SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V12I11P103

Volume 12 Issue 11, 15-17, November 2025
© 2025 Seventh Sense Research Group®

Original Article

An Optimized Algorithm for the Prime Factorization Problem

Ayman Timjicht

Computer Science, Digital Social Commerce LTD, Covent Garden, London, United Kingdom.

[1]*Corresponding Author : aymantimjichtofficel@gmail.com*

Received: 21 September 2025 Revised: 29 October 2025 Accepted: 14 November 2025 Published: 30 November 2025

Abstract *- This article introduces a new scientific discovery in the field of computational number theory: an enhanced integer factorization method that unifies an optimized form of a prime generator algorithm with an advanced trial division framework. This integration produces a factorization technique that is both faster and more structurally elegant than conventional approaches. The discovery lies in demonstrating how a modern prime generator can be adapted and expanded to drive the factorization process with greater precision, reducing unnecessary computations and improving overall performance. The paper details the conceptual foundations of the method, outlines its computational advantages, and explains the process choices that make the algorithm both practical and theoretically elegant. The article describes the transformation of a simple prime generator algorithm into a solution for an NP problem, achieving optimized time and space efficiency.*

Keywords *- Algorithmic Complexity, Computational Number Theory, Prime Factorization, Prime Generation, Trial Division.*

1. Introduction

Prime factorization is the process of breaking down a composite number into its prime factors, revealing the unique building blocks of integers. While this concept seems straightforward, it plays a significant role in various fields, particularly in cryptography and number theory. Although it is easy to compute for small numbers, the prime factorization of large integers presents a significant challenge, leading researchers to classify it as a problem in NP (Nondeterministic Polynomial Time). This article explores the intricacies of prime factorization, its applications, and the implications of its computational complexity. As we delve into this fascinating topic, we will uncover why understanding prime factors is essential in both theoretical and practical realms. After examining prior studies, it becomes clear that an optimized prime generation strategy may enhance the practicality of traditional approaches and open pathways for further algorithmic refinement.

The aim of the present study is therefore to introduce an improved factorization method that incorporates prime generation algorithms with an optimized space-time complexity in a structured trial division framework, enabling faster prime generation, reduced redundancy, and more efficient extraction of prime components. This work is expected to benefit the computational mathematics community by offering a process that is both easy to implement and adaptable to cryptography and security of systems; the discoveries make the RSA algorithm vulnerable and not protected. In this article, we analyze the computational structure, correctness, and complexity of the approach.

2. Materials and Methods

This chapter outlines the standards and the methodology followed in the project

2.1. Hardware Standards

Standard desktop/server with a minimum 4 GB RAM and a modern CPU. An alternative is a High-performance workstation. Pros: Faster execution for large integers; Cons: Costly and unnecessary for educational scope.

2.2. The Methodology of Prime Factorisation

2.2.1. Generating All Primes Up to $\sqrt{N}$

To perform the prime factorization of a number N, the first step is to identify the prime numbers that compose the number. To do this, we should generate a list of prime numbers up to $\sqrt{N}$. I utilized the Sieve of Atkin for this purpose. The Sieve of Atkin is a method for generating a list of prime numbers.

However, we can also use other algorithms to determine prime numbers below N. These algorithms must be optimized for both space and time complexity. The complexity of the Sieve of Atkin is N. In the practical use, we calculate the prime numbers under $\sqrt{N}$ The complexity in our case is $\sqrt{N}$.

2.2.2. Prime Factorization Algorithm

This stage performs the core factorization by systematically dividing the target number by counting the exponent for prime divisors. Initialization: An empty data structure, factors (a map or dictionary), is initialized to store the prime factors and their corresponding exponents, {p: e}. A temporary variable n is initialized with the value of the input N. The algorithm proceeds by iterating through the list of primes p less $\left[\sqrt{N}\right]$+1 since the limit should be an integer, the nearest integer bigger $\sqrt{N}$ is $\left[\sqrt{N}\right]$+1.

The loop incorporates a stopping condition: if $p^2 >$ n, the iteration terminates. The reason is that the prime numbers less than the root of n are sufficient for the determination of the divisors of n. This is highly efficient because if the square of the current prime p exceeds the remaining value of n, then any further factors of n must be n itself (meaning n is prime). The factor extraction process involves initializing an exponent count to zero for each prime p in the list. Division is repeatedly performed until n is no longer divisible by p:

```
While n (mod p) = 0:
        n ← n//p
        count ← count + 1
```

Storage: If count > 0,

The prime factor p and its exponent count are recorded in the factors map. The final stage accounts for the possibility that the number N has a prime factor larger than √the square root of N. After the loop completes, the remaining value of n is inspected. If the residual value n > 1, it signifies that the remaining n is itself a significant prime factor of the original number N.(Since all factors ≤ √N have already been removed).

This prime factor is stored with an exponent of 1. The mathematical principle is that if n is not divisible by primes less than or equal to the root of N with $N \geq n$, then n is prime. For prime numbers, the factorisation of a prime number is the same number n or n^1.

3. Results and Discussion

This chapter outlines the process of prime factorization for the number N. We choose the number 120; the prime decomposition of the number involves multiple prime factors raised to various powers. The final implementation of the algorithm, written in Python code as an example of an academic coding language, the process can be written in multiple languages and for low-level languages guarding the same process, I executed the algorithm on a Google Cloud platform called Google Colab, providing a clear visualization of the results.

3.1. The Factors Map of the Algorithm

The factors map containing the complete prime factorization of N is returned.

Table 2. The factors map of the number 120

Step	Stage/Operation	Calculation	n (Remaining Value)	factors Map
1	Stage 1: Limit	L=$\left[\sqrt{120}\right]$+1=10+1=11	120	{}
2	Stage 1: Sieving	primes =[2,3,5,7,11]	120	{}
3	Stage 2: p=2	120/2=60; 60/2=30; 30/2=15. Count = 3.	15	{2:3}
4	Stage 2: p=3	15/3=5. Count = 1.	5	{2:3,3:1}
5	Stage 2: p=5	5/5=1. Count = 1.	1	{2:3,3:1,5:1}
6	Stage 2: p=7	7^2=49, 49$> n = 1$. Stop condition met.	1	{2:3,3:1,5:1}
7	Stage 3: Residual Check	n=1. No final factor needed.	1	{2:3,3:1,5:1}

3.2. Python Implementation

In this section, we present a code for the algorithm, written in Python, along with its implementation in Google Colab. The code :

```
def sieve_of_atkin(limit):
    if limit <= 2:
        return []
    is_prime = [False] * limit
    sqrt_lim = int(math.isqrt(limit)) + 1
    for x in range(1, sqrt_lim):
        for y in range(1, sqrt_lim):
            n = 4*x*x + y*y
            if n < limit and (n % 12 == 1 or n % 12 == 5):
                is_prime[n] = not is_prime[n]
            n = 3*x*x + y*y
            if n < limit and (n % 12 == 7):
                is_prime[n] = not is_prime[n]
            n = 3*x*x - y*y
        if x > y and n < limit and (n % 12 == 11):
            is_prime[n] = not is_prime[n]
    for r in range(5, sqrt_lim):
        if is_prime[r]:
            r2 = r*r
            for k in range(r2, limit, r2):
                is_prime[k] = False
    primes = []
    if limit > 2:
        primes.append(2)
    if limit > 3:
        primes.append(3)
    for i in range(5, limit):
        if is_prime[i]:
            primes.append(i)
    return primes
def prime_factors_with_powers(N):
    primes = sieve_of_atkin(int(math.isqrt(N)) + 1)
    factors = {}
    n = N
    For p in primes:
        if p*p > n:
            break
        count = 0
        while n % p == 0:
            n //= p
            count += 1
        if count > 0:
            factors[p] = count
    if n > 1:
        factors[n] = 1
    return factors
N = 120
factors = prime_factors_with_powers(N)
print(factors)
```

You can execute this algorithm interactively in Google Colab. The prime factorisation is the second Algorithm: https://colab.research.google.com/drive/1bH9RV43ZuRpEoy0fk51HFkDDreIQcaO7?usp=sharing

References

[1] Kadir Duman et al., “Enhancing Mechanical Properties of Polyurea through Cellulose Nano Crystals (CNF) Reinforcement,” *Scientific Bulletin of the University Politehnica of Bucharest*, vol. 87, no. 1, pp. 1-16, 2025. [Publisher Link]

[2] Mircea Ghidarcea, and Decebal Popescu, “Prime Number Sieving—A Systematic Review with Performance Analysis,” *Mathematics*, vol. 12, no. 4, pp. 1-20, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Muhammad Khoiruddin Harahap, and Nurul Khairina, “The Comparison of Methods for Generating Prime Numbers between The Sieve of Eratosthenes, Atkins, and Sundaram,” *SinkrOn*, vol. 3, no. 2, pp. 293-298, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[4] Richard Crandall, and Carl B. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, 2005. [Google Scholar] [Publisher Link]

[5] John M. Pollard, “A Monte Carlo Method for Factorization,” *BIT Numerical Mathematics*, vol. 15, no. 3, pp. 331-334, 1975. [Google Scholar] [Publisher Link]