PRIMALITY TESTING ALGORITHMS

Prerona Chatterjee
(Roll No.: 142123029)

Department of Mathematics
IIT Guwahati

April, 2016

Importance of Prime Numbers

Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So, it is of great interest to study different properties of prime numbers, especially those properties that allow one to determine efficiently if a number is prime. Such efficient tests are also useful in practice. For example, a number of cryptographic protocols need large prime numbers. Infact, primality testing is one of the fundamental problems in computational number theory with important applications in complexity theory, coding theory, cryptography, computer algebra systems and elsewhere.

Developments in the field

- Most of the early algorithms are based on Fermat's little theorem and differ in their treatment to handle the Fermat's pseudoprimes.

Developments in the field

- Most of the early algorithms are based on Fermat's little theorem and differ in their treatment to handle the Fermat's pseudoprimes.
- Biswas and Agrawal introduced a new technique, which generalises the idea of Fermat's little theorem to a similar identity over polynomial rings which resulted in a simple probabilistic polynomial time algorithm which completely bypasses the issue of pseudoprimes.

Developments in the field

- Most of the early algorithms are based on Fermat's little theorem and differ in their treatment to handle the Fermat's pseudoprimes.
- Biswas and Agrawal introduced a new technique, which generalises the idea of Fermat's little theorem to a similar identity over polynomial rings which resulted in a simple probabilistic polynomial time algorithm which completely bypasses the issue of pseudoprimes.
- In 2004, Manindra Agrawal, Neeraj Kayal and Nitin Saxena, gave a deterministic polynomial time algorithm by derandomizing the previous algorithm.

Introduction

The Problem

Core Idea behind the Algorithms

Preliminaries

The Miller-Rabin Primality Testing Algorithm

The Agrawal-Biswas Primality Testing Algorithm

The AKS Primality Testing Algorithm

Aim of the Project

In this project we study some Primality testing algorithms (probabilistic or deterministic) that take time polynomial in $\log n$. Our main focus is to study the AKS algorithm which gives an unconditional deterministic polynomial time algorithm that determines whether a given input number is prime or not. However, before that we study two of its precursers, namely, the Miller-Rabin test and Agrawal-Biswas test, which give probablistic polynomial time algorithms to check for primality.

Aim of the Project

In this project we study some Primality testing algorithms (probabilistic or deterministic) that take time polynomial in $\log n$. Our main focus is to study the AKS algorithm which gives an unconditional deterministic polynomial time algorithm that determines whether a given input number is prime or not. However, before that we study two of its precursers, namely, the Miller-Rabin test and Agrawal-Biswas test, which give probablistic polynomial time algorithms to check for primality.

The problem we are looking at:

The goal of primality testing is to devise an algorithm which, given an integer n, decides whether n is a prime number. To represent an integer n we need $O(\log n)$ bits. Hence, the input size is $O(\log n)$ and by polynomial-time algorithm we mean algorithms with running time polynomial in $\log(n)$.

Formal Statement of the Problem

Formally, the problem stated as follows:

Formal Statement of the Problem

Formally, the problem stated as follows:

Problem

Give a (probabilistic or deterministic) algorithm which, given an integer n in binary representation, decides whether n is a prime number in time $O(\log^k n)$ where k is an integer independent of n.

Difficulty in the Naive Approach and how to overcome it

- The naive algorithm which tries to find a factor by checking whether n is divisible by $2, 3, \ldots, \lfloor \sqrt{n} \rfloor$ takes $\Theta(\sqrt{n})$ time which is exponential in the input-length.

Difficulty in the Naive Approach and how to overcome it

- The naive algorithm which tries to find a factor by checking whether n is divisible by $2, 3, \ldots, \lfloor \sqrt{n} \rfloor$ takes $\Theta(\sqrt{n})$ time which is exponential in the input-length.
- Integer factorization is a much harder problem than primality testing and not even a probabilistic polynomial time algorithm is known for that.

Difficulty in the Naive Approach and how to overcome it

- The naive algorithm which tries to find a factor by checking whether n is divisible by $2, 3, \ldots, \lfloor \sqrt{n} \rfloor$ takes $\Theta(\sqrt{n})$ time which is exponential in the input-length.
- Integer factorization is a much harder problem than primality testing and not even a probabilistic polynomial time algorithm is known for that.
- The effective approach to primality testing is to come up with mathematical identities which are satisfied by n if and only if n is a prime. This allows us to solve the decision problem without having knowledge of the factors of n.

The Core Idea

1. If n is less than some constant then check whether n is prime by *brute force*. Also, if $n = a^k$ for some integers a and k , where $k > 2$, then return COMPOSITE (ISPOWER(n) shows than it can be checked efficiently).

The Core Idea

1. If n is less than some constant then check whether n is prime by *brute force*. Also, if $n = a^k$ for some integers a and k , where $k > 2$, then return COMPOSITE ($\textsc{isPower}(n)$ shows than it can be checked efficiently).
2. *"Suitably choose"* a ring R_n and a subset S of R_n.

The Core Idea

1. If n is less than some constant then check whether n is prime by *brute force*. Also, if $n = a^k$ for some integers a and k , where $k > 2$, then return COMPOSITE ($\textsc{isPower}(n)$ shows than it can be checked efficiently).
2. *"Suitably choose"* a ring R_n and a subset S of R_n.
3. For suitably chosen elements $a_1, \ldots, a_l \in S$, if a_i does not satisfy a *"special identity"* then return COMPOSITE (else we may need to perform some *"other checks"*).

The Core Idea

1. If n is less than some constant then check whether n is prime by *brute force*. Also, if $n = a^k$ for some integers a and k , where $k > 2$, then return COMPOSITE ($\textsc{isPower}(n)$ shows than it can be checked efficiently).
2. *"Suitably choose"* a ring R_n and a subset S of R_n.
3. For suitably chosen elements $a_1, \ldots, a_l \in S$, if a_i does not satisfy a *"special identity"* then return COMPOSITE (else we may need to perform some *"other checks"*).
4. If each a_i satisfies the *"special identity"* in R_n then return PRIME.

The Miller-Rabin Test

- $R_n = \mathbb{Z}_n$

The Miller-Rabin Test

- $R_n = \mathbb{Z}_n$
- $S = \mathbb{Z}_n^*$

The Miller-Rabin Test

- $R_n = \mathbb{Z}_n$
- $S = \mathbb{Z}_n^*$
- $a_1, \ldots, a_l$ are randomly chosen elements from $\mathbb{Z}_n^*$

The Miller-Rabin Test

- $R_n = \mathbb{Z}_n$
- $S = \mathbb{Z}_n^*$
- $a_1, \ldots, a_l$ are randomly chosen elements from $\mathbb{Z}_n^*$
- The *"special identity"* is $a_i^{n-1} = 1$. This identity is motivated by Fermat's little theorem which says, if n is prime then for all $a \in \mathbb{Z}_n^*$, $a^{n-1} = 1$.

The Agrawal-Biswas Test

- Fermat's little theorem is generalised to: $(x+a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$)

The Agrawal-Biswas Test

- Fermat's little theorem is generalised to: $(x+a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$)
- $R_n = \mathbb{Z}_n[x]$

The Agrawal-Biswas Test

- Fermat's little theorem is generalised to: $(x+a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$)
- $R_n = \mathbb{Z}_n[x]$
- S is the set of all monic polynomial of degree $\lceil \log n \rceil$

The Agrawal-Biswas Test

- Fermat's little theorem is generalised to: $(x+a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$)
- $R_n = \mathbb{Z}_n[x]$
- S is the set of all monic polynomial of degree $\lceil \log n \rceil$
- $Q(x)$ is randomly chosen from S

The Agrawal-Biswas Test

- Fermat's little theorem is generalised to: $(x+a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$)
- $R_n = \mathbb{Z}_n[x]$
- S is the set of all monic polynomial of degree $\lceil \log n \rceil$
- $Q(x)$ is randomly chosen from S
- Check whether $(x+1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$

The Probabilistic Nature of these Algorithms

- Both of these algorithms are probabilistic and exhibits one sided error.

The Probabilistic Nature of these Algorithms

- Both of these algorithms are probabilistic and exhibits one sided error.
- If the output is COMPOSITE then n is a composite number.

The Probabilistic Nature of these Algorithms

- Both of these algorithms are probabilistic and exhibits one sided error.
- If the output is COMPOSITE then n is a composite number.
- If the ouput is PRIME then, n is prime with a high probability greater than a constant value.

The Probabilistic Nature of these Algorithms

- Both of these algorithms are probabilistic and exhibits one sided error.
- If the output is COMPOSITE then n is a composite number.
- If the ouput is PRIME then, n is prime with a high probability greater than a constant value.
- The probability can be boosted arbitrarity close to 1 by increasing I by a constant factor. However, without checking the identity for $O(|S|)$ elements in S we cannot make the probability to be exactly 1.

The Probabilistic Nature of these Algorithms

- Both of these algorithms are probabilistic and exhibits one sided error.
- If the output is COMPOSITE then n is a composite number.
- If the ouput is PRIME then, n is prime with a high probability greater than a constant value.
- The probability can be boosted arbitrarity close to 1 by increasing l by a constant factor. However, without checking the identity for $O(|S|)$ elements in S we cannot make the probability to be exactly 1.
- Since, for both the algorithms, the size of S is superpolynomial, it would take superpolynomial time to make them deterministic.

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.
- Equivalently, (by theorem 8) check: For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.
- Equivalently, (by theorem 8) check: For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$
- This is exactly like the Agrawal-Biswas test, except the size of S is reduced from $O(n^{\log n})$ to $\log^6 n$, which allows us to check the "identity" exhaustively for all the elements in S which leads to the deterministic algorithm.

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.
- Equivalently, (by theorem 8) check: For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$
- This is exactly like the Agrawal-Biswas test, except the size of S is reduced from $O(n^{\log n})$ to $\log^6 n$, which allows us to check the "identity" exhaustively for all the elements in S which leads to the deterministic algorithm.
- $R_n = \mathbb{Z}_n/(x^r - 1)$ where $r = O(\log^5 n)$

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.
- Equivalently, (by theorem 8) check: For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$
- This is exactly like the Agrawal-Biswas test, except the size of S is reduced from $O(n^{\log n})$ to $\log^6 n$, which allows us to check the "identity" exhaustively for all the elements in S which leads to the deterministic algorithm.
- $R_n = \mathbb{Z}_n/(x^r - 1)$ where $r = O(\log^5 n)$
- $S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$

The AKS Test

- Check whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$.
- Equivalently, (by theorem 8) check: For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$
- This is exactly like the Agrawal-Biswas test, except the size of S is reduced from $O(n^{\log n})$ to $\log^6 n$, which allows us to check the "identity" exhaustively for all the elements in S which leads to the deterministic algorithm.
- $R_n = \mathbb{Z}_n/(x^r - 1)$ where $r = O(\log^5 n)$
- $S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$
- For all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

Groups

- Order of a finite group is the cardinality of that group.

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

Groups

- Order of a finite group is the cardinality of that group.
- Order of a group element g is he smallest positive integer n for which g^n is identity. If no such integer exists then order of g is defined to be zero.

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

Groups

- Order of a finite group is the cardinality of that group.
- Order of a group element g is he smallest positive integer n for which g^n is identity. If no such integer exists then order of g is defined to be zero.

Proposition

The follwing properties of groups are well-known.

1. (**Lagrange Theorem**) If H is a subgroup of a finite group G, the order of H divides the order of G.

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

Groups

- Order of a finite group is the cardinality of that group.
- Order of a group element g is he smallest positive integer n for which g^n is identity. If no such integer exists then order of g is defined to be zero.

Proposition

The follwing properties of groups are well-known.

1. (**Lagrange Theorem**) If H is a subgroup of a finite group G, the order of H divides the order of G.
2. If g is an element of a finite group G, the order of g divides the order of G.

We state some of the very basic definitions and results about algebra and algorithms which will later be used.

Groups

- Order of a finite group is the cardinality of that group.
- Order of a group element g is he smallest positive integer n for which g^n is identity. If no such integer exists then order of g is defined to be zero.

Proposition

The follwing properties of groups are well-known.

1. (**Lagrange Theorem**) If H is a subgroup of a finite group G, the order of H divides the order of G.
2. If g is an element of a finite group G, the order of g divides the order of G.
3. Let H be a non-empty finite subset of a group G. H is a subgroup of G iff $\forall a, b \in H$, $ab \in H$.

Rings

- By "ring" we always mean commutative ring with multiplicative identity.

Rings

- By "ring" we always mean commutative ring with multiplicative identity.
- For a ring R, the set of units forms a multiplicative group which we write as R^*.

Rings

- By "ring" we always mean commutative ring with multiplicative identity.
- For a ring R, the set of units forms a multiplicative group which we write as R^*.
- For an element r in ring R, rR is the ideal generated by r and R/rR is the quotient ring.

Rings

- By "ring" we always mean commutative ring with multiplicative identity.
- For a ring R, the set of units forms a multiplicative group which we write as R^*.
- For an element r in ring R, rR is the ideal generated by r and R/rR is the quotient ring.
- For $a \in R$, $a + rR$ is the coset of rR in R with respect to a. However, we often abuse the notation to identify $a + rR$ with a. Hence, whenever we use an expression of like, $a \in R/rR$, where a is an element of R itself, it should be understood that we mean $a + rR \in R/rR$.Order of a finite group is the cardinality of that group.

Rings (contd...)

- If R is a euclidean domain then,

Proposition

1. For $a \in R$, $a \in R/rR^*$ if and only if $gcd(a, r) = 1$.
2. R/rR is a field if and only if r is a prime element of R.

Rings (contd...)

- If R is a euclidean domain then,

Proposition

1. For $a \in R$, $a \in R/rR^*$ if and only if $gcd(a, r) = 1$.
2. R/rR is a field if and only if r is a prime element of R.

- $\mathbb{Z}$ denotes the ring of integers. For $n \in \mathbb{Z} \setminus \{0\}$, $\mathbb{Z}/n\mathbb{Z}$ is written as $\mathbb{Z}_n$. As mentioned above, when a is an integer, expression of the form $a \in \mathbb{Z}_n$ actually mean $a + n\mathbb{Z} \in \mathbb{Z}_n$. For ring R, $R[x]$ is the univariate polynomial ring over R and we use $R[x]/f(x)$ to denote the quotient ring $R[x]/f(x)R[x]$.

Rings (contd...)

- If R is a euclidean domain then,

Proposition

1. For $a \in R$, $a \in R/rR^*$ if and only if $gcd(a, r) = 1$.
2. R/rR is a field if and only if r is a prime element of R.

- $\mathbb{Z}$ denotes the ring of integers. For $n \in \mathbb{Z} \setminus \{0\}$, $\mathbb{Z}/n\mathbb{Z}$ is written as $\mathbb{Z}_n$. As mentioned above, when a is an integer, expression of the form $a \in \mathbb{Z}_n$ actually mean $a + n\mathbb{Z} \in \mathbb{Z}_n$. For ring R, $R[x]$ is the univariate polynomial ring over R and we use $R[x]/f(x)$ to denote the quotient ring $R[x]/f(x)R[x]$.
- Cartesian product of two rings R_1 and R_2 can be identified as a ring by defining $(a, b) + (c, d) = (a + c, b + d)$ and $(a, b)(c, d) = (ac, bd)$ for all $a, c \in R_1$ and $b, d \in R_2$.

Fields

From Proposition 18.2 it follows that $\mathbb{Z}_n$ is a field iff n is prime and, for a finite field $\mathbb{F}$, $\mathbb{F}[x]/f(x)$ is a field iff $f(x)$ is irreducible over $\mathbb{F}$. It is also well-known that there exists a finite field of order q iff $q = p^d$ where p is prime and d is a positive integer. All finite fields of same order are isomorphic, and hence we can talk about "the" finite field of order q denoting it by $\mathbb{F}_q$. For $q = p^d$, p is the *characteristic* of $\mathbb{F}_q$.

Fields

From Proposition 18.2 it follows that $\mathbb{Z}_n$ is a field iff n is prime and, for a finite field $\mathbb{F}$, $\mathbb{F}[x]/f(x)$ is a field iff $f(x)$ is irreducible over $\mathbb{F}$. It is also well-known that there exists a finite field of order q iff $q = p^d$ where p is prime and d is a positive integer. All finite fields of same order are isomorphic, and hence we can talk about "the" finite field of order q denoting it by $\mathbb{F}_q$. For $q = p^d$, p is the *characteristic* of $\mathbb{F}_q$.

Theorem (Fermat's Little Theorem)

Let p be a prime number. For all $a \in \mathbb{F}_p^*$, $a^{p-1} = 1$.

Fields

From Proposition 18.2 it follows that $\mathbb{Z}_n$ is a field iff n is prime and, for a finite field $\mathbb{F}$, $\mathbb{F}[x]/f(x)$ is a field iff $f(x)$ is irreducible over $\mathbb{F}$. It is also well-known that there exists a finite field of order q iff $q = p^d$ where p is prime and d is a positive integer. All finite fields of same order are isomorphic, and hence we can talk about "the" finite field of order q denoting it by $\mathbb{F}_q$. For $q = p^d$, p is the *characteristic* of $\mathbb{F}_q$.

Theorem (Fermat's Little Theorem)

Let p be a prime number. For all $a \in \mathbb{F}_p^*$, $a^{p-1} = 1$.

Proof.

Since p is prime, by Proposition 18.1, $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ and $|\mathbb{F}_p^*| = p - 1$. By Proposition 16.2, for all $a \in \mathbb{F}_p^*$, $a^{|\mathbb{F}_p^*|} = 1$. □

Theorem

Let $n \in \mathbb{Z}$ and $n = n_1 n_2$ such that $gcd(n_1, n_2) = 1$, then $\mathbb{Z}_n$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. In particular $\phi : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ as defined below is an isomorphism:

$$\phi(x + n\mathbb{Z}) = (x + n_1\mathbb{Z}, x + n_2\mathbb{Z}).$$

Thus in particular, $\mathbb{Z}_n^*$ is isomorphic to $\mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$

Theorem

Let $n \in \mathbb{Z}$ and $n = n_1 n_2$ such that $gcd(n_1, n_2) = 1$, then $\mathbb{Z}_n$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. In particular $\phi : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ as defined below is an isomorphism:

$$\phi(x + n\mathbb{Z}) = (x + n_1\mathbb{Z}, x + n_2\mathbb{Z}).$$

Thus in particular, $\mathbb{Z}_n^*$ is isomorphic to $\mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$

For a field $\mathbb{F}$, the polynomial ring $\mathbb{F}[x]$ is an integral domain which implies the following.

Theorem

If $\mathbb{F}$ is a field and $f(x) \in \mathbb{F}[x]$ is a polynomial of degree d, then f cannot have more than d roots.

Some Basic Algorithms

Inputs are an integer or integers of $O(n)$. Hence in binary representation, the input size is $O(\log n)$. Thus in order to have polynomial time algorithm, it is required that the time complexities are polynomial in $\log n$.

Some Basic Algorithms

Inputs are an integer or integers of $O(n)$. Hence in binary representation, the input size is $O(\log n)$. Thus in order to have polynomial time algorithm, it is required that the time complexities are polynomial in $\log n$.

- $\textsc{isPower}(n)$ checks whether the integer n can be expressed as x^k where x, k are integers and $k > 2$. Step 3 of $\textsc{isPower}(n)$ can be done in $O(\log n)$ time and hence the total time complexity is $O(\log^2 n)$.

Some Basic Algorithms

Inputs are an integer or integers of $O(n)$. Hence in binary representation, the input size is $O(\log n)$. Thus in order to have polynomial time algorithm, it is required that the time complexities are polynomial in $\log n$.

- $\textsc{isPower}(n)$ checks whether the integer n can be expressed as x^k where x, k are integers and $k > 2$. Step 3 of $\textsc{isPower}(n)$ can be done in $O(\log n)$ time and hence the total time complexity is $O(\log^2 n)$.
- $\textsc{gcd}(a, b)$ computes the gcd of two integers using Euclid's method. If a and b are $O(n)$, it can be shown that the while loop in $\textsc{gcd}(a, b)$ iterates $O(\log n)$ times.

Some Basic Algorithms

Inputs are an integer or integers of $O(n)$. Hence in binary representation, the input size is $O(\log n)$. Thus in order to have polynomial time algorithm, it is required that the time complexities are polynomial in $\log n$.

- $\textsc{isPower}(n)$ checks whether the integer n can be expressed as x^k where x, k are integers and $k > 2$. Step 3 of $\textsc{isPower}(n)$ can be done in $O(\log n)$ time and hence the total time complexity is $O(\log^2 n)$.
- $\textsc{gcd}(a, b)$ computes the gcd of two integers using Euclid's method. If a and b are $O(n)$, it can be shown that the while loop in $\textsc{gcd}(a, b)$ iterates $O(\log n)$ times.
- $\textsc{Mod-Exp}(a, b, n)$ gives an algorithm which takes time polynomial in $\log n$ for modular exponentiation. We cannot directly compute a^b first and then take modulo n because it will take $O(n \log n)$ bits to represent a^b and hence the time will be superpolynomial in input size.

Algorithm 1 Check if a Number is an Integral Power

```
1: procedure isPower(n)
2:     for k = 1 to lg n do
3:         Use bisection method to find the largest integer x s.t. x^k ≤ n
4:         if x^k = n then
5:             return True
6:         end if
7:     end for
8:     return False
9: end procedure
```

Algorithm 2 Compute GCD of two integers

```
1: procedure gcd(a, b)
2:     while b does not divide a do
3:         b' = a mod b
4:         a = b
5:         b = b'
6:     end while
7:     return b
8: end procedure
```

Algorithm 3 Modular Exponentiation: Given integers a, b, n compute $a^b \mod n$

1: **procedure** MOD-EXP(a, b, n)
2: $c = 0$
3: $d = 1$
4: let $\{b_k, b_{k-1}, ..., b_0\}$ be the binary representation of b
5: **for** i=k **downto** 0 **do**
6: $c = 2c$
7: $d = d^2 \pmod n$
8: **if** $b_i == 1$ **then**
9: $c = c + 1$
10: $d = d.a \pmod n$
11: **end if**
12: **end for**
13: return d
14: **end procedure**

The Key Concepts

We know that if n is prime then, by Fermat's little theorem

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{1}$$

The Key Concepts

We know that if n is prime then, by Fermat's little theorem

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{1}$$

However, there are composite numbers called 'Fermat's pseudoprimes' for which equation 1 is also true.

The Key Concepts

We know that if n is prime then, by Fermat's little theorem

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{1}$$

However, there are composite numbers called 'Fermat's pseudoprimes' for which equation 1 is also true.
There are 2 key concepts behind the algorithm.

The Key Concepts

We know that if n is prime then, by Fermat's little theorem

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{1}$$

However, there are composite numbers called 'Fermat's pseudoprimes' for which equation 1 is also true.
There are 2 key concepts behind the algorithm.

- If n is not a pseudoprime then the converse of equation 1 is almost true. In particular, for at least half of the elements a in $\mathbb{Z}_n^*$ equation 1 does not hold.

The Key Concepts

We know that if n is prime then, by Fermat's little theorem

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{1}$$

However, there are composite numbers called 'Fermat's pseudoprimes' for which equation 1 is also true.
There are 2 key concepts behind the algorithm.

- If n is not a pseudoprime then the converse of equation 1 is almost true. In particular, for at least half of the elements a in $\mathbb{Z}_n^*$ equation 1 does not hold.
- If n is a pseudoprime then with very high probability we can find a non trivial square root of unity in $\mathbb{Z}_n$ which actually allows us to factorise n.

The Algorithm

Let n be the given number which we have to test for primality.

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.
- Pick some a in the range $\{1, 2, ..., n-1\}$ and check whether $\gcd(n, a) = 1$

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.
- Pick some a in the range $\{1, 2, ..., n-1\}$ and check whether $\gcd(n, a) = 1$
- Otherwise, check for non-trivial square-roots of unity in $\mathbb{Z}_n$ by first putting $x = a^u$ and then repeatedly squaring it till x becomes equal to a^{n-1}.

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.
- Pick some a in the range $\{1, 2, ..., n-1\}$ and check whether $\gcd(n, a) = 1$
- Otherwise, check for non-trivial square-roots of unity in $\mathbb{Z}_n$ by first putting $x = a^u$ and then repeatedly squaring it till x becomes equal to a^{n-1}.
- Check the Fermat's condition, namely check whether $a^{n-1} = 1$ for $a \in \mathbb{Z}_n^*$.

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.
- Pick some a in the range $\{1, 2, ..., n-1\}$ and check whether $\gcd(n, a) = 1$
- Otherwise, check for non-trivial square-roots of unity in $\mathbb{Z}_n$ by first putting $x = a^u$ and then repeatedly squaring it till x becomes equal to a^{n-1}.
- Check the Fermat's condition, namely check whether $a^{n-1} = 1$ for $a \in \mathbb{Z}_n^*$.
- If n fails in either of the two tests, then n is definitely composite. Otherwise n is probably prime.

The Algorithm

Let n be the given number which we have to test for primality.

- Check if n is an integral power. Otherwise, rewrite $n-1$ as $2^t u$.
- Pick some a in the range $\{1, 2, ..., n-1\}$ and check whether $\gcd(n, a) = 1$
- Otherwise, check for non-trivial square-roots of unity in $\mathbb{Z}_n$ by first putting $x = a^u$ and then repeatedly squaring it till x becomes equal to a^{n-1}.
- Check the Fermat's condition, namely check whether $a^{n-1} = 1$ for $a \in \mathbb{Z}_n^*$.
- If n fails in either of the two tests, then n is definitely composite. Otherwise n is probably prime.
- We can repeat the process a constant number of times by taking different a's so as to decrease the probability of error.

Algorithm 4 Miller-Rabin Primality Test

```
1: procedure MillerRabin(n)
2:     if isPower(n) or n is even with n ≠ 2 then
3:         return COMPOSITE
4:     end if
5:     Choose a to be a random number in the range {1, 2, ..., n − 1}
6:     if GCD(a, n) ≠ 1 then
7:         return COMPOSITE
8:     end if
9:     Find t, u such that (n − 1) = 2^t u and u is odd
10:    Put x_0 = Mod-Exp(a, u, n)
11:    for i=1 to t do
12:        x_i = x_{i−1}^2 (mod n)
13:        if x_i == 1 and x_{i−1} ≠ 1 and x_{i−1} ≠ −1 then
14:            return COMPOSITE
15:            //Checking for non-trivial square roots of unity
16:        end if
17:    end for
18:    if x_t ≠ 1 then
19:        return COMPOSITE //Checking Fermat's Condition
20:    end if
21:    return PRIME
22: end procedure
```

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- If n is composite, we divide the analysis into two cases in both of which, we try to find a proper subgroup of $\mathbb{Z}_n^*$ which contains all the elements for which $\textsc{MillerRabin}(n)$ returns PRIME and hence all the elements for which the algorithm can err.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- If n is composite, we divide the analysis into two cases in both of which, we try to find a proper subgroup of $\mathbb{Z}_n^*$ which contains all the elements for which $\textsc{MillerRabin}(n)$ returns PRIME and hence all the elements for which the algorithm can err.

Thus, the error in the algorithm becomes less than $\frac{1}{2}$, and so can be made arbitrarily small by repeating the procedure a constant number of times.

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.
On the other hand, if n is composite, we have the following two cases:

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.

On the other hand, if n is composite, we have the following two cases:

Case 1: n is not a pseudoprime

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.
On the other hand, if n is composite, we have the following two cases:

Case 1: n is not a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.
On the other hand, if n is composite, we have the following two cases:

Case 1: n is not a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\exists x \in \mathbb{Z}_n^*$ such that $x^{n-1} \neq 1 \pmod{n}$

Theorem

If n is prime, $\textsc{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\textsc{MillerRabin}(n)$ is always PRIME.
On the other hand, if n is composite, we have the following two cases:

Case 1: n is not a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\exists x \in \mathbb{Z}_n^*$ such that $x^{n-1} \neq 1 \pmod{n}$
- $B = \{z \in \mathbb{Z}_n^* : z^{n-1} = 1 \pmod{n}\}$ is a proper non empty subgroup of $\mathbb{Z}_n^*$

Theorem

If n is prime, $\text{MillerRabin}(n)$ outputs PRIME with probability one, and if n is composite, it outputs COMPOSITE with probability $> \frac{1}{2}$.

Proof

As explained in the key concepts, if n is prime, then the output of the procedure $\text{MillerRabin}(n)$ is always PRIME.
On the other hand, if n is composite, we have the following two cases:

Case 1: n is not a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\exists x \in \mathbb{Z}_n^*$ such that $x^{n-1} \neq 1 \pmod{n}$
- $B = \{z \in \mathbb{Z}_n^* : z^{n-1} = 1 \pmod{n}\}$ is a proper non empty subgroup of $\mathbb{Z}_n^*$
- $|B| \leq \frac{|\mathbb{Z}_n^*|}{2}$

Proof (contd...)

Case 2: n is a pseudoprime

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*,\ x^{n-1} = 1 \pmod{n}$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*$, $x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*$, $x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$
- For any n such that the algorithm has reached 6, n is not an integral power

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*$, $x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$
- For any n such that the algorithm has reached 6, n is not an integral power
- $\exists n_1, n_2 \in \mathbb{N}$ such that $n = n_1 n_2$ where $gcd(n_1, n_2) = 1$ and so by Theorem 3, $\mathbb{Z}_n^* \cong \mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*$, $x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$
- For any n such that the algorithm has reached 6, n is not an integral power
- $\exists n_1, n_2 \in \mathbb{N}$ such that $n = n_1 n_2$ where $gcd(n_1, n_2) = 1$ and so by Theorem 3, $\mathbb{Z}_n^* \cong \mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$
- $\exists a \in \mathbb{Z}_n^* \backslash B$ such that $a \cong (1, -1)$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*,\ x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$
- For any n such that the algorithm has reached 6, n is not an integral power
- $\exists n_1, n_2 \in \mathbb{N}$ such that $n = n_1 n_2$ where $gcd(n_1, n_2) = 1$ and so by Theorem 3, $\mathbb{Z}_n^* \cong \mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$
- $\exists a \in \mathbb{Z}_n^* \backslash B$ such that $a \cong (1, -1)$
- B is a proper subgroup of $\mathbb{Z}_n^*$

Proof (contd...)

Case 2: n is a pseudoprime

- Any a that has reached line 9 is in $\mathbb{Z}_n^*$
- $\forall x \in \mathbb{Z}_n^*,\ x^{n-1} = 1 \pmod{n}$
- $B = \{y \in \mathbb{Z}_n^* : y = \pm 1 \pmod{n}\}$ is a subgroup of $\mathbb{Z}_n^*$
- For any n such that the algorithm has reached 6, n is not an integral power
- $\exists n_1, n_2 \in \mathbb{N}$ such that $n = n_1 n_2$ where $gcd(n_1, n_2) = 1$ and so by Theorem 3, $\mathbb{Z}_n^* \cong \mathbb{Z}_{n_1}^* \times \mathbb{Z}_{n_2}^*$
- $\exists a \in \mathbb{Z}_n^* \backslash B$ such that $a \cong (1, -1)$
- B is a proper subgroup of $\mathbb{Z}_n^*$
- $|B| \leq \frac{|\mathbb{Z}_n^*|}{2}$

Proof (contd...)

- For any 'a' such that PRIME is returned by $\textsc{MillerRabin}(n)$, a is a member of B

Proof (contd...)

- For any 'a' such that PRIME is returned by $\textsc{MillerRabin}(n)$, a is a member of B
- Any element in $\mathbb{Z}_n^*$ that is a non-witness to the compositeness of n is a member of B

Proof (contd...)

- For any 'a' such that PRIME is returned by $\textsc{MillerRabin}(n)$, a is a member of B
- Any element in $\mathbb{Z}_n^*$ that is a non-witness to the compositeness of n is a member of B
- The probability of error is $\leq \frac{|\mathbb{Z}_n^*|/2}{|\mathbb{Z}_n^*|} = \frac{1}{2}$

Proof (contd...)

- For any 'a' such that PRIME is returned by $\textsc{MillerRabin}(n)$, a is a member of B
- Any element in $\mathbb{Z}_n^*$ that is a non-witness to the compositeness of n is a member of B
- The probability of error is $\leq \frac{|\mathbb{Z}_n^*|/2}{|\mathbb{Z}_n^*|} = \frac{1}{2}$
- The output is COMPOSITE with probability $> \frac{1}{2}$

Running Time Analysis

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time

Running Time Analysis

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time
- Step 6 takes $O(\log n)$ time

Running Time Analysis

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time
- Step 6 takes $O(\log n)$ time
- Step 9 takes $O(\log n)$ time since t (and hence u) can be found in atmost $\log n$ steps of dividing $(n-1)$ by 2

Running Time Analysis

- If n is the input, $\text{ISPOWER}(n)$ takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time
- Step 6 takes $O(\log n)$ time
- Step 9 takes $O(\log n)$ time since t (and hence u) can be found in atmost $\log n$ steps of dividing $(n-1)$ by 2
- $\text{MOD-EXP}(a, b, n)$ gives a polynomial time algorithm in $\log n$ for modular exponentiation as we have already seen in Section 21 and thus Step 10 takes time polynomial in $\log n$.

Running Time Analysis

- If n is the input, $\text{ISPOWER}(n)$ takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time
- Step 6 takes $O(\log n)$ time
- Step 9 takes $O(\log n)$ time since t (and hence u) can be found in atmost $\log n$ steps of dividing $(n-1)$ by 2
- $\text{MOD-EXP}(a, b, n)$ gives a polynomial time algorithm in $\log n$ for modular exponentiation as we have already seen in Section 21 and thus Step 10 takes time polynomial in $\log n$.
- Since no step in the for loop takes more that polynomial time in $\log n$, and since $t = O(\log n)$, $\text{MILLER-RABIN}(n)$ requires no more than time polynomial in $\log n$.

The Key Concept

- It is a randomised primality testing algorithm which reduces primality testing for a number n to testing if a specific univariate identity over $\mathbb{Z}_n$ holds.

The Key Concept

- It is a randomised primality testing algorithm which reduces primality testing for a number n to testing if a specific univariate identity over $\mathbb{Z}_n$ holds.
  It uses the following generalisation of Fermat's little theorem over polynomial ring.

Theorem

Let $P_n(x) = (a + x)^n - (a + x^n)$ *where* $a \in \mathbb{Z}_n^*$ *and* $n \in \mathbb{N}$.
Then, $P_n(x) = 0$ *in* $\mathbb{Z}_n[x]$ *iff* n *is prime.*

The Key Concept

- It is a randomised primality testing algorithm which reduces primality testing for a number n to testing if a specific univariate identity over $\mathbb{Z}_n$ holds.
  It uses the following generalisation of Fermat's little theorem over polynomial ring.

Theorem

Let $P_n(x) = (a+x)^n - (a+x^n)$ *where* $a \in \mathbb{Z}_n^*$ *and* $n \in \mathbb{N}$.
Then, $P_n(x) = 0$ *in* $\mathbb{Z}_n[x]$ *iff* n *is prime.*

- Unlike Fermat's little theorem, the identity is always false when n is a composite number.

The Key Concept

- It is a randomised primality testing algorithm which reduces primality testing for a number n to testing if a specific univariate identity over $\mathbb{Z}_n$ holds.
  It uses the following generalisation of Fermat's little theorem over polynomial ring.

Theorem

Let $P_n(x) = (a+x)^n - (a+x^n)$ *where* $a \in \mathbb{Z}_n^*$ *and* $n \in \mathbb{N}$. *Then,* $P_n(x) = 0$ *in* $\mathbb{Z}_n[x]$ *iff* n *is prime.*

- Unlike Fermat's little theorem, the identity is always false when n is a composite number.
- The issue of pseudoprimes is completely sidestepped leading to the following simple algorithm: If $(x+1)^n = (x^n+1) \mod n$ then return PRIME else return COMPOSITE.

The Key Concept (contd.)

- The polynomials in both side expands into $O(n)$ term and we need $O(n)$ time to compute them.

The Key Concept (contd.)

- The polynomials in both side expands into $O(n)$ term and we need $O(n)$ time to compute them.
- We check the identity modulo a randomly chosen monic polynomial of degree $\log n$ at the cost of introducing one sided error.

The Key Concept (contd.)

- The polynomials in both side expands into $O(n)$ term and we need $O(n)$ time to compute them.
- We check the identity modulo a randomly chosen monic polynomial of degree $\log n$ at the cost of introducing one sided error.
- The analysis of the algoritm shows that even in this case, when n is COMPOSITE, the identity does not hold with probability greater than $2/3$.

The Algorithm

- Choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$

The Algorithm

- Choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$
- Check whether $(x+1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$

The Algorithm

- Choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$
- Check whether $(x+1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$
- Using Theorem 6 with $a = 1$, we see that if that does not happen, then n is definitely composite

The Algorithm

- Choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$
- Check whether $(x+1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$
- Using Theorem 6 with $a = 1$, we see that if that does not happen, then n is definitely composite
- Otherwise n is prime with a probability $> \frac{2}{3}$

The Algorithm

- Choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$
- Check whether $(x+1)^n - (x^n+1)$ is zero modulo the polynomial $Q(x)$
- Using Theorem 6 with $a = 1$, we see that if that does not happen, then n is definitely composite
- Otherwise n is prime with a probability $> \frac{2}{3}$
- The process can be repeated a constant number of times by taking different $Q(x)$'s so as to decrease the probability of error arbitrarily.

Algorithm 5 Agrawal-Biswas Primality Test

procedure ABPRIME(n)
  if $n = 2, 3, 5, 7, 11, 13$ **then**
    return PRIME
  else
    if n is divisible by any of the above numbers **then**
      return COMPOSITE
    end if
  end if
  if ISPOWER(n) **then**
    return COMPOSITE
  end if
  $P_n(x) = (1 + x)^n - (1 - x^n)$
  Choose $Q(x)$ to be a random $\lceil \log n \rceil$ degree monic polynomial in $\mathbb{Z}_n[x]$
  if $Q(x)$ divides $P_n(x)$ over $\mathbb{Z}_n$ **then**
    return PRIME
  else
    return COMPOSITE
  end if
end procedure

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- However, if n is composite, the probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) > \frac{2}{3}$, so that the error in the algorithm becomes $\leq \frac{2}{3}$.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- However, if n is composite, the probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) > \frac{2}{3}$, so that the error in the algorithm becomes $\leq \frac{2}{3}$.

To show this:

- Define a set $\mathcal{I}$ of monic irreducible polynomials having degree in a certain range with upper bound l.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- However, if n is composite, the probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) > \frac{2}{3}$, so that the error in the algorithm becomes $\leq \frac{2}{3}$.

To show this:

- Define a set $\mathcal{I}$ of monic irreducible polynomials having degree in a certain range with upper bound l.
- For each polynomial f in $\mathcal{I}$, define a set C_f of degree l polynomials having f as a factor.

Correctness

- If n is prime, then it can be trivially shown that the output is always correct.
- However, if n is composite, the probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) > \frac{2}{3}$, so that the error in the algorithm becomes $\leq \frac{2}{3}$.

To show this:

- Define a set $\mathcal{I}$ of monic irreducible polynomials having degree in a certain range with upper bound l.
- For each polynomial f in $\mathcal{I}$, define a set C_f of degree l polynomials having f as a factor.
- Note that these C_f's are mutually disjoint, and hence find a lower bound for the number of degree l polynomials with factors in $\mathcal{I}$.

Correctness (contd.)

- Get an upper bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x)$, and hence get a lower bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x)$.

Correctness (contd.)

- Get an upper bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x)$, and hence get a lower bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x)$.
- Note that this also gives a lower bound on the number of monic polynomials of degree l that do not divide $P_n(x)$ and hence we get the required result.

Correctness (contd.)

- Get an upper bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x)$, and hence get a lower bound on the number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x)$.
- Note that this also gives a lower bound on the number of monic polynomials of degree l that do not divide $P_n(x)$ and hence we get the required result.

We formalise this as follows:

Theorem

If n is prime, $\mathrm{ABPRIME}(n)$ outputs PRIME with probability one, and if n is composite, its probability of error $\leq \frac{2}{3}$.

Proof

- If n is prime, $P_n(x) = 0$ in $\mathbb{Z}_n[x]$
  $\Rightarrow$ for any $Q(x)$ chosen, $Q(x) \mid P_n(x)$, and so the output of the algorithm is PRIME.

Proof

- If n is prime, $P_n(x) = 0$ in $\mathbb{Z}_n[x]$
  $\Rightarrow$ for any $Q(x)$ chosen, $Q(x) \mid P_n(x)$, and so the output of the algorithm is PRIME.
- If n is composite, $P_n(x) \neq 0$ in $\mathbb{Z}_n[x]$

Proof

- If n is prime, $P_n(x) = 0$ in $\mathbb{Z}_n[x]$
  $\Rightarrow$ for any $Q(x)$ chosen, $Q(x) \mid P_n(x)$, and so the output of the algorithm is PRIME.
- If n is composite, $P_n(x) \neq 0$ in $\mathbb{Z}_n[x]$
- Let p be any prime factor of n

Proof

- If n is prime, $P_n(x) = 0$ in $\mathbb{Z}_n[x]$
  $\Rightarrow$ for any $Q(x)$ chosen, $Q(x) \mid P_n(x)$, and so the output of the algorithm is PRIME.
- If n is composite, $P_n(x) \neq 0$ in $\mathbb{Z}_n[x]$
- Let p be any prime factor of n
- The algorithm is correct when n is a prime power or when divisible by primes upto 13, and so we only have to analyse when n is odd, not a prime power and its every prime factor is atleast 17.

Proof (contd.)

- $l = \lceil \log n \rceil$

Proof (contd.)

- $l = \lceil \log n \rceil$
- $\mathcal{I}$ is the set of all monic irreducible polynomials of degree between $1 + \frac{l}{2}$ and l over $\mathbb{F}_p$

Proof (contd.)

- $l = \lceil \log n \rceil$
- $\mathcal{I}$ is the set of all monic irreducible polynomials of degree between $1 + \frac{l}{2}$ and l over $\mathbb{F}_p$
- $I(d)$ be the number of monic irreducible polynomials of degree d over $\mathbb{F}_p$

Proof (contd.)

- $l = \lceil \log n \rceil$
- $\mathcal{I}$ is the set of all monic irreducible polynomials of degree between $1 + \frac{l}{2}$ and l over $\mathbb{F}_p$
- $I(d)$ be the number of monic irreducible polynomials of degree d over $\mathbb{F}_p$
- By the distribution theorem of irreducible polynomials[5],

$$\frac{p^k}{k} - p^{\frac{k}{2}} \leq I(d) \leq \frac{p^k}{k} + p^{\frac{k}{2}}$$

Proof (contd.)

- $l = \lceil \log n \rceil$
- $\mathcal{I}$ is the set of all monic irreducible polynomials of degree between $1 + \frac{l}{2}$ and l over $\mathbb{F}_p$
- $I(d)$ be the number of monic irreducible polynomials of degree d over $\mathbb{F}_p$
- By the distribution theorem of irreducible polynomials[5],
$$\frac{p^k}{k} - p^{\frac{k}{2}} \leq I(d) \leq \frac{p^k}{k} + p^{\frac{k}{2}}$$
- For $f \in \mathcal{I}$, C_f is the set of l degree polynomials that have f as a factor.

Proof (contd.)

- $|C_f| = p^{l-deg(f)}$

Proof (contd.)

- $|C_f| = p^{l-deg(f)}$
- $C_{f_1} \cap C_{f_1} = \emptyset$

Proof (contd.)

- $|C_f| = p^{l-deg(f)}$
- $C_{f_1} \cap C_{f_1} = \emptyset$

$$\sum_{f \in \mathcal{I}} |C_f| = \sum_{k=1+\frac{l}{2}}^{l} I(k) p^{l-k}$$

$$\geq \sum_{k=1+\frac{l}{2}}^{l} \frac{p^k}{k} p^{l-k} - p^{\frac{k}{2}} p^{l-k}$$

$$= \sum_{k=1+\frac{l}{2}}^{l} p^l \left(\frac{1}{k} - \frac{1}{p^{\frac{k}{2}}} \right)$$

$$= p^l \sum_{k=1+\frac{l}{2}}^{l} \left(\frac{1}{k} - \frac{1}{p^{k-2}} \right) \geq \left(\ln 2 - \frac{1}{48} \right) p^l$$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$
- The number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$
- The number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$
- The total number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$
- The number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$
- The total number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$
- The total number of monic polynomials with degree l that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$
- The number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$
- The total number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$
- The total number of monic polynomials with degree l that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$
- The probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) \geq (\ln 2 - \frac{1}{48} - \frac{1}{8nl}) > \frac{2}{3}$

Proof (contd.)

- The total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 - \frac{1}{48}\right) p^l$.
- $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$
- The number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$
- The number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$
- The total number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$
- The total number of monic polynomials with degree l that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$
- The probability that a monic polynomial p with $deg(p) = l$ does not divide $P_n(x) \geq (\ln 2 - \frac{1}{48} - \frac{1}{8nl}) > \frac{2}{3}$
- The output is COMPOSITE with probability $> \frac{2}{3}$

Running Time Analysis

- n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 9 takes $O(\log^2 n)$ time

Running Time Analysis

- n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 9 takes $O(\log^2 n)$ time
- In Step 14, the algorithm does $O(\log n)$ multiplications of two degree $O(\log n)$ polynomials over $\mathbb{Z}_n$ and computes same number of remainders modulo a third degree $O(\log n)$ polynomial and each of these requires $O^{\sim}(\log^3 n)$

Running Time Analysis

- n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and thus, Step 9 takes $O(\log^2 n)$ time
- In Step 14, the algorithm does $O(\log n)$ multiplications of two degree $O(\log n)$ polynomials over $\mathbb{Z}_n$ and computes same number of remainders modulo a third degree $O(\log n)$ polynomial and each of these requires $O^{\sim}(\log^3 n)$
- Since these are the only two non trivial steps in the algorithm, the time complexity of the algorithm is $O^{\sim}(\log^4 n)$

Moving towards the AKS Algorithm

1. The AKS algorithm is a derandomisation this algorithm.

Moving towards the AKS Algorithm

1. The AKS algorithm is a derandomisation this algorithm.
2. S is the set of all monic polynomials of degree $\lceil \log n \rceil$ in $\mathbb{Z}_n$ and if n is composite then there are at most $|S|/3$ many $Q(x)$'s in S such that $P(x) = 0 \mod Q(x)$.

Moving towards the AKS Algorithm

1. The AKS algorithm is a derandomisation this algorithm.
2. S is the set of all monic polynomials of degree $\lceil \log n \rceil$ in $\mathbb{Z}_n$ and if n is composite then there are at most $|S|/3$ many $Q(x)$'s in S such that $P(x) = 0 \mod Q(x)$.
3. A naive approach to derandomize this algorithm is to check whether $P(x) = 0 \mod Q(x)$ for more than $|S|/3$ many $Q(x)$. But this will take $O(|S|)$ time and $|S| = O(n^{\log n})$.

Moving towards the AKS Algorithm

1. The AKS algorithm is a derandomisation this algorithm.
2. S is the set of all monic polynomials of degree $\lceil \log n \rceil$ in $\mathbb{Z}_n$ and if n is composite then there are at most $|S|/3$ many $Q(x)$'s in S such that $P(x) = 0 \mod Q(x)$.
3. A naive approach to derandomize this algorithm is to check whether $P(x) = 0 \mod Q(x)$ for more than $|S|/3$ many $Q(x)$. But this will take $O(|S|)$ time and $|S| = O(n^{\log n})$.
4. The AKS algorithm implies that, we can compute a $O(\log^5 n)$ integer r and a $O(\log^6 n)$ integer l s.t. if n is composite then there exists an integer a between 1 and l s.t. $P(x) \neq 0 \mod (x + a)^r - 1$.

Moving towards the AKS Algorithm

1. The AKS algorithm is a derandomisation this algorithm.
2. S is the set of all monic polynomials of degree $\lceil \log n \rceil$ in $\mathbb{Z}_n$ and if n is composite then there are at most $|S|/3$ many $Q(x)$'s in S such that $P(x) = 0 \mod Q(x)$.
3. A naive approach to derandomize this algorithm is to check whether $P(x) = 0 \mod Q(x)$ for more than $|S|/3$ many $Q(x)$. But this will take $O(|S|)$ time and $|S| = O(n^{\log n})$.
4. The AKS algorithm implies that, we can compute a $O(\log^5 n)$ integer r and a $O(\log^6 n)$ integer l s.t. if n is composite then there exists an integer a between 1 and l s.t. $P(x) \neq 0 \mod (x+a)^r - 1$.
5. The AKS algorithm does not use exactly this test. It checks whether for all a between 1 and l, $(x-a)^n = (x^n - a)$ over $\mathbb{Z}_n[x]/(x^r - 1)$.

The following lemma shows that these two tests are equivalent.

Lemma

Fix any $r > 0$ and any $l > 0$. Then,

$$(x+1)^n = (x^n+1) \pmod{n, (x+a)^r - 1} \text{ for } 1 \leq a \leq l \quad (2)$$

if and only if

$$(x-a)^n = (x^n-a) \pmod{n, x^r - 1} \text{ for } 1 \leq a \leq l \quad (3)$$

This can be easily proved using induction on l.

The Algorithm

1. Determine whether the input is an integral power
2. Determine whether the input has a small prime divisor
3. Check whether $(x + a)^n = (x^n + a) \pmod{n, x^r - 1}$.

The Algorithm

1. Determine whether the input is an integral power
2. Determine whether the input has a small prime divisor
3. Check whether $(x+a)^n = (x^n + a) \pmod{n, x^r - 1}$.

The third step is equivalent to the condition that for all $Q(x) \in S = \{(x+a)^r - 1 \mid a = 1, \ldots, l\}$, $(x+1)^n - (x^n + 1)$ is zero modulo $Q(x)$.

Algorithm 6 AKS Primality Test

procedure AKS(n)
  if isPower(n) **then**
    return COMPOSITE
  end if
  Find the smallest r such that $o_r(n) > 4\log^2 n$
  Set $l = o_r(n) - 1$ //$o_r(n)$ is the order of n modulo r
  if $1 < gcd(a, n) < n$ for any $a \in \{1, 2, ..., r\}$ **then**
    return COMPOSITE
  end if
  if $n \leq r$ **then**
    return PRIME
  end if
  for $a = 1$ **to** l **do**
    if $(x + a)^n \neq (x^n + a) \pmod{n, x^r - 1}$ **then**
      return COMPOSITE
    end if
  end for
  return PRIME
end procedure

Correctness

- If n is prime, then it can be trivially shown that the output is always correct

Correctness

- If n is prime, then it can be trivially shown that the output is always correct
- For the converse, if AKS(n) returns PRIME in step 11, then it is easy to show that n is prime

Correctness

- If n is prime, then it can be trivially shown that the output is always correct
- For the converse, if AKS(n) returns PRIME in step 11, then it is easy to show that n is prime
- If AKS(n) returns PRIME in step 18, then we assume that n is composite and that it has a non-trivial prime factor p

Correctness

- If n is prime, then it can be trivially shown that the output is always correct
- For the converse, if AKS(n) returns PRIME in step 11, then it is easy to show that n is prime
- If AKS(n) returns PRIME in step 18, then we assume that n is composite and that it has a non-trivial prime factor p
- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial

Correctness

- If n is prime, then it can be trivially shown that the output is always correct
- For the converse, if AKS(n) returns PRIME in step 11, then it is easy to show that n is prime
- If AKS(n) returns PRIME in step 18, then we assume that n is composite and that it has a non-trivial prime factor p
- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- Consider $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$, which is a subgroup of F^* where $l = o_r(n) - 1$

Correctness

- If n is prime, then it can be trivially shown that the output is always correct
- For the converse, if AKS(n) returns PRIME in step 11, then it is easy to show that n is prime
- If AKS(n) returns PRIME in step 18, then we assume that n is composite and that it has a non-trivial prime factor p
- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- Consider $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$, which is a subgroup of F^* where $l = o_r(n) - 1$
- Contradictory bounds are derived for $\mid \mathcal{G} \mid$

Theorem

If n is prime, AKS(n) returns PRIME.

Proof.

Suppose n is prime.

- $n \neq p^k$ for any $k > 1$ and p prime and so, COMPOSITE cannot be returned by AKS(n) at step 3

Theorem

If n is prime, AKS(n) returns PRIME.

Proof.

Suppose n is prime.

- $n \neq p^k$ for any $k > 1$ and p prime and so, COMPOSITE cannot be returned by AKS(n) at step 3
- $\mathrm{GCD}(a, b)$ is 1 if $n \nmid a$ and n if $n \mid a$, and so COMPOSITE cannot be returned by AKS(n) at step 8

Theorem

If n is prime, AKS(n) returns PRIME.

Proof.

Suppose n is prime.

- $n \neq p^k$ for any $k > 1$ and p prime and so, COMPOSITE cannot be returned by AKS(n) at step 3
- $\mathrm{GCD}(a, b)$ is 1 if $n \nmid a$ and n if $n \mid a$, and so COMPOSITE cannot be returned by AKS(n) at step 8
- $(a + x)^n = (a + x^n) \pmod{n}$ $\forall a$ and so $(a + x)^n = a + x^n \pmod{n, x^r - 1}$ $\forall a \in \{1, 2, ..., l\}$. Thus, COMPOSITE cannot be returned by AKS(n) at step 15

Theorem

If n is prime, AKS(n) returns PRIME.

Proof.

Suppose n is prime.

- $n \neq p^k$ for any $k > 1$ and p prime and so, COMPOSITE cannot be returned by AKS(n) at step 3
- $\mathrm{GCD}(a, b)$ is 1 if $n \nmid a$ and n if $n \mid a$, and so COMPOSITE cannot be returned by AKS(n) at step 8
- $(a + x)^n = (a + x^n) \pmod{n}$ $\forall a$ and so $(a + x)^n = a + x^n \pmod{n, x^r - 1}$ $\forall a \in \{1, 2, ..., l\}$. Thus, COMPOSITE cannot be returned by AKS(n) at step 15

Hence, PRIME must be returned by AKS(n). □

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Proof.

Suppose AKS(n) returns prime at step 11.

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Proof.

Suppose AKS(n) returns prime at step 11.

- $n \leq r$

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Proof.

Suppose AKS(n) returns prime at step 11.

- $n \leq r$
- If n were composite, $\exists p < n \leq r$ such that $1 < gcd(p, n) = p < n$

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Proof.

Suppose AKS(n) returns prime at step 11.

- $n \leq r$
- If n were composite, $\exists p < n \leq r$ such that $1 < gcd(p, n) = p < n$
- Then COMPOSITE would have been returned at step 8

Theorem

If AKS(n) returns PRIME in step 11, then n is prime.

Proof.

Suppose AKS(n) returns prime at step 11.

- $n \leq r$
- If n were composite, $\exists p < n \leq r$ such that $1 < gcd(p, n) = p < n$
- Then COMPOSITE would have been returned at step 8
- This is not possible, as the program has reached step 11

□

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$
- $R = \mathbb{Z}_n[x]/< x^r - 1 >$

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$
- $R = \mathbb{Z}_n[x]/< x^r - 1 >$
- l is as defined in the algorithm

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$
- $R = \mathbb{Z}_n[x]/ < x^r - 1 >$
- l is as defined in the algorithm
- $(a + x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$
- $R = \mathbb{Z}_n[x]/< x^r - 1 >$
- l is as defined in the algorithm
- $(a + x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$
- Let n be composite, and p be a prime divisor of n

Theorem

If AKS(n) returns PRIME in step 18, then n is prime.

Proof

Suppose AKS(n) returns PRIME in step 18.

- $n > r$ and $gcd(n, r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$
- $R = \mathbb{Z}_n[x]/< x^r - 1 >$
- l is as defined in the algorithm
- $(a + x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$
- Let n be composite, and p be a prime divisor of n
- $gcd(p, r) = 1$ as $gcd(n, r) = 1$, and thus $n, p \in \mathbb{Z}_r^*$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$
- $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ is a subgroup of F^*

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$
- $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ is a subgroup of F^*
- m is said to be introspective for $f(x) \in F$ if $f(x^m) = f(x)^m$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$
- $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ is a subgroup of F^*
- m is said to be introspective for $f(x) \in F$ if $f(x^m) = f(x)^m$
- m is introspective for f $\forall m \in G$ and $\forall f \in \mathcal{G}$

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$
- $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ is a subgroup of F^*
- m is said to be introspective for $f(x) \in F$ if $f(x^m) = f(x)^m$
- m is introspective for f $\forall m \in G$ and $\forall f \in \mathcal{G}$
- m_1, m_2 are introspective for $f \Rightarrow m_1 m_2$ is introspective for f

Proof (contd.)

- $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial
- $(a+x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$, and so $(a+x)^n = a + x^n$ in F, $\forall a \in \{1, 2, ..., l\}$
- $G =< n, p >$ is a subgroup of $\mathbb{Z}_r^*$, and $t = \mid G \mid$
- $G \leq \mathbb{Z}_r^* \Rightarrow t < r$
- $< n > \leq < n, p > \Rightarrow o_r(n) < t$
- $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ is a subgroup of F^*
- m is said to be introspective for $f(x) \in F$ if $f(x^m) = f(x)^m$
- m is introspective for f $\forall m \in G$ and $\forall f \in \mathcal{G}$
- m_1, m_2 are introspective for $f \Rightarrow m_1 m_2$ is introspective for f
- m is introspective for $f, g \Rightarrow m$ is introspective for fg

Proof (contd.)

Claim: $| \mathcal{G} | > 2^{t-1}$

Proof (contd.)

Claim: $|\mathcal{G}| > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$

Proof (contd.)

Claim: $|\mathcal{G}| > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots

Proof (contd.)

Claim: $|\mathcal{G}| > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$

Proof (contd.)

Claim: $|\mathcal{G}| > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$
- In F, x is the r^{th} root of unity, ζ_r, and so, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m$ $\forall m \in G$

Proof (contd.)

Claim: $| \mathcal{G} | > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$
- In F, x is the r^{th} root of unity, ζ_r, and so, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m$ $\forall m \in G$
- ζ_r^m is a root for $g = f_{K_1} - f_{K_2}$ $\forall m \in G$ where g is a polynomial of degree $< t - 1$ and $| G | = t$

Proof (contd.)

Claim: $| \mathcal{G} | > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$
- In F, x is the r^{th} root of unity, ζ_r, and so, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m$ $\forall m \in G$
- ζ_r^m is a root for $g = f_{K_1} - f_{K_2}$ $\forall m \in G$ where g is a polynomial of degree $< t - 1$ and $| G | = t$
- This is not possible unless $f_{k_1} = f_{k_2}$ which is a contradiction

Proof (contd.)

Claim: $| \mathcal{G} | > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$
- In F, x is the r^{th} root of unity, ζ_r, and so, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m$ $\forall m \in G$
- ζ_r^m is a root for $g = f_{K_1} - f_{K_2}$ $\forall m \in G$ where g is a polynomial of degree $< t - 1$ and $| G | = t$
- This is not possible unless $f_{k_1} = f_{k_2}$ which is a contradiction
- $f_{K_1} \neq f_{K_2}$ in F for $K_1 \neq K_2$ and hence in $\mathcal{G}$

Proof (contd.)

Claim: $| \mathcal{G} | > 2^{t-1}$

- For each $K \subseteq \{1, 2, ..., l\}$, consider $f_K(x) = \prod_{a \in K}(x - a) \in \mathbb{Z}[x]$
- Each f_K is distinct since they have distinct roots
- There are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$
- In F, x is the r^{th} root of unity, ζ_r, and so, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m$ $\forall m \in G$
- ζ_r^m is a root for $g = f_{K_1} - f_{K_2}$ $\forall m \in G$ where g is a polynomial of degree $< t - 1$ and $| G | = t$
- This is not possible unless $f_{k_1} = f_{k_2}$ which is a contradiction
- $f_{K_1} \neq f_{K_2}$ in F for $K_1 \neq K_2$ and hence in $\mathcal{G}$
- $| \mathcal{G} | > 2^{t-1}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t} + 1)^2 > t$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t} + 1)^2 > t$
- Considering the elements of S modulo r, they become elements of G

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t} + 1)^2 > t$
- Considering the elements of S modulo r, they become elements of G
- Since $| G | = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod{r}$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t}+1)^2 > t$
- Considering the elements of S modulo r, they become elements of G
- Since $| G | = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod{r}$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$
- $\forall f \in \mathcal{G}$, $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2+rk}) = f(x^{m_2}x^{rk}) = f(x^{m_2}) = f(x)^{m_2}$ as $x^r = 1$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t}+1)^2 > t$
- Considering the elements of S modulo r, they become elements of G
- Since $| G | = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod{r}$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$
- $\forall f \in \mathcal{G}$, $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2+rk}) = f(x^{m_2}x^{rk}) = f(x^{m_2}) = f(x)^{m_2}$ as $x^r = 1$
- $f(x)$ is a root of $g = x^{m_1} - x^{m_2}$ $\forall f \in \mathcal{G}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t}+1)^2 > t$
- Considering the elements of S modulo r, they become elements of G
- Since $| G | = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod{r}$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$
- $\forall f \in \mathcal{G}$, $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2+rk}) = f(x^{m_2}x^{rk}) = f(x^{m_2}) = f(x)^{m_2}$ as $x^r = 1$
- $f(x)$ is a root of $g = x^{m_1} - x^{m_2}$ $\forall f \in \mathcal{G}$
- $deg(g) = max\{m_1, m_2\} = n^{\sqrt{t}} p^{\sqrt{2}} < n^{\sqrt{t}} n^{\sqrt{t}} = n^{2\sqrt{t}}$

Proof (contd.)

Claim: $| \mathcal{G} | \leq n^{2\sqrt{t}}$

- $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$
- If n is not a prime power, then each $n^i p^j$ is distinct, and hence $| S | = (\sqrt{t}+1)^2 > t$
- Considering the elements of S modulo r, they become elements of G
- Since $| G | = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod{r}$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$
- $\forall f \in \mathcal{G}$, $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2+rk}) = f(x^{m_2}x^{rk}) = f(x^{m_2}) = f(x)^{m_2}$ as $x^r = 1$
- $f(x)$ is a root of $g = x^{m_1} - x^{m_2}$ $\forall f \in \mathcal{G}$
- $deg(g) = max\{m_1, m_2\} = n^{\sqrt{t}} p^{\sqrt{2}} < n^{\sqrt{t}} n^{\sqrt{t}} = n^{2\sqrt{t}}$
- The number of roots of $g = n^{2\sqrt{t}}$ and so $| \mathcal{G} | \leq n^{2\sqrt{t}}$

Proof (contd.)

- $2^{t-1} \leq | \mathcal{G} | \leq n^{2\sqrt{t}}$

Proof (contd.)

- $2^{t-1} \leq |\mathcal{G}| \leq n^{2\sqrt{t}}$
- As $t > 4\log^2 n$, the bounds for $|\mathcal{G}|$ are contradictory

Proof (contd.)

- $2^{t-1} \leq |\mathcal{G}| \leq n^{2\sqrt{t}}$
- As $t > 4\log^2 n$, the bounds for $|\mathcal{G}|$ are contradictory

We thus have the required contradiction if we assume n to be composite.

Proof (contd.)

- $2^{t-1} \leq |\mathcal{G}| \leq n^{2\sqrt{t}}$
- As $t > 4\log^2 n$, the bounds for $|\mathcal{G}|$ are contradictory

We thus have the required contradiction if we assume n to be composite.

Running Time Analysis

Before we start the running time analysis, we need the following lemma:

Lemma

Let LCM(m) denote the lcm of the first m numbers. For m odd, LCM(m) $\geq 2^{m-1}$.

Running Time Analysis

Before we start the running time analysis, we need the following lemma:

Lemma

Let LCM(m) denote the lcm of the first m numbers. For m odd, LCM(m) $\geq 2^{m-1}$.

Using this, we can show the following result.

Lemma

There exists an $r \leq 16 \lg^5 n$, *such that* $o_r(n) > 4 \lg^2 n$.

Running Time Analysis (contd.)

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time

Running Time Analysis (contd.)

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod{r}$ for every $k \leq 4\log^2 n$

Running Time Analysis (contd.)

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod{r}$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$

Running Time Analysis (contd.)

- If n is the input, $\text{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$

Running Time Analysis (contd.)

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$
- Each gcd computation takes time $O(\log^2 n)$ and since r is of $O(\log^5 n)$, Step 8 will also be taking only time $O(\log^7 n)$

Running Time Analysis (contd.)

- If n is the input, $\textsc{isPower}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$
- Each gcd computation takes time $O(\log^2 n)$ and since r is of $O(\log^5 n)$, Step 8 will also be taking only time $O(\log^7 n)$
- We have to verify l equations where $l = o_r(n) - 1 < o_r(n)$ which divides $\phi(r)$ and hence, $l < \phi(r) < r$ is $O(\log^5 n)$

Running Time Analysis (contd.)

- If n is the input, $\text{ISPOWER}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$
- Each gcd computation takes time $O(\log^2 n)$ and since r is of $O(\log^5 n)$, Step 8 will also be taking only time $O(\log^7 n)$
- We have to verify l equations where $l = o_r(n) - 1 < o_r(n)$ which divides $\phi(r)$ and hence, $l < \phi(r) < r$ is $O(\log^5 n)$
- Each equation takes $O(\log n)$ multipications of r degree

Running Time Analysis (contd.)

- If n is the input, $\text{ISPOWER}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$
- Each gcd computation takes time $O(\log^2 n)$ and since r is of $O(\log^5 n)$, Step 8 will also be taking only time $O(\log^7 n)$
- We have to verify l equations where $l = o_r(n) - 1 < o_r(n)$ which divides $\phi(r)$ and hence, $l < \phi(r) < r$ is $O(\log^5 n)$
- Each equation takes $O(\log n)$ multipications of r degree

Running Time Analysis (contd.)

- If n is the input, $\text{ISPOWER}(n)$ takes $O(\log^2 n)$ time and so, step 3 takes $O(\log^2 n)$ time
- To find an r such that $o_r(n) > 4\log^2 n$, successive values of r are tried and tested whether $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$
- For a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$
- Only $O(\log^5 n)$ different r's need to be checked, and so the total complexity of this step is polynomial in $\log n$
- Each gcd computation takes time $O(\log^2 n)$ and since r is of $O(\log^5 n)$, Step 8 will also be taking only time $O(\log^7 n)$
- We have to verify l equations where $l = o_r(n) - 1 < o_r(n)$ which divides $\phi(r)$ and hence, $l < \phi(r) < r$ is $O(\log^5 n)$
- Each equation takes $O(\log n)$ multipications of r degree


Manindra Agrawal and Somenath Biswas.
Primality and identity testing via chinese remaindering.
Journal of the ACM (JACM), 50(4):429–443, 2003.


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


Leiserson Coremen.
Rivest, stein introduction to algorithm.
PHI publication.


David Steven Dummit and Richard M Foote.
Abstract algebra, volume 1984.
Wiley Hoboken, 2004.


Rudolf Lidl and Harald Niederreiter.
Introduction to finite fields and their applications.
Cambridge University Prepp, 1986.

Gary L Miller.

Riemann's hypothesis and tests for primality.

Journal of computer and system sciences, 13(3):300–317, 1976.

Michael O Rabin.

Probabilistic algorithm for testing primality.

Journal of number theory, 12(1):128–138, 1980.

Thankyou