S2Template

18 August 2025 16:06

R0C0
R1C0
R1C1
R2C0
R2C1
R2C2

0 0 } - 1
1 0
1 1 } -2
2 0
2 1
2 2 } 3

```
for (int i=0; i<n; i++)
  for (int j=0; j<=i; j++)
    { print (i,j).
```

1. Number Systems Basics

Terminology:

- **Number System:** A way to represent numbers (e.g., Decimal, Binary).
- **Integer:** A whole number (positive, negative, or zero).
- **Divisor:** A number that divides another without leaving a remainder (e.g., 3 is a divisor of 12).

1. GCD (Greatest Common Divisor) Applications

- **Fraction Simplification**: Used to reduce fractions to their simplest form (e.g., $\frac{8}{12} = \frac{2}{3}$ by dividing numerator and denominator by GCD(8,12)=4).
- **Scheduling & Time Management**: Helps find repeating cycles (e.g., traffic lights syncing every GCD of their intervals).
- **Computer Algorithms**: Used in the **Euclidean algorithm** for efficient computation.
- **Cryptography**: Essential in **RSA encryption** and modular arithmetic.
- **Manufacturing**: Optimizing cutting stock problems (e.g., minimizing waste when cutting rods into equal lengths).

2. LCM (Least Common Multiple) Applications

- **Synchronization**: Used to find when two periodic events coincide (e.g., planets aligning, clocks chiming together).
- **Digital Signal Processing (DSP)**: Helps in determining sampling rates.
- **Time Management**: Calculating when two recurring events (like bus schedules) will next occur simultaneously.
- **Computer Science**: Used in memory management and cache optimization.

3. Prime Numbers in Real-World Applications

- **Cryptography (RSA, ECC)**: Primes are the backbone of secure encryption systems.
- **Hashing Algorithms**: Used in hash functions for efficient data retrieval.
- **Error Detection & Correction**: Used in checksums and digital signatures.
- **Random Number Generation**: Critical in simulations and cryptography.
- **Quantum Computing**: Shor's algorithm (for factorization) relies on primes.

Combined Use Cases

- **Banking & Security**: GCD helps in key generation, while primes secure transactions.
- **Telecommunications**: LCM helps in scheduling data packets, while primes ensure encryption.
- **Game Development**: Used in procedural generation and optimizing collision detection.

Method 1

36 = 2 X 2 X 3 X 3
60 = 2 X 2 X 3 X 5

GCD = Multiplication of Common Factors
= 2 X 2 X 3
= 12

Method 2

```
FUNCTION gcd_brute_force(a, b)
 FOR i FROM MIN(a, b) DOWNTO 1
    IF a % i == 0 AND b % i == 0
      RETURN i  // Early exit on first common divisor (the largest)
    END IF
  END FOR
  RETURN 1  // Fallback if no common divisor found (co-prime case)
END FUNCTION
```

When Brute-Force Backward Fails

If $a = 10^9 + 1$ and $b = 10^{18} + 7$ (co-prime), the loop runs 10^9 **times** before returning 1.

Never use brute-force for large numbers!

Euclidean algorithm
Replace (a, b) with (b, a % b)

```
Numbers:     [48, 18]
             └─[48 % 18 = 12]─┐
                              ↓
Numbers:     [18, 12]
             └─[18 % 12 = 6]──┐
                              ↓
Numbers:     [12, 6]
             └─[12 % 6 = 0]──→ GCD = 6
```

Check

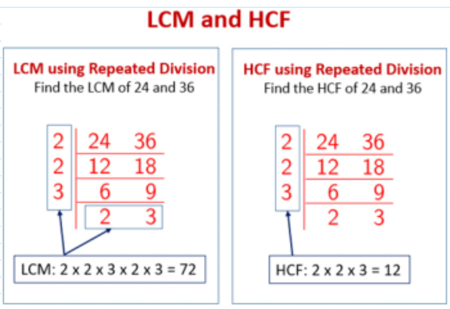
1220 mod 516 = 188
516 mod 188 = 140
188 mod 140 = 48
140 mod 48 = 44
48 mod 44 = 4
44 mod 4 = 0
4 = GCD

Why Euclidean Algorithm Wins
For large numbers (e.g., (a = 10^9\, b = 10^18)):
- **Brute-Force**: Takes billions of iterations (slow).
- **Euclidean**: Computes in ~60 steps (logarithmic time).

Why Euclidean Algorithm Wins

1. **Speed**:
   - Factorization becomes **infeasible** for numbers with large prime factors (e.g., RSA-2048).
   - Euclidean handles 10^{100}-digit numbers in milliseconds.
2. **Simplicity**:
   - No need to generate primes or factorize.
3. **Real-World Use**:
   - Used in cryptography (e.g., RSA), computer algebra systems.

1. Greatest Common Divisor (GCD) - Euclidean Algorithm
```plaintext
FUNCTION gcd(a, b)
    WHILE b != 0
        temp = b
        b = a MOD b
        a = temp
    END WHILE
    RETURN a
END FUNCTION
```



You can find the **LCM (Least Common Multiple) from the GCD (Greatest Common Divisor)** using this formula:

$$\mathrm{LCM}(a, b) = \frac{a \times b}{\mathrm{GCD}(a, b)}$$

Explanation:

- Multiply the two numbers.
- Divide the product by their GCD.

Example for 12 and 16:

GCD(12,16) = 4
LCM(12,16) = (12 × 16) ÷ 4 = 192 ÷ 4 = 48

Least Common Multiple (LCM) Using GCD**
```plaintext
FUNCTION lcm(a, b)
   RETURN (a * b) / gcd(a, b)
END FUNCTION
```

Divisor: A number that divides another without leaving a remainder (e.g., 3 is a divisor of 12).

Write all numbers that divide 36 completely

1
2
3
4
6
9
12
18

Prime Numbers

11

$\frac{11}{1}$ $\frac{11}{2}$ $\frac{11}{3}$ $\frac{11}{4}$ $\frac{11}{5}$ $\frac{11}{6}$ $\frac{11}{7}$ $\frac{11}{8}$ $\frac{11}{9}$ $\frac{11}{10}$ $\frac{11}{11}$

12

$\frac{12}{1}$ $\frac{12}{2}$ $\frac{12}{3}$ $\frac{12}{4}$ $\frac{12}{5}$ $\frac{12}{6}$ $\frac{12}{7}$ $\frac{12}{8}$ $\frac{12}{9}$ $\frac{12}{10}$ $\frac{12}{11}$ $\frac{12}{12}$

36	40
1 x 36 = 36	1 x 40 = 40
2 x 18 = 36	2 x 20 = 40
3 x 12 = 36	4 x 10 = 40
4 x 9 = 36	5 x 8 = 40
6 x 6 = 36	8 x 5 = 40
9 x 4 = 36	10 x 4 = 40
12 x 3 = 36	20 x 2 = 40
18 x 2 = 36	40 x 1 = 40
36 x 1 = 36	

3. Primality Test (Check if a Number is Prime)
```plaintext
FUNCTION is_prime(n)
   IF n <= 1
      RETURN False
   END IF
   FOR i FROM 2 TO sqrt(n)
      IF n MOD i == 0
         RETURN False
      END IF
   END FOR
   RETURN True
END FUNCTION
```

Find Prime Numbers Till 25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

4. Sieve of Eratosthenes (Generate All Primes Up to N)
```plaintext
FUNCTION sieve(n)
   is_prime = ARRAY of size (n+1) initialized to True
   is_prime[0] = False
   is_prime[1] = False
   FOR p FROM 2 TO sqrt(n)
      IF is_prime[p] == True
         FOR multiple FROM p*p TO n STEP p
            is_prime[multiple] = False
         END FOR
      END IF
   END FOR
   primes = []
   FOR i FROM 2 TO n
      IF is_prime[i] == True
         APPEND i to primes
      END IF
```

```
    END FOR
    RETURN primes
END FUNCTION
```

Let's illustrate where the inner loop starts for different values of p when running your sieve algorithm for $n = 25$:

- For $p = 2$, multiples loop starts at $2 \times 2 = 4$
- For $p = 3$, multiples loop starts at $3 \times 3 = 9$
- For $p = 5$, multiples loop starts at $5 \times 5 = 25$

So for $n = 25$:

- When $p = 2$, mark multiples at: 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
- When $p = 3$, mark multiples at: 9, 12, 15, 18, 21, 24
- When $p = 5$, mark multiples at: 25

Explanation:

For each prime p, the inner loop starts from $p \times p$ because smaller multiples (like $2p, 3p, \ldots, (p-1)p$) have already been marked by smaller primes. For example, 10 will be marked when looping for $p = 2$, so when $p = 5$, there's no need to mark 10 again.

```
def sieve(n):
    is_prime = [True] * (n + 1)
    is_prime[0] = False
    is_prime[1] = False

    p = 2
    counter = 0
    while p * p <= n:
        print("outer loop for : ", p)
        if is_prime[p]:
                for multiple in range(p * p, n + 1, p):
                    print("setting flag for ", multiple, " Times " , counter)
                    is_prime[multiple] = False
                    counter += 1
        p += 1

    primes = []
    for i in range(2, n + 1):
        if is_prime[i]:
            primes.append(i)

    return primes

print(sieve(25))
```