Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Advances in Factoring and Primality Testing: From Classical to Quantum Algorithms

Anas A. Abudaqa

anas86@oc.edu.sa

Onaizah Colleges (OC)

Nujud Alyami

King Fahd University of Petroleum and Minerals

Mostefa Kara

University of El Oued

Farid Binbeshr

King Fahd University of Petroleum and Minerals

Muhammad Imam

King Fahd University of Petroleum and Minerals

Systematic Review

Keywords: Primality testing, Integer factoring, Quantum computing, Shor's algorithm, Post-quantum Cryptography, Computational Costs

Posted Date: July 21st, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10413688/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

ADVANCES IN FACTORING AND PRIMALITY TESTING: FROM CLASSICAL TO QUANTUM ALGORITHMS

Anas A. Abudaqa
Department of Computer Science
College of Engineering and Information Technology
Onaizah Colleges (OC)
Qassim, SA 56447
anas86@oc.edu.sa

Nujud Alyami
Department of Physics
College of Engineering and Physics
King Fahd University (KFUPM)
Dhahran, SA 31261
nujudaly@gmail.com

Mostefa Kara
LIAP Laboratory
University of El Oued
El Oued, Algeria 39000
karamostefa@univ-eloued.dz

Farid Binbeshr
IRC-ISS Research Center
King Fahd University (KFUPM)
Dhahran, SA 31261
farid.binbeshr@kfupm.edu.sa

Muhammad Imam
Department of Computer Engineering
King Fahd University (KFUPM)
Dhahran, SA 31261
mimam@kfupm.edu.sa

July 19, 2026

ABSTRACT

Many modern asymmetric encryption methods rely on prime numbers, as they have distinctive properties. For instance, the security of RSA cryptosystem relies on the computational difficulty of factoring a large composite number in its prime factors, a problem that remains challenging for classical computers but potentially solvable using quantum algorithms. On the other hand, generating large prime numbers is also challenging due to their irregular distribution among integers, necessitating the use of primality testing algorithms to verify candidate primes. In this paper, we intensively review and classify various classical and quantum algorithms for factorization and primality testing, highlighting their advantages, limitations, speed/accuracy tradeoffs, time complexities, along with a brief summary. Furthermore, we apply and compare these algorithms to gain practical insights and conduct a comprehensive performance comparison. The insights from this paper show that while quantum factoring algorithms, particularly Shor's algorithm and its refinements, have introduced significant advancements over their classical counterparts, quantum primality testing algorithms have not demonstrated comparable advantages.

Keywords Primality testing · Integer factoring · Quantum computing · Shor's algorithm · Post-quantum Cryptography · Computational Costs.

1 Introduction

The study of prime numbers dates back to around 300 B.C., with early contributions from ancient Greek mathematicians [1]. Substantial theoretical advances were sparse until the 17th century, notably with Fermat's Little Theorem [2], which laid foundational principles in number theory. Among the central problems related to prime numbers are integer factorization and primality testing. These problems are not only of theoretical interest but also serve as the backbone of modern public-key cryptography, such as the RSA cryptosystem [3] and Diffie-Hellman key exchange [4], which rely on the computational hardness of factoring large integers and solving discrete logarithms, respectively. If an efficient (i.e., polynomial-time) solution to the factoring problem were discovered, the security of RSA would be entirely compromised [5]. Currently, all known classical factoring algorithms run in sub-exponential time, supporting the belief that integer factorization is intractable for sufficiently large inputs [6].

Quantum factoring algorithms provide substantial improvements over classical factoring methods. Particularly, Shor's algorithm [7], introduced in 1994, demonstrated polynomial time complexity over quantum computers [8] for solving integer factoring and discrete logarithm problems, which are traditionally considered infeasible for classical computers. Accordingly, cryptographers have become increasingly concerned about advances in quantum computing, as the availability of a large-scale quantum computer would enable Shor's algorithm to break all widely used public-key cryptosystems (PKC) [9]. The exponential speedup introduced by Shor's algorithm has led to subsequent research [10, 11] to refine the algorithm by reducing resource requirements, such as qubit count and gate depth. Moreover, to demonstrate the algorithm's viability, experimental work has been conducted on smaller-scale quantum computers [12, 13, 14]. Shor's algorithm, which relies on the circuit model of quantum computing, is not the only paradigm proposed for solving the integer factorization problem. Another promising approach is adiabatic quantum computation [15]. In general, while classical factoring algorithms face scalability limitations for very large integers, quantum algorithms offer significant potential for addressing the factorization problem more efficiently. This potential has established quantum factoring as a highly active and critical area of research, with implications for cryptography and computational theory.

Primality testing also plays a crucial role in cryptography, especially in key generation, where large prime numbers are required. Generating primes over large intervals becomes computationally challenging. A common approach involves selecting a random or pseudorandom candidate and verifying its primality using a suitable primality test. Compared to factoring, quantum advancements in primality testing have not surpassed the efficiency of their classical counterparts. Classical algorithms remain computationally competitive for verifying large primes, leaving limited room for quantum superiority in this area. A recent milestone was achieved in October 2024, with the discovery of the largest known prime number, $2^{136,279,841} - 1$, by Great Internet Mersenne Prime Search (GIMPS) [16]. This record was set using classical primality testing methods such as the Lucas-Lehmer test [17], executed on a distributed computing network leveraging GPUs. The achievement highlights the ongoing relevance and power of classical approaches in specific computational tasks despite the rising prominence of quantum computing.

In this survey paper, we aim to deeply explore various classical and quantum algorithms for factorization and primality testing, highlighting their advantages, limitations, speed/accuracy tradeoffs, time complexities, and providing a brief summary. In particular, we focus on the area of quantum factoring algorithms, presenting a comprehensive view of theoretical advancements, practical implementations, and alternative approaches beyond Shor's traditional framework.

1.1 Related Work

In this section, we review various methods employed in the literature to evaluate factoring and primality testing algorithms.

K. Bimpikis et al. [18] survey four classical factoring algorithms: the Naive Trial Division, Pollard's Rho, Quadratic Sieve (QS), and Number Field Sieve (NFS). For each algorithm, it provides a summary of the algorithm, potential improvement, and time complexity. The last section of the paper is dedicated to a brief introduction of Shor's algorithm.

F. Boudot et al. [6] review all the classical (non-quantum) integer factoring algorithms, with a focus on their cryptanalysis effect on the current number-theoretic public key cryptosystems.

M. Ghidarcea et al. [19] focus, in their systematic survey, on discovering primes by solely sieving techniques.

L. Monier [20] compares the efficiency of the Miller-Rabin and Solovay-Strassen primality tests using a mathematical model, concluding that Miller-Rabin is superior. Similarly, R. Canfield [21] compares Solovay-Strassen, Miller, and AKS tests, providing practical implementations in Maple for various input sizes.

Other researchers have explored specific aspects of primality testing algorithms. R. Schoof [22] conducts a concise survey of four algorithms, detailing their construction, efficiency, and applications. M. Kida [23] extends the discussion by proposing variations of Fermat, Miller-Rabin, and Pocklington tests based on group schemes over finite fields. M. Perrenoud [24] offers a comparative analysis of randomized and deterministic algorithms, emphasizing the efficiency of AKS over Fermat's test within C++ implementations.

Building frameworks and performing practical comparisons have also been a focus. C. Duta et al. [25] propose a framework in C# to evaluate 14 algorithms, albeit with classification issues. Y. Kasarabada [26, 27] explores Java and Python implementations for the Baillie-PSW test, though their scope is limited to a single test.

In the quantum realm, papers like D. Bernstein et al. [28] and J. Li et al. [29] provide innovative methods that prioritize resource efficiency. However, these papers primarily explore isolated techniques rather than providing a holistic review.

Closely related to our work, N. Kunihiro [30] expands on resource estimation and experimental implementations of Shor's algorithm. The work delves into modular exponentiation methods, presenting constructions like R-ADD, GT-ADD, and Q-ADD, and provides precise resource requirements for factoring 768-bit and 2048-bit composite

numbers. In addition, he evaluates quantum circuits used in experiments targeting small composite numbers, such as 15 and 21, highlighting the challenges of using explicit order information in these setups. The work critiques the oversimplified implementations of Shor's algorithm, such as those discussed by A. Smolin et al. [31], emphasizing their limitations for general-purpose factoring. Compared to our survey, this work is narrowly focused on modular exponentiation and specific experimental implementations. While it provides a detailed resource breakdown and practical applications, it lacks a broader theoretical discussion and comparative analysis across classical, quantum, and hybrid factoring paradigms.

1.2 Contribution and Paper Organization

This paper addresses the gap introduced in the literature by:

- Firstly: presenting a novel classification of 11 classical and 4 quantum primality-testing algorithms and five classical factoring algorithms to examine advantages, limitations, and speed/accuracy tradeoffs.
- Secondly: This survey examines deeply the area of quantum factoring algorithms by providing a comprehensive overview of theoretical advances, practical implementations, and alternative algorithms beyond Shor's original framework.
- Thirdly: we address trade-offs in hybrid quantum-classical strategies and the implications for emerging quantum hardware, offering a broader scope of applicability and future directions.

The rest of the paper is organized as follows: Section 2 introduces a brief background and terminology. In Section 3, the taxonomy and survey of classical primality testing and factorizing algorithms are presented. In Section 4, the taxonomy of quantum primality testing and factorizing algorithms is illustrated. Section 5 is dedicated to a global discussion and analysis. Finally, we conclude our discussion in Section 6.

2 Background

Factoring and primality testing are two fundamental problems in computational number theory, each with distinct objectives and applications. Integer factoring algorithms aim to decompose a composite number into its prime factors; so, mathematically, it is a search problem. This process is computationally intensive for large integers and forms the basis of the security of some cryptographic systems. In contrast, primality-testing algorithms determine whether a given number is prime without necessarily identifying its factors; thus, mathematically, it is a decision problem. Classical algorithms like the Miller-Rabin test (probabilistic) and the AKS algorithm (deterministic and polynomial-time) are highly efficient and widely used in cryptography to verify large primes during key generation.

From a number-theoretic perspective, numbers used for primality testing can be classified into two groups: generic numbers and numbers with special properties. Following, we give definitions for these special numbers and symbols.

Mersenne number [32] is a number in the form $2^n - 1$, where n is a prime number. If such a number is prime, it is called a Mersenne prime. Examples of Mersenne primes are M=1, 3, 5, and 4423. Not all such numbers are primes, e.g., M=11 gives $2^{11} - 1 = 2047 = 23 \times 89$. The largest Mersenne prime number known to date is $2^{136,279,841} - 1$ (discovered by GIMPS).

Fermat number [32] is a number in the form $2^m + 1$. If this number is prime, then m must be of the form $m = 2^n$. Fermat initially conjectured that all numbers of this form would be prime; Euler disproved this by showing that $2^{2^5} + 1$ is composite. There are three important facts about Fermat numbers:

1. Only the first four Fermat numbers are known to be prime: 5, 17, 257, and 65537
2. The next 243 Fermat numbers have been proven composite
3. Current mathematical conjecture suggests no other Fermat primes exist beyond these first four

Proth numbers [33] are integers of the form $t \times 2^n + 1$, where t is an odd positive integer and n is a positive integer satisfying $2^n > t$. Numbers of this form play an important role in computational number theory because their particular structure allows efficient primality testing. In particular, Proth numbers can be tested for primality using Proth's theorem, which provides a deterministic condition based on modular exponentiation. This makes them especially useful for searching large prime numbers and for benchmarking primality testing algorithms in experimental studies.

Next, we introduce the Legendre and Jacobi symbols, two fundamental tools in number theory that play a critical role in modern primality testing algorithms.

The *Legendre symbol* [34], introduced by *Adrien-Marie Legendre*, is a classical function in number theory used to characterize quadratic residues modulo a prime. A quadratic residue modulo a prime p is an integer a for which there exists an integer x such that

$$x^2 \equiv a \pmod{p}.$$

In other words, a is a quadratic residue modulo p if it is congruent to a perfect square modulo p. If no such integer x exists, then a is called a quadratic nonresidue modulo p.

For an integer a and an odd prime p, the Legendre symbol is denoted by $\left(\frac{a}{p}\right)$ and defined as

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p \mid a, \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ -1 & \text{if } a \text{ is a quadratic nonresidue modulo } p. \end{cases}$$

Rather than representing a particular class of prime numbers, the Legendre symbol serves as a fundamental algebraic tool in modular arithmetic. It plays a key role in several primality-testing techniques and theoretical results concerning quadratic residues. In particular, it is frequently employed in proofs and algorithms related to special prime forms, including those appearing in Proth-type primality tests, where quadratic residue properties modulo a candidate prime are exploited to establish primality conditions.

Jacobi symbol [34], On the other hand, is defined as in (1)

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i} \tag{1}$$

where n is an odd positive integer and $\left(\frac{a}{p_i}\right)$ denotes the previously defined Legendre symbol.

The Legendre symbol can be computed efficiently when the modulus is a prime number, and its properties are readily applicable. However, when dealing with large composite integers for which factorization is difficult, the Jacobi symbol [34] becomes particularly useful. Unlike the Legendre symbol, the Jacobi symbol can be evaluated efficiently without requiring the factorization of either a or n, making it practical for computations involving large integers.

3 Taxonomy of Classical Algorithms

3.1 Primality Testing Algorithms

Primality testing algorithms can be generally classified into two categories: deterministic and randomized. However, after extensive research, it turns out that these algorithms should be classified into five categories: 1) Randomized Monte-Carlo primality tests, 2) Randomized Las-Vegas primality tests, 3) Deterministic primality tests, and 4) Heuristic primality tests. In the following subsections, firstly, we define the notations and symbols being used throughout this subsection. Then, we delve into the details of these primality testing algorithms.

- n – the integer being tested for primality
- p – a prime number
- $\log n$ – the number of bits in the binary representation of n
- k – the number of iterations/repetitions of a randomized test
- a – a randomly chosen base integer, typically with $1 < a < n$
- $\gcd(a, n)$ – the greatest common divisor of a and n
- $a \equiv b \pmod{n}$ – a is congruent to b modulo n (i.e., n divides $a - b$)
- (a/p) – the Legendre symbol.
- (a/n) – the Jacobi symbol (generalization of Legendre symbol for composite n)
- q, q_i – prime factors
- f – large factor (not necessarily a prime).
- $M_p = 2^p - 1$ – Mersenne number with prime exponent p
- $F_n = 2^{2^n} + 1$ – the nth Fermat number
- U_k, V_k – Lucas sequences defined by parameters P and Q

3.1.1 Monte-Carlo Randomized Primality Tests

In this category, an algorithm tests a large integer for primality. If the number is identified as prime, then an error that depends on the chosen certainty value should be considered, and the number is called a probable prime. In contrast, if the number is found to be a composite, then the finding is said to be 100% accurate. Therefore, Monte-Carlo randomized primality tests are more accurately called compositeness tests. In this section, three such primality tests are discussed.

Fermat Primality Test Equation (2) expresses Fermat's Little Theorem, which states that for any prime number p and any integer a such that $\gcd(a, p) = 1$, the following congruence holds:

$$a^{p-1} \equiv 1 \pmod{p}. \tag{2}$$

This theorem forms the theoretical foundation of the Fermat primality test, which uses this property to probabilistically assess whether a given integer is prime [35].

Algorithm: To test the primality of a number n, a random number a is chosen between $(1, n)$ and then a check is made to see if (2) holds. If the chosen a does not satisfy the equality, then a is called a *Fermat witness* and n is composite. If many distinguished values of a satisfy the equality, then n is probably prime. However, if n is an odd composite integer and a satisfies the equality, then a is called a *Fermat liar* (to primality for n), and n is called a *pseudoprime* [36].

Algorithm 1 Fermat Primality Test

```
Input: Odd integer n, security parameter k
Output: "composite" or "probably prime"
for i = 1 to k do
    Pick random a ∈ [2, n − 2]
    if a^{n−1} ≢ 1 (mod n) then return "composite"
    end if
end for
return "probably prime"
```

Complexity and Limitations: The time complexity of the test is $O(k \log^3 n)$. *Carmichael numbers* [37] are composite integers that are always declared by the Fermat test as primes. Namely, a Carmichael number always passes the condition of the Fermat test for any base integer a. W. Alford et al. [38] prove that there is an infinite quantity of Carmichael numbers. The Fermat test is considered a fast primality test, especially if the input number is composite. The main limitation of this algorithm is that the probability of failure for Carmichael numbers is 1.

Solovay-Strassen Primality Test The Solovay–Strassen primality test was introduced by Robert Solovay and Volker Strassen [39] as a probabilistic algorithm for determining whether a given integer n is composite or probably prime. The test was widely used in the early development of public-key cryptography, particularly in implementations related to the RSA cryptosystem.

The method relies on Euler's criterion [35], which states that if p is an odd prime and a is an integer such that $\gcd(a, p) = 1$, then

$$a^{(p-1)/2} \equiv (a/p) \pmod{p}, \tag{3}$$

where (a/p) denotes the Legendre symbol.

Algorithm: The Solovay–Strassen test applies this property by selecting random integers a with $1 < a < n$. The Jacobi symbol (a/n) is computed and compared with $a^{(n-1)/2} \bmod n$. If the congruence does not hold, n is declared composite; otherwise, n is considered probably prime. Repeating the test with different values of a reduces the probability of error.

Complexity and Properties: The time complexity of the Solovay-Strassen test is $O(k \log^3 n)$, where k denotes the number of iterations. The probability that a composite number passes a single iteration of the Euler test is at most $1/2$. After k independent iterations, the probability of incorrectly declaring a composite number as a prime is bounded by $(1/2)^k$. Composite numbers that satisfy the Euler criterion for a given base are called *Euler pseudoprimes*.

The main feature of this test is that ordinary pseudoprime numbers cannot consistently pass it. However, there exist Euler pseudoprimes, which are odd composite numbers that satisfy the test congruence for a specific base a. Such numbers may therefore pass the Solovay-Strassen test for that particular base. Nevertheless, this property does not hold for all bases, and choosing different random values of a quickly reveals the compositeness of n. This distinguishes

Algorithm 2 Solovay–Strassen Primality Test

Input: Odd integer $n > 2$, number of rounds k
Output: "composite" or "probably prime"
for $i = 1$ to k **do**
    Choose random a with $2 \leq a \leq n-1$
    Compute $x \leftarrow a^{(n-1)/2} \bmod n$
    Compute Jacobi symbol $j \leftarrow (a/n)$
    if $x = 0$ or $x \neq j \bmod n$ **then return** "composite" ▷ a is an Euler witness
    end if
end for
return "probably prime"

Euler pseudoprimes from Carmichael numbers, which satisfy Fermat's congruence for all bases relatively prime to n. Another drawback of the test is the additional computational cost associated with evaluating the Jacobi symbol during each iteration.

Miller-Rabin Primality Test The Miller-Rabin, also known as the strong pseudoprime primality test [40], was devised by Gary Miller. The first version was deterministic, assuming the correctness of the extended Riemann hypothesis (ERH) [40]. Then Michael Rabin revised it to be a probabilistic algorithm [41]. The test detects the primality of a number by testing two criteria: First, recall the Fermat test, and if the input number n is declared as a composite, the test exits early. Second, the test detects a non-trivial square root of $1 \pmod{n}$. If there is one, then n is a composite. Otherwise, n is a probable prime.

Miller's Test Procedure:

1. Suppose $a^{n-1} \equiv 1 \pmod{n}$ for all $\gcd(a, n) = 1$.
2. Write $n - 1$ as $2^s \times t$ where t is an odd integer and $s \geq 1$.
3. We say that n passes Miller's test for the base a iff:
    - $a^t \equiv 1 \pmod{n}$, or
    - $a^{2^j \times t} \equiv -1 \pmod{n}$ for some $j \in \{0, 1, \ldots, s-1\}$.

Algorithm 3 Miller-Rabin Primality Test

Input: Odd integer $n > 2$, number of rounds k
Output: "composite" or "probably prime"
Write $n - 1 = 2^s \times t$ with t odd
for $i = 1$ to k **do**
    Choose random a with $2 \leq a \leq n-2$
    Compute $x \leftarrow a^t \bmod n$
    if $x = 1$ or $x = n - 1$ **then**
        continue ▷ probably prime for this round
    end if
    for $j = 1$ to $s - 1$ **do**
        $x \leftarrow x^2 \bmod n$
        if $x = n - 1$ **then**
            continue outer loop
        end if
    end for
    return "composite" ▷ a is a strong witness
end for
return "probably prime"

Complexity and Properties: This test is used in many practical cryptographic systems because of its efficiency. The output of this algorithm is a proof that the number is a composite or the number is a probable prime. Miller-Rabin is a polynomial-time algorithm with a time complexity of $O(k \log^3 n)$. The probability of declaring a composite number as a prime is less than $(1/4)^k$. Composite numbers that pass the Miller-Rabin test are called *strong pseudoprimes*, and they are much fewer than the pseudoprimes. Furthermore, all Carmichael numbers are classified as composite.

Several notions of pseudoprimes arise from different probabilistic primality tests. These classes are related hierarchically. Let n be an odd composite integer and a a base such that $\gcd(a, n) = 1$. Then the following inclusion holds:

$$\text{Strong pseudoprimes (base } a) \subset \text{Euler pseudoprimes (base } a) \subset \text{Fermat pseudoprimes (base } a).$$

In other words, every strong pseudoprime to base a is also an Euler pseudoprime to base a, and every Euler pseudoprime is a Fermat pseudoprime to the same base. However, the converses do not generally hold. This hierarchy explains why the Miller-Rabin test (which detects strong pseudoprimes) is more reliable than the Euler or Fermat primality tests.

3.1.2 Las-Vegas Randomized Primality Tests

In contrast to the Monte-Carlo primality tests, which may return a false positive, this type of test always declares an input number n as a prime with 100% certainty but with a running time that varies randomly. Therefore, these tests are often called *primality proving tests*.

Proth Test The Proth primality test [42] was proposed by François Proth to determine whether a given Proth number n is prime. Proth Theorem states: if p is a Proth number in the form $t \times 2^n + 1$, where t is an odd integer such that $t < 2^n$, then p is a prime if Eq. (4) holds. In this case, p is called a *Proth prime*. An integer a that satisfies the condition of the theorem can be found by iterating k times and calculating Jacobi symbol (a/n), until $(a/n) = -1$. So, there exists an a such that Eq. (4) holds.

$$a^{(p-1)/2} \equiv -1 \pmod{p}. \tag{4}$$

Algorithm 4 Proth Primality Test

Input: Proth number $p = t \times 2^n + 1$ with t odd and $t < 2^n$
Output: "prime" or "composite"
for k **times do**
    Choose $a \in_R \{2, 3, \ldots, n-1\}$ ▷ Uniform random selection
    Compute Jacobi symbol $j \leftarrow (a/p)$
    if $j = -1$ **then**
        if $a^{(p-1)/2} \equiv -1 \pmod{p}$ **then return** "prime"
        else
            return "composite"
        end if
    end if
end for
return "composite" ▷ no suitable a found

Complexity and Properties: If p is prime, half of the possible values of a satisfy $(a/n) = -1$ [43]. Once such a is found, the test declares n as definitely prime and exits. If the loop is executed for k iterations and no such a is found, then n is declared as composite. The largest ever Proth prime number is $10223 \times 2^{31172165} + 1$, which was discovered in 2016 [44]. The average time complexity of the Proth test is $O\big(k((t \log t + \log n)\ \log n)\big)$.

Lucas Primality Test Lucas primality test [45] is one of four tests proposed by Édouard Lucas to check whether a given number n is prime. Lucas test is considered a very efficient method for proving primality for moderately sized primes.

Lucas theorem: let n be an odd positive integer; if there exists an integer a such that $1 < a < n$ and Eq. (5) holds, and if all prime factors $q_1, q_2, \ldots, q_s$ of $n-1$ yield Eq. (6), then n is a prime. If no such number a exists, then n is probably composite.

$$a^{n-1} \equiv 1 \pmod{n}. \tag{5}$$

$$a^{(n-1)/q_i} \not\equiv 1 \pmod{n}. \tag{6}$$

Algorithm 5 Lucas Primality Test

Input: Integer $n > 1$, prime factors $q_1, \ldots, q_s$ of $n-1$
Output: "prime" or "composite"
for k **times do**
    Choose $a \in_R \{2, 3, \ldots, n-1\}$ ▷ Uniform random selection
    if $a^{n-1} \equiv 1 \pmod{n}$ **then**
        valid $\leftarrow$ true
        for each prime factor q_i **do**
            if $a^{(n-1)/q_i} \equiv 1 \pmod{n}$ **then**
                valid $\leftarrow$ false
                break
            end if
        end for
        if *valid* **then return** "prime"
        end if
    end if
end for
return "composite"

Complexity and Properties: In this test, the witnesses of primality are the prime factors of $n-1$. Lucas' test can work efficiently with any generic number, including Mersenne and Fermat. One apparent drawback of this test is that it requires the prime factors of $n-1$ to be already known. The time complexity of the test is $O(k.w(n-1).log^3 n)$ where $w(n-1)$ refers to the distinct prime factors of $n-1$.

Pocklington Test In contrast to Lucas primality test which utilizes full factorization of $n-1$, the Pocklington primality test [46] is a Las-Vegas primality algorithm that proves an integer n is prime by utilizing a partial factorization of $(n-1)$.

Pocklington's theorem: states that given an odd integer $n \geq 3$, a witness a $(1 < a < n)$, $n-1 = f.r$ such that f and r are relatively primes and $f > \sqrt{n} - 1$, and prime factors $q_1, q_2, \ldots, q_s$ of f. Then n is prime if the following two conditions are fulfilled:

1. $a^{n-1} \equiv 1 \pmod{n}$.
2. $\gcd(a^{(n-1)/q} - 1, n) = 1$.

Algorithm 6 Pocklington Primality Test

Input: Odd integer $n \geq 3$, factor f with $f|(n-1)$ and $f > \sqrt{n} - 1$, prime factors $q_1, \ldots, q_s$ of f
Output: "prime", "composite", or "inconclusive"
for k **times do**
    Choose $a \in_R \{2, 3, \ldots, n-1\}$ ▷ Uniform random selection
    if $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
        return "composite" ▷ condition (1) fails
    end if
    for each prime factor q_i of f **do**
        if $\gcd(a^{(n-1)/q} - 1, n) \neq 1$ **then**
            valid $\leftarrow$ false
            break
        end if
    end for
    if *valid* **then return** "prime"
    end if
end for
return "inconclusive"

Properties: If a fails to satisfy condition (1), then n is composite, and the test terminates. If condition (1) holds but a does not satisfy condition (2), the test is inconclusive, and another random base a is selected for a new iteration. If a satisfies both conditions, then n is guaranteed to be prime. The time complexity of the test is $O(k.w(f).log^3 n)$ where $w(f)$ refers to the distinct prime factors of f.

3.1.3 Deterministic Primality Tests

These algorithms provide an accurate decision about the input number, whether it is a prime or composite. However, they are general considered impractical due to the immense computational resources required for extremely large numbers.

Trial Division (Naïve) Test This is the simplest deterministic primality test [1]. For an input number n, this test investigates if any integer m, from 2 to $\sqrt{n}$, can divide n evenly. If n is divided evenly by any m, the number is a composite; otherwise, it is prime.

Algorithm 7 Trial Division Primality Test

1: **Input:** Odd Integer $n > 1$
2: **Output:** "prime" or "composite"
3: **for** $m = 3$ to $\lfloor \sqrt{n} \rfloor$ step 2 **do**
4: **if** $n \bmod m = 0$ **then return** "composite"
5: **end if**
6: **end for**
7: **return** "prime"

Complexity and properties: This test always returns a certain answer. However, for large numbers, the algorithm is too slow. Trial division takes $O(2^{\log n/2})$ operations to check if an $\log n$-bit number is prime, which is exponential in the size of the input.

Pepin Test Pepin primality test [47] is based on the Fermat test. In 1877, Pepin proved a method for testing the primality of Fermat numbers. The output of this algorithm is a proof that the number is prime or its primality could not be proved.

Pepin's theorem states: that if F_n is a Fermat number where n is greater than zero, such that $F_n = 2^{2^n} + 1$ is the *nth* Fermat number, then F_n is a prime, iff Eq. (7) holds.

$$3^{(F_n-1)/2} \equiv -1 \pmod{F_n} \tag{7}$$

Algorithm 8 Pepin Primality Test

1: **Input:** Fermat number $F_n = 2^{2^n} + 1$ with $n > 0$
2: **Output:** "prime" or "composite"
3: Compute $r \leftarrow 3^{(F_n-1)/2} \bmod F_n$
4: **if** $r \equiv -1 \pmod{F_n}$ **then return** "prime"
5: **else return** "composite"
6: **end if**

Complexity and Properties: Other bases may be used in place of 3, for example, 5 or 7 [48]. The time complexity of this algorithm is $O(\log^2 n)$.

Lucas-Lehmer Test The Lucas–Lehmer test was originally introduced by Édouard Lucas and later refined by Derrick H. Lehmer. It provides an efficient deterministic primality test for Mersenne numbers.
Lucas–Lehmer theorem: Let $M_p = 2^p - 1$ be a Mersenne number where p is an odd prime. Since p is exponentially smaller than M_p, the primality of p can first be verified using a simple method such as trial division. Then M_p can be tested for primality using the following procedure.

Algorithm 9 Lucas-Lehmer Primality Test

1: **Input:** Mersenne number $M_p = 2^p - 1$ with p prime
2: **Output:** "prime" or "composite"
3: $s \leftarrow 4$
4: **for** $i = 1$ to $p - 2$ **do**
5: $s \leftarrow (s^2 - 2) \bmod M_p$
6: **end for**
7: **if** $s \equiv 0 \pmod{M_p}$ **then return** "prime"
8: **else return** "composite"
9: **end if**

Properties: After completing the $(p - 2)$ iterations, if the final value of s is congruent to $0 \bmod M_p$, then M_p is prime; otherwise, M_p is composite.

AKS Test M. Agrawal and his two students proposed a deterministic test [49] that runs in polynomial time, for determining if an integer number is prime. The test is known as the Agrawal-Kayal-Saxena (AKS) primality test. The test is considered distinguished and exceptional. Its exceptionalism stems from its ability to prove the primality of any given integer in deterministic and polynomial time. The idea of AKS is based on a generalization of Fermat's little theorem to polynomials with coefficients in $\mathbb{Z}/\mathbb{Z}_n$ where n is any prime number.

Algorithm 10 AKS Primality Test

Input: Integer $n > 1$
Output: "prime" or "composite"
if $n = a^b$ for some $a \in \mathbb{N}$ and $b > 1$ **then return** "composite"
end if
Find smallest r such that $\text{ord}_r(n) > \log^2 n$
if $1 < \gcd(a, n) < n$ for some $a \leq r$ **then return** "composite"
end if
if $n \leq r$ **then return** "prime"
end if
for $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ **do**
    if $(X + a)^n \not\equiv X^n + a \pmod{X^r - 1, n}$ **then return** "composite"
    end if
end for
return "prime"

Complexity: Initially, the complexity of this test was $O(\log^{11.5} n)$ [50], but a lot of improvements have been made from 2002 to this time, and as a result, the complexity has become $O(\log^6 n)$ [50].

3.1.4 Heuristic Primality Tests

In this type of primality test, no mathematical proof or correctness is provided. The idea behind these tests is proposed heuristically, e.g., by mixing two tests.

Baillie-PSW Test *Theoretical Foundation:* The Baillie-PSW (BPSW) test is a heuristic primality test that combines the Miller-Rabin test with a Lucas-sequence test [51, 52]. For an input integer n, the test proceeds as follows: Initially, the test verifies if n has small prime factors (smaller than 1000). Secondly, the test applies the Miller-Rabin test using base $a = 2$. Finally, the input n should be tested using the Lucas-sequence test.

The Lucas-sequence test is based on sequences U_k and V_k defined recursively by integer parameters P and Q:

$$U_0 = 0, \quad U_1 = 1, \quad U_k = PU_{k-1} - QU_{k-2} \text{ for } k \geq 2,$$
$$V_0 = 2, \quad V_1 = P, \quad V_k = PV_{k-1} - QV_{k-2} \text{ for } k \geq 2.$$

The test checks specific congruence conditions on these sequences modulo n; failure of any condition implies that n is composite, while passing suggests n is a probable prime. *Properties:* No composite number is known to pass the BPSW test, making it highly

Algorithm 11 Baillie-PSW Primality Test

Input: Odd integer $n > 2$
Output: "composite" or "probably prime"
Step 1: Trial division by small primes up to 1000
if n has a small prime divisor **then return** "composite"
end if
Step 2: Miller-Rabin test with base $a = 2$
if Miller-Rabin$(n, a = 2)$ returns "composite" **then return** "composite"
end if
Step 3: Lucas-sequence test
Select parameters P, Q using Selfridge's method
Compute Lucas sequences U_k and V_k modulo n
if Lucas test conditions fail **then return** "composite"
end if
return "probably prime"

reliable in practice. However, it remains a heuristic test because no proof exists that such pseudoprimes cannot exist. If the input n

passes the aforementioned tests, then n is a probable prime or a BPSW pseudoprime. On the other hand, if n fails to pass any of the tests, then n is a composite. This test is considered to be very efficient, since no composite number less than 2^{64} has passed it (no BPSW pseudoprimes) [53].

Table 1 shows the taxonomy of the classical primality tests and compares their characteristics, pros, cons, time complexity, and probability of error.

3.2 Classical Integer Factorization Algorithms

While primality testing determines whether an integer n is prime, integer factorization seeks to decompose a composite number into its constituent prime factors. The security of the RSA cryptosystem relies on the mathematical asymmetry between the ease of multiplying two large primes and the extreme difficulty of reversing the process. Classical factoring algorithms are broadly categorized into two groups: special-purpose algorithms, which exploit specific properties of the factors, and general-purpose algorithms, which depend solely on the magnitude of n.

These methods are most effective when the target number n has factors with specific characteristics, such as being small or having a specific distance from other numbers.

Trial division [56]: The most fundamental method involves dividing n by every prime $p \leq \sqrt{n}$. While it is the only deterministic method that guarantees finding the smallest factor, its exponential time complexity $O(\sqrt{n})$ makes it computationally expensive for integers exceeding 100 bits. In modern hybrid systems, it is often used as a "pre-filtering" step to remove small factors before applying more complex algorithms.

Pollard's rho algorithm [57]: Introduced in 1975, this Monte Carlo method relies on the Birthday Paradox and Floyd's cycle-finding algorithm. It explores a sequence of integers generated by a polynomial function (typically $f(x) = x^2 + 1 \mod n$). A collision in this sequence $\mod p$ (where p is a factor of n) allows the factor to be extracted via a Greatest Common Divisor (GCD) calculation. Its expected running time is $O(\sqrt{p})$, making it highly effective for finding factors up to 20 digits.

Lenstra's Elliptic Curve Method (ECM) [58]: A sophisticated generalization of Pollard's $p-1$ algorithm, ECM utilizes the group law on elliptic curves over finite fields. By choosing random elliptic curves, the algorithm attempts to find a curve whose order is smooth (composed of small primes). It is currently the best algorithm for finding factors of intermediate size (40-60 digits) and is a critical component in distributed factoring projects.

General-purpose algorithms are the primary tools for attacking RSA-level integers, as they do not rely on the factors being small. Most modern methods are based on the "Congruence of Squares" principle: finding x, y such that $x^2 \equiv y^2 \pmod{n}$, which implies n divides $(x-y)(x+y)$.

Quadratic Sieve (QS) [59]: The Quadratic Sieve was the fastest algorithm for general integers until the early 1990s. It functions by collecting "smooth" values of a polynomial and using linear algebra (Gaussian elimination over $GF(2)$) to find a combination of these values that forms a perfect square. It remains the method of choice for numbers between 50 and 100 decimal digits.

General Number Field Sieve (GNFS) [60]: The GNFS is the current state-of-the-art for factoring large, general integers. Unlike the QS, which works over the integers, the GNFS performs sieving over algebraic number fields. This allows for a much more efficient search for smooth elements. Its complexity is sub-exponential. This algorithm is responsible for all recent records in factoring RSA challenge numbers [6]. Its implementation is highly parallelizable, requiring massive computational clusters and significant memory resources for the matrix step.

A common notation for expressing subexponential complexity is the L-notation:

$$L_n[\alpha, c] = \exp\left((c + o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}\right) \tag{8}$$

where $0 \leq \alpha \leq 1$. Polynomial-time algorithms correspond to $\alpha = 0$, while exponential algorithms have $\alpha = 1$. The most efficient modern factorization algorithms achieve $\alpha = 1/3$ (GNFS) or $\alpha = 1/2$ (QS, ECM). Table 2 provides a comprehensive comparison of the five major algorithmic approaches.

3.3 Implementation and experimental Evaluation

To evaluate the performance of the considered primality testing algorithms, we implemented eleven representative algorithms covering deterministic and probabilistic approaches. The experiments were conducted on a personal computer equipped with an Intel Core i7-2430M processor running at 2.40 GHz, with 16 GB of RAM, operating under a 64-bit version of Windows 11.

All algorithms were implemented using the same programming environment to ensure fair comparison. The implementations rely on standard arbitrary-precision arithmetic libraries to handle large integer operations required by primality testing procedures. Execution times were measured using system timing functions with millisecond precision.

To evaluate algorithm performance under comparable conditions, several experimental scenarios were designed using integers of different structural forms, including general integers, Fermat numbers, Mersenne numbers, and Proth numbers. The input size range was limited to integers of up to 20 decimal digits in order to maintain computational feasibility on classical hardware while still allowing meaningful performance comparisons among the algorithms, i.e., this range was strategically selected to capture the divergence in asymptotic scaling between different algorithmic classes while ensuring high-resolution timing data on modern

Reference	Characteristics	Pros	Cons	Time Complexity	Probability of Error
Fermat [35]	Randomized (Monte-Carlo); Compositeness test.	Very simple to implement; Basis for many tests.	Failure rate may reach 1; Pseudoprimes can pass.	$O(k \log^3 n)$	Carmichael = 1; Pseudoprimes $\approx 2.45 \times 10^{-4}$.
Solovay-Strassen [39]	Randomized (Monte-Carlo); Compositeness test.	Flags pseudoprimes as composites.	Euler pseudoprimes may pass; Jacobi-symbol overhead.	$O(k \log^3 n)^3$	$\leq (1/2)^k$.
Miller-Rabin [40]	Randomized (Monte-Carlo); compositeness test.	Fast, efficient; Euler pseudoprimes detected.	Strong pseudoprimes may pass.	$O(k \log^3 n)$	$\leq (1/2)^k$.
Lucas [45]	Randomized (Las-Vegas); Primality test.	Valid for general or special forms.	Requires prime factors of $n-1$; May not terminate on composites.	$O(k.w(n-1).log^3 n)$	0 if prime.
Proth [54]	Randomized (Las-Vegas); Primality test.	Fast and reliable for Proth numbers.	Limited to Proth numbers.	$O\big(k((t \log t + \log n)\ \log n)\big)$	0 if prime; ≈ 0 if base covers 50
Pocklington [46]	Randomized (Las-Vegas); Primality test.	More efficient than Lucas test if a factor $f > \sqrt{n} - 1$ is known.	requires the factor f to be known.	$O(k.w(f).log^3 n)$	0 if prime.
Lucas-Lehmer [17]	Deterministic; Compositeness & Primality test.	Accurate for Mersenne numbers; Practical for large primes.	Slow for large n; needs factors of $n-1$.	$O(n^2 \log n \log \log n)$	0.
Trivial Division [1]	Deterministic; Compositeness & Primality test.	Reliable for $n \leq 10$ digits.	Infeasible for large n.	$O(\sqrt{n})$	0.
Pepin [55]	Deterministic; Compositeness & Primality test.	Efficient for Fermat-form numbers.	Infeasible for large Fermat numbers; Poor on other forms.	$O((\log^2 n))$	0.
AKS [49]	Deterministic; Compositeness & Primality test.	Polynomial-time deterministic test.	Very slow for small/moderate n.	$O((\log^6 n))$	0.
Baillie-PSW [51, 52]	Heuristic compositeness test.	No known pseudoprime passes.	Lacks formal proof.	$O(k \log^3 n)$	0 for $n < 2^{64}$.

Table 1: Comprehensive Comparison Between Different Classical Primality Tests

Table 2: Comprehensive Comparison of Classical Integer Factorization Algorithms

Algorithm	Complexity	Characteristics	Best Use Case
Trial Division	$O(\sqrt{n})$ or $O(\pi(\sqrt{n}))$; exponential in input length	Simple deterministic method; low memory requirements; straightforward implementation	Very small integers
Pollard Rho	$O(n^{1/4})$ for balanced semiprimes; $O(p^{1/2})$ to find factor p; subexponential	Randomized and memory-light; simple implementation using polynomial iteration and gcd checks	Finding small to medium-sized prime factors quickly (when factor $p \ll \sqrt{n}$); fast fallback before heavier sieving methods
Elliptic Curve Method (ECM)	$L_p[1/2, c] \approx \exp(c\sqrt{\log p \log\log p})$ for factor p; subexponential in factor size	Performance depends on target factor p rather than n; excellent for isolated medium factors	Locating isolated medium-size prime factors (up to dozens of digits) in large composites; primary tool in "factor hunting" stage of large factorizations
Quadratic Sieve (QS)	$L_n[1/2, 1] = \exp((1 + o(1))\sqrt{\log n \cdot \log\log n})$; subexponential	General-purpose deterministic algorithm; uses factor base and sieving to collect relations	Medium-range integers (practically competitive up to $\sim$100 decimal digits)
General Number Field Sieve (GNFS)	$L_n[1/3, (64/9)^{1/3}] \approx \exp(c(\log n)^{1/3}(\log\log n)^{2/3})$; subexponential	Asymptotically fastest known classical algorithm	Very large integers (large RSA moduli, numbers substantially above QS practical range)

classical architectures. Yet, in Appendix A, we show the results of testing primality for large numbers, specifically, Proth and Mersenne numbers.

The experimental evaluation focuses on comparing the relative efficiency of the algorithms in terms of execution time and scalability with respect to input size. The results are presented in the following figures, which illustrate the performance differences between deterministic, probabilistic, and specialized primality testing methods.

The overall results in terms of execution time, speed, and accuracy are summarized and listed in Table 3. The more stars in the speed and accuracy columns, the more speed and accuracy. For detailed experimental configuration and results, we refer the reader to Appendix.A.

Test Name	Tested Numbers (%)	Execution Time (ms)	Speed	Accuracy	Compared with
Fermat	11621P	0.508956	⋆ ⋆ ⋆	⋆⋆	- Miller-Rabin - Solovay-Strassen
	11611C	2.824186			
	2860486327P	0.483302			
	127647878463584414 71P	8.185518			
	127647878463584414 81C	0.727727			
Miller-Rabin	11621P	1.747619	⋆ ⋆ ⋆	⋆ ⋆ ⋆	- Fermat - Solovay-Strassen - Proth - Baillie-PSW
	11611C	0.474660			
	2860486327P	2.987280			
	2860486317C	0.659142			
	127647878463584414 71P	7.3708062			
	127647878463584414 81C	0.929735			
	45057C	0.671879			
	6710886401C	1.315023			
Proth	18433P	0.567288	⋆ ⋆ ⋆	⋆ ⋆ ⋆	- Lucas - Pocklington - Miller-Rabin - Pepin - Baillie-PSW
	45057C	1.707508			
	2281701377P	0.784949			
	6710886401C	4.1291498			
	65537P	0.358261			
	32769C	1.522590			
	4294967297C	11.50602			
Lucas	18433P	84.29452	⋆⋆	⋆ ⋆ ⋆	- Pocklington - Proth
	45057C	103.8156			
	2281701377P	46663.94			
	6710886401C	183237.6			

Continue on the next page

Table 3 – *Continued from previous page*

Test Name	Tested Numbers (%)	Execution Time (ms)	Speed	Accuracy	Compared with
Pocklington	18433P 45057C 2281701377P 6710886401C	108.233 145.337 88651.8 416631.6	⋆⋆	⋆ ⋆ ⋆	- Lucas - Proth
Solovay-Strassen	11621P 11611C 2860486327P 127647878463584414 71P 127647878463584414 81C	13.41032 1.070337 25.96425 62.23205 3.21238	⋆⋆	⋆ ⋆ ⋆	- Fermat - Miller-Rabin
Naïve	8191P 2047C 2147483647P 137438953471C 65537P 32769C 4294967297C	0.90630 0.63920 114.252 2.36577 1.91386 0.61518 5.09161	⋆	⋆ ⋆ ⋆ ⋆ ⋆	- Pepin - AKS - Lucas-Lehmer
Pepin	65537P 32769C 4294967297C	0.76835 0.60004 0.75261	⋆⋆	⋆ ⋆ ⋆ ⋆ ⋆	- Naïve - AKS - Proth - Baillie-PSW
AKS	65537P 32769C 4294967297C 8191P 2047C 2147483647P 137438953471C	1088.48 49.5588 2717.46 399.29 13.0188 184922.93 874.78	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆ ⋆ ⋆	- Naïve - Pepin - Lucas-Lehmer
Lucas-Lehmer	8191P 2047C 2147483647P 137438953471C	0.59918 0.60184 0.84789 1.27422	⋆ ⋆ ⋆	⋆ ⋆ ⋆ ⋆ ⋆	- Naïve - AKS - Baillie-PSW
Baillie-PSW	65537P 32769C 4294967297C 8191P 2047C 2147483647P 137438953471C 18433P 45057C 2281701377P 6710886401C 11621P 11611C 2860486327P 2860486317C 127647878463584414 71P 127647878463584414 81C	1.08419 0.46696 2.06584 1.33820 0.76450 1.50403 3.76512 1.01218 0.47971 2.12836 1.12986 1.66610 0.22784 1.89539 0.28240 7.22091 0.49775	⋆ ⋆ ⋆⋆	⋆ ⋆ ⋆⋆	- Miller-Rabin - Proth - Pepin - Lucas-Lehmer

Table 3: Summarizing the results (execution time, speed, and accuracy).

4 Taxonomy of Quantum Algorithms

4.1 Quantum Primality Tests

Throughout this section on quantum primality testing algorithms, we use the following notation and symbols:

- n – the number being tested for primality
- k – threshold parameter.
- p, q – prime numbers.
- p_j, q_i – prime factors.
- $\mathbb{Z}_n^*$ – is the set of all integers relatively prime to n under multiplication modulo n.
- a_j – integers satisfying specific congruence conditions in Brillhart's equation
- m – a randomly selected integer for testing primality conditions

- r – the order of a modulo n, i.e., the smallest positive integer such that $a^r \equiv 1 \pmod{n}$
- $\text{ord}(a)$ or $\text{ord}_n(a)$ – the multiplicative order of a modulo n
- $\varphi(n)$ – Euler's totient function.
- P – precision parameter in quantum algorithms (typically a power of 2)
- R – number of repetitions or ancilla register size in quantum algorithms
- Q – number of distinct quantum states in subroutine loops
- $|u_m\rangle$ – quantum state of ancilla qubits
- F – quantum Fourier transform operator
- G_m – Grover unitary operator
- $W_k(a)$ – witness function mapping quantum states to $\{0, 1\}$
- d – dimension parameter for qudit systems

4.1.1 Chau and Lo Algorithm

Chau and Lo algorithm [61] is a probabilistic quantum algorithm that combines the classical Pocklington-Lehmer $n-1$ primality test with the quantum integer factoring algorithm (Shor's). To examine the primality of an odd number n, firstly, the complete factoring of $n-1$ number is calculated by utilizing Shor's algorithm. Secondly, the conditions asserted by the Pocklington-Lehmer algorithm are verified classically, namely, verifying that the complete factoring of $n-1$ has been obtained. Initially, the authors showed that this method is upper bounded by $O((\log n)^3(\log\log n)^2 \log\log\log n)$ qubit operations. Further to enhance the performance, they implemented trial divisions to eliminate all prime factors of $n-1$ that are less than k. This process can be completed through k divisions, which takes $O(k \log n \log\log n \log\log\log n)$ time. Following the trial division, their focus shifted to the prime factors of $n-1$ that exceed k. It is evident that there are at most $O\left(\frac{\log n}{\log k}\right)$ distinct prime factors of $n-1$ greater than k. By integrating the trial division with Shor's algorithm, $n-1$ can be fully factored in

$$O\left(\log n \log\log n \log\log\log n \left(k + \frac{\log^2 n \log\log n}{\log k}\right)\right) \text{ time.}$$

The optimal solution was achieved when the number of trial divisions was chosen as $k \sim \frac{(\log^2 n)}{\log\log n}$. Consequently, the factorization of $n-1$ requires only

$$O\left(\log^3 n \log\log n \log\log\log n\right) \text{ qubit operations.}$$

If there exists a prime number table up to k, the prime number theorem indicates that only $O\left(\frac{k}{\log k}\right)$ trial divisions are needed. Following this reasoning, they determined that the optimal solution occurs when $k \sim \log^2 n$. Nevertheless, the optimal number of elementary qubit operations remains

$$O\left((\log^3 n) \log\log n \log\log\log n\right).$$

To expedite the verification of primality, they adopted a variation of the Pocklington-Lehmer test introduced by Brillhart et al. [62]. Let n be the number to test, and let $p_1, p_2, \ldots, p_k$ be the prime factors of $n-1$. Brillhart's equation requires finding integers a_j such that

$$a_j^{(n-1)/p_j} \not\equiv 1 \pmod{n} \quad \text{for each prime factor } p_j.$$

They randomly select an integer m and check if it satisfies this condition. For each prime factor p_j, the probability that a randomly chosen m fulfills the constraint is at least $\frac{1}{2}$. Thus, on average, half of the constraints are satisfied for each m. Consequently, by randomly selecting m and verifying Brillhart's equation several times, they can be reasonably certain of identifying all required a_j values.

By combining quantum factorization with trial division and Brillhart's equation, the authors achieved a quantum primality test with a runtime of $O\left((\log n)^3 \log\log n \log\log\log n\right)$

4.1.2 Donis-Vela and Garcia-Escartin Algorithm

In a quantum primality test with order finding [63], the algorithm seeks the order of an integer a modulo n, which is the smallest positive integer r such that

$$a^r \equiv 1 \pmod{n}.$$

This order-finding step is central to the algorithm: if the order satisfies certain properties, it can be used to efficiently determine whether n is prime.

Algorithm 12 Chau and Lo Quantum Primality Test

Input: An odd integer $n > 2$.
Output: "prime" or "composite".
Step 1: Factorization of $n-1$
Set threshold $k \sim \frac{(\log n)^2}{\log \log n}$.
Perform **Trial Division** on $n-1$ for all primes up to k.
For the remaining unfactored part of $n-1$, apply **Shor's Algorithm**.
Let the set of distinct prime factors of $n-1$ be $\{p_1, p_2, \ldots, p_m\}$.
Step 2: Brillhart's Verification (Classical)
Choose a random integer $m \in \{2, \ldots, n-1\}$.
Check Fermat's Little Theorem: $m^{n-1} \equiv 1 \pmod{n}$.
if $m^{n-1} \not\equiv 1 \pmod{n}$ **then**
    return n is **Composite**.
end if
for each prime factor $p_j \in \{p_1, \ldots, p_m\}$ **do**
    Find an integer a_j (by testing random m) such that:
    $a_j^{(n-1)/p_j} \not\equiv 1 \pmod{n}$.
    Note: On average, two random choices of m satisfy this for a given p_j.
end for
Step 3: Conclusion
if all p_j satisfy the Brillhart conditions **then**
    return n is **Prime** (with certainty).
else
    return n is **Composite**.
end if

The authors in [63] assert that they have minimized the number of steps by leveraging insights from the analysis of quantum order findings. The factor $\log\ n$ in the complexity of quantum order finding arises from the average number of measurements needed to identify two divisors of the order, which allows the exact value $ord(a)$ to be determined. They indicated that, through classical processing that tests small multiples of the extracted values from each measurement, the $\log n$ repetitions can be reduced to a constant number.

Algorithm 13 Donis-Vela and Garcia-Escartin [63]

Input: An integer n.
Output: "prime" or "composite".
Choose at random an integer $1 < a < n$.
Compute $\gcd(a, n)$.
if $\gcd(a, n) \neq 1$ **then**
    Declare n composite; return $\gcd(a, n)$ as a proof.
else
    Compute $a^{\frac{n-1}{2}} \mod n$.
    if $a^{\frac{n-1}{2}} \not\equiv \pm 1 \pmod{n}$ **then**
        Declare n composite; return a as a witness.
    else if $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ **then**
        Go back to Step 1.
    else if $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$ **then**
        Quantum Order Finding: Compute $\text{ord}(a)$.
        if $\text{ord}(a) = n-1$ **then**
            Declare n prime; return a as a quantum certificate of primality.
        else
            Go back to Step 1.
        end if
    end if
end if

Their algorithm (Algorithm 13) verified the order of random integers within the group until it could conclusively prove primality or compositeness. Initially, it selected a random integer from $\mathbb{Z}_n^*$, avoiding $a = 1$ due to its trivial order of 1. This is achieved by selecting an integer smaller than n and ensuring $\gcd(a, n) = 1$. If not, n is composite, and it can return the factor as proof. If $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$, the order of a is at most $\frac{n-1}{2}$, which provides no information on n's primality. In such cases, it restarted by choosing a new random integer. The process continues until $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, at which point it applies the quantum order-finding algorithm. If $\text{ord}(a) = n - 1$, then n is prime with certainty, and it can conclude the procedure by returning a as a quantum certificate of primality.

The classical pre-screening ensures that $\text{ord}(a) \mid (n - 1)$, stemming from the condition $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, which implies $a^{n-1} \equiv 1 \pmod{n}$. Before moving to the quantum stage, it performed basic checks to minimize the number of quantum operations. These operations are the most technologically demanding and are replaced with classical steps whenever possible. It verified $a^{n-1} \equiv 1 \pmod{n}$ indirectly by checking $a^{\frac{n-1}{2}}$. If this value equals ± 1 and passes Fermat's test, it proceeds; otherwise, n is not prime and a serves as a witness to its compositeness.

Their algorithm introduced an alternative quantum primality test that leverages quantum order finding to directly prove that an integer is prime by identifying an element $a \in \mathbb{Z}_n^*$ with order $n - 1$. The quantum process utilized the same circuits as Shor's algorithm and could act as a preliminary step in quantum-based factorization. Shor's algorithm assumes its input n is not of the form p^k or $2p^k$ for a prime p and an integer $k \geq 1$. For odd n, the focus shifts to identifying primes. Efficient classical methods exist for detecting prime powers p^k for $k \geq 2$, though their quantum test can also be adapted for this purpose.

Specifically, the proposed method demonstrates that an integer $n > 1$ possesses a primitive root of order $\varphi(n)$ only when $n \in \{2, 4, p^k, 2p^k\}$. In these cases, there are $\varphi(\varphi(n))$ such roots. The analysis applies similarly to this context. For $n = p^k$, $\varphi(n) = p^{k-1}(p - 1)$. If a primitive root is found, its order $\text{ord}(a)$ satisfies $(\text{ord}(a), n) = p^{k-1} \neq 1$, which can be used to factor n. Verification can be performed by repeated division $p = (\text{ord}(a), n)$, confirming n as a prime power with a polynomial number of operations in terms of the bit length of n. The bound of $\log n$ order-finding steps holds, and the probability of locating a primitive root is $\frac{\varphi(\varphi(n))}{\varphi(n)}$. Performing $3 \log \log \varphi(n)$ repetitions ensures a high likelihood of finding a valid basis. Given $\varphi(n) \leq n - 1$, the bounds for primes also apply in these cases.

They claimed that the algorithm decreased the asymptotic complexity of the Chau-Lo quantum primality test from $O((\log n)(\log \log n)n^3)$ to $O((\log \log n)(\log n)^3 n^2)$. This improvement came with the trade-off of replacing the classical primality certificate, which included the factors of $n - 1$, with a quantum certificate consisting of an element a of order $n - 1$ that can be validated on a quantum computer. The test certifies the primality of an integer with certainty and can be combined with the Miller-Rabin test in initial screening to identify composite numbers with high probability. The complexity of the test is comparable to classical compositeness tests that achieve a negligible error probability, and it is asymptotically quadratic, which is more efficient than classical primality tests that prove certainty. Such tests are typically limited to special forms of integers or, in the case of the AKS primality test, require a number of operations polynomial in the number of bits, specifically $O((\log^6 n))$ for an n-bit integer.

4.1.3 Arlini and Hosoya Algorithm

The authors introduced a method [64] to construct a quantum variant of the classical probabilistic algorithms formulated by Rabin. Their approach heavily utilized fundamental components of known quantum networks, specifically Grover's operator for quantum database search, Shor's Fourier transform for determining the periodicity of a function, and their integration in the counting algorithm referenced in [65]. Their quantum algorithm for primality testing significantly relied on the COUnT algorithm to estimate the number of witnesses to the compositeness of k, utilizing $R \log P$ ancilla qubits $|u_{m,i}\rangle \sim |m\rangle \otimes |P\rangle$, where $i \in [0, P)$ and P is an integer power of 2 determined later. These qubits, which are ultimately measured, are crucial for enhancing the constructive interference of the "good" states.

The COUnT algorithm can be succinctly summarized by the following sequence of operations:

1. Initialize the quantum state:
$$|u_0\rangle = \sum_m |m\rangle|a\rangle,$$
where $|m\rangle$ represents the computational basis states of the counting register and $|a\rangle$ encodes ancillary information.
2. Apply the unitary operator F and the controlled operation G_m that depends on m:
$$(F \otimes I) \left(\sum_m |m\rangle \, |G_m\rangle|a\rangle \right).$$
3. Measure the counting register $|u_m\rangle$ to extract the desired information.

Here, F is the quantum Fourier transform applied to the counting register, and G_m is a unitary operator that encodes the relevant problem-specific information for each m.

The proposed quantum algorithm for determining the primality of a number k is probabilistic in the following manner: if the final measurement of the $R \log P$ ancilla qubits yields at least one qubit that differs from $|u_0\rangle$, it can be concluded with certainty that

k is composite; conversely, if all ancilla qubits are in the state $|u_0\rangle$, the authors assert an error probability smaller than $O(P^{-2R})$ that k is prime. The application of $R \log P$ ancilla qubits and the repeated use of Fourier transforms aim to refine the constructive interference effects based on the measurement of the period u_k, thereby serving as the quantum counterpart to the multiple random tests employed in the classical primality algorithm by Rabin.

Furthermore, if the final measurement step is omitted, the algorithm remains unitary and fully reversible, allowing it to be integrated as an intermediate unitary transformation within a larger, more intricate algorithm.

Let $W_k(a)$ be a function that maps quantum states $|a\rangle$ to $\{0, 1\}$, indicating whether a certain condition k is satisfied, and let f_k be a corresponding numerical value associated with the state $|a\rangle$ (e.g., a frequency, count, or score). The domain of $W_k(a)$ can then be partitioned into two sets:

$$\{|a\rangle : W_k(a) = 1, f_k = 0\} \quad \text{and} \quad \{|a\rangle : W_k(a) = 0, f_k > P/3\}.$$

This partitioning is used to guide subsequent operations in the algorithm.

The authors noted that utilizing the COUnT algorithm for primality testing is not strictly necessary. Instead, one could construct the state

$$|a\rangle|uW_k(a)\rangle$$

for the bases a of the integer k, measure the ancilla qubit $|uW_k(a)\rangle$, and repeat this process h times. This would yield the same error probability $P_e \sim 2^{-2h}$ (after h trials), computational complexity

$$S_{\text{quantum}} = O(\text{poly}(\log k))$$

and required memory space

$$M = 5 \times \log k + 11$$

as in the algorithm with $R = 1$, $P = O(1)$ repeated h times, and the classical algorithm of Ref. [66]. However, this method of repeated trials does not maintain the unitarity and reversibility of the primality testing subroutine.

Consequently, the computational complexity of the quantum algorithm can be expressed as

$$S_Q = O\left(\log n \log Q + (\log^2 Q) Q (\log n S_1)\right),$$

where Q denotes the number of distinct quantum states involved in the subroutine loop for bases a, and for the subloop:

$$S_1 = O\left(\log n \log P + P S_G + (\log^2 P)\right),$$

and $S_G = O\left(\text{poly}(\log n)\right)$, ultimately leading to the polynomial complexity

$$S_Q = O(QPS_G) = O\left(\text{poly}(\log n)\right).$$

4.1.4 Dos Santos and Maziero Algorithm

The authors in [67] demonstrate that the entanglement dynamics of two harmonic oscillators, initially in a separable-coherent state, can identify prime numbers. To support this, they introduce a generalized framework. Finally, they detail a deterministic algorithm for implementing this concept on scalable, fault-tolerant, qubit-based quantum computers. They claimed that the diagonal unitary operations utilized in their algorithm demonstrate a polynomial-time complexity of degree two, in contrast to the exponential complexity associated with general diagonal unitaries [67].

Their methodology consists of several key steps. First, it adapts the definitions to suit the characteristics of qubit-based quantum computers. Next, it selects an appropriate initial state that can be prepared efficiently. Then, it uses the techniques described in Welch et al. [68] to prepare an evolved state, leading to a surprising reduction in gate costs and achieving exponential savings compared to the general case. Following this, it measures the reduced purity, a process that can be performed efficiently as noted in [69]. Afterward, it computes the Fourier modes of the reduced purity function through numerical integration methods, as presented by [70]. With a dataset that includes all points within the range n, their algorithm allows for the deterministic identification of Fourier modes related to prime numbers, enabling the differentiation between prime and composite numbers. They also quantify the number of gates used at each step, focusing particularly on Z-rotations, Controlled-NOT gates, and Hadamard gates.

By utilizing a bipartite Hamiltonian and examining the Fourier modes of the reduced purity, they can differentiate between prime and composite numbers within the range $2 \leq n \leq 2(d-1)$. The implementation of a qubit-based system requires transforming a 2-qudit system into a qubit system, and the corresponding unitary gate can be realized in polynomial time, which is a significant improvement over the anticipated exponential gate requirements. The quantum circuit operates in three stages, with the total gate

Algorithm 14 Carlini-Hosoya (Quantum Miller-Rabin) Test

Input: Odd integer n, precision parameters P and R.
Output: "prime" or "composite".
Step 1: Quantum State Preparation
Initialize $R \log P$ ancilla qubits and the counting register:

$$|u_0\rangle = \sum_m |m\rangle|a\rangle$$

Step 2: COUnT Algorithm (Quantum Amplitude Estimation)
for each iteration $i = 1$ to R **do**
    Apply the **Grover Operator** G as a controlled operation G_m:
    This operator encodes the Miller-Rabin witness condition $W_n(a)$.
    Apply the **Quantum Fourier Transform** F to the counting register:

$$|\Psi\rangle = (F \otimes I) \sum_m |m\rangle G_m |a\rangle$$

end for
Step 3: Measurement and Decision
Measure the $R \log P$ ancilla qubits.
if at least one measured qubit differs from the state $|u_0\rangle$ **then**
    return n is **Composite** (with certainty).
else
    return n is **Prime** (with error probability $O(P^{-2R})$).
end if
Note: If measurement is omitted, the process remains a **Unitary Transformation** that can be used as a subroutine in larger quantum algorithms.

cost demonstrating quadratic scaling relative to the number of digits in $n = 2(d-1)$. The authors noted that while implementation on quantum hardware is achievable, it encounters challenges such as qubit connectivity. However, alternatives such as trapped ion quantum computers or modified methods for gate preparation and reduced purity measurement could potentially address these issues.

Algorithm 15 Dos Santos and Maziero (Entanglement-Based) Primality Test

Input: Range of integers $2 \leq n \leq 2(d-1)$ to be tested.
Output: "prime" or "composite".
Step 1: System Initialization
Map a 2-qudit system into a $2 \log_2 d$ qubit-based system.
Prepare the initial separable-coherent state efficiently using Hadamard and rotation gates.
Step 2: Quantum Evolution
Apply the bipartite Hamiltonian evolution through a diagonal unitary operator U.
Note: Gate cost scales polynomially ($\mathcal{O}(\text{poly}(\log n))$) using Welch's method for diagonal unitaries.
Step 3: Purity Measurement
Measure the **reduced purity** of one of the subsystems (using the direct measurement protocol [69]).
Step 4: Classical Post-processing
Collect the purity values for all points within the range n.
Compute the **Fourier modes** of the reduced purity function using numerical integration.
Step 5: Identification
if Fourier mode corresponds to a peak at n **then**
    Declare n is **Prime**.
else
    Declare n is **Composite**.
end if

Table 4 compares the four quantum primality testing algorithms presented in this section.

Table 4: Comparative Analysis of the Four Quantum Primality Testing Algorithms.

Algorithm	Quantum Primitive	Type	Complexity
Donis-Vela	Order Finding	Probabilistic	$\mathcal{O}(\log^3 n)$
Chau and Lo	Shor's Factorization	Probabilistic	$\mathcal{O}(\log^3 n \log\log n)$
Carlini-Hosoya	Amplitude Estimation	Probabilistic	$\mathcal{O}(\text{poly}(\log n))$
Dos Santos	Entanglement Dynamics	Deterministic	$\mathcal{O}(\log^2 n)$ (Quadratic)

4.2 Quantum Integer Factoring Algorithms

Throughout the discussion of quantum integer factoring algorithms, the following symbols and notations are used.

- N – The composite integer to be factored.
- n – the number of bits in the binary representation of N.
- p, q: Prime factors of the composite number N.
- a: A randomly selected integer coprime to N.
- r: The period; the smallest positive integer such that $a^r \equiv 1 \pmod N$.
- $(\mathbb{Z}/N\mathbb{Z})^\times$: The multiplicative group of integers modulo N.
- $o(x)$: The multiplicative order of x in the group $(\mathbb{Z}/N\mathbb{Z})^\times$.
- d: The bit size of the discrete logarithm.
- $|g\rangle$: A quantum state corresponding to group element g.
- $|f\rangle$: A superposition state encoding the function f.
- b: Bandwidth parameter in banded QFT implementations.
- $P(n, b)$: Performance ratio comparing banded QFT to full QFT.
- s: Tunable tradeoff factor in hybrid quantum-classical algorithms (e.g., Ekerå's work).
- M: Modulus used in the quantum Fourier transform, typically $M = N^2/c$.
- q: Power-of-two modulus in standard Shor's algorithm ($n^2 \leq q < 2n^2$).
- L: Subexponential factor, $L = \exp((\log n)^{1/3}(\log\log n)^{2/3})$.
- $P'(c)$: Success probability in modified algorithms.
- ξ_b: Scaling factor in exponential regime for banded QFT.

Quantum algorithms have significantly advanced the field of integer factoring, which is crucial for cryptographic systems. One of the most notable algorithms is Shor's algorithm, which efficiently factors large integers and solves the discrete logarithm problem exponentially faster than classical algorithms. Another important quantum algorithm is Grover's algorithm [71] which is primarily used for search. This breakthrough poses a significant challenge to traditional cryptographic methods.

Quantum integer factoring algorithms, while diverse, are primarily built upon Shor's algorithm and its extensive body of theoretical, simulated, and experimental work. For clarity, we categorize these algorithms into four groups based on their focus and development: a) Shor's algorithm and theoretical tuning, b) Shor's algorithm over simulation, c) implementation of Shor's algorithm on gate-based quantum computers, and d) alternative quantum integer factoring algorithms. Before we delve into these details, we introduce Shor's algorithm in the following subsection.

4.2.1 Shor's Algorithm

Shor's algorithm is a quantum algorithm designed to factorize large integers efficiently. It operates by leveraging quantum parallelism and the quantum Fourier transform (QFT) to determine the periodicity of a specific function, which in turn reveals the factors of the given number N. Below, we outline the steps of Shor's algorithm, followed by a numerical example to demonstrate its workings.

1. Input and Initial Checks: Let N be the integer to factorize. If N is even or a power of a prime, output the factor directly. Otherwise, proceed to the next steps.
2. Random Selection: Randomly select an integer a such that $1 < a < N$. Compute $\gcd(a, N)$:
   - If $\gcd(a, N) > 1$, return $\gcd(a, N)$ as a factor of N.
   - Otherwise, continue to the next step.
3. Period Finding (Quantum Part): Construct the quantum state and apply QFT to find the period r of the function $f_a(x) = a^x \mod N$. The period r is the smallest integer such that $a^r \equiv 1 \pmod N$.

4. Validation of r: If r is odd or if $a^{r/2} \equiv -1 \pmod{n}$, repeat the process with a different a.
   Determine Factors: If r is even, compute $\gcd(a^{r/2} \pm 1, N)$. These values yield the nontrivial factors of N.

Without loss of generality, we give the following numerical example that demonstrates the algorithm by factoring $n = 15$.

1. Initial checks $N = 15$ is neither even nor a power of a prime, so we proceed.
2. Random Selection: Choose $a = 2$. Compute $\gcd(2, 15) = 1$. Since a is coprime to N, continue.
3. Compute $f_a(x) = a^x \mod N$:
$$f_2(0) = 1, f_2(1) = 2, f_2(2) = 4, f_2(3) = 8, f_2(4) = 1 \mod 15.$$
4. The function $f_2(x)$ repeats with a period $r = 4$.
5. Validate r: The period $r = 4$ is even.
6. Compute $a^{r/2} \mod n$: $2^{4/2} = 4 \mod 15$.
7. Now, calculate:
$$\gcd(4 - 1, 15) = \gcd(3, 15) = 3, \quad \gcd(4 + 1, 15) = \gcd(5, 15) = 5.$$
8. Factors Found: The nontrivial factors of n are 3 and 5, so $N = 15 = 3 \times 5$.

The worst-case complexity of Shor's algorithm is given by: $O\left(n^2 \log n \log\log n\right)$, where n is the number of bits needed to represent the number N. This polynomial time complexity contrasts with the best-known classical algorithms for factoring, which operate in exponential time: $O\left(e^{cn^{1/3}(\log^{2/3} n)}\right)$, where c is some constant. Giving Shor's algorithm a significant speedup over classical methods.

4.2.2 Shor's Algorithm and Theoretical Variants

The theoretical advancements to Shor's algorithm have driven significant improvements in both its resource efficiency and practical implementation, focusing on algorithmic modifications that address qubit count, circuit depth, and physical feasibility (Table 5). The work of R. Van Meter et al. [72] demonstrates that the execution of modular exponentiation, a computational bottleneck in Shor's algorithm, can be accelerated significantly through architectural considerations. They show that, for a storage space of $100n$, which is 20 times the minimum $5n$, modular exponentiation can be executed 200 to 700 times faster than the optimized baseline algorithms when applied to $n = 128$-bit numbers. Importantly, their results highlight how neighbor-only architectures are constrained to $O(n)$-time addition, while non-neighbor architectures achieve $O(\log n)$, demonstrating that physical characteristics of quantum architectures play a pivotal role in determining real-world performance. Despite these modifications, the underlying modular exponentiation complexity remains $O(n^3)$.

A cornerstone of Antipov et al.[73] contribution is the simplified transpilation protocol, which translates operations into sets of gates, 3-qubit Toffoli, CNOTs, and single-qubit operations, while minimizing circuit depth and gate count. For a system with $N = 2^5$, they demonstrate that Shor's algorithm requires 71,452 two-qubit native gates and achieves a circuit depth of $48,615 \times 3$. Resource estimation in their work highlights the scalability of Shor's algorithm for contemporary quantum devices, providing a roadmap for practical implementations.

Building on these developments, Gouzien and Sangouard [74] propose an innovative hybrid quantum computing architecture that combines quantum processors with multimode memory units. This architecture dramatically reduces the required number of physical qubits for factoring large integers. For example, they establish that 2048-bit RSA integers can be factored in 177 days using only 13,436 physical qubits, compared to the significantly higher qubit counts required by conventional planar grid systems. This efficiency is achieved through temporal and spatial multiplexing of qubit states within the memory, enabling minimal processor qubit usage while preserving operational reliability and low error rates.

Knill [75] proposes a variant of Shor's algorithm designed to increase the success probability of order finding by optimizing (QFT) modulus. While the standard algorithm utilizes a power-of-two modulus q (where $n^2 \leq q < 2n^2$), Knill explores alternative moduli to enhance the overlap between the measured state and the ideal order information. This modification improves the likelihood of a successful measurement in a single run, though it introduces greater complexity in the modular arithmetic and the phase estimation subroutine.

By shifting from Shor's standard one-dimensional period finding to a multidimensional lattice-based framework, Regev [10] proposed an efficient quantum factoring . The algorithm significantly reduces the number of quantum gates and iterations required for integer factorization. Specifically, the algorithm factors n-bit integers by running a quantum circuit with $\widetilde{O}(n^{3/2})$ gates, compared to shor's $O(n^2)$, independently for $\sqrt{n} + 4$ times, followed by polynomial-time classical post-processing. However, this comes at the cost of requiring more qubits ($O(n^{3/2})$). The correctness of the algorithm is based on a number-theoretic heuristic assumption reminiscent of those used in subexponential classical factorization algorithms. Regev's breakthrough is considered a foundational paper for the subsequent research [76, 77, 78, 79, 80, 81]

Building on Regev's foundational result, Ragavan and Vaikuntanathan [82, 76] provided two substantial improvements to the algorithm, addressing its space efficiency and its noise tolerance. Their main contribution was the construction of a quantum factoring

circuit that uses $O(n \log n)$ qubits and $O(n^{3/2} \log n)$ gates, thereby combining the gate-count advantage of Regev's algorithm with the space efficiency of Shor's original circuit (up to a logarithmic factor). This was achieved by implementing efficient and reversible exponentiation using Fibonacci numbers in the exponent, rather than the standard powers of two, adapting classical reversible techniques by Kaliski [83] to the quantum setting. Their second contribution demonstrated that Regev's classical post-processing procedure can be modified to tolerate a constant fraction of corrupt quantum circuit runs, using lattice reduction techniques to detect and filter out erroneous samples. In Regev's original analysis, all $\approx \sqrt{n}$ circuit runs were required to succeed, a condition that is difficult to guarantee on near-term hardware. The noise-robust variant relaxes this requirement, substantially improving the practical viability of the algorithm.

In a parallel development, Pilatte [78] resolved one of the principal open questions surrounding Regev's algorithm by providing an unconditional proof of its correctness. Regev's original analysis relied on a number-theoretic conjecture regarding the representability of elements in $(\mathbb{Z}/N\mathbb{Z})^\times$ as short products of small primes. Pilatte proved a version of this conjecture using tools from analytic number theory, specifically zero-density estimates for Dirichlet L-functions. As a consequence, the correctness of both Regev's algorithm and the space-efficient variant by Ragavan and Vaikuntanathan was established unconditionally, up to logarithmic factors. Moreover, these results extend naturally to the discrete logarithm variants proposed by Ekerå and Gärtner.

Ekerå and Gärtner [77] extended Regev's factoring algorithm to compute discrete logarithms in a natural way. Specifically, they adapted the multi-dimensional structure of Regev's algorithm to the discrete logarithm problem (DLP) in cyclic subgroups of $\mathbb{Z}_N^*$, and further discussed extensions to order finding and complete factorization via order finding. For all of these algorithms, they provided detailed practical implementation considerations, including the robustness of the post-processing step. Their extension is significant because it demonstrates that the algorithmic improvements introduced by Regev are not limited to the factoring problem but generalize to the broader class of cryptanalytic tasks that underpin modern public-key cryptography.

Subsequently, Ekerå and Gärtner [80] published a high-level cost comparison between Regev's quantum algorithm (with their extensions), and shor-based algorithms for factoring and computing discrete logarithms. Targeting cryptographically relevant problem instances and accounting for the space-saving optimizations of Ragavan and Vaikuntanathan, their analysis concluded that Regev's algorithm without space-saving optimizations may achieve a per-run advantage but not an overall advantage if non-computational quantum memory is inexpensive. With the space-saving optimizations applied, Regev's algorithm does not achieve an advantage, as it uses more computational memory and performs more work per run compared to the shor's based algorithms. This finding provides an important practical perspective on the relative merits of the new algorithmic family.

A major advance in reducing qubit requirements was achieved by Chevignard et al. [79], who focused on minimizing the number of logical qubits needed for quantum factorization. Their approach combines a novel arithmetic circuit based on the Residue Number System (RNS) with May and Schlieper's output truncation technique and the Ekerå–Håstad variant of Shor's algorithm. Using only $d + o(n)$ work qubits, where d is the bit size of the discrete logarithm, they demonstrated that n-bit RSA moduli can be factored using $n/2 + o(n)$ qubits. For RSA-2048, they estimated that only 1,730 logical qubits would suffice for a single run, requiring approximately 2^{36} Toffoli gates, with the algorithm needing on average 40 independent runs to succeed. While the gate count is significantly higher than in conventional approaches, the dramatic reduction in qubit count represents a fundamentally different point on the space-time tradeoff curve for quantum factoring.

Building directly on these results, Gidney [81] demonstrated that the resource requirements for factoring RSA-2048 can be reduced to fewer than one million noisy physical qubits with an execution time of less than one week. This represents a 20-fold reduction in qubit count compared to the earlier estimate of 20 million qubits by Gidney and Ekerå [84]. The improvement arises from three key innovations: the use of approximate residue arithmetic from Chevignard et al. [79], the storage of idle logical qubits using yoked surface codes, and the replacement of magic state distillation with magic state cultivation. While the runtime increases from approximately 8 hours to under one week, the Toffoli gate count is reduced by over 100-fold relative to the Chevignard et al. [79] approach, achieving approximately 6.5×10^9 Toffoli gates. The assumptions underlying these estimates remain consistent with those in the 2019 work: a square grid of qubits with nearest-neighbor connectivity, a uniform gate error rate of 10^{-3}, a surface code cycle time of 1 microsecond, and a control system reaction time of 10 microseconds.

4.2.3 Shor's Algorithm Over Simulation

The practicality of simulating Shor's algorithm has been greatly enhanced through the development of universal quantum computer simulators, as demonstrated by the work of K. De Raedt et al. [88]. Their simulator efficiently propagates the quantum statevector for systems of up to 36 qubits using up to 4096 processors and 1 TB of memory on high-performance computing platforms such as IBM BlueGene/L, Cray X1E, and clusters of PCs. Remarkably, the simulator exhibits nearly ideal scaling with the number of processors, making it suitable not only for simulating Shor's algorithm but also as a benchmark for testing the computational power of next-generation high-performance computing systems. By leveraging parallelism, their implementation achieves a significant step forward in enabling practical and scalable simulations of large quantum systems.

Building on this, Tankasala and Ilatikhameneh [89] introduce Quantum-Kit (Q-Kit), a desktop application designed for efficient quantum circuit simulations. Q-Kit's computational engine enables simulations of Shor's algorithm for integers up to 24 bits on a modest desktop equipped with an Intel Core i5 processor and 12 GB of RAM. Utilizing Kitaev's one-qubit trick [90], Q-Kit reduces the qubit requirements for the algorithm, allowing the factorization of a 20-bit integer with only 21 qubits in 2.3 minutes, a task previously requiring 60 qubits and a supercomputer. Moreover, the simulation of a 13-bit integer with 39 qubits was shown to be 35x faster than existing methods, highlighting the platform's ability to balance computational power with accessibility. The integration of

Table 5: Comparison of theoretical Advancements to Shor's Algorithm. NAFA stands for Not Apparent to the Authors.

Algorithm Variant	Qubit Requirements	Circuit Size	Key Features	Limitations
Beauregard [85]	$2n+3$	$O(n^3)$	Modular exponentiation via Fourier-transform addition.	High constant factors in gate complexity.
Van Meter [72]	$100n$	$O(n^3)$	Faster modular exponentiation (200×-700× speedup).	Requires non-neighbor architectures.
Fowler *et al.* [86]	$2n+3$	$O(n^3)$	Linear nearest-neighbor implementation.	Performance similar to concurrent architectures.
Parker & Plenio [87]	$\log_2 n+1$	$O(n^3)$	Uses maximally mixed states to reduce prep overhead.	Assumes robust entanglement.
Knill [75]	$O(n)$	$O\big((c+1)^2 n^2 \log n \log\log n\big)$	Boosts success probability via an LCM trick.	Dependent on modular-exponentiation cost.
Gouzien & Sangouard [74]	13,436 (for 2048-bit RSA)	Not explicitly stated	Reduced physical qubit count via temporal and spatial multiplexing.	Long factoring time; practical implementation requires reliable multimode memory.
Regev [10]	$O(n^{3/2})$	$O(n^{3/2}\log n)$	Minimal quantum runs ($\sqrt{n}+4$) with classical post-processing.	Relies on heuristic assumptions.
Ragavan & Vaikuntanathan [76]	$O(n\log n)$	$O(n^{3/2}\log n)$	Improved space efficiency and noise tolereance	Relies on heuristic assumptions; Ancilla qubit overhead.
Pilatte [78]	$O(n\log^3 n)$	$O(n^{3/2}\log^3 n)$	Unconditional proof of Regev's algorithm and subsequent variants.	Chosen parameters for the least quadratic non-residue problem.
Chevignard et al. [79]	$n/2+o(n)$ $\sim$ 1700 logical	$O(n^3)$	Compression technique; reduces factoring RSA-2048 to $\approx$ 1730 logical qubits.	NAFA.
Ekerå & Gärtner [77]	$O(n^{3/2})$	$O(n^{3/2}\log n)$	Extends Regev's algorithm to efficiently compute discrete logarithms (DLP) problem.	Inherits Regev's high space requirements and originally relied on heuristic assumptions.
Gidney & Ekerå [11]	20 million physical $\sim$ 6000 logical	$0.3n^3$ Toffoli gates in 8 hours	A physical resource estimate rather than asymptotic theory.	NAFA.
Gidney [81]	one million Physical $\sim$ 1400 logical	$\sim 6.5\times10^9$ Toffoli in one week	Vulnerable classical PKC systems should be deprecated after 2030 and disallowed after 2035.	NAFA
Ekerå & Gärtner [80]	Varies by algorithm	Varies by algorithm	Regev's factoring algorithm and its variations do not achieve an advantage over Shor-based algorithms for cryptographically relevant problems	NAFA

hybrid circuits, combining quantum and classical operations, further expands the tool's versatility, making it both user-friendly and computationally robust.

Further advancing the field, Willsch et al. [14] demonstrated the potential of GPU-based supercomputers in advancing the simulation capabilities of Shor's algorithm for large semiprimes. Their work surpassed prior limits by factoring numbers as large as $549,755,813,701 = 712,321 \times 771,781$, using iterative Shor's algorithm executed on up to 2048 GPUs. A key finding from their simulations was an unexpected success rate exceeding theoretical predictions due to lucky cases, where a factor could still be derived even when sufficient conditions were not met. Moreover, they highlighted the robustness of their post-processing technique, which could raise the success probability close to unity with a single quantum run. The study also noted that Shor's algorithm displayed resilience against quantum hardware errors, reinforcing its viability for future large-scale cryptographic applications.

Nam and Blümel[91] propose a streamlined version of Shor's algorithm that replaces the full quantum Fourier transform (QFT) with a banded version, where each qubit interacts only with its b nearest neighbors. This modification significantly reduces hardware requirements while preserving computational accuracy. The performance of this banded QFT is measured by $P(n,b)$, the ratio of success rates for the banded and full QFT versions. In the non-exponential regime ($n < n_t(b)$), the performance scales as:

$$P(n,b) \approx \exp\left[-\frac{\varphi^2_{\max}(n,b)}{100}\right],$$

while in the exponential regime ($n > n_t(b)$), the scaling is given by:

$$P(n,b) \approx 2^{-\xi_b(n-8)},$$

where $n_t(b)$, the transition point between regimes, is approximately:

$$n_t(b) \approx b + 5.9 + \sqrt{7.7(b+2)} - 47 \quad \text{for } b \geq 8.$$

Here, $\xi_b \approx 1.1 \times 2^{-2b}$ reflects the scaling factor. For large n, the performance measure in the exponential regime follows:

$$P(n,b) \approx 2^{-\xi_b^{(a)} n}, \quad \text{where } \xi_b^{(a)} \approx \frac{\pi^2}{12\ln(2)} \times 2^{-2b} \approx 1.19 \times 2^{-2b}.$$

This analytical and numerical model accurately predicts the scaling behavior of Shor's algorithm in both regimes, ensuring efficient utilization of quantum hardware. For practical applications, it is shown that a bandwidth $b = 8$ is sufficient to factor RSA-2048 numbers with a 95% success rate, achieving significant reductions in hardware requirements.

Another approach to simulating Shor's algorithm utilizes so-called Feynman simulators, which prioritize accessing certain amplitudes from the full statevector, potentially requiring fewer computational resources. These simulators are often based on tensor networks or matrix product states, leveraging advanced mathematical frameworks to reduce the space complexity while maintaining the algorithm's accuracy.

D. S. Wang *et al.* [92] demonstrated that under the matrix product state (MPS) formalism, the states produced in Shor's algorithm can be represented using $O(\max(4lr^2, 2^{2l}))$ space, where l is the number of bits in the number to factorize, and r is the order found in the related order-finding problem. This formalism significantly reduces the space complexity from the $O(2^{3l})$ required by amplitude formalism approaches, allowing simulations as large as 42 qubits on a single processor with 32GB RAM. Moreover, their implementation on a distributed memory environment enabled simulations of up to 45 qubits using 8 cores and 16GB RAM in approximately one hour.

Further expanding on tensor network methods, Y. Zhao *et al.* [93] proposed a distributed computing scheme for tensor network contraction based on the density matrix. Their algorithm segments large undirected graphs with high tree width into smaller subgraphs, enabling distributed contraction on a supercomputer. Using this method, they successfully simulated a 120-qubit 3-regular QAOA algorithm on a 4096-core supercomputer. The strategy optimized tree width reduction through heuristic edge-cutting algorithms, ensuring scalability and efficient use of computational resources.

Ekera's algorithm is particularly relevant for cryptographic tasks, such as discrete logarithms in Schnorr groups, where the group order is unknown. It achieves a significant improvement in resource efficiency compared to traditional methods. The algorithm also introduces a flexible tradeoff mechanism, where the number of runs required scales asymptotically as $s + 1$ for a fixed tradeoff factor s, as the problem size m, defined as the smallest integer such that $2^{m-1} \leq r < 2^m$, with r representing the order of the group, is increased. This parameter m, reflecting the bit length of the group order, plays a crucial role in the algorithm's efficiency. For example, with lattice-based post-processing, a single run is sufficient to achieve a success probability greater than 99% if a small number of lattice vectors are enumerated.

To support the analysis and implementation of these algorithms, the Qunundrum software suite [94] was developed. This tool enables the simulation of probability distributions generated by Shor's, Seifert's, and Ekerå's methods, providing valuable insights into their performance under various tradeoff conditions. These simulations facilitate both theoretical exploration and practical application, particularly in quantum cryptographic settings (Table 6).

Table 6: Simulation Tools and Capabilities for Shor's Algorithm

Tool Name	Qubit Limit	Runtime Performance	Memory Usage	Notable Applications
De Raedt's Simulator	Up to 36 qubits	Nearly ideal scaling on 4096 processors	1 TB	Benchmarking high-performance computing systems
Quantum-Kit (Q-Kit)	24-bit integers	2.3 min for 20-bit integers on a desktop	12 GB	Accessible desktop simulations with Kitaev's trick
Tensor Network Simulators	42 qubits (single), 45 qubits (distributed)	Approx. 1 hour on 8 cores	32 GB (single), 16 GB (distributed)	Distributed-memory simulations for order-finding
MPS-Based Simulators	120 qubits	Scalable on 4096-core supercomputers	Varies	Simulating QAOA and large-scale tensor networks

4.2.4 Implementations of Shor's algorithm on Gate-Based Quantum Computers

Implementing Shor's algorithm on existing gate-based quantum computers has been a critical benchmark in the progress of quantum computation. Many implementations have simplified the experimental setup by utilizing prior knowledge of the factors, reducing the complexity of the algorithm for demonstration purposes. Despite these adaptations, experimental realizations have steadily advanced, showcasing innovative methods to address the inherent challenges of scaling and precision while providing valuable insights into the capabilities of gate-based quantum systems.

Martín-López et al. [95] pioneered a resource-efficient approach to Shor's algorithm by introducing qubit recycling. In their implementation, the standard n-qubit control register was replaced with a single qubit that was reused n times, reducing the overall number of qubits required by a factor of three. Using a two-photon system, they successfully factored $N = 21$, with the algorithmic output clearly distinguishable from noise. This technique demonstrated that significant resource reductions could be achieved without sacrificing the fundamental characteristics of the algorithm, marking a step toward scalable implementations applicable to various physical architectures.

Monz et al. [96] extended this progress by implementing a scalable version of Shor's algorithm on an ion-trap quantum computer. Their realization employed seven logical qubits and four auxiliary cache qubits to perform modular arithmetic operations required for factoring $N = 15$. This implementation achieved a success probability exceeding 90%, demonstrating the potential of ion-trap systems for precise quantum control. Fidelity assessments for key components, including modular multipliers and controlled quantum operations, highlighted the challenges of maintaining accuracy in more complex circuits, with fidelities ranging between 68% and 86%.

Further exploration was conducted by Amico et al. [12], who implemented Shor's algorithm on IBM's superconducting quantum hardware. By employing a semi-classical quantum Fourier transform, they minimized qubit requirements, enabling the factorization of integers $N = 15, 21$, and 35. The study highlighted the growing limitations of noisy intermediate-scale quantum (NISQ) devices, as errors from two-qubit gates accumulated, impacting the success rate for larger values of N. Nevertheless, their approach demonstrated that quantum algorithms could be adapted to function effectively on current quantum hardware with careful optimization.

Vandersypen et al. [97] provided one of the earliest demonstrations of Shor's algorithm using nuclear magnetic resonance (NMR) technology. Their implementation, which factored $N = 15$, utilized seven nuclear spin-1/2 nuclei as qubits, manipulated at room temperature. While the scalability of NMR systems remains limited, their work demonstrated the feasibility of precise quantum control and served as a foundational example of experimental quantum computation.

Photonic systems have also played a significant role in demonstrating Shor's algorithm. Politi et al. [98] successfully implemented a compiled version of the algorithm on an integrated photonic chip, factoring $N = 15$ using four single-photon qubits. Similarly, Lanyon et al. [99] demonstrated the algorithm in a photonic system, employing coherent control and exploiting quantum entanglement to achieve the necessary computational steps. These studies emphasized the potential of photonic platforms for scalable quantum computing, leveraging their inherent resilience to noise and ability to maintain entanglement over extended periods.

Skosana and Tame [100] demonstrated Shor's algorithm on IBM quantum processors, factoring $N = 21$. Their implementation introduced a novel use of approximate Toffoli gates with residual phase shifts, enabling functional correctness while reducing hardware demands. This proof-of-concept highlighted the potential of hybrid approaches that balance algorithmic complexity and hardware limitations to achieve practical outcomes.

Table 7 shows a comparison of true vs. compiled implementations of Shor's algorithm.

Table 7: Comparison of True vs. Compiled Implementations of Shor's Algorithm

Implementation	**Prior Knowledge Used**	**Qubit Resources**	**Period-Finding?**	**Remarks**
Martín-López et al. [95]	No	Reduced via qubit recycling	No	Demonstrates scalability with two-photon system for $N = 21$.
Monz et al. [96]	No	7 logical + 4 auxiliary qubits	Partially	Scalable ion-trap implementation with success rate > 90 %.
Amico et al. [12]	No	Reduced using semi-classical QFT	No	Highlights NISQ limitations for $N = 15, 21, 35$.
Smolin et al. [31]	Extensive (known solution)	Only 2 coherent qubits	No	Critiques misleading demos; focuses on true period-finding vs. mere control.
Vandersypen et al. [97]	Yes (simplified alg.)	7 nuclear spins (NMR)	Partially	Early NMR demo factoring $N = 15$, highlights control techniques.
Politi et al. [98]	Yes (compiled alg.)	4 photonic qubits	Partially	Integrated photonic chip for $N = 15$, demo of quantum control.
Lanyon et al. [99]	Yes (precomputed circuit)	Single-photon system	Partially	Emphasizes entanglement/control for $N = 15$ over scalability.
Skosana & Tame [100]	Yes (approximate gates)	5 qubits	Partially	Approximate Toffolis for $N = 21$; limited by hardware noise.
De Raedt et al. [88]	No	Up to 36 qubits	Yes	Universal simulator with near-ideal scaling; HPC benchmarks.
Tankasala & Ilatikhameneh [89]	No	21 qubits (24-bit ints)	Yes	Demonstrates Kitaev's trick; efficient desktop sims.
Nam & Blümel [91]	No	Bandwidth-limited QFT	Yes	RSA-2048 factorization with $b = 8$, 95 % success.
Ekerå et al. [11]	No	Fully parallelized runs	Yes	Trade-offs for discrete logs and order finding.
Willsch et al. [14]	No	Simulated up to 5.5×10^{11} using GPUs	Yes	> 50 % success via lucky cases; post-processing for large semiprimes.

4.2.5 Alternative Quantum Factoring Algorithms

Recent advancements in quantum algorithms have proposed alternative approaches to solving the factoring problem using gate-based quantum computers. Among these, Martin Ekerå's work stands out for its innovative use of Shor's discrete logarithm quantum algorithm.

Ekerå [11] extended Shor's discrete logarithm quantum algorithm by enabling it to compute both the order and discrete logarithm in cyclic groups simultaneously. By introducing a tunable tradeoff factor s, his algorithm balances quantum and classical resources, requiring exponents of size $m + 2m/s$ bits to optimize computational efficiency. Building upon this, Ekerå and Håstad [101] further advanced the framework by capturing the probability distribution induced by the quantum algorithm, enabling efficient simulation and analysis of outputs. Their hybrid approach combines quantum runs with classical post-processing to achieve success probabilities exceeding $1/8$, providing a robust and flexible solution to m-bit discrete logarithm problems. These contributions significantly enhance the practicality of quantum algorithms for cryptographic applications like RSA and Diffie-Hellman, offering an efficient and adaptable framework for solving complex discrete logarithm and order-finding problems. Moreover, Ekerå and Håstad [102] in their work, reduced the quantum exponentiation requirements for factoring m-bit integers. By reformulating the problem as a short discrete logarithm problem, the quantum exponentiation length is reduced from $2m$ bits, as required in Shor's original algorithm, to approximately $\left(\frac{1}{2} + \frac{1}{s}\right) m$ bits, where $s \geq 1$ is a tunable parameter.

Bernstein et al.[28] approach achieved a significant improvements in resource efficiency by leveraging sublinear numbers of qubits, specifically $(\log n)^{2/3+o(1)}$, to factor an integer n in time $L^{q+o(1)}$, where $q = \sqrt[3]{8}/3 \approx 1.387$, and $L = \exp((\log n)^{1/3}(\log\log n)^{2/3})$.

The authors mentioned that while the algorithm's qubit requirements are significantly reduced, the time complexity is asymptotically worse than Shor's algorithm. This tradeoff reflects a central theme of their work: achieving practical quantum computations with limited resources, albeit at the cost of longer runtimes.

Among the foundational contributions to quantum algorithms is the unifying generalization of Shor's algorithm under the framework of the abelian hidden subgroup problem (HSP). As highlighted by Jozsa [103], this problem elegantly generalizes periodicity determination to any finite abelian group. For a group G and a function $f : G \to X$, the goal is to determine the hidden subgroup K, defined as $K = \{k \in G : f(k + g) = f(g), \forall g \in G\}$. This subgroup K encodes the periodic structure of f, much like the cyclic subgroup of multiples of the period r in integer factorization.

Quantum algorithms leverage the Fourier transform over G to extract information about K. Through superposition states such as $|f\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle|f(g)\rangle$, and subsequent transformations, the quantum computer isolates the structure of K. The Fourier transform over G projects the periodic structure of f into a dual group space, allowing uniform sampling of labels directly related to K's generators. This technique underscores the fundamental quantum speedup, reducing the complexity of determining K to $O(\text{poly}(\log |G|))$.

Beyond the traditional framework of the hidden subgroup problem, Li et al. [29] proposed an algorithm tailored to solve the integer square-free decomposition problem, offering an efficient and exact solution, unlike Shor's algorithm, which is probabilistic in nature. Their algorithm uniquely determines the square-free part of a given integer N and tests for square-freeness by leveraging the properties of Gauss sums and employing the quantum Fourier transform (QFT).

Moreover, their algorithm achieves high computational reliability by concluding efficiently in $O((\log\log n)^2)$ steps, bypassing the need for $k = O(\log n)$ iterations.

Table 8 illustrates a comparative analysis of alternative quantum factoring algorithms.

Table 8: Comparative Analysis of Alternative Quantum Factoring Algorithms

Algorithm	Key Approach	Time Complexity	Qubit Requirements	Notable Features
Shor's Algorithm [7]	Quantum period finding via modular exponentiation	$O(\log^3 n)$	$O(\log n)$	Probabilistic; high overhead for modular exponentiation.
Ekerå & Håstad [102]	Hybrid quantum-classical for discrete logarithms	$O(\log^2 n)$	Reduced vs. Shor's	Exact method; tunable s-parameter for tradeoffs.
Bernstein et al. [28]	Adapted Number Field Sieve (NFS)	$L^{1.387+o(1)}$	Sublinear $(\log n)^{2/3+o(1)}$	Low-resource; relies on classical pre-computation.
Jozsa [103]	Hidden subgroup generalization for abelian groups	$O(\text{poly}(\log \lvert G\rvert))$	$O(\log \lvert G\rvert)$	Unified framework for periodicity; foundational quantum speedups.
Li et al. [29]	Square-free decomposition via Gauss sums and QFT	$O((\log\log n)^2)$	Reduced vs. Shor's	Deterministic; simpler modular ops; exact results.

4.3 Experimental Milestones in Quantum Integer Factorization

While the theoretical efficiency of Shor's algorithm is well-established, its physical implementation remains constrained by the coherence times and gate fidelities of current NISQ-era hardware. The first landmark experimental verification occurred in 2001, where IBM and Stanford University utilized a 7-qubit Nuclear Magnetic Resonance (NMR) quantum computer to factor $15 = 3 \times 5$ [104]. This was followed by photonic implementations in 2012, which successfully factored 21 using a compiled version of the algorithm that reduced the required number of qubits through specific prior knowledge of the number's structure [105].

Subsequent advancements have utilized diverse architectures, including trapped-ion systems and superconducting circuits. In 2016, researchers implemented a scalable version of Shor's algorithm on a trapped-ion quantum computer, factoring 15 with high confidence and demonstrating the potential for modular scalability [106]. More recently, the focus has shifted toward hybrid algorithms such as Variational Quantum Factoring (VQF). These approaches map the factoring problem onto an Ising Hamiltonian, allowing for the factorization of larger integers, such as $1,099,551,473,989$, on noisy hardware by leveraging classical optimization loops to reduce quantum resource requirements [107]. These experimental efforts highlight a critical transition from proof-of-concept demonstrations to the development of noise-resilient techniques aimed at scaling toward cryptographically relevant bit-lengths.

4.4 Quantum Factorization Summary

Quantum algorithms, spearheaded by Shor's groundbreaking work and Regev's algorithm on integer factorization and discrete logarithms, have catalyzed significant progress in the fields of computational complexity and cryptography. This work has explored the theoretical underpinnings, practical implementations, and novel adaptations of Shor's algorithm, alongside alternative approaches that extend its utility or address its limitations. From early demonstrations leveraging photonic, NMR, and ion-trap systems to contemporary advances in hybrid quantum-classical frameworks, the evolution of factoring algorithms has highlighted the synergy between algorithmic innovation and hardware capabilities.

One notable theme in recent research is the shift toward resource-efficient quantum algorithms. Bernstein's and Li's methods exemplify this trend [28, 108], prioritizing reduced qubit and computational demands while maintaining or improving upon the performance of traditional algorithms like Shor's. These methods underscore the importance of tailoring algorithms to current and near-term quantum hardware capabilities, emphasizing practicality without sacrificing theoretical rigor.

Another key insight from this work is the emphasis on tradeoffs in algorithm design. Efforts such as Bernstein's focus on balancing quantum and classical computational resources, while Ekerå's parameterized approach introduces tunable factors to optimize resource usage [11]. These tradeoffs highlight a broader trend in quantum algorithm development, one that prioritizes adaptability and resource efficiency to overcome the practical constraints of noisy intermediate-scale quantum (NISQ) devices.

Despite these advancements, significant challenges remain. The inherent limitations of noisy intermediate-scale quantum (NISQ) devices, such as gate fidelity and coherence times, constrain the scalability of quantum algorithms. Furthermore, the exponential resources required for error correction and fault tolerance underscore the need for breakthroughs in both quantum hardware and algorithm optimization.

5 Analysis and Discussion

The security of the RSA cryptosystem is fundamentally tied to the gap between the efficiency of primality testing (used for key generation) and the difficulty of integer factorization (required for cryptanalysis). Based on our findings, we evaluate these implications across three critical dimensions.

5.1 RSA Security: Classical vs. Quantum Models

In the classical model, RSA-2048 is considered secure because the computational effort to factor it using the General Number Field Sieve (GNFS) exceeds the capacity of current supercomputers. To date, the best classical factoring record [6] achieved is an 829-bit RSA integer. However, the quantum model introduces Shor's algorithm, which reduces this problem from sub-exponential to polynomial time complexity $O((\log N)^3)$. While our results confirm that quantum primality testing currently offers no advantage over classical methods like Baillie-PSW, the quantum threat to factorization is significant. Recent improvements, such as Regev's algorithm [10], further reduce gate counts to $\widetilde{O}(n^{3/2})$, accelerating the potential timeline for compromising RSA security margins.

5.2 Estimated Break-even Points

The break-even point for quantum advantage is defined as the threshold where a quantum computer outperforms the best classical approach. For algorithmic break-even, our results show that deterministic tests (AKS) only become more efficient than trial division at approximately 2^{64} on classical hardware. For quantum factoring break-even, factoring classically remains infeasible for RSA-2048. However, current resource estimations suggest that a hybrid quantum architecture using multimode memory could factor a 2048-bit key in approximately 177 days using only 13,436 physical qubits [109]. This indicates that the break-even point for practical quantum cryptanalysis is approaching the 10^4 qubit range.

5.3 Relevance to Post-Quantum Cryptography (PQC)

The scaling of Shor's and Regev's algorithms directly drives the global transition to PQC [10]. The review of alternative quantum factoring algorithms (Table 8) underscores that as hardware approaches the 10^4 qubit threshold, RSA security margins will effectively vanish. This necessitates two immediate shifts: (a) short-term mitigation, increasing RSA key sizes to 4096-bit to extend the "quantum-resistant" window, though this provides only temporary relief due to polynomial scaling; (b) long-term migration, Adopting lattice-based or hash-based primitives that are resistant to the hidden subgroup problem (HSP) framework, which underpins the quantum speedup for factoring.

6 Conclusion and Future Directions

This paper has presented a comprehensive survey and analysis of classical and quantum algorithms for integer factorization and primality testing, highlighting their theoretical foundations, computational efficiency, and cryptographic significance. Our survey starts by presenting a novel classification of 11 classical and four quantum primality testing algorithms to examine advantages, limitations, and speed/accuracy tradeoffs. Then, we explore the significant advances achieved by quantum algorithms in integer factorization, beginning with Shor's algorithm and extending through numerous subsequent refinements aimed at enhancing its efficiency and reducing resource requirements. These developments collectively highlight the profound implications of quantum computing-based computers for modern cryptographic systems.

The journey toward practical quantum factoring is far from over. While theoretical progress continues to refine algorithms, the collaborative efforts of theorists and experimentalists will remain pivotal in bridging the gap between theoretical promise and practical deployment. By leveraging the lessons of past research, the quantum computing community can build a pathway to achieving secure and scalable quantum technologies that redefine computational and cryptographic paradigms.

Future directions in the field should focus on:

- Expanding alternative algorithms: Continued exploration of methods beyond Shor's framework, including those addressing problems outside the hidden subgroup paradigm.
- Refining tradeoff techniques: Developing more sophisticated parameterized models and hybrid quantum-classical strategies to optimize resource allocation.
- Optimizing modular arithmetic: Developing efficient algorithms for modular exponentiation and arithmetic operations, which constitute the bottleneck of quantum factoring.
- Enhancing hardware scalability: Prioritizing the development of scalable quantum processors with improved error rates, gate fidelities, and integration of advanced error correction codes.

7 Acknowledgments

The authors would like to thank Onaizah Colleges and the Interdisciplinary Research Center for Intelligent Secure Systems (IRC-ISS), King Fahd University of Petroleum and Minerals (project INSS2510) for their support.

References

[1] Song Y Yan. Primality testing and integer factorization in public-key cryptography. 2009.

[2] Ulrich Daepp and Pamela Gorkin. Fermat's little theorem. In *Reading, Writing, and Proving*, pages 315–323. Springer, 2011.

[3] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[4] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[5] Yumin Dong, Hengrui Liu, Yanying Fu, and Xuanxuan Che. Improving the success rate of quantum algorithm attacking rsa encryption system. *Journal of Applied Physics*, 134(2), 2023.

[6] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. The state of the art in integer factoring and breaking public-key cryptography. *IEEE Security & Privacy*, 20(2):80–86, 2022.

[7] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[8] Maryam Abbasi, Filipe Cardoso, Paulo Váz, José Silva, and Pedro Martins. A practical performance benchmark of post-quantum cryptography across heterogeneous computing environments. *Cryptography*, 9(2):32, 2025.

[9] Anas A Abudaqa, Khaled Alshehri, and Muhamad Felemban. On the homomorphic properties of kyber and mceliece with application to post-quantum private set intersection. *Cryptography*, 9(4):66, 2025.

[10] Oded Regev. An efficient quantum factoring algorithm. *arXiv preprint arXiv:2308.06572*, 2023.

[11] Martin Ekerå. Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. *Journal of Mathematical Cryptology*, 15(1):359–407, 2021.

[12] Mirko Amico, Zain H Saleem, and Muir Kumph. Experimental study of shor's factoring algorithm using the ibm q experience. *Physical Review A*, 100(1):012305, 2019.

[13] Michael R Geller and Zhongyuan Zhou. Factoring 51 and 85 with 8 qubits. *Scientific reports*, 3(1), 2013.

[14] Dennis Willsch, Madita Willsch, Fengping Jin, Hans De Raedt, and Kristel Michielsen. Large-scale simulation of shor's quantum factoring algorithm. *Mathematics*, 11(19):4222, 2023.

[15] Daniel Turyansky, Yehonatan Zolti, Yuval Cohen, and Adi Pick. Pulse optimization in adiabatic quantum computation and control. *Physical Review Research*, 8(1):013206, 2026.

[16] Great Internet Mersenne Prime Search (GIMPS). 2^{136279841}-1 is the new largest known prime number. `https://www.mersenne.org`, 2025. Accessed: 2024-11-20.

[17] EL Roettger and HC Williams. Some remarks concerning the lucas-lehmer primality test. *Journal of Integer Sequences*, 28(2):3, 2025.

[18] Kostas Bimpikis and Ragesh Jaiswal. Modern factoring algorithms. *University of California, San Diego*, 2005.

[19] Mircea Ghidarcea and Decebal Popescu. Prime number sieving—a systematic review with performance analysis. *Algorithms*, 17(4):157, 2024.

[20] Louis Monier. Evaluation and comparison of two efficient probabilistic primality testing algorithms. *Theoretical Computer Science*, 12(1):97–108, 1980.

[21] Renee Marie Canfield. *Three primality tests and maple implementation*. PhD thesis, University of Georgia, 2008.

[22] René Schoof. Four primality testing algorithms. *arXiv preprint arXiv:0801.3840*, 2008.

[23] Masanari Kida. Primality tests using algebraic groups. *Experimental Mathematics*, 13(4):421–427, 2004.

[24] Monica Perrenoud. Randomized and deterministic primality testing. Technical report, 2009.

[25] Cristina-Loredana Duta, Laura Gheorghe, and Nicolae Tapus. Framework for evaluation and comparison of primality testing algorithms. In *2015 20th International Conference on Control Systems and Computer Science*, pages 483–490. IEEE, 2015.

[26] Yasaswy Kasarabada. *A Verilog description and efficient hardware implementation of the Baillie-PSW primality test*. PhD thesis, University of Cincinnati, 2016.

[27] Carla Purdy, Yasaswy Kasarabada, and George Purdy. Hardware implementation of the baillie-psw primality test. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 651–654. IEEE, 2017.

[28] Daniel J Bernstein, Jean-François Biasse, and Michele Mosca. A low-resource quantum factoring algorithm. *Proceedings of Mycrypt*, 2017. Supported by the Horizon 2020 program, NSF grants 1018836 and 1314919, and other institutions.

[29] Jun Li, Xinhua Peng, Jiangfeng Du, and Dieter Suter. An efficient exact quantum algorithm for the integer square-free decomposition problem. *Scientific Reports*, 2:260, 2012.

[30] Noboru Kunihiro. Quantum factoring algorithm: Resource estimation and survey of experiments. In *International Symposium on Mathematics, Quantum Theory, and Cryptography: Proceedings of MQC 2019*, pages 39–55. Springer Singapore, 2021.

[31] John A Smolin, Graeme Smith, and Alex Vargo. Pretending to factor large numbers on a quantum computer. *arXiv preprint arXiv:1301.7007*, 2013.

[32] Safia Seffah, Salah Eddine Rihane, and Alain Togbé. Fermat or mersenne numbers as products of two k-pell numbers: S. seffah et al. *Indian Journal of Pure and Applied Mathematics*, pages 1–12, 2025.

[33] Harald A Helfgott and David J Platt. Numerical verification of the ternary goldbach conjecture up to 8.875· 1030. *Experimental Mathematics*, 22(4):406–409, 2013.

[34] David M Burton. *Elementary number theory*. Tata McGraw-Hill Education, 2006.

[35] OAS Karamzadeh. 108.20 euler's totient theorem and fermat's little theorem are generalisations of one another! *The Mathematical Gazette*, 108(572):313–316, 2024.

[36] Jens Kruse Andersen and Harvey Dubner. Pseudoprime statistics to 10^{19}. *Experimental Mathematics*, 16(2):209–213, 2007.

[37] Andrew Shallue and Jonathan Webster. Algorithms for carmichael numbers. *arXiv preprint arXiv:2506.09903*, 2025.

[38] William R Alford, Andrew Granville, and Carl Pomerance. There are infinitely many carmichael numbers. *Annals of Mathematics*, pages 703–722, 1994.

[39] Robert Solovay and Volker Strassen. A fast monte-carlo test for primality. *SIAM journal on Computing*, 6(1):84–85, 1977.

[40] Pedro Cosme da Costa Vieira. A step-by-step explanation of the miller–rabin test for large prime numbers. 2026.

[41] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[42] Tsz-Wo Sze. Deterministic primality proving on proth numbers. *arXiv preprint arXiv:0812.2596*, 2008.

[43] José Grau, Antonio Oller-Marcén, and Daniel Sadornil. A primality test for $k\,p^n + 1$ numbers. *Mathematics of Computation*, 84(291):505–512, 2015.

[44] Caldwell Chris K. Largest known Proth prime . `https://googology.fandom.com/wiki/Largest_known_Proth_prime`, 2016. [Online; accessed 11-January-2026].

[45] Bowman Hall. U-bit collapse in arnault composites: Probing the boundary of strong lucas pseudoprimes. *arXiv preprint arXiv:2601.19817*, 2026.

[46] Andrew Shallue and Jonathan Webster. Tabulating carmichael numbers n= p qr with small p. *Research in Number Theory*, 8(4):84, 2022.

[47] John H Jaroma. Equivalence of pepin's and the lucas-lehmer tests. *European Journal of Pure and Applied Mathematics*, 2(3):352–360, 2009.

[48] Alena Šolcová and Michał Křižek. Fermat and mersenne numbers in pepin's test. *Demonstratio Mathematica*, 39(4):737–742, 2006.

[49] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[50] Alberto Bedodi and Francesco Pappalardi. *Primality tests in polynomial time*. PhD thesis, Master's thesis, Universitá Degli Studi Roma TRE, 2010.

[51] Eric W Weisstein. Baillie-psw primality test. 2004.

[52] TR Nicely. The baillie-psw primality test. 2005.

[53] Getaneh Awulachew Zimbele, Baye Yemataw Adane, and Sofonias Yitagesu Techan. Mmsc-rsa: Miller-rabin, montgomery, sliding window, and shamir's chinese remainder theorem optimization-based rsa. 2026.

[54] Aditya Kumar. Improved rsa algorithm based on cloud database using proth number and mersenne prime number. 2020.

[55] Pe Pepin. Sur la formule $2^{2^n} + 1$. *des seances de l'Academie des sciences*, 85:329–331, 1877.

[56] Murat Sahin. Generalized trial division. *Int. J. Contemp. Math. Sciences*, 6(2):59–64, 2011.

[57] JM POLLALRD. A monte oarlo method for factorization. 1975.

[58] Hendrik W Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

[59] Carl Pomerance. The quadratic sieve factoring algorithm. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 169–182. Springer, 1984.

[60] Arjen K Lenstra and Hendrik W Lenstra. *The development of the number field sieve*, volume 1554. Springer Science & Business Media, 1993.

[61] HF Chau and H-K Lo. Primality test via quantum factorization. *International Journal of Modern Physics C*, 8(02):131–138, 1997.

[62] John Brillhart, Derrick H Lehmer, and John L Selfridge. New primality criteria and factorizations of 2ˆ{m}$\pm$1. *Mathematics of computation*, 29(130):620–647, 1975.

[63] Alvaro Donis-Vela and Juan Carlos Garcia-Escartin. A quantum primality test with order finding. *arXiv preprint arXiv:1711.02616*, 2017.

[64] Alberto Carlini and A Hosoya. Quantum probabilistic subroutines and problems in number theory. *Physical Review A*, 62(3):032312, 2000.

[65] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[66] Joseph F Traub. *Algorithms and complexity: new directions and recent results*. Academic Press, Inc., 1976.

[67] Victor F dos Santos and Jonas Maziero. Using quantum computers to identify prime numbers via entanglement dynamics. *Physical Review A*, 110(2):022405, 2024.

[68] Jonathan Welch, Daniel Greenbaum, Sarah Mostame, and Alan Aspuru-Guzik. Efficient quantum circuits for diagonal unitaries without ancillas. *New Journal of Physics*, 16(3):033040, 2014.

[69] Artur K Ekert, Carolina Moura Alves, Daniel KL Oi, Michał Horodecki, Paweł Horodecki, and Leong Chuan Kwek. Direct estimations of linear and nonlinear functionals of a quantum state. *Physical review letters*, 88(21):217901, 2002.

[70] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[71] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[72] Rodney Van Meter and Kohei M Itoh. Fast quantum modular exponentiation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 71(5):052320, 2005.

[73] A. V. Antipov, E. O. Kiktenko, and A. K. Fedorov. Efficient realization of quantum primitives for shor's algorithm using pennylane library. *arXiv preprint arXiv:2201.05426v2*, 2022.

[74] Élie Gouzien and Nicolas Sangouard. Factoring 2048-bit rsa integers in 177 days with 13,436 qubits and a multimode memory. *Physical Review Letters*, 127(14):140503, 2021.

[75] E. Knill. On shor's quantum factor finding algorithm: Increasing the probability of success and tradeoffs involving the fourier transform modulus. Technical Report LAUR-95-2225, Los Alamos National Laboratory, 1995. Performed under the auspices of the U.S. Department of Energy under Contract No. W-7405-ENG-36.

[76] Seyoon Ragavan and Vinod Vaikuntanathan. Space-efficient and noise-robust quantum factoring: S. ragavan, v. vaikuntanathan. *Journal of Cryptology*, 39(2):14, 2026.

[77] Martin Ekerå and Joel Gärtner. Extending regev's factoring algorithm to compute discrete logarithms. In *International Conference on Post-Quantum Cryptography*, pages 211–242. Springer, 2024.

[78] Cédric Pilatte. Unconditional correctness of recent quantum algorithms for factoring and computing discrete logarithms. In *Forum of Mathematics, Pi*, volume 14, page e5. Cambridge University Press, 2026.

[79] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum factoring. In *Annual International Cryptology Conference*, pages 384–415. Springer, 2025.

[80] Martin Ekerå and Joel Gärtner. A high-level comparison of state-of-the-art quantum algorithms for breaking asymmetric cryptography. *IACR Communications in Cryptology*, 2(1), 2025.

[81] Craig Gidney. How to factor 2048 bit rsa integers with less than a million noisy qubits. *arXiv preprint arXiv:2505.15917*, 2025.

[82] Seyoon Ragavan and Vinod Vaikuntanathan. Space-efficient and noise-robust quantum factoring. In *Annual International Cryptology Conference*, pages 107–140. Springer, 2024.

[83] Burton S Kaliski Jr. Targeted fibonacci exponentiation. *arXiv preprint arXiv:1711.02491*, 2017.

[84] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[85] Stephane Beauregard. Circuit for shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095*, 2002.

[86] Austin G Fowler, Simon J Devitt, and Lloyd CL Hollenberg. Implementation of shor's algorithm on a linear nearest neighbour qubit array. *arXiv preprint quant-ph/0402196*, 2004.

[87] S. Parker and M. B. Plenio. Efficient factorization with a single pure qubit and logn mixed qubits. *arXiv preprint quant-ph/0001066v3*, 2005.

[88] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, Th. Lippert, H. Watanabe, and N. Ito. Massive parallel quantum computer simulator. *arXiv preprint arXiv:quant-ph/0608239*, August 30 2006. Available at `https://arxiv.org/abs/quant-ph/0608239`.

[89] Archana Tankasala and Hesameddin Ilatikhameneh. Quantum-kit: simulating shor's factorization of 24-bit number on desktop. *arXiv preprint arXiv:1908.07187*, 2019.

[90] A Yu Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.

[91] YS Nam and R Blümel. Scaling laws for shor's algorithm with a banded quantum fourier transform. *arXiv preprint arXiv:1302.5844*, 2013.

[92] D. S. Wang, C. D. Hill, and L. C. L. Hollenberg. Simulations of shor's algorithm using matrix product states. *arXiv preprint arXiv:1501.07644*, 2015. Centre for Quantum Computation and Communication Technology, University of Melbourne, Parkville, Victoria, Australia.

[93] Ya-Qian Zhao, Ren-Gang Li, Jin-Zhe Jiang, Chen Li, Hong-Zhen Li, En-Dong Wang, Wei-Feng Gong, Xin Zhang, and Zhi-Qiang Wei. Simulation of quantum computing on classical supercomputers. *arXiv preprint arXiv:2010.14962*, 2020.

[94] Martin Ekerå. Qunundrum: Software for simulating quantum order finding and related algorithms. `https://github.com/ekera/qunundrum`, 2022. Accessed: 2023-12-21.

[95] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, and Jeremy L. O'Brien. Experimental realisation of shor's quantum factoring algorithm using qubit recycling. *arXiv preprint arXiv:1111.4147*, 2012.

[96] T. Monz, D. Nigg, E. A. Martinez, M. F. Brandl, P. Schindler, R. Rines, S. X. Wang, I. L. Chuang, and R. Blatt. Realization of a scalable shor algorithm. *arXiv preprint arXiv:1507.08852*, 2015.

[97] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *arXiv preprint quant-ph/0112176*, 2001.

[98] Alberto Politi, Jonathan CF Matthews, and Jeremy L O'Brien. Shor's quantum factoring algorithm on a photonic chip. *arXiv preprint arXiv:0911.1242*, 2009.

[99] B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James, A. Gilchrist, and A. G. White. Experimental demonstration of shor's algorithm with quantum entanglement. *arXiv preprint arXiv:0705.1398*, 2007.

[100] Unathi Skosana and Mark Tame. Demonstration of shor's factoring algorithm for $n = 21$ on ibm quantum processors. *arXiv preprint arXiv:2103.13855*, September 20 2022.

[101] Martin Ekerå. On post-processing in the quantum algorithm for computing short discrete logarithms. *Designs, Codes and Cryptography*, 88:2313–2335, 2020.

[102] Martin Ekerå and Johan Håstad. Quantum algorithms for computing short discrete logarithms and factoring rsa integers. *arXiv preprint arXiv:1702.00249*, 2017.

[103] Richard Jozsa. Quantum factoring, discrete logarithms and the hidden subgroup problem, 2000. Available at `http://arxiv.org/abs/quant-ph/0012084v1`.

[104] Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.

[105] Hao Yu, Alexander O Govorov, Hai-Zhi Song, and Zhiming Wang. Time-encoded photonic quantum states: Generation, processing, and applications. *Applied Physics Reviews*, 11(4), 2024.

[106] Thomas Monz, Daniel Nigg, Esteban A Martinez, Matthias F Brandl, Philipp Schindler, Richard Rines, Shannon X Wang, Isaac L Chuang, and Rainer Blatt. Realization of a scalable shor algorithm. *Science*, 351(6277):1068–1070, 2016.

[107] Amir H Karamlou, William Muschinske, Ray Winik, Terry P Orlando, William D Oliver, and Fei Yan. Analyzing the performance of variational quantum factoring on a superconducting quantum processor. *npj Quantum Information*, 7(1):156, 2021.

[108] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pages 1001–1014, 2019.

[109] Élie Gouzien and Nicolas Sangouard. Factoring 2048-bit rsa integers in 177 days with 13 436 qubits and a multimode memory. *Physical review letters*, 127(14):140503, 2021.

[110] Lalitha Kiran Nemana and Vadlamudi Ch Venkaiah. An empirical study towards refining the aks primality testing algorithm. *IACR Cryptol. ePrint Arch.*, 2016:362, 2016.

[111] Neil James Sloane and David Wilson. Fermat primes: primes of the form $2^{2^k} + 1$, for some $k >= 0$. . `http://oeis.org/A019434`, 2010. [Online; accessed 2-June-2020].

A Experimental Work for Classical Primality Testing Algorithms

A.1 Experimental Setup for Monte-Carlo Tests

For the Monte-Carlo primality tests (Fermat, Miller-Rabin, Solovay-Strassen), six different numbers with different digit sizes are used as inputs for these tests. These numbers are listed in Table 9.

Table 9: Different Generic Numbers.

Value	**#digits**	**Type**
11621	5	Prime
11611	5	Composite
2860486327	10	Prime
2860486317	10	Composite
12764787846358441471	20	Prime
12764787846358441481	20	Composite

Results of Figure 1 show how each algorithm is efficient from the perspective of speed. The results show that Miller is often the fastest. Fermat is always faster than Solovay, but the latter is more accurate. The Miller-Rabin test is always preferable among all of these tests due to its speed and accuracy. The important observation from Figure 1 is that these tests are considered as compositeness tests; thus, the 20-digit composite number takes less time than the 5-digit prime number.

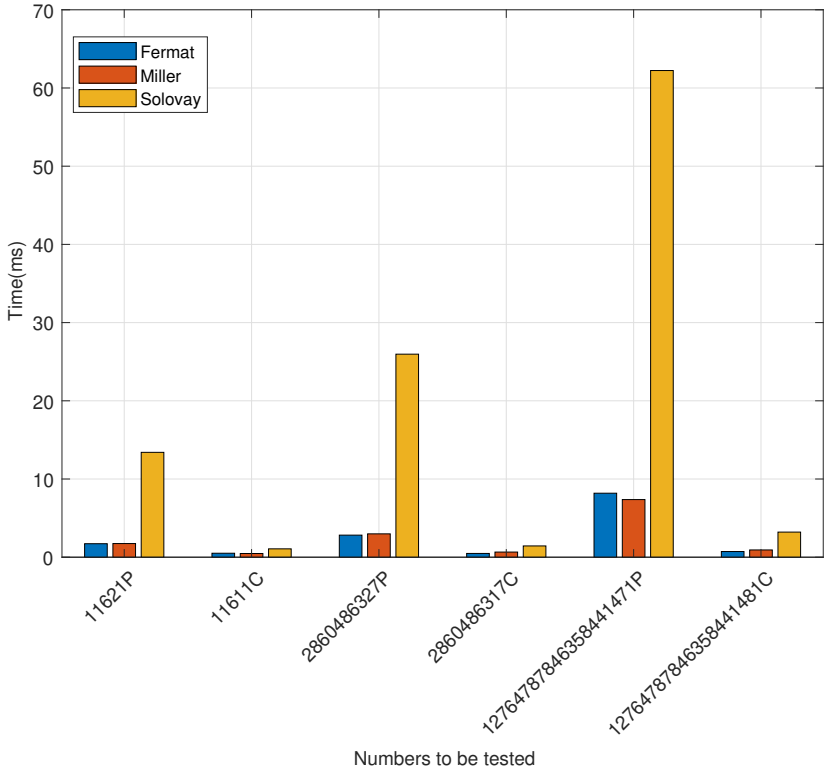


Figure 1: Performance comparison of Monte-Carlo randomized primality tests. The figure illustrates the computational cost of three probabilistic primality testing algorithms when applied to integers of increasing size.

A.2 Experimental Setup for Las-Vegas Test

Proth, Lucas, and Picklington tests are randomized Las-Vegas primality tests, which means that they always give the right answer or no answer. Four different Proth numbers in the form $(x \times 2^n + 1)$ with two different digit sizes are used as inputs for these tests, where x is an integer number. These numbers are listed in Table 10. Figure 2 shows the results of testing these numbers with

Table 10: Different Proth numbers values.

Value	**#digits**	**Type**
18433 $(9 \times 2^{11} + 1)$	5	Prime
45057 $(11 \times 2^{12} + 1)$	5	Composite
2281701377 $(17 \times 2^{27} + 1)$	10	Prime
6710886401 $(25 \times 2^{28} + 1)$	10	Composite

the Las-Vegas primality tests. The results show that the Proth test outperforms other tests. The reason behind that is due to the $n - 1$ factorization needed in both Lucas and Picklington tests. Indeed, this factorization may add a large computational overhead. Lucas and Pocklington could be preferable over Proth when the number to be tested is not in the Proth form, since the Proth test's certainty is approved by Proth numbers only, while Lucas and Picklington tests are able to check any generic number. From Figure 2, it can also be observed that these tests can decide much more quickly the primality of a prime number than the compositeness of a composite number.

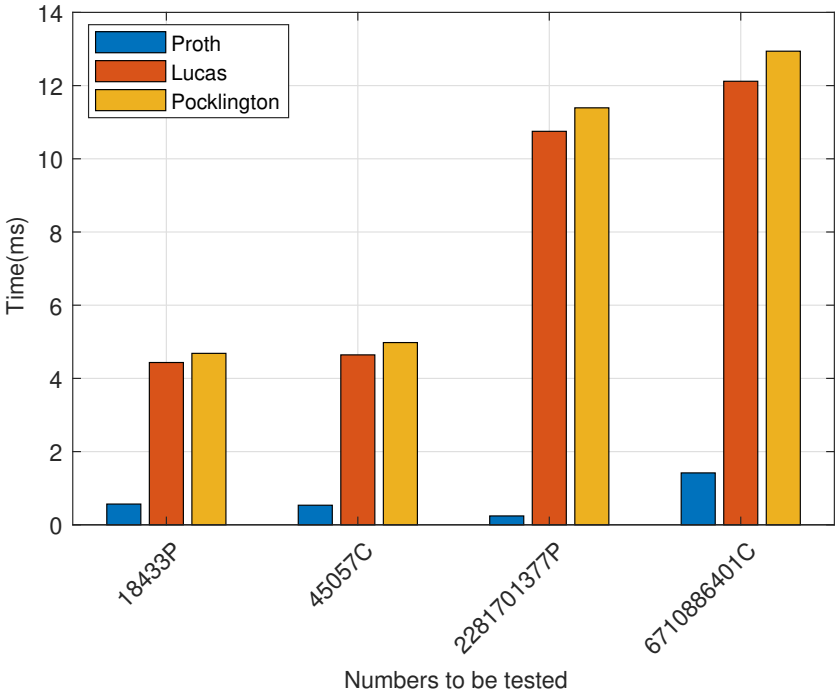


Figure 2: Performance comparison of Las-Vegas randomized primality tests using Proth numbers as inputs. The figure shows how different Las-Vegas primality testing algorithms behave when applied to integers of the form $k \times 2^n + 1$.

Figure 3 shows the results of applying Miller-Rabin and Proth tests on composite Proth numbers. The results show that Miller is much faster in approving the compositeness. Therefore, it is always suggested and advised that when testing a number for primality, a compositeness test be applied first, and if the number passes that test as a probable prime, then a Las-Vegas test could be applied to prove the primality.

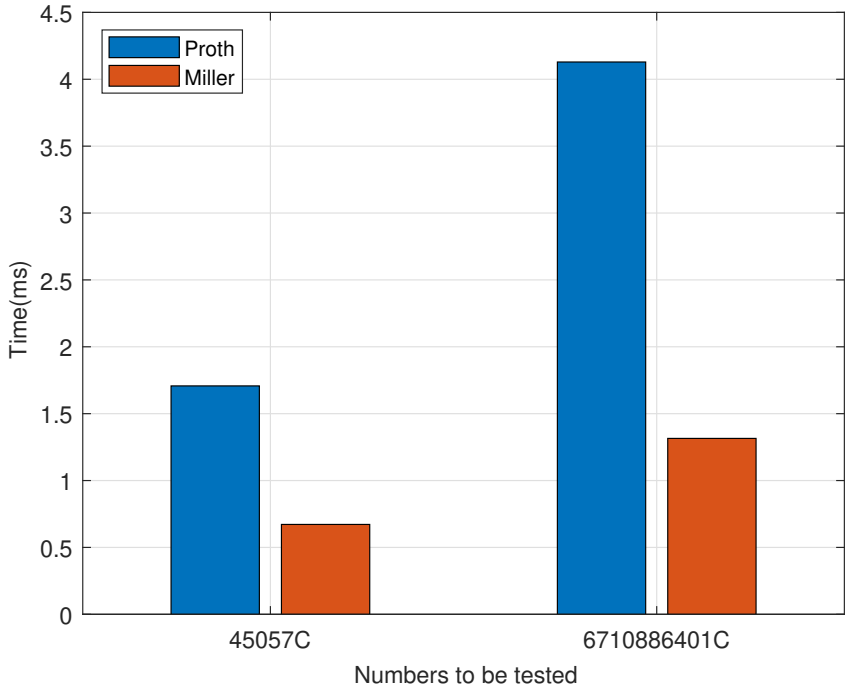


Figure 3: Comparison between Proth and Miller primality tests when applied to composite numbers. The figure evaluates the ability of both algorithms to detect compositeness efficiently across inputs of increasing size.

Although Figure 2 shows that Lucas outperforms Pocklington, this is not always the case. Pocklington is fast and smart when there is a q prime factor of $n-1$, such that $q > \sqrt{(n-1)}$. For example, a number $n = 18439$ has a prime factor $q = 439 > (\sqrt{(n-1)} = 135)$. Figure 4 shows that Pocklington accomplished the approval of primality of the input (18439) much faster than Lucas (about 50% faster).

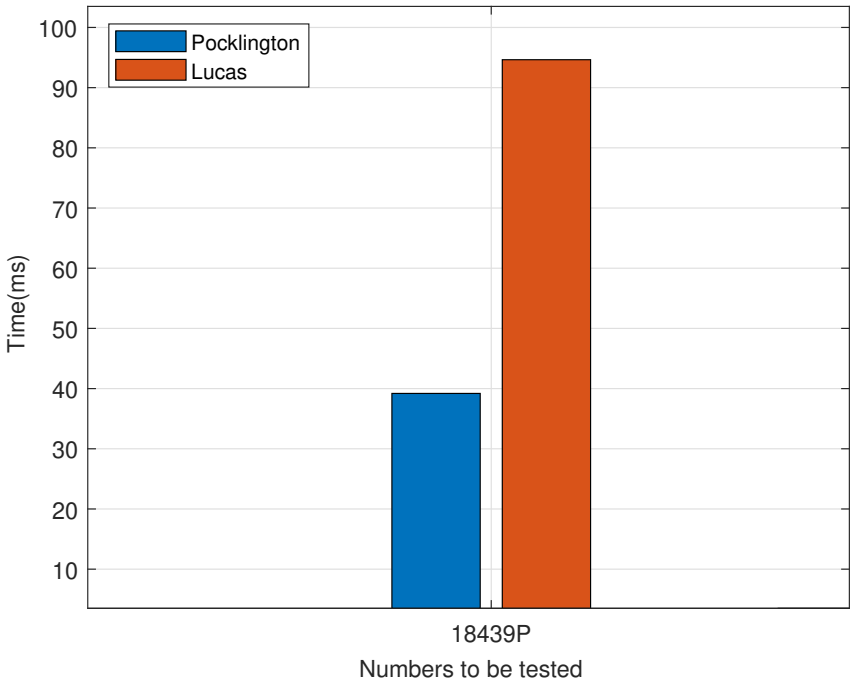


Figure 4: Performance comparison between Lucas and Pocklington primality tests when a prime factor $q > \sqrt{n-1}$ of $n-1$ is known.

A.3 Experimental Setup for Deterministic Tests

For a fair comparison between deterministic tests, a distinction should be made between Pepin and Lucas-Lehmer. While Pepin specialized in the primality detection of Fermat numbers, the Lucas-Lehmer algorithm is designed to detect primality for Mersenne numbers, and hence, two different experiments are conducted to compare these types of primality tests. The Naïve and AKS are general deterministic tests for any format.

Fermat numbers which are listed in Table 11 are used as inputs for a comparison between Pepin, Naïve, and AKS tests [110]. The results depicted in Figure 5 show that Pepin's test is the most efficient for small and moderate Fermat numbers. For very large integers, one might consider using the AKS test due to its polynomial-time worst-case guarantee [110], although we did not benchmark it in this work because of its high practical cost.

Furthermore, if the Pepin test is used to test non-Fermat numbers, it may give false negative results in many cases. For example, Pepin declares 7 and 11 as composite numbers. Also, Naïve can be fast if the input number is composite and has a small factor, which makes the test exit very early. In general, these tests can usually detect a composite number at an early stage.

Table 11: Fermat numbers parameters

Value	**#digits**	**Type**
65537 $(2^{16}+1)$	5	Prime
32769 $(2^{15}+1)$	5	Composite
4294967297 $(2^{32}+1)$	10	Composite
18446744073709551617 $(2^{64}+1)$	20	Composite

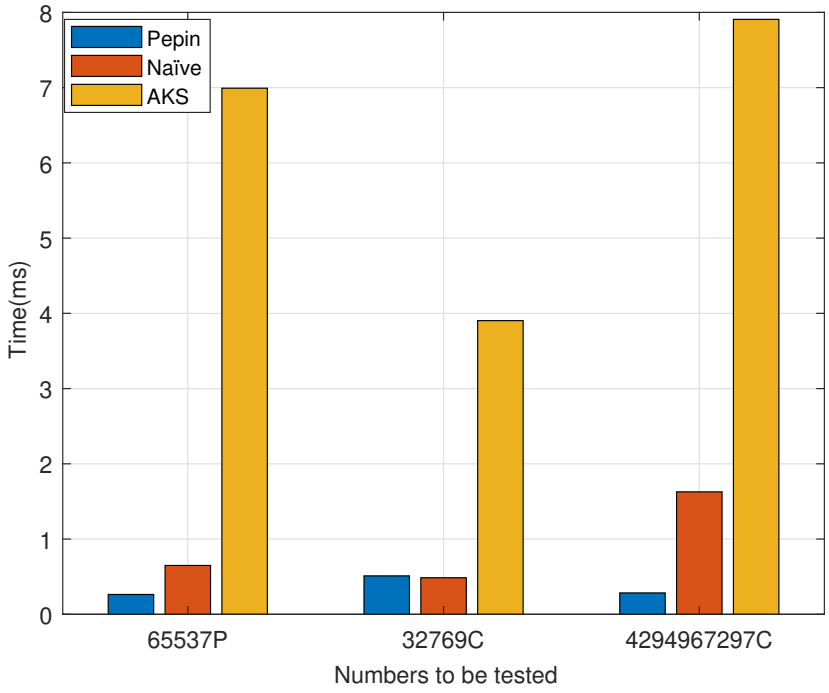


Figure 5: Comparison of deterministic primality tests using Fermat numbers as inputs. The figure evaluates the performance of several deterministic algorithms when applied to integers of the form $F_n = 2^{2^n} + 1$.

Figure 6 shows how the computations are dramatically increased in the naive method when the number of digits of a prime input number is increased. On the other hand, an input of a composite number consisting of 10 or 20 digits takes much less time than an input of a 5-digit prime number. This is because the 10 and 20-digit composite numbers are divided by 3 and 7, respectively, which means that the test exits at a very early stage.

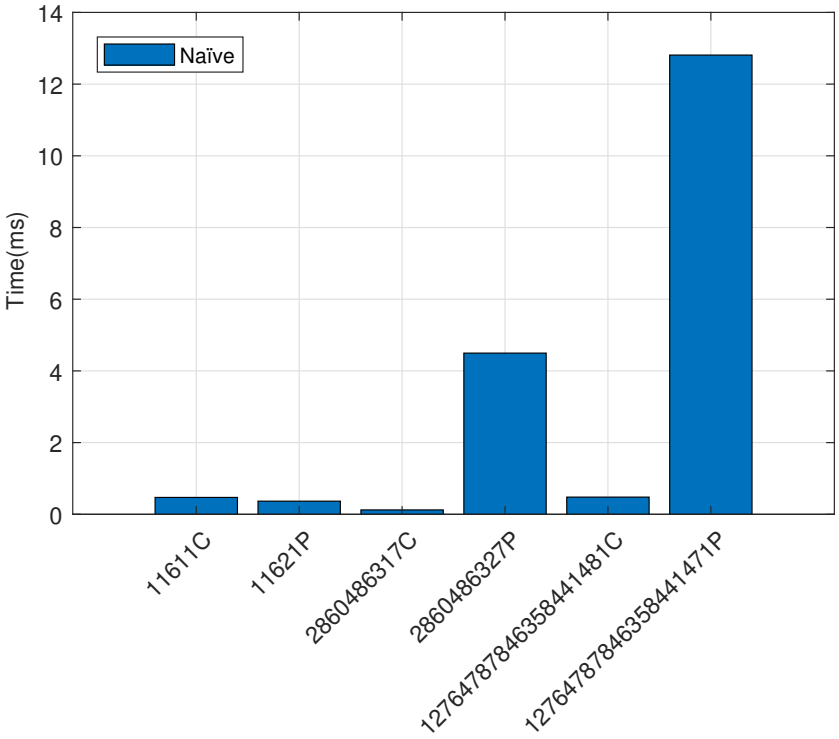


Figure 6: Behavior of the naïve primality testing algorithm for integers of increasing size. The figure illustrates the growth of computational cost when the basic trial-division method is applied to progressively larger numbers.

Since Fermat is a Proth number with $k = 1$, Figure 7 shows the results of comparing Proth with Pepin for testing Fermat numbers. The results show that Proth performs very well when the Fermat number is prime. However, since the largest discovered Fermat prime is 65537 [111], it turns out that the Proth test is not preferable for testing Fermat numbers unless a person wants to discover a new massive Fermat prime, if such a prime were available.

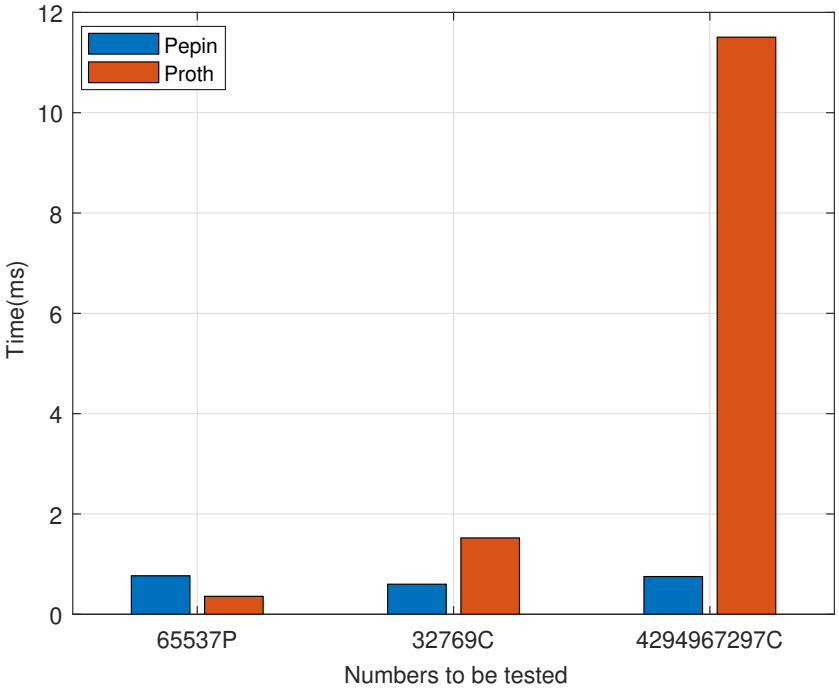


Figure 7: Performance comparison between the Pepin and Proth tests for Fermat numbers. The figure evaluates how each algorithm behaves when applied to integers of the form $F_n = 2^{2^n} + 1$.

Mersenne numbers, which are listed in Table 12, are used as inputs for comparing Lucas-Lehmer, Naïve, and AKS. The results, which are depicted in Figure 8 shows that Lucas-Lehmer outperforms the other two tests. Lucas-Lehmer is widely used in practice to test the primality of very large Mersenne numbers. In general, AKS is very slow in practice, and as the number of digits increases, the computations dramatically increase. AKS even works worse than the Naïve test till a certain point (about 2^{64}) then it beats Naïve.

Table 12: Characteristics of 4 Mersenne numbers.

Value	**#digits**	**Type**
8191 $(2^{13} - 1)$	4	Prime
2047 $(2^{11} - 1)$	4	Composite
2147483647 $(2^{31} - 1)$	10	Prime
137438953471 $(2^{37} - 1)$	12	Composite

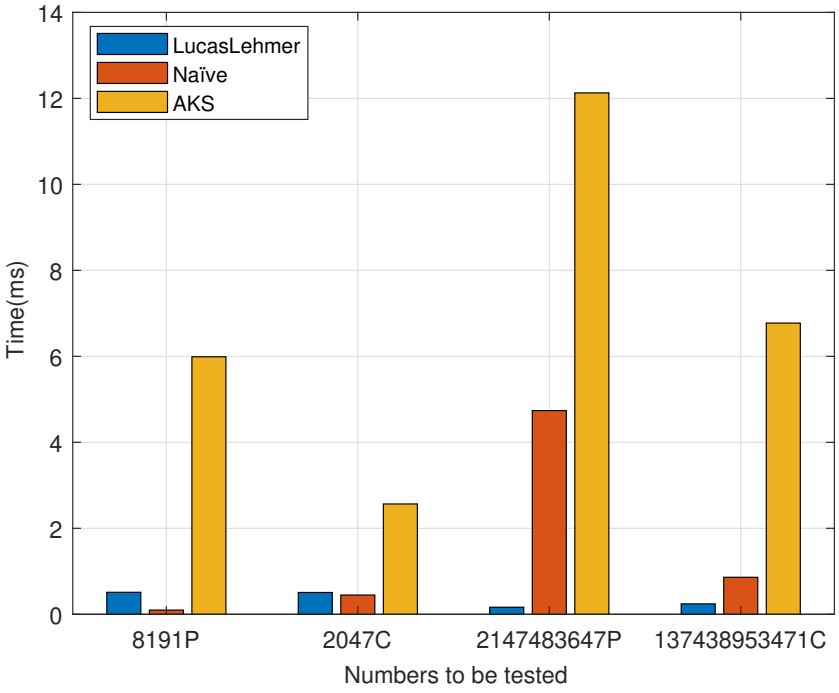


Figure 8: Comparison of three deterministic primality tests using Mersenne numbers as inputs. The figure evaluates the computational performance of the considered algorithms when applied to numbers of the form $M_p = 2^p - 1$.

A.4 Experimental Setup for Heuristic Test

Since Baillie-PSW is a generic test that can be applied to any number format, it could be compared with any primality test, but the most efficient test is selected. For this scenario, the Baillie test is compared with Miller for generic numbers, Proth for Proth numbers, Lucas-Lehmer for Mersenne numbers, and Pepin for Fermat numbers.

Figure 9 shows the results of comparing Baillie with Miller for random prime and composite numbers. The figure shows that Baillie outperforms Miller in all test cases.

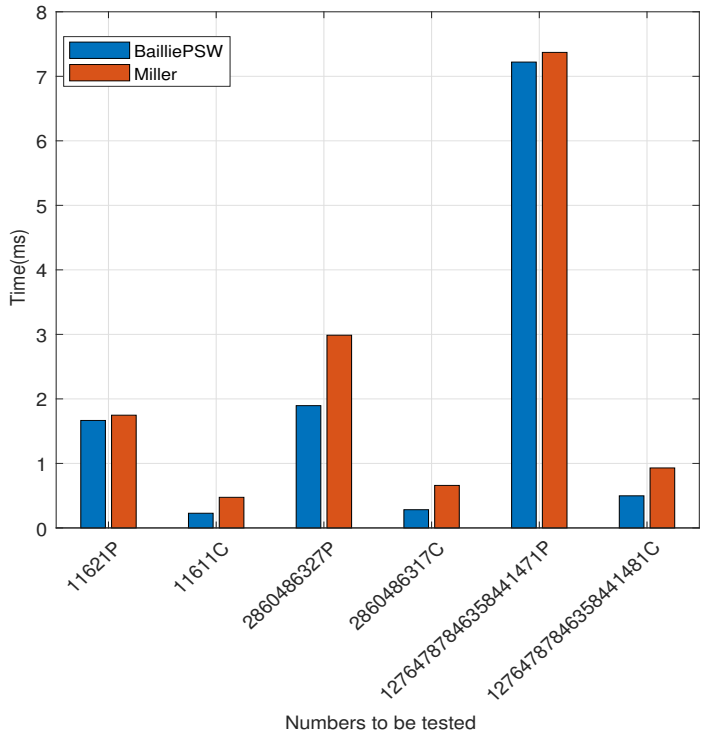


Figure 9: Comparison between the Baillie primality test and the most efficient Monte-Carlo probabilistic test, namely Miller-Rabin. The figure illustrates the computational cost of both methods when applied to integers of increasing size.

Figure 10 shows the results of comparing Baillie with Proth for Proth numbers listed in Table 11. The results show that Baillie outperforms Proth in the case of determining the compositeness of a composite number.

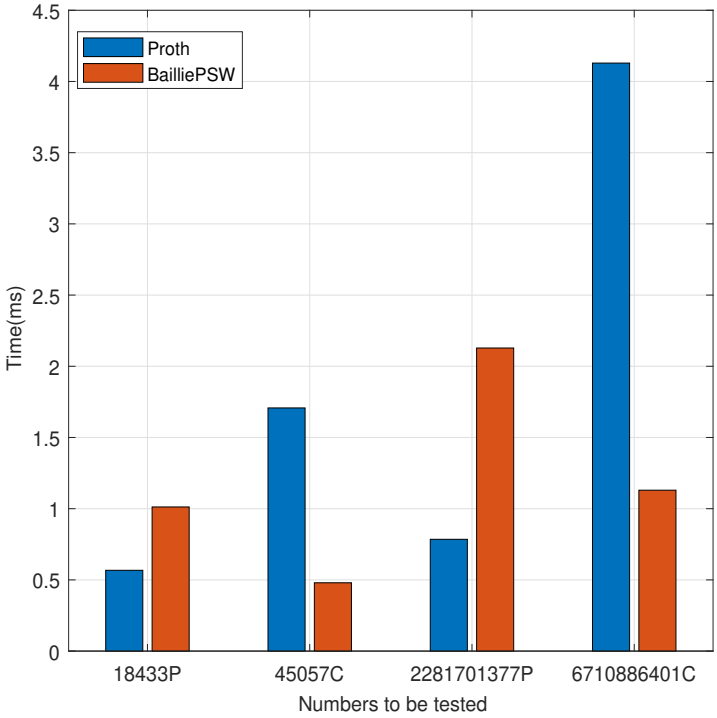


Figure 10: Performance comparison between the Baillie primality test and the most efficient Las-Vegas test, Proth. The figure evaluates the computational effort required by each method to determine primality for numbers of increasing size.

Figure 11 shows the results of comparing Baillie with Pepin for three different Fermat numbers, whereas Figure 12 shows the results of comparing Baillie with Lucas-Lehmer for four Mersenne numbers.

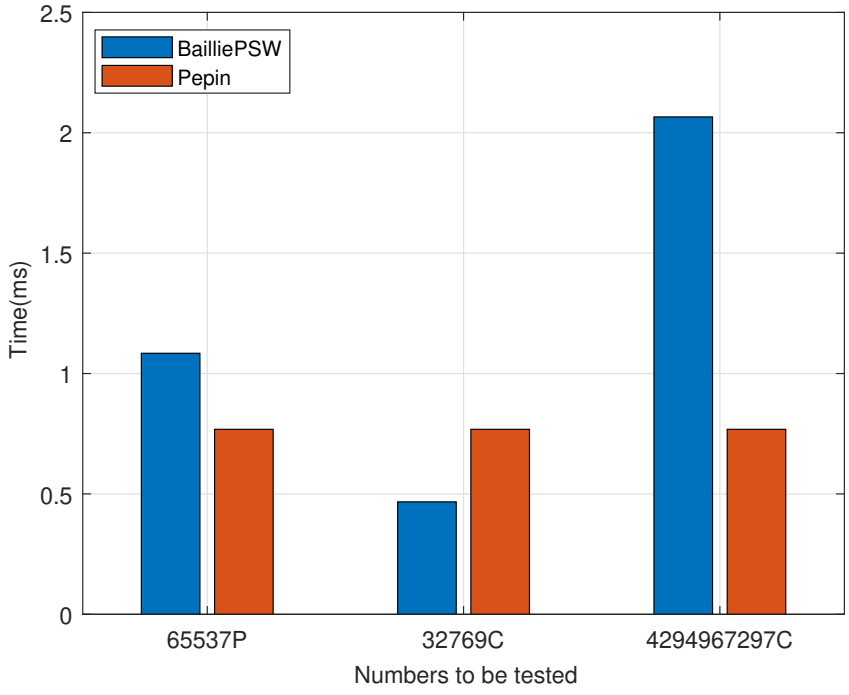


Figure 11: Comparison between the Baillie primality test and the Pepin test for Fermat numbers. The figure evaluates the computational cost of both methods when applied to integers of the form $F_p = 2^{2^p} + 1$.

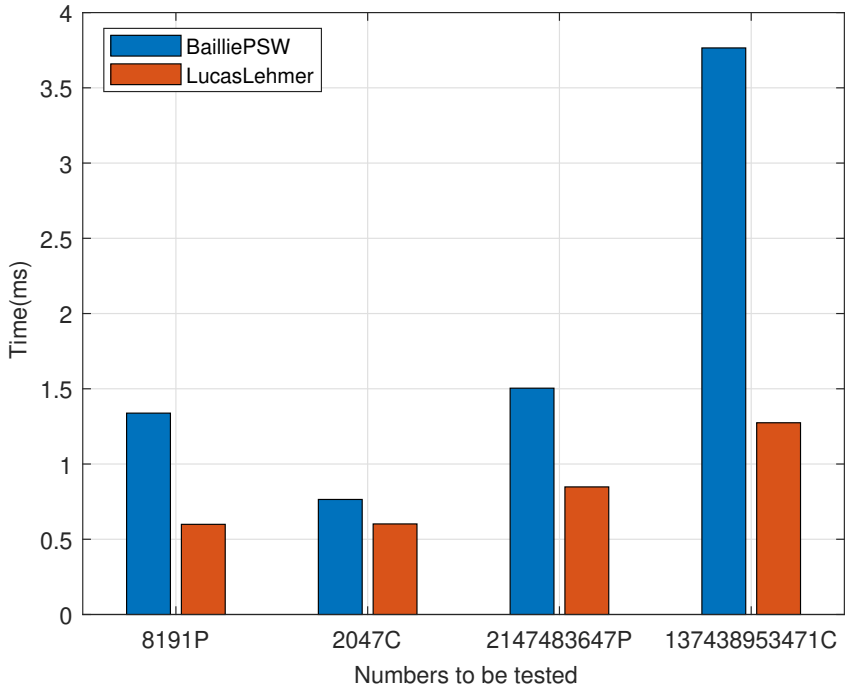


Figure 12: Comparison between the Baillie primality test and the Lucas-Lehmer test for Mersenne numbers. The figure illustrates the performance of both algorithms when applied to integers of the form $M_p = 2^p - 1$.

Overall, these four plots show how Baillie-PSW brings one test to all number types with acceptable speed and without sacrificing correctness.

A.5 Experimental Setup for Large Inputs

In this section, we test large numbers , i.e.,$\geq$ 1204-bit numbers. Table 13 lists two large Mersenne numbers and Figure 13 shows the results of comparing five different primality tests (Fermat, Miller-Rabin,Solovay-Strassen, LucasLehmar, and BailliPSW). Table 14 lists two large Proth numbers and Figure 14 shows the results of comparing five different primality tests (Fermat, Miller-Rabin,Solovay-Strassen, Proth, and BailliPSW).

9.

Table 13: Two big Mersenne numbers.

Number Value	**#-digits**	**Type**
$(2^{1279}-1)$ 104079321946643990819252403273640855386152622472667048053191123504036080596733602980122394417323241848424216139542810077913835662483234649081399066056773207629241295093892203457731833496615835504729594205476898112116936771475484788669625013844382602917323488853111608285384165850282556046662248318909188018470682222031405210266984354887329580288780508697361869007147207105557031687290871040793219466439908192524032736408553861526224726670480531911235040360805967336029801223944173232418484242161395428100779138356624832346490813990660567732076292412950938922034577318334966158355047295942054768981121169367714754847886696250138443826029173234888531116082853841658502825560466622483189091880184706822220314052102669843548873295802878050869736186900714720710555703168729087.	386 (1279-bit)	Prime
$(2^{1278}-1)$ 520396609733219954096262016368204276930763112363335240265955617520180402983668014900611972086616209242121080697714050389569178312416173245406995330283866038146206475469461017288659167483079177523647971027384490560584683857377423943348125069221913014586617444265558041426920829251412780233311241594545940092353411110157026051334921774436647901443902543486809345035736035527785158436454 3.	385 (1278-bit)	Composite

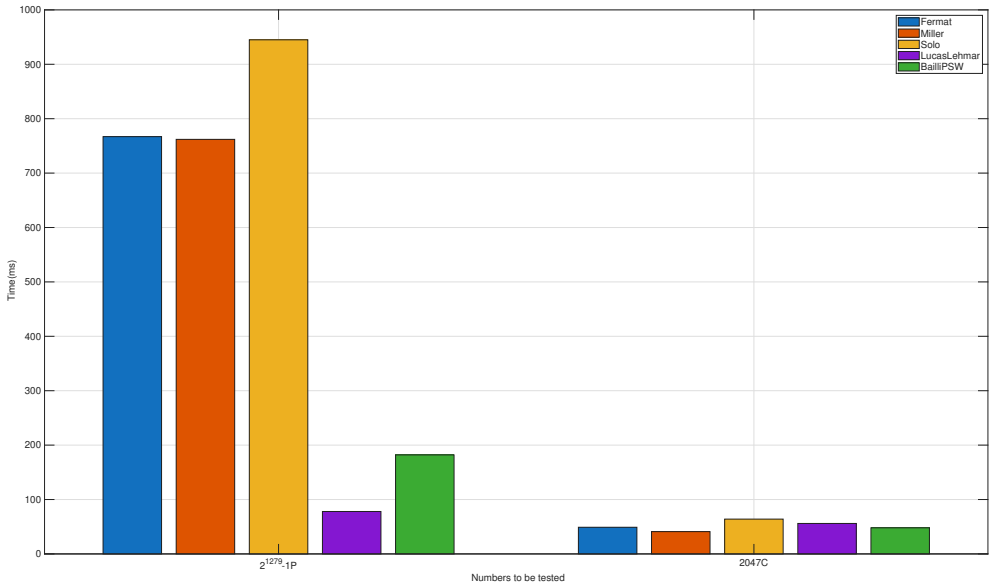


Figure 13: Comparison of 5 primality testing algorithms (Two large Mersenne numbers as inputs).

Table 14: Two big Proth numbers.

Number Value	#-digits	Type
$((9 \times 2^{1305} + 1))$ 628618055555659215267581230166752705405768544228540098057476108591560470547331078596072562327585421312050840169366511732689410107274356671076220136835790852267106484607589120125472982633553290861994272487948348949312103570419264921520874538657271033683019873170408460752060725642005077377705102873111479437378065110297103838622861108603538251409916669561986734338781125466399443537148477 4924289.	394 (1309-bit)	Prime
$(9 \times 2^{1303} - 1)$ 157154513888914803816895307541688176351442136057135024514369027147890117636832769649018140581896355328012710042341627933172352526818589167769055034208947713066776621151897280031368245658388322715498568121987087237328025892604816230380218634664317758420754968292602115188015181410501269344426275718277869859344516277574275959655715277150884562852479167390496683584695281366599860884287119 3731073.	385 (1309-bit)	Composite

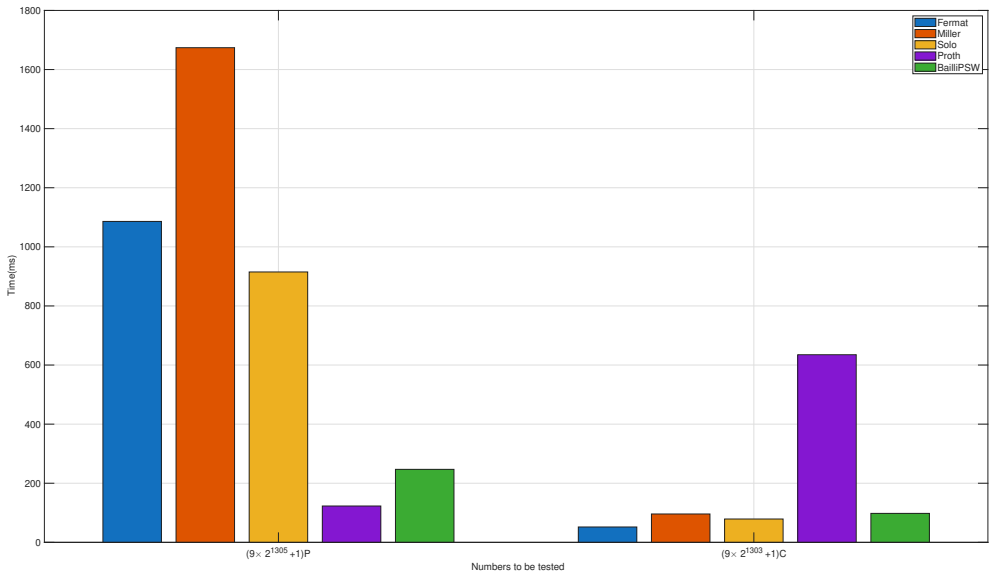


Figure 14: Comparison of 5 primality testing algorithms (Two large Proth numbers as inputs).