Number Theory

By : Shivansh (CF : shiv_codegen)

Goals :

- Basic Primality Testing
- Sieve of Eratosthenes
- Prime Factorization
- Binary Exponentiation
- GCD/LCM and their properties

Basic Primality Testing

A primality test is an algorithm for determining whether a number is prime or composite.

There are many algorithms for primality testing such as:

- ✓ Brute-Force
- ✓ Square Root method
- ✓ Using Sieve (precomputation)

Brute Force Approach :

Check for every integer from 2 to n-1, if it divides n.

```
bool is_prime(int n)
{
    for (int i = 2; i < n; i++)
        if (n % i == 0) return false;
    return n > 1;
}
```

Some important observations :

- ✓ Factors always appear in pair.
- ✓ If number given *n* is not prime, then it must be possible to factor it into two values *A* and *B*, such that *A.B = n*.
- ✓ Hence with this we can see, either one of the two has to be less than root(n).
- ✓ Therefore, if the number is not a prime, it must have atleast one factor less than or equal to the square root of the number.

```
bool is_prime(int n)
{
    for (int i = 2; i*i <= n; i++)
        if (n % i == 0) return false;
    return n > 1;
}
```

Sieve of Eratosthenes

An algorithm for finding all the prime numbers in just O(n*loglogn) operations.

Main Idea :

- ✓ Don't check for all numbers, multiples of prime numbers are always composite.
- ✓ Mark multiples of prime numbers as composite.

Time complexity : n*log(log(n)) - Here

```
void sieve(int n)
{
        bool primes[n+1];
        fill(primes, primes+n+1, true);
        primes[0] = primes[1] = false;
        for (int i = 2; i*i <= n; i++) {
                if (primes[i])
                {
                        for (int j = i*i; j <= n; j += i)
                                primes[j] = false;
                }
        }
}
```

Prime Factorization

Prime Factorization is finding all the prime factors of a given number.

There are multiple ways to factor primes, such as:

✓ Trial Division
✓ Sieve method (precomputation)

Trial Division Method

Here, we will use the fact that the smallest divisor of any number N is prime, and is less than square root of N.

N can be represented as product of powers of primes.

```
vector<int> factor(int n)
{
    vector<int> facts;
    for (long long d = 2; d * d <= n; d++) {
        while (n % d == 0) {
            facts.push_back(d);
            n /= d;
        }
    }
    if (n > 1)
        facts.push_back(n);

    return facts;
}
```

Problem

You are given an integer n.

Check if n has an **odd** divisor, greater than one (does there exist such a number x (x>1)

that n is divisible by x and x is odd).

$2 \leq n \leq 10^{14}$ - Here

Problem

[Problem 1](#)

[Problem 2](#)

[Problem 3](#)

Binary Exponentiation

It is used to compute the value of A^B efficiently.

The naïve approach takes $O(B)$ whereas binary exponentiation has time complexity of $\mathrm{O}(\log_2 B)$.

Recursive Approach

```
int binary_expo(int a, int b)
{
      if (b == 0) return 1;
      int c = binary_expo(a, b/2);
      if (b % 2 == 0) return c*c;
      return c*c*a;
}
```

Iterative Approach

```
int binary_expo(int a, int b) {
   int c = 1;
   while (b) {
      if (b % 2) c *= a;
      a *= a;
      b /= 2;
   }
   return c;
}
```

Greatest Common Divisor

$GCD(a, b)$ is the greatest common divisor of *a* and *b*.

$LCM(a, b)$ is the least common multiple of *a* and *b*.

To calculate gcd efficiently, we use Euclidean Algorithm.

It states that $GCD(a, b) = GCD(b, a\%b)$.
when $b = 0$, the solution is a.

Time Complexity : O(log(min(a,b))) - [Proof]

```
int gcd_(int a, int b)
{
	if (b == 0) return a;
	return gcd(a, a%b);
}
```

Least Common Divisor

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

Properties

- ✓ $\gcd(a, b)$ can be represented as product of $\min({p_i}^{a_i}, {p_i}^{b_i})$ for each prime factor.
- ✓ $lcm(a, b)$ can be represented as product of $max({p_i}^{a_i}, {p_i}^{b_i})$ for each prime factor.
- ✓ $\gcd(a, b, c, \dots)$ is same as $\gcd(\gcd(\gcd(a, b), c), \dots)$.
- ✓ $lcm(a, b, c, \dots)$is same as $\mathrm{lcm}(\mathrm{lcm}(\mathrm{lcm}(a, b), c), \dots)$.
- ✓ $\gcd(a, a + 1) = 1$.

Problem : [Here]