**INTEGER FACTORIZATION**

**I. Aim**

To learn various algorithms on integer factorization and their time complexities.

**II. Background**

**1.Introduction**

Integer factorization needs great computational powers compared to primality testing, which does not require the same computations.

**2.Methodologies used for factorization**

Algorithms for integer factorization can split into two groups, the **special** and the **general** algorithms.

All algorithms described depend on the fact that the number n to be factored is a composite, so it should test for primality before attempting to factor it.

**2.1. Special Algorithms**

A special-purpose factoring algorithm's running time depends on the number's properties to be factored or on one of its unknown factors: size, unique form, etc. That is, the numbers on which these algorithms perform operations are of some special class. So, the running time depends on various algorithms. Given an integer of unknown form, these methods are usually applied before general-purpose methods to remove small factors.

**Following are some special algorithms used for integer factorization:**

**2.1.1. Trial Division**

Trial division is the fastest method for small composite number. it divides possible factors to see if the remainder is zero. It does not fail for hard composites, and it just takes a long time.

Trial division is useless for composite numbers with only large factors, but for smaller composites, it is a practical algorithm, and it is a good exercise to implement it.

Time complexity=O(sqrt(n))

**2.1.2. Pollard’s p − 1 method**

John M. Pollard proposed this new algorithm for factoring integers. Which is based on Fermat’s Little Theorem.

It specializes in integers with some factor p where p − 1 is **B-smooth** for some relatively small B.

**B-smooth**: Let B be a positive integer, then an integer n is said to be B-smooth with respect to a bound if all its prime factors are less than equal to B.

Time complexity=O(B \* ( ln(n) / ln(B) ) B: smoothness bound.

**2.1.3. Pollard’s ρ(rho) method**

John m. Pollard proposed a Monte Carlo method for the factorization of integers. The Pollard ρ method of factoring showed some new methods usable for future factoring algorithms. It has some similarities with the p − 1 method, but it has a better way of choosing the divisors.

**It is specialized for composite integers having small factors**.

The idea of the ρ-method described in these steps :

1. Calculate a sequence of integers a0,..., am that are periodically recurrent modulo p.

2. Find the period of the sequence, i.e., find i and j such that ai ≡ aj mod p.

3. Identify the factor p.

Time complexity= O( n1/4 )

**2.1.4. Elliptic Curve Method (ECM)**

Lenstra in 1985 suggested using elliptic curves for factoring composites; shortly after that, Brent refined the ideas of Lenstra. The ECM method is like Pollard’s p−1 method; it is almost the same, but Lenstra applied it to the points on an elliptic curve.

John m. Pollard works are in the field of the multiplicative group of the finite field K, and Lenstra used a group based on the points of an elliptic curve. The group operations of an elliptic curve group are more expensive, but it allows using several groups at the same time.

Time complexity= O( ( L(p) ^ sqrt(2) + o(1) ) + M(log n) )

L(p)= e ^ sqrt(log p log log p)

**2.2 General Algorithms**

Some of the special methods for factoring are not usable to factor RSA composites, because the primes used are chosen to be hard. The algorithms usable for RSA composites are the general ones, and for the size of composites used in RSA today, only the number field sieve that is applicable.

All modern general-purpose factoring algorithms based on the same principle and idea, namely congruent squares.

**Congruent Squares**

Congruent square is a pseudonym for Legendre’s congruence.

Legendre’s Congruence:

x^2 ≡ y ^2 mod n, 0 ≤ x ≤ y ≤ n, x != y, x + y != n

If some integers x and y satisfy Legendre’s Congruence, then gcd(n, x − y) and gcd(n, x + y) are mostly nontrivial factors of n.

Modern factoring methods all consist of the three necessary steps

1. Find a set of relations that are smooth over some factor base.

2. Solve the system of linear equations and search the connections that yield squares.

3. Calculate the gcd of the composite, and the squares are available.

These are the steps used by modern factoring methods; the difference is how they find the pairs of integers that satisfy Legendre’s Congruence.

**2.2.1. Continued Fractions(CFRAC)**

Although we had congruent squares, CFRAC is known as the first modern method. The continued fractions method (CFRAC) is unusual because it uses an entirely different approach for finding the desired squares than any of the following methods. CFRAC uses continued fractions. A continued fraction is a way of number representation and is used to represent real numbers with integers.

Continued Fraction uses the convergents of the continued fraction of (n)^(1/2), and the smooth over the factor base kept as relations. The relations are put through a linear algebra step and the ones that yield a square then used for the gcd step.

Time complexity=O(n ^ ( sqrt( 1.5 \* ( ln ln n / ln n))))

**2.2.2 Quadratic Sieve**

Quadratic Sieve (QS) avoids a large part of the useless divisions done in CFRAC.

QS is also based on the idea of congruent squares, like its name indicates it uses a quadratic polynomial and quadratic residues. For integers in the 60-110 digit range, it is still the fastest method.

Time Complexity= O( ( L(p) ^ sqrt(2) + o(1) ) + M(log n))

L(p)= e ^ sqrt(log p log log p)

**2.2.3 General Number Field Sieve**

GNFS is the fastest general algorithm known today, but because of its complexity and overhead, it is only faster than the QS for numbers larger than 110-120 digit range. The number field sieve uses algebraic number fields.

The number field sieve is similar to the quadratic sieve, but it differs on one major point: the field it works in.

Time complexity= O( ( e^ c + O(1) \* (log n)1/3 \* (log log n)2/3 )

For Special NFS c = (32/9)1/3

For General NFS c= (64/9)1/3

**III. Work Focused:**

**General Number Field Sieve Algorithm**

GNFS is the fastest Algorithm known till today.

The algorithm consists of 5 main steps, mutually dependent and can not be made in parallel, but some of the levels can be parallel internally.

**Step 1**: Polynomial Selection

The two main factors that influence the yield of a polynomial are the size and root properties. A polynomial f (x) is said to be good if it has a good return, i.e., it has the right combination of size and root properties.

A polynomial f (x) has a suitable size property if the values taken by f (x) are small. f(x) can achieve by selecting a polynomial with small Coefficients.

If f(x) has many roots modulo small primes, then polynomial f (x) is said to have excellent root properties.

The polynomial f (x) has to have the following properties:

1. irreducible over Z[x]

2. has a root modulo n

polynomials then put through a small sieving test and thereby identifying the one with the best yield.

**Step 2:** Factor Bases

The algorithm needs a well-defined domain to work in, and this is specified with the factor bases.

**Step 3:** Sieving

It is the most time-consuming part. The need for the sieving step is to find good relations, that is, elements (a, b) with the following properties:

• gcd(a, b) = 1

• a + bm is smooth over the rational factor base

• b deg(f ) f (a/b) is smooth over the algebraic factor base

The purpose of the sieving stage is to collect as many relations as possible (at least one larger than the elements in all of the bases combined). The sieving step results in a set S of relations.

**Step 4:** Linear Algebra

We Want to find numbers which form a product that is a square.

This is equivalent to solving a system of linear equations and can be done by building a matrix and eliminating it. For a number to be a true square, the elements in its unique factorization must have an even power, and this property is what we use to find elements that are squares.

This matrix can then put on reduced echelon form, and from this, we can derive solutions that yield a square (Gaussian Elimination).

**Step 5**: Square Root

We need the square root of the solution, a rational square root Y and an

algebraic square root X.

The rational square root is trivial; a wide variety of methods for deriving the square root is known.

Finding the square root of an algebraic integer is equivalent to finding the square root of a polynomial over an extension field.

**IV. Status of Integer Factorization**

The number field sieve is the most complex factoring algorithm, but it is also the fastest and has successfully factored a 512-bit composite. Since the number field sieve arrived around 1990, there has not been any significant new ideas, only optimizations to the number field sieve.

It is impossible to predict the future of Integer Factorisation, but if we look at current development, there is some research in **Factorization Circuits and Factorization in Graph Theory**. Research has been done for many years on designing circuits for factoring, and the idea seems to be a central topic. The main problem is the price of such a circuit compared to mainstream PC architectures.

**Research Work on Integer Factorisation:**

GIFT – Graphical Integer Factorization Technique

GIFT is an ongoing research algorithm, involving a new integer factorization method that joins number theory with graph theory. It changes any positive integer into a directed graph. From such a graph, we can quickly identify NODES, STICKS, LOOPS, and SELF-LOOPS. Each component has a specific numerical value, and any two or more constitute a single FORMULA. Joining all formulas creates the EQUATION or the final mathematical relation. To solve it for any UNKNOWN, we do a search experiment or run simultaneously many computers to generate a multiple of any prime factor. It is best to search for such in two distinct ways. The first basic algorithm of GIFT looks for a possible large while the second algorithm considers smaller numerical values. In other words, the first algorithm performs a fast search (by increments of powers of ten) while the second algorithm searches sequentially; that is, it does not jump any integer. If successful, we factored (possibly in a short time) the given modulus.GIFT needs mathematical relation, but the computer finds the mathematical relationship, the human user does not need to intervene in any manner. Such is the main advantage of GIFT.

To the best of our knowledge, this is the first-ever integer factorization method based on visual computing.

**V. Conclusion**

1. **Trial Division** does not fail for hard composites, and it just takes a long time.
2. An attempt to use **Pollard’s p-1 method** on non-B-smooth integers will fail.
3. In 1980 Richard P. Brent refined the algorithm of **Pollard’s rho method**, and it is his version that is commonly used. It is about **25% faster than the original Pollard version**, and it has been applied to some large composites like the 8th Fermat number

in 1980, it took 2 hours on a UNIVAC. The difference from the original version is in the

way it detects cycles in sequence

1. **ECM** is the fastest of the special algorithms.
2. For many years **CFRAC** was the fastest algorithm for large composites; it first surpassed when the quadratic sieve emerged, and later the number field sieve.
3. One of the shortcomings of the CFRAC algorithm is that a lot of useless divisions are performed. CFRAC does not have a way of assuring that the divisions that do are on numbers that are smooth and end up as a relation. Carl Pomerance presented a new factoring algorithm called the **Quadratic Sieve (QS)**, which avoided many of the useless divisions done in CFRAC.
4. The **time complexity of the QS algorithm is the same as that of the ECM**, but the **ECM operations are more expensive**. So, for composites with only substantial prime factors, the QS method is faster than ECM.
5. The  **General** **Number Field Sieve(GNFS)** is the fastest general algorithm known today. Still, because of its complexity and overhead, it is only quicker than the QS for numbers more significant than 110-120 digits.

**Factoring strategy:**

If we were to factor a random integer we would not start by applying a general purpose algorithm. We should start by applying the ECM to reveal small factors and if it is not capable to factor it in reasonable time one should roll in the heavy artillery and apply the QS (for n < 120 digits) or else the number field sieve.

If we know that n is a RSA modulus then there is no reason to use anything other than the number field sieve.

**VI. References**

[1] Advances in Computational research, JAYBHAY N.\* AND JADHAV P. MCA Department, Marathwada Institute of Technology, Aurangabad- 431 028, MS, India.

[2] Integer factorization, Per Leslie Jensen, Department of Computer Science, University of Copenhagen, Fall 2005.

[3] João Carlos Leandro da Silva, Giuseppe de Nicolellis, rainbowofprimes.com

[4] A Guide to General Number Field Sieve Integer Factorization,Gireesh Pandey,

Defence Research and Development Organisation