine its sign, we see that we can limit our attention to a smaller interval $[a_1, b_1]$:

1. If $f(m) = 0$, we have stumbled upon the zero c and there is nothing more to do so we set $[a_1, b_1] = [m, m]$.
2. If $f(m) \neq 0$, then f has opposite signs at the two ends of either the subinterval $[a, m]$ or the other subinterval $[m, b]$. In the former case we set $[a_1, b_1] = [a, m]$, in the latter case we set $[a_1, b_1] = [m, b]$.

This process can be repeated with the new interval $[a_1, b_1]$, and we then obtain a new interval $[a_2, b_2]$. If we repeat again and again we obtain a sequence of ever smaller intervals $[a_3, b_3]$, … , $[a_n, b_n]$, … . We note that c is located inside each of the subintervals, but the length of the intervals is halved each time. We can then conclude that if we stop this process after n steps, we know that the current midpoint satisfies $|c - m_n| \leq (b - a)/2^n$. This is a very different approach toward solving an equation than balancing the two sides of the equals sign. Rather, this approach, which is known as the bisection method, is inherently algorithmic, and the big idea is that by iterating steps in this process, we can ensure that we can find as close an approximation to the actual zero as we would like. In fact, this idea can be adapted into a proof of the Intermediate Value Theorem. In addition, this idea can be made more specific in the form of pseudocode that describes the steps more precisely, as in Figure 1.

```
a_0 = a; b_0 = b;
for i = 1, 2, ..., N
    m_{i-1} = (a_{i-1} + b_{i-1})/2;
    if f(m_{i-1}) == 0
        a_i = b_i = m_{i-1};
    if f(a_{i-1}) f(m_{i-1}) < 0
        a_i = a_{i-1};
        b_i = m_{i-1};
    else
        a_i = m_{i-1};
        b_i = b_{i-1};

m_N = (a_N + b_N)/2;
```

Figure 1: Pseudocode for an algorithm that represents the bisection method

This algorithm may be converted into code in a suitable language, and we then have our own substitute for the "solve-button" on advanced calculators. This algorithmic approach to solving equations naturally raises some questions, such as *Is this a valid way to solve an equation? What is gained by converting the algorithm into code and running the resulting program on examples?*
What do students gain from understanding such an approach to solving equations? We would consider the approach of iteratively finding the zeros of the function via the bisection method as we described it as inherently algorithmic because it involves first designing and then implementing an

iterative series of steps, which could generalize regardless of the function. The iterative, algorithmic process is foregrounded in this approach.

An algorithmic approach from a mathematician interview – summing primes.

As another example of an algorithmic approach, and the importance and potential usefulness of such an approach, we draw on an excerpt from a conversation with a mathematician, pseudonym Michael, about the role of algorithms in teaching mathematics at the post-secondary level. The excerpt comes from one of a set of interviews we conducted with research mathematicians about how they use computation in their work. In the first of these interviews, the interviewer (the first author of this paper) noticed the mathematician returning to the importance of algorithms and what he called "algorithmic approaches," so the interviewer asked him to expound on these ideas. Michael, a mathematical biologist, began with an example from a mathematics course that he regularly taught:

> Michael: It's been interesting to observe that there are some people who just don't get how to do algorithms…So for instance, a really simple example that I always have at least one student ask me in the computational course—I ask them to sum the first hundred prime numbers using MatLab, and MatLab has this function called Primes. And the way that function works is you put in a number, and it returns all the primes less than that number. And I always have students ask me, 'but I only want the first hundred. I don't know which prime is the hundredth one.' So, and it's funny because other people just go, obviously, 'Oh pick a big number and just take the first hundred…And so that second one was a series of steps: pick a large number, plug it in the function, take the first hundred, and you're done. The other ones are like, I don't know if they don't, can't translate the question into a series of steps like that.

To us, Michael articulated an algorithmic approach that some students were able to leverage. Notably, he seemed to be describing a difference between students who do or do not have such an approach at hand (or, we would interpret, do or do not think in such a way on such a problem), and he was also implying some practical ramifications for not being able to think in such a way. That is, without the algorithmic approach he described students get stuck on the problem and do not know how to proceed. We consider this a description of an algorithm and not a procedure because it suggests an underlying approach that could be generalizable regardless of which prime is being considered. Thus, it is important to help students gain access to these approaches and to try to make some progress on these ideas. This example sets up a distinction in this professor's mind that there is something akin to thinking algorithmically (or to think in such a way as to leverage that approach), there are affordances to such an approach, and not everyone automatically uses that approach.

An algorithmic approach in a student interview – articulating "the way a computer thinks."

As a final example, we offer an example from an interview with a pair of undergraduate engineering majors who were enrolled in a vector calculus class, Corey and CJ (pseudonyms). These data came from a paired teaching experiment that occurred for a total of 12 hours over nine 60-90-minute sessions, during which the students wrote Python programs to list outcomes and solve combinatorial problems. We present an episode from the fifth session, in which the students were solving the Marbles problem: *Suppose you have six different marbles in a bag. Write out all of the*

possible ways you could pick two marbles out of the bag, without replacement. They were asked to write code that would solve the problem. CJ listed the outcomes by hand as seen in Figure 2.

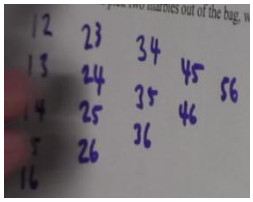

Figure 2: CJ's list of outcomes for the Marbles problem

Confident of their list, the students began to think about how to code the problem. CJ had an insight based on the list he had written by hand: "If we're just trying to list – the second one it looks at has to be bigger than that one." The interviewer asked him to follow up on this. CJ wrote the code in Figure 3, which lists pairs from the set of marbles via nested looping, where the conditional statement only prints outcomes in which the second term in the pair is bigger than the previous term. The *total* here acts as a counter, which is incremented each time and is printed at the end (yielding 15).

```
marbles = [1, 2, 3, 4, 5, 6]
total = 0

for i in marbles:
    for j in marbles:
        if i != j and j > i:
            print(i, j)
            total = total + 1
print(total)
```

```
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
15
```

Figure 3: CJ and Corey's code and output for the Marbles problem

Int. 1:	CJ, you observed something in there about the second column.
CJ:	If it just goes through **systematically the way a computer thinks**, looking at the next one and then using it; then our second column always has to be bigger than our first column. If *j* is bigger than *i*, then it won't ever print 2 and then 1. […] The *i* not equal to j that just makes it so it can't be 1, 1; or 2, 2. If *j* was bigger than *i*, then print it. These are all the orders, it won't ever do the same combination twice because it won't choose 1 and 2; then 2 and 1.
Corey:	[Corey adds to the code in Figure 3] And *j* is greater than *i*.
Int. 1:	Cool. If you do it, can you describe how many you think you'll get and what you think the outcomes will look like? Do you think – ?
CJ:	I think it'll go through how I did it.
Corey:	I think it'll go exactly the way he did it.
Int. 1:	Great. What about your program makes you think it'll run through in that way?
CJ:	Looking at *i* first, it's gonna go through everything that could have 1, and everything that's bigger than 1. Then it's gonna go to the next part; then it's gonna go to 2; then it's gonna look at everything that's bigger than two and print everything with that.

CJ reasoned about an algorithmic approach in this problem. This is seen when he discussed going systematically "the way a computer thinks," which suggests and underlying, generalizable approach

rather than just one procedure. He was thinking about what algorithmic process the computer might adopt to list the outcomes. He connected his listing process to what the computer might do and correctly asserted that their code would list the outcomes as he had. We argue that the students adopted an algorithmic approach in writing their code, and they thought about what process the computer might complete to accomplish a task. In this sense, kind of algorithmic thinking we are describing is important students' engagement with computational practices like programming. In addition to helping students think about computational practices like programming, such an approach can also useful in highlighting combinatorial ideas. Lockwood & De Chenne (2020) explored how students' reasoning about conditional statements like "if j != i" and "if j > i" supported students' reasoning about permutations and combinations, respectively. Thus, the algorithmic thinking used to reason about programming can then also help to enrich students' mathematical understanding as well.

To summarize this section, we highlighted three instances of algorithmic approaches that emerged in our data and our mathematical experiences. These examples are meant to illustrate what we mean by algorithmic approaches and also to motivate our focus on better understanding such approaches. Each example shares common features – the construction of a systematic, step-by-step process which could (though does not have to) be programmed into a machine. We suggest that there is some way of thinking that underlies these algorithmic approaches, and this is what we want to try to characterize and understand. AT could be the construct that underlies algorithmic approaches, and, ultimately, these examples compel us to explore what AT might entail and why it might be important. Having exemplified what we mean by an algorithmic approach, we now briefly review mathematics education literature for existing characterizations of AT.

Algorithmic Thinking in Mathematics Education.

There are some perspectives in math education literature that frame AT as characterizing *a way of thinking* rather than thinking about algorithms themselves. This is in line with the approaches we exemplified in this paper. The phrase "algorithmic thinking" appears occasionally in mathematics education literature, and the construct is used in a variety of ways. Knuth (1985) proposed a distinction between "algorithmic thinking" and "mathematical thinking" as an effort to distinguish the thought processes of computer scientists from those of mathematicians. Ultimately, Knuth concluded that there is no such singular concept as mathematical thinking, nor is there for algorithmic thinking, but rather each is comprised of a set of modes of thought. Many of these modes overlap (e.g., formula manipulation, abstract reasoning, information structures) with a few exceptions. Notably, "mathematical thinking" includes the infinite while algorithmic thinking does not; and algorithmic thinking accounts for problem complexity (i.e., the "cost" of running an algorithm), which does not typically surface in mathematics. Knuth's characterization of algorithmic thinking seemed to contribute primarily to highlighting some of the overlap—and some of the distinctions—of the disciplines of mathematics and computer science.

The NCTM handbook reported on differences between algorithmic thinking and recursive thinking. Although this is an interesting characterization, we note that their definition is quite broad and could apply to almost any process or mathematical situation. They define algorithmic thinking as

follows:

> *Algorithmic thinking* is a method of thinking and guiding thought processes that uses step-by-step procedures, requires inputs and produces outputs, requires decisions about the quality and appropriateness of information coming in and information going out, and monitors the thought processes as a means of controlling and directing the thinking process. In essence, algorithmic thinking is simultaneously a method of thinking and a means for thinking about one's thinking.

Schwank (1993) investigated what she referred to as different mental models that students might apply to algorithmic thinking, although she does not define algorithmic thinking in this paper. The types of tasks that she used to characterize algorithmic thinking were more aligned with typical computer science or computer programming tasks. In this work, the term "algorithmic thinking" implicitly referred to the types of thinking necessary to construct algorithms, although Schwank's primary argument was that there are multiple ways of doing so. More recently, Abramovich (2015) used the phrase "algorithmic thinking" somewhat analogously to how other researchers have used "procedural knowledge" in conversations about the relationship between procedural knowledge and conceptual understanding for learning mathematics. Abramovich described how problem posing—the cyclical act of extending a concrete (often procedural) problem to a more generalized case—could promote a link between procedural skills and conceptual understanding. There continue to be new contributions to the discussion of algorithmic thinking and how it can be integrated meaningfully in to the mathematics curriculum (e.g., Stephens, 2018; Stephens & Kadijevich (2020)). Stephens (2018) argues for AT as one type of reasoning and suggests that we consider ways to leverage powerful developments in programming to improve mathematics education.

Another line of research emphasizes activities and practices related to the creation of algorithms, such as *algorithmatizing*, which is an approach through which students develop algorithms through carefully chosen contextual problems that require students to model a particular situation, reflect on their solution procedures, and develop increasingly sophisticated models and procedures that can translate to other situations (Gavemeijer & van Galen, 2003, p. 114). The process of learning to use and understand algorithms implies their construction—or reinvention—on the part of students, rather than the acquisition of algorithms as existing objects. At the post-secondary level, students in a differential equations class reinvented Euler's method for approximating solutions to differential equations, following the careful selection of tasks that facilitated the generalization of procedures (Rasmussen & King, 2000; Rasmussen et al., 2005). By engaging in algorithmitizing, which involves reasoning about and developing algorithms, students must necessarily think critically about what is entailed in that algorithm. Thus, algorithmitizing can be thought of as an external activity that reflects thinking about an algorithm.

To summarize, in the mathematics education research literature, the term AT has been used and construed in a variety of ways. There are similarities in terms of presenting a general approach toward solving problems, but we think there is room to specify a more consistent and coherent characterization of AT in math education. (Regrettably, due to space, we do not elaborate AT in the computer science education literature, although that is another rich resource and set of perspectives).

Future Work Toward Characterizing Algorithmic Thinking

Having exemplified algorithmic approaches that we feel represent AT, and having briefly reviewed some relevant literature, we conclude with some ideas for more work that needs to be done to explore and elaborate the construct of AT. There is not clear consensus among research communities about what AT is or how it should be characterized or defined, and more work should be done to study explore what is entailed in AT. Once characterizations of AT are established, empirical studies can be designed to explore how students, instructors, and mathematicians engage with and employ AT. One potential future area of research is to explore how students interpret, evaluate, and/or debug existing algorithms. In addition, we acknowledge that AT should be framed within other kinds of thinking – such as computational thinking (Wing, 2006, 2008), mathematical thinking, or recursive thinking – and we believe that more work is needed to relate such constructs and terms. In this brief paper we do not have space to explore AT's relationship to those other kinds of thinking, but there is potential for rich theoretical investigations. It would also be worthwhile to explore the computer science education literature as it relates to AT and to compare and contrast the development and use of algorithms in computer science versus mathematics and other fields.

Acknowledgment

This work is funded in part by the National Science Foundation, Grant # 1650943.

References

Abramovich, S. (2015). Mathematical problem posing as a link between algorithmic thinking and conceptual knowledge. *The Teaching of Mathematics, 18*(2), 45–60.

Futschek, G., & Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. *Constructionism*, 1–10.

Gravemeijer, K., & van Galen, F. (2003). Facts and algorithms as products of students' own mathematical activity. In J. Kilpatrick, W. G. Martin, & D. Schifter (Eds.), *A research companion to principles and standards for school mathematics*. NCTM.

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92*(3), 170–181. https://doi.org/10.1080/00029890.1985.11971572

Lockwood, E. & De Chenne, A. (2020). Using conditional statements in Python to reason about sets of outcomes in combinatorial problems. *International Journal of Research in Undergraduate Mathematics Education, 6,* 303–346. doi:10.1007/s40753-019-00108-2

National Council of Teachers of Mathematics. (2000). *Principles and standards for school mathematics*. NCTM.

Rasmussen, C., & King, K. (2000). Locating starting points in differential equations: A realistic mathematics approach. *International Journal of Mathematical Education in Science and Technology, 31,* 161–172. https://doi.org/10.1080/002073900287219

Rasmussen, C., Zandieh, M., King, K., & Teppo, A. (2005). Advancing mathematical activity: A practice-oriented view of advanced mathematical thinking. *Mathematical Thinking and Learning, 7*(1), 51–73. https://doi.org/10.1207/s15327833mtl0701_4

Schwank, I. (1993). On the analysis of cognitive structures in algorithmic thinking. *Journal of Mathematical Behavior, 12*, 209–231. https://doi.org/10.1007/978-3-662-11334-9_22

Stephens M (2018) Embedding algorithmic thinking more clearly in the mathematics curriculum. In: Shimizu Y, Withal R (eds) Proceedings of ICMI study 24 School mathematics curriculum reforms: challenges, changes and opportunities. University of Tsukuba, pp 483–490.

Stephens, M., & Kadijevich, D.M. (2020). Computational/Algorithmic Thinking. In: Lerman S. (eds) Encyclopedia of Mathematics Education. Springer. https://doi.org/10.1007/978-3-030-15789-0_100044

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A, 366*, 3717-3725.