Uluslararası İleri Doğa Bilimleri ve Mühendislik Araştırmaları Dergisi
Sayı 8, S. 110-117, 3, 2024

Araştırma Makalesi

International Journal of Advanced Natural Sciences and Engineering Researches
Volume 8, pp. 110-117, 3, 2024

Research Article

https://as-proceeding.com/index.php/ijanser
ISSN: 2980-0811

Creative programming. Assessing and improving mathematical algorithms with students

Robert Kosova[1], Shkelqim Hajrulla[2], Anna Maria Kosova[3], Asad Kodra[4], Ali Xhafa[5], Igli Kapxhiu[5], Renato Psatha[5]

[1]*Department of Mathematics. Faculty of IT. University "A. Moisiu" Durres. Albania*
[2]*Computer Engineering Department. Epoka University. Tirana. Albania.*
[3]*Department of Computer Science. Faculty of IT. University "A. Moisiu" Durres. Albania.*
[4]*High school "Rilindja". Durres. Albania.*
[5]*Department of Mathematics. Faculty of IT. University "A. Moisiu" Durres. Albania*

Email of the corresponding author: robertkosova@uamd.edu.al

(Received:18 April 2024, Accepted: 25 April 2024)

(2nd International Conference on Scientific and Innovative Studies ICSIS 2024, April 18-19, 2024)

ATIF/REFERENCE: Kosova, R., Hajrulla, S., Kosova, A. M., Kodra, A., Xhafa, A., Kapxhiu, I. & Psatha, R. (2024). Creative programming. Assessing and improving mathematical algorithms with students. *International Journal of Advanced Natural Sciences and Engineering Researches*, 8(3), 110-117.

Abstract **–**Mathematics and algorithms have a common history dating back to ancient times. Since the beginning of mathematics, with the theorems, concepts, and definitions of mathematics, we also find algorithms that accompany many of the oldest proofs of mathematics, such as Euclid's Algorithm, the Sieve of Eratosthenes, etc. Mathematics and algorithms are two highly intertwined disciplines that form the backbone of today's modern science and technology. While mathematics provides the theoretical framework for understanding abstract concepts and solving complex problems, algorithms provide practical methods for applying mathematical ideas and solving real-world problems efficiently. Nowadays, with the rapid development of technology, the advancement of computing skills, and the use of artificial intelligence, mathematics is helping the development of algorithms, and algorithms are also helping mathematicians understand the most complex and difficult mathematical problems, including working on conjectures. In the process of developing a computer program, for example, Python, it is very important for students to understand that everything starts with math and reasoning to reach a clear and complete understanding of the problem, then the process of solving the problem and writing codes in any language, Python, Java, C++, Basic, etc. In this article, we will consider one of the oldest and most popular problems in mathematics, the Euclidean algorithm, solved in different Python programs to check the complexity time for each and encourage students to do creative programming.

Keywords – Algorithms, Optimization, Improving, Mathematics, Numbers, Primes.

I. INTRODUCTION

The history of mathematics and algorithms is deeply intertwined, with each discipline influencing and shaping the other over millennia of human civilization. The development of modern computing in the 20th century marked a significant turning point in the relationship between mathematics and algorithms. At their core, mathematics and algorithms share fundamental principles and concepts in their study and practice.

Algorithms have been used successfully in many math problems that seem impossible to solve theoretically. Many computational calculations have provided evidence for many open mathematical conjectures such as the Collatz conjecture, the twin prime conjecture, the perfect number conjecture, and more others [1-3]. There are also cases where computational calculations have led to the rejection or refutation of conjectures such as the Euler's Sum of Powers Conjecture, the Euler's Prime Generating Polynomial, the primality of Euclid's numbers, [4].

When evaluating algorithms, several parameters can be considered to assess their effectiveness. These parameters help to understand the performance, efficiency, and practicality of the algorithm [5-6].

1. Time complexity refers to the amount of time taken by an algorithm to complete its execution as a function of the input size. It provides insight into how the algorithm scales with increasing input sizes. Algorithms with lower time complexity are generally considered more efficient [7].

2. Space complexity measures the amount of memory required by an algorithm to execute as a function of the input size. It indicates how much memory the algorithm consumes during its execution. Algorithms with lower space complexity are preferred, especially when dealing with large input sizes or limited memory resources.

3. Accuracy refers to the algorithm's ability to correctly identify prime numbers and distinguish them from composite numbers. An accurate algorithm should correctly identify all prime numbers within the specified range without producing false positives or false negatives.

4. Scalability assesses how well the algorithm performs as the input size increases. A scalable algorithm should maintain reasonable performance even for large input sizes without significantly increasing execution time or memory usage.

5. Ease of Implementation evaluates how straightforward it is to implement and use the algorithm in practical applications. An algorithm that is easy to understand, implement, and integrate into existing systems is preferable.

6. Robustness measures the algorithm's resilience to various edge cases, input types, and potential errors. A robust algorithm should handle different scenarios gracefully and produce consistent results across different inputs.

7. Versatility assesses the algorithm's flexibility and applicability to different use cases and scenarios. A versatile algorithm should be suitable for a wide range of applications and capable of adapting to different requirements and constraints.

8. Parallelization: For some applications, the ability to parallelize the algorithm across multiple processors or threads can significantly improve performance. Evaluating whether the algorithm supports parallel execution and how effectively it utilizes parallel resources can be important, especially for high-performance computing environments [8-10].

9. Determinism vs. Probabilism: Some algorithms, such as deterministic primality tests like the AKS test, provide deterministic results, while others, like probabilistic tests such as the Miller-Rabin test, offer probabilistic results. Deterministic algorithms guarantee correctness but may have higher time complexity, while probabilistic algorithms provide faster execution but with a small probability of error [11].

Input Size: This represents the amount of data the algorithm needs to process. It's often denoted by the symbol "n.".

Operations: These are the basic instructions the algorithm performs, like comparisons, calculations, or loops. A special notation, O(n), is used to express time complexity. It captures the dominant factor affecting execution time as the input size grows.

There are different categories of time complexity based on how the number of operations grows with the input size:

Constant Time (O(1)): The number of operations remains constant regardless of the input size. This is ideal but uncommon for algorithms processing large datasets.

Logarithmic Time (O(log n)): The number of operations grows logarithmically with the input size. This means the execution time increases slowly, even for massive inputs.

Linear Time (O(n)): The number of operations grows proportionally to the input size. This is a good balance between efficiency and simplicity.

Quadratic Time (O(n^2)): The number of operations grows quadratically with the input size. This means even a moderate increase in input size can significantly slow down the algorithm.

Exponential Time (O(c^n)): The number of operations grows exponentially with the input size (where c is a constant greater than 1). These algorithms become very slow for even small inputs and are generally undesirable, Table 1.

Table 1. Time complexity to assess algorithms

	Name	Time Complexity
1	Constant Time	O(1)
2	Logarithmic Time	O(log n)
3	Linear Time	O(n)
4	Quasilinear Time	O(n log n)
5	Quadratic Time	O(n^2)
6	Exponential Time	O(c^n)
7	Factorial Time	O(n!)

II. MATERIALS AND METHOD

The famous Euclidean algorithm is chosen to be studied and analyzed by the students. The purpose is to analyze the present algorithms and Python programs on the Internet and select the ones with the minimum run time. Meanwhile, students are encouraged to try to improve the programs to improve their running time [12]. The purpose is to engage the students to do more than just learn and write codes to execute programs or calculate to solve simple math problems [13].

Creative programming in mathematical algorithms involves applying imaginative and unconventional approaches to solving mathematical problems computationally. This encompasses a wide range of mathematical domains, including but not limited to arithmetic, algebra, calculus, geometry, number theory, and optimization [14].

Innovative Problem Solving: Creative programmers develop new algorithms or adapt existing ones to solve mathematical problems in unique ways. This might involve combining different mathematical techniques, devising new heuristics, or introducing innovative data structures [15].

Efficiency and Optimization: Finding creative ways to optimize mathematical algorithms for improved efficiency or performance is crucial. This could involve reducing time complexity, minimizing memory usage, or enhancing numerical stability [16].

Exploration of Alternative Approaches: Rather than relying solely on conventional methods, creative programmers explore alternative approaches to solving mathematical problems. This might involve unconventional problem formulations or leveraging insights from other disciplines[17].

Visualization and Interpretation: Creative programming in mathematical algorithms often involves visualizing mathematical concepts to gain deeper insights or to communicate complex ideas more effectively. This can include generating graphical representations of data, geometric shapes, or mathematical functions[18].

Symbolic and Numerical Computation: Creative programmers may combine symbolic and numerical computation techniques to solve mathematical problems more efficiently. This could involve symbolic manipulation of algebraic expressions, coupled with numerical methods for approximation or optimization [19].

Interdisciplinary Applications: Creative programming in mathematical algorithms often extends beyond pure mathematics to interdisciplinary applications. This might involve applying mathematical concepts to fields such as physics, engineering, computer science, finance, or biology [20].
Experimentation and Innovation: Experimenting with different mathematical algorithms, exploring new problem-solving strategies, and pushing the boundaries of mathematical knowledge are essential aspects of creative programming in this domain [21-22].

The Eucledian algorithm:
The Eucledian algorithm is attributed to the ancient Greek mathematician Euclid, who lived around 300 BCE. The algorithm, known as the Euclidean algorithm, is used to compute the greatest common divisor (GCD) of two integers. Euclid described this algorithm in his work "Elements," specifically in Book VII, Proposition 2, where he outlined the method for finding the largest number that divides two given numbers without leaving a remainder.
Several definitions and concepts are taken from the book to help understand the algorithm.
Proposition 1
Two unequal numbers are laid down, and the lesser is continually subtracted, in turn, from the greater. If the remainder never measures the number preceding it until a unit remains, then the original numbers will be prime to one another, figure 1.

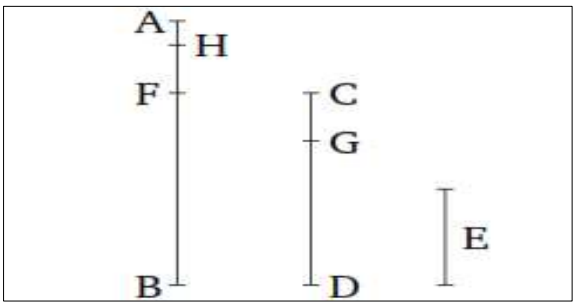

Figure 1. GCD (AB, CD)

Proposition 7.1.
For two [unequal] numbers, AB and CD, the lesser being continually subtracted, in turn, from the greater, let the remainder never measure the number preceding it, until a unit remains. I say that AB and CD are prime to one another—that is to say, that a unit alone measures both AB and CD. If AB and CD are not prime to one another, then some number will measure them. Let (some number) measure them, and let it be E. And let CD measuring BF leave FA less than itself, and let AF measuring DG leave GC less than itself, and let GC measuring FH leave a unit, HA. In fact, since E measures CD and CD measures BF, E thus also measures BF. And (E) also measures the whole of BA. Thus, (E) will also measure the remainder. AF. And AF measures DG. Thus, E also measures DG. And (E) also measures the whole of DC. Thus, (E) will also measure the remainder, CG. And CG measures FH. Thus, E also measures FH. And (E) also measures the whole of FA. Thus, (E) will also measure the remaining unit, AH, despite being a number. The very thing is impossible. Thus, some numbers do not measure both the numbers AB and CD. Thus, AB and CD are prime to one another. which is the very thing it was required to show.
Proposition 7.2.
To find the greatest common measure of two given numbers (which are not prime to one another).
Let AB and CD be the two given numbers that are not prime to one another. So it is required to find the greatest common measure of AB and CD. In fact, if CD measures AB, CD is thus a common measure of CD and AB, since CD also measures itself. And it is manifest that it is also the greatest common measure. For nothing greater than CD can measure CD. But if CD does not measure AB, then some number will remain from AB and CD, the lesser being continually subtracted, in turn, from the greater, which will measure the number preceding it. A unit will not be left. But if not, AB and CD will be prime to one another

[Prop. 7.1]. The very opposite thing was assumed. Thus, some number will remain, which will measure the number preceding it. And let CD measuring BE leave EA less than itself, and let EA measuring DF leave FC less than itself, and let CF measure AE. Therefore, since CF measures AE and AE measures DF, CF will thus also measure DF. And it also measures itself. Thus, it will also measure the whole CD. And CD measures BE. Thus, CF also measures BE. And it also measures EA. Thus, it will also measure the whole of BA. And it also measures CD. Thus, CF measures both AB and CD. Thus, CF is a common measure of AB and CD. So, I say that it is also the greatest common measure. If CF is not the greatest common measure of AB and CD, then some number that is greater than CF will measure the numbers AB and CD. Let it measure (AB and CD), and let it be G. And since G measures CD, and CD measures BE, G thus also measures BE. And it also measures the whole of BA. Thus, it will also measure the remainder, AE. And AE measures DF. Thus, G will also measure DF. And it also measures the whole of DC. Thus, it will also measure the remainder CF; the greater the measurement, the lesser. The very thing is impossible. Thus, some number that is greater than CF cannot measure the numbers AB and CD. Thus, CF is the greatest common measure of AB and CD. which is the very thing it was required to show.

Corollary

So, it is manifest from this that if a number measures two numbers, then it will also measure their greatest common Measure, which is the very thing it was required to show.

III. RESULTS AND DISCUSSION

The Eucledian algorithm:

The time complexity of the basic Euclidean algorithm is O(log(min(a,b))). The space complexity of the basic Euclidean algorithm is O(1), meaning it requires constant space regardless of the input size. The time complexity of the extended Euclidean algorithm is O(log(min(a,b))). The Extended Euclidean algorithm not only finds the greatest common divisor of two numbers but also computes the coefficients of Bézout's identity, which are used to express the GCD as a linear combination of the input numbers.

Basic algorithm:

Input a, b>a,
If b<a, exchange a and b.
b=b-a,
If b>a, then b=b-a.
Repeat until one is a multiple of the other.

Extended algorithm:

Input two positive integers, a and b.
Output The greatest common divisor is gcd(a,b).
If a<b, exchange a and b.
If a=0, then gcd(a,b)=b.
If b=0, then gcd(a,b)=a.
Divide a by b and get the remainder, r.
If r = 0, report b as the GCD of a and b,
Replace a by b, and replace b by r.
Return to the previous step.
a=k1a+r1,
a=k2r1+r2,
r1=k3.r2+r4,
……..
Gcd(a,b)=rn, when rn+1=0

Divisors method:
Input a, b.
If a=0, (b=0), then gcd=b, (gcd=a).
If a=1, (b=1), then (gcd=1),
Find the minimum (a,b).
Check all positive integers from 1 to the minimum.
Find the common divisors d of a and b.
If d is a common divisor, then gcd=d.
Continue until the end; the last greatest common divisor is gcd(a, b).
Among many other mathematical problems, such as the GCD, finding primes using the sieve of Eratosthenes, finding perfect numbers, etc., the Eucledian algorithm was chosen to be discussed with students. The students were encouraged to read and learn about the algorithms on the internet and the codes in C++, Java, Python, Basic, etc. The students were also asked to write their own codes, basic and simple, as a faithful translation of the algorithms [20]. The codes were written in Python for the three variables of finding gcd(a,b), the basic algorithm, the extended algorithm, and the divisors method. For small numbers, all the algorithms have a quasi-same run time. For very large numbers, the extended algorithm is much faster, Table 2.

Table 2. Running time of three algorithms

(a,b)	Gcd	Basic	Extended	Divisors method
(12,23)	1	4.65731 sec	2.81384 sec	3.43629 sec
(23,58)	1	5.24234 sec	3.14620 sec	3.09732 sec
(1234,754332)	2	8.56016 sec	8.17925 sec	12.4848 sec
(123456, 9854223654113)	8	26.0903 sec	8.81198 sec	8.47392 sec
(456851246, 9854223654113)	1	23.3109 sec	11.0191 sec	14.3578 sec

IV. CONCLUSION

Mathematical algorithms are important for solving math problems. For the math students, it starts by learning what has been achieved until now and then trying to improve it. Creative programming should be implemented for the students to encourage them to change, improve, and create their own programs, methods, and algorithms. That is the best way to make better algorithms in the future. The AKS algorithm is the best example of creative programming.
To encourage students to develop better solutions, the following strategies are necessary:
Problem-Based Learning: Present students with real-world problems or open-ended challenges that require mathematical reasoning to solve. Encourage them to explore different approaches and solutions, emphasizing creativity and critical thinking.
Project-Based Learning: Engage students in long-term projects that integrate mathematical concepts with other subjects or real-world scenarios. Projects could involve designing experiments, building models, or analyzing data, allowing students to see the practical applications of math.
Encourage Multiple Solutions: Instead of focusing solely on finding the "right" answer, encourage students to explore multiple solutions to a problem. Discuss the advantages and disadvantages of each approach and encourage peer-to-peer learning by sharing different strategies.
Promote mathematical modeling: Teach students how to create mathematical models to represent and solve real-world problems. Encourage them to make assumptions, gather data, and iterate on their models to improve accuracy and relevance.
Emphasize problem-solving strategies: Teach problem-solving strategies such as breaking down problems into smaller parts, looking for patterns, and making connections between different concepts. Provide

opportunities for students to practice these strategies through guided exercises and collaborative problem-solving activities.

Use Technology Wisely: Integrate technology tools such as graphing calculators, spreadsheets, and mathematical software into the curriculum. These tools can help students visualize concepts, explore mathematical relationships, and experiment with different approaches to problem-solving.

Cultivate a Growth Mindset: Foster a classroom culture that values effort, persistence, and learning from mistakes. Encourage students to embrace challenges, take risks, and view mistakes as opportunities for growth rather than failures.

Encourage Collaboration and Peer Learning: Create opportunities for students to work collaboratively, share ideas, and learn from each other. Peer-to-peer interactions can stimulate creativity, promote deeper understanding, and foster a supportive learning community.

Creative programming refers to the use of programming languages and computational tools to express creativity, generate artistic output, and solve problems in innovative ways. Creative programming encourages exploration, experimentation, and personal expression through code.

Expressive Coding: Creative programming emphasizes using code as a medium for self-expression and artistic exploration. Programmers leverage the flexibility and power of programming languages to create unique and visually engaging outputs.

Creative Coding Tools and Libraries: There are various programming environments, frameworks, and libraries specifically designed for creative programming, such as Processing, p5.js, open Frameworks, Cinder, and Max/MSP. These tools provide abstraction layers and APIs that simplify complex tasks and enable rapid prototyping of creative projects.

Cross-Disciplinary Exploration: Creative programming encourages interdisciplinary exploration by combining programming with other fields such as art, design, music, science, and education. It provides opportunities for individuals from diverse backgrounds to engage with technology creatively and develop innovative solutions to complex problems.

ACKNOWLEDGMENT

This publication was made possible with the financial support of the UAMD (University “A.Moisiu” Durres, Albania) project "Developing Programs and Applications (Windows and Android) for Mathematics Problems During the Teaching Process to enrich the teaching process and form a creative programming culture.” . The contents are the responsibility of the author; the opinion expressed in them is not necessarily the opinion of UAMD.

REFERENCES

[1] Kosova, R., Kapçiu, R., Hajrulla, S., & Kosova, A. M. (2023). A Review of Mathematical Conjectures: Exploring Engaging Topics for University Mathematics Students. *International Journal of Advanced Natural Sciences and Engineering Researches (IJANSER), 7*(11), 180–186. https://doi.org/10.59287/as-ijanser.581

[2] Kosova, R., Kapçiu, R., Hajrulla, S., & Kosova, A. M. (2023). The Collatz Conjecture: Bridging Mathematics and Computational Exploration with Python. *International Journal of Advanced Natural Sciences and Engineering Researches (IJANSER), 7*(11), 328–334. https://doi.org/10.59287/as-ijanser.637

[3] Kosova, R. (2023). Math conjectures with Python. In *Annual Internationale Conference of Public University “Kadri Zeka” in Gjilan–UKZ AC2023. P* (p. 40).

[4] Kosova, R., Bushi, F., Kapçiu, R., Cullhaj, F., & Kosova, A. M. (2024). A review of primarily tests and algorithms: Engaging students to code for mathematics . *International Journal of Advanced Natural Sciences and Engineering Researches, 8*(2), 182–195. Retrieved from https://as-proceeding.com/index.php/ijanser/article/view/1711

[5] Stephens, M., & Kadijevich, D. M. (2020). Computational/algorithmic thinking. *Encyclopedia of mathematics education*, 117-123.

[6] Agrawal, M., Kayal, N., & Saxena, N. (2004). PRIMES is in P. *Annals of mathematics*, 781-793.

[7] Zaka, O. (2022). Computing efficiently the weighted greatest common divisor. *arXiv preprint arXiv:2210.07961*.

[8] Kalluci, E., & Hoxha, F. (2015). The parallel computation of polynomial zeros in a cluster. In *The First European Conference on Physics and Mathematics* (pp. 17-23).

[9] Kapçiu, R., Hoxha, F., & Kalluci, E. Parallelized methods for solving polynomial equations.

[10] Kapçiu, R., & Hoxha, F. The Usage of OpenMp Platform in Solving Polynomial Equations.

[11] Goldenberg, E. P., & Carter, C. J. (2021). Programming as a language for young children to express and explore mathematics in school. *British Journal of Educational Technology*, *52*(3), 969-985.
[12] Kosova, R., Thanasi, T., Mukli, L., & Përllumbi, L. N. (2016). Traditional mathematics and new methods of teaching through programming together with students.
[13] Kynigos, C., & Diamantidis, D. (2022). Creativity in engineering mathematical models through programming. *ZDM–Mathematics Education*, *54*(1), 149-162.
[14] Kosova, A. G. R. The Performance of University Students and High School Factors. Statistical Analyses And ANCOVA.
[15] KALLUCI, E. (2015). New Root-Finding Methods for Nonlinear Equations. *American Academic & Scholarly Research Journal*, *7*(7).
[16] Sotirofski, K., Kukeli, A., & Kalemi, E. (2010). Challenges of teaching computer science in transition countries: Albanian university case. *Journal of College Teaching & Learning (TLC)*, *7*(3).
[17] Stana, P. E. A., Toti, P. E. L., Kosova, P. R., & Prodani, P. F. (2023). The Future of Durrës-Smart University&Smart City. *Journal of Survey in Fisheries Sciences*, *10*(2S), 1971-1981.
[18] Diab, A. (2021). Development of sieve of Eratosthenes and sieve of Sundaram's proof. *arXiv preprint arXiv:2102.06653*.
[19] Gjana, A., & Kosova, R. Traditional Class, and Online Class Teaching. Comparing the Students Performance Using ANCOVA. *Journal of Multidisciplinary Engineering Science and Technology (JMEST)*, 14806-14811.
[20] Gjana, A., & Kosova, R. Traditional Class, and Online Class Teaching. Comparing the Students Performance Using ANCOVA. *Journal of Multidisciplinary Engineering Science and Technology (JMEST)*, 14806-14811.
[21] Hajrulla, S., Demir, T., Bezati, L., & Kosova, R. (2023). The impact of constructive learning applied to the teaching of numerical methods. *CONSTRUCTIVE MATHEMATICS: FOUNDATION AND PRACTICE*, 39.
[22] Hajrulla, S., Abou Jaoudeh, G. M., Kosova, R., & Isufi, H. (2024). Optimization Problems through Numerical Methods and Simulations.