Intro to Number Theory

Lecture and
LaTeX by Eric Xingyu Yang
Monash University
Rosen [1], [2] are the main references

Semester one 2025

Contents

Chapter 1

Divisibility and Modular Arithmetic

1.1 Division

One of the most fundamental concepts in number theory is the division of integers. Whether a number is divisible by another number? How can we determine the divisibility of two numbers? How can we express one number in terms of another? These questions lead us to the foundation of number theory, which is one of the most abstruse and beautiful branches of mathematics.

Number theory plays a crucial role in many fields:

- **Cryptography**: The security of many encryption algorithms relies on the properties of prime numbers and modular arithmetic.
- **Algebraic Geometry**: Number theory is used to study the properties of algebraic varieties over finite fields. This is crucial in coding theory and cryptography as well. This is also known as the hardest math branch encompassing almost all the math fields. An example is Elliptic Curve Cryptography(ECC).
- **Computer Science**: In computer science, number theory is used in algorithms, data structures, and complexity theory. A typical example is the design of hash functions.
- **Optimization and Operations Research**: Number theory is used in optimization problems, such as integer programming and combinatorial optimization. In some cases, we can use number theory notions to simplify the problem, such as **modular congruence** and **Diophantine equations**.

It might be trivial, since we have been using division since we were kids, but it is necessary to give division a formal definition.

Definition 1.1.1: True Division

Formally, the division of two real numbers $a, b \in \mathbb{R}$ is defined as:

$$a \div b = c$$

where c is the unique number such that:

$$a = b \times c.$$

In other words, $a \div b = c \iff a = b \times c$. The division is defined for all $a \in \mathbb{R}$ and $b \in \mathbb{R} \setminus \{0\}$.

The division is not defined for $b = 0$ because it leads to an undefined result. This is actually a very polemical topic in the history of mathematics, and gave birth to the **division by zero** paradox, contributing to the formalization of infinity, which is a milestone in the development of calculus and analysis.

While this is only an intro, in number theory, we are more interested in the division of integers. Why?

Fact 1.1.2

- **Integers are the building blocks of all numbers**: All numbers can be expressed as a combination of integers.
- **Integers are discrete**: Unlike real numbers, integers are discrete and can be counted. This makes them easier to work with in many cases.
- **Unique factorization**: Every integer can be **uniquely** factored into other numbers, which is a fundamental property of integers.

Definition 1.1.3: Division with Remainder

The division of a number a by a non-zero number b is denoted by $a \div b$ or $\frac{a}{b}$, and it gives the quotient q and possibly a remainder r. The operation can be written as:

$$a = b \times q + r$$

where $0 \leq r < b$ and $q \in \mathbb{Z}$.

Definition 1.1.4: Divisibility

If a and b are integers with $a \neq 0$, we say that a divides b if there is an integer c such that $b = a \times c$ (or equivalently, if b an is an integer). When a divides b we say that a is a factor or divisor of b, and that b is a multiple of a. The notation a | b denotes that a divides b. We write $a \nmid b$ when a does not divide b

Divisibility lays the foundation of number theory, and later we will introduce how to use

it to define modular arithmetic.

Example.

For example, we have $4 \mid 20$, because $20 \div 4$ gives an integer, however, $4 \nmid 21$, as the answer cannot be represented as integer. Besides, we have:

$$a \mid b \iff \exists c(ac = b). \tag{1.1}$$

Problem.

Prove that if a and b are integers and a divides b, then a is odd or b is even.

Proof. We could try proof by contrapositive. Suppose for a is even and b is odd, $a, b \in \mathbb{Z}$, $a \mid b$. Then $b = ka$ for some integer k. Make $a = 2n$, $b = 2n + 1$, $n \in \mathbb{Z}$, then there should be $2n + 1 = 2kn$. However, in this case we have $k = \dfrac{2n+1}{2n}$, meaning that k is not an integer, which contradict the assumption. This completes the proof. □

We may find more properties of divisibility.

Proposition 1.1.5

- **Transitive property**: If $a \mid b$ and $b \mid c$, then $a \mid c$.
  - If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.
  - If $a \mid b$ then $a \mid bc$ for any integer c.
- **Reflexive property**: For any integer a, $a \mid a$.
- **Symmetric property**: If $a \mid b$, then $b \nmid a$ unless $a = b$.
- **Trivial property**: If $a = 0$, then $0 \mid b$ for all integers b.

Proof. If $a \mid b$ and $b \mid c$, then there exist integers p and q such that $b = ap$ and $c = bq$. Substituting the expression for b into the equation for c gives $c = (ap)q = a(pq)$. Since pq is an integer, $a \mid c$.

Since $a \mid b$ and $a \mid c$, there exist integers m and n such that $b = am$ and $c = an$. Therefore, $b + c = am + an = a(m + n)$, and since $m + n$ is an integer, it follows that $a \mid (b + c)$.

Given $a \mid b$, there exists an integer k such that $b = ak$. For any integer c, $bc = a(kc)$. Since kc is an integer (because the product of two integers is an integer), $a \mid bc$.

□

Problem.

Prove that if a, b, and c are integers, where $a \neq 0$, such that $a \mid b$ and $a \mid c$, then $a \mid mb + nc$ whenever m and n are integers.

Proof. Since $a \mid b$ and $a \mid c$, we have $a \mid b + c$. Hence, we have $b + c = ak_1$ for some integer k_1. Let $b = ak_1 - c$ and $c = ak_1 - b$ in $mb + nc$, we have:

$$\begin{aligned} mb + nc &= m(ak_1 - c) + n(ak_1 - b) \\ &= ak_1(m + n) - mc - nb \\ &= a(m + n)k_1 - mc - nb \end{aligned}$$

Since $b = ak_2$ and $c = ak_3$, we have:

$$\begin{aligned} mb + nc &= a(m + n)k_1 - m(ak_2) - n(ak_3) \\ &= a(m + n)k_1 - ak_2m - ak_3n \\ &= a((m + n)k_1 - k_2m - k_3n), \end{aligned}$$

where $(m + n)k_1 - k_2m - k_3n \in \mathbb{Z}$. Hence, $a \mid mb + nc$. □

1.2 Modular Arithmetic

1.2.1 Div and Mod Operator

Notation.

div and **mod** are two operators that are used to perform division and find the remainder of a division operation, respectively.

- **div**: The div operator is used to perform integer division. It returns the quotient of the division operation, discarding any remainder. For example, $7 \div 3 = 2$ because $3 \times 2 = 6$ and the remainder is discarded.
- **mod**: The mod operator is used to find the remainder of a division operation. It returns the remainder after dividing one number by another. For example, $7 \bmod 3 = 1$ because $7 = 3 \times 2 + 1$.

  We also use % to denote the mod operator, which is more common in programming languages. But we will use **mod** in mathematical context.

Remark.

Note that, the **mod** operator should be bold in a printed document, since we need to differentiate it with $\equiv \pmod{}$, which is used for modular congruence.

You may find that the **div** and **mod** operators are actually binary and can be encapsulated by binary equations.

Definition 1.2.1: Div and Mod

Let $\mathbb{Z}$ denote the set of all integers. We define two functions as follows:

1. The integer division function, denoted $\text{div} : \mathbb{Z} \times (\mathbb{Z} \setminus \{0\}) \to \mathbb{Z}$, is defined by

$$\text{div}(a, b) = \begin{cases} \left\lfloor \frac{a}{b} \right\rfloor, & \text{if } \frac{a}{b} \geq 0, \\ \left\lceil \frac{a}{b} \right\rceil, & \text{if } \frac{a}{b} < 0, \end{cases}$$

   where $\lfloor x \rfloor$ is the floor function returning the greatest integer less than or equal to x, and $\lceil x \rceil$ is the ceiling function returning the smallest integer greater than or equal to x.

   This choice ensures that the quotient $\text{div}(a, b)$ is selected to guarantee a non-negative remainder in the corresponding modulo operation.

2. The modulo function, denoted $\text{mod} : \mathbb{Z} \times (\mathbb{Z} \setminus \{0\}) \to \mathbb{Z}$, is defined by

$$\text{mod}(a, b) = a - b \cdot \text{div}(a, b),$$

   which yields the unique remainder satisfying

$$0 \leq \text{mod}(a, b) < |b|.$$

Rounding direction of $\text{div}(a, b)$ depends on signs of a and b as summarized in the following table:

Case	a	b	$\frac{a}{b}$	$\text{div}(a, b)$ rounding
1	$+$	$+$	$+$	floor
2	$+$	$-$	$-$	ceiling
3	$-$	$+$	$-$	ceiling
4	$-$	$-$	$+$	floor

Here, "floor" means $\left\lfloor \frac{a}{b} \right\rfloor$ and "ceiling" means $\left\lceil \frac{a}{b} \right\rceil$.

These definitions ensure that for any integers a, b with $b \neq 0$, the division algorithm holds:

$$a = b \cdot \text{div}(a, b) + \text{mod}(a, b),$$

with the remainder $\text{mod}(a, b)$ uniquely satisfying

$$0 \leq \text{mod}(a, b) < |b|.$$

Let's try to prove them!

Lemma 1.2.2: Range of Mod Operation

For any integers a and b with $b \neq 0$, the modulo operation satisfies:

$$\text{mod}(a, b) \in [0, |b|),$$

This means that the result of the modulo operation is always a non-negative integer less than the absolute value of b.

Proof for Lemma

According to the Division Algorithm, for any integer a and non-zero integer b, there exist unique integers q and r such that:

$$a = bq + r,$$

and the remainder satisfies

$$0 \leq r < |b|.$$

Here, we let

$$q = \text{div}(a, b), \quad r = \text{mod}(a, b).$$

This means that the value of the modulo operation $\text{mod}(a, b)$ is the remainder r, satisfying the above inequality. By definition, we have

$$\text{mod}(a, b) = a - b \cdot \text{div}(a, b).$$

To ensure $0 \leq \text{mod}(a, b) < |b|$, we need to choose a suitable quotient so that the remainder falls within this interval. In the definition based on the floor function, when $b > 0$, selecting $\text{div}(a, b) = \left\lfloor \frac{a}{b} \right\rfloor$ can meet the requirement. But when $b < 0$, $\left\lfloor \frac{a}{b} \right\rfloor$ may not satisfy this condition. Therefore, to ensure that the remainder is non-negative and less than $|b|$, when $b < 0$, we should define

$$\text{div}(a, b) = \left\lceil \frac{a}{b} \right\rceil,$$

such that

$$0 \leq a - b \cdot \text{div}(a, b) < |b|.$$

In summary, regardless of whether b is positive or negative, there always exists a suitable integer quotient $\text{div}(a, b)$ such that the remainder $\text{mod}(a, b)$ satisfies

$$0 \leq \text{mod}(a, b) < |b|.$$

This completes the proof. ∎

Problem.

Prove that for any integers a and b with $b \neq 0$,

$$a = b \cdot \text{div}(a, b) + \text{mod}(a, b).$$

Proof. By the division algorithm, for any integers a and b with $b \neq 0$, there exist unique integers q and r such that

$$a = bq + r,$$

where

$$0 \leq r < |b|.$$

Here, $\text{div}(a, b)$ is defined as the quotient q, and $\text{mod}(a, b)$ is defined as the remainder r. Hence, by definition,

$$a = b \cdot \text{div}(a, b) + \text{mod}(a, b),$$

with

$$0 \leq \text{mod}(a, b) < |b|.$$

This completes the proof. □

Problem.

Use the div and mod functions from the division algorithm to define the following three Boolean functions:

AND Function: Define the AND operation for Boolean variables $x, y \in \{0, 1\}$, i.e.,

$$\text{and}(x, y) = ?$$

OR Function: Define the OR operation for Boolean variables $x, y \in \{0, 1\}$, i.e.,

$$\text{or}(x, y) = ?$$

XOR Function: Define the XOR operation for Boolean variables $x, y \in \{0, 1\}$, i.e.,

$$\text{xor}(x, y) = ?$$

Please define these functions using expressions containing only div and mod.

Solution

Assuming $x, y \in \{0, 1\}$, consider the sum $x + y$ which can be 0, 1, or 2. Using the division algorithm with divisor 2, we can represent $x + y$ as:

$$x + y = 2 \cdot \text{div}(x + y, 2) + \text{mod}(x + y, 2).$$

1. **AND Function** The AND operation corresponds to whether both inputs are 1, which happens only if the sum is 2:

   $$\text{and}(x, y) = \text{div}(x + y, 2).$$

   This yields 1 only when $x = y = 1$.

Figure 1.1: Truth Table for AND: $\text{and}(x, y) = \text{div}(x + y, 2)$

x	y	$x + y$	$\text{and}(x, y)$
0	0	0	$\text{div}(0, 2) = 0$
0	1	1	$\text{div}(1, 2) = 0$
1	0	1	$\text{div}(1, 2) = 0$
1	1	2	$\text{div}(2, 2) = 1$

2. **OR Function** Using De Morgan's law, the OR can be written in terms of AND as:

$$\text{or}(x, y) = 1 - \text{and}(1 - x, 1 - y).$$

Using the expression for AND above:

$$\text{or}(x, y) = 1 - \text{div}\big(2 - (x + y), 2\big).$$

This expression is 1 if either input is 1, and 0 only when both are 0.

Figure 1.2: Truth Table for OR: $\text{or}(x, y) = 1 - \text{div}(2 - (x + y), 2)$

x	y	$x + y$	$2 - (x + y)$	$\text{div}(2 - (x + y), 2)$	$\text{or}(x, y)$
0	0	0	2	1	$1 - 1 = 0$
0	1	1	1	0	$1 - 0 = 1$
1	0	1	1	0	$1 - 0 = 1$
1	1	2	0	0	$1 - 0 = 1$

3. **XOR Function** The XOR operation corresponds to the parity of $x + y$:

$$\text{xor}(x, y) = \text{mod}(x + y, 2).$$

Intuitively, this gives 1 when exactly one of x, y is 1, and 0 otherwise.

Figure 1.3: Truth Table for XOR: $\text{xor}(x, y) = \text{mod}(x + y, 2)$

x	y	$x + y$	$\text{xor}(x, y)$
0	0	0	$\text{mod}(0, 2) = 0$
0	1	1	$\text{mod}(1, 2) = 1$
1	0	1	$\text{mod}(1, 2) = 1$
1	1	2	$\text{mod}(2, 2) = 0$

1.2.2 Modular Congruence

Chapter 2

Primes, GCD, LCM

2.1 Prime Numbers

Definition 2.1.1: Prime Number

An integer p greater than 1 is called prime if the only positive factors of p are 1 and p. A positive integer that is greater than 1 and is not prime is called **composite**.

The reason why prime numbers are so fascinating to mathematicians is that they have so many interesting and unique property, even except for what we have seen in its definition. Below is what we call the fundamental theorem of arithmetic.

Theorem 2.1.2: Fundamental Theorem of Arithmetic

For every integer $n > 1$, there exists a unique factorization into prime numbers, up to the order of the factors. Specifically, n can be expressed as

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

where $p_1 < p_2 < \ldots < p_k$ are prime numbers and $a_1, a_2, \ldots, a_k$ are positive integers. This factorization is unique, apart from the order of the prime factors.

Proof. We prove the theorem in two parts: existence and uniqueness.

Existence: We prove by mathematical induction that every integer greater than 1 can be written as a product of primes.

Base Case: For $n = 2$, the statement holds true since 2 is itself a prime number.

Inductive Step: Assume the statement holds for all integers greater than 1 and less than n. Now consider the integer n.

- If n is prime, then it is trivially a product of primes (itself).
- If n is not prime, it can be written as $n = a \cdot b$ where $1 < a, b < n$. By the inductive hypothesis, both a and b can be factored into a product of primes. Therefore, n can also be expressed as a product of primes by combining the prime factorization of a and b.

This completes the proof of existence.

Uniqueness: Assume, for the sake of contradiction, that there are two distinct prime factorization of n:

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k} = q_1^{b_1} \cdot q_2^{b_2} \cdot \ldots \cdot q_m^{b_m}$$

where p_i and q_j are prime numbers, and a_i, b_j are positive integers. To proceed to the rest of the proof, we need to use Euclid's lemma. (Just take it as a known result for now, we will revisit this in Bezout's identity.)

Lemma 2.1.3: Euclid's Lemma

Let p be a prime number. If p divides the product ab, where a and b are integers, then p divides a or p divides b.

Proof for Lemma

Assume p is a prime that divides ab but does not divide a. We need to show that p must divide b.

Since p does not divide a, the greatest common divisor (gcd) of a and p is 1, i.e., $\gcd(a, p) = 1$. According to Bezout's identity, there exist integers x and y such that:

$$ax + py = 1$$

Multiplying both sides of the equation by b, we get:

$$abx + pby = b$$

Since p divides ab (by assumption), p divides abx. Also, p obviously divides pby. Hence, p divides the sum $abx + pby$, which means p divides b.

This completes the proof, showing that if p divides ab and does not divide a, then p must divide b, in accordance with Euclid's lemma. ■

By Euclid's lemma, if a prime divides the product of two numbers, it must divide at least one of those numbers. Thus, p_1 must divide some q_j on the right-hand side. Since q_j is prime, we conclude that $p_1 = q_j$. Applying this argument symmetrically and repeatedly, we find that each set of prime factors must be identical to the other, contradicting the assumption of two distinct factorization.

Therefore, the prime factorization of any integer greater than 1 is unique, up to the order of the factors, which completes the proof of the Fundamental Theorem of Arithmetic. □

Example.

100 can be taken as $100 = 2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2$. Both 2 and 5 are primes.

2.1.1 Prime Factorization Algorithm

We can use an iterative algorithm to find the prime factorization of a number. The algorithm works by dividing the number by the smallest prime factor repeatedly until the number

becomes 1. The prime factors are collected in a list. The algorithm can be implemented as follows:

Algorithm 1 Prime Factorization

```
function PrimeFactorization(n)
    factors ← an empty list
    for p ← 2 to ∞ do                          ▷ Iterate over all primes
        while n  mod p == 0 do
            Add p to factors
            n ← n/p
        end while
        if p > n/p then
            break
        end if
    end for
    if n > 1 then
        Add n to factors
    end if
    return factors
end function
```

Example.

To find the prime factorization of 8964:

- Start with $n = 8964$.
- Divide by 2: $8964 \div 2 = 4482$ (add 2 to factors).
- Divide by 2: $4482 \div 2 = 2241$ (add 2 to factors).
- Now, 2241 is not divisible by 2, so we try the next prime, which is 3.
- Divide by 3: $2241 \div 3 = 747$ (add 3 to factors).
- Divide by 3: $747 \div 3 = 249$ (add 3 to factors).
- Divide by 3: $249 \div 3 = 83$ (add 3 to factors).
- Now, 83 is a prime number.
- The final prime factorization is: $8964 = 2^2 \cdot 3^3 \cdot 83^1$.

This algorithm seems trivial, yet it is actually very challenging to implement in practice. Because it takes huge efforts to find the first n primes, and the algorithm is not efficient. Some more efficient algorithms for the purpose are:

- **Pollard's rho algorithm**: This is a probabilistic algorithm for integer factorization that is particularly effective for large numbers.
- **Elliptic Curve Factorization**: This is another advanced method for integer factorization that uses properties of elliptic curves.
- **General Number Field Sieve (GNFS)**: This is the most efficient classical algorithm for factoring large integers.

2.2 Prime Testing and Algorithms

The property of primes brings us to a question: how can we determine whether a number is prime or not? If we want to get many primes, how long will it take? What is the complexity of the algorithm? How can we find the next prime number after a given number?

Definition 2.2.1: Prime Testing

There are several algorithms for testing the primality of a number. Some of the most common ones include:

- **Trial Division**: This is the simplest method, where we check if a number n is divisible by any prime number less than or equal to $\sqrt{n}$. If it is, then n is composite; otherwise, it is prime.
- **Sieve of Eratosthenes**: This algorithm generates all prime numbers up to a given limit n by iteratively marking the multiples of each prime starting from 2.
- **Fermat's Little Theorem**: This theorem provides a probabilistic test for primality. It states that if p is a prime and a is an integer not divisible by p, then $a^{p-1} \equiv 1 \mod p$.

We will introduce the first two algorithms in detail, and the Fermat's Little Theorem will be introduced later for other purposes.

2.2.1 Trial Division

Trial division is a straightforward and naive algorithm for testing the primality of a number. The idea is simple, we can search for primes linearly from 2 to to any range of numbers we want to check. However, to make sure whether a number is prime or not, we can actually limit the search space to the square root of the number.

Proposition 2.2.2

This is because if n is divisible by any number greater than $\sqrt{n}$, it must also be divisible by a number less than $\sqrt{n}$.

Proof. If n is composite, by the definition of a composite integer, we know that it has a factor a with $1 < a < n$. Hence, by the definition of a factor of a positive integer, we have $n = ab$, where b is a positive integer greater than 1. We will show that $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. If $a > \sqrt{n}$ and $b > \sqrt{n}$, then $ab > \sqrt{n} \cdot \sqrt{n} = n$, which is a contradiction. Consequently, $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Because both a and b are divisors of n, we see that n has a positive divisor not exceeding $\sqrt{n}$. This divisor is either prime or, by the fundamental theorem of arithmetic, has a prime divisor less than itself. In either case, n has a prime divisor less than or equal to $\sqrt{n}$. □

The algorithm works as follows:

Algorithm 2 TrialDivision(n)

Require: Integer $n > 1$

Ensure: **True** if n is prime, otherwise **False**

```
for d ← 2 to ⌊√n⌋ do
    if d | n then
        return False                    ▷ n is divisible by d, so not prime
    end if
end for
return True                             ▷ n has no divisors other than 1 and itself
```

Example.

> For example, to check if 29 is prime, we only need to check divisibility by $2, 3, 4, 5$ (up to $\lfloor\sqrt{29}\rfloor = 5$). Since 29 is not divisible by any of these numbers, it is prime.

Complexity of Trial Division

The time complexity of the trial division algorithm is $O(\sqrt{n})$ because we only need to check divisibility up to $\sqrt{n}$. This makes it inefficient for large numbers, especially when n is very large.

Space complexity is $O(1)$ since we only need a constant amount of space to store the variables used in the algorithm.

Correctness of Trial Division

The proof is straightforward. You may also try to prove it using induction or loop invariant yourself.

2.2.2 Sieve of Eratosthenes

The Sieve of Eratosthenes is an efficient algorithm for finding all prime numbers up to a specified integer n. The algorithm works by iteratively marking the multiples of each prime number starting from 2. The remaining unmarked numbers are primes.

Algorithm 3 SieveOfEratosthenes(n)

Require: Integer $n > 1$

Ensure: List of prime numbers up to n

```
Create a list of integers from 2 to n
for p ← 2 to √n do
    if p is not marked then
        for k ← p² to n with step p do
            Mark k as composite
        end for
    end if
end for
Return the unmarked numbers as primes
```

Example.

To find all prime numbers up to 30:

- Start with a list of numbers from 2 to 30.
- Mark 2 and its multiples: $4, 6, 8, \ldots, 30$.
- Mark 3 and its multiples: $9, 12, 15, \ldots, 30$.
- Continue this process until reaching $\sqrt{30} \approx 5.48$.
- The remaining unmarked numbers are the primes: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29$.

Complexity of Sieve of Eratosthenes

The time complexity of the Sieve of Eratosthenes is $O(n \log(\log(n)))$, which is significantly faster than trial division for large n. The space complexity is $O(n)$ because we need to store the list of integers from 2 to n.

We can show the complexity formally:

Lemma 2.2.3: Sieve of Eratosthenes Complexity

The Sieve of Eratosthenes has a time complexity of $O(n \log(\log(n)))$ and a space complexity of $O(n)$.

Proof for Lemma

To analyze the time complexity, consider the algorithm proceeds by iteratively marking the multiples of each prime p starting from 2 up to $\sqrt{n}$. For each prime p, the number of multiples marked is approximately $\frac{n}{p} - 1$.

Therefore, the total number of marking operations is bounded by

$$T(n) \leq \sum_{\substack{p \leq \sqrt{n} \\ p \text{ prime}}} \left(\frac{n}{p} - 1 \right) \leq n \sum_{p \leq \sqrt{n}} \frac{1}{p}.$$

It is a classical result in analytic number theory (see Mertens' theorems) that the sum of reciprocals of primes up to x satisfies

$$\sum_{p \leq x} \frac{1}{p} = \log \log x + O(1).$$

By substituting $x = \sqrt{n}$, we get

$$\sum_{p \leq \sqrt{n}} \frac{1}{p} = \log \log \sqrt{n} + O(1) = \log \frac{1}{2} \log n + O(1) = \log \log n + O(1).$$

Hence, the time complexity satisfies

$$T(n) = O\big(n \log \log n\big).$$

Regarding space complexity, the algorithm requires $O(n)$ space to store a boolean array (or similar data structure) representing the primality status of each integer from 2 up to n.

This completes the proof of the time and space complexity of the Sieve of Eratosthenes. ∎

Apart from Sieve of Eratosthenes, Sieve of Ktkin is also a well-known algorithm for finding all prime numbers up to a specified integer n. It is more efficient than the Sieve of Eratosthenes for large values of n.

2.3 Greatest Common Divisor and Least Common Multiple

2.3.1 Greatest Common Divisor

Definition 2.3.1: Greatest Common Divisor

The greatest common divisor (gcd) of two integers a and b, denoted $\gcd(a, b)$, is the largest positive integer that divides both a and b without leaving a remainder.

Euclidean Algorithm

Chapter 3

Modular Equations and Diophantine Equations

3.1 Solving Linear Congruence

3.1.1 Modular Inverse

3.2 Diophantine Equations

3.2.1 Linear Diophantine Equations

3.3 System of Linear Congruence

3.3.1 Chinese Remainder Theorem

3.4 Fermat's Little Theorem

Chapter 4

Applications of Number Theory

4.1 Cryptography

4.1.1 Basics of Cryptosystems

4.1.2 Diffie-Hellman Key Exchange

RSA Algorithm

4.2 Hash Functions

4.2.1 Modular Hash Functions

4.2.2 Multiplicative Hash Functions

4.2.3 Polynomial Hash Functions

4.2.4 SHA-256

4.3 Representation of Integers

4.3.1 Modular Exponentiation Algorithm

4.3.2 Base Representation and Conversion

Bibliography

[1] K. Rosen, *Discrete Mathematics and Its Applications*, 8th ed. New York (N.Y.): McGraw-Hill Education, Aug. 10, 2018, 2240 pp., ISBN: 978-1-260-09199-1.

[2] K. Rosen, *Elementary Number Theory and Its Application, 6th Edition*, 6th edition. Boston: Pearson, Mar. 30, 2010, 752 pp., ISBN: 978-0-321-50031-1.

[3] E. Lehman, F. T. Leighton, and A. R. Meyer, *Mathematics for Computer Science (Lehman, Leighton, and Meyer)*. Mar. 8, 2017, ISBN: 978-988-8407-06-4. [Online]. Available: `https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_and_Computation_Fundamentals/Mathematics_for_Computer_Science_(Lehman_Leighton_and_Meyer)`.