MILLER - RABIN PRIMALITY TEST

Project Group 02

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

—

Under the Guidance of
Dr. Nguyen An Khuong

Dept. of Computer Science and Engineering
University of Technology, VNU - HCMC

May 24, 2024

Overview

Introduction to Primality Test

- **What are prime numbers?**
  - Prime numbers, divisible only by 1 and themselves, are fundamental in mathematics, with widespread importance in fields like cryptography and computer science.
- **What is primality test and why is it important?**
  - **Primality test problem:** *Given a positive integer n, determine whether n is a prime number.*
  - Primality tests are used to generate prime numbers, which are essential for cryptographic schemes such as RSA, Pallier.
- **History of primality tests**
  - Ancient Greeks and Egyptians: Trial Division and Sieve of Eratosthenes.
  - Deterministic algorithms like the Lucas-Lehmer and AKS are not always practical for large prime numbers due to their high cost.
  - Fermat test is efficient but completely fails against certain types of composite numbers such as Camichel numbers.
- **Miller - Rabin primality test**
  - 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

- Stated by Fermat in 1640 but apparently no proof was published at the time.
- The first known proof was given by Leibnitz

Fermat's Little Theorem

Let p be a prime number. If a is an integer coprime to p then $a^{p-1} \equiv 1 \pmod{p}$.

Mathematical Background

Computing Exponents

We would like to compute $a^b \pmod{n}$. A naive approach would be to multiply a by itself b times $\rightarrow$ Requires b multiplications.

There is another method to reduce to $O(\log b)$ multiplications.

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

Time Complexity: $O(\log b \log^2 n)$ (or $O(\log b \log n \log\log n)$ if integer multiplication is performed by FFT).

Miller-Rabin Algorithm

Idea of the algorithm

- Sample any a coprime to n.
- If $a^{n-1} \not\equiv 1 \pmod{n}$ then n is a composite (Fermat's little theorem!).
- So assume $a^{n-1} \equiv 1 \pmod{n}$, we write $n - 1 = 2^s \cdot d$ where d is odd. In this case

$$a^{n-1} - 1 = (a^d - 1)(a^d + 1)(a^{2d} + 1)(a^{4d} + 1) \dots (a^{2^{s-1}d} + 1).$$

- If n is a prime then n must divide $a^d - 1$ or $a^{2^r d} + 1$ for some $0 \leq r \leq s - 1$.
- So we perform a loop to find the **smallest** r such that $a^{2^r d} - 1$ is divisible by n. Then $r = 0$ or $a^{2^{r-1} d} + 1$ is divisible by n.
- If none holds then n is a composite, otherwise, n could possibility a prime.
- Repeat the above process k times.

Miller-Rabin Algorithm

Description and steps

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:
   1 - Compute $y \equiv x^2 \pmod{n}$.
   2 - **If** $y \equiv 1 \pmod{n}$ and $x \not\equiv \pm 1 \mod n$ then **return** 0.
   3 - Compute $x = y$.
4. **If** $x \not\equiv 1 \pmod{n}$ **then return** 0.
5. **Return** 1.

Miller-Rabin Algorithm

Description and steps

MILLER-RABIN(n, k)

Input: n, k
Output: 0/1

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

Miller-Rabin Algorithm

Flowchart of Witness algorithm

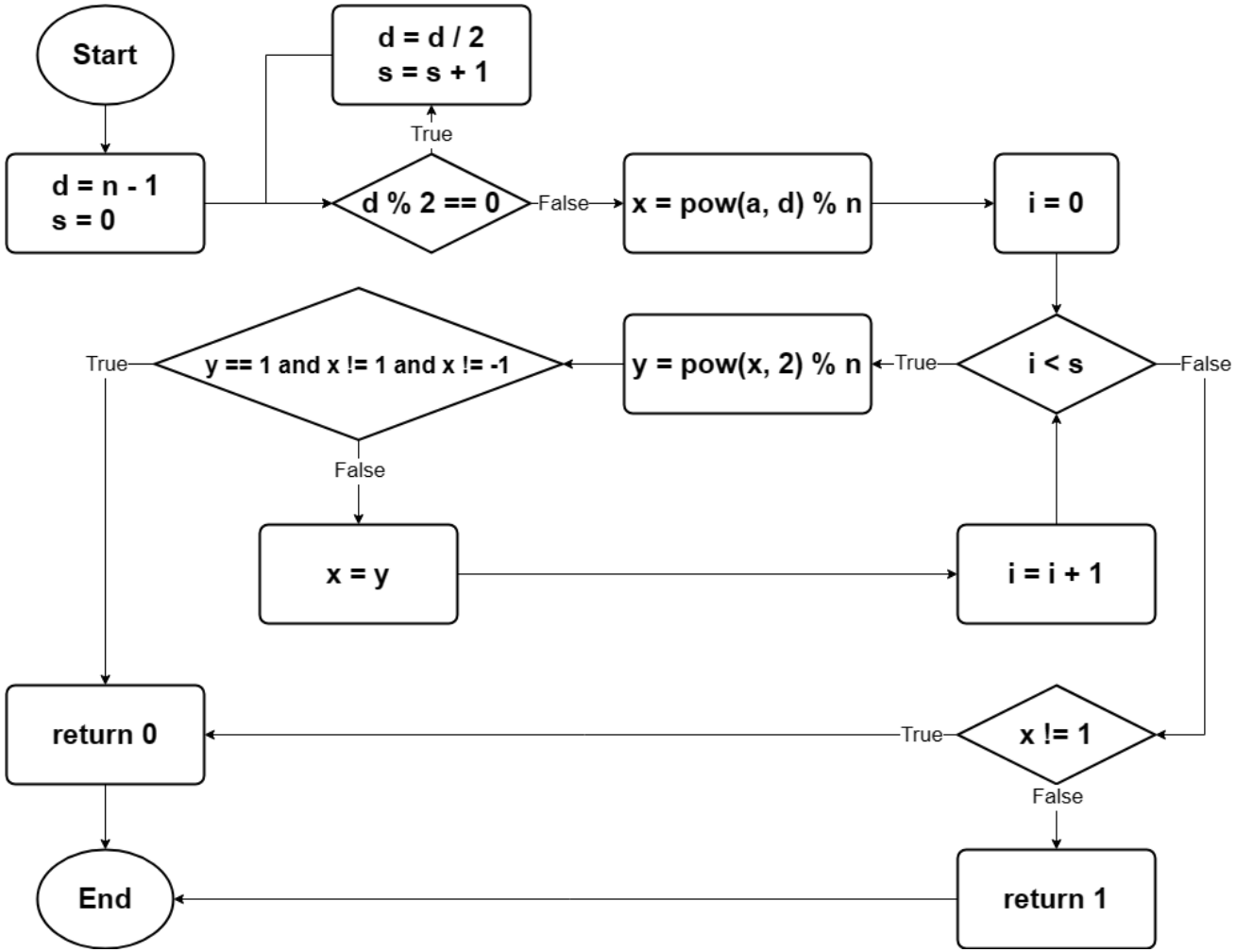

Figure: Flowchart of Witness Algorithm

Miller-Rabin Algorithm

Flowchart of Miller-Rabin algorithm

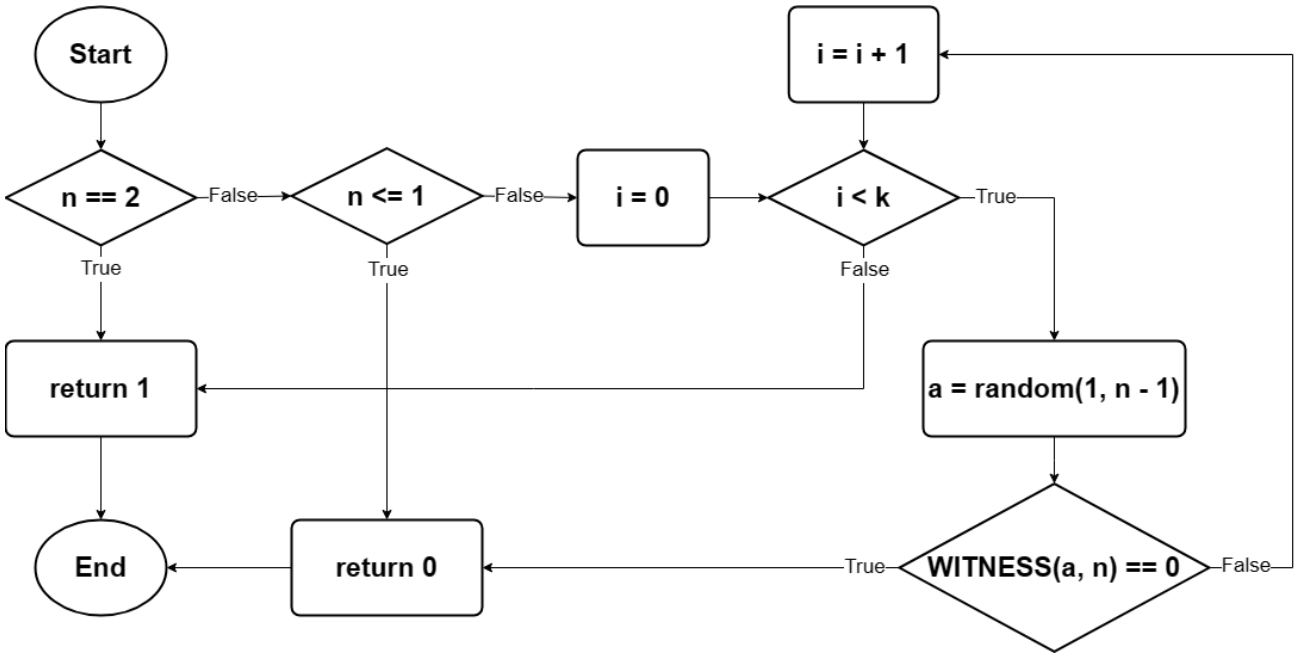

Figure: Flowchart of Miller-Rabin Algorithm

Miller-Rabin Algorithm

Paradigm

- It is a *randomized design paradigm*, because in each iteration, it samples a random value $a \in \{2, 3, \ldots, n-2\}$.
- It is a Monte Carlo algorithm, because it always **terminate after at most k iterations** and **may return a false positive** (i.e., declare a composite number as prime) with positive probability.
- However, the error probability can be reduced by increasing the number of iterations. The randomized algorithm design paradigm makes it more efficient than its deterministic counterparts, such as AKS.

Miller-Rabin Algorithm

Toy example

Toy Example: Test if $n = 37$ is prime. For simplicity, we consider $k = 2$.
Step 1: The number $n = 37$ pass the base cases
Step 2: We find that

$$n - 1 = 36 = 2^2 \times 9,$$

so we have $d = 9$ and $s = 2$.

Run 1 (k = 1)

Step 3: Pick a random number $a = 2$.
Step 4: Compute $x = 2^9 \pmod{37} = 31$.
Step 5 ($s = 0$): Compute $y = x^2 \pmod{37} = 36$.
Step 6 ($s = 0$): Since $y = 36 \neq 1$, we continue.
Step 7 ($s = 0$): Compute $x = y = 36$.
Step 8 ($s = 1$): Compute $y = x^2 \pmod{37} = 1$.
Step 9 ($s = 1$): Since $y = 1$ but $x = 36 = n - 1$, we continue.
Step 10 ($s = 1$): Compute $x = y = 1$.
Step 11: Since $x = 1$, the algorithm WITNESS(a, n) returns 1.

Run 2 (k = 2)

Step 12: Pick a random number $a = 3$.
Step 13: Compute $x = 3^9 \pmod{37} = 36$.
Step 14 ($s = 0$): Compute $y = x^2 \pmod{37} = 1$.
Step 15 ($s = 0$): Since $y = 1$ but $x = 36 = n - 1$, we continue.
Step 16 ($s = 0$): Compute $x = y = 1$.
Step 17 ($s = 1$): Compute $y = x^2 \pmod{37} = 1$.
Step 18 ($s = 1$): Since $y = 1$ but $x = 1$, we continue.
Step 19 ($s = 1$): Compute $x = y = 1$.
Step 20: Since $x = 1$, the algorithm WITNESS(a, n) returns 1.

After $k = 2$ iterations, MILLER-RABIN now returns 1, hence 37 is a prime number.

Early Termination

We will check the following conditions in WITNESS and terminate immediately:

- $a^d \equiv 1 \pmod{n}$ in Line 2. If yes return 1.
- $a^{2^l d} \equiv -1 \pmod{n}$ in Line 3.2. If yes return 1 otherwise return 0.

Parallel Computing

The standard Miller-Rabin test can be optimized using parallel computation:

1. Decompose the number $n-1$ into $2^s \cdot d$, where d is odd.
2. Perform multiple rounds of the test in parallel using multiple threads. Each thread picks a random number a and executes WITNESS(a, n).
3. Return 1 iff all threads return 1.

Deterministic variants

Assuming the truth of the generalized Riemann hypothesis (GRH), we just need to check $a \in [2, O(\log^2 n)]$. This provides 100% accuracy.

MILLER-RABIN(n, k)

Input: n, k
Output: 0/1

1. **If** n is even and $n \neq 2$ 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.

Complexity Analysis and Proof of Correctness

Time complexity

We consider WITNESS algorithm:

- To compute $x \equiv a^d \pmod{n}$ using repeated squaring incurs $O(\log d \log^2 n) = O(\log^3 n)$.
- In the for loop of WITNESS, we have $s = O(\log n)$ iterations. Each involves computing $y = x^2 \pmod{n}$ which incurs $O(\log^2 n)$ bit operations. So this makes $O(\log^3 n)$ (or $O(\log^2 n \log\log n)$ if FFT is used for integer multiplication).

Since there are k iterations for executing WITNESS, the time complexity will be multiplied by k.

Conclusion: Overall time complexity of MILLER-RABIN is $O(k \cdot \log^3 n)$.

Complexity Analysis and Proof of Correctness

Space complexity

It is determined by the number of bits required to store the input and any intermediate variables.

- The space required to store the input number (n) is $O(\log n)$, assuming a binary representation.
- The space required to store intermediate variables is also $O(\log n)$.

Conclusion: Overall space complexity of MILLER-RABIN is $O(\log n)$.

Lemma 1 (Correctness)

If n is a prime, then WITNESS returns 1 with probability 1.

Lemma 2 (Error Rate of WITNESS)

If n is a composite number, then WITNESS returns 1 with probability $1/2$.

In MILLER-RABIN, we execute WITNESS k times and return 1 iff WITNESS return 1 all k times. Since the error rate of WITNESS is $1/2$, the error rate of MILLER-RABIN is $1/2^k$.

Remark: It is possible to improve the bound from $1/2^k$ to $1/4^k$.

We sketch the proof of Lemma 1. The proof of Lemma 2 is complicated, hence its proof is moved to appendix.

Proof sketch:

- Consider the WITNESS algorithm and n is prime. We prove that WITNESS(a, n) always returns 1 for all a not divisible by n.
- If $a^d \equiv 1 \pmod{n}$ then in the **for** loop the values x and y are always equal to 1 $\rightarrow$ WITNESS(a, n) returns 1.
- Otherwise, let r be the **smallest** integer s.t $a^{2^r d} \equiv 1 \pmod{n} \rightarrow r$ is positive. Note that in the i-th iteration, $y \equiv a^{2^i d} \pmod{n}$.
  - In iterations 1 to r, $y \not\equiv 1 \pmod{n} \rightarrow$ the check in Step 2.2 is passed.
  - In the r-th iteration, $y \equiv a^{2^r d} \equiv 1 \pmod{n}$ and $x \equiv a^{2^{r-1} d} \equiv -1 \pmod{n}$ (since n is a prime) $\rightarrow$ the check in Step 2.2 is passed.
  - In iterations $r+1$ to s $x \equiv y \equiv 1 \pmod{n} \rightarrow$ the check in Step 2.2 is passed.
- Hence the check in Step 2.2 is always passed and WITNESS returns 1 $\rightarrow$ MILLER-RABIN returns 1 as well.

- **Strong pseudoprime:** Composite n such that WITNESS$(a, n) = 1$ for certain a. This n satisfies: $n \mid a^d - 1$ or $n \mid a^{2^r d} + 1$ for some $0 \leq r \leq s - 1$. If all k times, this a is chosen, then MILLER-RABIN to return $1 \rightarrow n$ is falsely identified as prime. Fortunately, it is proved that such probability is at most $1/2^k$ for all n.
- k **is small:** Since the error probability depends on $k \rightarrow$ The smaller the value k, the bigger the error probability. Having $k = 64$ is suitable for cryptographic applications.
- **Fixed choice of bases:** The value a is chosen from a fixed set, for example, the set of primes less than 1000. Unfortunately, given a set S, it is possible to choose a composite n such that WITNESS$(a, n) = 1$ for all $a \in S \rightarrow$ MILLER-RABIN falsely identifies n as a prime with probability 1. This should be avoided.

Implementation and Benchmark

Implementation

- We provide an implementation of the Miller-Rabin Algorithm in Python.
- The source code of our implementation is available at: `https://github.com/redevil24/Advanced-Algorithm-Assignment`
- We use the early-termination implementation of Miller-Rabin for benchmarking.
- We can use bitwise operations in some situations. For example, when checking if $n \equiv 1 \pmod 2$, we can instead check that $n\&1 = 1$.

Implementation and Benchmark

Benchmark

We compare the result of our Miller-Rabin implementation with Fermat and AKS algorithm.

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

Implementation and Benchmark

Benchmark

Algorithm	Input (N)	Primality	Output	Time (ms)
Miller-Rabin ($k = 5$)	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
	41041 (Chamichael)	Composite	Composite	0.53
	15,485,863	Prime	Prime	0.54
	987,654,321	Composite	Composite	0.47

$\rightarrow$ Miller-Rabin algorithm significantly outperforms the AKS algorithm on running time, defeats Fermat primality test in term of accuracy.

Applications and Case Study

Applications

- Primality testing can be used for generating **large prime numbers**.
- Large prime numbers are used in many cryptosystems: RSA, ECDSA, ElGamal, Pallier and many more.
- They are also used for designing hash functions to hash data and store them in a hash table: $h(x) \equiv a \cdot x + b \pmod{p}$.
- They are also used for pseudo-random number generation. For example: Least Congruential Generator (LCG), where the i-th state s_i of the LCG is given by the recurrence $s_{i+1} \equiv a \cdot s_i + b \pmod{p}$.

RSA Cryptosystem

1. Bob chooses an integer e and two prime numbers p, q such that $\text{GCD}((p-1)(q-1), e) = 1$. He computes $N = p \cdot q$ and d such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$. 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 $M \equiv C^d \pmod{N}$.

The security of RSA lies in the assumption that p, q must be unknown. Since the complexity of the factorization algorithm depends on the size of $N \rightarrow$ The bigger p, q, the more secure the cryptosystem becomes.

It is recommended that the generated prime factors for RSA should be at least 1024 bits.

Question: How can we generate such large prime numbers?

GENPRIME(n)

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

The **expected number** of iterations in GENPRIME is $\Theta(n)$.

We need to efficiently check whether x is a prime number using the ISPRIME algorithm and **have to do so many times** $\rightarrow$ This is where primality test is needed.

AKS is slow and impractical, while Fermat test always produces false positive result when tested against Camichel numbers.

Miller-Rabin algorithm can be applied as the ISPRIME algorithm Step 2 above due to its high accuracy and efficiency.

Discussion and Conclusion

Miller Rabin vs Other Works

- **Deterministic Approach**
  - Trial Division: $O(\sqrt{n})$ complexity.
  - Atkin–Morain: Running time is heuristic.
  - Lucas-Lehmer: Can only test Mersene prime numbers.
  - AKS: Best deterministic algorithm, require $\tilde{O}(\log^6 n)$ running time.
- **Probabilistic Approach**
  - Fermat test: Fails against **Camichel** numbers.
  - Solovay–Strassen: Error bound on each iteration is $1/2$ instead of $1/4$.

Discussion and Conclusion

Ethical implications of the Miller-Rabin algorithm

- The Miller-Rabin algorithm, while pivotal in cryptography and computational mathematics, raises ethical concerns as it may provide false positive results, impacting privacy and security when misused (k is set too small, a is generated by an adversarial PRNG, or is sampled in a fixed set).
- Such misuse can cause the algorithm to falsely identify composites as a prime number $\rightarrow$ The cryptosystem employing the algorithm becomes insecure.
- Thus, all steps of the Miller-Rabin algorithm must be strictly followed and this is crucial for upholding ethical principles in cryptographic technology.

Discussion and Conclusion

Conclusion & Future Research Directions

In this assignmnent, we have:

- Understand and analyze in detail the advanced aspects of Miller-Rabin algorithm.
- Understand the applications of Miller-Rabin algorithm, especially in generating large prime numbers, which is essential for cryptographic schemes such as RSA.

Future research could delve into:

- Refining the algorithm's probabilistic tests to reduce false positive.
- Optimizing witness selection.
- Finding more applications related to the algorithms.
- Parallel computing and optimization techniques to enhance the algorithm's performance.
- New variants or extensions of the Miller-Rabin algorithm with improved theoretical guarantees and practical performance .

Thanks for your listening !

Chinese Remainder Theorem (CRT)

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:

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

Definition (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}.

Definition (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.

Lemma 3

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 $|\mathbb{G}|$.

Appendix (Proof of Correctness)

Proof of Lemma 2

Proof sketch:

- Say a is a witness of n iff WITNESS$(a, n) = 0$. Need to prove that the number of non-witness of n does not exceed $n/2$.
- Idea: Prove that the set of non-witnesses is contained in a **proper** subgroup S of $\mathbb{Z}_n^\star$. Then by Lemma 3, $|S| \leq n/2$.
- First, if a is a non-witness of n then $a \in \mathbb{Z}_n^\star$.
- If there is some $a \in \mathbb{Z}_n^\star$ such that $a^{n-1} \not\equiv 1 \pmod{n}$. Choose $S = \{x \in \mathbb{Z}_n^\star \mid x^{n-1} \equiv 1 \pmod{n}\}$ and verify that S is a group.
- If $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in\in \mathbb{Z}_n^\star$ then:
  - Prove that n is not a prime power.
  - Let r be the smallest positive such that $a^{2^r d} \equiv 1 \pmod{n}$ for all $a \in\in \mathbb{Z}_n^\star$, note that r is positive. Choose $S = \{x \in \mathbb{Z}_n^\star \mid x^{2^{r-1} d} \equiv \pm 1 \pmod{n}\}$.
  - Let $n = n_1 n_2 \ldots, n_k$ where n_i be prime powers. Note that there exists a' such that $(a')^{2^{r-1} d} \not\equiv 1 \pmod{n}$. Prove that there is some i such that $(a')^{2^{r-1} d} \equiv -1 \pmod{n_i}$.
  - Choose a such that $a \equiv a' \pmod{n_i}$ and $a \equiv 1 \pmod{n/n_i}$ by CRT.
  - We see that $a \in \mathbb{Z}_n^\star$ but $a \notin S$ since both n_i and n/n_i are greater than 1.
- In either way, S is a proper subgroup of $n \rightarrow$ Done.