Is n a Prime Number?

Manindra Agrawal

IIT Kanpur

March 27, 2006, Delft

Overview

Outline

The Problem

Given a number n, decide if it is prime.

Easy: try dividing by all numbers less than n.

The Problem

Given a number n, decide if it is prime.

Easy: try dividing by all numbers less than n.

The Problem

Given a number n, decide if it is prime efficiently.

Not so easy: several non-obvious methods have been found.

The Problem

Given a number n, decide if it is prime efficiently.

Not so easy: several non-obvious methods have been found.

Efficiently Solving a Problem

- There should exist an algorithm for solving the problem taking a polynomial in input size number of steps.
- For our problem, this means an algorithm taking $\log^{O(1)} n$ steps.

Caveat: An algorithm taking $\log^{12} n$ steps would be slower than an algorithm taking $\log^{\log\log\log n} n$ steps for all practical values of n!

Notation:

- $\log$ is logarithm base 2.
- $O\tilde{}(\log^c n)$ stands for $O(\log^c n \log\log^{O(1)} n)$.

Efficiently Solving a Problem

- There should exist an algorithm for solving the problem taking a polynomial in input size number of steps.
- For our problem, this means an algorithm taking $\log^{O(1)} n$ steps.

Caveat: An algorithm taking $\log^{12} n$ steps would be slower than an algorithm taking $\log^{\log\log\log n} n$ steps for all practical values of n!

Notation:

- $\log$ is logarithm base 2.
- $\tilde{O}(\log^c n)$ stands for $O(\log^c n \log\log^{O(1)} n)$.

Efficiently Solving a Problem

- There should exist an algorithm for solving the problem taking a polynomial in input size number of steps.
- For our problem, this means an algorithm taking $\log^{O(1)} n$ steps.

CAVEAT: An algorithm taking $\log^{12} n$ steps would be slower than an algorithm taking $\log^{\log\log\log n} n$ steps for all practical values of n!

NOTATION:

- $\log$ is logarithm base 2.
- $O\tilde{}(\log^c n)$ stands for $O(\log^c n \log\log^{O(1)} n)$.

Efficiently Solving a Problem

- There should exist an algorithm for solving the problem taking a polynomial in input size number of steps.
- For our problem, this means an algorithm taking $\log^{O(1)} n$ steps.

CAVEAT: An algorithm taking $\log^{12} n$ steps would be slower than an algorithm taking $\log^{\log\log\log n} n$ steps for all practical values of n!

NOTATION:

- $\log$ is logarithm base 2.
- $O\tilde{~}(\log^c n)$ stands for $O(\log^c n \log\log^{O(1)} n)$.

Outline

The Sieve of Eratosthenes

Proposed by Eratosthenes (ca. 300 BCE).

1. List all numbers from 2 to n in a sequence.
2. Take the smallest uncrossed number from the sequence and cross out all its multiples.
3. If n is uncrossed when the smallest uncrossed number is greater than $\sqrt{n}$ then n is prime otherwise composite.

Time Complexity

- If n is prime, algorithm crosses out all the first $\sqrt{n}$ numbers before giving the answer.
- So the number of steps needed is $\Omega(\sqrt{n})$.

Time Complexity

- If n is prime, algorithm crosses out all the first $\sqrt{n}$ numbers before giving the answer.
- So the number of steps needed is $\Omega(\sqrt{n})$.

Wilson's Theorem

Based on Wilson's theorem (1770).

Theorem

n is prime iff $(n-1)! = -1 \pmod{n}$.

- Computing $(n-1)! \pmod{n}$ naïvely requires $\Omega(n)$ steps.
- No significantly better method is known!

Wilson's Theorem

Based on Wilson's theorem (1770).

Theorem

n is prime iff $(n-1)! = -1 \ (mod\ n)$.

- Computing $(n-1)! \ (mod\ n)$ naïvely requires $\Omega(n)$ steps.
- No significantly better method is known!

Outline

Fundamental Idea

Nearly all the efficient algorithms for the problem use the following idea.

- Identify a finite group G related to number n.
- Design an efficienty testable property $P(\cdot)$ of the elements G such that $P(e)$ has different values depending on whether n is prime.
- The element e is either from a small set (in deterministic algorithms) or a random element of G (in randomized algorithms).

Fundamental Idea

Nearly all the efficient algorithms for the problem use the following idea.

- Identify a finite group G related to number n.
- Design an efficienty testable property $P(\cdot)$ of the elements G such that $P(e)$ has different values depending on whether n is prime.
- The element e is either from a small set (in deterministic algorithms) or a random element of G (in randomized algorithms).

Fundamental Idea

Nearly all the efficient algorithms for the problem use the following idea.

- Identify a finite group G related to number n.
- Design an efficienty testable property $P(\cdot)$ of the elements G such that $P(e)$ has different values depending on whether n is prime.
- The element e is either from a small set (in deterministic algorithms) or a random element of G (in randomized algorithms).

Fundamental Idea

Nearly all the efficient algorithms for the problem use the following idea.

- Identify a finite group G related to number n.
- Design an efficienty testable property $P(\cdot)$ of the elements G such that $P(e)$ has different values depending on whether n is prime.
- The element e is either from a small set (in deterministic algorithms) or a random element of G (in randomized algorithms).

Groups and Properties

- The group G is often:
  - A subgroup of Z_n^* or $Z_n^*[\zeta]$ for an extension ring $Z_n[\zeta]$.
  - A subgroup of $E(Z_n)$, the set of points on an elliptic curve modulo n.
- The properties vary, but are from two broad themes.

Groups and Properties

- The group G is often:
  - A subgroup of Z_n^* or $Z_n^*[\zeta]$ for an extension ring $Z_n[\zeta]$.
  - A subgroup of $E(Z_n)$, the set of points on an elliptic curve modulo n.
- The properties vary, but are from two broad themes.

Groups and Properties

- The group G is often:
  - A subgroup of Z_n^* or $Z_n^*[\zeta]$ for an extension ring $Z_n[\zeta]$.
  - A subgroup of $E(Z_n)$, the set of points on an elliptic curve modulo n.
- The properties vary, but are from two broad themes.

Theme I: Factorization of Group Size

- Compute a complete, or partial, factorization of the size of G assuming that n is prime.
- Use the knowledge of this factorization to design a suitable property.

Theme I: Factorization of Group Size

- Compute a complete, or partial, factorization of the size of G assuming that n is prime.
- Use the knowledge of this factorization to design a suitable property.

Theme II: Fermat's Little Theorem

Theorem (Fermat, 1660s)

If n is prime then for every e, $e^n = e \pmod{n}$.

- Group $G = Z_n^*$ and property P is $P(e) \equiv e^n = e$ in G.
- This property of Z_n is not a sufficient test for primality of n.
- So try to extend this property to a neccessary and sufficient condition.

Theme II: Fermat's Little Theorem

Theorem (Fermat, 1660s)

If n is prime then for every e, $e^n = e \ (mod\ n)$.

- Group $G = Z_n^*$ and property P is $P(e) \equiv e^n = e$ in G.
- This property of Z_n is not a sufficient test for primality of n.
- So try to extend this property to a neccessary and sufficient condition.

Outline

1. The Problem
2. Two Simple, and Slow, Methods
3. Modern Methods
4. **Algorithms Based on Factorization of Group Size**
5. Algorithms Based on Fermat's Little Theorem
6. An Algorithm Outside the Two Themes

Lucas Theorem

Theorem (E. Lucas, 1891)

Let $n - 1 = \prod_{i=1}^{t} p_i^{d_i}$ where p_i's are distinct primes. n is prime iff there is an $e \in Z_n$ such that $e^{n-1} = 1$ and $\gcd(e^{\frac{n-1}{p_i}} - 1, n) = 1$ for every $1 \leq i \leq t$.

- The theorem also holds for a random choice of e.
- We can choose $G = Z_n^*$ and P to be the property above.
- The test will be efficient only for numbers n such that $n - 1$ is smooth.

Lucas Theorem

Theorem (E. Lucas, 1891)

Let $n - 1 = \prod_{i=1}^{t} p_i^{d_i}$ where p_i's are distinct primes. n is prime iff there is an $e \in Z_n$ such that $e^{n-1} = 1$ and $\gcd(e^{\frac{n-1}{p_i}} - 1, n) = 1$ for every $1 \leq i \leq t$.

- The theorem also holds for a random choice of e.
- We can choose $G = Z_n^*$ and P to be the property above.
- The test will be efficient only for numbers n such that $n - 1$ is smooth.

Lucas Theorem

Theorem (E. Lucas, 1891)

Let $n - 1 = \prod_{i=1}^{t} p_i^{d_i}$ where p_i's are distinct primes. n is prime iff there is an $e \in Z_n$ such that $e^{n-1} = 1$ and $\gcd(e^{\frac{n-1}{p_i}} - 1, n) = 1$ for every $1 \leq i \leq t$.

- The theorem also holds for a random choice of e.
- We can choose $G = Z_n^*$ and P to be the property above.
- The test will be efficient only for numbers n such that $n - 1$ is smooth.

Lucas-Lehmer Test

- G is a subgroup of $Z_n^*[\sqrt{3}]$ containing elements of order $n+1$.
- The property P is: $P(e) \equiv e^{\frac{n+1}{2}} = -1$ in $Z_n[\sqrt{3}]$.
- Works only for special Mersenne primes of the form $n = 2^p - 1$, p prime.
- For such n's, $n + 1 = 2^p$.
- The property needs to be tested only for $e = 2 + \sqrt{3}$.

Lucas-Lehmer Test

- G is a subgroup of $Z_n^*[\sqrt{3}]$ containing elements of order $n+1$.
- The property P is: $P(e) \equiv e^{\frac{n+1}{2}} = -1$ in $Z_n[\sqrt{3}]$.
- Works only for special Mersenne primes of the form $n = 2^p - 1$, p prime.
- For such n's, $n + 1 = 2^p$.
- The property needs to be tested only for $e = 2 + \sqrt{3}$.

Lucas-Lehmer Test

- G is a subgroup of $Z_n^*[\sqrt{3}]$ containing elements of order $n+1$.
- The property P is: $P(e) \equiv e^{\frac{n+1}{2}} = -1$ in $Z_n[\sqrt{3}]$.
- Works only for special Mersenne primes of the form $n = 2^p - 1$, p prime.
- For such n's, $n + 1 = 2^p$.
- The property needs to be tested only for $e = 2 + \sqrt{3}$.

Time Complexity

- Raising $2+\sqrt{3}$ to $\frac{n+1}{2}$th power requires $O(\log n)$ multiplication operations in Z_n.
- Overall time complexity is $O\tilde{~}(\log^2 n)$.

Time Complexity

- Raising $2+\sqrt{3}$ to $\frac{n+1}{2}$th power requires $O(\log n)$ multiplication operations in Z_n.
- Overall time complexity is $O\tilde{~}(\log^2 n)$.

Pocklington-Lehmer Test

Theorem (Pocklington, 1914)

If there exists a e such that $e^{n-1} = 1 \pmod n$ and $\gcd(e^{\frac{n-1}{p_j}} - 1, n) = 1$ for distinct primes $p_1, p_2, \ldots, p_t$ dividing $n-1$ then every prime factor of n has the form $k \cdot \prod_{j=1}^{t} p_j + 1$.

- Similar to Lucas's theorem.
- Let $G = Z_n^*$ and property P precisely as in the theorem.
- The property is tested for a random e.
- For the test to work, we need $\prod_{j=1}^{t} \geq \sqrt{n}$.

Pocklington-Lehmer Test

Theorem (Pocklington, 1914)

If there exists a e such that $e^{n-1} = 1 \pmod{n}$ and $\gcd(e^{\frac{n-1}{p_j}} - 1, n) = 1$ for distinct primes $p_1, p_2, \ldots, p_t$ dividing $n-1$ then every prime factor of n has the form $k \cdot \prod_{j=1}^{t} p_j + 1$.

- Similar to Lucas's theorem.
- Let $G = Z_n^*$ and property P precisely as in the theorem.
- The property is tested for a random e.
- For the test to work, we need $\prod_{j=1}^{t} \geq \sqrt{n}$.

Pocklington-Lehmer Test

Theorem (Pocklington, 1914)

If there exists a e such that $e^{n-1} = 1 \pmod n$ and $\gcd(e^{\frac{n-1}{p_j}} - 1, n) = 1$ for distinct primes $p_1, p_2, \ldots, p_t$ dividing $n-1$ then every prime factor of n has the form $k \cdot \prod_{j=1}^{t} p_j + 1$.

- Similar to Lucas's theorem.
- Let $G = Z_n^*$ and property P precisely as in the theorem.
- The property is tested for a random e.
- For the test to work, we need $\prod_{j=1}^{t} \geq \sqrt{n}$.

Time Complexity

- Depends on the difficulty of finding prime factorizations of $n-1$ whose product is at least $\sqrt{n}$.
- Other operations can be carried out efficiently.

Time Complexity

- Depends on the difficulty of finding prime factorizations of $n - 1$ whose product is at least $\sqrt{n}$.
- Other operations can be carried out efficiently.

Elliptic Curves Based Tests

- Elliptic curves give rise to groups of different sizes associated with the given number.
- With good probability, some of these groups have sizes that can be easily factored.
- This motivated primality testing based on elliptic curves.

Elliptic Curves Based Tests

- Elliptic curves give rise to groups of different sizes associated with the given number.
- With good probability, some of these groups have sizes that can be easily factored.
- This motivated primality testing based on elliptic curves.

Elliptic Curves Based Tests

- Elliptic curves give rise to groups of different sizes associated with the given number.
- With good probability, some of these groups have sizes that can be easily factored.
- This motivated primality testing based on elliptic curves.

Goldwasser-Kilian Test

- This is a randomized primality proving algorithm.
- Under a reasonable hypothesis, it is polynomial time on all inputs.
- Unconditionally, it is polynomial time on all but negligible fraction of numbers.

Goldwasser-Kilian Test

- This is a randomized primality proving algorithm.
- Under a reasonable hypothesis, it is polynomial time on all inputs.
- Unconditionally, it is polynomial time on all but negligible fraction of numbers.

Goldwasser-Kilian Test

- This is a randomized primality proving algorithm.
- Under a reasonable hypothesis, it is polynomial time on all inputs.
- Unconditionally, it is polynomial time on all but negligible fraction of numbers.

Goldwasser-Kilian Test

- Consider a random elliptic curve over Z_n.
- By a theorem of Lenstra (1987), the number of points of the curve is nearly uniformly distributed in the interval $[n+1-2\sqrt{n}, n+1+2\sqrt{n}]$ for prime n.
- Assuming a conjecture about the density of primes in small intervals, it follows that there are curves with $2q$ points, for q prime, with reasonable probability.

Goldwasser-Kilian Test

- Consider a random elliptic curve over Z_n.
- By a theorem of Lenstra (1987), the number of points of the curve is nearly uniformly distributed in the interval $[n+1-2\sqrt{n}, n+1+2\sqrt{n}]$ for prime n.
- Assuming a conjecture about the density of primes in small intervals, it follows that there are curves with $2q$ points, for q prime, with reasonable probability.

Goldwasser-Kilian Test

- Consider a random elliptic curve over Z_n.
- By a theorem of Lenstra (1987), the number of points of the curve is nearly uniformly distributed in the interval $[n+1-2\sqrt{n}, n+1+2\sqrt{n}]$ for prime n.
- Assuming a conjecture about the density of primes in small intervals, it follows that there are curves with $2q$ points, for q prime, with reasonable probability.

Goldwasser-Kilian Test

Theorem (Goldwasser-Kilian)

Suppose $E(Z_n)$ is an elliptic curve with $2q$ points. If q is prime and there exists $A \in E(Z_n) \neq O$ such that $q \cdot A = O$ then either n is provably prime or provably composite.

Proof.

- Let p be a prime factor of n with $p \leq \sqrt{n}$.
- We have $q \cdot A = O$ in $E(Z_p)$ as well.
- If $A = O$ in $E(Z_p)$ then n can be factored.
- Otherwise, since q is prime, $|E(Z_p)| \geq q$.
- If $2q < n + 1 - 2\sqrt{n}$ then n must be composite.
- Otherwise, $p + 1 + 2\sqrt{p} > \frac{n}{2} - \sqrt{n}$ which is not possible.

□

Goldwasser-Kilian Test

Theorem (Goldwasser-Kilian)

Suppose $E(Z_n)$ is an elliptic curve with $2q$ points. If q is prime and there exists $A \in E(Z_n) \neq O$ such that $q \cdot A = O$ then either n is provably prime or provably composite.

Proof.

- Let p be a prime factor of n with $p \leq \sqrt{n}$.
- We have $q \cdot A = O$ in $E(Z_p)$ as well.
- If $A = O$ in $E(Z_p)$ then n can be factored.
- Otherwise, since q is prime, $|E(Z_p)| \geq q$.
- If $2q < n + 1 - 2\sqrt{n}$ then n must be composite.
- Otherwise, $p + 1 + 2\sqrt{p} > \frac{n}{2} - \sqrt{n}$ which is not possible.

□

Goldwasser-Kilian Test

Theorem (Goldwasser-Kilian)

Suppose $E(Z_n)$ is an elliptic curve with $2q$ points. If q is prime and there exists $A \in E(Z_n) \neq O$ such that $q \cdot A = O$ then either n is provably prime or provably composite.

Proof.

- Let p be a prime factor of n with $p \leq \sqrt{n}$.
- We have $q \cdot A = O$ in $E(Z_p)$ as well.
- If $A = O$ in $E(Z_p)$ then n can be factored.
- Otherwise, since q is prime, $|E(Z_p)| \geq q$.
- If $2q < n + 1 - 2\sqrt{n}$ then n must be composite.
- Otherwise, $p + 1 + 2\sqrt{p} > \frac{n}{2} - \sqrt{n}$ which is not possible.

□

Goldwasser-Kilian Test

Theorem (Goldwasser-Kilian)

Suppose $E(Z_n)$ is an elliptic curve with $2q$ points. If q is prime and there exists $A \in E(Z_n) \neq O$ such that $q \cdot A = O$ then either n is provably prime or provably composite.

Proof.

- Let p be a prime factor of n with $p \leq \sqrt{n}$.
- We have $q \cdot A = O$ in $E(Z_p)$ as well.
- If $A = O$ in $E(Z_p)$ then n can be factored.
- Otherwise, since q is prime, $|E(Z_p)| \geq q$.
- If $2q < n + 1 - 2\sqrt{n}$ then n must be composite.
- Otherwise, $p + 1 + 2\sqrt{p} > \frac{n}{2} - \sqrt{n}$ which is not possible.

□

Goldwasser-Kilian Test

1. Find a random elliptic curve over Z_n with $2q$ points.
2. Prove primality of q recursively.
3. Randomly select an A such that $q \cdot A = O$.
4. Infer n to be prime or composite.

Goldwasser-Kilian Test

1. Find a random elliptic curve over Z_n with $2q$ points.
2. Prove primality of q recursively.
3. Randomly select an A such that $q \cdot A = O$.
4. Infer n to be prime or composite.

Goldwasser-Kilian Test

1. Find a random elliptic curve over Z_n with $2q$ points.
2. Prove primality of q recursively.
3. Randomly select an A such that $q \cdot A = O$.
4. Infer n to be prime or composite.

Goldwasser-Kilian Test

1. Find a random elliptic curve over Z_n with $2q$ points.
2. Prove primality of q recursively.
3. Randomly select an A such that $q \cdot A = O$.
4. Infer n to be prime or composite.

Analysis

- The algorithm never incorrectly classifies a composite number.
- With high probability it correctly classifies prime numbers.
- The running time is $O(\log^{11} n)$.
- Improvements by Atkin and others result in a conjectured running time of $\tilde{O}(\log^4 n)$.

Analysis

- The algorithm never incorrectly classifies a composite number.
- With high probability it correctly classifies prime numbers.
- The running time is $O(\log^{11} n)$.
- Improvements by Atkin and others result in a conjectured running time of $\tilde{O}(\log^4 n)$.

Analysis

- The algorithm never incorrectly classifies a composite number.
- With high probability it correctly classifies prime numbers.
- The running time is $O(\log^{11} n)$.
- Improvements by Atkin and others result in a conjectured running time of $\tilde{O}(\log^4 n)$.

Analysis

- The algorithm never incorrectly classifies a composite number.
- With high probability it correctly classifies prime numbers.
- The running time is $O(\log^{11} n)$.
- Improvements by Atkin and others result in a conjectured running time of $\tilde{O}(\log^4 n)$.

Adleman-Huang Test

- The previous test is not unconditionally polynomial time on a small fraction of numbers.
- Adleman-Huang (1992) removed this drawback.
- They first used hyperelliptic curves to reduce the problem of testing for n to that of a nearly random integer of similar size.
- Then the previous test works with high probability.
- The time complexity becomes $O(\log^c n)$ for $c > 30$!

Adleman-Huang Test

- The previous test is not unconditionally polynomial time on a small fraction of numbers.
- Adleman-Huang (1992) removed this drawback.
- They first used hyperelliptic curves to reduce the problem of testing for n to that of a nearly random integer of similar size.
- Then the previous test works with high probability.
- The time complexity becomes $O(\log^c n)$ for $c > 30$!

Adleman-Huang Test

- The previous test is not unconditionally polynomial time on a small fraction of numbers.
- Adleman-Huang (1992) removed this drawback.
- They first used hyperelliptic curves to reduce the problem of testing for n to that of a nearly random integer of similar size.
- Then the previous test works with high probability.
- The time complexity becomes $O(\log^c n)$ for $c > 30$!

Outline

Solovay-Strassen Test

A Restatement of FLT

If n is odd prime then for every e, $1 \leq e < n$, $e^{\frac{n-1}{2}} = \pm 1 \ (mod\ n)$.

- When n is prime, e is a quadratic residue in Z_n iff $e^{\frac{n-1}{2}} = 1 \ (mod\ n)$.
- Therefore, if n is prime then

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$

Solovay-Strassen Test

A Restatement of FLT

If n is odd prime then for every e, $1 \leq e < n$, $e^{\frac{n-1}{2}} = \pm 1 \ (mod\ n)$.

- When n is prime, e is a quadratic residue in Z_n iff $e^{\frac{n-1}{2}} = 1 \ (mod\ n)$.
- Therefore, if n is prime then

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$

Solovay-Strassen Test

A Restatement of FLT

If n is odd prime then for every e, $1 \leq e < n$, $e^{\frac{n-1}{2}} = \pm 1 \; (mod \; n)$.

- When n is prime, e is a quadratic residue in Z_n iff $e^{\frac{n-1}{2}} = 1 \; (mod \; n)$.
- Therefore, if n is prime then

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \; (mod \; n).$$

Solovay-Strassen Test

- Proposed by Solovay and Strassen (1973).
- A randomized algorithm based on above property.
- Never incorrectly classifies primes and correctly classifies composites with probability at least $\frac{1}{2}$.

Solovay-Strassen Test

- Proposed by Solovay and Strassen (1973).
- A randomized algorithm based on above property.
- Never incorrectly classifies primes and correctly classifies composites with probability at least $\frac{1}{2}$.

Solovay-Strassen Test

- Proposed by Solovay and Strassen (1973).
- A randomized algorithm based on above property.
- Never incorrectly classifies primes and correctly classifies composites with probability at least $\frac{1}{2}$.

Solovay-Strassen Test

1. If n is an exact power, it is composite.
2. For a random e in Z_n, test if
$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$
3. If yes, classify n as prime otherwise it is proven composite.

- The time complexity is $\tilde{O}(\log^2 n)$.

Solovay-Strassen Test

1. If n is an exact power, it is composite.
2. For a random e in Z_n, test if

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$

3. If yes, classify n as prime otherwise it is proven composite.

- The time complexity is $\tilde{O}(\log^2 n)$.

Solovay-Strassen Test

1. If n is an exact power, it is composite.
2. For a random e in Z_n, test if

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$

3. If yes, classify n as prime otherwise it is proven composite.

- The time complexity is $O\tilde{}(\log^2 n)$.

Solovay-Strassen Test

1. If n is an exact power, it is composite.
2. For a random e in Z_n, test if

$$\left(\frac{e}{n}\right) = e^{\frac{n-1}{2}} \ (mod\ n).$$

3. If yes, classify n as prime otherwise it is proven composite.

- The time complexity is $O\tilde{}(\log^2 n)$.

Analysis

- Consider the case when n is a product of two primes p and q.
- Let $a, b \in Z_p$, $c \in Z_q$ with a residue and b non-residue in Z_p.
- Clearly, $< a, c >^{\frac{n-1}{2}} = < b, c >^{\frac{n-1}{2}} \pmod q$.
- If $< a, c >^{\frac{n-1}{2}} \neq < b, c >^{\frac{n-1}{2}} \pmod n$ then one of them is not in $\{1, -1\}$ and so compositeness of n is proven.
- Otherwise, either

$$\left(\frac{< a, c >}{n}\right) \neq < a, c >^{\frac{n-1}{2}} \pmod n,$$

or

$$\left(\frac{< b, c >}{n}\right) \neq < b, c >^{\frac{n-1}{2}} \pmod n.$$

Analysis

- Consider the case when n is a product of two primes p and q.
- Let $a, b \in Z_p$, $c \in Z_q$ with a residue and b non-residue in Z_p.
- Clearly, $< a, c >^{\frac{n-1}{2}} = < b, c >^{\frac{n-1}{2}} \pmod q$.
- If $< a, c >^{\frac{n-1}{2}} \neq < b, c >^{\frac{n-1}{2}} \pmod n$ then one of them is not in $\{1, -1\}$ and so compositeness of n is proven.
- Otherwise, either

$$\left(\frac{< a, c >}{n}\right) \neq < a, c >^{\frac{n-1}{2}} \pmod n,$$

or

$$\left(\frac{< b, c >}{n}\right) \neq < b, c >^{\frac{n-1}{2}} \pmod n.$$

Analysis

- Consider the case when n is a product of two primes p and q.
- Let $a, b \in Z_p$, $c \in Z_q$ with a residue and b non-residue in Z_p.
- Clearly, $< a, c >^{\frac{n-1}{2}} = < b, c >^{\frac{n-1}{2}} \pmod{q}$.
- If $< a, c >^{\frac{n-1}{2}} \neq < b, c >^{\frac{n-1}{2}} \pmod{n}$ then one of them is not in $\{1, -1\}$ and so compositeness of n is proven.
- Otherwise, either

$$\left(\frac{< a, c >}{n}\right) \neq < a, c >^{\frac{n-1}{2}} \pmod{n},$$

or

$$\left(\frac{< b, c >}{n}\right) \neq < b, c >^{\frac{n-1}{2}} \pmod{n}.$$

Analysis

- Consider the case when n is a product of two primes p and q.
- Let $a, b \in Z_p$, $c \in Z_q$ with a residue and b non-residue in Z_p.
- Clearly, $< a, c >^{\frac{n-1}{2}} = < b, c >^{\frac{n-1}{2}} \pmod{q}$.
- If $< a, c >^{\frac{n-1}{2}} \neq < b, c >^{\frac{n-1}{2}} \pmod{n}$ then one of them is not in $\{1, -1\}$ and so compositeness of n is proven.
- Otherwise, either

$$\left(\frac{< a, c >}{n}\right) \neq < a, c >^{\frac{n-1}{2}} \pmod{n},$$

or

$$\left(\frac{< b, c >}{n}\right) \neq < b, c >^{\frac{n-1}{2}} \pmod{n}.$$

Miller's Test

Theorem (Another Restatement of FLT)

If n is odd prime and $n = 1 + 2^s \cdot t$, t odd, then for every e, $1 \leq e < n$, the sequence $e^{2^{s-1} \cdot t} \pmod{n}$, $e^{2^{s-2} \cdot t} \pmod{n}$, $\ldots$, $e^t \pmod{n}$ has either all 1's or a -1 somewhere.

Miller's Test

- This theorem is the basis for Miller's test (1973).
- It is a deterministic polynomial time test.
- It is correct under Extended Riemann Hypothesis.

Miller's Test

- This theorem is the basis for Miller's test (1973).
- It is a deterministic polynomial time test.
- It is correct under Extended Riemann Hypothesis.

Miller's Test

- This theorem is the basis for Miller's test (1973).
- It is a deterministic polynomial time test.
- It is correct under Extended Riemann Hypothesis.

Miller's Test

1. If n is an exact power, it is composite.
2. For each e, $1 < e \leq 4\log^2 n$, check if the sequence $e^{2^{s-1}\cdot t}$ $(mod\ n)$, $e^{2^{s-2}\cdot t}$ $(mod\ n)$, ..., e^t $(mod\ n)$ has either all 1's or a -1 somewhere.
3. If yes, classify n as prime otherwise composite.

- The time complexity of the test is $O\tilde{}(\log^4 n)$.

Miller's Test

1. If n is an exact power, it is composite.
2. For each e, $1 < e \leq 4 \log^2 n$, check if the sequence $e^{2^{s-1} \cdot t}$ $(mod\ n)$, $e^{2^{s-2} \cdot t}$ $(mod\ n)$, ..., e^t $(mod\ n)$ has either all 1's or a -1 somewhere.
3. If yes, classify n as prime otherwise composite.

- The time complexity of the test is $O\tilde{}(\log^4 n)$.

Miller's Test

1. If n is an exact power, it is composite.
2. For each e, $1 < e \leq 4 \log^2 n$, check if the sequence $e^{2^{s-1} \cdot t} \pmod{n}$, $e^{2^{s-2} \cdot t} \pmod{n}$, ..., $e^{t} \pmod{n}$ has either all 1's or a -1 somewhere.
3. If yes, classify n as prime otherwise composite.

- The time complexity of the test is $\tilde{O}(\log^4 n)$.

Miller's Test

1. If n is an exact power, it is composite.
2. For each e, $1 < e \leq 4\log^2 n$, check if the sequence $e^{2^{s-1}\cdot t}$ $(mod\ n)$, $e^{2^{s-2}\cdot t}$ $(mod\ n)$, ..., e^t $(mod\ n)$ has either all 1's or a -1 somewhere.
3. If yes, classify n as prime otherwise composite.

- The time complexity of the test is $\tilde{O}(\log^4 n)$.

Rabin's Test

- A modification of Miller's algorithm proposed soon after (1974).
- Selects e randomly instead of trying all e in the range $[2, 4\log^2 n]$.
- Randomized algorithm that never classfies primes incorrectly and correctly classifies composites with probabilty at least $\frac{3}{4}$.
- Time complexity is $\tilde{O}(\log^2 n)$.
- The most popular primality testing algorithm.

Rabin's Test

- A modification of Miller's algorithm proposed soon after (1974).
- Selects e randomly instead of trying all e in the range $[2, 4\log^2 n]$.
- Randomized algorithm that never classfies primes incorrectly and correctly classifies composites with probabilty at least $\frac{3}{4}$.
- Time complexity is $O\tilde{}(\log^2 n)$.
- The most popular primality testing algorithm.

Rabin's Test

- A modification of Miller's algorithm proposed soon after (1974).
- Selects e randomly instead of trying all e in the range $[2, 4\log^2 n]$.
- Randomized algorithm that never classfies primes incorrectly and correctly classifies composites with probabilty at least $\frac{3}{4}$.
- Time complexity is $\tilde{O}(\log^2 n)$.
- The most popular primality testing algorithm.

Rabin's Test

- A modification of Miller's algorithm proposed soon after (1974).
- Selects e randomly instead of trying all e in the range $[2, 4\log^2 n]$.
- Randomized algorithm that never classfies primes incorrectly and correctly classifies composites with probabilty at least $\frac{3}{4}$.
- Time complexity is $\tilde{O}(\log^2 n)$.
- The most popular primality testing algorithm.

AKS Test

Theorem (A Generalization of FLT)

If n is prime then for every e, $1 \leq e < n$, $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$, $\zeta^r = 1$.

- A test proposed in 2002 based on this generalization.
- The only known deterministic, unconditionally correct, polynomial time algorithm.

AKS Test

Theorem (A Generalization of FLT)

If n is prime then for every e, $1 \leq e < n$, $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$, $\zeta^r = 1$.

- A test proposed in 2002 based on this generalization.
- The only known deterministic, unconditionally correct, polynomial time algorithm.

AKS Test

Theorem (A Generalization of FLT)

If n is prime then for every e, $1 \leq e < n$, $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$, $\zeta^r = 1$.

- A test proposed in 2002 based on this generalization.
- The only known deterministic, unconditionally correct, polynomial time algorithm.

AKS Test

1. If n is an exact power or has a small divisor, it is composite.
2. Select a small number r carefully, let $\zeta^r = 1$ and consider $Z_n[\zeta]$.
3. For each e, $1 \leq e \leq 2\sqrt{r}\log n$, check if $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$.
4. If yes, n is prime otherwise composite.

AKS Test

1. If n is an exact power or has a small divisor, it is composite.
2. Select a small number r carefully, let $\zeta^r = 1$ and consider $Z_n[\zeta]$.
3. For each e, $1 \leq e \leq 2\sqrt{r}\log n$, check if $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$.
4. If yes, n is prime otherwise composite.

AKS Test

1. If n is an exact power or has a small divisor, it is composite.
2. Select a small number r carefully, let $\zeta^r = 1$ and consider $Z_n[\zeta]$.
3. For each e, $1 \leq e \leq 2\sqrt{r}\log n$, check if $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$.
4. If yes, n is prime otherwise composite.

AKS Test

1. If n is an exact power or has a small divisor, it is composite.
2. Select a small number r carefully, let $\zeta^r = 1$ and consider $Z_n[\zeta]$.
3. For each e, $1 \leq e \leq 2\sqrt{r}\log n$, check if $(\zeta + e)^n = \zeta^n + e$ in $Z_n[\zeta]$.
4. If yes, n is prime otherwise composite.

Analysis

- Suppose n has at least two prime factors and let p be one of them.
- Let $S \subseteq Z_p[\zeta]$ such that for every element $f(\zeta) \in S$, $f\zeta)^n = f(\zeta^n)$ in $Z_p[\zeta]$.
- It follows that for every $f(\zeta) \in S$, $f(\zeta)^m = f(\zeta^m)$ for any m of the form $n^i \cdot p^j$.
- Since n is not a power of p, this places an upper bound on the size of S.
- If $\zeta + e \in S$ for every $1 \leq e \leq 2\sqrt{r} \log n$, then all their products are also in S.
- This makes the size of S bigger than the upper bound above.

Analysis

- Suppose n has at least two prime factors and let p be one of them.
- Let $S \subseteq Z_p[\zeta]$ such that for every element $f(\zeta) \in S$, $f\zeta)^n = f(\zeta^n)$ in $Z_p[\zeta]$.
- It follows that for every $f(\zeta) \in S$, $f(\zeta)^m = f(\zeta^m)$ for any m of the form $n^i \cdot p^j$.
- Since n is not a power of p, this places an upper bound on the size of S.
- If $\zeta + e \in S$ for every $1 \leq e \leq 2\sqrt{r} \log n$, then all their products are also in S.
- This makes the size of S bigger than the upper bound above.

Analysis

- Suppose n has at least two prime factors and let p be one of them.
- Let $S \subseteq Z_p[\zeta]$ such that for every element $f(\zeta) \in S$, $f\zeta)^n = f(\zeta^n)$ in $Z_p[\zeta]$.
- It follows that for every $f(\zeta) \in S$, $f(\zeta)^m = f(\zeta^m)$ for any m of the form $n^i \cdot p^j$.
- Since n is not a power of p, this places an upper bound on the size of S.
- If $\zeta + e \in S$ for every $1 \leq e \leq 2\sqrt{r}\log n$, then all their products are also in S.
- This makes the size of S bigger than the upper bound above.

Analysis

- Suppose n has at least two prime factors and let p be one of them.
- Let $S \subseteq Z_p[\zeta]$ such that for every element $f(\zeta) \in S$, $f\zeta)^n = f(\zeta^n)$ in $Z_p[\zeta]$.
- It follows that for every $f(\zeta) \in S$, $f(\zeta)^m = f(\zeta^m)$ for any m of the form $n^i \cdot p^j$.
- Since n is not a power of p, this places an upper bound on the size of S.
- If $\zeta + e \in S$ for every $1 \leq e \leq 2\sqrt{r}\log n$, then all their products are also in S.
- This makes the size of S bigger than the upper bound above.

Analysis

- Suppose n has at least two prime factors and let p be one of them.
- Let $S \subseteq Z_p[\zeta]$ such that for every element $f(\zeta) \in S$, $f\zeta)^n = f(\zeta^n)$ in $Z_p[\zeta]$.
- It follows that for every $f(\zeta) \in S$, $f(\zeta)^m = f(\zeta^m)$ for any m of the form $n^i \cdot p^j$.
- Since n is not a power of p, this places an upper bound on the size of S.
- If $\zeta + e \in S$ for every $1 \leq e \leq 2\sqrt{r} \log n$, then all their products are also in S.
- This makes the size of S bigger than the upper bound above.

Time Complexity

- Number r is $O(\log^5 n)$.
- Time complexity of the algorithm is $O\tilde{}(\log^{12} n)$.
- An improvement by Hendrik Lenstra (2002) reduces the time complexity to $O\tilde{}(\log^{15/2} n)$.
- Lenstra and Pomerance (2003) further reduce the time complexity to $O\tilde{}(\log^6 n)$.
- Bernstein (2003) reduced the time complexity to $O\tilde{}(\log^4 n)$ at the cost of making it randomized.

Time Complexity

- Number r is $O(\log^5 n)$.
- Time complexity of the algorithm is $\tilde{O}(\log^{12} n)$.
- An improvement by Hendrik Lenstra (2002) reduces the time complexity to $\tilde{O}(\log^{15/2} n)$.
- Lenstra and Pomerance (2003) further reduce the time complexity to $\tilde{O}(\log^6 n)$.
- Bernstein (2003) reduced the time complexity to $\tilde{O}(\log^4 n)$ at the cost of making it randomized.

Time Complexity

- Number r is $O(\log^5 n)$.
- Time complexity of the algorithm is $O\tilde{~}(\log^{12} n)$.
- An improvement by Hendrik Lenstra (2002) reduces the time complexity to $O\tilde{~}(\log^{15/2} n)$.
- Lenstra and Pomerance (2003) further reduce the time complexity to $O\tilde{~}(\log^6 n)$.
- Bernstein (2003) reduced the time complexity to $O\tilde{~}(\log^4 n)$ at the cost of making it randomized.

Time Complexity

- Number r is $O(\log^5 n)$.
- Time complexity of the algorithm is $\tilde{O}(\log^{12} n)$.
- An improvement by Hendrik Lenstra (2002) reduces the time complexity to $\tilde{O}(\log^{15/2} n)$.
- Lenstra and Pomerance (2003) further reduce the time complexity to $\tilde{O}(\log^6 n)$.
- Bernstein (2003) reduced the time complexity to $\tilde{O}(\log^4 n)$ at the cost of making it randomized.

Time Complexity

- Number r is $O(\log^5 n)$.
- Time complexity of the algorithm is $\tilde{O}(\log^{12} n)$.
- An improvement by Hendrik Lenstra (2002) reduces the time complexity to $\tilde{O}(\log^{15/2} n)$.
- Lenstra and Pomerance (2003) further reduce the time complexity to $\tilde{O}(\log^6 n)$.
- Bernstein (2003) reduced the time complexity to $\tilde{O}(\log^4 n)$ at the cost of making it randomized.

Outline

Adleman-Pomerance-Rumeli Test

- Proposed in 1980.
- Is conceptually the most complex algorithm of them all.
- Uses multiple groups, ideas derived from both themes, plus new ones!
- It is a deterministic algorithm with time complexity $\log^{O(\log\log\log n)} n$.
- Was speeded up by Cohen and Lenstra (1981).

Adleman-Pomerance-Rumeli Test

- Proposed in 1980.
- Is conceptually the most complex algorithm of them all.
- Uses multiple groups, ideas derived from both themes, plus new ones!
- It is a deterministic algorithm with time complexity $\log^{O(\log\log\log n)} n$.
- Was speeded up by Cohen and Lenstra (1981).

Adleman-Pomerance-Rumeli Test

- Proposed in 1980.
- Is conceptually the most complex algorithm of them all.
- Uses multiple groups, ideas derived from both themes, plus new ones!
- It is a deterministic algorithm with time complexity $\log^{O(\log\log\log n)} n$.
- Was speeded up by Cohen and Lenstra (1981).

Adleman-Pomerance-Rumeli Test

- Proposed in 1980.
- Is conceptually the most complex algorithm of them all.
- Uses multiple groups, ideas derived from both themes, plus new ones!
- It is a deterministic algorithm with time complexity $\log^{O(\log\log\log n)} n$.
- Was speeded up by Cohen and Lenstra (1981).

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Tries to compute a factor of n.
- Let p be a factor of n, $p \leq \sqrt{n}$.
- Find two sets of primes $\{q_1, q_2, \ldots, q_t\}$ and $\{r_1, r_2, \ldots, r_u\}$ satisfying:
  - $\prod_{i=1}^{t} q_i = \log^{O(\log\log\log n)} n$.
  - For each $j \leq u$, $r_j - 1$ is square-free and has only q_i's as prime divisors.
  - $\prod_{j=1}^{u} r_j > \sqrt{n}$.

Overview of the Algorithm

- Let g_j be a generator for the group $F_{r_j}^*$.
- Let $p = g_j^{\gamma_j}\ (mod\ r_j)$ and $\gamma_j = \delta_{i,j}\ (mod\ q_i)$ for every $q_i \mid r_j - 1$.
- Compute 'associated' primes $r_{j_i} \in \{r_1, r_2, \ldots, r_u\}$ for each q_i.
- Cycle through all tuples $(\alpha_1, \alpha_2, \ldots, \alpha_t)$ with $0 \le \alpha_i < q_i$.
- From a given tuple $(\alpha_1, \alpha_2, \ldots, \alpha_t)$, derive numbers $\beta_{i,j}$ for $1 \le j \le u$, $1 \le i \le t$ and $q_i \mid r_j - 1$ such that
  - If $p = g_{j_i}^{\alpha_i}\ (mod\ r_j)$ for every j then $\delta_{i,j} = \beta_{i,j}$ for every j and for every i with $q_i \mid r_j - 1$.

Overview of the Algorithm

- Let g_j be a generator for the group $F_{r_j}^*$.
- Let $p = g_j^{\gamma_j}$ $(mod\ r_j)$ and $\gamma_j = \delta_{i,j}$ $(mod\ q_i)$ for every $q_i \mid r_j - 1$.
- Compute 'associated' primes $r_{j_i} \in \{r_1, r_2, \ldots, r_u\}$ for each q_i.
- Cycle through all tuples $(\alpha_1, \alpha_2, \ldots, \alpha_t)$ with $0 \leq \alpha_i < q_i$.
- From a given tuple $(\alpha_1, \alpha_2, \ldots, \alpha_t)$, derive numbers $\beta_{i,j}$ for $1 \leq j \leq u$, $1 \leq i \leq t$ and $q_i \mid r_j - 1$ such that
  - If $p = g_{j_i}^{\alpha_i}$ $(mod\ r_j)$ for every j then $\delta_{i,j} = \beta_{i,j}$ for every j and for every i with $q_i \mid r_j - 1$.

Overview of the Algorithm

- Let g_j be a generator for the group $F_{r_j}^*$.
- Let $p = g_j^{\gamma_j}$ $(mod\ r_j)$ and $\gamma_j = \delta_{i,j}$ $(mod\ q_i)$ for every $q_i \mid r_j - 1$.
- Compute 'associated' primes $r_{j_i} \in \{r_1, r_2, \ldots, r_u\}$ for each q_i.
- Cycle through all tuples $(\alpha_1, \alpha_2, \ldots, \alpha_t)$ with $0 \leq \alpha_i < q_i$.
- From a given tuple $(\alpha_1, \alpha_2, \ldots, \alpha_t)$, derive numbers $\beta_{i,j}$ for $1 \leq j \leq u$, $1 \leq i \leq t$ and $q_i \mid r_j - 1$ such that
  - If $p = g_{j_i}^{\alpha_i}$ $(mod\ r_j)$ for every i then $\delta_{i,j} = \beta_{i,j}$ for every j and for every i with $q_i \mid r_j - 1$.

Overview of the Algorithm

- Let g_j be a generator for the group $F^*_{r_j}$.
- Let $p = g_j^{\gamma_j} \pmod{r_j}$ and $\gamma_j = \delta_{i,j} \pmod{q_i}$ for every $q_i \mid r_j - 1$.
- Compute 'associated' primes $r_{j_i} \in \{r_1, r_2, \ldots, r_u\}$ for each q_i.
- Cycle through all tuples $(\alpha_1, \alpha_2, \ldots, \alpha_t)$ with $0 \leq \alpha_i < q_i$.
- From a given tuple $(\alpha_1, \alpha_2, \ldots, \alpha_t)$, derive numbers $\beta_{i,j}$ for $1 \leq j \leq u$, $1 \leq i \leq t$ and $q_i \mid r_j - 1$ such that
  - If $p = g_{j_i}^{\alpha_i} \pmod{r_j}$ for every j then $\delta_{i,j} = \beta_{i,j}$ for every j and for every i with $q_i \mid r_j - 1$.

Overview of the Algorithm

- Let g_j be a generator for the group $F_{r_j}^*$.
- Let $p = g_j^{\gamma_j}$ $(mod\ r_j)$ and $\gamma_j = \delta_{i,j}$ $(mod\ q_i)$ for every $q_i \mid r_j - 1$.
- Compute 'associated' primes $r_{j_i} \in \{r_1, r_2, \ldots, r_u\}$ for each q_i.
- Cycle through all tuples $(\alpha_1, \alpha_2, \ldots, \alpha_t)$ with $0 \leq \alpha_i < q_i$.
- From a given tuple $(\alpha_1, \alpha_2, \ldots, \alpha_t)$, derive numbers $\beta_{i,j}$ for $1 \leq j \leq u$, $1 \leq i \leq t$ and $q_i \mid r_j - 1$ such that
  - If $p = g_{j_i}^{\alpha_i}$ $(mod\ r_j)$ for every j then $\delta_{i,j} = \beta_{i,j}$ for every j and for every i with $q_i \mid r_j - 1$.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p\ (mod\ \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p \ (mod \ \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p \ (mod \ \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p \ (mod\ \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p \ (mod \ \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Overview of the Algorithm

- From $\delta_{i,j}$'s, p can be constructed easily:
  - Use Chinese remaindering to compute γ_j's from $\delta_{i,j}$'s.
  - Use Chinese remaindering to compute $p \; (mod \; \prod_{j=1}^{u} r_j)$ from $g_j^{\gamma_j}$'s.
  - Since $\prod_{j=1}^{u} r_j > \sqrt{n} \geq p$, the residue equals p.
- $\beta_{i,j}$'s are computed using higher reciprocity laws in extension rings $Z_n[\zeta_i]$, $\zeta_i^{q_i} = 1$.
- For most of the composite numbers, the algorithm will fail during computation of $\beta_{i,j}$'s.

Outstanding Questions

- Is there a 'practical', polynomial time deterministic primality test?
- Is there a 'practical', provably polynomial time, primality proving test?
- Are there radically different ways of testing primality efficiently?

Outstanding Questions

- Is there a 'practical', polynomial time deterministic primality test?
- Is there a 'practical', provably polynomial time, primality proving test?
- Are there radically different ways of testing primality efficiently?

Outstanding Questions

- Is there a 'practical', polynomial time deterministic primality test?
- Is there a 'practical', provably polynomial time, primality proving test?
- Are there radically different ways of testing primality efficiently?