GCD (Greatest Common Divisor)

gcd(a, b) is the greatest positive integer that divides both a and b.

eg:-

$36 = 2 \times 2 \times 3 \times 3 = 2^2 \times 3^2$

$60 = 2 \times 2 \times 3 \times 5 = 2^2 \times 3^1 \times 5^1$

Take lowest power of each common factor

$GCD = 2^2 \times 3^1 = 4 \times 3 = 12$

Properties

1. gcd(a,0) = |a|

Simply put, gcd of non-zero number with zero is always the non-zero number

↳ What are divisors of 0?
Every integer divides 0.

For example:

1 divides 0, because 0 = 1 * 0

2 divides 0, because 0 = 2 * 0

100 divides 0, because 0 = 100 * 0
So the set of divisors of 0 is: { ±1, ±2, ±3, … } (all integers).

↳ What are divisors of a ($\neq 0$)?

Just the usual finite set.
Example: divisors of 18 are { ±1, ±2, ±3, ±6, ±9, ±18 }.

↳ Common divisors of (0, a):
These are exactly the divisors of a, since every divisor of a also divides 0.

So the greatest common divisor is |a|.

And, also by convention gcd(0,0) = 0

2. gcd(a, b) = gcd(|a|, |b|) (signs don't matter)

3. If a divides b, then gcd(a, b) = |a|

4. gcd(a,b) = gcd(a-b, b), where a >= b

Think about common divisors

Suppose some integer dd divides both a and b.

Since $d|a$ and $d|b$, we can write:

$a=d\cdot m, \quad b=d\cdot n$

Then $a-b=d\cdot(m-n)$, so d also divides $(a - b)$.

So:
Any common divisor of (a, b) is also a common divisor of (a-b, b).

Euclidean Algorithm

We know:

gcd(a,b) = gcd(a−b,b)

So the idea here is simple we will keep subtracting the bigger number from smaller number till one it gets 0 and then the other number, b in this case is our answer

gcd(48,18)

(48,18) → (30,18) → (12,18) → (12,6) → (6,6) → (0,6) ⇒ 6

Downside: If a is much bigger than b, you'll subtract many times (slow).

Subtraction one step at a time is the same as subtracting b from a q times at once, where

$a = q\cdot b + r, \quad 0 \le r < b$ (Euclidean division)

Here r = a mod b.

Since subtracting b once keeps gcd unchanged, subtracting it q times also keeps gcd unchanged:

gcd(a,b) = gcd(a−qb, b) = gcd(r, b).

So we get the Euclidean step:

gcd(a,b) = gcd(b, amodb).

This single step can shrink the numbers a lot compared to one subtraction.

If a<b, then amodb = a. The identity gcd(a,b)=gcd(b,amodb) still holds;
it just means the next pair is (b,a)—a swap—and then reduction continues normally.

Implementations

1.Euclidean (modulo) — Iterative (preferred in CP)

```
long long gcd_euclid(long long a, long long b) {
    a = llabs(a); b = llabs(b);
    while (b != 0) {
        long long r = a % b; // compresses many subtractions at once
        a = b;
        b = r;
    }
    return a; // last non-zero = gcd
}
```

2.Euclidean (modulo) — Recursive (short & clean)

```
long long gcd_rec(long long a, long long b) {
    a = llabs(a); b = llabs(b);
    return (b == 0) ? a : gcd_rec(b, a % b);
}
```

```
Time: O(log min(a,b)) steps in the worst case (slowest for consecutive Fibonacci numbers).

Space:

    Iterative: O(1)

    Recursive: O(log min(a,b)) (call stack)
```

Least common multiple

Calculating the least common multiple (commonly denoted LCM) can be reduced to calculating the GCD with the following simple formula:

$$\mathrm{lcm}(a, b) = \frac{a \cdot b}{\gcd(a, b)}$$

Thus, LCM can be calculated using the Euclidean algorithm with the same time complexity:

A possible implementation, that cleverly avoids integer overflows by first dividing a with the GCD, is given here:

```
int lcm (int a, int b) {
    return a / gcd(a, b) * b;
}
```