CASE STUDY

Embedding algorithmic thinking into 1st year undergraduate programmes

David O'Sullivan, Department of Mathematics, The London School of Economics and Political Science (LSE), London, United Kingdom. Email: d.r.osullivan@lse.ac.uk

Abstract

We describe the embedding of algorithms and algorithmic thinking into a first-year undergraduate pure mathematics course at the London School of Economics (LSE). Responding to growing demand for computational thinking skills, we redesigned 'Introduction to Abstract Mathematics' to integrate algorithmic approaches throughout the curriculum rather than as an isolated topic. Using the Knowledge, Skills and Behaviours (KSBs) framework adapted from degree apprenticeships, we developed learning outcomes that distinguish knowing algorithms from thinking algorithmically. Our approach reformulates traditional proofs using constructive and algorithmic methods, introduces loop invariants alongside mathematical induction, and shows how algorithms can both prove mathematical statements and have their own properties proven.

Keywords: Algorithmic thinking, threshold concepts, curriculum design, computational thinking, learning outcomes.

1. Introduction

Algorithms shape everyday life, from social media recommendation systems to financial trading platforms (Cormen et al., 2022; O'Neil, 2016). Meanwhile, the graduate job market is more competitive than ever, with employers increasingly seeking strong analytical and computational thinking skills (High Fliers Research, 2024; CBI/Pearson, 2023). Computational thinking has become essential across disciplines, and embedding it in mathematics education can enhance students' problem-solving and deepen their understanding of mathematical structures (Lockwood et al., 2016; Weintrop et al., 2016).

This work sits within a broader movement to modernise mathematics education for the 21st century. The increasing mathematisation of social sciences, particularly at LSE where economics, finance, and data science dominate, demands that pure mathematics courses evolve while maintaining their foundational role. Our approach addresses this challenge by treating algorithmic thinking not as an addition to the curriculum, but as a fundamental way of engaging with mathematical concepts.

Our intervention has three key features:

1. We integrate algorithmic thinking throughout the course rather than as an isolated topic;
2. We maintain focus on mathematical reasoning without requiring specific programming skills;
3. We explicitly distinguish knowledge of algorithms from the development of algorithmic thinking as a mathematical practice.

2. Theoretical Framework

2.1. Algorithms as threshold concepts

Our approach is grounded in the threshold concepts framework developed by Meyer and Land (2003, 2006). Threshold concepts are transformative ideas that, once understood, fundamentally change how students perceive and practice within a discipline. They are characterised as being

transformative, irreversible, integrative, bounded, and often troublesome for learners. We propose that algorithmic thinking represents a threshold concept in mathematics education for several reasons:

1. Transformative: Understanding algorithms transforms how students approach existence proofs and constructions; once they grasp that proving something exists can be achieved by constructing it algorithmically, their whole approach to such problems changes;
2. Irreversible: Once acquired, algorithmic thinking is difficult to unlearn; students report being unable to view mathematical problems without considering algorithmic solutions;
3. Integrative: Algorithmic thinking connects previously disparate areas; number theory, linear algebra, and group theory share common algorithmic threads that expose hidden connections;
4. Bounded: Algorithmic thinking has clear boundaries within mathematics, distinguishing it from both pure abstraction and mere calculation.

Framing algorithmic thinking as a threshold concept has direct consequences for teaching. Because the concept is *integrative* (drawing together number theory, linear algebra, and group theory) it is poorly served by isolated treatment - its connective power becomes visible only when encountered repeatedly across topics. And because crossing the threshold is *transformative* and often *troublesome*, students need sustained, scaffolded exposure rather than a single dedicated unit. These observations motivate the central design decisions of the next section: distinguishing knowing algorithms from thinking algorithmically, and embedding the latter throughout the course rather than as a standalone topic.

2.2. Constructivism and algorithmic thinking

Our pedagogical approach is rooted in constructivist learning theory (Piaget, 1952; Vygotsky, 1978), which holds that learners actively build knowledge through experience and reflection rather than receiving it passively. Algorithmic thinking aligns naturally with this: when students develop algorithms, they literally construct mathematical objects. The Extended Euclidean Algorithm, for example, does not merely prove Bézout's Identity exists, it constructs the coefficients. Such active construction helps students internalise abstract concepts through concrete manipulation.

Following Vygotsky's Zone of Proximal Development (ZPD), we use algorithms to scaffold learning. Their step-by-step nature provides intermediate stages between what students can do independently and what they can achieve with guidance; loop invariants, for instance, scaffold the transition from computational verification to formal proof.

The integration of algorithmic thinking supports movement through Bloom's Revised Taxonomy (Anderson and Krathwohl, 2001), as presented in Table 1.

Notably, algorithmic thinking provides concrete pathways to higher-order skills: creating an algorithm (the highest level) becomes accessible earlier than creating proofs, giving students success experiences that build confidence and motivation.

Table 1. Bloom's level, traditional approach and algorithmic enhancement

Bloom's Level	Traditional Approach	Algorithmic Enhancement
Remember	Memorise theorem statements	Recall algorithm steps
Understand	Explain proof logic	Trace algorithm execution
Apply	Use theorems in problems	Execute algorithms on examples
Analyse	Decompose proof structure	Analyse algorithm complexity
Evaluate	Assess proof validity	Compare algorithm efficiency
Create	Construct new proofs	Design new algorithms

3. Learning Outcome Design

Building on this foundation, we adopted an approach to learning outcomes taken from degree apprenticeships: Knowledge, Skills and Behaviours (KSBs) (Institute for Apprenticeships and Technical Education, 2023). The KSB framework aligns well with the Quality Assurance Agency (QAA) Subject Benchmark Statement for Mathematics, Statistics and Operational Research (QAA, 2023), which emphasises both subject knowledge and the development of mathematical thinking skills.

The KSB framework maps naturally onto our theoretical foundations:

- Knowledge aligns with lower levels of Bloom's Taxonomy (remember, understand);
- Skills correspond to middle levels (apply, analyse);
- Behaviours represent higher levels (evaluate, create) and the transformative aspect of threshold concepts.

Thinking about the introduction of the topic in this way led to two key observations:

1. 'Algorithms' and 'algorithmic thinking' are not the same thing - one represents knowledge while the other represents a skill that must be developed through practice;
2. The development of the 'skill' aspect would be best served by embedding algorithms across the whole term rather than treating them as an isolated topic.

Students learn best, and skills develop more deeply, when presented in the context in which they are to be used (Brown, Collins and Duguid, 1989; Hallam and Ireson, 2003). This is the standpoint taken in arguing for authentic assessment (Wiggins, 1990) and is consistent with situated learning theory (Lave and Wenger, 1991).

Considering the course's immediate and ongoing aims, we produced the following learning outcomes:

1. Define and explain basic concepts and propositions in the theory of algorithms;
2. Apply algorithmic techniques to prove mathematical statements;
3. Analyse and prove properties of algorithms including correctness and complexity;
4. Employ algorithmic thinking to approach computational problems and construct proofs.

3.1. Knowledge

The knowledge students should acquire is captured in learning outcome 1: a carefully selected subset of the core concepts and theory of algorithms, presented as four fundamental questions we can ask of any algorithm:

1. Does the algorithm terminate?
2. Does the algorithm give the correct answer? If so, when?
3. How fast does the algorithm converge to the correct answer?
4. How many operations does the algorithm take?

As this is primarily a mathematics course, we deliberately limited asymptotic analysis to big-O notation, excluding other measures of complexity such as Θ and Ω. We omitted data structures and algorithm paradigms (divide-and-conquer, greedy methods, etc.) entirely, as these are covered in a more specialised 'Algorithms and Data Structures' course available in later years.

3.2. Skills

Learning outcomes 2 and 3 cover the two complementary skills identified in the design phase. The first is employing algorithms as tools to prove mathematical statements or solve problems.

Students learn to:

- Use algorithms to demonstrate existence (e.g., proving a solution exists by providing an algorithm that finds it);
- Apply algorithmic techniques to prove mathematical statements (using named algorithms to prove known properties of mathematical objects);
- Leverage algorithmic reductions to establish relationships between problems.

The second skill is correctness proofs and complexity analysis for known mathematical algorithms. We selected algorithms that arise naturally in the course's topics:

- Number theory: Euclid's algorithm and the Chinese Remainder Theorem;
- Group theory: Burnside's Lemma for counting orbits;
- Linear algebra: Basis Extension Lemma and Gram-Schmidt orthogonalisation process.

Students are introduced to loop invariants (Hoare, 1969) and their relationship to mathematical induction, seeing how inductive proofs arise when analysing recursive algorithms. For complexity analysis, they learn to prove worst-case time bounds using techniques familiar from their mathematical training.

3.3. Behaviour

Learning outcome 4 assesses students' ability to use algorithms within their toolbox of proof strategies. This is the threshold crossing: students internalise algorithmic thinking as a fundamental mathematical practice. They learn to:

- Consider algorithmic approaches when faced with existence proofs;
- Use computational thinking to structure mathematical arguments;
- Recognise patterns that suggest particular algorithmic approaches.

In doing so, they develop intuition about what makes problems computationally hard or easy, preparing them for more advanced study in both pure and applied mathematics. This behavioural

change is the transformative aspect of the threshold concept, fundamentally altering how students approach mathematical problems.

4. Example: The Euclidean Algorithm and Extended Euclidean Algorithm

The Euclidean Algorithm gives students their first exposure to a loop invariant, illustrating the threshold concept in action. Loop invariants were introduced by Hoare in his work on program correctness (Hoare, 1969; see also Floyd, 1967). Formally, a loop invariant is a property that holds before and after each iteration of a loop, serving as the inductive hypothesis in correctness proofs: by maintaining it throughout execution, we guarantee the desired outcome on termination.

This example demonstrates our scaffolding principle: students build from concrete computation to abstract reasoning through the intermediate step of algorithmic analysis.

Having been shown an example calculation of $\gcd(a, b)$, the greatest common divisor for two natural numbers a and b, students are presented with the algorithm in Figure 1.

Input: Integers a, b with $a > b > 0$
Output: $gcd(a, b)$
function GCD(a, b)
  $r \leftarrow a \bmod b$
  while $r \neq 0$ **do**
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a \bmod b$
  end while
  return b
end function

Figure 1. Algorithm presented to students for the calculation of $\gcd(a, b)$ for two natural numbers a and b

We make and prove two claims, covering learning outcomes 2 and 3:

1. The algorithm terminates;
2. The algorithm is correct.

The proof of the first claim is a standard one, invoking the well-ordering principle that is part of the axiomatisation of the natural numbers. At each iteration, b strictly decreases (since $0 \leq r < b$), and since b is a non-negative integer, it must eventually reach 0. This connects the other fundamentals of the course (axiomatisation, abstraction) with the study of algorithms.

The proof of the second claim is more interesting. This can, of course, be approached by induction. But alternatively, correctness can be proved simply by proving that the greatest common divisor is a loop invariant in this algorithm:

Theorem: Let a, b be non-negative integers such that $a > b > 0$. Then

$$gcd(a, b) = gcd(b, r) \text{ where } r = a \ (mod\ b).$$

Since students already know proof by induction, the two strategies can be compared directly: the loop-invariant proof is markedly more succinct and intuitive.

Having verified termination and correctness, students meet complexity analysis, a skill that bridges pure mathematics and computer science. We present it in stages, from intuitive observation to formal proof.

The intuitive observation is that each iteration reduces the problem size: after two iterations, at least one number has halved. This leads to the formal result:

Theorem: The Euclidean algorithm has time complexity $O(\log \min(a, b))$.

The number of iterations of the Euclidean algorithm is at most $\lfloor \log_{\varphi}(\sqrt{5} \cdot \min(a, b)) \rfloor + 1$. Since each iteration involves one division operation (which takes $O(\log a \cdot \log b)$ bit operations), the total complexity is $O(\log \min(a, b))$ iterations.

This leads naturally into the study of Bézout's Identity:

Theorem: For any positive integers a and b, there exist integers x and y such that

$$gcd(a, b) = ax + by.$$

Rather than a non-constructive existence proof, we prove this by providing an algorithm that explicitly constructs the coefficients x and y. The Extended Euclidean Algorithm maintains the same $O(\log \min(a, b))$ complexity while additionally tracking the linear combination coefficients, demonstrating that constructive proofs need not sacrifice efficiency.

5. Implementation Strategy

Having decided to embed algorithmic thinking throughout the course, we updated the syllabus as follows:

1. Reformulate mathematical arguments algorithmically. For some results we replaced non-constructive proofs with constructive ones. For example, demonstrating the Chinese Remainder Theorem through an explicit algorithm that constructs the solution rather than proving existence abstractly. For others, we replaced inductive proofs with loop invariants;
2. Create dedicated time for algorithmic study. We replaced the worked examples in the weekly 'examples lecture' with weekly 'deep-dives' into that week's main algorithm(s);
3. Provide context from the start. We introduced the motivation for and general principles of algorithms in the first lecture of term;
4. Formalise the theory at the appropriate point. In the final lecture of term we treated algorithms formally (definitions, abstraction, and results on asymptotic analysis) only once their full context had been established.

6. Transferability

Although this intervention was developed for a cohort with strong incoming mathematical preparation, we believe its core design transfers to settings where students are less mathematically proficient, and may even suit them better. It is worth separating the portable elements from the local conditions that enabled them. The portable elements are the KSB framing, the distinction between knowing algorithms and thinking algorithmically, the four organising questions, embedding algorithmic thinking throughout rather than in isolation, and using constructive proofs as scaffolding toward abstract reasoning.

For such a cohort, the constructive route is arguably an advantage rather than an obstacle. Building an object explicitly offers a concrete foothold before the leap to non-constructive abstraction,

consistent with the ZPD scaffolding described above. Adapting the approach would likely involve slower pacing, lightening or omitting the formal final-lecture treatment of asymptotic analysis, choosing gentler exemplar algorithms and proofs, and adjusting assessment accordingly. A direct evaluation in such a setting would be a valuable next step.

7. Conclusion

By adopting a KSB framework and integrating algorithmic approaches throughout the curriculum, we have created a course that maintains mathematical rigour while developing the computational thinking skills essential for modern graduates.

While formal evaluation is planned for the next academic year, the 2024/25 delivery was a first implementation rather than a study designed to gather data, and we report no systematic findings here. Course feedback forms drew too few responses to support meaningful analysis. We can offer only the informal observation that students engaged readily with the material, and several voluntarily explored and implemented the algorithms beyond what was required — but we present this as an impression to be tested, not as evidence.

The intervention shows that pure mathematics courses can incorporate algorithmic thinking while keeping their essential character, giving students additional tools for mathematical reasoning and preparing them for the increasingly algorithmic nature of academic research and professional practice.

8. References

Anderson, L.W. and Krathwohl, D.R, 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman.

Brown, J.S., Collins, A. and Duguid, P., 1989. *Situated cognition and the culture of learning*, Educational Researcher, 18(1), pp. 32-42.

CBI/Pearson, 2023. *Education and Skills Survey 2023*. London: Confederation of British Industry.

Collins, A., Brown, J.S. and Newman, S.E., 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L.B. Resnick, ed. *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum. pp. 453-494.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C., 2022 *Introduction to Algorithms*. 4th edn. Cambridge, MA: MIT Press.

Evans, J.S.B. and Stanovich, K.E., 2013. *Dual-process theories of higher cognition: Advancing the debate*. Perspectives on Psychological Science, 8(3), pp. 223-241.

Floyd, R.W., 1967. *Assigning meanings to programs*. Proceedings of Symposia in Applied Mathematics, 19, pp. 19-32.

Hallam, S. and Ireson, J., 2003. *Secondary school teachers' attitudes towards and beliefs about ability grouping*. British Journal of Educational Psychology, 73(3), pp. 343-356.

Henderson, P.B., Cortina, T.J. and Wing, J.M., 2007. *Computational thinking*. ACM SIGCSE Bulletin, 39(1), pp. 195-196.

High Fliers Research, 2024. *The Graduate Market in 2024: Annual Review of Graduate Vacancies & Starting Salaries*. London: High Fliers Research Limited.

Hoare, C.A.R., 1969. *An axiomatic basis for computer programming*. Communications of the ACM, 12(10), pp. 576-580.

Institute for Apprenticeships and Technical Education, 2023. Developing Knowledge, Skills and Behaviours (KSBs). Available at: https://www.instituteforapprenticeships.org [Accessed: December 2025].

Kahneman, D., 2011. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux.

Lave, J. and Wenger, E., 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press.

Lockwood, E., DeJarnette, A.F. and Thomas, M., 2016. *Computing as a mathematical disciplinary practice*. Journal of Mathematical Behavior, 41, pp. 171-183.

Meyer, J.H.F. and Land, R., 2003. Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. In C. Rust, ed. *Improving Student Learning: Improving Student Learning Theory and Practice – Ten Years On*. Oxford: OCSLD, pp. 412-424.

Meyer, J.H.F. and Land, R., 2006. *Overcoming Barriers to Student Understanding: Threshold Concepts and Troublesome Knowledge*. London: Routledge.

O'Neil, C., 2016. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. New York: Crown.

Piaget, J., 1952. *The Origins of Intelligence in Children*. New York: International Universities Press.

QAA, 2023. *Subject Benchmark Statement: Mathematics, Statistics and Operational Research*. Gloucester: Quality Assurance Agency for Higher Education.

Vygotsky, L.S., 1978. *Mind in Society: The Development of Higher Psychological Processes*. Cambridge, MA: Harvard University Press.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. and Wilensky, U., 2016. *Defining computational thinking for mathematics and science classrooms*. Journal of Science Education and Technology, 25(1), pp. 127-147.

Wiggins, G., 1990. *The case for authentic assessment*. Practical Assessment, Research, and Evaluation, 2(2).