Primality Testing and Factorization

Satender

MS16069

A dissertation submitted for the partial fulfillment
of BS-MS dual degree in Mathematical Science

Indian Institute of Science Education and Research Mohali

May 2021

Certificate of Examination

This is to certify that the dissertation titled "Primality Testing and Factorization" submitted by Mr. Satender (Reg. No. MS16069) for the partial fulfilment of BS-MS dual degree program of the Institute, has been examined by the thesis committee duly appointed by the Institute. The committee finds the work done by the candidate satisfactory and recommends that the report be accepted.

Prof. Kapil Hari Pranjape   Dr. Amit Kulshrestha   Dr. Chandrakant S. Aribam

(Supervisor)

Dated : May 27, 2021

Declaration

The work presented in this dissertation has been carried out by me under the guidance of Dr. Chandrakant S. Aribam at the Indian Institute of Science Education and Research Mohali.
This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgement of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Satender
(Candidate)

Dated: May 27, 2021

ABSTRACT

Primality Testing and
Factorization
by
Satender
Indian Institute of Science Education and Research, Mohali, 2021

This thesis is a detailed study of primality testing and factorization algorithms. In the first part, we study about famous algorithms such as Fermat's factorization scheme, Robin-Miller Test, Solovay-Strassen Test, Continued Fraction Factoring Algorithm, Pollard-rho and $p-1$ test etc., then we study deterministic polynomial time AKS-Algorithm. In, second part we study about Quadratic sieve algorithm and polynomial time lattice reduction algorithm, The LLL-Algorithm. Then we study in detail about polynomial factorization in finite field $\mathbb{Z}/p\mathbb{Z}$ and in field of rationals, $\mathbb{Q}$. In the last part, we study polynomial factorization using the LLL-Algorithm.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

First and foremost I would like to express my special thanks to my thesis supervisor Dr. Chandrakant S. Aribam. It was a great experience working under his guidance throughout the year. I very much appreciate his way of handling students by letting them learn on their own pace in a stress free environment. I thank him for his time and the guidance without which I would not have learned as much.

I am grateful to Professor Kapil Hari Paranjape for his insightful comments and guidance for better implementation of my work. I thank my committee member Dr. Amit Kulshrestha for his valuable feedback.

I am thankful to IISER Mohali library for providing me with academic resources required for my thesis and allowing students to avail library services in these difficult times.

And, I am deeply thankful to my friends and family for their love and support which has always boosted me and kept me motivated to achieve my goals.

NOTATIONS

$\mathbb{F}_p$: Finite field with p number of elements.

$\mathbb{R}$: Field of real numbers.

$\mathbb{Z}$: Ring of integers.

$\lfloor a \rfloor$: Greatest integer less than or equal to a.

$\lceil a \rceil$: Least integer greater than or equal to a.

$\lfloor a \rceil$: Nearest integer to a, also known as round of a.

$\phi(n)$: Number of integer less than or equal to n that are coprime with n.

$gcd(a, b)$: Greatest common divisor of a and b.

$lcm(a, b)$: Least common multiple of a and b

$\mathbb{Z}/n\mathbb{Z}$: Ring of integers modulo n.

$||.||$: Euclidean norm

iff : If and only if.

CHAPTER 1

SOME PRIMALITY TESTING AND FACTORING ALGORITHMS

1.1 SOME TESTS FOR PRIMALITY AND COMPOSITENESS

From Fermat's theorem [6, page 88] we get that if p is a prime, then $a^p \equiv a \ (mod\ p)$ for any integer a. This provides us with a way of recognizing most composite numbers. We need a converse of this theorem which was first given by Edouard Lucas in 1876.

1.1.1 LUCAS AND POCKLINGTON TESTS

Theorem 1.1 (Lucas Test). *If $x^{N-1} \ (mod\ N) = 1$ and $x^{(N-1)/p} \ (mod\ N) \neq 1$ for some integer x and for all prime factors p of $N-1$, then N is prime.*

Proof. Let k be the order of x modulo N. Hence $k \mid (N-1) \implies (N-1) = kj$ for some j. If $j > 1$, then let q be a prime divisor of j. Thus, there exists some integer j_1 such that $j = qj_1$. We have:

$$x^{(N-1)/q} = (x^k)^{j_1} \equiv 1^{j_1} = 1 \ (mod\ N)$$

Which contradicts our second condition. Hence, we have $j = 1 \implies k = (N-1)$. But as we know $\phi(N) \leq (N-1)$, so here $\phi(N) = (N-1) \implies N$ is prime, where $\phi(N)$ is the Euler's totient function which gives us the number of integers less than or equal to N that are coprime with N. □

Example 1.2. *We need to search for some integer x as defined above which can be a lot of work sometimes. Let $N = 1000000007$, then we have $(N-1) = 2 \times 500000003$. Taking base $x = 2$, we see that $2^{1000000006/2} \ (mod\ 1000000007) = 1$. But 1000000007 is lower of the twin prime pair, primes of form $(p, p+2)$. Base $x = 5$ successfully tells us that it is a prime.*

For algorithm refer to [A.1.3]. Above test was improved in late 1960s in which instead of finding a suitable single integer, a, suitable bases are allowed for each prime factor p_i of $(N-1)$.

Theorem 1.3. *If there exists an integer x_i for each prime factor p_i of $(N-1)$ satisfying $x_i^{(N-1)}$ $(mod\ N) = 1$ and $x_i^{(N-1)/p_i}$ $(mod\ N) \neq 1$, then N is a prime integer.*

Proof. Let $(N-1) = \{p_i^{k_i}\}_{i=1}^{r}$ be the prime factorization of $(N-1)$ and h_i be the order of x_i modulo N. As $h_i \mid (N-1)$ but $h_i \nmid (N-1)/p_i$, we get $p_i^{k_i}$ as one of the factors of h_i. But, as for all i, $h_i \mid \phi(N) \implies p_i^{k_i} \mid \phi(N) \implies (N-1) \mid \phi(N)$. As $\phi(N) \geq (N-1)$, we have N a prime integer. □

Later in 1914, Henry Pocklington showed that we do not need complete prime factorization of $(N-1)$ but we only need to factor it such that the factored part exceeds $\sqrt{N}$.

Theorem 1.4 (Pocklington Test). *Let $m_1 m_2 = N-1$ with $gcd(m_1, m_2) = 1$ and $m_1 \geq \sqrt{N}$ such that $m_1 = p_1^{k_1} p_2^{k_2} .....p_r^{k_r}$. If there exists some $x_i \in \mathbb{Z}$ for each prime factor p_i satisfying x_i^{N-1} $(mod\ N) = 1$ and $gcd(x_i^{(N-1)/p_i} - 1, N) = 1$, then N is a prime integer.*

Proof. We give proof by contradiction, assume that p is any prime divisor of N. And take h_i to be the order of x_i modulo p . Then we have $h_i \mid (p-1)$ but x_i^{N-1} $(mod\ N) = 1 \implies x_i^{N-1}$ $(mod\ p) = 1$ which implies $h_i \mid (N-1)$. From second condition we get $x_i^{(N-1)/p_i}$ $(mod\ p) \neq 1 \implies h_i \nmid (N-1)/p_i$. So we have $p_i^{k_i} \mid h_i\ \forall\ i \in \{1, 2, ..., r\}$ $\implies m_i \mid (p-1)$. But $m_i \geq \sqrt{N}$, thus all the prime factors of N must be greater than $\sqrt{N}$, a contradiction. Hence, N is prime. □

Example 1.5. *Taking $N = 1000000009$, next prime to the one in example (1.2). $N-1 = 54936 \times 18203$ with $gcd(54936, 18203) = 1$. Take $m = 54936$ having prime factors $\{2, 3, 7, 109\}$. We do not have to find single base satisfying both the conditions as we did in Lucas test, but distinct x for each factor would do. Here, for $x = 2$ we have $2^{1000000008}$ $(mod\ 1000000009) = 1$ and $gcd(2^{1000000008/p} - 1, 1000000009) = 1$ only for $p = 3, 7$ and 109. Whereas $x = 13$ satisfies both the conditions for all prime factors of 1000000008.*

1.1.2 Miller-Rabin Test

It is a direct test for compositeness which announces that either n is definitely composite or of undecided nature. Miller-Rabin Test is also called a probabilistic primality test because for a chosen base x, the coming theorem is satisfied in case of composite integers also, such bases are called *liars* for compositeness for given composite integer.

Theorem 1.6. *Let p be an odd prime with $(p-1) = 2^s m$, where m is odd and $s \geq 1$. Then any integer $1 < x < p-1$ will satisfy $x^m \equiv 1 \ (mod \ p)$ or $x^{2^j m} \equiv -1 \ (mod \ p)$ for some $0 \leq j < s$.*

Proof. If an integer x has order k modulo p then $k \mid (p-1)$. Let k to be odd then $k \mid m \implies m = kk_1$ for some integer k_1. So we have

$$x^m = (x^k)^{k_1} \equiv 1 \ (mod \ p)$$

Now assume k to be even. Write $k = 2^{j+1}d, j \geq 0$ with d an odd integer. Then $2^{j+1}d \mid 2^s m \implies j+1 \leq s$ *and* $d \mid m$ and from $x^{2^{j+1}d} \equiv 1 \ (mod \ p)$ we get $x^{2^j d} \equiv -1 \ (mod \ p)$. Now take $m = dt$ for some odd t then:

$$x^{2^j m} = (x^{2^j d})^t \equiv -1 \ (mod \ p)$$

As required. □

Test for compositeness : First, choose a random integer $1 < x < N$ for given integer N. Then if $x^m \not\equiv 1 \ (mod \ N)$ and $x^{2^j m} \not\equiv -1 \ (mod \ N) \ \forall j = 0, 1, ..., s-1$, where s and m are calculated as in (1.6), then N is definitely composite by above theorem. Such base x is called a *witness* for compositeness of N.
For some composite N, let $W(N)$ be the set of all base satisfying (1.6) then $|W(N)| < \phi(N)/4$ [20], where $|W(N)|$ denotes number of elements in set $W(N)$. For algorithm refer [A.1.2].

Example 1.7. *Consider $N = 69741$. Then $N - 1 = 69740 = 2^2 \times 17435$. Then taking $x = 3$ both the conditions of Miller-Rabin test are not satisfied, hence 69741 is composite. See that for bases $x = 11908, 17011, 18712, 39124$ we have $x^{17435} \ (mod \ 69741) = 1$ but not for all bases $1 < x < N - 1$.*

1.1.3 Solovay-Strassen Test

This test is similar to Miller-Rabin Test as it is also a probabilistic primality test which recognizes composites with probability at least $\frac{1}{2}$. Before this we need few lemmas regarding primitive roots and quadratic residues which are discusses below in brief.

Definition 1.8. *1. If for some $x \in \mathbb{Z}$ satisfying $gcd(x, N) = 1$ we have order of x equals $\phi(N)$ modulo N i.e. $x^{\phi(N)} \equiv 1 \ (mod\ N)$ but $x^r \not\equiv 1 \ (mod\ N)\ \forall\ r < \phi(N)$, then x is said to be a primitive root modulo N.*

2. If for some $x \in \mathbb{Z}$ we have $x^2 \equiv x \ (mod\ N)$; $x \in \mathbb{Z}$, for integer $N \geq 2$ with $gcd(x, N) = 1$ then x is called a quadratic residue of N. If $gcd(x, N) = 1$ but $x^2 \not\equiv x \ (mod\ N)$ For any $x \in \mathbb{Z}$ then it is called a quadratic nonresidue of N.

Theorem 1.9. *If K is a finite field, then K^* is a cyclic group. Here $K^* = K - \{0\}$.*

Proof. Let number of elements in K, denoted as $|K|$, be p. Then $|K^*| = p - 1$. Let $\mathcal{B}_d = \{b \in K^* \mid ord_{K^*}(b) = d\}$ be the sets for all divisors d of $p - 1$ where $ord_{K^*}(b)$ denotes the order of element b in group K^*. Take an arbitrary element $x \in \mathcal{B}_d$. Denote the cyclic group generated by x as $\langle x \rangle$. Then we have $|\langle x \rangle| = d$ and for all $0 \leq i < d$ we have $(x^i)^d = 1$. So, all the roots of polynomial $X^d - 1$ in K are the elements of $\langle x \rangle$. Now, if $b \in \mathcal{B}_d$ then we have $b^d = 1$ in K, making it a root of $X^d - 1$ which implies $b \in \langle x \rangle$. Therefore, $\mathcal{B}_d \subseteq \langle x \rangle$. As number of elements in group $\langle x \rangle$ of order d is $\phi(d)$[13, page 65], where ϕ is Euler's totient function. So, $|\mathcal{B}_d| = \phi(d)$.
Hence, we have a disjoint partition of group K^* in terms of $\mathcal{B}_d$'s containing order d elements for all divisors d of $p - 1$. So, we have

$$|K^*| = p - 1 = \sum_{d|p-1} |\mathcal{B}_d|$$

And as we have $p - 1 = \sum_{d|p-1} \phi(d)$ [13, page 70], thus all of the $\mathcal{B}_d$'s are non-empty sets. So, $\mathcal{B}_{p-1}$ will be a non-empty set with each of $\phi(p - 1)$ elements $b \in \mathcal{B}_{p-1}$ as generators of K^*. □

So, by above theorem we see that for prime number p, Z_p^* is a cyclic group with $\phi(p - 1)$ generators (also called primitive elements modulo p).

Lemma 1.10 (Euler's Criteria). *Let p be an odd prime number. Then set, S, of quadratic residues of p is a subgroup of $\mathbb{Z}_p^*$ with $|S| = (p - 1)/2$. Also, for $x \in \mathbb{Z}_p^*$ we have:*

$$x^{p-1/2} \ (mod\ p) = \begin{cases} 1, & \text{If } x \text{ is a quadratic residue mod } p \\ -1, & \text{If } x \text{ is a quadratic non-residue mod } p. \end{cases}$$

Proof. As p is prime, we have $\mathbb{Z}_p^*$ a cyclic group by (1.9). Let g be a generator of $\mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{1, g, g^2, ..., g^{p-2}\}$. As $g^{(p-1)/2} = 1$ but $g^{(p-1)/2} \neq 1$ in $\mathbb{Z}_p^*$ then we must have $g^{(p-1)/2} = -1$. Then all $(p-1)/2$ even powered elements g^{2i}, $0 \leq i < (p-1)$, are squares in $\mathbb{Z}_p^*$ with $(g^{2i})^{(p-1)/2} = g^{i(p-1)} = 1$ and we have odd powered elements satisfying (g^{2i+1})=$g^{i(p-1)}g^{(p-1)/2}$. □

Remark 1.11. *All $1 \leq x < N$ satisfying Euler's criteria for odd composite N are called $E-liars$, otherwise, $E-witnesses$. Similarly, $F-liar$ if Fermat's test is satisfied and $F-witness$ otherwise.*

Example 1.12. *Let $N = 47$, then number of quadratic residues = number of quadratic non-residues = 23. If $N = 49$, an odd composite, then we get $[1, 18, 19, 30, 31, 48]$ as our set of $E-liars$.*

Now we will define an established notation for indicating whether a number $x \in \mathbb{Z}$ is a quadratic residue modulo p or not.

Definition 1.13. *1. For a prime number $p \geq 3$ and some $x \in \mathbb{Z}$, let*

$$\left(\frac{x}{p}\right) = \begin{cases} 0, & \textit{If } x = k.p \textit{ for some integer } k \\ 1, & \textit{If } x \textit{ is a quadratic residue mod } p \\ -1, & \textit{If } x \textit{ is a quadratic non-residue mod } p. \end{cases}$$

Then we call $\left(\frac{x}{p}\right)$ the $Legendre\ Symbol\ of\ x\ and\ p$.

2. We can generalize Legendre symbol for an odd integer N, it is called the Jacobi Symbol. For some odd $N \in \mathbb{Z}$ with prime factorization $N = p_1^{e_1}p_2^{e_2}p_3^{e_3}....p_r^{e_r}$ we define Jacobi symbol as:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p_1}\right)^{e_1}\left(\frac{x}{p_2}\right)^{e_2}\left(\frac{x}{p_3}\right)^{e_3}.......\left(\frac{x}{p_r}\right)^{e_r}$$

Calculating Jacobi symbol

First we will give few properties of Jacobi symbol in following remark and then see how to compute Jacobi symbol for some odd integer N without factoring it into primes.

Remark 1.14. *Let $N, M \geq 3$ be odd integers. Then if $x, y \in \mathbb{Z}$, then we have:*

1. $\left(\frac{xy}{N}\right) = \left(\frac{x}{N}\right)\left(\frac{y}{N}\right)$
2. $\left(\frac{x}{NM}\right) = \left(\frac{x}{N}\right)\left(\frac{x}{M}\right)$

3. $\left(\frac{x}{N}\right) = \left(\frac{x \bmod N}{N}\right)$
4. $\left(\frac{0}{N}\right) = 0$ *and* $\left(\frac{1}{N}\right) = 1$.

Above properties can be seen easily from 1.13 and 1.10*.*

Lemma 1.15 (Gauss's Lemma)**.** *Let $H_p = \{1, 2, ..., \frac{p-1}{2}\}$ for some odd prime p and for some $x \in \mathbb{Z}$ with $gcd(p, x) = 1$ consider the set*

$$S_p(x) = \{x \ (mod\ p), 2x \ (mod\ p), ..., \frac{p-1}{2}x \ (mod\ p)\}$$

If $k_p(x)$ denotes the elements in $S_p(x)$ that belong to $\mathbb{Z}_p^* - H_p$ then

$$\left(\frac{x}{p}\right) = (-1)^{k_p(x)}$$

Proof. See that all elements in $S_p(x)$ are distinct as Z_p^* is a field. Let $r_1, r_2, ..., r_{k_p(x)}$ be the elements in $S_p(x)$ that exceed $p/2$, and $s_1, s_2, ..., s_t$, $t = \frac{1}{2}(p-1) - k_p(x)$, be the remaining elements such that $0 < s_i < p/2$. Then we have $\{p - r_i \mid 1 \leq i \leq k_p(a)\} \subseteq H_p$. Now we will show that H_p is a disjoint union of $(p - r_i)'s$ and $s_j's$ as defined above. We only need to show that no $(p - r_i)$ is equal to s_j. We will show that by contradiction. So, assume $(p - r_i) = s_j$ for some i and j. Then for some $u, v \in H_p$ we have $r_i = ux \ (mod\ p)$ and $s_j = vx \ (mod\ p)$. Then we have $0 \equiv p \equiv r_i + s_j \equiv (u+v)x \ (mod\ p)$. Which implies $(u+v) \equiv 0 \ (mod\ p)$, which can not happen as $1 < u + v \leq p - 1$.
Thus we have

$$\begin{aligned}\prod_{h \in H_p} h &= \prod_{i=1}^{k_p(x)} (p - r_i) \prod_{j=1}^{t} s_i \\ &\equiv (-1)^{k_p(x)} \prod_{i=1}^{k_p(x)} r_i \prod_{j=1}^{t} s_i \ (mod\ p) \\ &\equiv (-1)^{k_p(x)} x^{(p-1)/2} \prod_{h \in H_p} h\end{aligned}$$

The last equivalence follows after putting values if r_i and s_j from $S_p(x)$. After cancelling $\prod_{h \in H_p} h$, we get $x^{(p-1)/2} \ (mod\ p) = (-1)^{k_p(x)}$. Now, by applying 1.10 we have $\left(\frac{x}{p}\right) = (-1)^{k_p(x)}$. ☐

Corollary 1.16. *Let $p \geq 3$ be an odd prime number, then*

$$\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$$

Proof. By 1.15 we get $\left(\frac{2}{p}\right) = (-1)^{k_p(2)}$. We have $S_p(2) = \{2, 4, ..., p-1\}$. As elements in $S_p(2)$ are less than p, we only need to count the number of elements that exceed $p/2$. See that $2k < p/2$ if and only if we have $k < p/4$. So $k_p(2) = \frac{p-1}{2} - \lfloor \frac{p}{4} \rfloor$ where $\lfloor . \rfloor$ denoted the greatest integer function. Now we have the following cases for $p \ (mod\ 8)$, for any odd prime p, and $k_p(2)$

1. $p = 8m + 1$, then $k_p(2) = 4m - 2m = 2m$
2. $p = 8m + 3$, then $k_p(2) = 4m + 1 - 2m = 2m + 1$
3. $p = 8m + 5$, then $k_p(2) = 4m + 2 - 2m - 1 = 2m + 1$
4. $p = 8m + 7$, then $k_p(2) = 4m + 3 - (2m + 1) = 2m + 2$

Hence, $k_p(2)$ is even if and only if $p \equiv 1 \ or\ 7 \ (mod\ 8)$. Thus we have

$$For\ p = 8m \pm 1,\ \frac{p^2 - 1}{8} = \frac{(8m \pm 1)^2 - 1}{8} = 8m^2 \pm 2m,\ which\ is\ even.$$
$$For\ p = 8m \pm 3,\ \frac{p^2 - 1}{8} = \frac{(8m \pm 3)^2 - 1}{8} = 8m^2 \pm 6m + 1,\ which\ is\ odd.$$

So, corollary follows. ☐

Proposition 1.17. *Let $N \geq 3$ be an odd integer, then*

$$\left(\frac{2}{N}\right) = (-1)^{(N^2-1)/8}$$

Proof. Assume prime factorization of $N = p_1 p_2 .... p_m$. Then we proceed by induction on m. Proposition follows for $m = 1$ by 1.16. Now assume that it holds for $m \geq 2$, that is for integers with less than m primes. If $N = n_1 n_2$, then 1.17 holds for both n_1 and n_2 by assumption. Thus, we have

$$\left(\frac{2}{N}\right) = \left(\frac{2}{n_1}\right)\left(\frac{2}{n_2}\right) = (-1)^{(n_1^2-1)/8 + (n_2^2-1)/8}$$

And as 8 divides $\frac{(n_1^2-1)(n_2^2-1)}{8}$, we have

$$\frac{N^2 - 1}{8} = \frac{(n_1^2 - 1)(n_2^2 - 1) + n_1^2 + n_2^2 - 2}{8} \equiv \frac{(n_1^2 - 1)}{8} + \frac{(n_2^2 - 1)}{8} \ (mod\ 2)$$

Which concludes our proof. ☐

Now we will give the steps for the computation of Jacobi symbol for an arbitrary integer x and odd number $N \geq 3$.

1. set $\left(\frac{x}{N}\right) \leftarrow \left(\frac{x \ (mod\ N)}{N}\right)$. (By 1.14(3))
2. If $x = 0$ output 0, if $x = 1$ output 1. (By 1.14(4))
3. While $4 \mid x$ set $\left(\frac{x}{N}\right) \leftarrow \left(\frac{x/4}{N}\right)$.
4. While $2 \mid x$, set $\left(\frac{x}{N}\right) \leftarrow \left(\frac{x/2}{N}\right)$ if $N \ (mod\ 8) = 1 \ or\ 7$, else $\left(\frac{x}{N}\right) \leftarrow -\left(\frac{x/2}{N}\right)$ if $N \ (mod\ 8) = 3 \ or\ 5$. (Follows from 1.17)
5. If $x \ or\ N \equiv 1 \ (mod\ 4)$, set $\left(\frac{x}{N}\right) \leftarrow \left(\frac{N \ (mod\ x)}{x}\right)$.
6. If $x \ and\ N \equiv 3 \ (mod\ 4)$, set $\left(\frac{x}{N}\right) \leftarrow -\left(\frac{N \ (mod\ x)}{x}\right)$.

For step 3, see that if $4 \mid x$ then $x = 4x_1$ for some integer x_1, then $\left(\frac{x}{N}\right) = \left(\frac{2}{N}\right)^2 \left(\frac{x_1}{N}\right) = \left(\frac{x_1}{N}\right)$. Step 5 and 6 follow from 1.14(3) and quadratic reciprocity law for odd integers [13, page 90].

Solovay-Strassen Test

Lemma 1.18. *Let p be an odd prime, then*

$$x^{(p-1/2)} \left(\frac{x}{p}\right) \equiv 1 \ (mod\ p) \ \forall\ x \in \{1, ..., p-1\}.$$

Proof. Can be seen directly from *Lemma* 1.10 and *definition* 1.13. □

Given an odd integer N, Steps for Solovay-Strassen primality test are as follows:

1. Choose a random base $x \in \{2, ..., N-1\}$.
2. if $x^{(N-1)/2}\left(\frac{x}{N}\right) \ (mod\ p) \neq 1$, output composite.
3. else output probable prime.

The algorithm is given in [A.1.4].

Remark 1.19. *Every $E - liar$ is also an $F - liar$ for N. It can be easily seen as if x is an $E - liar \implies x^{(N-1/2)}\left(\frac{x}{N}\right) \ mod\ N = 1$. Then on squaring we get $x^{N-1} \ mod\ N = 1$. And clearly, F-witness implies $E - witness$.*

Now from below lemmas we will see that in case of composite integers there exists *E-witnesses* and there is more than 1/2 probability of encountering such witness in case of composite integers.

Lemma 1.20. *Let N be odd composite positive integer, then there exists $x \in \mathbb{Z}_N^*$ such that $gcd(x, N) = 1$ and $x^{(N-1/2)} \not\equiv \left(\frac{x}{N}\right)$ $(mod\ N)$.*

Proof. Case 1: Assume N to be product of several distinct primes. Take $N = p.m$, for odd $m \geq 3$ and odd prime p such that $p \nmid m$. Let $b \in \mathbb{Z}_p^*$ such that $\left(\frac{b}{p}\right) = -1$. Applying Chinese remainder theorem, we see that x, $1 \leq x < N$, satisfies:

$$x \equiv b\ (mod\ p),\ and$$
$$x \equiv 1\ (mod\ N)$$

Claim: $x \in \mathbb{Z}_N^*$ and is an $E - witness$.
Proof: Clearly, $p \nmid x$ and $gcd(x, m) = 1$. So, $x \in \mathbb{Z}_N^*$ and

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{m}\right) = \left(\frac{b}{p}\right)\left(\frac{1}{m}\right) = -1$$

Assume x to be $E - liar \implies x^{N-1/2} \equiv -1$ as $\left(\frac{x}{N}\right) \equiv -1\ (mod\ N)$. And as $m \mid N$ we have $x^{N-1/2} \equiv -1\ (mod\ m)$, a contradiction as $x \equiv 1\ (mod\ m)$.
Case 2: Assume N is divisible by p^2, $p \geq 3$. Write $N = p^k m\ for\ k \geq 2$ and odd m such that $p \nmid m$. Then $m = 1$, $x = 1 + p$ works. For $m \geq 3$, on using Chinese remainder theorem, we have x, $1 \leq x < p^2.m \leq N$ with:

$$x \equiv 1 + p\ (mod\ p^2),\ and$$
$$x \equiv 1\ (mod\ m)$$

Clearly, $p^2 \mid x$ and $gcd(x, m) = 1 \implies gcd(x, N) = 1$, hence $x \in \mathbb{Z}_N^*$. Now, assume x to be $F - liar \implies x^{N-1} \equiv 1\ (mod\ N)$. As $p^2 \mid N$ we have $x^{N-1} \equiv 1\ (mod\ p^2)$. Then on using binomial theorem we get:

$$x^{N-1} \equiv (1 + p)^{N-1} \equiv 1 + (N - 1)p\ (mod\ p^2)$$

Hence, $(N - 1)p \equiv 0\ (mod\ p^2)$ implying $p^2 \mid N - 1 = p^k.m - 1$, which is not possible. Thus, x is a *F-witness* which implies it is an *E-witness* also. □

Below lemma shows that the probability that a randomly chosen integer x for Solovay-Strassen test would be an *E-witness* for odd composite integer N is more than 1/2.

Lemma 1.21. *Let $N \geq 3$ be an odd composite integer with $1 \leq x \leq N-1$, then*

$$\frac{|\{x \mid gcd(x,N) = 1 \; and \; x^{N-1/2} \equiv \left(\frac{x}{N}\right) \; (mod \; N)\}|}{N-1} < \frac{1}{2}$$

Proof. Lets split N into disjoint sets as:

$$\begin{aligned} A &= \{a \mid gcd(a,N) = 1, \; a \; is \; a \; E-liar\} \\ B &= \{b \mid gcd(b,N) = 1, b \; is \; a \; E-witness\} \\ C &= \{c \mid gcd(c,N) > 1\} \end{aligned}$$

Where $1 \leq a,b,c < N$. As N is composite, $|C| > 0$, and $|B| > 0$ by *Lemma* 1.20. Take some $b_0 \in B$. Then $Ab_0 = \{ab_0 \; mod \; N \mid a \in A\}$. As

$$gcd(a,N) = 1, \; gcd(b_0,N) = 1 \implies gcd(ab_0,N) = 1.$$

Then we have,

$$(ab_0)^{N-1/2} \equiv a^{N-1/2} b_0^{N-1/2} \equiv \left(\frac{a}{N}\right) b_0^{N-1/2} \; (mod \; N)$$

As $ab_0 \; (mod \; N) \in A \; or \; B$, first assume that $ab_0 \; (mod \; N) \in A$. Then,

$$(ab_0)^{N-1/2} \equiv \left(\frac{ab_0}{N}\right) = \left(\frac{a}{N}\right)\left(\frac{b_0}{N}\right) \equiv \left(\frac{a}{N}\right) b_0^{N-1/2} \; (mod \; N).$$

Means $\left(\frac{b_0}{N}\right) \equiv b_0^{N-1/2} \; (mod \; N)$, a contradiction as $b_0 \in B \implies Ab_0 \subset B$. Take $a, a' \in A$, if $ab_0 \equiv a'b_0 \; (mod \; N) \implies a \equiv a' \; (mod \; N) \implies a = a'$ in A. Hence any two elements ab_0, $a'b_0 \in Ab_0$ for distinct $a, a' \in A$ are distinct. Thus $|Ab_0| = |A|$ but, $|A| = |Ab_0| \leq |B|$. So, $N-1 = |A| + |B| + |C| \geq |A| + |A| + 1 = 2|A| + 1 > 2|A|$ which implies $|A| < (N-1)/2$. $\square$

Example 1.22. *Taking $N = 49$, as we have $[1, 18, 19, 30, 31, 48]$ as our set of $E-liars$, remaining integers x such that $gcd(x,N) = 1$ are 36 in number. Hence probability that randomly chosen base $1 \leq x \leq N-1$ will be an $E-liar$ here is $\frac{6}{48} = \frac{1}{8}$.*

1.2 Some Factorization Algorithms

1.2.1 Fermat-Kraitchik Factorization Method

The idea for Fermat's factorization scheme is that for any odd integer N, on solving $N = x^2 - y^2$ we can get factorization $N = (x-y)(x+y)$. Conversely, for any odd $N = ab$, $a \geq b \geq 1$ we have $N = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$. See that $(a+b)/2$ and $(a-b)/2$ will be non-negative integers because N is taken to be odd.
Now we look at the implementation of the above idea. Start from an integer k such that $k^2 \geq N$. Then look successively at:

$$k^2 - N,\ (k+1)^2 - N, ...,\ (k+j)^2 - N\ ;\ j = 0, 1, 2, ...$$

Until we find a perfect square. The algorithm for checking a perfect square is given in [A.2.5]. This process is finite because eventually we will arrive at trivial factorization $N = N.1$ when $k + j = \frac{N+1}{2}$.

It returns large factors which are not necessarily prime. Direct method of trying to factor by numbers less than $\sqrt{N}$ works better for integers with small factors but in case of large integers $\sqrt{N}$ is large, increasing the number of computations to find factors near $\sqrt{N}$.

A generalization of above method makes things simpler in theory but it can be more time consuming practically. Instead of solving for $x^2 - y^2 = N$, we solve for $x^2 - y^2 = k.N,\ x \not\equiv \pm y\ (mod\ N)$ for some integer k. Which means:

$$x^2 \equiv y^2\ (mod\ N) \implies x^2 - y^2 \equiv 0\ (mod\ N)\ i.e.\ (x-y)(x+y) \equiv 0\ (mod\ N)$$

Hence, we get $gcd(x-y, N)$ and $gcd(x+y, N)$ as our non-trivial divisors of N.

It is better in theory as now we need to check for $x^2 - k.N$ for perfect square for any value of k whereas earlier $x^2 - N$ had to be a square so we would need to calculate x^2 every time. While implementing we can put a bound on k value to avoid running loop for longer time.

Maurice Kraitchik introduced a more efficient variant of Fermat's factorization scheme in 1920. Instead of looking for a perfect square from $x^2 - N$ directly we look for a sequence $(x_1^2 - N), (x_2^2 - N), ..., (x_r^2 - N)$ with product of these quantities forming a perfect square, say y^2. Then, we have:

$$(x_1^2 - N)(x_2^2 - N)......(x_r^2 - N) = y^2$$
$$\implies (x_1x_2.....x_r)^2 = y^2 \ (mod\ N)$$

Which will give us non-trivial factors $gcd(x_1x_2....x_r - y, N)$ and $gcd(x_1x_2....x_r + y, N)$ if $(x_1x_2....x_r) \not\equiv \pm y \ (mod\ N)$. This technique became a basis for modern methods such as quadratic sieve algorithm. Algorithms for both generalized and non-generalised factorization are given in [A.2.1].

Example 1.23. *Take $N = 3471$. Then we start with $k = 59 > \sqrt{3471}$. After 5 steps we get $k = 59 + 5 = 64$ with $64^2 - 3471 \equiv 25^2 \ (mod\ 3471)$. Thus, we have $gcd(64 - 25, 3471) = 39$ and $gcd(64 + 25, 3471) = 89$ as our divisors as $3471 = 39 \times 89$.*

1.2.2 Pollard's Methods

Pollard's rho-method

John Pollard proposed an efficient method for finding factors up to 20 digits. Assume that we know N to be a composite odd integer. In this method, first we choose some $degree \geq 2$ polynomial $A(x)$ with integer coefficients. For example, a quadratic polynomial $A(x) = x^2 + q,\ q \neq 0, -2$. Then take an initial value x_0 and form sequence $x_1, x_2, x_3....$ computed as:

$$x_{k+1} \equiv A(x_k) \ (mod\ N) \tag{1.1}$$

If m is a non-trivial divisor of N then there may exists x_i, x_j such that $x_i \equiv x_j \ (mod\ m)$ but $x_i \not\equiv x_j \ (mod\ N)$ as number of congruence classes modulo m is less than that of modulo N. If so, then $m \mid (x_i - x_j)$ but $N \nmid (x_i - x_j)$ which implies $m_1 = gcd(x_i - x_j, N)$ is a non-trivial divisor of n.
It becomes more time consuming to compute $gcd(x_i - x_j, N)$ as i increases($i > j$) so it is more efficient if we check for cases where $i = 2j$ as then we would have $x_{2j} \equiv x_j \ (mod\ m)$, m is yet undiscovered. If $x_i \equiv x_j \ (mod\ m)$ $(i > j)$ then we have $A(x_i) \equiv A(x_j) \ (mod\ m)$ which implies $x_{i+1} \equiv x_{j+1} \ (mod\ m)$. Means we have a repetition of values with some finite period $i - j$. So we will keep checking for $m = gcd(x_{2j} - x_j, N)$, $j \geq 1$ and there will exist an integer j such that $1 < m < N$. This is popularly known as Pollard's rho-method, it was used to factor Fermat number F_8 by Brent and Pollard in 1980 using $A(x) = x^{2^{10}} + 1$ and $x_0 = 3$ in only 2 hours of computer time.

A Brent's modification to above method is as follows. Let $\mathcal{T}$ be the period of sequence (1.1) and $\mathcal{M}$ be such that for all $j \geq \mathcal{M}$ we have $x_{j+\mathcal{T}} \equiv x_j \ (mod\ m)$. Define

$\mathcal{L}(k) = 2^r$ such that $2^r \leq k < 2^{r+1}$. Then we have $\mathcal{L}(k) = 2^{\lfloor \log_2 k \rfloor}$.
Claim : There exists an k such that $x_k \equiv x_{\mathcal{L}(k)-1} \ (mod\ m)$.

Proof. If we take $k = 2^r + \mathcal{T} - 1$ where $r = \lceil \log_2 max(\mathcal{M}+1, \mathcal{T}) \rceil$ then

$$\begin{aligned} 2^r = 2^{\lceil \log_2 max(\mathcal{M}+1,\mathcal{T}) \rceil} &\geq 2^{\log_2 max(\mathcal{M}+1,\mathcal{T})} \\ &\geq max(\mathcal{M}+1, \mathcal{T}) \\ &> \mathcal{T} - 1 \end{aligned} \tag{1.2}$$

Then from (1.2) we get $\mathcal{L}(k) = 2^r$. Thus $k - (\mathcal{L}(k) - 1) = 2^r + \mathcal{T} - 1 - 2^r + 1 = \mathcal{T}$. Hence, $x_k \equiv x_{\mathcal{L}(k)-1} \ (mod\ m)$, which proves the existence of k. □

For algorithm using Brent's modification refer to [A.2.2], Where we look at x_j values for $j = 2^r - 1$ for some integer r, and for all $2^r \leq k < 2^{r+1}$ check if $gcd(x_k - x_{2^r-1}, N)$ is a non-trivial factor of N.

Example 1.24. *Assume $N = 3267$ with $A(x) = x^2 + 2x + 1$, $x_0 = 2$. Using Brent's method we see that for $r = 2$, we have $x_{2^2-1} = A(x_2) = 100$. Now going over $2^2 \leq k < 2^3$, we get $gcd(x_4 - 100, 3267) = gcd(300, 3267) = 3$ as a non-trivial factor of 3267.*

Pollard's $p - 1$ Method

First we need the definition below.

Definition 1.25. *A positive integer $N = p_1^{e_1} p_2^{e_2} .....p_r^{e_r}$ is called $\mathcal{B}$-smooth if for all $i \in \{1, 2, ..., r\}$ we have $p_i \leq \mathcal{B}$ for some positive integer $\mathcal{B}$. It is called $\mathcal{B}$-powersmooth if $p_i^{e_i} \leq \mathcal{B} \ \forall\ i \in \{1, 2, ..., r\}$.*

Now the idea behind this method is as following. Let p be a prime factor of N. Choose integer $x > 1$ randomly such that $gcd(x, N) = 1$. Then by Fermat's theorem we have $x^{p-1} \equiv 1 \ (mod\ p)$. And if we have an positive integer $\mathcal{B}$ such that $(p-1)$ is $\mathcal{B}$-*powersmooth*, then $(p - 1) \mid L$ where $L = lcm(1, 2, ..., \mathcal{B})$. Hence, we will have $x^L \equiv 1 \ (mod\ p)$ which implies $gcd(x^L - 1, N) > 1$ and we will have a non-trivial divisor as long as $x^L \not\equiv 1 \ (mod\ N)$. If not then choose another such x and go again. Note that this method could also fail if we do not have any prime p such that $p - 1$ is $\mathcal{B}$-*powersmooth*. For algorithm refer to [A.2.2].

Example 1.26. *Taking $N = 57845$ and bound $\mathcal{B} = 4$ such that $L = lcm(1, 2, 3, 4) = 12$. Then choosing base x randomly between 1 and $N - 1$, we get $x = 17941$ with $gcd(17941, 57845) = 1$. Now, we have $17941^{12} \ (mod\ 57845) = 14836$. Thus, a divisor* *of* 57845 *is* $gcd(14836-1, 57845) = 115$. *Our bound worked perfectly as for one of the prime factors of* N, 5, *we have* $5-1=4=2^2 \leq \mathcal{B}$.

1.2.3 Continued Fraction Factoring Algorithm(CFFA)

Continued Fractions

Continued fractions provide us surprisingly good rational approximations to real numbers. Any rational number can be written as a finite simple continued fraction. Continued fractions are mainly used to solve linear diophantine equations and Pell-type equations. $Finite\ continued\ fractions$ are of the form:

$$b_0 + \cfrac{1}{b_1 + \cfrac{1}{b_2 + \cfrac{1}{\ddots \cfrac{1}{b_{n-1} + \frac{1}{b_n}}}}}$$

where $b_i \in \mathbb{R}$, all are positive except possibly b_0. We call a continued fraction simple if $b_i \in \mathbb{Z}$. We will denote a finite continued fraction as $[b_0; b_1, b_2, ..., b_n]$ and similarly write infinite continued fraction as $[b_0; b_1, b_2, b_3, ...]$

If we expand a rational number $p/q,\ q > 0$ using Euclid's algorithm then we get the equations:

$$\begin{aligned} p &= qp_0 + r_1 \quad 0 < r_1 < q \\ q &= r_1p_1 + r_2 \quad 0 < r_2 < r_1 \\ r_1 &= r_2p_2 + r_3 \quad 0 < r_3 < r_2 \\ &. \\ &. \\ r_{n-1} &= r_np_n + 0 \end{aligned}$$

From above equations see that we can write p/q as $[p_0; p_1, p_2, ..., p_n]$.

Similarly, for an irrational $b = \sqrt{N}$, we define infinite continued fraction $[b_0; b_1, b_2, b_3, ...]$ as $b_i = \lfloor \mathcal{X}_i \rfloor$ where $\mathcal{X}_i$ is defined as:

$$\mathcal{X}_i = \frac{1}{\mathcal{X}_{i-1} - b_{i-1}}, \quad where\ \mathcal{X}_0 = b\ ;\ i \geq 1 \tag{1.3}$$

Here $\lfloor \mathcal{X} \rfloor$ represents greatest integer function less than or equal to $\mathcal{X}$.

Definition 1.27. *We define k^{th} convergent, $\mathcal{C}_k$, for continued fraction $[b_0; b_1, b_2, ..., b_n]$ as :*

$$\mathcal{C}_k = [b_0; b_1, b_2, ..., b_k] \quad 1 \leq k \leq n \tag{1.4}$$

Remark 1.28. 1. *The k^{th} convergents form strictly increasing and decreasing sequences for even and odd k values respectively.*

2. *$\mathcal{C}_{odd} < \mathcal{C}_{even}$, Means all odd subscript convergents are greater than even subscript convergents.*

Theorem 1.29. *Let $\mathcal{C}_k$ be as in (1.4), then $\mathcal{C}_k = \mathcal{P}_k / \mathcal{Q}_k$. Where $\mathcal{P}_k$ and $\mathcal{Q}_k$, with conditions $\mathcal{P}_{-2} = \mathcal{Q}_{-1} = 0$ and $\mathcal{P}_{-1} = \mathcal{Q}_{-2} = 1$, are defined as:*

$$\begin{aligned} \mathcal{P}_k &= b_k \mathcal{P}_{k-1} + \mathcal{P}_{k-2} \\ \mathcal{Q}_k &= b_k \mathcal{Q}_{k-1} + \mathcal{Q}_{k-2} \end{aligned} \tag{1.5}$$

With $k = 0, 1, ..., n$.

Proof. See [6, page 312]. □

Theorem 1.30. *Let $\mathcal{P}_k$ and $\mathcal{Q}_k$ be as in (1.5), then:*

$$\mathcal{P}_k \mathcal{Q}_{k-1} - \mathcal{Q}_k \mathcal{P}_{k-1} = (-1)^{k-1} \quad 1 \leq k \leq n$$

Proof. Proof is simply by induction on k, [6, page 314]. □

Lemma 1.31. *For $\sqrt{N} = [b_0; b_1, b_2, b_3, ...]$ and $\mathcal{X}_k$ as in (1.3), if we define $\mathcal{S}_k$ and $\mathcal{T}_k$ as:*

$$\mathcal{S}_0 = 0 \qquad \mathcal{T}_0 = 1 \tag{1.6}$$

$$\mathcal{S}_{k+1} = b_k \mathcal{T}_k - \mathcal{S}_k \qquad \mathcal{T}_{k+1} = \frac{N - \mathcal{S}_{k+1}^2}{\mathcal{T}_k},\ k = 0, 1, 2, 3, ... \tag{1.7}$$

Then $\mathcal{S}_k$ and $\mathcal{T}_k$ are integers and

1. $\mathcal{T}_k \mid (N - \mathcal{S}_k^2)$
2. $\mathcal{X}_k = (\mathcal{S}_k + \sqrt{N})/\mathcal{T}_k,\ k \geq 0$

Proof. Refer [6, page 344]. □

The use of infinite continued fractions in finding solution to the Pell-Fermat equation $x^2 - Ny^2 = 1$, where N is a positive non-square integer, asserts that any positive solution (x, y) is $(\mathcal{P}, \mathcal{Q})$ where $\mathcal{P}/\mathcal{Q}$ is a convergent of infinite continued fraction expansion of $\sqrt{N}$ [6, page 340]. We will now prove a theorem from which the solution to the equation $x^2 - Ny^2 = 1$ is clear.

Theorem 1.32. *Let $\mathcal{C}_k, \mathcal{P}_k, \mathcal{Q}_k, \mathcal{T}_k$ be as in (1.4), (1.5), (1.6) and (1.7), then we have $\mathcal{P}_k^2 - N\mathcal{Q}_k^2 = (-1)^{k+1}\mathcal{T}_{k+1}, k = 0, 1, 2, .., $ where $\mathcal{T}_{k+1} > 0$.*

Proof. If we write $\sqrt{N}$ as $[b_0; b_1, b_2, ..., b_k, \mathcal{X}_{k+1}]$, then we have $\sqrt{N} = (\mathcal{X}_{k+1}\mathcal{P}_k + \mathcal{P}_{k-1})/(\mathcal{X}_{k+1}\mathcal{Q}_k + \mathcal{Q}_{k-1})$. On using $\mathcal{X}_{k+1}$ value from condition 2 of *lemma* 1.31 and simplifying we get:

$$\sqrt{N}(\mathcal{S}_{k+1}\mathcal{Q}_k + \mathcal{T}_{k+1}\mathcal{Q}_{k-1} - p_k) = (\mathcal{S}_{k+1}\mathcal{P}_k + \mathcal{T}_{k+1}\mathcal{P}_{k-1} - d\mathcal{Q}_k)$$

As RHS is rational but $\sqrt{N}$ is not, we must have $\mathcal{S}_{k+1}\mathcal{Q}_k + \mathcal{T}_{k+1}\mathcal{Q}_{k-1} = \mathcal{P}_k$ and $\mathcal{S}_{k+1}\mathcal{P}_k + \mathcal{T}_{k+1}\mathcal{P}_{k-1} = d\mathcal{Q}_k$. Then

$$\begin{aligned}\mathcal{P}_k^2 - N\mathcal{Q}_k^2 &= \mathcal{P}_k(\mathcal{S}_{k+1}\mathcal{Q}_k + \mathcal{T}_{k+1}\mathcal{Q}_{k-1}) - \mathcal{Q}_k(\mathcal{S}_{k+1}\mathcal{P}_k + \mathcal{T}_{k+1}\mathcal{P}_{k-1}) \\ &= \mathcal{T}_{k+1}(\mathcal{P}_k\mathcal{Q}_{k-1} - \mathcal{P}_{k-1}\mathcal{Q}_k)\end{aligned}$$

By *Theorem* 1.30, we get $\mathcal{P}_k^2 - N\mathcal{Q}_k^2 = (-1)^{k+1}\mathcal{T}_{k+1}$. Now we have to prove $\mathcal{T}_{k+1} > 0$. From *Remark* 1.28 we get $\mathcal{C}_{2K} < \sqrt{N} < \mathcal{C}_{2k+1}$, $k \geq 0$. Then from equation $(\mathcal{P}_k^2 - N\mathcal{Q}_k^2)/(\mathcal{P}_{k-1}^2 - N\mathcal{Q}_{k-1}^2) = -(\mathcal{T}_{k+1}/\mathcal{T}_k)$, see that $(\mathcal{T}_{k+1}/\mathcal{T}_k)$ is always positive. So, starting from $\mathcal{T}_1 > 0$ we climb to $\mathcal{T}_{k+1} > 0$ as required.b □

Continued fraction factoring algorithm

The continued fraction factoring algorithm is given in [A.2.3]. The description for algorithm is as follows. For a non-square positive integer N assume its continued fraction expansion be $[b_0; b_1, b_2, ...]$. Then with $\mathcal{P}_k$, $\mathcal{Q}_k$, $\mathcal{T}_k$ as in (1.5), (1.6) and (1.7), we have:

$$\mathcal{P}_{k-1}^2 - N\mathcal{Q}_{k-1}^2 = (-1)^k\mathcal{T}_k, \quad (k \geq 1)$$

Hence, $\mathcal{P}_{k-1}^2 \equiv (-1)^k\mathcal{T}_k\ (mod\ N)$. So, if $\mathcal{T}_k = y^2$, a perfect square, for some even k with $\mathcal{P}_{k-1} \not\equiv \pm y\ (mod\ N)$ then we have $gcd(\mathcal{P}_{k-1} + y, N)$ and $gcd(\mathcal{P}_{k-1} - y, N)$ as non-trivial

divisors of N. If we get trivial factors then look for another such $\mathcal{T}_k$ and try again.
If we do not arrive at required $\mathcal{T}_k$ after a lot of iterations, then we can look for set of $\mathcal{T}_{k_i}$ values making a perfect square, $(\mathcal{T}_{k_1}\mathcal{T}_{k_2}....\mathcal{T}_{k_r}) = y^2$ with $\sum_{i=1}^{r} k_i \ (mod\ 2) = 0$, such that:

$$y^2 = \mathcal{T}_{k_1}\mathcal{T}_{k_2}....\mathcal{T}_{k_r} \equiv (\mathcal{P}_{k_1-1}\mathcal{P}_{k_2-1}....\mathcal{P}_{k_r-1})^2 \ (mod\ N)$$

And then look at $gcd((\mathcal{P}_{k_1-1}\mathcal{P}_{k_2-1}....\mathcal{P}_{k_r-1}) \pm y, N)$ for non-trivial factors.

Example 1.33. *Take $N = 2059$. See that $\mathcal{T}_2\mathcal{T}_8 = 3^2 5^2$. Thus, required congruences are*

$$\mathcal{P}_1^2 = 91^2 \equiv (-1)^2 45 \ (mod\ 2059), \text{ and } \mathcal{P}_7^2 = 1758^2 \equiv 5 \ (mod\ 2059)$$

Hence, we get after reducing modulo 2059, $1435^2 \equiv 15^2 \ (mod\ 2059)$. So, divisors are $gcd(1435 \pm 15, 2059) = 29, 71$.

1.2.4 Quadratic Sieve Algorithm

The idea for Quadratic sieve algorithm is similar as in the factorization methods discussed above. To factor N we need to get a solution to $x^2 \equiv y^2 \ (mod\ N)$ and then compute $gcd(x-y, N)$ and $gcd(x+y, N)$ to see if these are non-trivial divisors of N. Here, we will use a quadratic polynomial $A(x) = x^2 - N$ to work with and compute the set $\mathcal{A} = A(x_1), A(x_2), ..., A(x_i)$ for some $x_i's$ and pick a subset $\mathcal{S} = A(x_{i_1}), A(x_{i_2}), ..., A(x_{i_r})$ such that $\prod_{j=1}^{r} A(x_{i_j}) = y^2$ for some $y \in \mathbb{Z}$. Then we get:

$$y^2 = A(x_{i_1})A(x_{i_2})....A(x_{i_r}) \equiv (x_{i_1}x_{i_2}....x_{i_r})^2 \ (mod\ N) \tag{1.8}$$

Hence, we check for $gcd(y - (x_{i_1}x_{i_2}....x_{i_r}), n)$ to be non-trivial. If not, then we look for another subset $\mathcal{S}$ and repeat the process.

For efficient implementation we need smaller $A(x_i)$ values which can be factored within a small factor base, say $\mathcal{F}$. For this we will calculate $A(x_i)$ for $x_i \in \mathcal{X}$ where $\mathcal{X} = [\lfloor\sqrt{N}\rfloor - \mathcal{M}, \lfloor\sqrt{N}\rfloor + \mathcal{M}]$ with $\mathcal{M} \in \mathbb{N}$. Here $[-\mathcal{M}, \mathcal{M}]$ is our *sieving interval*. Now, fix some bound $\mathcal{B} \in \mathbb{N}$ such that for prime $p \in \mathcal{F}$ we have $p \leq \mathcal{B}$. Also, any prime p will divide any $A(x_i)$ if and only if the Legendre symbol $\left(\frac{N}{p}\right) = 1$, so we will only add such primes to $\mathcal{F}$. To see this, assume for some $p \in \mathcal{F}$ that $p \mid A(x_i)$ with x_i as defined above. Then $A(x_i) \equiv 0 \ (mod\ p)$ which implies $x_i^2 \equiv N \ (mod\ p)$. So, N is a quadratic residue modulo p. Converse can be easily seen.

Let $\mathcal{F} = \{p_1, p_2, p_3, ..., p_k\}$ be our factor base satisfying above discussed conditions. Now let $\mathcal{A} = \{A(x_1), A(x_2), A(x_3), ..., A(x_r)\}$ such that all $A(x_i) \in \mathcal{A}$ are $\mathcal{F}$-

smooth with factorization as:

$$\begin{aligned}
A(x_1) &= (-1)^{e_{10}} p_1^{e_{11}} p_2^{e_{12}} p_3^{e_{13}} \dots p_k^{e_{1k}} \\
A(x_2) &= (-1)^{e_{20}} p_1^{e_{21}} p_2^{e_{22}} p_3^{e_{23}} \dots p_k^{e_{2k}} \\
&. \\
&. \\
A(x_r) &= (-1)^{e_{r0}} p_1^{e_{r1}} p_2^{e_{r2}} p_3^{e_{r3}} \dots p_k^{e_{rk}}
\end{aligned} \tag{1.9}$$

Now to find a set $\mathcal{S} \subset \mathcal{A}$ with product of its elements forming a perfect square we need to use Gaussian elimination in $\mathbb{Z}/2\mathbb{Z}$. Reduce all the powers of primes from (1.9) to modulo 2, means, $e_{ij} \leftarrow e_{ij}\ (mod\ 2),\ 1 \leq i \leq r,\ 0 \leq j \leq k$. Let $M_{r\times k+1}$ be the matrix of such values.

Use identity matrix $N_{r\times r}$ to keep track of rows of matrix M multiplied. Find first row in M with entry 1 in first column and add it to all other rows with 1 in first column and then remove that row from matrix M. Do the same with the rows of matrix N. Note that addition in both matrices is modulo 2. Again find such such row for column 2 and so on. See that after going through particular column, the rows below have zero in that column. We repeat the above steps until we get a row with all zero entries, which will eventually happen if $|\mathcal{A}| \geq |\mathcal{F}|$. After we have required row in M then check the same row in N, in which entries corresponds to the rows from M that were used to get the square, hence we now have our required set $\mathcal{S}$. Find factors using (1.8). If we do not get a non-trivial factor then we repeat the process further we find another such row. The code for algorithm is given in [A.2.4].

Example 1.34. *Let* $N = 9487$. *Choose our factor base to be* $\mathcal{F} = \{-1, 2, 3, 7, 11, 13, 17, 19, 29\}$. *See that for all primes* $p \in \mathcal{F}$ *we have* $\left(\frac{N}{p}\right) = 1$. *Choose* $\mathcal{M} = 16$.

x_i	$A(x_i)$	-1	2	3	7	11	13	17	19	29
81	$-2926 = -1.2.7.11.19$	1	1	0	1	1	0	0	1	0
84	$-2431 = -1.11.13.17$	1	0	0	0	1	1	1	0	0
85	$-2262 =-1.2.3.13.29$	1	1	1	0	0	1	0	0	1
89	$-1566 = -1.2.3^3.29$	1	1	1	0	0	0	0	0	1
95	$-462 = -1.2.3.7.11$	1	1	1	1	1	0	0	0	0
97	$-78 = -1.2.3.13$	1	1	1	0	0	1	0	0	0
98	$117= 3^2.13$	0	0	0	0	0	1	0	0	0
100	$513 = 3^3.19$	0	0	1	0	0	0	0	1	0
101	$714 = 2.3.7.17$	0	1	1	1	0	0	1	0	0
103	$1122= 2.3.11.17$	0	1	1	0	1	0	1	0	0
109	$2394 = 2.3^2.7.19$	0	1	0	1	0	0	0	1	0

First potential combination we get is $A(85), A(89), A(98)$, *from which we get* $(85.89.98)^2 \equiv (2.27.13.29)^2 \ (mod\ 9487) \implies 741370^2 \equiv 20358^2 \ (mod\ 9487)$ *but see that* $741370 \ (mod\ 9487) = 20358 \ (mod\ 9487)$. *So we have to look for another combination. We get another suitable values for* $A(81), A(95), A(100)$ *such that* $(81.95.100)^2 \equiv (2.9.7.11.19)^2 \ (mod\ 9487) \implies 1053^2 \equiv 7360^2 \ (mod\ 9487)$. *From this we get divisors* $gcd(1053 \pm 7360, 9487) = 179, 53$.

CHAPTER 2

THE AKS-ALGORITHM

The AKS-Algorithm is the first primality testing algorithm that is unconditional, deterministic, runs in polynomial time and works on any general number. The AKS Test is based on the following lemma for prime integers which is a generalisation of Fermat's theorem:

Lemma 2.1. *Let $n \in \mathbb{N} \geq 2$ and take some $a \in \mathbb{Z}$ such that $gcd(a, n) = 1$. Then n is prime if and only if*

$$(X + a)^n \ = X^n + a \ in \ \mathbb{Z}/n\mathbb{Z}[X]$$

Proof. Using binomial theorem:

$$(X + a)^n = X^n + \sum_{0<i<n} \binom{n}{i} a^i X^{n-i} + a^n.$$

First, assume n to be a prime integer then $\binom{n}{i}(mod\ n) = 0$, $0 < i < n$, and $a^n \equiv a\ (mod\ n)$ by Fermat's theorem. Thus $(X + a)^n = X^n + a$. Now, assume n to be a composite integer and take a prime integer p dividing n such that $p^s \mid n$ but $p^{s+1} \nmid n$ for some $s \geq 1$. Now take coefficient of X^{n-p}, $\binom{n}{p}a^p$, and from $p^s \nmid \binom{n}{p}$ and $gcd(a, n) = 1$ we have $p \nmid a^p$. Hence, $(X + a)^n \neq (X^n + a)$ in $\mathbb{Z}/n\mathbb{Z}[X]$. □

We will denote $\mathbb{Z}/n\mathbb{Z}[X]$ as $\mathbb{Z}_n[X]$ from here on. Reducing modulo n becomes time consuming as n increases because then we have to evaluate n coefficients in worst case in the LHS. Hence, a simple way to reduce coefficients of polynomials is to reduce modulo some polynomial $X^r - 1$ for appropriately small r. Then we will have to check the following equivalence relation:

$$(X + a)^n \ \equiv X^n + a\ (mod\ X^r - 1, n)$$

See that, $X^n + a\ (mod\ X^r - 1, n) = X^{n\ (mod\ r)} + a$. Hence, when n is prime, we have, in $\mathbb{Z}_n[X]$, $(X + a)^n (mod\ X^r - 1) = X^{n\ (mod\ r)} + a$ for every a and r. There are various bounds on r for which it is sufficient to run AKS-test. We will use log for base 2 logarithms and ln for base e in this chapter. One such lemma is given below without proof:

Lemma 2.2. *For all $n \in \mathbb{N} \geq 2$ we have a prime integer $r \leq 20\lceil log\ n\rceil^5$ such that either r divides n or $r \nmid n$ and $ord_r(n) > 4\lceil log\ n\rceil^2$, where $ord_r(n)$ denotes the smallest positive integer, say k, such that $n^k \equiv 1\ (mod\ r)$.*

Proof. Refer [13, page 120]. □

One another improvement in bound of r suggests that above lemma also holds for $r \leq 8\lceil log\ n\rceil^3(log(log\ n))^3$[13, page 121]. Now we will look at the main theorem proving that with certain conditions satisfied by integers a and r for some n, we can be sure that n is a power of a prime.

Theorem 2.3 (Main Theorem). *Take integer $n \geq 3$ and a prime integer $r < n$ satisfying the following conditions:*

a) $a \nmid n$ for $2 \leq a \leq r$

b) $ord_r(n) > 4(log(n))^2$ *, and*

c) $(X + a)^n \equiv X^n + a\ (mod\ X^r - 1, n)$ *for* $1 \leq a \leq 2\sqrt{r}log(n)$.

Then n is a power of a prime.

The proof of above theorem is build gradually from various lemmas and remarks that are given below.

Remark 2.4. *Assume $L = \lfloor 2\sqrt{r}log(n)\rfloor$ with $p \leq n/2$, a prime factor of n. Then we have:*
(i) $p > r$, and $r \nmid n$ (ii) $r > L$ (iii) $1 \leq (a' - a) < p$ for $1 \leq a < a' \leq L$
We can get other conditions from ii, which is as: $r > ord_r(n) > 4(log(n))^2$ which implies $\sqrt{r} > 2log(n)$. Hence, $r > 2\sqrt{r}log(n) \geq L$.

Definition 2.5. *For some polynomial $f(X) \in \mathbb{Z}_p[X]$ and $u \in \mathbb{N}$ we say that u is* introspective *for $f(x)$ if $[f(X)]^u \equiv f(X^u)\ (mod\ X^r - 1, p)$.*

Lemma 2.6.
1. *If u and u' are introspective numbers for polynomial $f(X)$, then mm' is also introspective for $f(X)$*
2. *If u is an introspective number for polynomials $f(X)$ and $g(X)$ then it is also introspective for $f(X)g(X)$.*

Proof. Refer [1, page 786-787]. □

Notice that the prime divisor p of n assumed in *Remark* 2.4 is used throughout the proof. Now we will construct sets P and U which will assist us in our proof further:

$$P = \left\{ \prod_{1 \le a \le L} (X+a)^{\beta_a} | \beta_a \ge 0 \; for \; 1 \le a \le L \} \right\} \subseteq \mathbb{Z}_p[X], \; and$$

$$U = \left\{ n^i p^j \; | i, j \ge 0 \right\}$$

Then from results of *Lemma* 2.6, we get the following lemma which doesn't require proof:

Lemma 2.7. *For some polynomial $f(X) \in P$ and $u \in U$ we have (in $\mathbb{Z}_p[X]$)*

$$f(X)^u \equiv f(X^u) \; (mod \; X^r - 1, p)$$

Now, recall some basis structures and properties from polynomials over rings and fields. Take $h \in \mathbb{Z}_p[X]$ to a be monic, irreducible polynomial such that $d = deg(h)$ which is equal to $order_r(p) > 1$. Also, as h is taken to be the divisor of $(X^{r-1} + X^{r-2} + ... + X + 1)$ then $h \mid (X^r - 1)$. And take finite field $F = \mathbb{Z}_p[X]/(h)$. Then we have $|F| = p^d$. Now, take subgroup $G = \{f(X) \; mod \; h | f(X) \in P\}$ of F^*, the multiplicative group. Now, let us see the following proposition regarding our field F.

Proposition 2.8. *Linear polynomials $X + a$, $1 \le a \le p$ are different in $\mathbb{Z}_p[X]$ and in F, and $X + a \; (mod \; h) \ne 0$.*

Proof. As $(X + a') - (X + a) = a' - a > 0$ for $1 \le a < a' \le L$, by (iii), *Remark* 2.4. Then $X + 1, X + 2, ..., X + L$ are distinct in $\mathbb{Z}_p[X]$ and F.
Now if $h \mid (X + a)$. Since h is monic and non-constant we have $h = X + a$, which is not possible as $deg(h) = order_r(p) > 1$. Hence, $h \nmid (X + a)$. □

Now take the root of polynomial $h(X)$, $\zeta = X \; mod \; h$ and recall that it is also the primitive r^{th} root of unity in $\mathbb{Z}_p$ (*i.e.* $\zeta^r = 1$ and $\zeta^{r'} \ne 1 \; \forall \; r' < r$).

Lemma 2.9. *Let $g(X) \in G$, (i.e. $g(X) = f(X) \; mod \; h(X)$), then in $F = \mathbb{Z}_p[X]/(h)$ we have $g^u = f(\zeta^u) \; \forall \; u \in U$.*

Proof. Clearly (in $\mathbb{Z}_p[X]$), $g^u \equiv f^u \; (mod \; h)$. Then By *Lemma* 2.7 we have $f^u \equiv f(X^u) \; (mod \; X^r - 1)$. As $h \mid (X^r - 1)$, we have,

$$f^u \equiv f(X^u) \; (mod \; h) \tag{2.1}$$

From $\zeta \equiv X \ (mod\ h)$ we have $X^u \equiv \zeta^u \ (mod\ h)$. On substituting it in above equation, we get

$$g^u \equiv f^u \equiv f(\zeta^u \ mod\ h) \ (mod\ h) = f(\zeta^u) \ (mod\ h)$$

Hence, in F, $g^u = f(\zeta^u) \ \forall \ u \in U$. $\square$

Remark 2.10. *If two polynomials f_1 and f_2 are distinct elements of P, then $f_1 \not\equiv f_2 \ (mod\ h)$. [13, page 128-129]*

Let's define another set $T = \{\zeta^u \mid u \in U\}$, $|T| = t$, which will be helpful in following *Lemmas*.

Lemma 2.11. *1. $r > t > 4(log\ n)^2$*

2. $|G| > \frac{1}{2} n^{2\sqrt{t}}$

3. If $U_0 = \{n^i p^j \mid 0 \le i, j \le \lfloor \sqrt{t} \rfloor\} \subseteq U$. then $|U_0| \le t$

Proof. 1. As $\zeta^r = 1$ we have $T \subseteq \langle \zeta \rangle = \{1, \zeta, ..., \zeta^{r-1}\}$. And $\zeta^u \ne 1 \ \forall \ u \in U$ (Note: $r \nmid n^i p^j$ for any $i, j \ge 0$). Hence, $t = |T| \le (r-1)$. Now we have $\zeta^{n^i} \ne \zeta^{n^k}$ iff $n^i \not\equiv n^k \ (mod\ r)$. Thus, by definition we have:

$$|\{\zeta^{n^i} \mid i \le 0\}| = |\{n^i \ mod\ r \mid i \ge 0\}| = ord_r(n)$$

Thus $t = |T| \ge order_r(n) > 4(log(n))^2$.

2. Take $\mu = min\{L, t-1\}$, and take

$$p_i, \ p_j \in \prod_{1 \le a \le \mu} (X + a)^{\beta_a}, \ \beta_a \in \{0, 1\} \ for \ 1 \le a \le \mu$$

So, $p_i, \ p_j \in P$ and $p_i \ne p_j$ in $\mathbb{Z}_p[X]$ and P, as these are products of different sets of irreducible factors. By *Remark* 2.10, $p_i \not\equiv p_j \ (mod\ h) \implies |G| \ge 2^\mu$. So, we have two cases:
Case 1: $\mu = L$, as $r > t$ from proof of 1, we have $\mu = \lfloor 2\sqrt{r} log\ n \rfloor > 2\sqrt{r} log\ n - 1 > 2\sqrt{t} log\ n - 1$.
Case 2 : $\mu = t - 1$, as $t > 4(log\ n)^2$ [From 1], $\mu = t - 1 > 2\sqrt{t} log\ n - 1$. So, in both cases, $|G| \ge 2^\mu > 2^{2\sqrt{t} log\ n - 1} = \frac{1}{2} n^{2\sqrt{t}}$.

3. First, we will prove the following claim:
Claim : $\forall \ u \in U_0, u < |G|$.

Proof : With $p \leq n/2, i, j \leq \lfloor \sqrt{t} \rfloor$ we have

$$n^i p^j \leq n^{\sqrt{t}} \left(\frac{1}{2}n\right)^{\sqrt{t}} = \left(\frac{1}{2}n^2\right)^{\sqrt{t}} \leq \frac{1}{2}n^{2\sqrt{t}} < |G|$$

Now, as $t = |T| = |\{\zeta^u \mid u \in U\}|$, if we show mapping $u \longmapsto \zeta^u$ to be injective we get $|U_0| \leq |T| = t$. Assume, for distinct $u, v \in U_0, \zeta^u = \zeta^v$. And let $g = f \; mod \; h \in G$. Then by *Lemma* 2.9 we have:

$$g^u = f(\zeta^u) = f(\zeta^v) = g^v$$

Hence, $g \in G$ is a root of $X^u - X^v \in \mathbb{Z}_p[X]$. But it means that all elements of G are roots of $X^u - X^v$. As $deg(X^u - X^v) \leq max\{u, v\} < |G|$ from claim above we have $X^u - X^v$ a zero polynomial which implies $u = v$, a contradiction.

□

Now we will give the proof of our main theorem:

Main Theorem Proof. From definition, $|U_0| = (\lfloor \sqrt{t} \rfloor + 1)^2 > (\sqrt{t})^2 = t$ for $0 \leq i, j \leq \sqrt{t}$. But from (3)-*Lemma* 2.11, we have $|U_0| \leq t$. Then by pigeonhole principle there exist distinct pairs $(i, j), (k, m)$ such that $n^i p^j = n^k p^m$. But $i \neq k$ (if so, $j = m \implies (i, j) = (k, m)$). Thus, we have

$$n^{i-k} = p^{m-j};\ i > k,\ m > j$$

Therefore, n does not have any prime factor other than p. □

The algorithm structure is as given in [A.1.5], which if runs for some integer $n \geq 2$ then output is prime if and only if n is a prime integer. [13, page 122-123]

CHAPTER 3

THE LLL-ALGORITHM

The LLL-Algorithm is an important algorithm for lattice basis reduction which gives an α-reduced lattice basis of any normal basis of a lattice in $\mathbb{R}^n$. We will discuss in brief its application in solving closest vector problem and its variants with deep insertion and linearly dependent vectors. Let's start with the definition of lattice.

Definition 3.1. *A subset L of $\mathbb{R}^n$ with basis $\{x_i\}_{i=1}^n$ of $\mathbb{R}^n$ is called a lattice if*

$$L = \left\{ \sum_{i=1}^{n} r_i x_i : Where\ r_i \in \mathbb{Z}\ \forall\ 1 \leq i \leq n \right\}$$

$\{x_i\}_{i=1}^n$ is called the basis of L and n the rank of lattice L.

Note : We will work with only full-rank lattices unless specified, means, n-dimensional lattices in $\mathbb{R}^n$.
Remark : If $\{x_1, x_2, ..., x_n\}$ and $\{y_1, y_2, ..., y_n\}$ are two bases for lattice L in $\mathbb{R}^n$ with corresponding matrices X and Y (vectors as rows). Then $Y = CX$ for $C_{n\times n}$ $s.t.$ $C_{ij} \in \mathbb{Z}$ with $det(C) = \pm 1$ (**Unimodular**).

3.1 GRAM-SCHMIDT ORTHOGONALIZATION(GSO)

GSO is used to convert a given arbitrary basis vector, $\{x_1, x_2, ..., x_n\}$ of $\mathbb{R}^n$, into a new orthogonal basis vector.

Definition 3.2. *Let $\{x_1, x_2, ..., x_n\}$ be a basis in $\mathbb{R}^n$, then the GSO of this basis is the vector $\{x_1^*, x_2^*, ..., x_n^*\}$ defined as :*

$$x_1^* = x_1$$
$$x_2^* = x_2 - \frac{< x_2, x_1^* >}{< x_1^*, x_1^* >} x_1^*$$
$$x_n^* = x_n - \sum_{j=1}^{n-1} \mu_{nj} x_j^*$$

Where,

$$\mu_{nj} = \frac{< x_n, x_j^* >}{< x_j^*, x_j^* >}$$

Where $< x_i, x_j >$ is inner dot product of vectors x_i and X_j.

Note 3.3. *See that* $||x_k^*|| \leq ||x_k||$ *for* $1 \leq k \leq n$. *As* $||x_k||^2 = ||x_k^* + \sum_{j=1}^{k-1} \mu_{kj} x_j^*||^2 = ||x_k^*|| + \delta$ *where* $\delta = \sum_{j=1}^{k-1} \mu_{kj}^2 ||x_j^*||^2$ *by orthogonality of GSO vectors.*

Theorem 3.4 (Hadamard's Inequality)**.** *Let* $X = (x_{ij})$ *be* $n \times n$ *matrix over* $\mathbb{R}$ *and let* $\mathcal{B} = max_{i,j}\{|x_{ij}|\}$. *Then* $|det(X)| \leq n^{n/2}\mathcal{B}^n$.

Proof. If we denote rows by $(x_i), 1 \leq i \leq n$, then $det(X) = 0$ if rows are linearly dependent. If not then assume $X^* = (x_i^*)$ be matrix by GSO vectors. Then as we know $det(X) = det(X^*)$. See that $det(X^*(X^*)^T) = \|x_1^*\|^2 ... \|x_n^*\|^2$ by orthogonality of GSO vectors. Then we have $|det(X^*)| = \|x_1^*\| ... \|x_n^*\|$. By $Note - 3.3$, we see that $|det(X)| = |det(X^*)| \leq \|x_1\| ... \|x_n\| \leq (n\mathcal{B}^2)^{n/2} = n^{n/2}\mathcal{B}^n$. □

Proposition 3.5. *Let* $\{x_i\}_{i=1}^n$ *be a basis of* $\mathbb{R}^n$ *with corresponding GSO vector being* $\{x_i^*\}_{i=1}^n$. *Let L be the lattice generated by* $\{x_i\}_{i=1}^n$, *then for any non-zero* $y \in L$ *we have:*

$$||y|| \geq min\{||x_1^*||, ||x_2^*||, ..., ||x_n^*||\}$$

Proof. See [4, page 51]. □

Now we will give definition of an α-reduce lattice basis vector for which first define α to be our **Reduction Parameter** such that $\frac{1}{4} < \alpha < 1$, Standard value used for α is $\frac{3}{4}$.

Definition 3.6. *Let* $\{x_i\}_{i=1}^n$ *be an ordered basis of lattice L in* $\mathbb{R}^n$, *with GSO being* $\{x_i^*\}_{i=1}^n$ *and* μ_{ij} *as defined above, it is called LLL-reduced with parameter* α *if it satisfies:*

$$|\mu_{ij}| \leq \frac{1}{2} \qquad for\ 1 \leq j < i \leq n \tag{3.1}$$

$$||x_i^* - \mu_{i,i-1}x_{i-1}^*||^2 \geq \alpha||x_{i-1}^*||^2 \qquad for\ 2 \leq i \leq n \tag{3.2}$$

Condition 3.1 is $SizeCondition$ whereas 3.2 is called $LovaszCondition$.

Theorem 3.7 (LLL-Theorem)**.** *Let* $\{x_i\}_1^n$ *be an* α*-reduced lattice basis for L. Then for any non-zero* $y \in L$ *we have:*

$$\|x_1\| \leq \beta^{(n-1)/2} \|y\|$$

Proof. From 3.6 we see that $\|x_i^*\|^2 \geq \frac{1}{\beta} \|x_{i-1}^*\|^2$. So, $\|x_1\|^2 = \|x_1^*\|^2 \leq \beta \|x_2^*\|^2 \leq .... \leq \beta^{(n-1)} \|x_n^*\|^2$. Hence, for $1 \leq i \leq n$, $\|x_i^*\|^2 \geq \beta^{(1-i)} \|x_1\|^2$. By proposition 3.5, $\|y\| \geq$

$min\{\|x_1^*\|, \|x_2^*\|, ..., \|x_n^*\|$. Let $\|x_i\| = max\{\{\|x_i\|\}_{i=1}^n\}$. Then, $\|y\| \geq \beta^{(1-i)/2}\|x_1\| \geq \beta^{(1-n)/2}\|x_1\|$. □

3.2 The LLL-Algorithm

First look at the function *reduce*(k,l) which is called when *SizeCondition* is not satisfied. It performs best possible reduction as $\lfloor \mu_{kj} \rceil$ is the nearest integer to μ_{kl}. See that after reducing y_k it still belongs to the basis.

Algorithm 1 reduce$(\mathbf{k},\mathbf{j})$ for reducing μ_{kj} $(j < k)$

```
if $\mu_{kj} > \frac{1}{2}$ then
    set $y_k \leftarrow y_k - \lfloor \mu_{kj} \rceil y_j$
    for $l = 1, 2, ..., j-1$ do
        set $\mu_{kl} \leftarrow \mu_{kl} - \lfloor \mu_{kj} \rceil \mu_{jl}$
    end
    set $\mu_{k,j} \leftarrow \mu_{kj} - \lfloor \mu_{kj} \rceil$
else
    continue
end
```

Lemma 3.8. *Assume $\{y_k^*\}_{k=1}^n$ be GSO vectors for lattice basis $\{y_k\}_{k=1}^n$. Then on calling function* **reduce$(\mathbf{k},\mathbf{j})$***, vectors $\{y_k^*\}_{k=1}^n$ remain unchanged.*

Proof. On calling *reduce*(k,j) we do $y_k \leftarrow y_k - \lfloor \mu_{kj} \rceil y_j$ *i.e.* let $z_k = y_k - \lfloor \mu_{kj} \rceil y_j$. Now, z_k^* is calculated as :

$$\begin{aligned}
z_k^* &= z_k - \sum_{l=1}^{k-1} \frac{< z_k, y_l^* >}{< y_l^*, y_l^* >} y_l^* \\
&= (y_k - \lfloor \mu_{kj} \rceil y_j) - \sum_{l=1}^{k-1} \frac{< y_k - \lfloor \mu_{kj} \rceil y_j, y_l^* >}{< y_l^*, y_l^* >} y_l^* \\
&= (y_k - \lfloor \mu_{kj} \rceil y_j) - \sum_{l=1}^{k-1} \mu_{kl} y_l^* + \Big(\sum_{l=1}^{j-1} \mu_{jl} y_l^* + y_j^*\Big) \lfloor \mu_{kj} \rceil \\
&= \Big(y_k - \sum_{l=1}^{k-1} \mu_{kl} y_l^*\Big) - \lfloor \mu_{kj} \rceil y_j + \lfloor \mu_{kj} \rceil y_j \\
&= y_k^*
\end{aligned}$$

□

Above lemma shows us that we only need to update μ_{kl} ; $l = 1, 2, ..., j$, as implemented in function $reduce(k, j)$. The calculation can be easily done by replacing y_k as $y_k - \lfloor \mu_{kj} \rceil y_j$ in the definition of μ_{kl}.

Algorithm 2 The LLL-Algorithm

```
Data: Basis $\{x_i\}_{i=1}^n$ of lattice $L \in \mathbb{R}^n$ with reduction parameter $\alpha$
Result: An $\alpha - reduced$ basis $\{y_i\}_{i=1}^n$ of $L$.
set $y_i \leftarrow x_i$
calculate $GSO, i.e. \{y_i^*\}_{i=1}^n$
set $k \leftarrow 2$
while $k \leq n$ do
    call reduce$(\mathbf{k}, \mathbf{k-1})$
    if LovaszCondition(k) then
        For $j = k-2, .., 1$ ; Do : Call reduce$(\mathbf{k}, \mathbf{j})$
        set $k \leftarrow k + 1$
    else
        swap$(y_k, y_{k-1})$
        update $GSO$
        set $k \leftarrow max(k-1, 2)$
    end
end
```

Our loop for *LovaszCondition* does not increase k if condition is not met, we swap vector y_k and y_{k-1} instead and reduce k by 1 and then run the loop again. If the condition is not met again then we keep exchanging the vectors and update GSO in every step. Actually, we do not need to update whole GSO but only $\mu'_{ij}s$ and new y_k^* and y_{k-1}^* which will be clear from lemma below.

Lemma 3.9. *Let Y and Z be the basis before and after exchanging vectors at positions k and $k-1$ respectively. And let Y^* and Z^* be their corresponding GSO vectors. Then we get:*

1. $z_i^* = y_i^*$ $(i \neq k-1, k)$
2. $||z_{k-1}^*||^2 < \alpha ||y_{k-1}^*||^2$

Proof. We have all the vectors same excepts from the exchanged vectors so $z_i = y_i \forall i \neq (k-1), k$. And as by definition, y_i^* and z_i^* are the projections of y_i and Z_i respectively onto orthogonal complement of the $span(\{y_j\}_{j=1}^{i-1})$ and $span(\{Z_j\}_{j=1}^{i-1})$ respectively. But $span(\{y_j\}_{j=1}^{i-1})$ $=span(\{Z_j\}_{j=1}^{i-1}) \implies (1)$.
We have $z_{k-1}^* = y_k - \sum_{l=1}^{k-2} \mu_{kl} y_l^*$. Putting value $y_k = y_k^* + \sum_{l=1}^{k-1} \mu_{kl} y_l^*$, we get $z_{k-1}^* = y_k^* + \mu_{k(k-1)} y_{k-1}^* \implies \left\|z_{k-1}^*\right\|^2 = \left\|y_k^*\right\|^2 + \mu_{k(k-1)}^2 \left\|y_{k-1}^*\right\|^2$. Now, as $LovaszCondition(k)$ is not satisfied, we have $\left\|y_k^*\right\|^2 < (\alpha - \mu_{k(k-1)}^2) \left\|y_{k-1}^*\right\|^2$. Which implies that $\left\|z_{k-1}^*\right\|^2 < (\alpha - \mu_{k(k-1)}^2) \left\|y_{k-1}^*\right\|^2 + \mu_{k(k-1)}^2 \left\|y_{k-1}^*\right\|^2 = \alpha \left\|y_{k-1}^*\right\|^2$. □

Now, we know know that if the algorithm terminates then resulting set of vectors will be our reduced basis. So, now we prove that the steps in our algorithm have finite upper bound.

Closest Vector Problem(CVP): For a lattice $L \in \mathbb{R}^n$ which basis $\{x_i\}_{i=1}^n$, CVP is the problem of finding a closest vector $y \in L$ to some vector $z \in \mathbb{R}^n$. Let $\{x_i^*\}_{i=1}^n$ be the GSO of basis vectors of L. Now consider the sublattice $L' = L \cap U$ with basis $\{x_i\}_{i=1}^{n-1}$ where $U \subset \mathbb{R}^n$ is the hyperplane. Nearest plane algorithm suggests that for any given arbitrary vector $z \in \mathbb{R}^n$ we can find a vector $y \in L$ such that the orthogonal distance between z and translated hyperplane $U + y$ is as small as possible.
So, if $z = \sum_{i=1}^n a_i x_i^*$ then take the nearest integer to a_n, $\lfloor a_n \rceil \in \mathbb{Z}_n$. Thus we have $z^* = \sum_{i=1}^{n-1} a_i x_i^* + y$, where $y = \lfloor a_n \rceil x_n \in L$, the orthogonal projection of z into $U + y$. Then see that $z^* - y \in L$. Now, recursively find vector $y' \in L'$ closest to $z - y$ and set $y \leftarrow y' + y$. In the end we will have to find the closest integer multiple of one non-zero real number to another. For details refer [2].

3.3 LLL-Algorithm with Deep Insertion and for Linearly dependent Vectors

I. In case of deep insertion instead of $swap(y_k, y_{k-1})$ we swap y_k with y_i vector such that i is the least such index not following the $LovaszCondition$. Doing this comes up with an computational disadvantage as now instead of just updating $|y_i^*|^2$ we have to update whole GSO again and in worst case it is no longer in polynomial time. Advantage of deep insertion that Schnorr and Euchner [7] reported is that in many cases it outputs considerably shorter lattice vectors than original algorithm.

II. Take lattice $L \in \mathbb{R}^n$ with dimension $dim(L) \leq n$ and some linearly dependent vectors $\{x_i\}_{i=1}^m$. And take the standard basis $\{e_1, e_2, ..., e_m\}$ in $\mathbb{R}^m$. Then see that we have the following linearly independent embedded vectors in R^{m+n}:

$$[e_1, x_1], [e_2, x_2], ..., [e_m.x_m]$$

And define vectors $y_i \in \mathbb{R}^{m+n}$, $1 \leq i \leq m$, as $y_i = [e_i, \beta^l x_i]$ where l is some positive integral scaling factor. Recall that $\beta = 4/(4\alpha - 1)$. See that y_i are linearly independent vectors. Now take L' be the lattice in $\mathbb{R}^{m+n}$ with $y_i's$ as basis vectors with $\{z_i\}_{i=1}^m$ being their reduced basis vectors. Now take

$$M = min\{||x||^2 \ where, \ x \in L, x \neq 0\}$$
$$N = min\{||y||^2 \ where, \ y \in L', y \neq 0\}$$

Then by *Theorem* 3.7, we have $||z_1|| \leq \beta^{m-1}N$ where $z_1 = [c, \beta^l x]$ for some $c \in \mathbb{Z}^m$ and $x \in L$. Hence, $||z_1||^2 = ||c||^2 + \beta^{2l}||x||^2$. Thus,

$$||c||^2 + \beta^{2l}||x||^2 \leq \beta^{m-1}N$$

Now we want to choose l such that above inequality implies $x = 0$. In other words, if l satisfies

$$\beta^{2l}M \geq \beta^{m-1}N \tag{3.3}$$

then we can say that $x = 0$. And as $\{x_i\}_{i=1}^m$ are linearly dependent then we have $\sum_{i=1}^m c_i x_i = 0$ for some vector $[c_1, ..., c_m] \neq \mathbf{0}$. And as $[c_1, ..., c_m, 0, ..., 0] \in L'$, we have $N \leq mC^2$. Therefore, from *equation* 3.3 we get:

$$\beta^{2l}M \geq \beta^{m-1}mC^2$$
$$\implies l \geq \frac{1}{2}\left((m-1) + log_\beta\left(\frac{mC^2}{M}\right)\right)$$

For such l we will have $x = 0$ and hence, $z_1 = [c, 0, 0, 0, ..., 0]$. One of the major problem with this modification is that it is very hard to find the values of M and C which are required for l bound.

3.4 LLL-Algorithm Termination

Choose $\{x_i\}_{i=1}^k$ out of $\{x_i\}_{i=1}^n$, $k \leq n$, basis vectors. Then we define $k^{th}-$ gram determinant to be :

$$d_k = det(G_k G_k^t) = \prod_{i=1}^k ||x_i^*||^2 \qquad [16]$$

Where G_k is the matrix with k-basis vectors, $\{x_i\}_{i=1}^k$ as rows and $\{x_i^*\}_{i=1}^k$ are corresponding GSO vectors.

Lemma 3.10. *Function* $\mathbf{reduce}(\mathbf{k}, \mathbf{j})$ *does not change* $\{d_i\}_{i=1}^n$ *and after* $swap(y_k, y_{k-1})$, d_i *changes only for* $i = k-1$ *s.t* $d'_{k-1} \leq \alpha d_{k-1}$.

Proof. First part can be seen directly from 3.8. Now for second part, after swapping y_k and y_{k-1}, $\{y_i^*\}_{i=1}^{k-2}$ remains same so no change in $\{d_i\}_{i=1}^{k-2}$. Now for $i \geq k$, let d'_i be new $i^{th}-$ gram determinant. We swap rows y_k and y_{k-1} in G_i and corresponding columns in $G_i^t \implies d'_i = (-1)^2 det(G_i G_i^t) = det(G_i G_i^t) = d_i$.
Now for $i = k-1$, if $\{y_i^*\}_{i=1}^{k-1}$ and $\{z_i^*\}_{i=1}^{k-1}$ are $GSO's$ before and after $swap(y_k, y_{k-1})$ then,

$$
\begin{aligned}
d'_{k-1} &= \prod_{i=1}^{k-1} ||z_i^*|| \\
&= ||z_{k-1}^*||^2 \prod_{i=1}^{k-2} ||y_i^*||^2 \quad (By\ 3.9(1)) \\
&\leq \alpha ||y_{k-1}^*||^2 \prod_{i=1}^{k-2} ||y_i^*||^2 \quad (By\ 3.9(2)) \\
&\leq \alpha d_{k-1}
\end{aligned}
$$

□

Lemma 3.11. *Define* $D = \prod_{k=1}^{n-1} d_k$ *with* D_0 *being its value at the start of* $LLL - Algorithm$. *Then* $D_0 \leq \mathcal{B}^{n(n-1)}$ *where* $\mathcal{B} = max(\{|x_i|\}_{i=1}^n)$, $\{x_i\}_{i=1}^n$ *being the input basis vectors.*

Proof.

$$
\begin{aligned}
D_0 &= \prod_{k=1}^{n-1} d_k \\
&= (||x_1^*||)(||x_1^*||||x_2^*||)...(||x_1^*||||x_2^*||...||x_{n-1}^*||) \\
&= \prod_{k=1}^{n-1} ||x_k^*||^{2(n-k)} \\
&\leq \prod_{k=1}^{n-1} ||x_k||^{2(n-k)} \\
&\leq \prod_{k=1}^{n-1} \mathcal{B}^{2(n-k)} \\
&\leq \mathcal{B}^{2(n-1+n-2+....+1)} = \mathcal{B}^{n(n-1)}
\end{aligned}
$$

□

Remark 3.12. *There exists non-zero $z \in L$ s.t. $||z||^2 \leq \left(\frac{4}{3}\right)^{\frac{i-1}{2}} d_i^{\frac{1}{i}}$ $(1 \leq i \leq n)$[8, page 31]. So, $D > 0$ always. Hence, $d_i \geq \left(\frac{3}{4}\right)^{\frac{i(i-1)}{2}} ||z||^2 > 0$ where z is the shortest non-zero vector in lattice L.*

Theorem 3.13 (Termination)**.** *Total number of passes through 'LovaszCondition(k)' loop is atmost*

$$\frac{-2\log \mathcal{B}}{\log \alpha} n(n-1) + (n-1)$$

Proof. Let E be the total number of calls to $swap(y_k, y_{k-1})$ and D as defined in 3.11. Then by 3.10, D decreases to atmost αD after each call. Then after E passes we have $D \leq \alpha^E D_0$. By use of *Remark* 3.12 we can write:

$$\begin{aligned}
\alpha^{-E} &\leq D_0 \\
\alpha^{-E} &\leq \mathcal{B}^{n(n-1)} \quad (By\ Lemma\ 3.11) \\
-E\ log\ \alpha &\leq n(n-1) log\ \mathcal{B} \\
\implies E &\leq \frac{-log\ \mathcal{B}}{log\ \alpha} n(n-1) \quad (as - log\ \alpha > 0)
\end{aligned}$$

If E' are the total number of passes for $reduce(k, j)$, then $E + E'$ are the total required passes. See that $E \leftarrow E + 1 \implies k \leftarrow k - 1$ and $E' \leftarrow E + 1 \implies k \leftarrow k + 1$. Thus, $E - E' + k$ is always a constant and as $k = 2$ in the start when $E = E' = 0$, we have $E - E' + k = 2$ always. Now, at the end $k = n + 1$ so, $E - E' = (n - 1)$ which implies that

$$E + E' = 2E + (n-1) = -\frac{2log\ \mathcal{B}}{log\ \alpha} n(n-1) + (n-1)$$

as required. □

CHAPTER 4

Polynomial Factorization

In this section we will describe algorithm for factoring polynomials with coefficients in ring $\mathbb{Z}$ (or $\mathbb{Q}$) and field $\mathbb{Z}/p\mathbb{Z}$. Factorization over base rings $\mathbb{Z}$ and $\mathbb{Q}$ is equivalent which is shown in *section* 4.2. For factorization over $\mathbb{Z}$, we first need factor polynomials over finite field $\mathbb{Z}/p\mathbb{Z}$ for some prime p and then work with obtained factors to get to irreducible factors over $\mathbb{Z}$. We will start with the following definition.

Definition 4.1. *Let polynomial $A = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$, then we define* ***Content*** *of A to be $gcd(a_0, a_1, ..., a_n)$ and denote it as $cont(A)$. We say polynomial A is primitive if $cont(A) = 1$.*

Note 4.2. *We denote formal derivative of a polynomial A as A' and we will denote polynomials as A and $A(x)$ interchangeably. Assume that we have algorithms for polynomial GCD at our disposal [9, page 113-117].*

4.1 Polynomial Factorization in field $\mathbb{F}_p$

Field $\mathbb{Z}/p\mathbb{Z}$ is denoted as $\mathbb{F}_p$ and all polynomials are taken in single variable. Take $A(x) \in \mathbb{F}_p[X]$, primitive and monic. Being in field does not restrict the generality as all elements have units. First we do a square-free factorization of A *i.e.* $A = \prod_{i=1}^{k} A_i(x)^i$, where k is the total number of square free factors $A_i's$ with multiplicity i, such that $gcd(A_i, A_j) = 1$, $i \neq j$, for $i, j \in \{1, 2, ..., k\}$. Then we perform a distinct degree factorization on each square-free factor as factors that are product of similar degree irreducible factors. Then we will finally factor each polynomial obtained from distinct degree factorization into irreducible polynomials of equal degree by using a method given by Cantor and Zassenhaus.

I. Yun's Square-free Factorization: Let $A = \prod_{i=1}^{k} A_i^i$, $A, A_i \in \mathbb{F}_p[X]$. See that $T = gcd(A, A') = \prod_{p\nmid i} A_i^{i-1} \prod_{p|i} A_i^i$, $i \in \{1, 2, ..., k\}$. Now take:

$$W = \frac{A}{T} = \prod_{p\nmid i} A_i \ \ , \ and$$

$$Y = \frac{A'}{T} = \sum_{i=1(p\nmid i)}^{k} iA_i' \prod_{j=1(j\neq i, p\nmid j)}^{k} A_j \ \ , \ and$$

$$Z = Y - W' = A_1 \left(\sum_{i=2,(p\nmid i)}^{k} (i-1)A_i' \prod_{j=2(\neq i, p\nmid j)}^{k} A_j \right), \ We\ get$$

$$gcd(W, Z) = A_1$$

Similarly, repeating the above process by upgrading values in order as, $W \leftarrow \frac{W}{A_1}$, $Y \leftarrow \frac{Z}{A_1}$ and $Z \leftarrow Y - W'$. Hence, we will get $gcd(W, Z) = A_2$ and so on.

Steps for algorithm are as follows:

SFF$(\mathbf{A}(\mathbf{x}))$: Returns square-free factors of primitive and monic $A(x)$ as shows above.

1. set $T \leftarrow gcd(A, A')$
2. set $W \leftarrow A(x)/T(x), Y \leftarrow A'(x)/T(x)$
3. set $Z \leftarrow Y(x) - W'(x)$
4. set $A_1 \leftarrow gcd(W, Z)$
5. Store A_1 and set $A \leftarrow A/A_1$
6. $a)$ set $W \leftarrow W/A_1, Y \leftarrow Z/A_1$
   $b)$ repeat steps (3) to (6a) until we get $A = 1$ in step (5).

II. Distinct Degree Factorization: We now have square-free and primitive $A_i's$ which we will factors as $A_i = \prod A_{id}$ where each $A_{id} = \prod_j f_{jd}$, $f_{jd}'s$ are irreducible factors with degree d and j runs over all such factors.

Let $f_d \in \mathbb{F}_p[X]$ be an irreducible factor $s.t.\ deg(f_d) = d$. Then consider the field K generated by f_d, $K = \mathbb{F}_p[X]/f_d(x)\mathbb{F}_p[X]$ with $|K| = p^d$. Now, $\forall\ x \in K$ we have $x^{p^d} = x$ as order of any non-zero element from multiplicative group K^* divides $p^d - 1$. Hence, polynomial $X^{p^d} - X = 0$ in K which means $f_d \mid X^{p^d} - X$.

Remark 4.3. *Every irreducible factor of $X^{p^d} - X$ that is not a factor of $X^{p^e} - X$ for any $e < d$ has degree exactly d.*

Now below are the steps for distinct degree factorization of each A_i:

DDF(A_i): Returns factors of A_i, A_{id}, each of which is a product of irreducible factors of degree d.

1. Set $d \leftarrow 1$. Then calculate $A_{id} \leftarrow gcd(A_i, X^{p^d} - X)$.
2. Store A_{id}, d and then set $A_i \leftarrow \frac{A_i}{A_{id}}$.
3. Make $d \leftarrow d + 1$. If $A_i = 1$ **Terminate**, else repeat *Step 1 to 3*.

Note 4.4. *If p^d is very large, we should calculate $gcd(A_i, X^{p^d} - X)$, as in step 1, using the following steps:*
$a)$. set $B \leftarrow X^p \ (mod\ A_i)$
$b)$. Repeatedly set $B \leftarrow B^p \ (mod\ A_i)$, $(d-1)$ times.
See that every time we are raising some polynomial B, with $deg(B) < A_i$, to power p. If $A_i(x)$ has large degree then we can raise the power in smaller steps in step (b) to reduce the running time of algorithm.

III. Equal-Degree Factorization: Now we have each A_{id} as product of irreducible factors f_d with $deg(f_d) = d$. So, we need to factor A_{id} to find all irreducible factors and we will have have full factorization of A. We have $deg(A_{id}) = kd$ where k is the total number of irreducible factors f_d. See that if $deg(A_{id}) = d$ then we are done so assume otherwise. We denote finite field with p^d elements by $\mathbb{F}_{p^d}$. We will factor A_{id} for $p < 2$ and $p > 2$ separately by following propositions:

For $p > 2$:

Proposition 4.5. *Take A_{id} as above, then for any $T \in \mathbb{F}_p[X]$ we have*

$$A_{id} = gcd(A_{id}, T).gcd(A_{id}, T^{\frac{p^d-1}{2}} + 1).gcd(A_{id}, T^{\frac{p^d-1}{2}} - 1)$$

Proof. As roots of $X^{p^d} - X$ are elements of field $\mathbb{F}_{p^d}$, they are distinct.
Claim: The set of roots of $T^{p^d} - T$ contains the roots of $X^{p^d} - X$ (or elements of F_{p^d}).
Proof: Assume $T(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$ where each $a_i \in \mathbb{F}_p$. Then $T^{p^d} = a_n^{p^d} x^{p^d n} + a_{n-1}^{p^d} x^{p^d n-1} + ... + a_0^{p^d}$. Hence, $a_i \in \mathbb{F}_p \implies a_i^p \equiv a_i \ (mod\ p) \implies$

$a_i^{p^d} \equiv a_i \ (mod\ p)$ and let $x \in \mathbb{F}_{p^d}$ then $x^{p^d} = x$ in $\mathbb{F}_{p^d}$. So, for such x, we have $T^{p^d} = a_n x^n + a_{n-1}x^{n-1} + ... + a_0 = T$.
Our claim implies that $X^{p^d} - X \mid T^{p^d} - T$. And as we saw earlier that for all irreducible $f_d \in \mathbb{F}_p[X]$, with $deg(f_d) = d$, $f_d \mid X^{p^d} - X \implies f_d \mid T^{p^d} - T$. Note that A_{id} is square-free and all $f_d's$ are factors of $T^{p^d} - T$. Now write, $T^{p^d} - T = T.(T^{\frac{p^d-1}{2}} + 1).(T^{\frac{p^d-1}{2}} - 1)$, see that these factors are pairwise co-prime. Which implies that $A_{id} = gcd(A_{id}, T).gcd(A_{id}, T^{\frac{p^d-1}{2}} + 1).gcd(A_{id}, T^{\frac{p^d-1}{2}} - 1)$. □

For $p = 2$:

Proposition 4.6. *Take polynomial $U(x) = X + X^2 + X^4 + ... + X^{2^d-1}$. If $p = 2$ and A_{id} is as above, the for any $T(x) \in \mathbb{F}_2[X]$ we have,*

$$A_{id} = gcd(A_{id}, U(T)).gcd(A_{id}, U(T) + 1)$$

Proof. In $\mathbb{F}_2[X]$ we have, $U(T)^2 = T^2 + T^4 + ... + T^{2^d}$. Then inside $\mathbb{F}_2[X]$,

$$\begin{aligned} U(T).(U(T)+1) &= (T^2 + T^4 + ... + T^{2^d}) + (T + T^2 + ... + T^{2^d-1}) \\ &= T + 2(T^2 + ... + T^{2^d-1}) + T^{2^d} \\ &= T^{2^d} + T \\ &= T^{2^d} - T \quad (as +1 \equiv -1\ (mod\ 2)) \end{aligned}$$

And from proof of *prop* 4.5, we know that all elements of $\mathbb{F}_{2^d}[X]$(*i.e.* roots of A_{id}) are roots of $T^{2^d} - T$. Hence, $A_{id} = gcd(A_{id}, U(T)).gcd(A_{id}, U(T) + 1)$. □

The steps for equal degree factorization, also known as Cantor and Zassenhaus split, are as follows:
EDF(A_{id}): (For $p > 2$) Returns $deg(A_{id})/d$ irreducible factors of A_{id}.

1. If $deg(A_{id}) = d$, result is A_{id}, else,
2. Choose $T \in \mathbb{F}_p$ randomly.
3. Set $B(x) \leftarrow gcd(A_{id}, T^{\frac{p^d-1}{2}} - 1)$, if $deg(B) = 0$ *or* $deg(A_{id})$, repeat *steps* 2 *and* 3.
4. Repeat *steps* 1 *to* 3 for $B(x)$ and $A(x)/B(x)$ recursively.

EDF(A_{id}): (For $p = 2$) Returns $deg(A_{id})/d$ irreducible factors of A_{id}.

1. If $deg(A_{id}) = d$, result is A_{id}, otherwise, set $T \leftarrow X$.
2. set $U \leftarrow T$ and then set $U \leftarrow T + U^2 \ (mod\ A)$, $(d-1)$ times.
3. set $B(x) \leftarrow gcd(A, U)$
4. if $deg(B) = 0$ *or* $deg(A_{id})$, set $T \leftarrow T.X^2$ and goto *step* 2.
5. Repeat *step* 1 *to* 4 for $B(x)$ and $A(x)/B(x)$ recursively.

4.2 Polynomial Factorization over $\mathbb{Z}$ or $\mathbb{Q}$

Factorization of a primitive polynomial in $\mathbb{Z}[X]$ is equivalent to factoring in $\mathbb{Q}[X]$. Let $A \in \mathbb{Z}[X]$ be a primitive polynomial with irreducible factorization $A = A_1 A_2 .... A_r$ with $A_i \in \mathbb{Q}[X]\ \forall\ i = 1, 2, ..., r$. Let l_i be the LCM of denominators of the coefficients of irreducible polynomial A_i, then $l_i A_i \in \mathbb{Z}[X]$. And, let $c_i = cont(l_i A_i)$. Write each A_i as primitive $\tilde{A}_i = \frac{l_i}{c_i} A_i \in \mathbb{Z}[X]$. Hence, we have

$$A = a\tilde{A}_1\tilde{A}_2.....\tilde{A}_r\ ; \quad where,\ a \in \mathbb{Q}$$

But both A and A_i are primitive, so $a = 1$. We have :

$$A = \tilde{A}_1\tilde{A}_2.....\tilde{A}_r\ ; \quad \tilde{A}_i \in \mathbb{Z}[X]$$

Conversely, let $A(x) \in \mathbb{Q}[X]$ and d be the LCM of denominators of coefficients of A. Then $dA(x) \in \mathbb{Z}[X]$ and let $c = cont(dA(x))$. Hence, we have $\frac{d}{c}A(x)$ primitive and in $\mathbb{Z}[X]$.

So, we only need to factor $A(x) \in \mathbb{Z}[X]$ such that $A = A_1^{e_1} A_2^{e_2} ..... A_r^{e_r}$ with $A_i(x) \in \mathbb{Z}[X]$. Now we reduce our problem further to factoring a square-free polynomial in $\mathbb{Z}[X]$. Notice that,

$$gcd(A, A') = \prod_{i=1}^{r} A_i^{e_i - 1}$$

Hence, we are now are left with following square-free polynomial to factor,

$$B(x) = \frac{A}{gcd(A, A')} = \prod_{i=1}^{r} A_i$$

Definition 4.7 (s-norm). *We define $s - norm$ ($s > 0$) for a polynomial $A(x) = \sum_{i=1}^{n} a_i x^i$, $A(x) \in \mathbb{C}[x]$, $a_i \in \mathbb{C}$ as*

$$\|A\|_s = \left(\sum_{i=1}^{n} |a_i|^s \right)^{1/s}$$

Where $|a_i|$ denotes the absolute value of a_i. We call $\|A\|_\infty = max\{|a_0|, |a_1|, ..., |a_n|\}$ the ***max-norm*** *of $A(x)$.*

For example, We write $1 - norm$ and $2 - norm$ as :

$$\|A\|_1 = |a_0| + |a_1| + ... + |a_n|$$
$$\|A\|_2 = \left(|a_0|^2 + |a_1|^2 + ... + |a_n|^2\right)^{1/2}$$

Remark 4.8. *See that, $\|A\|_\infty \leq \|A\|_2 \leq \|A\|_1 \leq (n+1)\,\|A\|_\infty$.*

Before we proceed with a factorization algorithm we will prove the following bound for factors of a polynomial in $\mathbb{Z}[X]$ by first proving few *lemmas*.

Theorem 4.9 (Mignotte's Bound)**.** *Let $A(x), B(x), C(x) \in \mathbb{Z}[X]$ satisfying $A(x) = B(x)C(x)$ with $deg(A) = n$, $deg(B) = m$ $deg(C) = k$ then :*

$$\|B\|_1\,\|C\|_1 \leq 2^{m+k}\,\|A\|_2 \leq (n+1)^{1/2}2^n\,\|A\|_\infty$$

Lemma 4.10. *For $A(x) \in \mathbb{C}[X]$ and $z \in \mathbb{C}$, we have*

$$\|(x-z)A(x)\|_2 = \|(\bar{z}x - 1)A(x)\|_2$$

Proof. Let $A = \sum_{i=0}^{n} a_i x^i \in \mathbb{C}[X]$ and set $a_{-1} = a_{n+1} = 0$ and denote complex conjugate of $z, a_i \in \mathbb{C}$ as $\bar{z}, \overline{a_i}$. Then we have,

$$\begin{aligned}
\|(x-z)A(x)\|_2^2 = \|xA(x) - zA(x)\| &= \sum_{i=0}^{n+1} |a_{i-1} - za_i|^2 \\
&= \sum_{i=0}^{n+1} (a_{i-1} - za_i)(\overline{a_{i-1}} - \overline{za_i}) \\
&= \sum_{i=0}^{n+1} |a_{i-1}|^2 + |z|^2|a_i|^2 - (z\overline{a_{i-1}}a_i + \bar{z}a_{i-1}\overline{a_i}) \\
&= \sum_{i=0}^{n+1} |a_i|^2 + |z|^2|a_{i-1}|^2 - (z\overline{a_{i-1}}a_i + \bar{z}a_{i-1}\overline{a_i}) \\
&= \sum_{i=0}^{n+1} (\bar{z}a_{i-1} - a_i)(z\overline{a_{i-1}} - \overline{a_i}) \\
&= \sum_{i=0}^{n+1} |\bar{z}a_{i-1} - a_i|^2 \\
&= \|(\bar{z}x - 1)A(x)\|_2^2
\end{aligned}$$

$\square$

Notation: Let $A(x) = \sum_{i=0}^{n} a_i x^i = a_n \prod_{i=1}^{n} (x - z_i)$, $A(x) \in \mathbb{C}[X]$ and $a_i, z_i \in \mathbb{C}$. Where $z_i' s$ are the roots of $A(x)$. Then we will use $M(A) = |a_n| . \prod_{i=1}^{n} max\{1, |z_i|\}$. See that if $A(x) = B(x)C(x)$ then $M(A) = M(B)M(C)$.

Lemma 4.11 (Landau's Inequality)**.** *For any polynomial $A(x) \in \mathbb{C}[X]$ we have $M(A) \leq \|A(x)\|_2$.*

Proof. Arrange the roots such that $|z_1|, |z_2|, \ldots.., |z_k| > 1$ and $|z_{k+1}|, |z_{k+2}|, \ldots.., |z_n| < 1$ for some $k \in \{1, 2, \ldots., n\}$. Then $M(A) = |a_n . z_1 z_2 \ldots.. z_k|$. Let

$$B(x) = a_n \prod_{i=1}^{k} (\overline{z_i} x - 1) \prod_{i=k+1}^{n} (x - z_i) = b_n x^n + b_{n-1} x^{n-1} + \ldots.. + b_0 \in \mathbb{C}[X]$$

Then,

$$M(A)^2 = |a_n . \overline{z_1 z_2} \ldots.. \overline{z_k}|^2 = |b_n|^2 \leq \|B\|_2^2$$

Now applying *lemma* 4.10 repeatedly we get:

$$\begin{aligned}
\|B\|_2^2 = \left\| \frac{B}{(\overline{z_1} x - 1)} (x - z_1) \right\| &= \left\| \frac{B}{(\overline{z_1} x - 1) . \overline{z_2} x - 2)} (x - z_1)(x - z_2) \right\| = \ldots\ldots.. \\
&= \left\| \frac{B}{\prod_{i=1}^{k} (\overline{z_i} x - 1)} \prod_{i=1}^{k} (x - z_i) \right\| \\
&= \left\| \frac{a_n \prod_{i=1}^{k} (\overline{z_i} x - 1) \prod_{i=k+1}^{n} (x - z_i)}{\prod_{i=1}^{k} (\overline{z_i} x - i)} \prod_{i=1}^{k} (x - z_i) \right\| \\
&= \left\| a_n \prod_{i=1}^{n} (x - z_i) \right\| \\
&= \|A\|_2^2
\end{aligned}$$

Hence, $M(A) \leq \|A\|_2$.

$\square$

Lemma 4.12. *Let $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{m} b_i x^i$, $A(x), B(x) \in \mathbb{C}[X]$ such that $B(x)$ divides $A(x)$($m \leq n$). Then we have :*

$$\|B\|_2 \leq \|B\|_1 \leq 2^m M(B) \leq \left| \frac{b_m}{a_n} \right| 2^m \|A\|_2$$

Proof. First, write $B(x) = b_m \prod_{i=1}^{m}(x-\alpha_i)$, $\alpha_i \in \mathbb{C}$. Expressing the coefficients of $B(x)$ in terms of roots:

$$b_i = (-1)^{m-i} b_m \sum_{\substack{S\subseteq\{1,..,m\}\\|S|=m-i}} \prod_{j\in S} \alpha_j$$

So,

$$\begin{aligned} |b_i| = |b_m| \left|\sum_S \prod_{j\in S} \alpha_j\right| &\leq |b_m| \sum_S \prod_{j\in S} |\alpha_j| \\ &\leq \binom{m}{i} M(B) \end{aligned} \tag{4.1}$$

Hence, we have

$$\|B\|_1 = \sum_{i=0}^{m} |b_i| \leq M(B) \sum_{i=0}^{m} \binom{m}{i} = 2^m M(B)$$

As the roots of $B(x)$ are also the roots of $A(x)$, let $\alpha_1, ...\alpha_m$ be the roots of $B(x)$ and $\alpha_{m+1}, ..., \alpha_n$ be the remaining roots of $A(x)$ (roots are counted with multiplicity). Then, we have

$$\begin{aligned} M(A) &= \left|\frac{a_n}{b_m}\right| M(B). \prod_{i=m+1}^{n} max\{1, |\alpha_i|\} \\ &\geq \left|\frac{a_n}{b_m}\right| M(B) \end{aligned}$$

Therefore we have :

$$M(B) \leq \left|\frac{b_m}{a_n}\right| M(A) \tag{4.2}$$

After combining the above results we get,

$$\begin{aligned} \|B\|_1 \leq 2^m M(B) &\leq 2^m \left|\frac{b_m}{a_n}\right| M(A) \\ &\leq \left|\frac{b_m}{a_n}\right| 2^m \|A\|_2 \quad (By\ Lemma\ 4.11) \end{aligned}$$

□

Now we are all set to prove the *Mignotte's Bound*.

***Theorem* 4.9.** Using *Lemma* 4.12 we get:

$$\begin{aligned} \|B\|_1 \|C\|_1 &\leq 2^{m+k} M(B) M(C) = 2^{m+k} M(A) \\ &\leq 2^{m+k} \|A\|_2 \quad (By\ Lemma\ 4.11) \\ &\leq 2^n (n+1)^{1/2} \|A\|_\infty \end{aligned}$$

□

Now we will prove a lemma that will be required later when factoring polynomial using LLL-algorithm.

Lemma 4.13 (Landau-Mignotte). *If $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{m} b_i x^i$, $A(x), B(x) \in \mathbb{Z}[X]$ such that $B(x) \mid A(x)$, then we have:*

$$\|B\|_2 \leq \binom{2m}{m}^{1/2} \|A\|_2 \tag{4.3}$$

Proof. From (4.1) we have $|b_i| \leq \binom{m}{i} M(B)$. From (4.2) we have $M(B) \leq \frac{|b_m|}{|a_n|} M(A)$, and as $B \mid A$ we have $\frac{|b_m|}{|a_n|} \leq 1$. Thus, above inequality becomes

$$|b_i| \leq \binom{m}{i} M(A) \leq \binom{m}{i} \|A\|_2 \quad (By\ [4.11])$$

Therefore, we have $\|B\|_2 = \left(\sum_{i=0}^{m} |b_i|^2\right)^{1/2} \leq \left(\sum_{i=0}^{m} \binom{m}{i}^2\right)^{1/2} \|A\|_2 = \binom{2m}{m}^{1/2} \|A\|_2$. □

Now, at first we will factor polynomials modulo p^k for large enough k and then compute factors corresponding to factorization in $\mathbb{Z}[X]$. For factorization modulo p^k we will lift our factorization from polynomial ring $(\mathbb{Z}/p\mathbb{Z})[X]$ to ring $(\mathbb{Z}/p^k\mathbb{Z})[X]$ using **Hensel Lifting** given below.

4.2.1 Hensel Lifting

I. First we will discuss the following theorem which provides us with a way to lift factorization of some polynomial from modulo p, for some prime p, to modulo p^k, $k \geq 1$. However, it is done by lifting factorization to only one power of p at a time. Later, we will give an efficient method of lifting the factorization.
The steps for the algorithms can be followed easily from the proof for all the methods given below.

Theorem 4.14. *Let p be any prime and let polynomials C, A_e, B_e, U, V in $\mathbb{Z}[X]$ s.t.*

$$C = A_e B_e \ (mod\ p^e)\ e \geq 1$$
$$UA_e + VB_e \equiv 1 (mod\ p) \ \ with,$$
$$deg(U) < deg(B_e) \ and\ deg(V) < deg(A_e)$$

Assume A_e monic then $\exists\ A_{e+1}, B_{e+1} \in \mathbb{Z}[X]$, unique modulo p^{e+1}, satisfying the above conditions for $e+1$, s.t. $A_e \equiv A_{e+1}\ (mod\ p^e)$ and $B_e \equiv B_{e+1}\ (mod\ p^e)$

Proof. Let $D = (C - A_e b_e)/p^e$ and let $A_{e+1} = A_e + p^e S$ and $B_{e+1} = B_e + p^e T$ with $S, T \in \mathbb{Z}[X]$. Now we prove the following claim which gives us the possible values of S and T: *Claim:* $S \equiv VD + WA_e \ (mod\ p)$ and $T \equiv UD + WB_e \ (mod\ p)$ for some $W \in \mathbb{Z}[X]$. *Proof:* We need to show that $C = A_{e+1}B_{e+1} \ (mod\ p^{e+1})$. Further putting the values:

$$\begin{aligned} p^e D + A_e B_e &\equiv A_{e+1}B_{e+1} \ (mod\ p^{e+1}) \\ &\equiv A_e B_e + p^e(A_e T + B_e S) \ (mod\ p^{e+1}) \end{aligned}$$

Hence,

$$\begin{aligned} & p^e D \equiv p^e(A_e T + B_e S) \ (mod\ p^{e+1}) \\ \implies & D \equiv A_e T + B_e S \ (mod\ p) \end{aligned}$$

Satisfying above equation with S and T values from our claim would prove our claim and hence the theorem as S, T being unique modulo p implies that A_{e+1} and B_{e+1} are unique modulo p^{e+1}. We have

$$\begin{aligned} A_e T + B_e S &\equiv A_e(UD + WB_e) + B_e(VD + WA_e) \ (mod\ p) \\ &\equiv A_e UD + B_e VD \ (mod\ p) \\ &\equiv D(A_e U + B_e V) \ (mod\ p) \\ &\equiv D \ (mod\ p) \end{aligned}$$

as required. □

II. Given integers p and q, need not be prime, we will look at following theorem that helps us in lifting the factorization from modulo q to modulo qr where $r = gcd(p, q)$.

Theorem 4.15. *Assume that*

$$C(x) = A(x)B(x) \ (mod\ q), \ UA + VB = 1 \ (mod\ p)$$

where $deg(U) < deg(B), deg(V) < deg(A), deg(C) = deg(A) + deg(B)$, *for* $C, A, B, U, V \in \mathbb{Z}[X]$. *Let* $gcd(l(A), r) = 1$, *where* $l(A)$ *indicates the leading coefficient of* $A(x)$. *Then there exists some polynomials* $A_1, B_1 \in \mathbb{Z}[X]$ *congruent to* A, B *modulo* q *with* $l(A_1) = l(A), deg(A_1) = deg(A), deg(B_1) = deg(B)$ *such that*

$$C(x) = A_1(x)B_1(x) \ (mod\ qr)$$

Further, if r *is prime, the result is unique modulo* qr.

Proof. Our problem is to find some $S(x), T(x) \in \mathbb{Z}[X]$ such that $A_1 = A + qS, B_1 = B + qT$ with $deg(S) < deg(A), deg(T) \leq deg(B)$, degree conditions follow from the condition that we want $deg(A_1) = deg(A)$ and $deg(B_1) = deg(B)$. We want S, T to also satisfy

$$\begin{aligned}
A_1B_1 = (A+qS)(B+qT) &\equiv C \ (mod\ qr),\ which\ implies,\\
q(AT+BS) + q^2(ST) &\equiv qP \ (mod\ qr)\\
(As\ C - AB = qP \ (mod\ qr)\ &for\ some\ P(x) \in \mathbb{Z}[X])\\
AT + BS + q(ST) &\equiv P \ (mod\ r) \ \ (as\ gcd(qr, q) = q)\\
AT + BS &\equiv P \ (mod\ r)
\end{aligned} \tag{4.4}$$

Claim: We have $T = UP + RB$ and $S = VP - RA$ for all $R(x) \in \mathbb{Z}[X]$ satisfying equation (4.4).
Proof: Putting values of $T(x)$ and $S(x)$ in LHS of last condition of equation (4.4) we get

$$\begin{aligned}
AT + BS &= A(UP + RB) + B(VP - RA)\\
&= P(UA + VB)\\
&\equiv P \ (mod\ r) \ \ (as\ UA + VB \equiv 1 \ (mod\ p) and\ r \mid p)
\end{aligned} \tag{4.5}$$

As $l(A)$ has an inverse modulo r, we can get a quotient $R(x)$, satisfying above conditions, such that $deg(VP - RA) < deg(A)$ by polynomial division with remainder. We can set $R(x)$ as quotient when VP is divided by $A(x)$. For this choice of $R(x)$ we have $deg(UP + RB) \leq deg(B)$ as $deg(AT + BS) \leq deg(C)$ by definition of T, S, then rest follows from equation (4.5).
uniqueness: solution obtained in claim for S and T value is unique when r is prime. For this see that if S_1 and T_1 is some other solution then we must have $AT + BS \equiv AT_1 + BS_1 \ (mod\ r)$ which implies $A(T - T_1) \equiv B(S_1 - S) \ (mod\ r)$. So, we must have $A \mid (S_1 - S)$ but $deg(A) > deg(S_1 - S)$. Therefore, solution is unique. □

III Now we will look at another method which lifts factorization from modulo p to p^2. Let $A \in \mathbb{Z}[X]$ and p be a prime number $s.t.\ p \nmid l(A)$ (leading coefficient of polynomial A). And, we have $B_1, C_1 \in \mathbb{Z}[X]\ s.t.\ A = B_1C_1 \ (mod\ p)$ with $gcd(\overline{B_1}, \overline{C_1}) = 1 \implies \exists\, U, V \in \mathbb{Z}[X]\ \ s.t. \overline{UB_1} + \overline{VC_1} = 1$ ($P \ (mod\ p)$ is denoted as $\overline{P}$ for polynomial P). We have $A = B_1C_1 + pV_1$ for some $V_1 \in \mathbb{Z}[X]$. Let $A - B_1C_1 = pV_1 = R$ and $UB_1 + VC_1 = 1 + pV_2$ for some $V_2 \in \mathbb{Z}[X]$.
Claim: Set $B_2 = B_1 + VR$ and $C_2 = C_1 + UR$ in $\mathbb{Z}[X]$. Then we have $A \equiv B_2C_2 \ (mod\ p^2)$.

Proof:

$$\begin{aligned} B_2C_2 &= (B_1 + VR)(C_1 + UR) \\ &= B_1C_1 + R(UB_1 + VC_1) + VUR^2 \\ &= A - (A - B_1C_1) + (A - B_1C_1)(1 + pV_2) + VUR^2 \\ &= A + (pV_1)(pV_2) + p^2VUV_1^2 \\ &\equiv A \ (mod\ p^2) \end{aligned}$$

We can use this method to list from any p^n to p^{2n}. Using any or combination of multiple factors based of our possible exponents can make out task efficient. See that in second algorithm taking pair as (p^n, p^n) lifts our factorization to p^{2n}.

4.2.2 Factorization in $\mathbb{Z}[X]$

If $A(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{Z}[X]$ and $B(x) = \sum_{j=0}^{m} b_j x^j \in \mathbb{Z}[X]$ are two polynomials then the matrix representing the linear map:

$$\psi_{A,B} : V_m \times V_n \longrightarrow V_{m+n}\ ;\ \psi_{A,B}(F,G) = FA + GB \tag{4.6}$$

is called the **Sylvester Matrix**, $\mathcal{S}(A,B)$, where V_m and V_n are m and n dimensional vector spaces over $\mathbb{Q}$ with basis $\{1, x, ..., x^{m-1}\}$ and $\{1, x, ..., x^{n-1}\}$ respectively.

Definition 4.16. *The **resultant**, $res(A,B)$, of polynomials $A, B \in \mathbb{Z}[X]$ is the integer determinant of $\mathcal{S}(A,B)$. And we define **discriminant** of any polynomial $A \in \mathbb{Z}[X]$ to be $res(A, A')$, where A' denotes the formal derivative of $A(x)$.*

Refer to [9, page 122] for algorithm to compute resultant of two polynomials.

Lemma 4.17. *Let $A(x), B(x) \in \mathbb{Z}[X]$ then $gcd(A,B) = 1$ if and only if $det(\mathcal{S}(A,B)) \neq 0$.*

Proof. Let $G(x) = gcd(A,B) \neq 1$. Then for some non-zero $S = B/G \in \mathbb{Z}[X]$ and $T = -A/G \in \mathbb{Z}[X]$ we have $SA + TB = 0 \implies \psi_{A,B}$, as in (4.6), is not an injective map. Thus, $det(\mathcal{S}(A,B)) = 0$.
Conversely, assume that there exists non-zero $(S,T) \in V_m \times V_n$ such that $\psi_{A,B}(S,T) = 0$. Then we have $SA + TB = 0$. First, let $S \neq 0$, then as $gcd(A,B) = 1$ and $TB = -SA$ we must have B a divisor of S, which is a contradiction as $deg(S) < deg(B)$. Similarly, we can't have $t \neq 0$. Therefore, $gcd(A,B) = 1 \iff det(\mathcal{S}(A,B)) = 0$. □

We only have to factor square-free and primitive polynomial $A(x) \in \mathbb{Z}[X]$ with $deg(A) = n$. Set a bound $s = (n+1)^{1/2}2^n \|A\|_\infty |l(A)|$, where $l(A)$ denotes the leading coefficient of $A(x)$. First we have to choose a prime p such that $A \ (mod\ p) \in (\mathbb{Z}/p\mathbb{Z})[X]$ is also square-free. Such a p would have to satisfy $gcd(\overline{A}, \overline{A}') = 1$ and $p \nmid l(A)$. From 4.17 we can see that $gcd(\overline{A}, \overline{A}') = 1 \iff disc(\overline{A}) \neq 0$. Now, take $k \in \mathbb{Z}$ such that $p^k > 2s$ and using Hensel lifting raise factorization to ring $(\mathbb{Z}/p^k\mathbb{Z})[X]$ $s.t.$

$$A(x) = l(A).A_1.A_2....A_r \ (mod\ p^k), \quad A_i \in \mathbb{Z}[X], l(A_i) = 1$$

Where we will use symmetric representation for coefficients of polynomials inside $(\mathbb{Z}/p^k\mathbb{Z})[X]$. Let $S \subseteq \{1, 2, ..., r\}$ and S_c be its complement. Take $B, C \in \mathbb{Z}[X]$ such that:

$$B \equiv l(A) \prod_{i \in S} A_i \ (mod\ p^k) \ , \ \ C \equiv l(A) \prod_{i \in S_c} A_i \ (mod\ p^k)$$

See that $\|B\|_\infty, \|C\|_\infty < \frac{p^k}{2}$. Now we will prove the following claim to complete our factorization.
Claim : $\|B\|_1 \|C\|_1 \leq s \iff l(A)A = BC$.

Proof. First, let $l(A)A = BC$. Applying *Theorem* 4.9 on polynomial $l(A)A$ we get

$$\|B\|_1 \|C\|_1 \leq |l(A)| \|A\|_\infty (n+1)^{1/2}2^n = s$$

Conversely, let $\|B\|_1 \|C\|_1 \leq s$. Then we have, by definition of B and C, $l(A)A \equiv BC \ (mod\ p^k)$. As we choose $p^k > 2s$, we have

$$\|BC\|_\infty \leq \|BC\|_1 \leq \|B\|_1 \|C\|_1 \leq s < \frac{p^k}{2}$$

Which means coefficients of BC belong to interval $(-p^k/2, p^k/2)$. So we have $l(A)A = BC$. □

The steps for Zassenhaus factorization algorithm are as follows: Let $A(x) \in \mathbb{Z}[X]$ be a square-free and primitive polynomial with degree n.

1. set $s \leftarrow (n+1)^{1/2}2^n \|A\|_\infty |l(A)|$
2. find p such that $p \nmid l(A)$ and $disc(\overline{A}) \neq 0$
3. set $k \leftarrow \lceil log_p(2s+1) \rceil$ (as $p^k > 2s$)
4. Factor $A \ (mod\ p)$ into irreducible factors as shown in *section* 4.1.

5. Lift factorizaion of $A(x)$ from modulo p to modulo p^k as $A(x) = l(A)A_1A_2...A_r \ (mod\ p^k)$ using Hensel lifting shown in *section* 4.2.1. (Note that $\|A_i\|_\infty < p^k/2$)

6. set $T \leftarrow \{1, 2, ..., r\}$

7. For $m = 1, 2, ..., r$, for all $m - element$ subsets of T, say S.
   (i) set $B \leftarrow l(A) \prod_{i \in S} A_i \ (mod\ p^k)$
   (ii) set $C \leftarrow l(A) \prod_{i \in S_c} A_i \ (mod\ p^k)$
   (iii) if $\|B\|_1 \|C\|_1 \leq s$
   output $B/cont(B)$ as a factor
   Set $T \leftarrow T - S$, $A \leftarrow C/cont(C)$

8. output A as final remaining factor

4.2.3 Polynomial Factorization in $\mathbb{Z}[X]$ Using LLL-Algorithm

Take $A(x) \in \mathbb{Z}[X]$, primitive and square-free polynomial with $deg(A) = n > 0$. First we will factor $A(x)$ into irreducible polynomials in $\mathbb{F}_p[X]$ and select a monic polynomial $B(x) \in \mathbb{Z}[X]$ such that $B(x)\ (mod\ p)$ is irreducible in $\mathbb{F}_p[X]$ with conditions:

$$\begin{aligned} &B\ (mod\ p^k)\ divides\ A\ (mod\ p^k)\ in\ (\mathbb{Z}/p^k\mathbb{Z})[X] \\ &B\ (mod\ p)^2\ does\ not\ divide\ A\ (mod\ p)\ in\ \mathbb{F}_p[X] \end{aligned} \tag{4.7}$$

See that $B\ (mod\ p) \mid A\ (mod\ p)$ as from (4.7) we get $B\ (mod\ p^k) \mid A\ (mod\ p^k)$. Hence, we have $(B\ mod\ p^k)\ mod\ p \mid (A\ mod\ p^k)\ mod\ p$ which gives us $B\ (mod\ p) \mid A\ (mod\ p)$. We can get such $B(x)$ by lifting factorization of $A(x)\ (mod\ p)$ from $\mathbf{F}_p[X]$ to $(\mathbb{Z}/p^k\mathbb{Z}[X])$. Now, let $deg(B(x)) = m_1, 0 < m_1 \leq n$.

Proposition 4.18. *There exists an irreducible factor $P(x) \in \mathbb{Z}[X]$ of $A(x)$ such that $B\ (mod\ p) \mid P\ (mod\ p)$, and $P(x)$ is unique upto sign. Further, if $Q(x) \mid A(x)$ in $\mathbb{Z}[X]$ then the following are equivalent:*

1. *$B\ (mod\ p) \mid Q\ (mod\ p)$ in $\mathbb{F}_p[X]$,*

2. *$B\ (mod\ p^k) \mid Q\ (mod\ p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$,*

3. *$P(x) \mid Q(x)$ in $\mathbb{Z}[X]$.*

Proof. Existence follows from $B\ (mod\ p)$ being irreducible in $\mathbb{F}_p[X]$ and equation (4.7). If $P(x) = B(x)$ then irreducibility follows from definition of $B(x)$. And, Uniqueness

follows from equation (4.7). Now we look as equivalences:
$(\mathbf{2}) \rightarrow (\mathbf{1})$: As $B\ (mod\ p^k) \mid Q\ (mod\ p^k)$ we have $(B\ mod\ p^k)\ mod\ p \mid (Q\ mod\ p^k)\ mod\ p \implies B\ (mod\ p) \mid Q\ (mod\ p)$.
$(\mathbf{3}) \rightarrow (\mathbf{1})$: As $P \mid Q$ we have $P\ (mod\ p) \mid Q\ (mod\ p)$. Thus, $B\ (mod\ p) \mid P\ (mod\ p) \implies B\ (mod\ p) \mid Q\ (mod\ p)$.
$(\mathbf{1}) \rightarrow (\mathbf{3})$: By (1) and equation (4.7) we get that $B\ (mod\ p) \nmid A/Q\ (mod\ p)$ in $\mathbb{F}_p[X]$. Thus, $P \nmid (A/Q)$ in $\mathbb{Z}[X] \implies P \mid Q$.
$(\mathbf{1}) \rightarrow (\mathbf{2})$: By definition of $B(x)$ and previous discussion we have $gcd(B\ (mod\ p), A/Q\ (mod\ p)) = 1$, hence, there exist some $\lambda_1, \mu_1 \in \mathbb{Z}[X]$ such that $\lambda_1\ (mod\ p).B\ (mod\ p) + \mu_1\ (mod\ p).A/Q\ (mod\ p) = 1$. Therefore, we have

$$\lambda_1.B + \mu_1.A/Q = 1 - pv_1\ for\ some\ v_1 \in \mathbb{Z}[X] \tag{4.8}$$

On multiplying (4.8) with $Q(1 + pv_1 + ... + p^{k-1}v_1^{k-1})$ we get:

$$\lambda_2.B + \mu_2.A = (1 - pv_1).Q(1 + .. + p^{k-1}v_1^{k-1})$$
$$\lambda_2.B + \mu_2.A = (1 - p^k v_1^k)Q, \ \ \lambda_2, \mu_2 \in \mathbb{Z}[X]$$
$$\lambda_2.B\ (mod\ p^k) + \mu_2.A\ (mod\ p^k) = Q\ (mod\ p^k)$$

As $B\ (mod\ p^k) \mid A\ (mod\ p^k)$, because $V \in M$ and $B\ (mod\ p^k) \mid b\ (mod\ p^k)$,we get LHS of above equation divisible by $B\ (mod\ p^k)$. Hence, $B\ (mod\ p^k) \mid Q\ (mod\ p^k)$.
See that, putting $Q = P$ gives us $B\ (mod\ p^k) \mid P\ (mod\ p^k)$ □

Now, fix an integer $m \geq deg(B) = m_1$. Take $\mathcal{L}$ to be the set of all degree $\leq m$ polynomials $C(x)$ such that $B\ (mod\ p^k) \mid C(x)\ (mod\ p^k)$, which also implies $B\ (mod\ p) \mid C(x)\ (mod\ p)$. See that the basis for $\mathcal{L}$ is given by the coefficients of following polynomials

$$\{p^k X^i : 0 \leq i < deg(B) = m_1\} \cup \{BX^j : 0 \leq m - deg(B)\} \tag{4.9}$$

Identify a polynomial $\sum_{i=0}^{m} a_i x^i$ as $(a_0, a_1, ..., a_m)$ in vector form. See that $B \mid BX^j$ and $p^k X^i \equiv 0\ (mod\ p)$. And that these are linearly independent elements which are total $m_1 + m - m_1 + 1 = m + 1$ in number. Thus, $\mathcal{L}$ with above basis is a lattice in $\mathcal{R}^{m+1}$. Also, notice that the basis vectors forms an upper triangular matrix with p^k as first m_1 diagonal entries and 1 as rest as $B(x)$ is monic, therefore $det(\mathcal{L}) = p^{k.deg(B)} = p^{km_1}$.

Proposition 4.19. *If some $b \in \mathcal{L}$ satisfies :*

$$p^{km_1} > \|A\|_2^m \|b\|_2^n \tag{4.10}$$

Then $P \mid b$ in $\mathbb{Z}[X]$, where $P(x), A(x)$ are as in 4.18, and $gcd(A, b) \neq 1$.

Proof. Assume $b \neq 0$ and $G = gcd(A, b)$. If we show that $B\ (mod\ p) \mid G\ (mod\ p)$, then by 4.18 $(1) \to (3)$, we have $P \mid G$ and hence $P \mid b$. We prove this by contradiction. Assume that $B\ (mod\ p) \nmid G\ (mod\ p)$, then for some $S_1, T_1, V_1 \in \mathbb{Z}[X]$ we have

$$S_1B + T_1G = 1 - pV_1 \tag{4.11}$$

Now define

$$M = \{SA + Tb : S, T \in \mathbb{Z}[X], deg(S) < deg(b) - deg(G)\ and\ deg(T) < deg(A) - deg(G)\}$$

And, call projection of M on $\mathbb{Z}X^{deg(G)} + \mathbb{Z}X^{deg(G)+1} + ... + \mathbb{Z}X^{deg(A)+deg(b)-deg(G)-1}$ be M_1.

Claim : Kernel of above projection is trivial, thus image has same rank as M.

Proof: Let $SA + Tb \in M$ goes to 0 in M_1, then $deg(SA + Tb) < deg(G)$. But as $G \mid A$ and $G \mid b$ we have $G \mid SA + Tb \implies SA + Tb = 0$.

Further, as $SA + Tb = 0 \implies SA/G + Tb/G = 0$. As $gcd(A/G, b/G) = 1$, we have $(A/G) \mid T$ but this can't happen as $deg(T) < deg(A/G)$, by definition. Thus, $T = 0$, and similarly, $S = 0$. Therefore we have the projections of $\{X^iA : 0 \le i < deg(b) - deg(G)\} \cup \{X^jb : 0 \le j < deg(A) - deg(G)\}$ on M_1 linearly independent and spanning M_1. Hence, M_1 is a lattice of rank $deg(A) + deg(b) - 2deg(G)$. And from *Hadamard's inequality* and condition given in proposition we have

$$det(M_1) \le \|A\|_2^{deg(b)-deg(G)} \|b\|_2^{deg(A)-deg(G)} \le \|A\|_2^{m} \|b\|_2^{deg(A)} < p^{km_1} \tag{4.12}$$

Claim : $\{V \in M : deg(V) < deg(G) + deg(B)\} \subset p^k\mathbf{Z}[X]$

Proof: Let V be as in claim then we have $G \mid V$. On multiplying (4.11) by $V/G(1 + pV_1 + ... + p^{k-1}V_1^{k-1})$ we get

$$S_2B + T_2V \equiv V/G\ (mod\ p^k),\ S_2, T_2 \in \mathbb{Z}[X] \tag{4.13}$$

As $B\ (mod\ p^k) \mid V\ (mod\ p^k)$ we have from (4.13), $B\ (mod\ p^k) \mid (V/G)\ (mod\ p^k)$ but $deg(B\ (mod\ p^k)) = deg(B)$ as it is monic whereas $deg((V/G)\ (mod\ p^k)) < deg(G) + deg(B) - deg(G) = deg(B)$, thus $V/G \equiv 0\ (mod\ p^k) \implies V \equiv 0\ (mod\ p^k) \implies claim.$

Now choose a basis $\{b_{deg(G)}, ..., b_{deg(A)+deg(b)-deg(G)-1}\}$ of M_1 such that b_i has degree i. Notice that the matrix from above basis will be an upper triangular matrix, hence determinant can be calculated by multiplying the leading coefficients of each basis vector. From previous claim we get that the leading coefficients of b_i are divisible by p^k. Thus, $deg(M_1) \ge p^{km_1}$, but on comparing with (4.12) we get a contradiction. □

Proposition 4.20. *If $\{x_1, x_2, ..., x_{m+1}\}$ is a reduced basis of lattice $\mathcal{L}$, as in (4.9), with*

$$p^{km_1} > 2^{mn/2}\binom{2m}{m}^{n/2} \|A\|_2^{m+n} \tag{4.14}$$

Then $deg(P(x)) \leq m \iff \|x_1\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}$. Where $P(x), A(x), p, k$ are as in [4.18], $deg(A(x)) = n, deg(B(x)) = m_1$.

Proof. If $\|x_1\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}$ then by [4.19] we have $P \mid x_1 \implies deg(P) \leq m$ as $deg(x_1) \leq m$. Now, assume $deg(x_1) \leq m$ then $x_1 \in L$. And, on applying (4.3), we have $\|x_1\|_2 \leq \binom{2m}{m}^{1/2}\|A\|_2$. By *Hadamard's inequality* we have $\|x_1\|_2 \leq 2^{m/2}\|P\|_2 \leq 2^{m/2}\binom{2m}{m}^{1/2}\|A\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}$ by (4.14). □

Proposition 4.21. *Let the notations be same as in [4.20], along with condition (4.14). And let $t \in \{1, 2, ..., m+1\}$ be the largest such value for which we have $\|x_t\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}$, then $deg(P(x)) = m - t + 1$ with $P(x) = gcd(x_1, ..., x_t)$ and $\|x_i\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}\ \forall\, 1 \leq i \leq t$.*

Proof. Let set $\mathcal{I}$ be set of all indices such that $i \in \mathcal{I}\ iff\ \|x_i\|_2 < (p^{km_1}/\|A\|_2^m)^{1/n}$. By [4.19], $P \mid x_i\ \forall\, i \in \mathcal{I}$ and so we have $P \mid P_1$ where $P_1 = gcd(\{x_i\}_{i\in\mathcal{I}})$. As $P_1 \mid x_i\ \forall\, i \in \mathcal{I}$ with $deg(x_i) \leq m$, so, $x_i \subset \mathbb{Z}P_1 + \mathbb{Z}P_1X + ... + \mathbb{Z}P_1X^{m-deg(P_1)}$. And as $x_i's$ are linearly independent they are atmost $m + 1 - deg(P_1)$ in number for $i \in \mathcal{I}$, so $|\mathcal{I}| \leq m + 1 - deg(P_1)$. From (4.3) result, $\|PX^j\|_2 = \|P\|_2 \leq \binom{2m}{m}^{1/2}\|A\|_2\ \forall j \geq 0$, and we have $PX^j \in \mathcal{L}$ for $0 \leq j \leq m - deg(P)$, so we now have

$$\|x_i\|_2 \leq 2^{m/2}\binom{2m}{m}^{1/2}\|A\|_2,\ \ 1 \leq i \leq m - deg(P) + 1$$

And by (4.14), we have $i \in \mathcal{I}\ \forall\, i \in \{1, 2, ..., m - deg(P) + 1\}$. From this and as number of elements in $\mathcal{I}$ are $\leq m - deg(P_1) + 1$ but $P \mid P_1 \implies deg(P) = deg(P_1) = m + 1 - t$, $\mathcal{I} = \{1, 2, ..., t\}$. Now, if we prove that P_1 is primitive than as $P \mid P_1$ and are of same degree we must have $P = P_1$. Choose some $i \in \mathcal{I}$ and c_i the content for corresponding x_i. Then as P is primitive, we must have $P \mid \frac{x_i}{c_i} \implies \frac{x_i}{c_i} \in \mathcal{L}$ as $P \in \mathcal{L}$. But x_i is a basis of lattice $\mathcal{L}$, so $c_i = 1$. So, x_i is primitive and $P_1 \mid x_i$ so P_1 must be primitive also, this completes the proof of proposition. □

Algorithm Steps: For given square-free and primitive $A(x) \in \mathbb{Z}[X]$ the steps for algorithm are as follows:

1. Choose smallest p such that $p \nmid l(A)$ and $disc(\overline{A}) \neq 0$

2. Factor $A(x)$ $(mod\ p)$ in $\mathbb{F}_p[X]$ (as shown in *section* 4.1) and store irreducible factors, $B(x)\ mod\ p$, in set $\mathcal{B}$.

3. Choose an arbitrary $B(x)\ mod\ p \in \mathcal{B}$, set $d = deg(B(x)\ mod\ p)$.

4. if $d = deg(A)$, output $A(x)$ and terminate, otherwise, go to *step* 5

5. Choose smallest k such that $p^{kd} > 2^{(n-1)n}\binom{2(n-1)}{n-1}^{n/2} \|A\|_2^{2n-1}$

6. set $m \leftarrow \lfloor (n-1)/2^u \rfloor$ such that $u = max\{l \mid d \leq (n-1)/2^l\}$.

7. Lift factorization $A(x) = B(x)B_1(x)$ $(mod\ p)$ from modulo p to modulo p^k, set $Q(x) \leftarrow C$ $(mod\ p^k)$ such that $C \equiv B$ $(mod\ p)$. (by Hensel lifting).

8. By $LLL - Algorithm$ get reduced basis $\{x_1, x_2, ..., x_{m+1}\}$ of lattice $\mathcal{L}$ given in (4.9).

9. if $\|x_1\| \geq (p^{kd}/\|A\|_2^m)^{1/n}$, then set $u \leftarrow u-1$ and $m \leftarrow \lfloor (n-1)/2^u \rfloor$, and goto *step* 7, otherwise, go to *step* 10.

10. set $t \leftarrow max\{j \mid j \in \{1, 2, ..., m+1\}\ s.t.\ \|x_j\| < (p^{kd}/\|A\|_2^m)^{1/n}\}$.

11. output irreducible factor $P(x) = gcd(x_1, x_2, ..., x_t)$.

12. if $A(x) = 1$, Terminate. Otherwise, remove all such $B\ mod\ p$ from $\mathcal{B}$ such that B $(mod\ p) \mid P$ $(mod\ p)$, set $A(x) \leftarrow A(x)/P(x)$ and go to *step* 3.

In *Step* 1 we are choosing a prime p such that primitive, squarefree polynomial $A(x) \in \mathbb{Z}[X]$ is square-free modulo p also. Which means $disc(\overline{A}) \neq 0$ by [4.17]. Hence, we will choose a smallest such p. In *Step* 5 we are choosing minimum such k satisfying (4.14) for worst case $m \geq deg(B(x))$, which will be $n-1$ $(n = deg(A(x)))$. *Step* 6 runs for m values less than or equal to $n-1$ for reducing the runtime of algorithm. Sooner or later will get required factor as in *step* 11. *Step* 9 can be seen directly as a result of 4.20. *Step* 10 follows from 4.21. As by 4.18, the obtained factor $P(x)$ in *step* 11 is unique upto sign for some chosen $B\ mod\ p$ from set $\mathcal{B}$, we will remove all such factors from $\mathcal{B}$ as done in *step* 12 and after removing $P(x)$ from $A(x)$ we will go through *step* 3 *to* 12 until $A(x)$ is completely factored.

Note 4.22. *If $A(x) \in \mathbb{Z}[X]$ is not a square free polynomial then we let $D(x) = gcd(A(x), A'(x))$. We will factor $A_0(x) = A(x)/D(x)$ using above algorithm. See that each irreducible factor of $D(x)$ in $\mathbb{Z}[X]$ divides $A_0(x)$. We will take each irreducible factor from $A_0(x)$ and by trial division get its exponent in $D(x)$. Thus, we will have complete factorization of $A(x)$.*

APPENDIX A

ALGORITHMS

Algorithms

A.1 Primality Testing Algorithms

A.1.1 Fermat's Little Test

Algorithm 3 Fermat's Test

```
Data: An integer $N \geq 2$.
Result: Returns whether $N$ is probable prime or definite composite.
set $x \leftarrow$ random integer $\in [2, N-1]$
if $x^p \not\equiv x \ (mod\ p)$ then
    output Definite Composite
else
    output Probable Prime
end
```

A.1.2 Miller-Rabin Test

Algorithm 4 Miller-Rabin Test

```
Data: An odd integer $N \geq 2$.
Result: Returns whether $N$ definite composite or undecided.
factor $N-1$ as $2^k m, k \geq 1$
set $x \leftarrow$ random integer $\in [2, N-2]$
if $x^m \ (mod\ N) \neq 1$ then
    if $x^{2^j m} \ (mod\ N) \ \neq N-1 \ \forall\, j \in \{0, ..., k-1\}$ then
        output Definite Composite
    else
        output Undecided
    end
else
    output Undecided
end
```

A.1.3 Lucas Test

Algorithm 5 Lucas Test

```
Data: An integer $N > 2$, Bound $\mathcal{B}$ for base values.
Result: Returns whether $N$ is definite prime or undecided.
set $x \leftarrow 2$
while $x < \mathcal{B}$
  if $x^{N-1}$ $(mod\ N) = 1$ and $x^{(N-1)/p}$ $(mod\ N) \neq 1\ \forall$ primes $p \mid (N-1)$ then
    output Definite Prime
    Terminate
else
    $x = x + 1$
end
output Undecided
```

A.1.4 Solovay Strassen Test

Algorithm 6 Solovay-Strassen Test

```
Data: An odd integer $N > 2$.
Result: Returns whether $N$ is definite composite or probable prime.
set $x \leftarrow$ random integer $\in [2, N-1]$
calculate $\left(\frac{x}{N}\right)$ by steps given in subsection 1.1.3 .
if $x^{N-1/2}.\left(\frac{x}{N}\right)$ $(mod\ N) \neq 1$ then
    output Definite Composite
    Terminate
else
    output Probable prime
end
```

A.1.5 The AKS-Algorithm

Algorithm 7 AKS-Algorithm

```
Data: An integer $N \geq 2$.
Result: Returns whether $N$ is prime or composite.
if $N = a^b$ for $a \in \mathbb{N}$ and $b > 1$; output $Composite$
Find smallest $r$ $s.t.$ $order_r(N) > 4\lceil log\ N \rceil^2$
if $1 < gcd(a, N) < N$ for some $a \leq r$; output $Composite$
if $N \leq r$; output $Prime$
for $1 \leq a \leq 2\lceil r \rceil \lceil log\ N \rceil$; Do :
    if $(X + a)^N \neq X^N + a\ (mod\ X^r - 1, N)$; output $Composite$
Return $Prime$
```

A.2 Factorization Algorithms

A.2.1 Fermat's Factorization Scheme(FFS)

Algorithm 8 Non-Generalized FFS

```
Data: An odd positive integer $n > 2$.
Result: Returns factors, not necessarily prime, of $n$.
set $k \leftarrow \lfloor \sqrt{n} \rfloor$
if $k < \sqrt{n}$; set $k \leftarrow k + 1$
set $y \leftarrow \lfloor k^2 - n \rfloor$
while $y^2 \, != (k^2 - n)$
    set $k \leftarrow k + 1$
    set $y \leftarrow \lfloor k^2 - n \rfloor$
Return $(k + y),\ (k - y)$
```

Algorithm 9 Generalized FFS

```
Data: An odd positive integer $n > 2$.
Result: Returns factors, not necessarily prime, of $n$.
set $k \leftarrow \lfloor \sqrt{n} \rfloor$
if $k < \sqrt{n}$; set $k \leftarrow k + 1$
set $y \leftarrow \lfloor k^2 - n \rfloor$
set $r = 2$
while $y^2 \mathrel{!}= (k^2 - n)$
  if $k^2 > rn$ then
    set $y \leftarrow \lfloor k^2 - rn \rfloor$
    set $k \leftarrow k + 1$
  else
    set $k \leftarrow k + 1$
    set $y \leftarrow \lfloor k^2 - n \rfloor$
Return $gcd(k + y, n),\ gcd(k - y, n)$
```

A.2.2 Pollard's Algorithms

Algorithm 10 Pollard-rho with Brent's Modification

```
Data: An odd integer $N > 2$, Bound $\mathcal{B}$ for sequence calculations.
Result: Returns divisor, not necessarily prime, of $N$, if found.
set $j \leftarrow 1$
while $j < \mathcal{B}$ do
  if $j$ is of form $2^r - 1$ then
    Calculate all $k$ s.t. $2^r \leq k < 2^{r+1}$
    if if $gcd(x_k - x_{2^r-1}, N)$ is non-trivial for any $k$ then
      output $divisor = gcd(x_k - x_{2^r-1}, N)$
      Terminate
    else
      $j = j + 1$
    end
else
  $j = j + 1$
end
```

Algorithm 11 Pollard $(p-1)$ Algorithm

```
Data: An odd integer $N > 2$, Bound $\mathcal{B}$ for $lcm$ calculation.
Result: Returns divisor, not necessarily prime, of $N$, if found.
set $x \leftarrow$ random integer from $[2, n-2]$
set $L \leftarrow lcm(1, 2, ..., \mathcal{B})$
set $M \leftarrow x^L \ (mod\ N)$
if $gcd(M-1, N)$ is non-trivial then
    output divisor=$gcd(M-1, N)$
else
    output Choose another $x$ or $\mathcal{B}$ value
end
```

A.2.3 Continued fraction factoring algorithm

Algorithm 12 CFFA

```
Data: An positive integer $N > 2$, A bound $\mathcal{B}$ for calculations.
Result: Returns factors (not necessarily prime) of $N$, if found.
set $\mathcal{X}_0 \leftarrow \sqrt{N}$
set $b_0 \leftarrow \lfloor \sqrt{N} \rfloor$
if $b_0 = \mathcal{X}_0$ then
    output Divisors = $\mathcal{X}_0$,$\mathcal{X}_0$ Terminate
else
    calculate $b_j, \mathcal{X}_i$ using (1.3)
    calculate $\mathcal{P}_k,\ \mathcal{Q}_k$ using (1.5)
    calculate $\mathcal{S}_k,\ \mathcal{T}_k$ using (1.6),(1.7)
    set $k \leftarrow 1$
    while $k < \mathcal{B}$
     if $\mathcal{T}_k$ is a perfect square for even $k$ then
        if $\mathcal{P}_{k-1} \not\equiv \pm\sqrt{\mathcal{T}_k}\ (mod\ N)$ then
            output Divisors = $gcd(\mathcal{P}_{k-1} + \sqrt{\mathcal{T}_k},\ N)$ and $gcd(\mathcal{P}_{k-1} - \sqrt{\mathcal{T}_k},\ N)$ Terminate
        else
            $k = k + 1$
        end
    else
        $k = k + 1$
    end
end
```

A.2.4 Quadratic Sieve Algorithm

Algorithm 13 QSA

Data: An integer $N \geq 2$, Bound $\mathcal{B}$ for primes in factor base and $\mathcal{M}$ for sieve interval.

Result: Returns divisors, not necessarily prime, of N, if found.

```
set $\mathcal{F} = \{-1, 2\}$
for $primes\ p \leq B$ do
  if$\left(\frac{N}{p}\right) = 1$; add $p$ to $\mathcal{F}$
set $A(x) = x^2 - N$
set $\mathcal{X} = \{\ \lfloor\sqrt{N}\rfloor - \mathcal{M}, ...., \lfloor\sqrt{N}\rfloor + \mathcal{M}\},\ \mathcal{A} = \{\ \}$
set $M_{|\mathcal{A}|\times|\mathcal{F}|}$ as empty matrix, $N_{|\mathcal{A}|\times|\mathcal{A}|}$ as identity matrix
for $x \in \mathcal{X}$ do
  add $A(x)$ to $\mathcal{A}$ if it is $\mathcal{F}$-smooth
for $A(x_i) \in \mathcal{A}$ do
  factor $A(x_i)$ as $(-1)^{e_{i1}} p_1^{e_{i2}} p_2^{e_{i3}} .... p_{|\mathcal{F}|}^{e_{i|\mathcal{F}|}}$; $1 \leq i \leq |\mathcal{A}|$
  set $[e_{i1}\ (mod\ 2), e_{i2}\ (mod\ 2), ....., e_{i|\mathcal{F}|}\ (mod\ 2)]$ as $i^{th}$ row of $M$
for column $j = \{1, 2, ..., |\mathcal{F}|\}$ of $M$
  if $r_i$ is the first row with 1 at $j^{th}$ place
    set $r_j \leftarrow r_j + r_i\ (mod\ 2)\ \forall\ i < j < |\mathcal{A}|$ and then remove $r_i$ from $M$
    do previous step with similar rows of $N$
    if some $i^{th}$ row, of $M$, $r_i = [0, 0, ..., 0]$
      set $y^2 = \mathcal{A}\ .\ r_{iN}$ (dot product), $r_{iN}$ is $i^{th}$ row of $N$
      set $x = \prod_{l=1}^{|\mathcal{A}|} x_l$ where $x_l$ are s.t. $A(x_l) \in \mathcal{A}$
      if $gcd(y - x, N)$ is $non - trivial$
        output divisors = $gcd(y - x, N)\ gcd(y + x, N)$
        Terminate
output try different values of $\mathcal{B}$ and $\mathcal{M}$
```

A.2.5 Other Algorithms

Algorithm 14 Binary search for checking perfect square

Data: An integer $N > 1$.

Result: Returns "True" if N is a perfect square, otherwise, returns "False"

```
set $left \leftarrow 1$
set $right \leftarrow N$
while $left \leq right,$ do
  set $mid \leftarrow (left + right)/2$
  if $(mid)^2 = N$; output $True$; Terminate
  if $(mid)^2 < N$; set $left \leftarrow mid + 1$
  else; set $right \leftarrow mid - 1$
output $False$
```

APPENDIX B

SOME PYTHON IMPLEMENTATIONS

Some Python Implementations

B.1 Primality Testing Algorithms

B.1.1 Fermat's Little Test

Input: A number n that is to be tested and an integer k for the number of times Fermat's test runs before it outputs 'Probable Prime'.
Output: Returns $'Definite\ Composite'$ if n is composite else, returns $'Probable\ Prime'$.

[45]:
```
import random
def fermat(n,k):
    for i in range(k):
        a = random.randint(2,n)
        if pow(a,n,n)!= pow(a,1,n):
            return "Definite Composite"
    return "Probable Prime"
```

[47]:
```
fermat(2323547635437657325245732 57,5)
```

[47]:
```
'Definite Composite'
```

[48]:
```
fermat(84893278521644348909450211 12361,15)
```

[48]:
```
'Probable Prime'
```

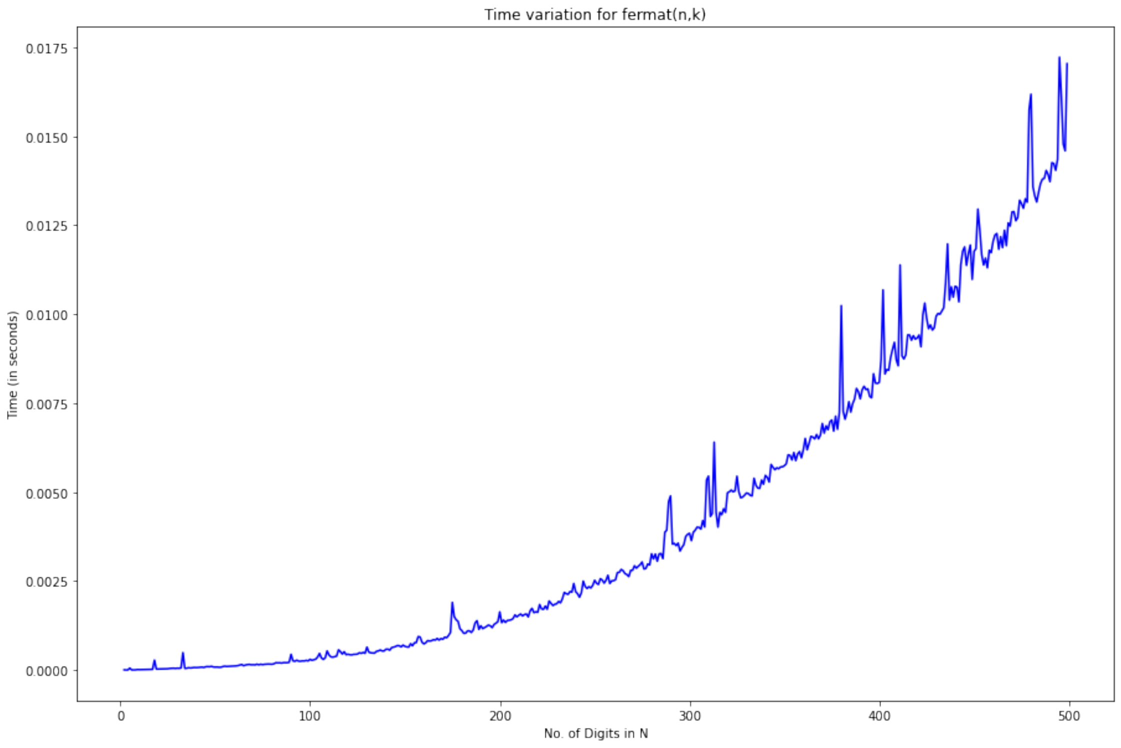
Time analysis

Here we show how time of the execution of above program varies. We check running time of the program for a random *n-digit* number where $3 \leq n \leq B$. B is given as an input to following program.

[26]:
```
import matplotlib.pyplot as plt
import time
def analysis(B):
    x = []
    y = []

    for n in range(2,B):
        start = 10**(n)
        end = 10**(n+1)
        x.append(n)
        L = random.randint(start,end)
        strt = time.perf_counter()
        ans =  fermat(L,10)
        end = time.perf_counter()
        y.append(end-strt)
    plt.figure(figsize=(15,10))
    plt.plot(x,y, color = 'b')
    plt.title("Time variation for fermat(n,k)")
    plt.xlabel('No. of Digits in N')
    plt.ylabel('Time (in seconds)')
    plt.show()
```

```
[27]: analysis(500)
```

Note B.1. *We do not provide functions used for analysis as above for further tests as those will be some close variations of above program.*



B.2 Miller-Rabin Test for Compositeness

Input: An odd number N, being tested for compositeness, and integer k such that test runs k times before returning *Undecided*.
Output: Returns *Composite* if n is composite, else, returns *Undecided*.

[55]:
```
import random
def rabin_miller(N,k): # N : Any odd integer
        m=(N-1)/2
        h=1
        while m%2==0:
            m=int(m/2)
            h = h+1
        m = int(m)
        for i in range(k):
            x = random.randint(2,N-1)
            if pow(x,m,N)!=1:
                l=0
                for j in range(h):
                    if pow(x,pow(2,j)*m,N)!=N-1:
                        l=l+1
                if l==h:
                    return "Composite"
        return "Undecided"
```

[27]:
```
rabin_miller(100123456789,5)
```

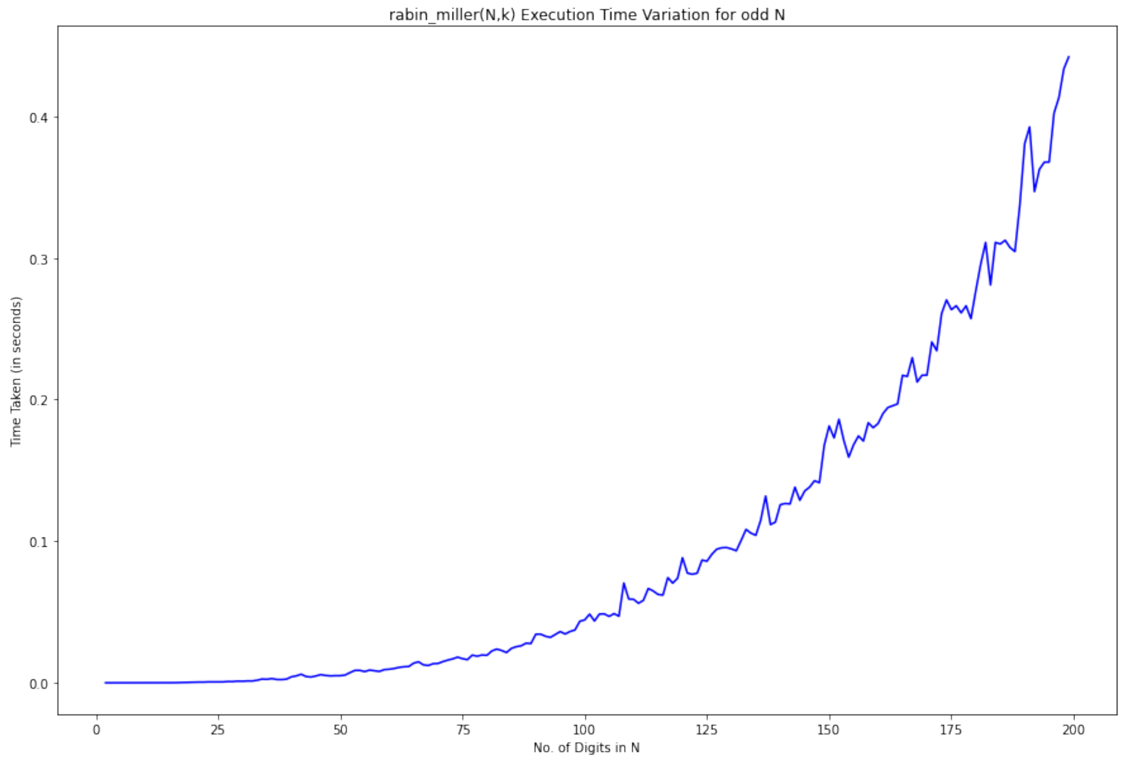
[27]:
```
'Undecided'
```

[30]:
```
rabin_miller(2323547635437657325245732 51,15)
```

[30]:
```
'Composite'
```

Time analysis

Below graph shows the variation of running time by above algorithm along with increase in number of digits of random input number. $y-axis$ denotes the execution time of program in seconds and $x-axis$ denotes the number of digits in input N. k was taken to be 10 for each input.



B.2.1 Solovay-Strassen Test

Input: An odd number $N \geq 3$ and $x \in \mathbb{N}$ for Jacobi symbol calculation.

Output: Returns Jacobi symbol $\left(\frac{x}{N}\right)$

Jacobi Symbol

[1]:
```
def jacobi(N,x):
        b=x%N
        c=N
        s=1
        while b>=2:
            while b%4==0:
                b = b/4
            if b%2 == 0:
                if c%8==3 or c%8==5:
                    s = -s
                b=b/2
            if b==1:
                break
            if b%4==c%4==3:
                s=-s
            b1=b
            b=c%b
            c=b1
        return s
```

Solovay-Strassen(SS) Test

Input: An odd number N to test for compositeness.
Output: Returns *Composite* if n is composite, else, returns *Probable Prime*.

[2]:
```
import math as m
import random as r
def sstest(number):
        base = r.randint(2,number-1)
        print("Base=",base)
        gcd = m.gcd(base,number)
        if gcd!=1:
            print("Composite")
        else:
            euler = pow(base,int((number-1)/2),number)
            jacob = jacobi(number,base)
            if euler != 1:
                if euler!= number-1:
                    print("Composite")
                else:
                    if jacob != -1:
                        print("Composite")
                    else:
                        print("Probable Prime")
            else:
                if jacob!=1:
```

```
                print("Composite")
            else:
                print("Probable Prime")
```

[3]:
```
sstest(20991129234731)
```

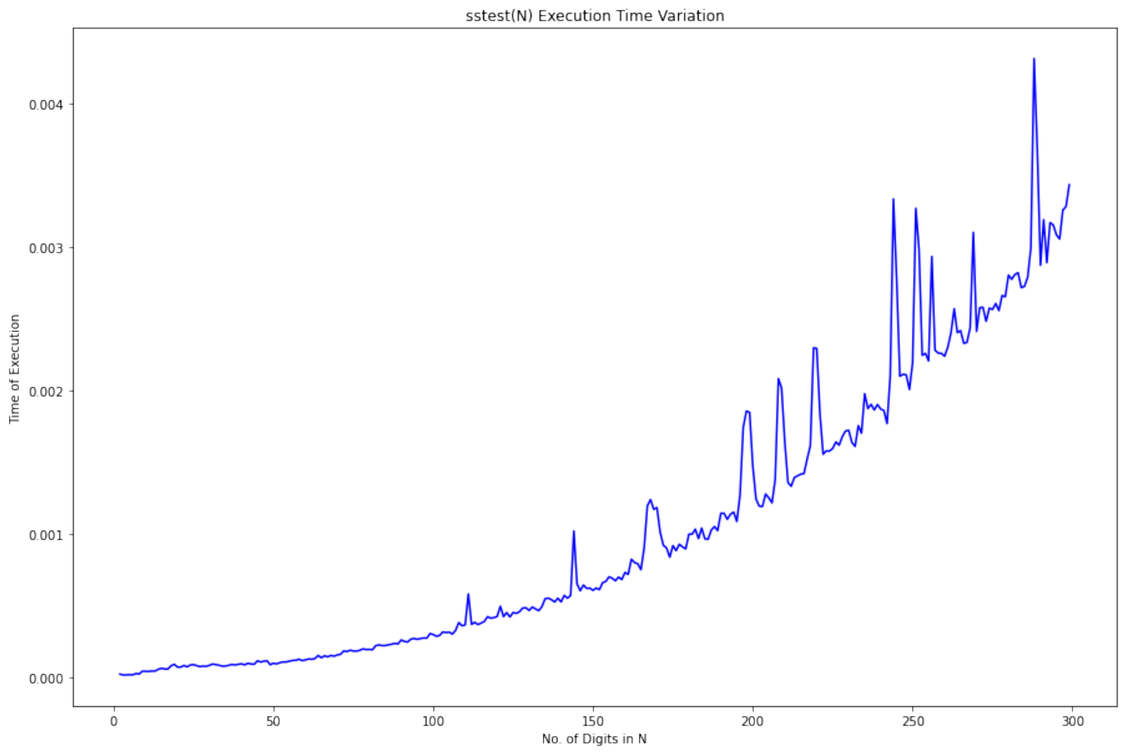
```
Base= 4471304183471
Probable Prime
```

[4]:
```
sstest(9487897987986068049)
```

```
Base= 1018217905385428701
Composite
```

Time analysis

Variation of time taken by above algorithm along with increase in number of digits of input number. A random integer is selected for each digit value and time is calculated.



B.2.2 Lucas Test

Input: An integer $N > 2$ and k for number of Lucas test on N.
Output: Returns $'prime'$ if suitable base is found in k-trials, otherwise, returns $'undecided'$.

```
[17]: from sympy import primefactors as pf
      import random
      def lucas(N,k): # N>2
          factors = pf(N-1)
          m = len(factors)
          for i in range(k):
              if m != 0:
                  a = random.randint(1,N)
                  if pow(a,N-1,N) == 1:                          # Condition 1
                      for factor in factors:
                          if pow(a,int((N-1)/factor),N) == 1: # Condition 2
                              break
                          else:
                              m = m-1
                  if m == 0:
                      return 'prime'
          return 'undecided'
```

```
[18]: lucas(244446464646677,10)
```

```
[18]: 'prime'
```

```
[19]: lucas(855937339524071340192676159497,10)
```

```
[19]: 'undecided'
```

Running Time

There are many factors that effect the running time of above algorithm. When a composite number does not satisfy first condition, it immediately goes out of the loop in search for another base a. If composites satisfy condition 1 (called, pseudoprimes with respect to that base) then they have to go through condition 2 for all prime factors of

$N-1$. First we see running time for some composites of different below degrees then analyse running time for prime number in next section.

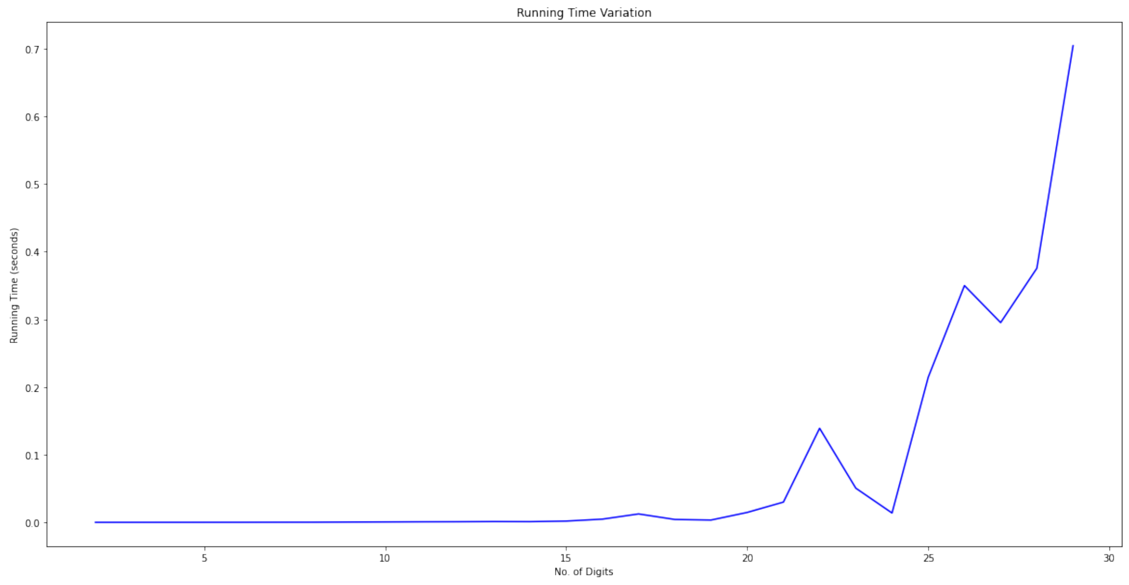
B.2.2.0.0.1 Some non-prime inputs:

[54]:
```
import time

# Digits : 10,20,30,40,50,60,70,100
Non_primes =␣
 ↪[7227163127,15054625465523761479,654243355136412450156070938749,
              84864089872746563832232915164613072532741,
              363056972346495169984662724961225802171007333049369,
              9267733525776697034821258986418149535338114226853703138834442,

            ␣
 ↪77317252851078874734257181329266724732094714837071451541449947282457165,
49852395997866172543327279057917989098170500901559037815859389465695332670054916982897264999507263037]
for number in Non_primes:
    s = time.perf_counter()
    res = lucas(number,10)
    e = time.perf_counter()
    print(e-s,res)
```

```
0.0008512999993399717 undecided
0.007540399999925285 undecided
0.0941777999967024 undecided
1.5742758999986108 undecided
1.4906166000000667 undecided
20.290683299999728 undecided
3.5245419000002585 undecided
10.123875099998259 undecided
```

B.2.2.0.0.2 Prime inputs: Graph below indicates the running time variation with respect to number of digits of input numbers in "$lucas(N,10)$", where N is taken to be prime to reduce irregularities in running time. A running time for a particular digit d is considered to be the average running time of randomly chosen 50 primes from 50 disjoint intervals, of equal length, from set $(10^{d-1},10^{d}]$. Number of digits in graph, on $x-axis$, vary from 3 to 30.

Some primes also take comparatively larger running time through Lucas test if the base for a prime number satisfying both the conditions is not encountered in k-runs.



B.3 Factorization Algorithms

B.3.1 Fermat's Factorization Scheme(FFS)

Input: An integer $n \geq 2$.
Output: Returns factors of n, not necessary prime.

Non-Generalized FFS

```
[8]: import math as m
     def fermat1(n):
         factors = []

         if n%2 == 0:
             while n%2 == 0:
                 n = n/2
             factors.append(2)
         n_root = m.sqrt(n)
         k = m.ceil(n_root)
```

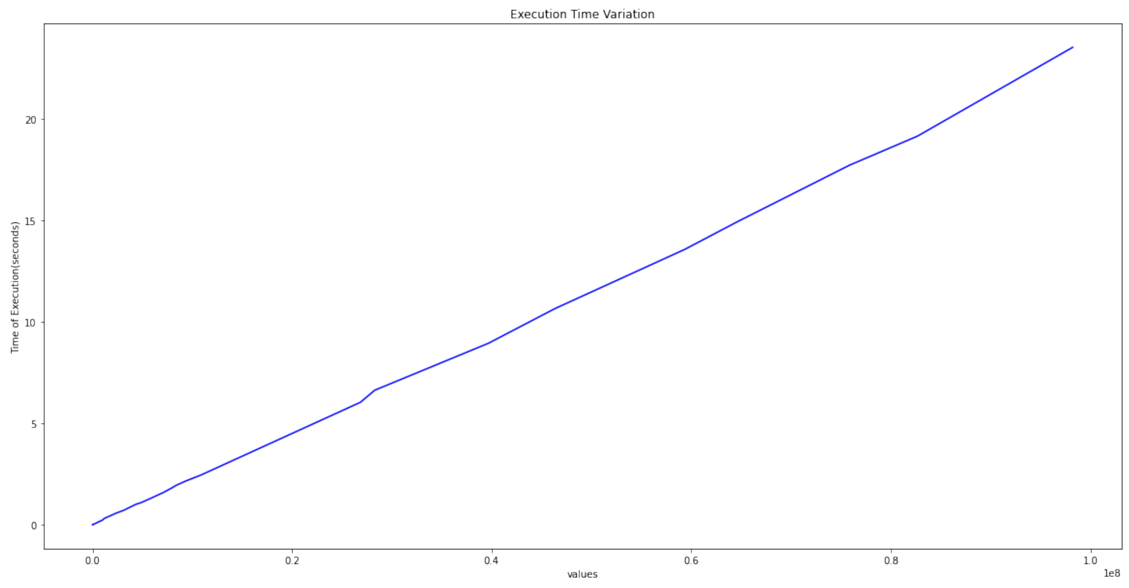
```
    if k == n_root:
        factors.append([n_root,n_root])
    else:
        y2 = int(k*k-n)
        y = m.isqrt(y2)
        while y*y!=y2:
            k=k+1
            y2=int(k*k-n)
            y=m.isqrt(y2)
        factors.append([k+y,k-y])
    return factors
```

[12]:
```
import time
strt = time.perf_counter()
print(fermat1(1234567895341))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
strt = time.perf_counter()
print(fermat1(3478283691346587))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
```

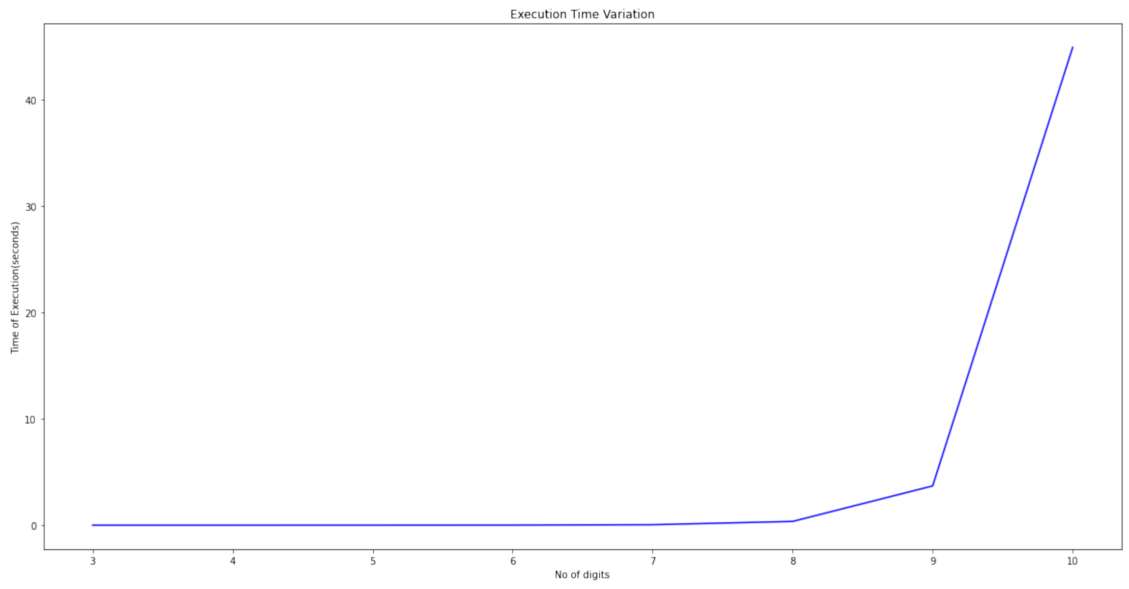
```
[[9924259, 124399]]
Time taken = 1.8144788999998127
[[149343049, 23290563]]
Time taken = 12.809771000000183
```

Time analysis

Fermat's factorization scheme, "$fermat1(n)$" searches for a perfect square, where if n is prime then the trivial factorization is reached. So, for primes the running time directly depends on the size of n.
The following plot indicates the running time for prime values ranging from 3 to 8 digits. Particular d-digit values are taken from equal length disjoint intervals of sets $(10^{d-1}, 10^{d}]$. 10 values(1 value per interval) per digit are taken and their graph is plotted with their running time(in seconds) on $y - axis$.



Now to see change in running time for odd non-prime numbers we plot a graph of average running time of 50 values, chosen as above, of a particular digit with respect to their digits. Digits range from 3 to 10.



Generalised FFS

Input: An integer $N \geq 2$.
Output: Returns factors of n, not necessary prime.

[44]:
```
import math as m
def fermat2(n):
    factors = []
    if n%2 == 0:
        while n%2 == 0:
            n = n/2
        factors.append(2)
    n = int(n)
    root_n = m.sqrt(n)
    x = m.ceil(root_n)
    if x == root_n:
        factors.append([x,x])
    else:
        x2 = x*x
        y2 = x2-n
        root_y2 = m.sqrt(y2)
        y = m.ceil(root_y2)
        k=2
        while y*y!=y2:
            if x2>k*n:
                y2 = x2-k*n
                root_y2 = m.sqrt(y2)
                y = m.ceil(root_y2)
                k=k+1
            else:
                x= x+1
                x2 = x*x
                y2 = x2-n
                root_y2 = m.sqrt(y2)
                y = m.ceil(root_y2)
        factors.append([m.gcd(int(x-y),n),m.gcd(int(x+y),n)])
    return factors
```

```
[36]: import time
strt = time.perf_counter()
print(fermat2(1234567895341))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
strt = time.perf_counter()
print(fermat2(3478283691346587))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
```

```
[[124399, 9924259]]
Time taken = 3.5098799999977928
[[23290563, 149343049]]
Time taken = 23.459397400001762
```

Time taken by non-generalised FFS for above factors was 1.8144788999998127 and 12.809771000000183 seconds respectively. Generalised factoring method comes in handy in theory while searching for a perfect square as we need to make $k^2 - rn$ a perfect square for some $r \geq 1$ and $k, k \geq \lceil\sqrt{n}\rceil$, instead of $k^2 - n$ as done in case of non-generalised FFS.

B.3.2 Pollard's $(p-1)$ algorithm

Input: An odd integer $n > 2$. A q value, large enough $s.t.$ for some prime divisor p we have $p - 1 \mid lcm(1, 2..., q)$.
Output: Returns a divisor of n, not necessary prime. Returns "Failure" if divisor is not found for some base or if "q" value is not large enough.

```
[2]: import math
import random as r
def pollard1(n,q): # Enter odd n
    list1 = [i for i in range(1,q+1)]
    lcm = 1
    for i in list1:
        lcm = lcm*i//math.gcd(lcm, i)
    base = r.randint(2,n-1)
    m=pow(base,lcm,n)
    gcd=math.gcd(m-1,n)
    if gcd != n and gcd != 1:
        return 'Factor=',gcd
    else:
        return 'Failure'
```

```
[7]: # Works better if factors are small
import time
st = time.perf_counter()
print(pollard1(1234567895341,100))
end = time.perf_counter()
print(end-st)
st = time.perf_counter()
print(pollard1(3478283691346587,100))
end = time.perf_counter()
print(end-st)
```

```
('Factor=', 473)
0.00041449999997666964
('Factor=', 43)
0.0001553000000171778
```

B.3.3 Pollard's Rho Algorithm

Input: An integer $n \geq 2$. Initial value $x1$ for sequence. $a \in \mathbb{Z}$ for polynomial $f(x) = x^2 + a$, $a \neq 0, -2$ and a bound for total number of sequence calculations.
Output: Returns a divisor of n, not necessary prime, along with the period of sequence. Returns "None" if divisor is not found for input entries.

```
[115]: # f(x) = x^2 + a
import math
def pollard2(n,x1,a,bound):      # a!= 0 and -2, x1 = initial value of␣
 ↪sequence {x_i}
    list1=[x1]
    for i in range(1,bound):
        x1=(pow(x1,2)+a)%n
        list1.append(x1)
    for i in range(1,int(bound/2)):
            n1=list1[2*i]-list1[i]
            gc=math.gcd(n1,n)
            if n1%n!=0 and gc>1:
                print("Period(w.r.t. Divisor)=",i)
                return "Divisor=",gc
                break
```

[120]:
```
import time
st = time.perf_counter()
print(pollard2(3478283691346587,1,1,100))
end = time.perf_counter()
print(end-st)
```

```
Period(w.r.t. Divisor)= 1
('Divisor=', 3)
0.0005506999987119343
```

[116]:
```
# Inputs are semiprimes

print("Pollard p-1 Method")
st = time.perf_counter()
pollard1(100004653457,500) #factoring a aemi-prime
end = time.perf_counter()
print(end-st)
st = time.perf_counter()
pollard1(1000000000099987889,6000) #factoring a aemi-prime
end = time.perf_counter()
print(end-st)

print("Pollard rho Method")
st = time.perf_counter()
pollard2(100004653457,1,1,200) #factoring a aemi-prime
end = time.perf_counter()
print(end-st)
st = time.perf_counter()
```

```
pollard2(1000000000099987889,1,1,9500) #factoring a aemi-prime
end = time.perf_counter()
print(end-st)
```

```
Pollard p-1 Method
0.0005899999996472616
0.020680599998740945
Pollard rho Method
Period(w.r.t. Divisor)= 78
0.0002509000005375128
Period(w.r.t. Divisor)= 4622
0.00977720000082627
```

Note B.2. *No separate graph analysis was done as parameters given as input in both algorithms vary with input numbers.*

B.3.4 Continued fraction factoring algorithm

Input: An integer $n > 2$. Bound B for total number of recursive calculations before termination.
Output: Returns two divisors of n, not necessary prime.

[130]:
```
import math as m
import sympy as sp
def cffa(n,b):               # b is bound for calculations
    xo=sp.sqrt(n)
    ao=m.floor(xo)
    if xo==ao:
        return xo,xo
    else:
        x1=1/(xo-ao)
        a1=m.floor(x1)
        po=ao
        p1=a1*po+1
        qo=1
        q1=a1
        k=1
        while k<b:
            k=k+1
            p=p1
            q=q1
            x1=1/(x1-a1)
            a1=m.floor(x1)
            p1=a1*p1+po
            q1=a1*q1+qo
            k1=(p1*p1)%n
            sq=sp.sqrt(k1)
            sq1=m.floor(sq)
            if sq==sq1:
                f1 = m.gcd(p1-sq1,n)
                if f1%n != 1 and f1%n != 0:
                    return f1,m.gcd(p1+sq1,n)
            po=p
            qo=q
```

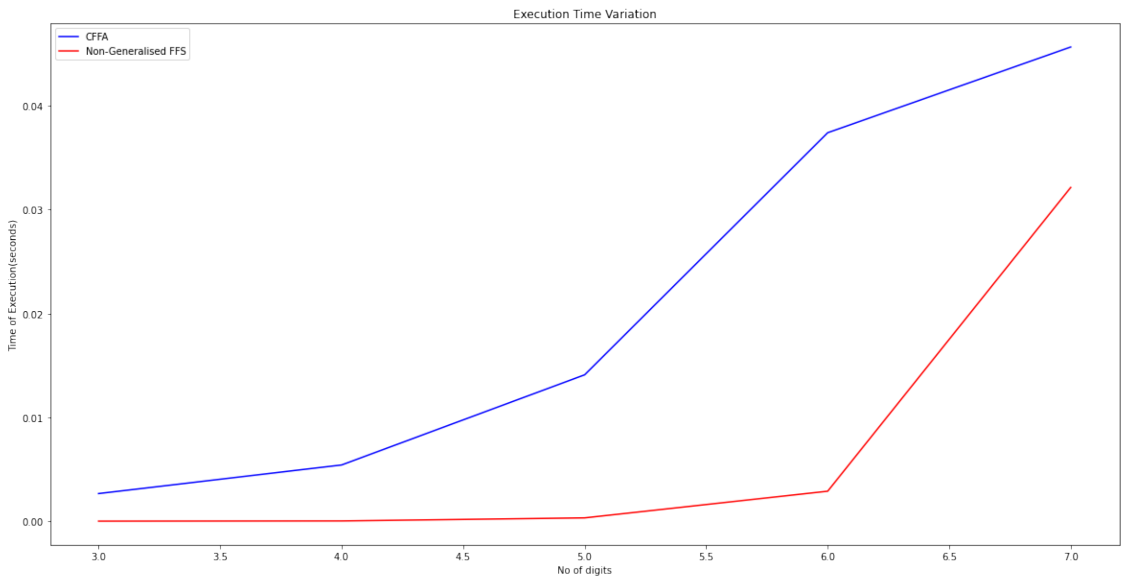
[135]:
```
import time
strt = time.perf_counter()
print(cffa(1234567895341,1000))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
```

```
(263, 4694174507)
Time taken = 0.4328593999998702
```

Time taken by fermat's non-generalized scheme ~ 1.8144788999998127

Time analysis

Plot below is a comparison between continued fraction factoring algorithm(CFFA) and fermat's non-generalised factorization scheme. $x - axis$ has the number of digits ranging from 3 to 8. $y - axis$ denotes the running time. For a specific digit, the running time is considered to be the average running time of 50 random odd non-prime numbers for both the algorithms. Although *CFFA* taken bound value as an input which can vary with input size but we return divisors as soon as they are found so we can choose a large bound so that all entries are factored.



B.3.5 QUADRATIC SIEVE ALGORITHM

Creating Factor Base

Note : Function $'jacobi()'$ for Jacobi symbol is given in B.2.1 which gives us value of Legendre symbol for prime numbers.

[69]:
```
def primes(B,n):
    factor_base = [2]
    for num in range(3,B + 1):
        for i in range(2, num):
            if (num % i) == 0:
                break
        else:
            if jacobi(num,n) == 1:
                factor_base.append(num)
    return factor_base
```

Working with Sieving Interval

[70]:
```
import math
def factoring(B,M,n): # B- Bound for prime factor base, M- sieving␣
↪interval bound, n- number to be factored
    power = 0
    dic = {}
    list_final=[]
    fx_list=[]
    x_list=[]
    gif = math.floor(pow(n,1/2))
    factor_base = primes(B,n)
    for i in range(gif-M,gif+M+1):
        num = i*i-n
        for j in factor_base:            # collecting powers of primes of␣
↪factor base
            while num%j == 0:
                power += 1
                num = num/j
            dic["{}".format(j)]= power
            power = 0
            num = i*i-n

        number = 1
        for j in factor_base:             # checking if number factored␣
↪within our factor base or not
            number = number*pow(j,dic["{}".format(j)])
        if number == abs(num):
            x_list.append(i)
            fx_list.append(abs(num))
            list_final.append(dic.copy())
    return list_final,x_list,fx_list
```

Gaussian Elimination Step to get factors

[71]:
```
import numpy as np
def qsa(B,M,n):                                    # Bound for factor base,␣
↪Bound for sieving interval, odd integer to be factored
```

```
    list1 = []
    final_list,x_values,fx_values = factoring(B,M,n) # vectors list,␣
↪x-values, f(x) values
    factor_base = primes(B,n)

    len_final_list = len(final_list)                # no. of rows
    len_factor_base= len(factor_base)               # factor_base length

    #....... Setting Matrices ..............

    for i in range(len_final_list):
        list1.append(list(final_list[i].values()))
    main_matrix = np.array(list1)                   # Matrix with␣
↪vectors corresponding to factors powers before modulo 2
    #..............................................
    working_matrix = (main_matrix.copy()) % 2       # Matrix After␣
↪modulo 2 calculation
    tracking_matrix = np.identity(len_final_list)   # Identity matrix␣
↪for row operations history


    worked_row_list = []                            # List of row␣
↪numbers that have been eliminated.
    for column_no in range(len_factor_base):
        for row_no in range(len_final_list):
            if row_no not in worked_row_list:

                curr_row = working_matrix[row_no,]

                for row in range(row_no,len_final_list):
                    if row not in worked_row_list:
                        if all([i==0 for i in (working_matrix[row,]%2)]):

                            # .............FINAL X VALUE..............
                            x_value = 1
                            for i,j in␣
↪zip(tracking_matrix[row,],x_values):
                                if i!= 0:
                                    x_value = (x_value*j)%n

                            #.............FINAL Y VALUE............
                            y_value = 1
                            sum_rows=np.zeros((1,len_factor_base))
                            for i in range(len_final_list):
                                if tracking_matrix[row,i]==1:
                                    sum_rows += main_matrix[i,]
                            for power,factor in␣
↪zip(sum_rows[0],factor_base):
                                y_value *= pow(factor,int(power/2),n)

                            # .... Checking if Non-trivial Divisor...... ␣
↪

                            divisor =  math.gcd(x_value+y_value,n)
                            if divisor not in [1,n]:
                                return "Divisors of {} are = {} and {} ".
↪format(n,divisor,int(n/divisor))
                                break

                #....... Gaussian Elimination..............
                if working_matrix[row_no,column_no] == 1:
                    for r in range(row_no+1,len_final_list):
                        if working_matrix[r,column_no]==1:
                            ␣
↪working_matrix[r,]=(working_matrix[r,]+curr_row)%2
                            tracking_matrix[r,]=(tracking_matrix[r,]+␣
↪tracking_matrix[row_no,])%2
                    worked_row_list.append(row_no)
```

[72]:

```
import time
strt = time.perf_counter()
```

```
print(qsa(10,10,4295229443))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
```

```
Divisors of 4295229443 are = 65539 and 65537
Time taken = 0.0007643000008101808
```

[73]:
```
strt = time.perf_counter()
print(qsa(80,1720,1234567895341))
end = time.perf_counter()
print(f'Time taken = {end-strt}')
```

```
Divisors of 1234567895341 are = 28710881287 and 43
Time taken = 0.055301099999269354
```

Time taken by continued fraction factoring algorithm ~ 0.43285939999998702

Time taken by fermat's non-generalized scheme ~ 1.81447889999998127

B.4 Polynomial factorization in $\mathbb{Z}[X]$ using $LLL-$algorithm

Example : Finding irreducible factors of
$f(x) = 8x^{10} + 28x^9 + 24x^8 + 36x^7 + 14x^6 + 35x^5 - 12x^4 + 22x^3 - 13x^2 + 5x - 3$ **in** $\mathbb{Z}[X]$

Solution: Factors are: $(x+3), (2x-1), (2x^2+1), (x^4+x^3+x^2+1)$

[35]:
```
import sympy as sp
from sympy import prime
from sympy.abc import x
from sympy import *
from sympy import GF
from sympy import sqf_part
from sympy import factor_list
from sympy import ZZ
```

[34]:
```
f = Poly(8*x**10 + 28*x**9 + 24*x**8 + 36*x**7 + 14*x**6 + 35*x**5 -␣
↪12*x**4 + 22*x**3 - 13*x**2 + 5*x - 3, x, domain='ZZ')
```

Note: f is a global variable.

B.4.0.0.0.0.1 Getting square-free part of $f(x)$.

[36]:
```
f = sqf_part(f)
f
```

[36]:

$$\mathrm{Poly}\left(4x^8 + 14x^7 + 10x^6 + 11x^5 + 2x^4 + 12x^3 - 7x^2 + 5x - 3, x, domain = \mathbb{Z}\right)$$

B.4.0.0.0.1 Choosing suitable $'p'$ values such that $f(x)\ (mod p)$ is square-free in $\mathbb{Z}_p[X]$.

[52]:
```
def factor_base(f, pbound):
    factor_base = []
    poly_list = []
    prime_list = [prime(i) for i in range(1,pbound+1)]
    for num in prime_list:
        coeff_f = f.all_coeffs()

        if coeff_f[0]%num != 0:

            #Reducing f(x) mod(num)
            K = GF(num)
            g = sp.Poly(coeff_f,x,domain = K)

            # Checking if square_free
            if sp.discriminant(g)!= 0:
                poly_list.append(g)
                factor_base.append(num)
    return factor_base,poly_list
```

[38]:
```
factor_base(f,10)
```

[38]:
```
([5, 13, 17, 29],
 [Poly(-x**8 - x**7 + x**5 + 2*x**4 + 2*x**3 - 2*x**2 + 2, x, modulus=5),
 Poly(4*x**8 + x**7 - 3*x**6 - 2*x**5 + 2*x**4 - x**3 + 6*x**2 + 5*x -␣
↪3, x,
modulus=13),
 Poly(4*x**8 - 3*x**7 - 7*x**6 - 6*x**5 + 2*x**4 - 5*x**3 - 7*x**2 + 5*x␣
↪- 3,
x, modulus=17),
 Poly(4*x**8 + 14*x**7 + 10*x**6 + 11*x**5 + 2*x**4 + 12*x**3 - 7*x**2 +␣
↪5*x -
3, x, modulus=29)])
```

B.4.0.1 FACTORING $f(x)$ $mod(p)$ IN $\mathbb{Z}_p[X]$.

B.4.0.1.0.1 Distinct-degree Factorization(DDF)

[39]:
```
from sympy import monic

def DDF(f,p):  # Enter 'square-free', 'primitive' f(x) \in Z[X].

    K = GF(p)
    f_modp = Poly(f, domain = K)
    f = monic(f_modp,domain=K)    # Converting to Monic in K.
    f_coef = f_modp.all_coeffs()
    degree_f = f.degree()

    # We have monic,square-free POLYNOMIAL =  f, COEFFICIENTS = f_coef,␣
↪PRIME = p
    F = f
    DDF_list = []
    degree = 1
    deg_list = []
    total_deg = 0
    while total_deg != degree_f:

        if p**degree > f.degree():
            B = sp.poly(x**p)
            for i in range(1,degree):
                B = sp.rem(B**p,f,domain = K)
        else:
            B = sp.poly(x**(p**degree),domain = K)
        factor = sp.gcd(f,Poly(B-sp.Poly([1,0],x)),domain = K)
        if factor != 1:
            deg_list.append(degree)
            DDF_list.append(factor)
            total_deg = total_deg + degree
        degree = degree+1

        f = sp.quo(f,factor)
        if f == 1:
            break
    return DDF_list, deg_list,p
```

B.4.0.1.0.2 Equal-Degree Factorization(EDF)

[40]:
```
def EDF(f,p):

    A_d,deg_list,p = DDF(f,p)
    factors_list = []
    irr_factors_list = []
    K = GF(p)
    n_factors = 0
    if p>2:

        for (poly,deg) in zip(A_d,deg_list):

            # Checking if Poly is an irreducible polynomial
            if deg == poly.degree():
                irr_factors_list.append(poly)
                n_factors = n_factors +1
            else:
                n_factors = n_factors + (poly.degree()/deg)
                T = sp.random_poly(x,poly.degree(),inf = 0,sup = p,domain␣
 ↪= K)
```

```
            d = 2
            d1 = ((p**deg)-1)/2
            T_2 = T**2
            while d <= d1:
                T_2 = sp.rem(T_2,poly)*T
                d = d+1
            F1 = sp.gcd(T_2+1,poly,domain = K)
            F2 = sp.gcd(T_2-1,poly,domain = K)
            F3 = sp.quo(poly,F1*F2,domain = K)

            deg_f1 = F1.degree()
            deg_f2 = F2.degree()
            deg_f3 = F3.degree()

            if deg_f1 != 0:
                if deg_f1 == deg:
                    irr_factors_list.append(F1)
                else:
                    factors_list.append([F1,deg])
            if deg_f2 != 0:
                if deg_f2 == deg:
                    irr_factors_list.append(F2)
                else:
                    factors_list.append([F2,deg])
            if deg_f3 != 0:
                if deg_f3 == deg:
                    irr_factors_list.append(F3)
                else:
                    factors_list.append([F3,deg])
    return factors_list,irr_factors_list,p, n_factors
```

B.4.0.1.0.3 Factoring completely using DDF and EDF as above

Programs below are to be hadled for each prime p separately.

===

```
[53]: factors_list, irr_factors_list,p,n_factors = EDF(f,5)  # Input :␣
       ↪polynomial in Z[X], primes bound, prime number
      K = GF(p)
      while len(irr_factors_list) != n_factors:
          list2 = []
          for elt in factors_list:
              polynomial = elt[0]
              degree = elt[1]
              T = sp.random_poly(x,degree,inf = 0,sup = p,domain = K)
              d = 2
              d1 = ((p**degree)-1)/2
              T_2 = T**2
              while d <= d1:
                  T_2 = sp.rem(T_2,polynomial)*T
                  d = d+1
              F1 = sp.gcd(T_2+1,polynomial,domain = K)
              F2 = sp.gcd(T_2-1,polynomial,domain = K)
              F3 = sp.quo(polynomial,F1*F2,domain = K)
              deg_f1 = F1.degree()
              deg_f2 = F2.degree()
              deg_f3 = F3.degree()
              if deg_f1 != 0:
                  if deg_f1 == degree:
                      irr_factors_list.append(F1)
                  else:
                      list2.append([F1,degree])
              if deg_f2 != 0:
                  if deg_f2 == degree:
                      irr_factors_list.append(F2)
                  else:
                      list2.append([F2,degree])
              if deg_f3 != 0:
                  if deg_f3 == degree:
                      irr_factors_list.append(F3)
                  else:
```

```
                list2.append([F3,degree])
    factors_list = list2.copy()
print("FACTORS=", irr_factors_list)
```

```
FACTORS= [Poly(x**2 - 2, x, modulus=5), Poly(x**4 + x**3 + x**2 + 1, x,
modulus=5), Poly(x - 2, x, modulus=5), Poly(x + 2, x, modulus=5)]
```

B.4.0.1.0.3.1 Storing all possible (h modp) in list H

[42]:
```
H = []
irr_factors_list
for factor in irr_factors_list:
    if sp.rem(f,Poly(factor,domain = ZZ)) != 0:
        H.append(factor)
H
```

[42]:
```
[Poly(x - 2, x, modulus=5),
 Poly(x + 2, x, modulus=5),
 Poly(x**2 - 2, x, modulus=5)]
```

B.4.0.1.0.3.2 Irreducible factors so far, (in Z[X]). Stored in irr_factors_list.

[43]:
```
for poly in H:
    irr_factors_list.remove(poly)
irr_factors_list
```

[43]:
```
[Poly(x**4 + x**3 + x**2 + 1, x, modulus=5)]
```

Note: Lists 'H' and 'irr_factors_list' are global variables.
===

B.4.0.1.0.4 Calculating k for suitable p^k value

[54]:
```
import scipy.special as ss
from sympy import Matrix
from sympy import ceiling,Float
def start(f,H,r,p):  # Input : f - polynomial, H[r]- Entries from H, p -␣
 ↪prime number from factor_base.
    h = H[r]
    deg_h = h.degree()
    n = f.degree()
    m = n-1
    M = Matrix(list(f.coeffs()))
    B = 2**(m*n/2) * (ss.binom(2*m,m))**(n/2) * M.norm()**(m+n)
    k = ceiling(ceiling(sp.log(B+1,p))/deg_h)
    return k
```

B.4.0.1.0.5 Hensel lifting

B.4.0.1.0.6 Lifting $(mod p)$ to $(mod p^2)$.

[55]:
```
def hensel_1(f,g,p): # Input: f - Polynomial, g - Monic factor in Z_p[X],␣
 ↪p - Prime from factor base
    K = GF(p)
    K_1 = GF(p*p)
    h = Poly(sp.quo(f,g,domain = K),domain = ZZ)
    t,s,l = sp.gcdex(g,h,domain = K)
    g = Poly(g,domain = ZZ)
    e = Poly(f-g*h,domain = K_1)
    q,r = sp.div(s*e,g,domain = K_1)
    g1 = Poly(g+r,domain = K_1)
    h1 = Poly(h+t*e+q*h,domain = K_1)
    return g1,h1
```

B.4.0.1.0.7 Lifting $(mod q)$ to $(mod qr)$ where $r = gcd(p,q)$ for some p, q (not necessarily prime).

[56]:
```
import math as m
def hensel_2(u,v,q,p):  # u = vw in Z_q s.t. av+bw = 1 in Z_p
    r = m.gcd(q,p)
    K = GF(q)
    K_1 = GF(q*r)
    w = Poly(sp.quo(u,v,domain = K),domain = ZZ)
    a,b,l = sp.gcdex(v,w,domain = GF(p))
    b = Poly(b,domain = ZZ)
    v = Poly(v,domain = ZZ)
    f = Poly(sp.quo(u-v*w,sp.Poly([q],x),domain = ZZ))
    t= sp.quo(b*f,v)
    v_1 = b*f - t*v
    w_1 = a*f + t*w
    V = v + q*v_1
    W = w + q*w_1
    V = Poly(V,domain = K_1)
    W = Poly(W,domain = K_1)
    return V,W
```

[47]:
```
def hmodpk(f,H,r,p):   # Input : f,H[r],p as in above programs

    k = start(f,H,r,p)
    h = Poly(H[r],domain = ZZ)
    o = 1
    while 2**o < k:   # k factored as p^(2^(r-1)).p^(k-2^(r-1))
        o = o+1
    n = 1
    q = p
    while n != o:
        h_1,g_1 = hensel_1(f,h,q)
        q = q*q
        h = Poly(h_1,domain = ZZ)
        n = n+1
    h_1,g_1 = hensel_2(f,h,q,int(p**(k-2**(o-1)))    )
    return h_1
```

B.4.0.2 Getting irreducible factor in $\mathbb{Z}[X]$ for each suitable $h \pmod{p}$

[48]:
```
import olll

def red_lattice(f,H,r,p):

    h_mod_pk = hmodpk(f,H,r,p)
    n = f.degree()
    l = h_mod_pk.degree()

    # Creating set of m values
    u = 0
    M = []
    while l <= (n-1)/(2**u):
        M.append(int((n-1)/(2**u)))
        u = u+1
    M.reverse()
    k = start(f,H,r,p)
    f_norm = Matrix(list(f.all_coeffs())).norm()

    for m in M:

        # CREATING LATTICE BASIS VECTORS
        P1 = []
        P2 = []
        for i in range(l):
            if i != 0:
                P1.append((p**k)*sp.poly(x**i))
            else:
                P1.append((p**k)*sp.Poly([1],x))
        for j in range(m-l+1):
            if j != 0:
                P2.append(Poly(h_mod_pk,domain = ZZ)*sp.poly(x**j))
            else:
                P2.append(Poly(h_mod_pk,domain = ZZ)*sp.Poly([1],x))

        # Managing dimension of vectors and collecting in one list     ␣
↪

        Lattice = []
        dim_vectors = m+1
        for poly in P1:
            poly_list = poly.all_coeffs()
            poly_list.reverse()
            for i in range(m+1):
                if i > len(poly_list)-1:
                    poly_list.append(0)
            Lattice.append(poly_list)
        for poly in P2:
            poly_list = poly.all_coeffs()
            poly_list.reverse()
            for i in range(m+1):
                if i > len(poly_list)-1:
                    poly_list.append(0)
            Lattice.append(poly_list)

        # GETTING REDUCED LATTICE BASIS VECTORS
        Lattice2 = []
        for vec in Lattice:
            L = []
            for num in vec:
                L.append(int(num))
            Lattice2.append(L)
        reduced_lattice = olll.reduction(Lattice2,3/4)

        # CHECKING IF deg(h0) <= m
        lattice_poly = []
        bound = (p**(k*l)/(f_norm**m))**(1/n)

        t = 0
        for vector in reduced_lattice:

            if Matrix(vector).norm() < bound:
                vector = list(vector)
                vector.reverse()
                lattice_poly.append(sp.Poly(vector,x,domain = ZZ))
                t = t+1
            else:
                break
        if t != 0:
            H_0 = lattice_poly[0]
            for i in range(t):
                H_0 = sp.gcd(H_0,lattice_poly[i])
            return H_0
```

B.4.0.3 Factoring completely into irreducibles in $\mathbb{Z}[X]$

```
[49]: def factorization(f,H,p):
          h_0 = red_lattice(f,H,0,p)
          print(h_0)
          fact_check = [h_0]
          for i in range(1,len(H)):
              if sp.rem(h_0,H[i]) != 0:
                  h_0 = red_lattice(f,H,i,p)
                  if h_0 not in fact_check:
                      print(h_0)
```

```
[50]: factorization(f,H,5)
      print(irr_factors_list)
```

```
Poly(x + 3, x, domain='ZZ')
Poly(2*x - 1, x, domain='ZZ')
Poly(2*x**2 + 1, x, domain='ZZ')
[Poly(x**4 + x**3 + x**2 + 1, x, modulus=5)]
```

BIBLIOGRAPHY

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

[2] L. Babai, *On lovász' lattice reduction and the nearest lattice point problem*, Combinatorica, 6 (1986), pp. 1–13.

[3] E. Berlekamp, *Factoring polynomials over large finite fields*, Mathematics of Computation, 24 (1970), pp. 713–735.

[4] M. R. Bremner, *Lattice Basis Reduction: An Introduction to the LLL Algorithm and Its Applications*, CRC Press, Boca Raton, 2011.

[5] D. M. Bressoud, *Factorization and Primality Testing*, Springer-Verlag, Berlin, Heidelberg, 1989.

[6] D. M. Burton, *Elementary Number Theory*, McGraw Hill Education, LLC, New York, 7 ed., 2012.

[7] M. E. C. P. Schnorr, *Lattice basis reduction: improved practical algorithms and solving subset sum problems*, Mathematical Programming, 66 (1994), pp. 181–199.

[8] J. Cassels, *An Introduction to the Geometry of Numbers*, Springer-Verlag, Berlin, Heidelberg, 1971.

[9] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, Berlin, Heidelberg, 1993.

[10] A. Das, *Computational Number Theory*, CRC Press, first edition ed., 2013.

[11] L. Dickson, *History of the theory of numbers, vol-1*, G.E. Stechert and company, 1934.

[12] U. Dieter, *How to calculate shortest vectors in a lattice*, Mathematics of Computation, 29 (1975), pp. 827–833.

[13] M. Dietzfelbinger, *Primality Testing in Polynomial Time : From randomized algorithms to primes is in P*, Springer, Berlin, Heidelberg, 1 ed., 1973.

[14] Dummit and Foote, *Abstract Algebra*, John Wiley and Sons inc., second edition ed., 1999.

[15] N. Gopalakrishnan, *University Algebra*, New Age International Ltd., third edition ed., 2015.

[16] M. H. Gunawan, W. Setya-Budhi and S. Gemawati, *On volumes of n-dimensional parallelepipeds in* $l^p spaces, Publikacije\ Elektrotehnickog\ fakulteta - serijamatematika, (2005), pp. 48 - -54.$

[17] I. Herstein, *Topics in Algebra*, John Wiley and Sons, second edition ed., 1975.

[18] K. H. Hoffman and R. Kunze, *Linear Algebra*, Prentice Hall, second edition ed.,

1971.

[19] T. W. HUNGERFORD, *Algebra*, Springer-Verlag, New York, first edition ed., 2004.

[20] R. R. ISHMUKHAMETOV ST, MUBARAKOV BG, *On the number of witnesses in the miller–rabin primality test*, Symmetry, 12.

[21] H. KEMPFERT, *On the factorization of polynomials*, Journal of Number Theory, 1 (1969), pp. 116–120.

[22] N. KOBLITZ, *A Course in Number Theory and Cryptography*, Springer-Verlag,New York, second edition ed., 2010.

[23] A. K. LENSTRA, H. W. L. JR, AND L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Mathematische Annalen, 261 (1982), pp. 515–534.

[24] W. J. LEVEQUE, *Topics in Number Theory, Volumes I and II*, Addison Wesley, 1956.

[25] F. T. LUK AND D. M. TRACY, *An improved lll algorithm*, Linear Algebra and its Applications, 428 (2008), pp. 441–452.

[26] M. MIGNOTTE, *An inequality about irreducible factors of integer polynomials*, Journal of Number Theory, 30 (1988), pp. 156–166.

[27] P. Q. NGUYEN AND B. V. (EDITORS), *The LLL Algorithm: Survey and Applications*, Springer, Heidelberg, 2010.

[28] M. POHST, *A modification of the lll reduction algorithm*, Journal of Symbolic Computation, 4(1) (1987), pp. 123–127.

[29] H. RIESEL, *Prime numbers and computer methods of factorization*, Progress in mathematics,Birkhausar, Boston, 126 (1994).

[30] A. S.H.FRIEDBERG AND L.E.SPENCE, *Linear Algebra*, Prentice-Hall, Englewood Cliffs, New Jersey, second edition ed., 1989.

[31] V. SHOP, *A Computational Introduction to number theory and algebra*, Cambridge university press, 2005.

[32] R. SILVERMAN, *The multiple polynomial quadratic sieve*, Mathematics of Computation, 48 (1987), pp. 329–340.

[33] J. VON ZUR GATHEN AND J. GERHARD, *Modern Computer Algebra*, Cambridge University Press, New York, 3 ed., 2013.