PRIMALITY TESTING ALGORITHMS

A Project Report Submitted

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

in

Mathematics and Computing

by

PRERONA CHATTERJEE

(Roll No. 142123029)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

April 2016

CERTIFICATE

This is to certify that the work contained in this report entitled

"PRIMALITY TESTING ALGORITHMS"

submitted by **Prerona Chatterjee** (**Roll No: 142123029**)

to Department of Mathematics, Indian Institute of Technology Guwahati

towards the requirement of the course **MA699 Project** has been carried

out by her under my supervision.

Guwahati - 781 039 (Dr. Sagarmoy Dutta)

April 2016 Project Supervisor

ABSTRACT

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. Of special interest are 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.

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.

Contents

1 Introduction **1**

1.1 The Problem 2

1.2 The Core Idea Behind the Algorithms 2

1.3 Outline of the Rest of Report 4

2 Preliminaries **5**

2.1 Groups, Rings and Fields 5

2.2 Some Basic Algorithms 8

3 Miller-Rabin Primality Testing Algorithm **10**

3.1 Key Concepts 10

3.2 The Algorithm 11

3.3 Correctness 13

3.4 Running Time Analysis 14

4 Moving towards the AKS Algorithm **15**

4.1 The Key Concept 15

4.2 The Algorithm 17

4.3 Correctness 17

4.4 Running Time Analysis 21
4.5 Link to the AKS Algorithm 21

5 The AKS Algorithm 25
5.1 The Algorithm 25
5.2 Correctness 27
5.3 Running Time Analysis 29

6 Conclusion 32

Bibliography 34

List of Algorithms

1 Check if a Number is an Integral Power 9
2 Compute GCD of two integers 9
3 Modular Exponentiation: Given integers a, b, n compute a^b mod n 9
4 Miller-Rabin Primality Test 12
5 Agrawal-Biswas Primality Test 18
6 AKS Primality Test 26

List of Notations

Symbol	Meaning
$\mathbb{Z}$	Ring of integers
$\mathbb{Z}_n$	Ring of integers modulo n
$\mathbb{Z}_n^*$	Group of units of $\mathbb{Z}_n$
$R[x]$	Polynomial ring over ring R
$R[x]/f(x)$	Ring obtained by quotienting $R[x]$ with principal ideal generated by $f(x)$
$\mathbb{F}_q$	Field of order q
$o_r(n)$	order of n in group $\mathbb{Z}_r$
ln	Natural logarithm
lg	Logarithm base 2
$\phi(n)$	The number of integers less than n which are co-prime to n

Chapter 1

Introduction

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. A number of non-deterministic polynomial time algorithms for the problem have been known for a long time [Miller [6]; Rabin [7]]. All these algorithms are based on Fermat's little theorem and differ in their treatment to handle the Fermat's pseudoprimes. In [1] Biswas and Agrawal introduced a new technique, which generalises the idea of Fermat's little theorem to a similar identity over polynomial rings. This resulted in a simple probabilistic polynomial time algorithm which completely bypasses the issue of pseudoprimes. Finally, in 2004 [2], Manindra Agrawal, Neeraj Kayal and Nitin Saxena, gave a deterinistic polynomial time algorithm by derandomizing the previous algorithm. In this project, our main focus is to study the AKS algorithm along with two of its precursers, namely, the Miller-Rabin test and Agrawal-Biswas test.

1.1 The Problem

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 $\log(n)$. Formally, the problem stated as follows:

Problem 1. 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.

1.2 The Core Idea Behind the Algorithms

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

Informally, the core idea behind all these algorithms can be stated as follows.

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).

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.

In Miller-Rabin test $R_n = \mathbb{Z}_n$, $S = \mathbb{Z}_n^*$ and $a_1, \ldots, a_l$ are randomly chosen elements from $\mathbb{Z}_n^*$. And 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 $a \in \mathbb{Z}_n^*$, $a^{n-1} = 1$.

In Agrawal-Biswas test $R_n = \mathbb{Z}_n[x]$ and they generalise Fermat's little theorem to the following: $(x + a)^n = x^n + a$ if and only if n is prime (where a is any element from $\mathbb{Z}_n^*$). They choose S as the set of all monic polynomial of degree $\lceil \log n \rceil$. $Q(x)$ is randomly chosen from S and it is checked whether $(x + 1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$.

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, with high probability, n is prime. 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.

In AKS test $R_n = \mathbb{Z}_n/(x^r - 1)$ where $r = O(\log^5 n)$. And it is checked whether $(x+a)^n = x^n + a$, in R_n, for $a = 1, 2, \ldots, l$ where $l = O(\log^6 n)$. This is equivalent (see theorem 4.5.1) to the condition: 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 $\log^6 n$. It allows us to check the "identity" exhaustively for all the elements in S which leads to the deterministic algorithm.

1.3 Outline of the Rest of Report

Chapter 2 recapitulates some standard results in algebra and some number theoretic algorithms which are used later. In Chapter 3 we cover the Miller-Rabin Primality Test which is still the most widely used Primality Testing algorithm. In Chapter 4 we analyse the algorithm given by M. Agrawal and S. Biswas which was a crucial intermediate step towards the AKS algorithm. In Chapter 5, we cover the AKS Algorithm which proved that primality testing is in P. And finally in Chapter 6, we briefly overview what we have covered in the project and discuss what further work can be done in this area.

Chapter 2

Preliminaries

In this section we state some of the very basic definitions and results about algebra and algorithms which will later be used. Readers are assumed to be already familiar with preliminary algebra and hence most of the results are stated without proof which can be found in standard undergraduate level textbooks [4].

2.1 Groups, Rings and Fields

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 2.1.1. *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 $\forall a, b \in H$, $ab \in H$.*

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 $a + rR \in R/rR$.

Proposition 2.1.2. *If R is a euclidean domain then,*

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 $R[x]/f(x)R[x]$.

From Proposition 2.1.2.2 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 2.1.3 (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 2.1.2.1, $\mathbb{F}_p^* = \mathbb{F}_p \backslash \{0\}$ and $|\mathbb{F}_p^*| = p-1$. By Proposition 2.1.1.2, for all $a \in \mathbb{F}_p^*$, $a^{|\mathbb{F}_p^*|} = 1$. $\square$

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 $b, d \in R_2$.

Theorem 2.1.4. *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:*

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

Proof. By definition, ϕ is a homomorphism. Notice that if we identify $\mathbb{Z}_m$ with the set of integers $\{0, \ldots, m-1\}$ then we can view ϕ as a map from $\{0, \ldots, n-1\}$ to $\{0, \ldots, n_1 - 1\} \times \{0, \ldots, n_2 - 1\}$ and $\phi(x) = (x_1, x_2)$ s.t. $x = x_i \bmod n_i$ (for $i = 1, 2$).

Now let $(x_1, x_2) \in \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. Since $gcd(n_1, n_2) = 1$, by Proposition 2.1.2.1, there exists $(x_2, x_1) \in \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ s.t. $n_1 x_1 = 1 \bmod n_2$ and $n_2 x_2 = 1 \bmod n_1$. Given any element $(a, b) \in \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$ if we choose $x = a n_2 x_2 + b n_1 x_1 \bmod n$, then $\phi(x) = (a, b)$ which implies that ϕ is surjective.

Finally, ϕ is injective as $\phi(x) = \phi(y)$ implies $\phi(x - y) = (0, 0)$ or $x - y$ is divisible by both n_1 and n_2. Since n_1 and n_2 are co-prime, n divides $x - y$ implying $x = y$ in $\mathbb{Z}_n$. $\square$

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

Theorem 2.1.5. *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.*

2.2 Some Basic Algorithms

Here we give some well-known number theoretic algorithims. 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 $\log n$.

isPower(n) checks whether the integer n can be expressed as x^k where x, k are integers and $k > 2$. gcd(a, b) computes the gcd of two integers using Euclid's method.

Step 3 of isPower(n) can be done in $O(\log n)$ time and hence the total time complexity is $O(\log^2 n)$. If a and b are $O(n)$, it can be shown that the while loop in gcd(a, b) iterates $O(\log n)$ times.

In computation number theory we are often required to compute $a^b \pmod{n}$ where a, b, n are integers such that a and b are $O(n)$. 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. Mod-Exp(a, b, n) gives an algorithm which takes time polynomial in $\log n$ for modular exponentiation.

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. $x^k \leq n$
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
```

Chapter 3

Miller-Rabin Primality Testing Algorithm

The Miller Rabin Primality Test is a probabilistic algorithm which determines whether a given number is prime. Its original version, due to Gary L. Miller[6], is deterministic, but the determinism relies on the unproven Extended Riemann hypothesis. Michael O. Rabin[7] later modified it to obtain an unconditional probabilistic algorithm.

3.1 Key Concepts

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

$$a^{n-1} = 1 \ \forall a \in \mathbb{Z}_n^* \tag{3.1}$$

However, there are composite numbers called 'Fermat's pseudoprimes' for which equation 3.1 is also true.
There are 2 key concepts behind the algorithm.

Firstly, if n is not a pseudoprime then the converse of equation 3.1 is almost true. In particular, for at least half of the elements a in $\mathbb{Z}_n^*$ equation 3.1 does not hold.

Secondly, 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.

3.2 The Algorithm

Let n be the given number which we have to test for primality. First we rewrite $n-1$ as $2^t u$. Next, we pick some a in the range $\{1, 2, ..., n-1\}$ and 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}. We then 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.

A small procedure realizing the above algorithm is given below:

Algorithm 4 Miller-Rabin Primality Test

```
procedure MillerRabin(n)
    if isPower($n$) or $n$ is even with $n \neq 2$ then
        return COMPOSITE
    end if
    Choose $a$ to be a random number in the range $\{1, 2, ..., n-1\}$
    if gcd$(a, n) \neq 1$ then
        return COMPOSITE
    end if
    Find $t$, $u$ such that $(n-1) = 2^t u$ and $u$ is odd
    Put $x_0 =$ Mod-Exp$(a, u, n)$
    for i=1 to t do
        $x_i = x_{i-1}^2 \pmod n$
        if $x_i == 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq -1$ then
            return COMPOSITE
            //Checking for non-trivial square roots of unity
        end if
    end for
    if $x_t \neq 1$ then
        return COMPOSITE //Checking Fermat's Condition
    end if
    return PRIME
end procedure
```

3.3 Correctness

We now analyse the correctness of the above algorithm. We see that if n is prime, then it can be trivially shown that the output is always correct. However, if n is composite, the analysis becomes a little tougher. We divide the analysis into two cases but in both of them, we basically try to find a proper subgroup of $\mathbb{Z}_n^*$ which contains all the elements for which 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. This is formalised in the following:

Theorem 3.3.1. *If n is prime,* MillerRabin(n) *outputs PRIME with probability one, and if n is composite, its probability of error $\leq \frac{1}{2}$.*

Proof. As explained in Section 3.1, if n is prime, then the output of the procedure MillerRabin(n) is always PRIME.
On the other hand, if n is composite, we have the following two cases:
Case 1: $\exists a \in \mathbb{Z}_n^*$ such that $a^{n-1} \neq 1 \pmod{n}$
We first note that any 'a' chosen in line 8 is in $\mathbb{Z}_n^*$.
Define $B = \{x \in \mathbb{Z}_n^* : x^{n-1} = 1 \pmod{n}\}$.
Then, $\forall a, b \in B, (ab)^{n-1} = (a^{n-1})(b^{n-1}) = 1 \pmod{n}$ and thus B is a subgroup of $\mathbb{Z}_n^*$ by property 2.1.1.3, which is nonempty as $1 \in B$.
However, by the condition of the case, $\exists a \in \mathbb{Z}_n^*$ such that $a \notin B$.
Thus B is a proper subgroup of $\mathbb{Z}_n^*$ and hence, $|B| \leq \frac{|\mathbb{Z}_n^*|}{2}$.
Case 2: $\forall a \in \mathbb{Z}_n^*\ a^{n-1} = 1 \pmod{n}$
Again, we note that any 'a' chosen in line 8 is in $\mathbb{Z}_n^*$ and that for any n such

that the algorithm has not returned before line 8, n is not an integral power.
Define $B = \{x \in \mathbb{Z}_n^* : x^{n-1} = \pm 1 \pmod{n}\}$.
Then, $\forall a, b \in B, (ab)^{n-1} = (a^{n-1})(b^{n-1}) = 1 \pmod{n}$ and thus B is a subgroup of $\mathbb{Z}_n^*$ by property 2.1.1.3 which is nonempty as $1 \in B$.
We now note that as $n \neq x^k$ for any $x, k \in \mathbb{N}$, $\exists n_1, n_2 \in \mathbb{N}$ such that $n = n_1 n_2$ where $gcd(n_1, n_2) = 1$ and hence by Theorem 2.1.4, $\mathbb{Z}_n \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$.
Thus, $\exists a \in \mathbb{Z}_n^* \backslash B$ such that $a \cong (1, -1)$ as $1 \cong (1, 1)$ and $-1 \cong (-1, -1)$ implying that B is a proper subgroup of $\mathbb{Z}_n^*$ and hence, $|B| \leq \frac{|\mathbb{Z}_n^*|}{2}$.

Now, in both cases we find that any 'a' such that PRIME is returned by MillerRabin(n), is a member of B, and so any element in $\mathbb{Z}_n^*$ that is a non-witness to the compositeness of n is a member of B.
So, the probability of error is $\leq \frac{|\mathbb{Z}_n^*|/2}{|\mathbb{Z}_n^*|} = \frac{1}{2}$. $\square$

3.4 Running Time Analysis

If n is the input, we have already seen in Section 2.2, that isPower(n) takes $O(\log^2 n)$ time and thus, Step 2 takes $O(\log^2 n)$ time. Next, 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. Also,Mod-Exp(a, b, n) gives a polynomial time algorithm in $\log n$ for modular exponentiation as we have already seen in Section 2.2 and thus Step 10 takes time polynomial in $\log n$. Finally, since no step in the for loop takes more that polynomial time in $\log n$, and since $t = O(\log n)$, we can easily see that Miller-Rabin(n) requires no more than time polynomial in $\log n$.

Chapter 4

Moving towards the AKS Algorithm

We now move to another simple randomised primality testing algorithm which reduces primality testing for a number n to testing if a specific univariate identity over $\mathbb{Z}_n$ holds. This algorithm is due to Manindra Agrawal and Somenath Biswas [1], and it was the derandomization of this algorithm which finally led to the AKS Algorithm.

4.1 The Key Concept

The key concept used in this algorithm is the following generalisation of Fermat's little theorem over polynomial ring.

Theorem 4.1.1. *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.*

Notice that, unlike Fermat's little theorem, the identity is always false when n is a composite number. This allows us to completely sidestep the issue of pseudoprimes and leads to a very simple algorithm: If $(x+1)^n = (x^n+1) \mod n$ then return PRIME else return COMPOSITE. However the polynomials in both side expands into $O(n)$ term and we need $O(n)$ time to compute them. So, as a compromise, we check the identity modulo a randomly chosen monic polynomial of degree $\log n$ at the cost of introducing one sided error. However 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. We now present the proof of theorem 4.1.1

Proof. (Theorem 4.1.1) Let n be prime. Then by Fermat's little theorem, $a^n = a \pmod n$. Thus, $P_n(x) = (a+x)^n - a - x^n = \sum_{j=1}^{n-1} \binom{n}{j} a^{n-j} x^n$. Note that $\forall j \in \{1, 2, ..., n-1\}$, in $\binom{n}{j}$, n appears in the numerator and since all the numbers appearing in the denominator are less than n, they do not divide n. So, $\forall j \in \{1, 2, ..., n-1\}$, $\binom{n}{j}$ is divisible by n. Thus $P_n(x) = 0$ in $\mathbb{Z}_n[x]$ if n is prime.

Conversely, let n be composite. Then, n has a prime divisor, p such that $n = p^k m$ and $gcd(p, m) = 1$. Consider the coefficient of $x^p = \binom{n}{p} a^{n-p}$. Now $gcd(n, a) = 1 \Rightarrow gcd(p, a) = 1$. So, $n | \binom{n}{p} a^{n-p} \Rightarrow p^k | \binom{n}{p} a^{n-p} \Rightarrow p^k | \binom{n}{p}$. However,

$$\begin{aligned} \binom{n}{p} &= \frac{n(n-1)...(n-p+1)}{p!} \\ &= \frac{p^k m(p^k m - 1)...(p^k m - p + 1)}{p!} \\ &= \frac{p^{k-1} m(p^k m - 1)...(p^k m - p + 1)}{(p-1)!} \end{aligned}$$

Note that $gcd(p, p^k m - i) = 1 \ \forall i \in \{1, 2, ..., p-1\}$.

So, $n \mid \binom{n}{p} \Rightarrow p^k \mid \binom{n}{p} \Rightarrow p^k m \mid p^{k-1} m \Rightarrow\Leftarrow$ as $gcd(p, m) = 1$. Hence, coefficient of $x^p \neq 0 \pmod{n}$ and $p \neq 0, n$, and so, $P_n(x) \neq 0$ in $\mathbb{Z}_n[x]$. $\square$

4.2 The Algorithm

In this algorithm, we first check for some small prime factors of n. If such factors are not found, we choose $Q(x)$ randomly to be monic polynomial of degree $\lceil \log n \rceil$, and check whether $(x+1)^n - (x^n + 1)$ is zero modulo the polynomial $Q(x)$. Using Theorem 4.1.1 with $a = 1$, we see that if that does not happen, then n is definitely composite. Otherwise n is probably prime and we can repeat the process a constant number of times by taking different $Q(x)$'s so as to decrease the probability of error arbitrarily. The pseudocode realising the above algorithm is given in Algorithm 5.

4.3 Correctness

We now analyse the correctness of the above algorithm. As in the case of Miller-Rabin(n), we see that if n is prime, then it can be trivially shown that the output is always correct. However, if n is composite, we want to show that 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 do this, we define a set $\mathcal{I}$ of monic irreducible polynomials having degree in a certain range with upper bound l. For each polynomial f in $\mathcal{I}$, we define a set C_f of degree l polynomials having f as a factor and noting

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
```

that these C_f's are mutually disjoint, we find an lower bound for the number of degree l polynomials with factors in $\mathcal{I}$. We then 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 $P_n(x)$. Finally noting that this also gives a lower bound on the number of monic polynomials of degree l that do not divide $P_n(x)$, we get the required result.

We formalise this as follows:

Theorem 4.3.1. *If n is prime,* 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, by Theorem 4.1.1, $P_n(x) = 0$ in $\mathbb{Z}_n[x]$. Thus, for any $Q(x)$ chosen, $Q(x) \mid P_n(x)$, and so the output of the algorithm is PRIME.

Now, if n is composite, $P_n(x) \neq 0$ in $\mathbb{Z}_n[x]$.
We, however note that the algorithm is correct when n is a prime power or when divisible by primes upto 13. Thus we only have to analyse when n is odd, not a prime power and its every prime factor is atleast 17.
Let $l = \lceil \log n \rceil$ and $\mathcal{I}$ be the set of all monic irreducible polynomials of degree between $1 + \frac{l}{2}$ and l over $\mathbb{F}_p$.
Next, let $I(d)$ be the number of monic irreducible polynomials of degree d over $\mathbb{F}_p$. We note that by the distribution theorem of irreducible polynomials[5],

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

Finally, for $f \in \mathcal{I}$, let C_f be the set of l degree polynomials that have f as a factor. Now, $g \in C_f \Rightarrow g = fq$ for some polynomial q with $deg(q) =$

$l - deg(f)$ and the number of polynomials in $\mathbb{F}_p$ with degree $(l - deg(f))$ is $p^{l-deg(f)}$. Thus, $|C_f| = p^{l-deg(f)}$.
Also, $f \in C_{f_1} \cap C_{f_1} \Rightarrow f_1|f$ and $f_2|f \Rightarrow lcm\{f_1, f_2\}|f$.
But, $gcd\{f_1, f_2\} = 1$ as f_1, f_2 are irreducible $\Rightarrow lcm\{f_1, f_2\} = f_1 f_2$
$\Rightarrow f_1 f_2 | f \Rightarrow\Leftarrow$ (as, $deg(f_1 f_2) = deg(f_1) + deg(f_2) > \frac{l}{2} + \frac{l}{2} = l = deg(f)$).
So,

$$\begin{aligned}
\sum_{f \in \mathcal{I}} |C_f| &= \sum_{k=1+\frac{l}{2}}^{l} I(k) p^{l-k} \\
&\geq \sum_{k=1+\frac{l}{2}}^{l} \frac{p^k}{k} p^{l-k} - p^{\frac{k}{2}} p^{l-k} \\
&= \sum_{k=1+\frac{l}{2}}^{l} p^l \left(\frac{1}{k} - \frac{1}{p^{\frac{k}{2}}} \right) \\
&= p^l \sum_{k=1+\frac{l}{2}}^{l} \left(\frac{1}{k} - \frac{1}{p^{k-2}} \right) \geq \left(\ln 2 - \frac{1}{48} \right) p^l
\end{aligned}$$

Also, $|C_f| = p^{l-deg(f)} \leq p^{\frac{l}{2}-1}$ because $deg(f) \geq 1 + \frac{l}{2}$. So, the total number of monic polynomials of degree l with factors in $\mathcal{I} \geq \left(\ln 2 \ - \frac{1}{48}\right) p^l$.
Now the number of monic irreducible polynomials of degree $> \frac{l}{2}$ over $\mathbb{F}_p$ that divide $P_n(x)$ is $< \frac{n}{l/2} = \frac{2n}{l}$ and so the number of monic polynomials of degree l with factors in $\mathcal{I}$ that divide $P_n(x) \leq \left(\frac{2n}{l}\right) p^{\frac{l}{2}-1} \leq \frac{p^l}{8nl}$ (as $p > 16$). Thus, the total number of monic polynomials of degree l with factors in $\mathcal{I}$ that do not divide $P_n(x) \geq \left(\ln 2 \ - \frac{1}{48} - \frac{1}{8nl}\right) p^l$, and so the total number of monic polynomials p with $deg(p) = l$ that do not divide $P_n(x) \geq \left(\ln 2 - \frac{1}{48} - \frac{1}{8nl}\right) p^l$.
Hence the probability that a monic polynomial p with $deg(p) = l$ does not

divide $P_n(x) \geq (\ln 2 - \frac{1}{48} - \frac{1}{8nl}) > \frac{2}{3}$ ($\because n > 16$, $l = \lg n > 4$).

Thus, if n is composite, it outputs COMPOSITE with probability $> \frac{2}{3}$, and so the probability of error $\leq \frac{2}{3}$. $\square$

4.4 Running Time Analysis

If n is the input, we have already seen in Section 2.2, that 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 $O^{\sim}(\log^4 n)$.

4.5 Link to the AKS Algorithm

The AKS algorithm can be seen as a derandomisation this algorithm. Let S be the set of all monic polynomials of degree $\lceil \log n \rceil$ in $\mathbb{Z}_n$. We know that, if n is composite then there are at most $|S|/3$ many $Q(x)$'s in S such that $P(x) = 0 \mod Q(x)$. Hence 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 $|S| = O(n^{\log n})$. 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 $(x+a)^r - 1$. However in AKS the authors did not use exactly this test.

Rather they checked 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 4.5.1. *Fix any $r > 0$ and any $l > 0$. Then,*

$$(x+1)^n = (x^n+1) \pmod{n, (x+a)^r - 1} \text{ for } 1 \le a \le l \tag{4.1}$$

if and only if

$$(x-a)^n = (x^n - a) \pmod{n, x^r - 1} \text{ for } 1 \le a \le l \tag{4.2}$$

Proof. The proof is by induction on l. So, we see first the result for $l = 1$.
Then, equation (4.1) $\equiv (x+1)^n = (x^n+1) \pmod{n, (x+1)^r - 1}$.
Putting $x-1$ for x, we have $x^n = ((x-1)^n + 1) \pmod{n, x^r - 1}$. Thus, $x^n - 1 = (x-1)^n \pmod{n, x^r - 1} \Rightarrow (x-1)^n = (x^n - 1) \pmod{n, x^r - 1}$ which is equation (4.2) with $a = 1$.
Conversely for $l = 1$, equation (4.2) $\equiv (x-1)^n = (x^n - 1) \pmod{n, x^r - 1}$.
So, putting $x+1$ for x, we have $x^n = ((x+1)^n - 1) \pmod{n, (x+1)^r - 1}$.
Thus, $x^n + 1 = (x+1)^n \pmod{n, (x+1)^r - 1} \Rightarrow (x+1)^n = (x^n + 1) \pmod{n, (x+1)^r - 1}$ which is equation (4.1) with $a = 1$.
So, the equivalence hold for $l = 1$.
Now, take $l > 1$ and let the equivalence hold for every $a \le l - 1$.
First, let equation (4.1) hold.
Then, $(x+1)^n = (x^n+1) \pmod{n, (x+a)^r - 1}$ for $1 \le a \le l$. Thus,

$$(x+1)^n = (x^n+1) \pmod{n, (x+l)^r - 1} \tag{4.3}$$

$$\text{and } (x+1)^n = (x^n+1) \pmod{n, (x+a)^r - 1} \text{ for } 1 \leq a \leq l-1. \tag{4.4}$$

Hence, substituting $x - l$ for l in equation (4.3) we have

$$(x-(l-1))^n = ((x-l)^n + 1) \pmod{n, x^r - 1}. \tag{4.5}$$

Also, by our induction hypothesis, equations (4.4) would mean that

$$(x-a)^n = (x^n - a) \pmod{n, x^r - 1} \text{ for } 1 \leq a \leq l-1. \tag{4.6}$$

Thus putting $a = l - 1$ in equation 4.6, and using equation 4.5 we get $x^n - (l-1) = ((x-l)^n + 1) \pmod{n, x^r - 1}$. Thus,
$x^n - l = (x-l)^n \pmod{n, x^r - 1} \Rightarrow (x-l)^n = (x^n - l) \pmod{n, x^r - 1}$.
Thus, $(x-a)^n = (x^n - a) \pmod{n, x^r - 1}$ for $1 \leq a \leq l$.
Conversely, let equation (4.2) hold.
Then $(x-a)^n = (x^n - a) \pmod{n, x^r - 1}$ for $1 \leq a \leq l$. Thus,

$$(x-l)^n = (x^n - l) \pmod{n, x^r - 1} \tag{4.7}$$

$$\text{and } (x-a)^n = (x^n - a) \pmod{n, x^r - 1} \text{ for } 1 \leq a \leq l-1 \tag{4.8}$$

So, substituting $x + 1$ for x in equation 4.7,

$$(x-(l-1))^n = ((x+1)^n - l) \pmod{n, (x+1)^r - 1} \tag{4.9}$$

Also, by our induction hypothesis, equations (4.8) would mean that

$$(x+1)^n = (x^n + 1) \pmod{n, (x+a)^r - 1} \text{ for } 1 \leq a \leq l-1 \tag{4.10}$$

For $a = 1$ in equation (4.10), and using equation 4.9 we get

$$(x - (l-1))^n = (x^n + 1 - l) \pmod{n, (x+1)^r - 1} \tag{4.11}$$

and substituting $x + 1$ for x in equation (4.11),

$$(x - (l-2))^n = ((x+1)^n - (l-1)) \pmod{n, (x+2)^r - 1} \tag{4.12}$$

For $a = 2$ in equation (4.10), and using equation (4.12) we get

$$(x - (l-2))^n = (x^n - (l-2)) \pmod{n, (x+2)^r - 1} \tag{4.13}$$

and substituting $x + 1$ for x in equation (4.13),

$$(x - (l-3))^n = ((x+1)^n - (l-2)) \pmod{n, (x+3)^r - 1} \tag{4.14}$$

Continuing like this, we get
$(x - (l - l))^n = ((x+1)^n - (l - (l-1)))\ (\bmod\ n, (x+l)^r - 1)$.
Thus, $x^n = ((x+1)^n - 1)\ (\bmod\ n, (x+l)^r - 1)$ and so, $x^n + 1 = (x+1)^n$ $(\bmod\ n, (x+l)^r - 1) \Rightarrow (x+1)^n = (x^n + 1)\ (\bmod\ n, (x+l)^r - 1)$.
Hence, $(x+1)^n = (x^n + 1)\ (\bmod\ n, (x+a)^r - 1)$ for $1 \le a \le l$. $\square$

Chapter 5

The AKS Algorithm

Since the beginning of complexity theory in the 1960's - when the notions of complexity theory were formalized and various complexity classes were defined - the problem of primality testing has been investigated intensively. The ultimate goal of this line of research has been to obtain an unconditional deterministic polynomial time algorithm for primality testing.

Despite the impressive progress made, it wasn't before AKS algorithm that this could be acheived. The AKS algorithm gives a deterministic $O(\log^{12} n)$ time algorithm for testing if a number is prime. The algorithm is based on a generalisation of Fermat's Little Theorem to polynomial rings over finite fields (ie. Theorem 4.1.1) and its equivalent formulation (see Lemma 4.5.1).

5.1 The Algorithm

The AKS algorithm can be divided into the following three parts:

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 the crucial part of this algorithm and as we have seen in Theorem 4.5.1, this 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 $Q(x)$. Thus, this is very similar to the Agrawal-Biswas test except for the fact that in the case we have changed the ring to $\mathbb{Z}_n/(x^r - 1)$ for a suitably chosen $r \in O(\log^5 n)$. This reduces the size of S which allows us to check exhaustively whether all the elements in S satisfy the equation.

Algorithm 6 AKS Primality Test

```
1: procedure AKS($n$)
2:     if isPower($n$) then
3:         return COMPOSITE
4:     end if
5:     Find the smallest $r$ such that $o_r(n) > 4\log^2 n$
6:     Set $l = o_r(n) - 1$ //$o_r(n)$ is the order of $n$ modulo $r$
7:     if $1 < gcd(a, n) < n$ for any $a \in \{1, 2, ..., r\}$ then
8:         return COMPOSITE
9:     end if
10:    if $n \leq r$ then
11:        return PRIME
12:    end if
13:    for $a = 1$ to $l$ do
14:        if $(x + a)^n \neq (x^n + a) \pmod{n, x^r - 1}$ then
15:            return COMPOSITE
16:        end if
17:    end for
18:    return PRIME
19: end procedure
```

5.2 Correctness

We now analyse the correctness of the above algorithm. As usual, if n is prime, then it can be trivially shown that the output is always correct.

Theorem 5.2.1. *If n is prime, AKS(n) returns PRIME.*

Proof. Suppose n is prime. Then, $n \neq p^k$ for any $k > 1$ and p prime and so, COMPOSITE cannot be returned by AKS(n) at step 3. Again, $gcd(n, a)$ is 1 if n∤a and n if n|a, and so COMPOSITE cannot be returned by AKS(n) at step 8. Finally, we have seen by Theorem 4.1.1 that $(a + x)^n = (a + x^n) \pmod{n}\ \forall a$ 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). $\square$

For the converse, we break the proof into two parts, of which the first is easy to prove.

Theorem 5.2.2. *If AKS(n) returns PRIME in step 11, then n is prime.*

Proof. Suppose AKS(n) returns prime at step 11. Then, $n \leq r$. So if n were composite, $\exists p < n \leq r$ such that $1 < gcd(p, n) = p < n$. But then COMPOSITE would have been returned at step 8 which is a contradiction as the program has reached step 11. $\square$

For the second part, we assume that n is composite and that it has a non-trivial prime factor p. We then define $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial and consider the subgroup $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ of F^* where $l = o_r(n) - 1$. Some analysis derives contradictory bounds for $\mid \mathcal{G} \mid$, thus getting the required contradiction.

Theorem 5.2.3. *If AKS(n) returns PRIME in step 18, then n is prime.*

Proof. Suppose AKS(n) returns PRIME in step 18. Then, $n > r$ and $gcd(n,r) = 1$ as otherwise COMPOSITE would have been returned at line 8 when $a = r$. Let $R = \mathbb{Z}_n[x]/< x^r - 1 >$ and l be as defined in the algorithm. Then by Theorem 4.1.1, $(a + x)^n = a + x^n$ in R, $\forall a \in \{1, 2, ..., l\}$.
If possible, let n be composite. Then, $\exists$ a prime p such that $p \mid n$. So, $gcd(p,r) = 1$ as $gcd(n,r) = 1$, and thus $n, p \in \mathbb{Z}_r^*$.
Define $F = \mathbb{F}_p[x]/ < h(x) >$ where $h(x)$ is the irreducible part of the r^{th} cyclotomic polynomial. So, $(a+x)^n = a+x^n$ in R, $\forall a \in \{1, 2, ..., l\}$ would mean that $(a+x)^n = a+x^n$ in F, $\forall a \in \{1, 2, ..., l\}$.
Now we have already noted that $n, p \in \mathbb{Z}_r^*$. So, we can consider a subgroup $G =< n, p >$ of $\mathbb{Z}_r^*$, and let $t = \mid G \mid$. We note that $G \leq \mathbb{Z}_r^*$ implies that $t < r$, and $< n > \leq < n, p >$ implies that $o_r(n) < t$.
Next, consider the subgroup $\mathcal{G} =< \{x + a : a \in \{1, 2, ..., l\}\} >$ of F^*.
Define an integer m to be introspective for a polynomial $f(x) \in F$ if $f(x^m) = f(x)^m$. We note that m is introspective for for f $\forall m \in G$ and $\forall f \in \mathcal{G}$. We also note some of the properties satisfied by introspective integers are that m_1, m_2 are introspective for $f \Rightarrow m_1 m_2$ is introspective for f and also, m is introspective for $f, g \Rightarrow m$ is introspective for fg.
<u>Claim</u>: $\mid \mathcal{G} \mid > 2^{t-1}$
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. Further there are $2^l > 2^{t-1}$ of such polynomials since there are 2^l subsets of $1, 2, ..., l$. Note that in F, x is the

r^{th} root of unity, ζ_r. So, $f_{K_1}(x) = f_{K_2}(x)$ in F for $K_1 \neq K_2$ would mean $f_{K_1}(\zeta_r) = f_{K_2}(\zeta_r)$ and so $f_{K_1}(\zeta_r)^m = f_{K_2}(\zeta_r)^m \; \forall m \in G$.
Thus, $f_{K_1}(\zeta_r^m) = f_{K_2}(\zeta_r^m) \; \forall m \in G$ and so, 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 $\mid G \mid= t$ which is not possible unless $f_{k_1} = f_{k_2} \Rightarrow\Leftarrow$.
Thus, $f_{K_1} \neq f_{K_2}$ in F for $K_1 \neq K_2$ and hence in $\mathcal{G}$. So $\mid \mathcal{G} \mid > 2^{t-1}$.
<u>Claim</u>: $\mid \mathcal{G} \mid \leq n^{2\sqrt{t}}$
Consider $S = \{n^i p^j, 0 \leq i, j \leq \sqrt{t}\}$. If n is not a prime power, then each $n^i p^j$ is distinct. Hence $\mid S \mid = (\sqrt{t}+1)^2 > t$. So, when we consider the elements of S modulo r, they become elements of G. Since $\mid G \mid = t$, $\exists m_1, m_2$ with $m_1 \neq m_2$ such that $m_1 = m_2 \pmod r$ and so $m_1 = m_2 + rk$ for some $k \in \mathbb{Z}$.
So, $\forall f \in \mathcal{G}$, $f(x)^{m_1} = f(x^{m_1}) = f(x^{m_2+rk}) = f(x^{m_2}x^{rk}) = f(x^{m_2}) = f(x)^{m_2}$ as $x^r = 1$. So, $f(x)$ is a root of $g = x^{m_1} - x^{m_2} \; \forall f \in \mathcal{G}$.
However, $deg(g) = max\{m_1, m_2\} = n^{\sqrt{t}}p^{\sqrt{2}} < n^{\sqrt{t}}n^{\sqrt{t}} = n^{2\sqrt{t}}$. Thus, the number of roots of $g = n^{2\sqrt{t}}$ and so $\mid \mathcal{G} \mid \leq n^{2\sqrt{t}}$.

Now, we have $2^{t-1} \leq \mid \mathcal{G} \mid \leq n^{2\sqrt{t}}$. However, as $t > 4 \log^2 n$, the bounds for $\mid \mathcal{G} \mid$ are contradictory. We thus have the required contradiction if we assume n to be composite. $\square$

5.3 Running Time Analysis

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

Lemma 5.3.1. *Let LCM(m) denote the lcm of the first m numbers. For m odd, LCM(m)* $\geq 2^{m-1}$.

Proof. Let $m = 2n + 1$. Consider the following integral:

$$\int_0^1 x^n(1+x)^n dx$$

We note that as $x \in [0,1]$, the function $x(1-x)$ attains its maximum at $x = \frac{1}{2}$. Thus $x(1-x) < \frac{1}{4}$, and so the integral is upper bounded by 2^{-2n}. But if we expand $(1-x)^n$ using binomial theorem, then we have

$$\begin{aligned} 2^{-2n} \geq \int_0^1 x^n(1-x)^n &= \int_0^1 \sum_{k=0}^{n}(-1)^k \binom{n}{k} x^{n+k} \\ &= \sum_{k=0}^{n}(-1)^k \binom{n}{k} \int_0^1 x^{n+k} \\ &= \sum_{k=0}^{n}(-1)^k \binom{n}{k} \frac{1}{n+k+1} = \frac{M}{N} \end{aligned}$$

Then clearly N is atmost LCM(m) and M is atleast 1.
Hence, $LCM(m)2^{-2n} > N2^{-2n} \geq 1$ and so $LCM(m) \geq 2^{2n} = 2^{m-1}$ □

Using this, we can show the following result.

Lemma 5.3.2. *There exists an $r \leq 16lg^5 n$, such that $o_r(n) > 4lg^2 n$.*

Proof. Suppose T is such that for all $1 \leq r \leq T$, $o_r(n) \leq 4\lg^2 n$. Then, for every r, there exists a $j < 4\lg^2 n$ such that r divides $n^j - 1$.
Thus, for each $1 \leq r \leq T$, r divides the product (taking $k = 4\lg^2 n$),
$\prod_{j=1}^{k}(n^j - 1) < \prod_{j=1}^{k} n^j = n^{\sum_{i=1}^{k}} = n^{\frac{k(k+1)}{2}} = n^{\frac{k^2+k}{2}} < n^{\frac{k^2+k^2}{2}} = n^{k^2}$.
Thus, the lcm of the first T numbers divides the product which is strictly less than $n^{16\lg^4 n} = 2^{16\lg^5 n}$ ($\because 2^{\lg(n)} = n$). The bound from Lemma 5.3.1 will force $T < 16\lg^5 n$. □

Now, we are ready to start the running time analysis. If n is the input, we have already seen in Section 2.2, that isPower(n) takes $O(\log^2 n)$ time. Next, we must find an r such that $o_r(n) > 4\log^2 n$. This can be done by trying out successive values of r and testing if $n^k \neq 1 \pmod r$ for every $k \leq 4\log^2 n$. So, for a particular r, this will involve at most $O(\log^2 n)$ multipications modulo r and so will take time $O^{\sim}(\log^2 n \log r)$. By Lemma 5.3.2, we know that only $O(\log^5 n)$ different r's need to be checked. Thus the total complexity of this step is polynomial in $\log n$.

Next we have to check the gcd of r numbers. Each gcd computation takes time $O(\log^2 n)$ as we have seen in Section 5.3.2 and since r is of $O(\log^5 n)$, this will also be taking only time $O(\log^7 n)$.

Finally, 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 polynomials with coefficients of size $O(\log n)$. So, each equation can be verified in time $O^{\sim}(r \log^2 n)$ steps. Thus the time taken for this step is $O(r^2 \log^2 n) = O(log^{12} n)$. As we can see, this step dominates all the other steps.

Hence the time complexity for the AKS algorithm is $O(log^{12} n)$ which is polynomial in $\log n$. The time complexity of this algorithm can be improved slightly, the current best being about $O(\log^6 n)$

Chapter 6

Conclusion

In this project, we have covered some of the polynomial time Primality testing Algorithms. Our main goal was the AKS algorithm which we covered in Chapter 5. It gives an unconditional deterministic polynomial time algorithm that determines whether a given input number is prime or not. However before that, in Chapters 3 and 4, we saw two of its precursers, namely, the Miller-Rabin test and Agrawal-Biswas test, which give probablistic polynomial time algorithms to check for primality.

In conclusion, we now look at some related topics of research. We have noted that the the Agrawal-Biswas test reduces primality testing to Polynomial Identity testing which is already active area of research. However, no deterministic polynomial time algorithm has been found for it as yet, even though a probabilistic polynomial time algorithm has been found for polynomials over a field. However if we consider polynomials over a ring, not even a probabilistic polynomial time algorithm is known yet.

Primality testing is a decision problem, and so the most natural way forward would be to find an algorithm which can factorise a given integer n. However, even for this problem no probabilistic polynomial time algorithm is known yet. There is, however, a quantum probabilistic polynomial time algorithm to factorise a given integer.

Bibliography

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

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

[3] Leiserson Coremen. Rivest, stein introduction to algorithm. *PHI publication.*

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

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

[6] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

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