VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED ALGORITHMS (CO5127)

Miller-Rabin Primarily Test

Advisors: Nguyen An Khuong

Group 02: Nguyen Minh Anh - 2370644
Pham Nhat Minh - 2370503
Phan Van Sy - 2370648
Le Tien Loc - 2370646
Hoang Nhat Quang - 2370696

HO CHI MINH CITY, MAR 2024

Contents

1 Member list & Workload

No.	Fullname	Student ID	%
1	Nguyen Minh Anh	2370644	20%
2	Pham Nhat Minh	2370503	20%
3	Phan Van Sy	2370648	20%
4	Le Tien Loc	2370646	20%
5	Hoang Nhat Quang	2370696	20%

The meeting minutes of our group is 180 minutes.

Meeting 01

- Leader: Pham Nhat Minh
- Date 31/01/2024
- Attendance: All members

No.	Fullname	Task
1	Le Tien Loc	Task 1: 1.1, 2.1, 2.2
2	Pham Nhat Minh	Task 2: 1.2, 2.2, 2.3 Task 6: 4.5, 4.6
3	Hoang Nhat Quang	Task 3: 3.1, 3.2, 3.3
4	Nguyen Minh Anh	Task 4: 3.4, 3.5, 4.4
5	Phan Van Sy	Task 5: 4.1, 4.2, 4.3

Meeting 02

- Leader: Pham Nhat Minh
- Date 09/03/2024
- Attendance: All members

No.	Fullname	Task
3	Hoang Nhat Quang	Task 1: 5.1, 5.2
4	Nguyen Minh Anh	Task 2: 5.2, 5.3
1	Phan Van Sy	Task 3: 5.4, 5.5
2	Pham Nhat Minh	Task 4: 5.6, 5.7 Task 6: 6.4, 6.5, 6.6, 6.7
5	Le Tien Loc	Task 5: 6.1, 6.2, 6.3

Meeting 03

- Leader: Pham Nhat Minh
- Date 27/03/2024
- Attendance: All members

No.	Fullname	Task
1	Le Tien Loc	Pressentation
2	Pham Nhat Minh	Pressentation
3	Hoang Nhat Quang	Pressentation
4	Nguyen Minh Anh	Pressentation
5	Phan Van Sy	Pressentation

The video of our report can be found in https://youtu.be/sgXD0bqw0bQ

2 Introduction

2.1 Foundations of the Miller-Rabin Algorithm: Purpose, Historical Context, and Problem Statement

Prime numbers, divisible only by 1 and themselves, are fundamental in mathematics, with widespread importance in fields like cryptography and computer science. They form the building blocks of integers through prime factorization, crucial for exploring number theory and developing theorems like the Goldbach conjecture. In cryptography, prime numbers are vital for secure communication, serving as the basis for algorithms like RSA encryption [15], Diffie-Hellman key exchange [7], and elliptic curve cryptography, thus greatly contributing to the context of data confidentiality and integrity. As a result, prime numbers are indispensable for modern cryptography, driving innovation and research in mathematics and computer science to address evolving security challenges.

The history of primality testing spans millennia, reflecting humanity's quest for efficient methods to identify prime numbers. The primality testing problem ca be described as follows:

Given an integer n, determine whether n is a prime number or a composite number.

Ancient civilizations like the Greeks and Egyptians pioneered rudimentary techniques such as trial division to recognize prime numbers. One early algorithm, the Sieve of Eratosthenes, developed around 200 BCE by the Greek mathematician Eratosthenes, efficiently sieved out primes up to a specified limit, marking a foundational concept in prime number theory. As mathematical knowledge expanded, mathematicians sought more sophisticated methods capable of handling larger numbers. With the advent of computers, demand surged for faster primality testing algorithms. Deterministic algorithms emerged as contenders, such as the Lucas-Lehmer test, which targets Mersenne primes, and the AKS primality test [1], a groundbreaking achievement in computational complexity theory. Despite their capabilities, these algorithms are not always practical for large prime numbers due to high computational requirements. The computational resources required to execute these algorithms can be prohibitive, especially for numbers with thousands or millions of digits. As a result, probabilistic primality testing algorithms, such as the Miller-Rabin algorithm, have gained popularity for their efficiency and scalability in handling large numbers.

The Miller-Rabin algorithm, developed in 1976 by Michael O. Rabin and Gary L. Miller [11, 14], represents a significant advancement in probabilistic primality testing. It offers a compromise between efficiency and accuracy, making it suitable for practical applications involving large numbers. The algorithm, grounded in modular arithmetic, number theory, and probability theory, conducts multiple iterations of a probabilistic test known as the Miller-Rabin test with randomly selected witnesses. Each iteration determines whether a given number n n is a probable prime or definitely composite. By varying the witnesses and adjusting the number of iterations, the algorithm attains a high confidence level in its primality assessments. Its primary objective is to efficiently ascertain the primality of large integers, crucial for numerous applications, notably cryptography, where robust cryptographic systems rely on large prime numbers. The algorithm's rapid and dependable identification of probable primes facilitates the generation of secure cryptographic keys and ensures the integrity of digital communications.

In essence, the Miller-Rabin algorithm marks a significant advancement in primality testing, addressing the escalating computational demands of modern cryptography and computer science. Its probabilistic approach has cemented its status as an essential tool for mathematicians, cryptographers, and computer scientists, underscoring its indispensable role in contemporary computational endeavors.

2.2 Analyzing the Miller-Rabin Algorithm: Characteristics, Advantages, and Limitations

The Miller-Rabin primality test algorithm stands as a cornerstone in the realm of primality testing, offering a robust probabilistic method for determining the primality of large numbers.

1. **Algorithm's key characteristics:**

   - **Probabilistic Nature:** Miller-Rabin algorithm relies on randomized choices during its execution to determine whether a given number is likely to be prime or composite. This probabilistic approach enables the algorithm to provide a fast and efficient means of primality testing, particularly for large numbers.
   - **Accuracy Level:** The Miller-Rabin primality test offers a high level of accuracy for determining the primality of most numbers. By performing multiple iterations with different randomly chosen bases, the algorithm can significantly reduce the probability of incorrectly identifying a composite number as prime (false positive). However, the accuracy of the Miller-Rabin test depends on the number of iterations performed and the properties of the number being tested.
   - **Efficiency:** The Miller-Rabin primality test is known for its efficiency, particularly when dealing with large numbers. It employs modular exponentiation operations, making it faster than certain other primality tests. Despite its efficiency, it still requires multiple iterations to ensure high accuracy, especially for cryptographic applications where reliability is crucial.

2. **Strengths:**

   - **Speed and Efficiency:** The Miller-Rabin primality test excels in speed and efficiency compared to other tests. Its probabilistic nature allows for quick computations, making it dynamic and well-suited for large numbers. Unlike Euler and Solovay-Strassen tests, it balances accuracy with computational cost, making it ideal for rapid primality verification in cryptography and number theory.
   - **Cryptographic Applications:** The Miller-Rabin test's speed and overwhelming accuracy makes it a preferred choice for primality testing in cryptographic applications. Widely utilized in cryptographic protocols and algorithms, its efficiency and accuracy ensures rapid and reliable verification of prime numbers, crucial for maintaining security in various cryptographic systems.
   - **Ease of Implementation:** The Miller-Rabin algorithm uses a straightforward approach: It only relies on modular exponentiation and probabilistic checks, making it accessible to everyone with the most basic programming skills. This simplicity makes it highly practical and can be found in various libraries, such as python's pycryprodome library.

3. **Weaknesses:**

   - **False Positives:** The Miller-Rabin primality test, though highly accurate, can occasionally label a composite number as prime due to its probabilistic nature. While the likelihood of this happening is very low, it's important to recognize the possibility of false positives.

3 Background

3.1 Number Theory

In this section, we recall the background of number theory, which is vital to understand Miller-Rabin algorithm.

3.1.1 Modular Arithmetic

Modular arithmetic is a fundamental branch of mathematics that deals with arithmetic operations involving remainders. It is based on the concept of congruence, which states that two integers have the same remainder when divided by a fixed integer called the modulus.

In modular arithmetic, the notation $a \equiv b \pmod{m}$ signifies that a and b have the same remainder when divided by m. This can also be expressed as a modulo m equals b modulo m. We can define the set of integer modulo m as $\mathbb{Z}_m$.

Example 3.1. Consider the set $\mathbb{Z}_8$ of integers modulo 8. In the expression $13 \equiv 5 \pmod{8}$, 13 and 5 have the same remainder when divided by 8, namely 5. Similarly, $21 \equiv 5 \pmod{8}$ because both 21 and 5 leave a remainder of 5 when divided by 8.

One of the fundamental properties of modular arithmetic is that we can define addition, subtraction, multiplication, and exponentiation within a fixed modulus m by replacing the result with its unique remainder modulo m, and when we perform these operations for any different pairs (a, b) and (a', b') that differs in $\mathbb{Z}$ but is equal in modulo m, then the result of these two pairs remains the same in modulo m. In other words, if a, b, a', b' are values such that $a \equiv a' \pmod{m}$ and $b \equiv b' \pmod{m}$, then it holds that $a + b \equiv a' + b' \pmod{m}$, $a \cdot b \equiv a' \cdot b' \pmod{m}$. Another important property when working with modulo m is that, if $ab \equiv 0 \pmod{m}$ and a is co-prime to m, then $b \equiv 0 \pmod{m}$. Finally, for the time complexity of addition and multiplications modulo m above, it is known that the addition operation requires $O(\log m)$ bits operations and the multiplication operation requires $O(\log^2 m)$ bit operations by naive multiplication and $O(\log m \log\log m)$ bit operations via Harvey-Hoeven algorithm [9]. However, for simplicity we use the $O(\log^2 m)$ bound for time complexity analysis. The time complexity of these operations is important for analyzing the complexity of Miller-Rabin algorithm in Section 5.1.1.

Example 3.2. Consider Example 3.1. Suppose we would like to compute $3 + 6 \pmod{8}$. We see that $3 + 6 = 9$, and since $9 \equiv 1 \pmod{8}$, then it holds that $3 + 6 \equiv 1 \pmod{8}$.

Modular arithmetic provides a framework for handling calculations involving remainders, offering efficient methods for computations that involve cyclical patterns or periodicity. In particular, modular arithmetic is very important for Miller-Rabin test as well, because as specified in Chapter 4, almost all operations in the algorithm will be performed in modulo m, where m is the integer needed for primality test.

3.1.2 Greatest Common Divisor and Multiplicative Inverse

We recall the definition of greatest common divisor and inverse.

Definition 1. Let a and b be integers. The greatest common divisor of a and b, denoted $\text{GCD}(a, b)$, is the largest positive integer n for which n divides a and n divides b.

For an integer n, we denote $\mathbb{Z}_n^\star = \{a \in \mathbb{Z}_n \mid \text{GCD}(a, n) = 1\}$. We have the following theorem:

Theorem 2. Let $a \in \mathbb{Z}_n^\star$, then there exists a integer b such that $ab \equiv 1 \pmod{n}$. Such integer b is called the **multiplicative inverse** of a modulo n.

Proof. For each $i \in \{1, 2, \ldots, n-1\}$ r_i be the remainder of $a \cdot i$ when divided by n. We prove that $\{r_1, r_2, \ldots, r_{n-1}\} = \{1, 2, \ldots, p-1\}$. Indeed, first, suppose there exists $i \neq j$ such that $r_i \equiv r_j \pmod{n}$, this means that $a(i-j) \equiv 0 \pmod{p}$, however, since a is co-prime to n, and $0 < |j-i| < n$, this mean that n does not divide $a(i-j)$, contradiction. Hence $r_1, r_2, \ldots, r_{n-1}$ are pairwise distinct, and since they must be in $\{1, 2, \ldots, n-1\}$, it holds that they must be a permutation of $1, 2, \ldots, n-1$, hence there is some b such that $ab \equiv 1 \pmod{n}$, as desired. □

3.1.3 Fermat's Little Theorem

We recall Fermat's little theorem, a classical but important result in number theory. This result was first stated by Fermat in 1640 but apparently no proof was published at the time and the first known proof was given by Leibnitz. As we will see in Chapter 4, the idea for Miller-Rabin algorithm is also based on this result. Fermat's little theorem can be formally stated as follows:

Theorem 3 (Fermat's little theorem)**.** Let p be a prime number. Then for any integer a such that $\text{GCD}(a, p) = 1$, it holds that:

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof. Recall in the proof of Theorem 2 by denoting $r_i \equiv a \cdot i \pmod{p}$, then $\{r_1, \ldots, r_{p-1}\} = \{1, 2, \ldots, p-1\}$. Hence $r_1 r_2 \ldots r_{p-1} \equiv 1 \cdot 2 \ldots (p-1) \pmod{p}$, or $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$. Since $(p-1)!$ is co-prime to p, this implies that $a^{p-1} - 1$ is divisible by p, as desired. □

3.1.4 Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is a fundamental result in number theory that provides a method for solving systems of congruences.

Theorem 4 (Chinese remainder theorem, simplified)**.** Let $n_1, n_2, \ldots, n_k$ be positive integers satisfying $\text{GCD}(n_i, n_j) = 1$ for all $1 \leq i < j \leq k$, meaning that the moduli n_i are pairwise coprime. The CRT states that for any sequence of integers $a_1, a_2, \ldots, a_k$, there exists a unique integer x modulo the product of the moduli $n_1 \cdot n_2 \cdot \ldots \cdot n_k$ such that:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

In other words, if we have a system of congruences where each modulus is pairwise coprime, then from the Chinese Remainder Theorem we can find a unique solution for x that satisfies all the congruences simultaneously.

Proof. We induct by k. The case $k = 1$ is trivial. Now suppose the statement is true for $k-1$, we need to prove that the statement is true for k. Let $N = n_1 n_2 \ldots_{n_{k-1}}$, by induction, we have found x' s.t $x' \equiv a_1 \pmod{n_i}$ for all $1 \leq i \leq k-1$. Note that all x s.t $x \equiv x' \pmod{N}$ then it also holds that $x \equiv a_1 \pmod{n_i}$ for all $1 \leq i \leq k-1$. Thus it sufficies to prove that there is an x such that $x \equiv x' \pmod{N}$ and $x \equiv a_k \pmod{n_k}$. This is equivalent to proving that there exists integers m, n such that $n_k \cdot m + a_k = N \cdot n + x'$. This is also equivalent to proving that there exists an integer m such that $n_k \cdot m \equiv x' - a_k \pmod{N}$. By letting n' to be the multiplicative inverse of n modulo N, then choosing $m = n'(x' - a_k)$ yields the desired result, hence the statement is true for K and our induction is complete. □

From Chinese Remainder Theorem, to find the result of $x \pmod{n_1 n_2 \dots n_k}$ for pairwise coprime numbers n_i, we can find the result of $x \pmod{n_i}$ for each i first, then combine these result to obtain $x \pmod{n_1 n_2 \dots n_k}$. In this assignment, the Chinese Remainder Theorem will be used to prove Lemma 2 and prove that the error rate of Miller-Rabin is less than $1/2$, which can be found in Section 5.2.

3.1.5 Computing Powers with Repeated Squaring

One of the common operations found in number theory is to raise an integer to a power modulo a number, i.e, given positive integers a, b, n, compute $a^b \pmod{n}$, known as *modular exponentiation*. The naive approach would be to multiply a by itself b time, then compute its remainder modulo n. However, this approach incurs $O(b \log^2 n)$ complexity, and thus is not practical when b is big. Hence, it is necessary to find another method for modular exponentiation. One efficient method for modular exponentiation is the repeated squaring algorithm. This algorithm reduces the number of multiplications required by observe the following recurrence relation:

$$a^b \pmod{n} = \begin{cases} (a^{b/2})^2 \pmod{n} & \text{if } b \equiv 0 \pmod{2} \\ (a^{(b-1)/2})^2 \cdot a \pmod{n} & \text{if } b \equiv 1 \pmod{2} \end{cases}$$

With this recurrence relation, the algorithm to compute $a^b \pmod{n}$ can be done in $O(\log b)$ multiplication steps can be described below:

Algorithm 1 POW(a, b, n)

Input: a, b, n
Output: The result of $a^b \pmod{n}$

1. **If** $b = 1$ **return** $a \pmod{n}$.
2. **If** b is even:
   (a) Compute $d =$POW$(a, b/2, n)$.
   (b) **Return** $d \cdot d \pmod{n}$
3. **Else**:
   (a) Compute $d =$POW$(a, (b-1)/2, n)$.
   (b) **Return** $d \cdot d \cdot a \pmod{n}$

In the algorithm, note that we have to recursively call POW at least $O(\log b)$ times for termination (when $b = 1$). In each time POW is called, we have to perform a constant number of integer multiplications modulo n. It is known that each multiplication modulo n requires $O(\log^2 n)$ bit operations, as specified in Section 3.1.1. Hence the total time complexity of performing modular exponentiation is $O(\log b \log^2 n)$ if naive multiplication is used and $O(\log b \log n \log \log n)$ if Harvey-Hoeven algorithm is used for integer multiplication. The repeated squaring method will be used in Algorithm 4.

3.2 Abstract Algebra

In this section, we recall the background of group theory. Group theory is important here because it will be used for the proof of correctness and error rate of Miller-Rabin's algorithm in

Section 5.2 to prove that the error rate of Miller-Rabin is less than 1/2.

Definition 5 (Groups)**.** A *group* is a set $\mathbb{G}$, together with a binary operation $\cdot$ that satisfies all of the following properties:

- **Closure.** If $a, b \in \mathbb{G}$, then $a \cdot b \in \mathbb{G}$.
- **Associativity.** For all $a, b, c \in \mathbb{G}$, we have: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- **Identity.** There exists an element $e \in \mathbb{G}$ such that for all $a \in \mathbb{G}$ we have $a \cdot e = e \cdot a = a$.
- **Inverse.** For every $a \in \mathbb{G}$, there exists a unique element $b \in \mathbb{G}$ such that $a \cdot b = b \cdot a = e$. The element b is called the **inverse** element of a and is denoted by a^{-1}.

Example 3.3. For $n \geq 2$, let $\mathbb{Z}_n = \{0, 1, ..., n-1\}$, together with the addition modulo n. Then $\mathbb{Z}_n$ associated with operation "+" forms a group, denoted by $(\mathbb{Z}_n, +)$. The identity element of the group is 0. For every $a \neq 0$, the inverse of a is $n - a$.

Definition 6 (Subgroups)**.** Let $\mathbb{G}$ be a group with binary operation $\cdot$, then a subset $\mathbb{H}$ of $\mathbb{G}$ is said to be *subgroup* of $\mathbb{G}$ if the identity element e of $\mathbb{G}$ is in $\mathbb{H}$, and $\mathbb{H}$ is also a group under operation $\cdot$ as well.

Example 3.4. Consider $(\mathbb{Z}, +)$. Then the set $2\mathbb{Z} = \{2x \mid x \in \mathbb{Z}\}$ is a subgroup of $\mathbb{Z}$, since $2\mathbb{Z} \subseteq \mathbb{Z}$, and it can be verified that $2\mathbb{Z}$ forms a group under operation $+$.

Finally, let us move to an important lemma of subgroup, which will be used in proof of correctness and error rate of Miller-Rabin's algorithm in Section 5.2.

Lemma 1. If $\mathbb{H}$ and $\mathbb{G}$ are both finite groups and $\mathbb{H}$ is a proper subgroup of $\mathbb{G}$ (i.e, $\mathbb{H}$ is a subgroup of $\mathbb{G}$ but not $\mathbb{G}$ itself), then $|\mathbb{H}|$ divides $|\mathbb{G}|$

Proof. For a set $S \subseteq \mathbb{G}$, we denote $S \cdot \mathbb{H} = \{s \cdot h \mid s \in S, h \in \mathbb{H}\}$. We consider the set S constructed via the following steps:

1. Initially, let $S = \perp$.
2. Pick any element s from $\mathbb{G}$ such that $s \notin S \cdot \mathbb{H}$ and update S to be $S = S \cup s$.
3. Repeat Step 2 until we can no longer pick any s from $\mathbb{G}$.

We prove that $|\mathbb{G}| = |S| \cdot |\mathbb{H}|$. Indeed, by the definition of S, every element in $\mathbb{G}$ can be written as $s \cdot h$ for some $s \in S$ and $h \in \mathbb{H}$, and every element of $S \cdot \mathbb{H}$ is also in $\mathbb{G}$ since both $S, \mathbb{H}$ are subsets of $\mathbb{G}$, hence $G = S \cdot \mathbb{H}$. Now, it sufficies to prove that $|S \cdot \mathbb{H}| = |S| \cdot |\mathbb{H}|$. We see that, $S \cdot \mathbb{H} = \bigcup_{s \in S} s \cdot \mathbb{H}$, thus it sufficies to prove that for any $s, s' \in S$, the two sets $s \cdot \mathbb{H}$ and $s' \cdot \mathbb{H}$ are disjoint. Indeed, suppose there exists two element $h, h' \in \mathbb{H}$ such that $s \cdot h = s' \cdot h'$, this means that $s' \in s \cdot \mathbb{H}$. However, this contradicts the definition of s' in Step 2, because when s' is added to S, we must have $s' \notin s \cdot \mathbb{H}$. Thus the sets $s \cdot \mathbb{H}$ are pairwise disjoint and thus $|S \cdot \mathbb{H}| = |S| \cdot |\mathbb{H}|$ since $|s \cdot \mathbb{H}| = |\mathbb{H}|$ for any $s \in S$, as desired. □

4 The Algorithms

4.1 Description and Steps

The Miller-Rabin algorithm is a probabilistic primality test that determines whether a given number is likely to be prime or not. The algorithm is based on two observations: First is Fermat's little theorem and second is the following Lemma:

Lemma 2. Consider positive integers n, a such that n is odd and $a^2 \equiv 1 \pmod{n}$. Then it holds that $a \equiv \pm 1 \pmod{n}$ if and only if n is a power of a prime number.

Proof. We see that $a^2 \equiv 1 \pmod{n}$ is equivalent to n divides $(a-1)(a+1)$. When $n = p^e$ is a prime number, we see that since $\text{GCD}(a-1, a+1)$ divides 2, then at most one of $a-1, a+1$ contains p in its factorization, thus it holds that p^e must divide entirely $a-1$ or $a+1$. In other words, $a \equiv \pm 1 \pmod{n}$.

When n is not a power of a prime, we write $n = n_1 n_2$ where $n_1, n_2 > 1$ and $\text{GCD}(n_1, n_2) = 1$. In this case, we can choose $a \equiv 1 \pmod{n_1}$ and $a \equiv -1 \pmod{n_2}$ by Chinese Remainder Theorem. We see that n divides $a^2 - 1$ and $a \not\equiv \pm 1 \pmod{n}$. □

Now, let us go back to Fermat's theorem: If n is a prime number, then it holds that $a^{n-1} \equiv 1 \pmod{n}$. Thus, naturally, we would check whether $a^{n-1} \equiv 1 \pmod{n}$ or not and if not, then n is definitely a composite. However, this is not sufficient to ensure that n is a prime, for example, when $n = 561$ then no matter what value a is chosen, then (3) still holds. Fortunately, this is where Lemma 2 comes in.

Now, assume that (3) holds, i.e, $a^{n-1} \equiv 1 \pmod{n}$, we write $n - 1 = 2^s d$ where d is odd, thus $a^{2^s d} \equiv 1 \pmod{n}$. Now by Lemma 2 if n is a prime then it must hold that $a^{2^{s-1} d} \equiv \pm 1 \pmod{n}$, thus if this does not hold then n is definitely a composite number. Suppose it passes the above check, then if $a^{2^{s-1} d} \equiv -1 \pmod{n}$, then n "could be possibly a prime". Otherwise, if $a^{2^{s-1} d} \equiv 1 \pmod{n}$, we continue to check if $a^{2^{s-2} d} \equiv \pm 1 \pmod{n}$ and similarly, if this does not hold then n is definitely a composite number. Again, if the case $a^{2^{s-2} d} \equiv 1 \pmod{n}$ happens then we continue down to $a^{2^{s-3} d}$ and so on until we reach a^d. If we could reach a^d, then if $a^d \equiv \pm 1 \pmod{n}$ then "could be possibly a prime", otherwise n is definitely a composite number.

The process above is the intuition if we start from $a^{2^s d}$ and end at a^d. However, for efficiency reasons, we could start at a^d and end at $a^{2^s d}$ reversely such that the above logic does not change. The idea for this can be described as follows:

- Initialize $x \equiv a^d \pmod{n}$.
- Check if $x^2 \equiv 1 \pmod{n}$ and check if $x \equiv \pm 1 \pmod{n}$. If not then n is definitely a composite. Then compute $x \equiv x^2 \pmod{n}$. Repeat the process above s times.
- Finally, check if $x \equiv 1 \pmod{n}$. If not, then n is definitely a composite.

By repeating the process above k times, we are then convinced that n is a prime with overwhelming probability. With the idea above, we are now ready to formally describe Miller-Rabin's algorithm. First, we describe the WITNESS algorithm in Algorithm 2. Then we describe MILLER-RABIN employing WITNESS as a sub-algorithm in Algorithm 4.

Algorithm 2 WITNESS(a, n)

Input: a, n
Output: 0/1

1. Compute $n - 1 = 2^s d$ where d is odd.
2. Compute $x =$POW(a, d, n).
3. **For** i from 1 to s do:
   (a) Compute $y \equiv x^2 \pmod{n}$.
   (b) **If** $y \equiv 1 \pmod{n}$ and $x \not\equiv \pm 1 \mod n$ then **return** 0.
   (c) Compute $x = y$.
4. **If** $x \not\equiv 1 \pmod{n}$ **then return** 0.
5. **Return** 1.

Algorithm 3 MILLER-RABIN(n, k)

Input: n, k
Output: 0/1

1. **If** n is even and $n \neq 2$ then **return** 0.
2. **If** $n = 2$ then **return** 1.
3. **For** i from 1 to k do:
   (a) Choose $a \xleftarrow{\$} \{2, 3, \ldots, n-2\}$
   (b) **If** WITNESS(a, n)=0 then **return** 0.
4. **Return** 1.

In the WITNESS algorithm, initially we set $x \equiv a^d \pmod{n}$. In the i-th iteration, we set x to be $x \equiv a^{2^i d} \pmod{n}$. The crucial point is when $i = r$, where r is the smallest integer such that $a^{2^r d} \equiv 1 \pmod{n}$. In this case, step 3$b$) simply check if $a^{2^{r-1} d} \equiv -1 \pmod{n}$ and if not, then n must be a composite. Note that n passes this check when $i = r$, then it also passes the remaining checks when $r + 1 \leq i \leq s$ as well, since at this time x and y are both equal to 1 (since $x \equiv a^{2^{i-1} d} \equiv 1 \pmod{n}$ when $i \geq r + 1$). If an integer a satisfies WITNESS$(a, n) = 0$ to be the *witness* of n. Otherwise, a is said to be a *non-witness* of n. The notation of witness and non-witness will be used for the proof of correctness and error rate in Section 5.2.

Finally, the MILLER-RABIN repeat the whole process of WITNESS k times by choosing a randomly and execute WITNESS(a, n). By choosing k to be large, if MILLER-RABIN returns 1 then we are convinced that n is a prime with overwhelming probability.

Next, we illustrate the flowchart of the algorithm. The flowchart of the Witness algorithm and overall of Miller-Rabin algorithm can be found in Figure 1, 2 below.

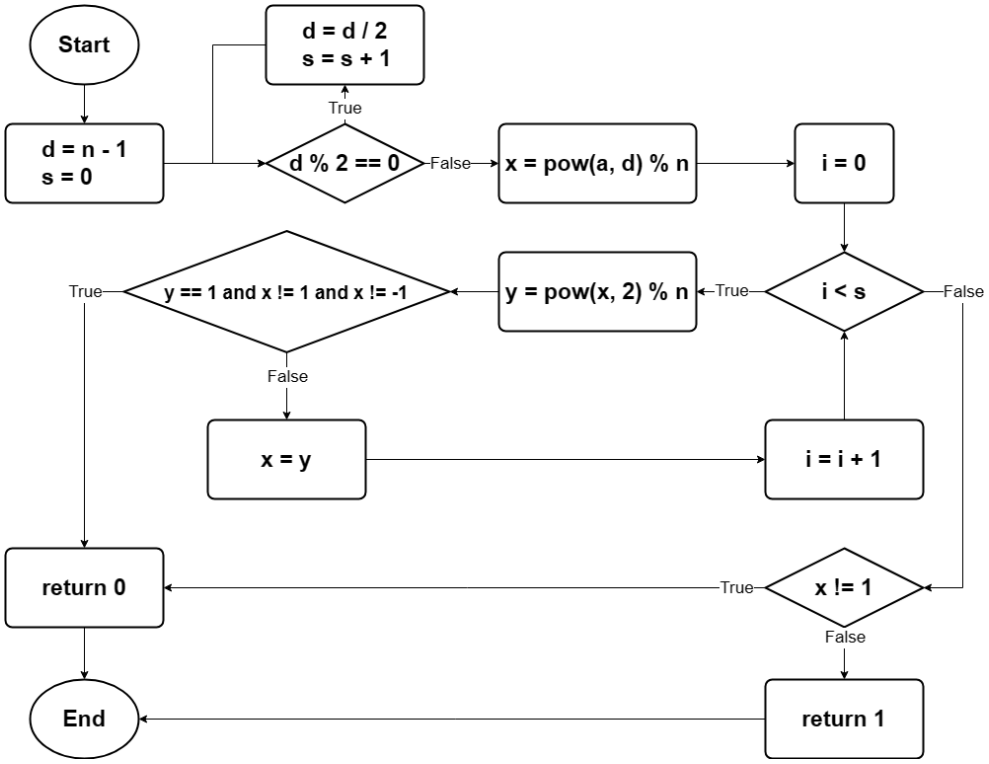

Figure 1: Flowchart of Witness Algorithm

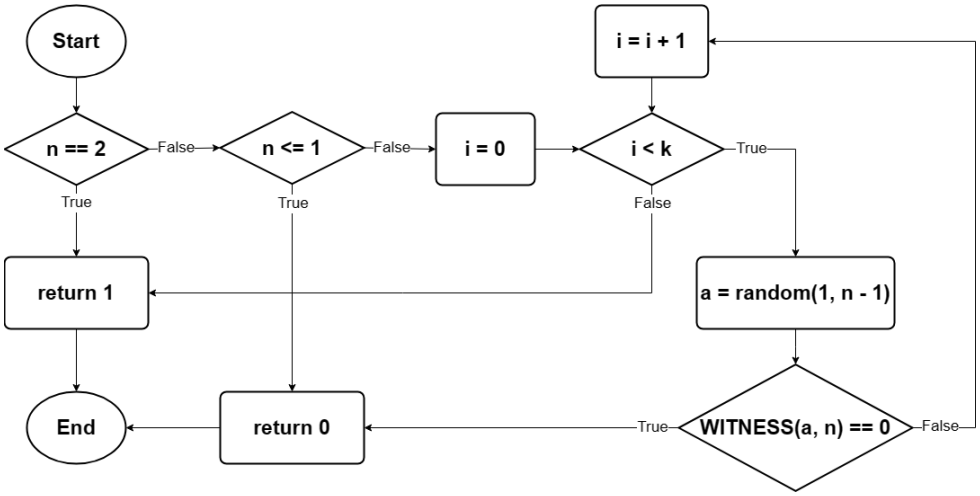

Figure 2: Flowchart of Miller-Rabin Algorithm

4.2 Algorithm's Paradigm

The Miller-Rabin algorithm is an example of a *randomized design paradigm*, which means that it relies on a source of random numbers and makes random choices during execution to

achieve a desired result. For the case of Miller-Rabin, the randomization can be seen in the **For** loop since each time, it chooses a random number a and executes WITNESS(a, n). Randomized algorithms are often used when deterministic algorithms are too slow, too complex, or do not exist for the problem at hand. Randomized algorithms can be further classified into two types: Monte Carlo and Las Vegas. Monte Carlo algorithms may produce incorrect results with some probability, but they always run in a fixed amount of time. Las Vegas algorithms always produce correct results, but they may run for an unpredictable amount of time.

The Miller-Rabin algorithm is a Monte Carlo algorithm, because it may return a false positive (i.e., declare a composite number as prime) with some probability, but it always terminates after exactly k iterations. The Monte Carlo design strategy affects the correctness and efficiency of Miller-Rabin as follows:

- **Efficiency:** The Miller-Rabin algorithm is indeed a randomized Monte Carlo type algorithm, which generally makes it more efficient for large integers compared to deterministic tests like the AKS algorithm. Since the algorithm always terminate at most k steps, and in each step, the time complexity is at most $O(\log^3 n)$ (which will be analysed in more detail in Chapter 5.1.1), the time complexity is at most $O(k \log^3 n)$, which is highly efficient.
- **Correctness:** The probability of a false positive can be reduced by increasing the number of iterations or bases tested, but it can never be eliminated completely. However, for most practical applications, the probability of a false positive is sufficiently small to be acceptable. The more iterations with different a are tried, the better the accuracy of the test. It can be shown that if n is composite, then at most $1/4$ of the bases a are strong liars for n.[14] As a consequence, if n is composite then running k iterations of the Miller–Rabin test will declare n probably prime with a probability at most 4^{-k}. Therefore, by choosing a sufficiently large k, the algorithm can achieve a high level of accuracy.

In summary, the Miller-Rabin algorithm is a fast and high accuracy primality test that uses a randomized design paradigm.

4.3 Toy Example

In this section, we give an example of Miller-Rabin test above.

Example 4.1. Suppose we want to test if $n = 37$ is prime. For simplicity, we choose $k = 2$.

- The number $n = 37$ pass the base cases in step 1.
- We find that
$$n - 1 = 36 = 2^2 \times 9,$$
so we have $d = 9$ and $s = 2$.
- Next, We pick a random number $a = 2$.
- Compute $x = 2^9 \pmod{37} = 31$.
- Compute $y = x^2 \pmod{37} = 36$. $(s = 0)$
- Since $y = 36 \neq 1$, we continue.$(s = 0)$
- Compute $x = y = 36$. $(s = 0)$
- Compute $y = x^2 \pmod{37} = 1$. $(s = 1)$

- Since $y = 1$ but $x = 36 = n - 1$, we continue.($s = 1$)
- Compute $x = y = 1$. ($s = 1$)
- Since $x = 1$, the algorithm WITNESS(a, n) returns 1, and we continue choosing another random $a = 3$ for MILLER-RABIN.
- Compute $x = 3^9 \pmod{37} = 36$.
- Compute $y = x^2 \pmod{37} = 1$. ($s = 0$)
- Since $y = 1$ but $x = 36 = n - 1$, we continue.($s = 0$)
- Compute $x = y = 1$. ($s = 0$)
- Compute $y = 1^2 \pmod{37} = 1$. ($s = 1$)
- Since $y = 1$ but $x = 1$, we continue.($s = 1$)
- Compute $x = y = 1$. ($s = 1$)
- Since $x = 1$, the algorithm WITNESS(a, n) returns 1
- Since we have done two iterations which return 1, we return true.

This shows that the Miller-Rabin algorithm is reliable with a toy example of primality checking for $n = 13$.

4.4 Optimization Techniques and Algorithm's Behaviour

4.4.1 Early Termination

Note that in the WITNESS algorithm, there are several cases where we can terminate the algorithm, instead of running the whole loop s times. This helps us in reducing the time complexity in several cases. They are as follows:

1. At line 2 of the WITNESS algorithm, if $x \equiv 1 \pmod{n}$, then we can return 1 immediately. This is because this means that $a^d \equiv 1 \pmod{n}$. Now, in the **For** loop (lines 3 a-c), then in the i-th iteration we have that $x \equiv a^{2^{s-1}d} \equiv 1 \pmod{n}$ and $y \equiv a^{2^s d} \equiv 1 \pmod{n}$, thus the check in line 3b is satisfied and the algorithm never returns 0. Thus the algorithm always returns 1 if $a^d \equiv 1 \pmod{n}$ and thus we can return 1 right after line 2 to reduce the computation complexity.

2. In addition to 1) above, at line 3b of the WITNESS algorithm, instead of checking if $x \equiv \pm 1 \pmod{n}$, we simply check if $x \equiv -1 \pmod{n}$ then we can return 1 immediately, otherwise return 0. To see why, we consider the case $a^d \not\equiv 1 \pmod{n}$ so that we proceed to the **For** loop (since if $a^d \equiv 1 \pmod{n}$ then we return 1 immediately according to above). Let $r \leq s$ be the smallest positive integer such that $a^{2^r d} \equiv 1 \pmod{n}$, if there is such an r. To see why, let us consider steps 1 to r, step r and steps $r + 1$ to s in the loop. In steps 1 to $r - 1$, we see that y is never equal to 1 mod n due to the definition of r, and thus nothing happens. In step r we see that $y \equiv a^{2^r d} \equiv 1 \pmod{n}$. Now, if n is a prime, then it must hold that $x \equiv a^{2^{r-1}d} \equiv -1 \pmod{n}$ according to the definition of r. Hence if $x \not\equiv 1 \pmod{n}$ then n is a composite. Otherwise, then from step $r + 1$ to s the values x and y are always equal to 1 mod n and thus the algorithm is garguanted to return 1. Hence executing these steps are not necessary. Hence checking if $x \equiv -1 \pmod{n}$ and then we can return 1 immediately and 0 otherwise could reduce the computation cost as well.

4.4.2 Parallel Computing

The standard Miller-Rabin test can be optimized using parallel computation, which can significantly reduce the time required to run the test. Without this optimization, the algorithm need to check k times with k different arbitrary number a alternately. Although for practical purposes, running the test around 20-30 times is often considered sufficient to achieve a reasonable level of certainty. However, in term of large number n, it may lead to a large number s which means each test would run upto s iteration. Consequently, the overall execution time of the algorithm is high. We can take a step further and get the most out of the hardware by running those checks parallel.

An overview of how the Miller-Rabin test can be implemented with parallel computation:

1. **Preprocessing:** Decompose the number $n-1$ into $2^s \cdot d$, where d is odd.
2. **Parallel Execution:** Perform multiple rounds of the test in parallel using multiple threads. Each thread picks a random number a and executes WITNESS(a, n).
3. **Aggregation:** Collect the results from all parallel executions. If any execution fails the test, return 0, otherwise return 1.

A modified version of the Miller-Rabin algorithm, implemented with parallel computation, has been shown to work up to 50% faster than the standard iterative algorithm [6]. Implementing the algorithm in a way that allows for parallel processing can significantly speed up the testing process and makes it a more reliable and faster option, especially for large choice of number k as well as large n (which may leads to a large s).

4.4.3 Deterministic variants

The Miller–Rabin algorithm can be made deterministic by trying all possible values of a below a certain limit. Taking n as the limit would imply $O(n)$ trials, hence the running time would be exponential with respect to the size $\log n$ of the input. To improve the running time, the challenge is then to lower the limit as much as possible while keeping the test reliable.

If the tested number n is composite, the strong liars a coprime to n are contained in a proper subgroup of the group $(\mathbf{Z}/n\mathbf{Z})^*$, which means that if we test all a from a set which generates $(\mathbf{Z}/n\mathbf{Z})^*$, one of them must lie outside the said subgroup, hence must be a witness for the compositeness of n. Miller proved that, assuming the truth of the generalized Riemann hypothesis (GRH), we just need to check $a \in [2, O(\log^2 n)]$ [11]. Bach then reduced it by simply considering $a \in [2, 2\log^2 n]$ [3].

However, when the number n to be tested is large enough, trying all $a < 2\log^2 n$ is not necessary, as much smaller sets of potential witnesses are known to suffice. For example, Pomerance, Selfridge, Wagstaff [5] and Jaeschke [10] have verified that:

- if $n < 2047$, it is enough to test $a = 2$;
- if $n \leq 3.10^9$, it is enough to test $a = 2, 3, 5$, and 7;
- if $n \leq 2^{64}$, it is enough to test $a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$, and 37.

Therefore, we have the algorithm version (accuracy 100%) of the test as follows:

Algorithm 4 MILLER-RABIN(n, k)

Input: n, k
Output: 0/1

1. **If** n is even then **return** 0.
2. Initialize $checkSet = \{p \mid p \text{ is prime } \wedge\ p < 2\log^2 n\}$;
3. **For** a in $checkSet$:

   If WITNESS(a, n)=0 then **return** 0.
4. **Return** 1.

This deterministic variant provides 100% accuracy (instead of the original one, where there is a $1/4^k$ error rate) and does not depend on k. Instead the factor k in time complexity is replaced by $O(\log^2 n/\log\log n)$ due to the prime number's theorem.

5 Complexity Analysis and Proof of Correctness

5.1 Miller-Rabin's Algorithm Complexity Analysis

5.1.1 Time Complexity

Let n be the number to be tested for primality, and k be the number of iterations (witnesses) used in the Miller-Rabin algorithm. The time complexity of the Miller-Rabin algorithm mainly depends on the number of iterations (k) and the size of n as follows.

First, we consider WITNESS algorithm. This algorithm requires only modular exponentiation. We can easily analyse the time complexity of WITNESS as follows: In line 2 of the WITNESS algorithm, we need to compute $x \equiv a^d \pmod{n}$. By Subsection 3.1.5, computing $a^d \pmod{n}$ using repeated squaring incurs $O(\log d \log^2 n) = O(\log^3 n)$ time complexity. Next, in the for loop (lines 3a, 3b and 3c), we have to repeat $s = O(\log n)$ iterations, and in each iteration, we need to compute $y = x^2 \pmod{n}$. It is known that each multiplication and division will take $O(\log^2 n)$ bit operations as specified in Section 3.1.1. So this makes $O(\log^3 n)$ bit operations in the for loop. Hence, the WITNESS algorithm incurs $O(\log^3 n)$ time complexity. Then, the number of iterations k directly affects the accuracy of the Miller-Rabin algorithm. Generally, a higher number of iterations leads to a higher probability of correctly identifying primes. However, the number of iterations also affects the time complexity. Thus, the overall time complexity can be expressed as $O(k \cdot \log^3 n)$ (or $O(k \cdot \log^2 n \log\log n)$ if integer multiplication is performed by Harvey-Hoeven algorithm) where k is the number of iterations and n is the number being tested for primality.

5.1.2 Space Complexity

The space complexity of the Miller-Rabin algorithm is determined by the number of bits required to store the input and any intermediate variables used in the algorithm.

To begin with, the space required to store the input number (n) is $O(\log n)$, assuming a binary representation. The space required to store intermediate variables during the algorithm execution is also $O(\log n)$ (since every number is in $\{0, 1, \ldots, n-1\}$) and there is a constant number of intermediate variables. Hence, we conclude that the overall space complexity of the Miller-Rabin algorithm is $O(\log n)$ since it dominates the space requirement.

5.2 Correctness and Error Rate of the Algorithm

In this section, we prove the correctness and error rate of Miller-Rabin's algorithm. First, we prove that the algorithm always output 1 for a prime number n via Lemma 3, then we prove that the algorithm outputs 1 for a composite number with probability at most $(1/2)^k$ via Lemma 4, which implies that the error rate of Miller-Rabin's algorithm is at most $(1/2)^k$. Note that there are two known proofs of error bounds for Miller-Rabin test, the first one is simple and provide the $(1/2)^k$ error bound, while the second proof is based on the idea of the first proof and thus, manages to reduce the error rate down to $(1/4)^k$. However, to achieve this bound it requires heavy casework and more lemmas such as Hensel lifting lemma for counting the total number of witnesses of n (see [8, Theorem 5.13]). Hence in we only present the proof of $(1/2)^k$ for simplicity, and the proof sketch for $1/4^k$ error rate can be found in Appendix 8.

Lemma 3 (Correctness)**.** If n is an odd prime number, then MILLER-RABIN outputs 1 with probability 1.

Proof. Suppose n is an odd prime number and $p-1 = 2^s \cdot d$ for some odd integer d. Now, consider the **for** loop of lines $3a$ to $3c$ of WITNESS. Initially x is initalized to be $x \equiv a^d \pmod{n}$ according

to line 2, and thus in the i-th iteration we see that $x \equiv a^{2^i d} \pmod{n}$ for all $i \leq s$. Thus in the **for** loop, the algorithm returns 0 if and only if $a^{2^i d} \equiv 1 \pmod{n}$ and $a^{2^{i-1} d} \not\equiv \pm 1 \pmod{n}$. However, since $a^{2^i d} \equiv 1 \pmod{n}$, we have that $(a^{2^{i-1} d} - 1)(a^{2^{i-1} d} + 1) \equiv 0 \pmod{n}$. Since n is a prime, one of $(a^{2^{i-1} d} - 1)$ and $(a^{2^{i-1} d} + 1)$ must be divisible by n, meaning that the algorithm does not terminate and return 0 during the loop. Finally, consider line 4 of WITNESS, we see that since $x \equiv a^{2^s d} \equiv a^{n-1} \equiv 1 \pmod{n}$, hence the algorithm does not return 0 after this line. Finally, since the MILLER-RABIN does not return 0 in any iteration executing WITNESS, it must output 1 in the end, as desired. □

Lemma 4 (Error Rate)**.** If n is an odd composite number, then MILLER-RABIN outputs 1 with probability at most $(1/2)^k$.

Proof. It suffices to prove that the number of non-witnesses is at most $(n-1)/2$. First, we prove that non-witness must be in $\mathbb{Z}_n^\star$. Indeed, for any non-witness a, if the algorithm WITNESS returns 1 then we must have $a^{n-1} \equiv 1 \pmod{n}$ as in line 4, thus $\text{GCD}(a, n) = 1$ or equivalently $a \in \mathbb{Z}_n^\star$.

Now we have proved that $a \in \mathbb{Z}_n^\star$, the next idea is to prove that any such non-witness a is contained in a proper subgroup S of $\mathbb{Z}_n^\star$, and since S is a proper subgroup of $\mathbb{Z}_n^\star$, we have $|S|$ divides $|\mathbb{Z}_n^\star|$ according to Lemma 1, thus $|S| \leq |\mathbb{Z}_n^\star|/2$, and hence the number of non-witnesses is less than $n/2$. To proceed with the proof, we consider two cases:

- Case 1 : There exists $c \in \mathbb{Z}_n^\star$ such that $c^{n-1} \not\equiv 1 \pmod{n}$. In this case, we consider

  $$S = \{x \in \mathbb{Z}_n^\star \mid x^{n-1} \equiv 1 \pmod{n}\}.$$

  We see that any non-witness a must satisfy $a \in S$. Thus it suffices to prove that S is a proper subgroup of $\mathbb{Z}_n^\star$. First, we see that $S \subseteq \mathbb{Z}_n^\star$ and $S \neq \mathbb{Z}_n^\star$ due to the existence of c. Second, we see that i) for any $x, y \in S$ then $xy \pmod{n}$ is also in S and ii) for any $x \in S$ let x' be the multiplicative inverse of $x \pmod{n}$ (which exists by Theorem 2), then since $x^{n-1} \equiv 1 \pmod{n}$, we have $(x')^{n-1} \equiv 1 \pmod{n}$, hence $x' \in S$, thus the set S forms a group, as desired.

- Case 2 : $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}_n^\star$. First, let us prove that n cannot be a prime power and not a prime. Indeed, suppose $n = p^k$ for some odd prime p and $k > 1$. Since $a^{p^k - 1} \equiv 1 \pmod{p^k}$ for all $a \in \mathbb{Z}_{p^k}^\star$ and $k \geq 2$, we consider $a = p - 1$ and thus it must holds that $(p-1)^{p^k - 1} \equiv 1 \pmod{p^2}$. However, by binomial expansion it holds that $(p-1)^{p^k - 1} \equiv -(p^k - 1)p + 1 \equiv -p + 1 \not\equiv 1 \pmod{p^2}$, contradiction. Hence n cannot be a prime power.

  Now, let t' denote the **smallest** non-negative integer such that $x^{2^{t'} d} \equiv 1 \pmod{n}$ for all $x \in \mathbb{Z}_n^\star$, and note that t' always exists and must be positive (otherwise choosing $x = -1$ would lead to contradiction). Consider any non-witness a, since $a^{2^{t'} d} \equiv 1 \pmod{n}$ and the algorithm returns 1, it must hold that $a^{2^{t'-1} d} \equiv \pm 1 \pmod{n}$ due to the **for** loop (lines 3a to 3c in the WITNESS algorithm). With this observation, we straightforwardly consider the set S to be

  $$S = \{x \in \mathbb{Z}_n^\star \mid x^{2^{t'-1} d} \equiv \pm 1 \pmod{n}\},$$

  and see that any non-witness a must be in S. Also, one can check that S is indeed a group. Finally, to show that S is a proper subgroup of $\mathbb{Z}_n^\star \setminus S$, let us find an element $c \in \mathbb{Z}_n^\star \setminus S$. By the definition of t', there exists a value v such that $v^{2^{t'} d} \equiv 1 \pmod{n}$ and $v^{2^{t'-1} d} \not\equiv 1$

(mod n). Let $n = \prod_{i=1}^{k} n_i$ where each n_i is a prime power and $k \geq 2$, we show that there exists i such that $v^{2^{t'-1}d} \equiv -1 \pmod{n_i}$. Indeed, since i) n_i divides $(v^{2^{t'-1}d}-1)(v^{2^{t'-1}d}+1)$, ii) 2 divides $\text{GCD}(v^{2^{t'-1}d}-1, v^{2^{t'-1}d}+1)$, and iii) n_i is an odd prime power, n_i must divide one of $v^{2^{t'-1}d}-1$ and $v^{2^{t'-1}d}+1$. If n_i does not divide $v^{2^{t'-1}d}+1$ for all i, then we must have n_i divides $v^{2^{t'-1}d}-1$ for all i and thus n divides $v^{2^{t'-1}d}-1$, however this contradicts the definition of v.

Hence there exists i such that n_i divides $v^{2^{t'-1}d}+1$. We choose c such that $c \equiv v \pmod{n_i}$ and $c \equiv 1 \pmod{n/n_i}$ by Chinese Remainder Theorem in Section 3.1.4, then $c^{2^{t'-1}d} \equiv -1 \pmod{n_i}$ and $c^{2^{t'-1}d} \equiv 1 \pmod{n/n_i}$. Since both n_i and n/n_i are both greater than 1 (n is not a prime power), it holds that $c \notin S$ and this implies that S is a subgroup of $\mathbb{Z}_n^\star$, as desired..

Now, if n is a composite number, we see that each **for** loop has probability at most $1/2$ to choose a non-witness a. By repeating k times, the Miller-Rabin returns 1 if for all the iterations, the integer a is a non-witness, i.e, WITNESS(a, n) returns 1. This happens with probability $(1/2)^k$, as desired. □

Hence, from the two Lemmas above, one can see that, if MILLER-RABIN returns 0, then n is definitely a composite number. Otherwise, if MILLER-RABIN returns 1, then n is a prime number with probability of at least $1 - (1/2)^k$.

5.3 Edge Cases and Considerations

In the previous section, we have proved that the Miller-Rabin test provides overwhelming accuracy, with an error rate at most $(1/2)^k$ for k iterations. While it is highly efficient and accurate in practice, there are certain edge cases and misuse of parameters where it might cause the algorithm to produce incorrect results, which must be avoided. Some of these cases are as below:

Strong Pseudoprime: A strong pseudoprime is a composite number n such that WITNESS(a, n) returns 1 for a certain base a. In this case n is said to be a strong pseudoprime to base a. These numbers are important because they could cause Miller-Rabin test to produce false positive results with positive probability. Formally, when we write $n-1 = 2^s d$ for some odd integer d, then n is a strong pseudoprime to base a iff

$$a^d \equiv 1 \pmod{n} \quad \text{or} \quad a^{2^r d} \equiv -1 \pmod{n} \quad \text{for some} \quad 0 \leq r \leq s-1$$

We can check that if a has the form above, then WITNESS(a, n) returns 1, and if a is chosen all k times then the Miller-Rabin test falsely identifies n as a prime number.

Example 5.1. For example, when $a = 3$, then we can check that $47197 = 109 \cdot 433$ is a strong pseudoprime to base 3. Indeed, we have $47196 = 2^2 \cdot 11799$ and since $3^{11799} \equiv 1 \pmod{47197}$, we conclude that 47197 is a composite number and is a strong pseudoprime to base 3.

Fortunately, there is no number that is a strong pseudoprime to all bases, which has been proved in Subsection 5.2, and the proof implies that there are at most $n/2$ such bases from 1 to n, meaning that no composite numbers could pass Miller-Rabin test completely. Nevertheless, it should be noted that Miller-Rabin test is more likely to falsely identify these numbers are primes, even with negligible probability, but it still could happen.

Small Number of Iterations: As we have seen, when the number of iterations is k, then the algorithm incorrectly identifies a composite number as 1 with error probability $(1/2^k)$. This means that the error rate greatly increases when the number of iterations is too small (for example, $k = 2$). This is indeed the case, which is specified in the example below:

Example 5.2. Consider $n = 121 = 11 \cdot 11$. Let us compute the probability when the algorithm returns 1 when the number of iterations k is small. Now let $k = 2$. We see that the number of non-witnesses of n are $a = \{1, 3, 9, 27, 81\}$. Hence, the probability that a non-witness of 341 is chosen is equal to $5/121 \approx 0.041$. When $k = 2$, the algorithm returns 0 if and only if for both times, a non-witness of 341 is chosen. Thus we conclude that the probability that the algorithm returns 1 is $25/14641 \approx 0.001707$, which is not negligible and thus not acceptable for applications. Hence, we should increase k to ensure that this probability is negligible. In practice, setting $k = 80$ should be enough for cryptographic applications.

Fixed Choice of Bases: In the algorithm, recall that the bases are chosen uniformly in $\mathbb{Z}_n^\star$. However, for efficiency, one may think of choosing a **fixed** choice of bases. For example, let $S = \{2, 3, 5, 7, \ldots, 11\}$ consisting of primes less than 13, and one might modify Miller-Algorithm as follows: The algorithm returns 1 if and only if WITNESS$(a, n) = 1$ for all $a \in S$, where WITNESS is defined in Algorithm 2. However, given a set S of bases, [2] proposed a heuristic way to choose a composite n that passes all the bases in S. Moreover, in [2, Appendix F], they have provided a large composite number that passes Java's Miller-Rabin test implementation, whose bases are chosen to be the set of primes between 2 and 1000. A more explicit example using small number can be found below:

Example 5.3. Suppose in Miller-Rabin's algorithm, instead of selecting a randomly in $\{0, 1, \ldots, n-1\}$, we instead simply select a to be in $\{2, 3, 5, 7, 11\}$ and thus the Miller-Rabin returns 1 if and only if WITNESS$(a, n) = 1$ for all $a \in \{2, 3, 5, 7, 11\}$. In this case, we could choose the number $n = 1253075960778449601068457392 3$. One can check that n is a composite number since it is divisible by 286472803 and all numbers from 2 to 11 are all non-witnesses of n. Hence, the Miller-Rabin algorithm accepts n is a prime in this case when it should be a composite. From this, we should not rely on any fixed choice of base and expect it to successfully test all numbers.

To mitigate these issues, especially when dealing with large numbers, it's common practice to use multiple rounds of the Miller-Rabin test with different randomly chosen bases and set the number of iteration to be large (for example, at least 80 times) to ensure that the algorithm behaves as expected with small error rate.

6 Implementations, Application and Case Studies

6.1 Implementation and Optimization Techniques

We provide a two-version implementation of the Miller-Rabin Algorithm in Python. Python is chosen since it is friendly for programming. The source code of our implementation is available at: https://github.com/redevil24/MCS-AdvancedAlgorithms_Assignment

One of the possible optimization techniques we can think of is to use bitwise operations instead of arithmetic operations in several situations, since bitwise operations are substantially faster in most programming languages and require less use of power and resources in processors. For example, when checking if $n \equiv 1 \pmod 2$ in Line 1 of MILLER-RABIN, we can instead check that $n\&1 = 1$ instead. In addition, when writing $n - 1 = 2^s \cdot d$, we can also use bitwise operations to find s, d instead of arithmetic as well. The idea is follows: Initialize $s = 0$ and $d = n - 1$, and while $d\&1 = 0$, we increase s by 1 and set $d := d >> 1$.

6.2 Benchmarking

We compare the result of our Miller-Rabin implementation with Fermat and AKS algorithm. We try with different numbers N and use accuracy (whether the algorithm correctly identifies N as a prime or composite) and execution time (in milliseconds) as the metrics for comparison. The result can be seen below.

Algorithm	Input (N)	Primality	Output	Time (ms)
Fermat ($k = 5$)	13	Prime	Prime	0.36
	127	Prime	Prime	0.45
	209	Composite	Composite	0.49
	561 (Chamichael)	Composite	Prime	0.50
	2027	Prime	Prime	0.51
	41041 (Chamichael)	Composite	Prime	0.53
	15,485,863	Prime	Prime	0.49
	987,654,321	Composite	Composite	0.51

Algorithm	Input (N)	Primality	Output	Time (ms)
AKS	13	Prime	Prime	0.54
	127	Prime	Prime	159.72
	209	Composite	Composite	0.59
	561 (Chamichael)	Composite	Composite	0.76
	2027	Prime	Prime	8369.16
	41041 (Chamichael)	Composite	Composite	1.36
	15,485,863	Prime	...	...
	987,654,321	Composite	Composite	4.98

Algorithm	Input (N)	Primality	Output	Time (ms)
Miller-Rabin	13	Prime	Prime	0.50
	127	Prime	Prime	0.44
	209	Composite	Composite	0.46
	561 (Chamichael)	Composite	Composite	0.46
	2027	Prime	Prime	0.51
($k = 5$)	41041 (Chamichael)	Composite	Composite	0.53
	15,485,863	Prime	Prime	0.54
	987,654,321	Composite	Composite	0.47

As we can see, the Miller-Rabin algorithm significantly outperforms the AKS algorithm, especially when identifying large prime numbers (at least 10 times faster). Finally, while the algorithm is only slightly slower that Fermat's test, the algorithm manages to achieve 100% accuracy, while Fermat's test fails completely against Camichel numbers. Hence, Miller-Rabin manages to balance both accuracy and efficiency, and its efficiency is still among the top.

6.3 Applications of Miller-Rabin

Primality testing has direct applications in modern scientific and technological domains that can address our real-world issues, like security and cryptography. For example, the Miller-Rabin algorithm is employed in cryptographic systems that verify the primality of large numbers, and it is also used to cryptographic keys for encryption scheme and digital signatures such as RSA, Pallier [13] and elliptic curve-based cryptosystems. They are also used for designing hash functions to map data into a value and store the hashed value in a hash table (for example, $h(x) \equiv ax + b \pmod{p}$ where a, b, p are given parameters). The modulus p is preferred to be a prime number to avoid collision, and thus primality test can be used for generating the random prime number for hashing. Finally, primality testing also has application in *linear congruential generator* (LCG), an algorithm used for generating pseudorandom numbers. To see why, recall that the i-th state s_i of a LCG is given by the recurrence $s_i \equiv as_{i-1} + b \pmod{p}$, where the modulus p can be a prime number. In this case, again, we require primality testing for generating a large prime number suitable for the LCG.

6.4 Case Study: RSA Cryptosystem and Random Prime Number Generation

As we have seen above, primarily testing has numerous applications. To see how primality testing can be used in these application, we now delve into a specific example. For instance, let us consider the problem of **generating large prime numbers**. It is a crucial problem in cryptography. To see why, let us consider RSA [15], a well known public key cryptosystem. The cryptosystem is as follows:

1. Suppose Bob and Alice want to communicate securely. Alice would like to send a message M to Bob. Bob intitially chooses an integer e and two prime numbers p, q such that $\text{GCD}((p-1)(q-1), e) = 1$ and compute $N = p \cdot q$. Bob then computes d to be the multiplicative inverse of e modulo $(p-1)(q-1)$. Finally, Bob publishes (e, N) and keeps p, q, d hidden.
2. Alice then computes $C \equiv M^e \pmod{N}$ and sends C to Bob.
3. Bob computes $M \equiv C^d \pmod{N}$. We see that $C^d \equiv M^{ed} \equiv 1 \pmod{N}$, which implies the correctness of the decryption process, since $ed \equiv 1 \pmod{\phi(N)}$

The security of RSA relies on the difficulty of factoring N into a product of prime powers, i.e, the prime factors p and q must be unknown. Thus, the bigger the primes p and q, the higher the security level of the RSA system. For RSA, it is recommended that the prime factors should be at least 1024 bits long. Not limited to RSA, large prime numbers are also used in various other cryptosystems such as ElGamal, Schnorr signature scheme. These cryptosystems require performing computations on elliptic curves over a finite field with prime characteristic. It is also recommended that the prime characteristic of the field for the elliptic curve should be 256 bits for cryptographic applications. To sum up, generating large prime numbers is a crucial problem in modern cryptography. However, a natural question is: *How can we generate such large prime numbers?* Fortunately, there is a well known method for generating an n-bit prime number, which can be found in various libraries for cryptography, such Python's *pycryptodome*. [1]. In this algorithm, suppose we are given the ISPRIME algorithm, that given an input x, output 1 if x is a prime number and 0 otherwise. The idea for generating an n-bit prime number in Python can be seen in Algorithm 5 below:

Algorithm 5 GENPRIME(n)

Input: n
Output: A prime x in $[2^{n-1}+1, 2^n-1]$

1. Sample $x \xleftarrow{\$} [2^{n-1}+1, 2^n-1]$.
2. **While** ISPRIME(x) $= 0$ do:
   (a) Sample $x \xleftarrow{\$} [2^{n-1}+1, 2^n-1]$.
3. **Return** x.

The expected runtime of GENPRIME(n) is equal to $2^{n-1}/(\pi(2^n)-\pi(2^{n-1})) = \Theta(n)$, where $\pi(k)$ denotes the number of prime numbers less than k, (this result comes from the *prime number theorem*, and we can prove that the probability that GENPRIME run within i steps is equal to $p(1-p)^{i-1}$, where $p = (\pi(2^n)-\pi(2^{n-1}))/2^{n-1} = 1/\Theta(n)$), making it an efficient prime generating algorithm. As we have seen above, in Step 2 of GENPRIME, we need to check whether x is a prime number using the ISPRIME algorithm and have to do so many times until x is confirmed to be a prime. Thus in this step, we need an algorithm for efficiently checking whether a large number is prime or not. Deterministic tests such as Atkin–Morain and AKS [1] are slow and impractical, and thus probabilistic tests such as Fermat, or Miller-Rabin are prefered. However, Fermat test always produces false positive result when tested against Camichel numbers, while Miller-Rabin test is slightly less efficient than Fermat test but is still efficient and provide high accuracy for all numbers. Hence, the Miller-Rabin algorithm can be applied as the ISPRIME algorithm Step 2 above due to its high accuracy and efficiency.

[1] https://github.com/Legrandin/pycryptodome/blob/master/lib/Crypto/Util/number.py

7 Discussion and Conclusion

7.1 Other Works

We provide a more detailed comparison of Miller-Rabin algorithm with other works. At a high level, primality test can be classified into two types of approaches: **deterministic approach** and **probabilistic approach**. We now compare Miller-Rabin test with other algorithms in these two categories as follows.

Deterministic Approach. The advantage of deterministic approach is its absolute correctness: It successfully verifies whether a number is a prime with probability 1. The simpliest method in this category is *trial division*: Gìven an integer n, simply check if n is divisible by any number from 2 to $\sqrt{n}$ and if yes then n is composite, otherwise n is prime. Although simple, the algorithm incurs $O(\sqrt{n})$ complexity, making it impractical for testing large numbers.

Fortunately, other deterministic tests, such as Atkin–Morain, Lucas-Lehmer and AKS achieve $O(\text{poly}(\log n))$ complexity. Among these, Lucas-Lehmer can only be used for testing Mersene prime numbers, and the running time of Atkin–Morain is heuristic, and thus it is not known to have polynomial time bounds for all inputs. Among deterministic algorithms, only the AKS algorithm is known to successfully verify all numbers and can be proved to terminate within $O(\text{poly}(\log n))$ complexity. However, deterministic algorithms are not widely used in practice due to their inferior performance to probabilistic algorithms. To see why, let us consider the running time of Miller-Rabin: It has $O(k \log^3 n)$ time complexity, and can be further optimized down to $O(k \log^2 n)$, while the best deterministic algorithm, AKS, require at least $O(\log^6 n)$ running time.

Probabilistic Approach. Probabilistic algorithms are usually more efficient than their deterministic counterpart, however, due to their probabilistic nature, they are known to provide false positive results. Fortunately, this false positive problem can be addressed by running the test many times and one will be convinced that the result is correct. The most simple algorithm in this category is the Fermat test. Its idea is simple, in each time, choose an integer a and check if $a^{n-1} \equiv 1 \pmod{n}$ and accept n is a prime if yes. Unfortunately, while the test is very efficient, there exist infinitely composite numbers that cause the test to provide false positive result with probability 1, known as Camichel numbers, making the test useless against these numbers.

Other well known tests in this category include Solovay–Strassen and Miller-Rabin test. Miller-Rabin test is also based on Fermat's little theorem but requires additional constraints which handles Camichel numbers, while Solovay–Strassen relies on Jacobi's symbol. Unlike Fermat test, no composite number could pass these tests with probability 1. They also have the same $O(k \log^3 n)$ time complexity as well. However, while Solovay–Strassen's error bound on each iteration is $1/2$, Miller-Rabin test manages to bound the error in each step down to $1/4$ (however, we do not prove this in our assignment but instead prove the $1/2$ version), and thus it provides more accuracy than Solovay–Strassen and is more widely used.

In conclusion, Miller-Rabin is a significant advancement in primality testing, suitable for practical applications. It manages to balance between efficiency and accuracy, hence achieving better efficiency than deterministic algorithms and better accuracy compared to other probabilistic algorithms, making it the most widely used algorithm. Ongoing research will further refine primality testing algorithms to meet evolving computational challenges.

7.2 Ethical implications of the Miller-Rabin algorithm

The Miller-Rabin algorithm, while pivotal in cryptography and computational mathematics, raises ethical concerns. Its probabilistic nature can cause the algorithm to provide false positive results, impacting privacy and security, hence strict measures must be followed to ensure that the accuracy of the algorithm is sufficient for cryptographic applications. For instance, the bases a need to be generated uniformly in $\mathbb{Z}_n$, and the value of k needs to be sufficiently large for security. If any of these measures are not strictly followed, for example, using an adversarial PRNG to generate a, or setting k to be small, then the algorithm, as well as the cryptographic system becomes insecure, leading to catastrophic results. Addressing these ethical considerations necessitates promoting inclusivity, transparency, and accountability in cryptographic system design and deployment. Thus, understanding the societal impact of the Miller-Rabin algorithm is crucial for upholding ethical principles in cryptographic technology.

7.3 Conclusion and Future Research Directions

The Miller-Rabin algorithm has undoubtedly made significant strides in probabilistic primality testing, providing a practical balance between computational efficiency and accuracy. However, its journey towards optimization and advancement is ongoing. Future research could delve into refining the algorithm's probabilistic tests to reduce false positives, particularly by exploring more sophisticated test criteria or optimizing witness selection for improved reliability across various number classes, or finding more applications related to the algorithms. Moreover, advancements in parallel computing and optimization techniques present promising avenues to enhance the algorithm's performance, enabling it to handle even larger numbers with greater efficiency on modern computing architectures. Furthermore, ongoing research in computational complexity theory and probabilistic algorithms may yield new variants or extensions of the Miller-Rabin algorithm with improved theoretical guarantees and practical performance such as [12, 6], or even use Miller-Rabin to generate RSA keys in a distributed manner [4]. Interdisciplinary collaborations among mathematicians, computer scientists, and cryptographers will remain instrumental in driving innovation in primality testing algorithms, leveraging diverse expertise to address current challenges and anticipate future needs in cryptography and computational mathematics. In conclusion, while the Miller-Rabin algorithm represents a significant milestone, its continuous evolution through research and collaboration holds the promise of further enhancing its accuracy, efficiency, and versatility, ensuring its continued relevance and impact in the dynamic landscape of computational mathematics and cryptography.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. *Annals of Mathematics*, 160:781–793, 2004.

[2] M. R. Albrecht, J. Massimo, K. G. Paterson, and J. Somorovsky. Prime and prejudice: Primality testing under adversarial conditions. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 281–298. ACM, 2018.

[3] E. Bach. Explicit bounds for primality testing and related problems. volume 55, pages 355–380, 1990.

[4] J. Burkhardt, I. Damgård, T. Frederiksen, S. Ghosh, and C. Orlandi. Improved distributed rsa key generation using the miller-rabin test. Cryptology ePrint Archive, Paper 2023/644, 2023. https://eprint.iacr.org/2023/644.

[5] J. Carl Pomerance; John L. Selfridge; Samuel S. Wagstaff. The pseudoprimes to 25.10. volume 35, page 1003–1026, 1980.

[6] L. Cherckesova, O. Safaryan, I. Trubchik, V. Chumakov, V. Yukhnov, and I. Yengibaryan. Modification and optimization of miller–rabin simplicity test algorithm implemented by parallel computation. In *IOP Conference Series: Materials Science and Engineering*, volume 1001, page 012064. IOP Publishing, 2020.

[7] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[8] J. Gallier and J. Quaintance. Notes on primality testing and public key cryptography. *Part I: Randomized Algorithms, Miller–Rabin and Solovay–Strassen Tests*, 277:278, 2019.

[9] D. Harvey and J. Van Der Hoeven. Integer multiplication in time o(nlog\,n). *Annals of Mathematics*, 193(2):563–617, 2021.

[10] G. Jaeschke. On strong pseudoprimes to several bases. volume 61, pages 915–926, 1993.

[11] G. L. Miller. Riemann's hypothesis and tests for primality. volume 13, pages 300–317, 1976.

[12] S. Narayanan. Improving the speed and accuracy of the miller-rabin primality test, 2014.

[13] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

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

[15] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

8 Appendix

We sketch the proof of Lemma 4, when the error rate is $1/4^k$ instead of $1/2^k$. We state without proof the following lemmas:

Lemma 5. Let p be an odd prime. Suppose there is an integer a such that $a^{2^r} \equiv -1 \pmod{p}$ then $p \equiv 1 \pmod{2^{r+1}}$.

Lemma 6. Let p be an odd prime and d be a positive integer. Then for all positive integer k, the number of a such that $a^d \equiv 1 \pmod{p^k}$ is at most $\text{GCD}(d, p-1)$. Similarly, the number of a such that $a^d \equiv -1 \pmod{p^k}$ is at most $\text{GCD}(d, p-1)$.

Proof Sketch of Lemma 4. The proof proceeds in the following steps

1. Let $S = \{a \in \mathbb{Z}_m \mid (a^d \equiv 1 \pmod{n}) \vee (\exists\, 0 \le r \le s, a^{2^r d} \equiv -1 \pmod{n})\}$.
2. Let $S_i = \{a \in \mathbb{Z}_m \mid (a^{2^i d} \equiv -1 \pmod{n})\}$ and $T = \{a \in \mathbb{Z}_m \mid (a^d \equiv 1 \pmod{n})\}$. Then $|S| = |T| + |S_0| + |S_1| + \cdots + |S_{r-1}|$ where r is the largest integer s.t $|S_{r-1}| > 0$. Then $p \equiv 1 \pmod{2^r}$ for all primes p divides n due to Lemma 5.
3. Let $A = \prod_{p|n, p \text{ is prime}} \text{GCD}(d, p-1)$. Then by Lemma 6, Lemma 6 and CRT, we can prove that $|S_i| \le 2^{ki} A$ since d is odd and $p-1$ is divisible by 2^i.
4. Hence $S/\phi(n) = A/\phi(n)(1 + \sum_{i=0}^{r-1} 2^{ik})$. Let $N = \prod_{p|n, p \text{ is prime}} p$ We can prove that $A \le \phi(N)/2^{rk}$ since $p-1$ is divisible by 2^r for all p divides n. Thus
$$\frac{|S|}{\phi(n)} \le \frac{\phi(N)}{\phi(n)} \left(\frac{1}{2^{rk}} \left(1 - \frac{1}{2^k - 1} \right) + \frac{1}{2^k - 1} \right)$$
5. Denote LHS and RHS be the left hand side and right hand side of the above equation. If $k \ge 3$ then RHS $\le 1/8(1 - 1/8) + 1/7 = 1/4$, as desired.
6. If $k = 2$ and $n \ne N$ (i.e, n is not a squarefree number) then $\phi(N)/\phi(n) \le 1/(2^r + 1)$ since every p divides n satisfied $p-1$ is divisible by 2^r. Hence RHS $\le 1/3(1/4(1-1/3)+1/3) \le 1/6$, as desired.
7. If $k = 1$ and $n \ne N$ then we consider $n = p^k$, and $\phi(N)/\phi(n) = 1/p^{k-1}$, which is less than $1/p$ or $1/3^2$, hence RHS$\le 1/4$ as desired.
8. If $k = 2$ and $n = pq$ where $p < q$. Let $p - 1 = 2^{a_1} b_1, q - 1 = 2^{a_2} b_2$, where b_1, b_2 is odd. The case $d \equiv 0 \pmod{b_i}$ for all i happen iff $b_1 = b_2 = b$. If $d \not\equiv 0 \pmod{b_i}$ for some i then $A \le \phi(N)/3 \cdot 2^{rk}$ and thus LHS $\le 1/3(1/4(1 - 1/3) + 1/3) = 1/6$. Otherwise, in this case we must have a_1 or a_2 is bigger than $r + 1$. In this case $A \le \phi(N)/2^{rk+1}$, thus LHS $\le 1/2(1/4(1 - 1/3) + 1/3) = 1/4$.
9. Thus, we are done.

□

THIS IS THE END OF OUR REPORT