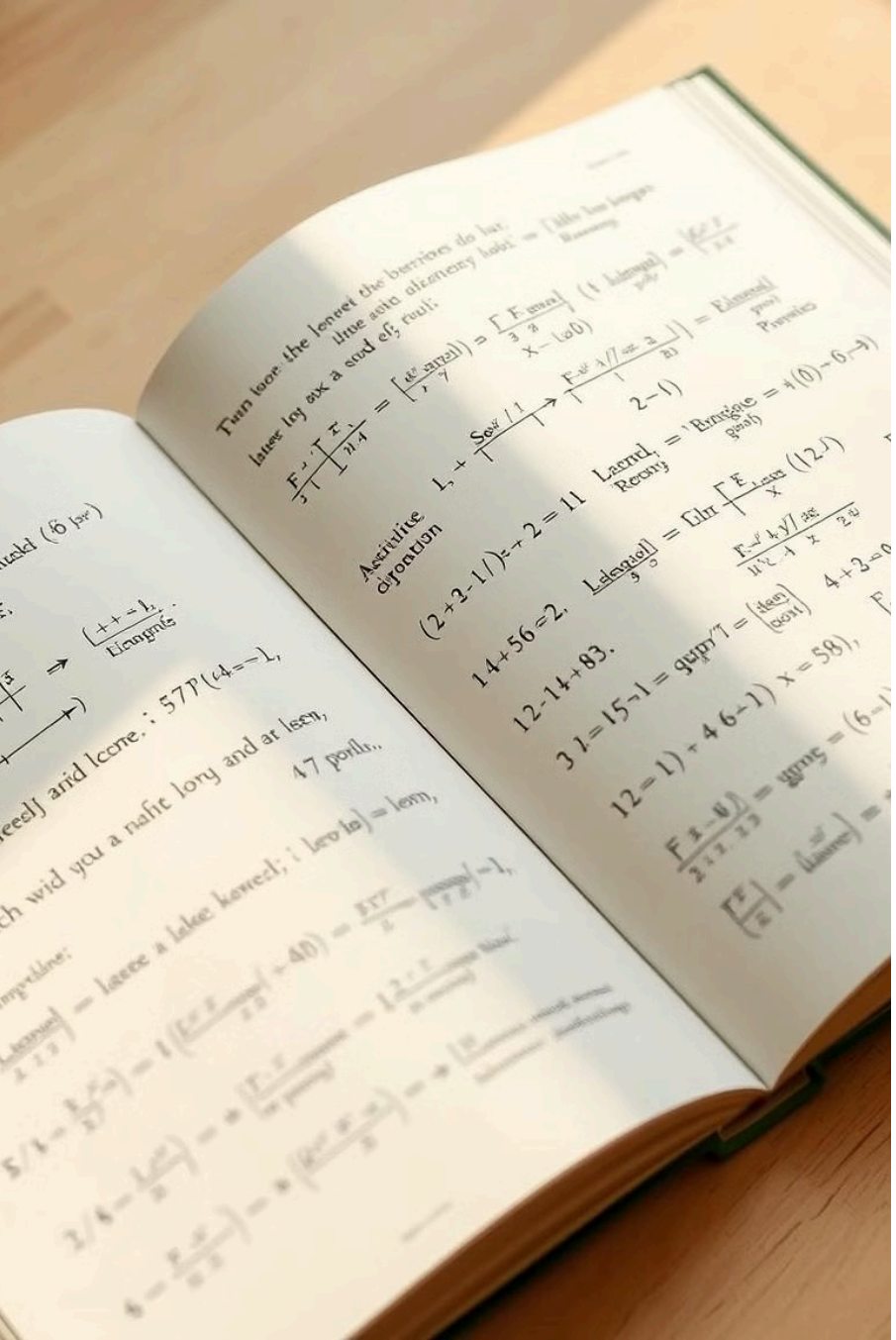


Introduction to Number Theory

- Number theory is a branch of mathematics focused on studying integers and their properties.
- It includes many concepts that are fundamental in in various fields in computer science, we'll look through these concepts and understand where they're applied in these fields

E **by Eric Kazungu**

Divisibility and Modular Arithmetic

- In this section we introduce the concept of divisibility as a a foundational step to understand modular arithmetic.
- Division of an integer by a positive integer produces a quotient and a remainder.
- With this in mind we can define divisibility as follows "*If a and b are integers with $a \neq 0$, we say that a divides b if there is an integer c such that $b = ac$ "* such that when one integer is divided by a second nonzero integer, the quotient must be an integer

The Division Algorithm:

It states that when an integer **a** is divided by a positive integer **d**, the result includes:

- A **quotient** q
- A **remainder** r

ie "**a=dq+r"**

NB *Despite its name, the "Division Algorithm" is* ***not an actual algorithm***

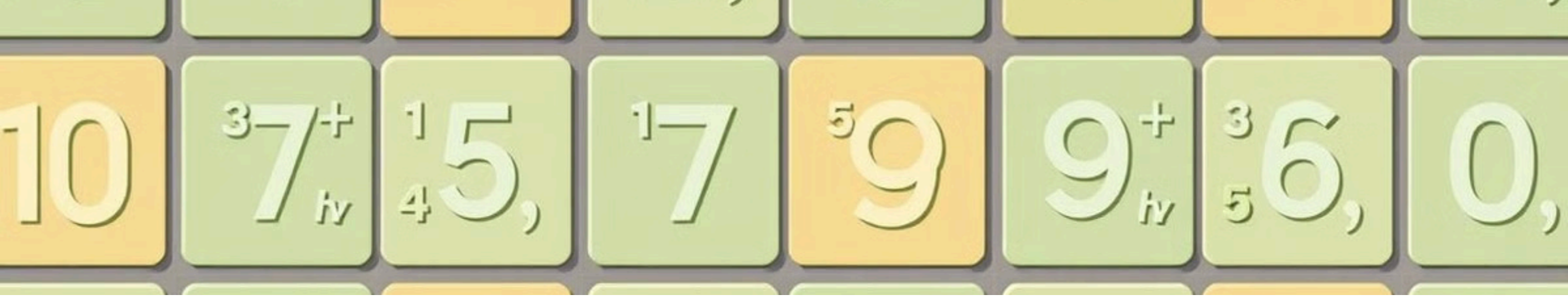
Modular Arithmetic

Like how in a clock the number "wraps around" once it hits 12, starting again from zero.

Modular arithmetic is a system of arithmetic for integers where numbers "wrap around" after reaching a certain value (the modulus).

Given integers a and m (where m>0), the notation **a mod m** represents the remainder when a is divided by m.

Eg in a clock $11+2 \equiv 1$ (13 mod 12 = 1)



Divisibility and Modular Arithmetic

Congruences (Modulo Operation)

Two integers a and b are **congruent modulo m** if they have the same remainder when divided by m.

This is written as: **$a \equiv b \pmod{m}$**

This implies that **$m \mid (a-b)$**

Modular Addition and Multiplication

- **Addition:**
  - $(a+b) \bmod m = ((a \bmod m)+(b \bmod m)) \bmod m$
- **Multiplication:**
  - $(a \times b) \bmod m = ((a \bmod m) \times (b \bmod m)) \bmod m$.

Integer Representations and Algorithm

- Integers can be expressed using any base greater than one
- While base 10 (decimal) is commonly used in everyday life, computer science often uses:
  - **Binary (base 2)**: Represents numbers using 0 and 1.
  - **Octal (base 8)**: Uses digits from 0 to 7.
  - **Hexadecimal (base 16)**: Uses digits 0-9 and letters A-F to represent values 10-15.

The Norm: Decimal Notation

It uses digits 0 to 9

An integer n can be written as:

$$n = a_k \cdot 10^k + a_{k-1} \cdot 10^{k-1} + \cdots + a_1 \cdot 10 + a_0$$

which just means in a nutshell

$$965 = 9 \cdot 10^2 + 6 \cdot 10 + 5.$$

General Base Representation (Base b)

this can be extended to any base to convert that base into the decimal representation.

For example a base 8 number can be converted to decimal in the following way

$$(245)_8 = 2 \cdot 8^2 + 4 \cdot 8 + 5 = 165$$

Conversion Algorithms

Base Conversion Algorithm

- Divide the decimal number n by the new base b.
- Record the remainder as the **rightmost digit**.
- Update n as the quotient.
- Repeat steps 1-3 until the quotient becomes zero.
- The base b representation is formed by reading the remainders **from bottom to top**.

Binary to Octal Conversion

- Group binary digits in **blocks of 3** from right to left.
- Convert each block to its octal equivalent.
- Example: (1111010111100) base 2= (37274) base 8

Binary to Hexadecimal Conversion

- Group binary digits in blocks of 4 from right to left.
- Convert each block to its hexadecimal equivalent.

Example:

(1111010111100) base 2= (3EBC) base 16

Octal to Binary Conversion

- Convert each octal digit to a **3-bit binary equivalent**.
- Example: (765) base 8= (111110101) base 2

Hexadecimal to Binary Conversion

Convert each hexadecimal digit to a 4-bit binary equivalent.

Example:

(A8D) base 16 = (101010001101) base 2

Algorithms for Integer Operations

- Binary arithmetic is fundamental in computer systems.
- Therefore, using efficient algorithms for integer addition, multiplication, division, and modular exponentiation are crucial for efficient computation.

The Addition Algorithm:

Steps to Perform Binary Addition:

1. Start by adding the rightmost bits (a0 and b0).
   - The sum is expressed as:

$$a_0 + b_0 = c_0 \cdot 2 + s_0$$

     - s0: Sum bit (rightmost bit of the result).
     - c0: Carry bit (either 0 or 1).
2. Move to the next pair of bits (a1 and b1) along with the carry from the previous step:

$$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$$

3. Continue adding pairs of bits along with the carry, moving from right to left.
4. The last stage adds the leftmost bits along with the final carry:

$$a_{n-1} + b_{n-1} + c_{n-2} = c_{n-1} \cdot 2 + s_{n-1}$$

   - The leading bit of the sum is:

$$s_n = c_{n-1}$$

5. The final binary sum of a and b is:

$$(s_n s_{n-1} \dots s_1 s_0)_2$$

Multiplication Algorithm

Steps to Perform Binary Multiplication:

1. Use the distributive property to express the product:

$$ab = a \cdot (b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_{n-1} \cdot 2^{n-1})$$

2. Compute each partial product:

$$a \cdot b_j \cdot 2^j$$

   - If bj = 1, shift a by j positions.
   - If bj=0, the partial product is 0.
3. Add the partial products to obtain the final product.

Primes and Greatest Common Divisors

- A **prime number** is an integer greater than 1 that is divisible only by 1 and itself.
- Prime numbers are crucial in modern **cryptographic systems**.
- Primes are essential for **public-key cryptography** in algorithms like **RSA,** where security is based on the **difficulty of factorizing large composite numbers**

Trial Division Algorithm

- The theorem this algorithm is based on states: If n is composite, it must have a **prime divisor less than or equal to the square root of n**
- Steps
  - Identify all primes up to the square root of n
  - Check Divisibility with each prime in the list
  - Then make a conclusion if no divisors are found, n is **prime** else n is composite

Prime Factorization

- This is a way to reduce a composite number into its primes and can is useful when calculating the GCD
- In GCD we factor both numbers into primes. Multiply the **smallest power** of each common prime.
- In LCM we multiply the **largest power** of each prime given a pair of numbers
- The process goes as follows
  - **Start with the smallest prime (2)** and check divisibility.
  - **If divisible, divide and continue with the quotient.**
  - **Repeat the process** with the next smallest prime.
  - **Stop when the quotient is prime** or equals 1.

The Sieve of Eratosthenes

This is an old algorithm to find all prime numbers upto a given positive integer n

Procedure

- Start with a list of integers from 1 to n (in this case, 100).
- Remove 1, as it is not prime.
- Mark all multiples of 2 (except 2 itself) as non-prime.
- Move to the next number (3) and mark all multiples of 3 (except 3 itself) as non-prime.
- Move to 5 and repeat the process.
- Move to 7 and do the same.
- Stop as the next prime (11) exceeds the square root of 100.
- The remaining unmarked numbers are prime.

Example

- Composite integers up to 100 must have a prime factor not exceeding 10.
- Primes less than 10 are: 2, 3, 5, and 7.
- Primes not exceeding 100 are either these four primes or numbers not divisible by any of them.

The result

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

The Sieve of Eratosthenes

The Infinitude of Primes

- This is a theorem that states that there are infinitely many primes
- **Proof by Contradiction (Euclid's Proof):**

1. Assume that the prime numbers are finite: p1,p2,...,pn.
2. Construct a new number Q=p1⋅p2⋅...⋅pn+1
3. **Observation:**
    - Q is either prime or divisible by a prime not in the original list.
    - None of the primes pj can divide Q because dividing Q by any prime leaves a remainder of 1.

Therefore there must exist a prime not in the original list, proving that the number of primes is infinite.

This proof however is nonconstructive meaning it shows the existence of a new prime but does not explicitly find it.

The Distribution of Primes

The ratio of $\pi(x)$, the number of primes not exceeding x, and $x / \ln x$ approaches 1 as x grows without bound. (Here ln x is the natural logarithm of x.)

In simple terms the number of primes up to x is approximately x/ln x

Euclidean Algorithm for Finding GCD

The key idea behind the Euclidean Algorithm is that any divisor of both a and b is also a divisor of r.

Here is an example with the procedure

- Divide the larger number (287) by the smaller number (91):
  - $287=91\times3+14$
- Reduce the problem to finding:
  - gcd (91 , 14)
- Repeat the process:
  - $91=14\times6+7$
- Reduce to finding:
  - gcd(14,7)
- Continue:
  - $14=7\times2+0$
- The last nonzero remainder is 7, so:
  - gcd(287,91)=7

Linear Combinations

- The GCD of two integers a and b can be expressed as a linear combination:
  - gcd(a,b)=sa+tb
  - s and t are integers called Bézout coefficients.

Example:

Find the GCD of 252 and 198 and express it as a linear combination:

First lets find the GCD

$252=198\times1+54$

$198=54\times3+36$

$54=36\times1+18$

$36=18\times2+0$

where 18 is the GCD

Working backwards to express 18 as a linear combination:

$18=54-1\times36$

$36=198-3\times54$

$18 = 54 - 1 \times (198 - 3 \times 54)$

$18= 4 \times 54 - 1 \times 198$

but $54 = 252 - 1 \times 198$

$18 = 4 \times (252 - 198) - 1 \times 198$

$18 = 4 \times 252 - 5 \times 198$

Final Linear Combination:

$18=4\times252-5\times198$

Solving Congruences

1

Linear Congruences

A linear congruence is a congruence relation of the form **ax ≡ b (mod m)**, where **a**, **b**, and **m** are integers, and **x** is the variable to be solved for. Solving linear congruences involves finding integer values of **x** that satisfy the congruence relation.

2

Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) provides a method for solving systems of linear congruences with different moduli, provided that the moduli are pairwise relatively prime. The CRT guarantees a unique solution modulo the product of the moduli.

3

Modular Inverses

The modular inverse of an integer **a** modulo **m** is an integer **x** such that **ax ≡ 1 (mod m)**. Modular inverses are useful for solving linear congruences and performing modular division.

Applications of Congruences

Hashing

- A technique used to efficiently store and retrieve data.A hashing function maps a key (e.g., customer ID) to a memory location.
- One common function is:
  - h(k)=k mod m
  - where k is the key, m is the number of memory locations and h(k) is the assigned memory location
- One major problem with hashing is that collisions, this is where two or more items are assigned the main memory location

Pseudorandom Number Generation

- Random numbers are needed for computer simulations, cryptography, and etc. True randomness is difficult to achieve, hence we use **pseudorandom numbers** that appear random but follow a deterministic pattern.
- One common method is Linear Congruential Method (LCG)

$$x_{n+1} = (ax_n + c) \mod m$$

- where:
  - xn is current number in the sequence, a is the multiplier "c' is the increment "m" is the modulus "x0" is the initial seed value
- However the major problems with deterministic systems is that once the initial parameters are known its possible to predict the next value

Check Digits

This is a technique to detect **errors in identification numbers**.

Other applications include

- **ISBN (Books)**: Uses a modulo-11 checksum.
- **Credit Cards**: Uses the **Luhn Algorithm** to detect single-digit errors.
- **Barcodes**: Retail product barcodes use check digits to verify correctness.