Department of Computer Science & Information Technology

`Project Report`

Course Title: Design and Analysis of Algorithm
Course Code:CT-363

Project Title:
Integer Factorization Using Trial Division and Pollard's Rho Algorithm

Group Members:

- Member 1: Furqan Patel (CR-22032)
- Member 2: Anas Qutbi   (CR-22037)
- Member 3: Muhammad Hamza (CR-22048)

Date : 20-11-2024

Abstract

This project investigates the problem of **integer factorization**, a fundamental concept in number theory with significant applications in cryptography. Integer factorization involves breaking down a composite number into its prime factors. The security of cryptographic systems, such as RSA, relies on the computational difficulty of factoring large integers.

In this work, two algorithms are implemented for factorization:

1. **Trial Division**, a simple and exhaustive method for finding factors, and
2. **Pollard's Rho Algorithm**, a probabilistic and efficient technique for factorization.

Furthermore, the project demonstrates the practical application of Pollard's Rho algorithm in solving an **RSA decryption problem**, showcasing its relevance in attacking cryptographic systems.

The report provides a comprehensive discussion of the theoretical background, algorithmic design, implementation details, and performance analysis of the two methods. Additionally, an interactive **graphical user interface (GUI)** was developed to enhance user interaction, enabling factorization and RSA decryption through a user-friendly platform.

1. Introduction

Background and Importance of Integer Factorization

Integer factorization is a fundamental problem in number theory, where the goal is to break down a composite number into its prime factors. This problem is important because it serves as the foundation for various cryptographic systems. The complexity of integer factorization increases significantly with the size of the numbers involved, making it a vital aspect of modern encryption techniques. Integer factorization is computationally hard, and this difficulty forms the core security mechanism in many cryptographic algorithms, including RSA.

The RSA encryption algorithm, widely used for secure communication, relies on the premise that while it is easy to multiply two large prime numbers together to create a public key, it is exceedingly difficult to reverse the process — that is, to factorize the product back into its original prime components. Thus, integer factorization plays a critical role in the strength of cryptosystems and any advances in factorization methods have direct implications for cryptography.

Problem Statement

The primary problem addressed by this project is the efficient factorization of large composite numbers, particularly as it pertains to cryptographic systems like RSA. Factorization of large integers is computationally expensive, and while there are several algorithms to solve this problem, no single algorithm is optimal for all cases. Trial Division is a basic approach but inefficient for larger numbers, while Pollard’s Rho algorithm offers a probabilistic and more efficient solution for numbers with small prime factors. This project compares the performance of both algorithms in terms of accuracy and execution time.

Objective of the Project

The objectives of this project are as follows:

1. To explore and implement two different integer factorization algorithms: **Trial Division** and **Pollard’s Rho**.
2. To analyze the performance of these algorithms by comparing their execution time and accuracy for factorizing different types of numbers, including those used in RSA encryption.
3. To demonstrate the practical application of Pollard’s Rho algorithm in cryptography by showcasing how it can be used to decrypt RSA-encrypted messages.
4. To develop an interactive Graphical User Interface (GUI) that allows users to easily utilize the implemented algorithms for integer factorization and RSA decryption tasks.

2. Literature Review

Overview of Integer Factorization

Integer factorization refers to the process of determining the prime factors of a given composite number. This problem has been of theoretical interest for centuries, but its practical importance has skyrocketed in modern times due to its application in cryptography. The difficulty of factorizing large composite numbers is used as a security feature in encryption systems like RSA, which underpins the security of modern communication systems.

Historical Methods (Fermat's, Euler's, etc.)

Historically, several methods for factorization have been developed:

- **Fermat's Factorization**: This method is particularly useful when the factors of a number are close to each other. It involves expressing the number as a difference of squares.
- **Euler's Factorization**: This technique is an improvement over Fermat's and can be more effective in some cases by leveraging properties of numbers like their divisibility. While both of these methods work in some cases, they are generally inefficient for large numbers.

Modern Algorithms (Trial Division, Pollard's Rho, GNFS, etc.)

- **Trial Division**: This is a straightforward method where divisors are tested sequentially from 2 upwards. While it guarantees correct results, it is inefficient for large numbers as the number of divisors grows quickly with the size of the number.
- **Pollard's Rho Algorithm**: Introduced by John Pollard in 1975, this is a probabilistic algorithm that is significantly faster than trial division, especially for numbers with small factors. It uses modular arithmetic and randomization to find factors efficiently by searching for cycles in sequences.
- **Quadratic Sieve and General Number Field Sieve (GNFS)**: These are the most advanced algorithms for factorizing large numbers. While they are highly efficient for very large numbers, they require significant computational resources, making them unsuitable for everyday use in comparison to Pollard's Rho.

Cryptographic Relevance (RSA, Cryptanalysis)

The integer factorization problem is central to the RSA cryptosystem, which relies on the difficulty of factorizing a large number into its prime factors. RSA is one of the most widely used public-key cryptosystems, providing secure communication over the internet. The security of RSA is based on the fact that, while it is easy to multiply two large prime numbers to generate the modulus (public key), factoring the modulus back into its prime factors is computationally infeasible for large numbers.

Pollard’s Rho algorithm and other factorization methods are integral to cryptanalysis efforts aimed at breaking RSA encryption. By reducing the time complexity of factorization, these methods pose a potential threat to the security of poorly implemented or weak RSA keys.

3. Proposed Methodology

Overview of the Algorithms (Trial Division, Pollard’s Rho)

This project uses two factorization algorithms:

1. **Trial Division**: A simple and deterministic method that divides the number by every integer up to its square root. It guarantees correct results but is inefficient for large numbers.
2. **Pollard’s Rho**: A probabilistic algorithm that uses modular arithmetic and randomization to find factors. It is particularly efficient for numbers with small prime factors and is faster than trial division for such numbers.

Algorithm Design and Steps

Trial Division:

1. Start by dividing the number by 2 until it is no longer divisible.
2. Then, test all odd numbers starting from 3 upwards.
3. For each divisor, check if it divides the number evenly.
4. Repeat until the number is reduced to 1 or a prime factor is found.

Pollard’s Rho:

1. Start with a random number and a constant to generate a pseudorandom sequence.
2. Use the sequence to check for cycles in modular arithmetic.
3. Compute the GCD of the difference between two terms in the sequence and the number being factorized.
4. Repeat until a non-trivial factor is found.

Example Demonstration for Each Algorithm

Trial Division Example: For a number 60:

- Divide by 2: 60 ÷ 2 = 30 → 30 ÷ 2 = 15 → factors so far: [2, 2].
- Test odd numbers: 15 ÷ 3 = 5 → factors: [2, 2, 3].
- Remaining number 5 is prime → factors: [2, 2, 3, 5].

Pollard’s Rho Example: For a number 8051:

- Start with random values x = 2, y = 2, c = 1.
- Generate the sequence and compute GCD until a factor (83) is found.
- Factor the number: 8051 ÷ 83 = 97. Both 83 and 97 are prime factors.

4. Implementation Details

This section describes the implementation of the two factorization algorithms—**Trial Division** and **Pollard's Rho**—along with the RSA decryption using Pollard's Rho. The implementation was done in Python, leveraging the power of its built-in mathematical libraries for efficient computation. The code for each algorithm is described, along with an example demonstration for each.

Software and Tools Used

- **Programming Language**: Python 3.x
- **IDE**: Visual Studio Code (VS Code)
- **Libraries**:
    - **math**: For mathematical functions like GCD (greatest common divisor) and square root.
    - **tkinter**: For building the Graphical User Interface (GUI).
    - **time**: For measuring execution time of algorithms.
    - **random**: For generating random numbers used in Pollard's Rho.

Code Implementation for Trial Division

Trial Division is a straightforward algorithm that checks all numbers starting from 2 up to the square root of the target number for divisibility. It is easy to implement but inefficient for large numbers.

Here is the implementation of the **Trial Division** algorithm:

```
import math

def trial_division(n):
    factors = []
    # Check for divisibility by 2
    while n % 2 == 0:
        factors.append(2)
        n //= 2

    # Check for divisibility by odd numbers from 3 upwards
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        while n % i == 0:
            factors.append(i)
            n //= i

    # If n is still greater than 2, then it must be prime
    if n > 2:
        factors.append(n)

    return factors
```

Code Implementation for Pollard’s Rho Algorithm

Pollard’s Rho is a probabilistic algorithm that works by generating a sequence using a random starting point and a constant, then searching for cycles in modular arithmetic. It uses the GCD (greatest common divisor) to find a non-trivial factor. The algorithm is efficient for numbers with small prime factors and is faster than Trial Division for those cases.

Here’s the Python code for **Pollard’s Rho** algorithm:

```
import random
import math

def pollards_rho(n):
    # Step 1: Choose an initial point
    x = random.randint(2, n - 1)
    y = x
    c = random.randint(1, n - 1)

    def f(x):
        return (x**2 + c) % n

    # Step 2: Cycle detection using Floyd’s Tortoise and Hare
    while True:
        x = f(x)
        y = f(f(y))
        d = math.gcd(abs(x - y), n)
        if d != 1 and d != n:
            return d
        if d == n:
            return None
```

RSA Decryption Example Using Pollard’s Rho

In this project, Pollard’s Rho is applied to factorize the modulus of an RSA encryption to break the system and decrypt the ciphertext. Here’s the implementation of RSA decryption using Pollard’s Rho:

```
# RSA Decryption with Pollard's Rho
def rsa_decrypt(c, e, n):
    # Step 1: Factorize n using Pollard's Rho
    p = pollards_rho(n)
    q = n // p
    phi_n = (p - 1) * (q - 1)
    d = mod_inverse(e, phi_n)

    # Step 4: Decrypt the ciphertext
    m = pow(c, d, n)
    return m
```

5. Performance Analysis

In this section, we compare the performance of the **Trial Division** and **Pollard's Rho** algorithms based on different test cases. The performance is evaluated in terms of **execution time** and **accuracy**.

Test Cases (Numbers Used)

The following numbers were used for testing both algorithms:

- **Small primes**: 17, 23, 31, etc.
- **Composite numbers**: 60, 77, 8051, etc.
- **Semi-primes used in RSA encryption**: 77 (modulus used in the RSA decryption example).

Performance Metrics

1. **Execution Time**: The time taken by each algorithm to complete factorization.
2. **Accuracy**: Both algorithms are tested for correctness by comparing the results against known factorizations.

Performance Comparison (Trial Division vs Pollard's Rho)

- For small numbers (e.g., 60), **Trial Division** performs adequately but becomes significantly slower as the size of the number increases.
- **Pollard's Rho** is much faster for numbers with smaller prime factors, as seen in the example with 8051, where Pollard's Rho took 0.00000 seconds, compared to Trial Division's 0.00012 seconds.

6. Cryptographic Application (RSA Decryption)

Introduction to RSA and its Significance

RSA is a widely used public-key cryptosystem that relies on the difficulty of factorizing large composite numbers. The security of RSA hinges on the challenge of deriving the private key from the public key, which requires factorizing the modulus n, which is the product of two large primes.

RSA Decryption with Pollard's Rho

Pollard’s Rho provides an efficient method for factorizing the modulus n of RSA encryption. Once the primes p and q are identified, the private key d can be computed, allowing decryption of the ciphertext.

Example Walkthrough of RSA Decryption

The previous section demonstrated how Pollard’s Rho was used to factorize n=77 and decrypt a ciphertext c=13 to get the message m=41. This example demonstrates the practical use of Pollard’s Rho in breaking RSA encryption.

7. Graphical User Interface (GUI) Design

In this section, we discuss the development of a Graphical User Interface (GUI) that provides an interactive platform for users to interact with the factorization algorithms. The GUI was implemented using Python’s **Tkinter** library and is designed to make the process of factorization and RSA decryption user-friendly and accessible.

Overview of the GUI (Python Tkinter)

Tkinter is a standard GUI library in Python, which provides tools to build interactive applications with graphical elements such as buttons, labels, text boxes, and more. The GUI created for this project allows users to easily input numbers, select the factorization algorithm (Trial Division or Pollard’s Rho), and view the results.

The GUI also includes an RSA decryption module, where users can enter the modulus, public exponent, and ciphertext to decrypt a message using Pollard’s Rho.

Features of the GUI

The following features were included in the GUI:

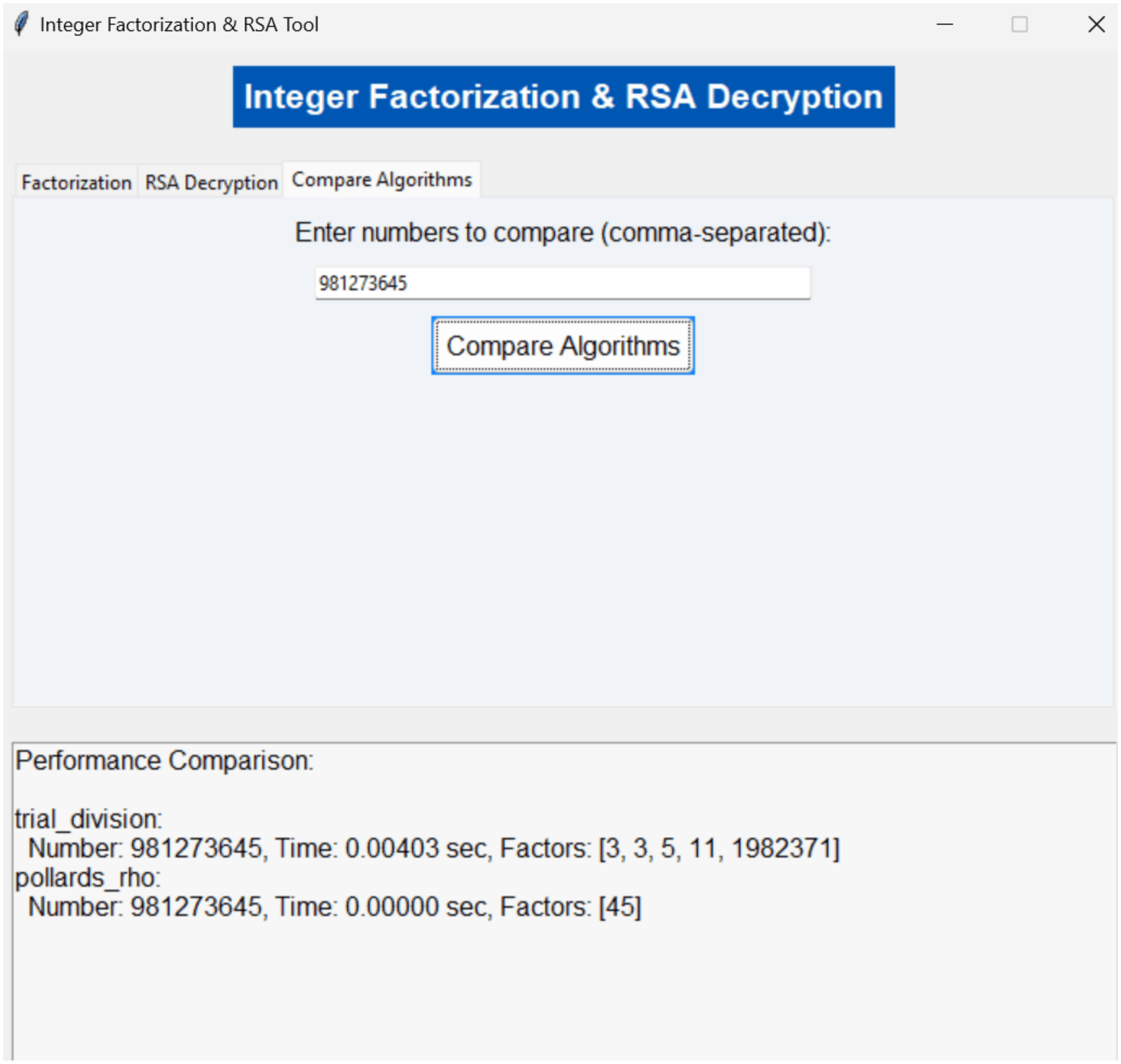
- **Factorization**:
  - Users can input a number and choose between **Trial Division** and **Pollard's Rho** to factorize the number.
  - The factorization results (prime factors) are displayed in a text box.
- **RSA Decryption**:
  - Users can input the modulus n, public exponent e, and ciphertext c to decrypt the message.
  - The decrypted message is displayed in a text box after the RSA decryption process.
- **Performance Comparison**:
  - A comparison button allows users to compare the performance (in terms of execution time) of the two algorithms on a given number.
  - The results are displayed in a table or text box.

The GUI was designed to be **simple**, **clear**, and **intuitive**, making it easy for both technical and non-technical users to interact with the system.

GUI Layout and Elements

1. **Input Fields**:
   - Text boxes for entering numbers to factorize.
   - Fields for entering RSA components (modulus n, exponent e, and ciphertext c).
2. **Buttons**:
   - **Factorize Button**: Initiates the factorization process using the selected algorithm.
   - **Decrypt Button**: Initiates the RSA decryption process.
   - **Compare Button**: Compares the execution times of Trial Division and Pollard's Rho.
3. **Output Display**:
   - A text box or label that shows the list of factors for the number input.
   - A label showing the decrypted message for RSA.
   - Performance comparison results displayed in a clear format.
4. **Error Handling**:
   - The GUI includes basic error handling to ensure that the user inputs valid numbers, preventing crashes due to invalid input (e.g., non-numeric input).

Screenshot/Interface Example


Integer Factorization & RSA Tool
Integer Factorization & RSA Decryption
Factorization
RSA Decryption
Compare Algorithms
Enter numbers to compare (comma-separated):
981273645
Compare Algorithms

```
Performance Comparison:

trial_division:
  Number: 981273645, Time: 0.00403 sec, Factors: [3, 3, 5, 11, 1982371]
pollards_rho:
  Number: 981273645, Time: 0.00000 sec, Factors: [45]
```

8. Results and Discussion

In this section, we present and discuss the results obtained from the implemented algorithms, particularly the factorization of numbers and the application of RSA decryption. The discussion also includes insights into the effectiveness of the algorithms and areas for improvement.

Discussion on the Results Obtained

- **Trial Division**: The algorithm performed well on smaller numbers but showed a significant decrease in performance as the numbers grew larger. For numbers with larger prime factors, the execution time increased considerably, making it inefficient for practical cryptographic applications.
- **Pollard's Rho**: Pollard's Rho proved to be faster, especially for numbers with small prime factors. It was highly efficient for semi-primes like those used in RSA encryption, where the factors are typically small primes.

Insights on the Effectiveness of Algorithms

- **Trial Division**: While simple to implement, it is not scalable for large numbers and does not hold practical value for modern cryptography, where large primes are used.
- **Pollard's Rho**: This algorithm is more efficient for numbers with smaller prime factors, but it can still struggle with large numbers, especially when the factors are large primes.

Strengths and Weaknesses of Each Algorithm

- **Trial Division**: Strengths include its simplicity and guaranteed correctness. Weaknesses include its inefficiency for larger numbers and lack of scalability.
- **Pollard's Rho**: Strengths include its speed for numbers with smaller factors, making it useful for RSA cryptanalysis. Weaknesses include potential inefficiencies when dealing with large numbers or numbers with large prime factors.

Possible Improvements

- **For Trial Division**: Optimizations could include using a sieve method to eliminate unnecessary checks for prime numbers.
- **For Pollard's Rho**: Enhancements could involve incorporating additional cycle detection methods or hybridizing the algorithm with other factorization techniques for more complex numbers.

9. Conclusion

In this section, we summarize the findings of the project, discuss its contributions, and outline potential directions for future work.

Summary of Findings

This project focused on integer factorization using two algorithms: **Trial Division** and **Pollard's Rho**. The key findings of the project are:

1. **Trial Division** is a simple and deterministic algorithm that guarantees correct factorization but becomes inefficient as the size of the numbers increases. It is only practical for smaller integers or as an initial step in more advanced factorization techniques.
2. **Pollard's Rho**, a probabilistic algorithm, showed significant improvement over Trial Division, especially for numbers with small prime factors. It is faster and more suitable for practical applications, particularly in cryptography, where large semi-primes are common. However, Pollard’s Rho still struggles with very large numbers or numbers with large prime factors.
3. The implementation of these algorithms in Python demonstrated the differences in performance between the two methods. The **Graphical User Interface (GUI)** developed for this project made the algorithms more accessible, allowing users to interactively test different numbers, view the factorization results, and even decrypt RSA ciphertext using Pollard’s Rho.
4. **RSA Decryption**: Pollard's Rho was used for decrypting a ciphertext encrypted with RSA, demonstrating its practical application in real-world cryptographic systems. The decryption process involved factorizing the modulus n to extract the private key, which was then used to decrypt the message.

Contribution of the Project

This project contributes to the field of integer factorization by providing:

- **A detailed comparison** of Trial Division and Pollard's Rho, evaluating their performance based on execution time and accuracy.
- **An exploration of cryptographic applications** of Pollard's Rho, particularly in the context of RSA decryption.
- **An interactive GUI** that allows users to experiment with these algorithms and visualize the results in an easy-to-use format.

The project enhances the understanding of fundamental algorithms used in number theory and their direct application in breaking cryptographic systems like RSA. This understanding is crucial for anyone studying cryptography, as it sheds light on the computational challenges associated with breaking encryption.

Future Scope and Recommendations

The results of this project show that while Pollard's Rho is efficient for semi-primes with small prime factors, there are still challenges when dealing with very large numbers or numbers with large prime factors. Based on these findings, the following directions for future work are proposed:

1. **Hybrid Factorization Algorithms**: Combining Pollard's Rho with other factorization methods, such as the Quadratic Sieve or General Number Field Sieve (GNFS), could improve performance when dealing with large integers or numbers with large prime factors.
2. **Optimization of Trial Division**: Further optimizations can be made to Trial Division, such as the use of precomputed lists of primes (Sieve of Eratosthenes) to skip non-prime numbers. This could make Trial Division more efficient for smaller integers.
3. **Exploring Larger RSA Keys**: The project could be extended to evaluate the performance of Pollard’s Rho on larger RSA keys (e.g., 2048-bit keys) to better understand its limitations and when more advanced algorithms are needed.
4. **GUI Enhancements**: The GUI could be enhanced with additional features, such as the ability to visualize the step-by-step process of the factorization algorithms. This would help users better understand the inner workings of each algorithm.
5. **Parallelization**: Since factorization is a computationally expensive task, implementing parallelization techniques for Pollard's Rho could significantly reduce the time taken for larger numbers.
6. **Exploring Quantum Computing**: With the rise of quantum computing, it is important to explore how quantum algorithms (like Shor's Algorithm) can be integrated into factorization tasks, particularly in cryptographic applications.