Number Theory

- Srivaths P

Goal:

To learn:

- Basic Primality Testing
- Sieve of Eratosthenes
- Prime Factorization
- Binary Exponentiation
- Basic Modular Arithmetic

Primality Testing

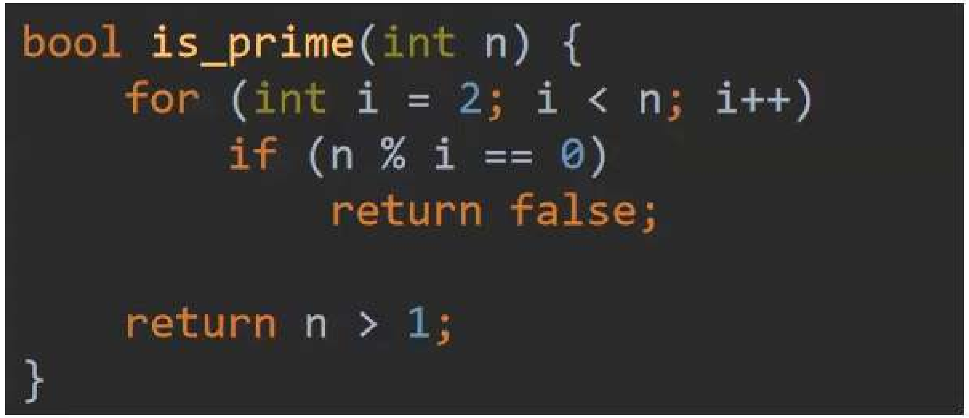
A primality test is an algorithm for determining whether a number is prime or composite.

There are many algorithms for primality testing such as:

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

Primality Testing – Code

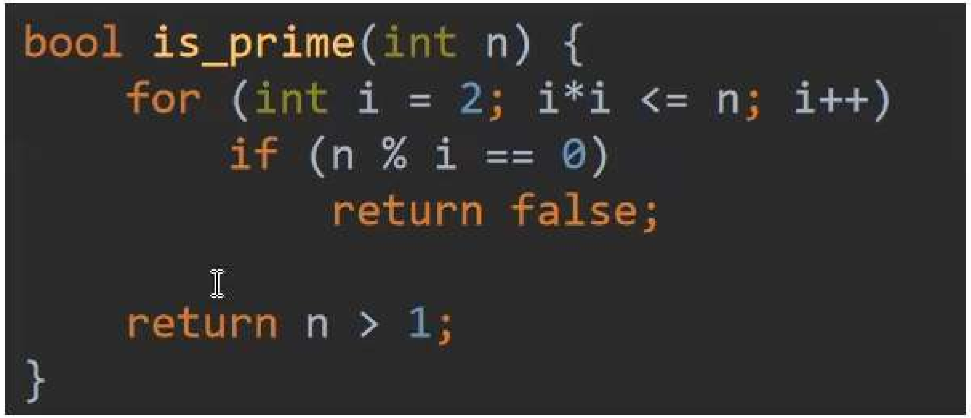
Brute-Force:



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

    return n > 1;
}
```

Sqrt method:



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

    return n > 1;
}
```

Square Root Method Proof

If the number N given is not a prime, then it must be possible to factor into two values A and B such that $AB = N$.

Here, it is impossible for both A and B to be greater than $\sqrt{N}$, as in that case the product will be greater than 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.

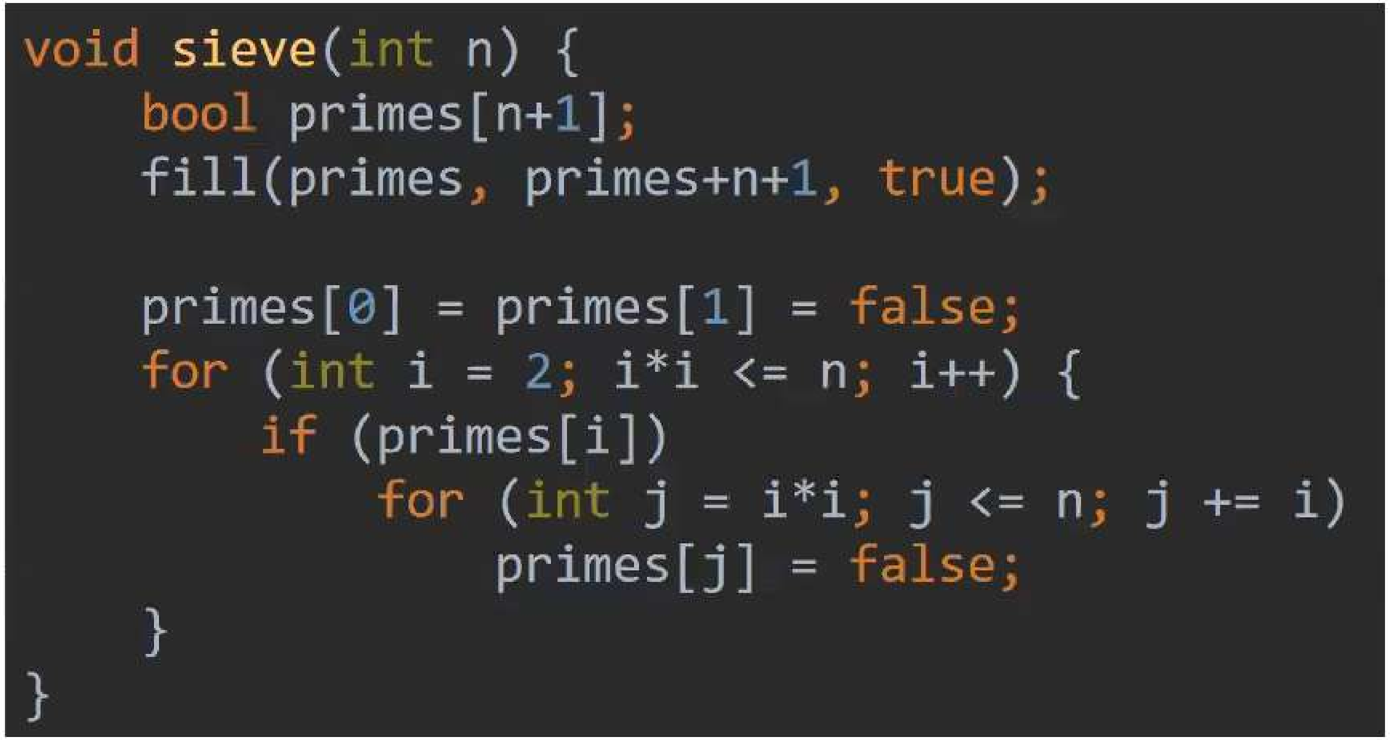
Square Root Method Proof

If the number N given is not a prime, then it must be possible to factor into two values A and B such that $AB = N$.

Here, it is impossible for both A and B to be greater than $\sqrt{N}$, as in that case the product will be greater than 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.

Sieve of Eratosthenes – Code



```
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;
    }
}
```

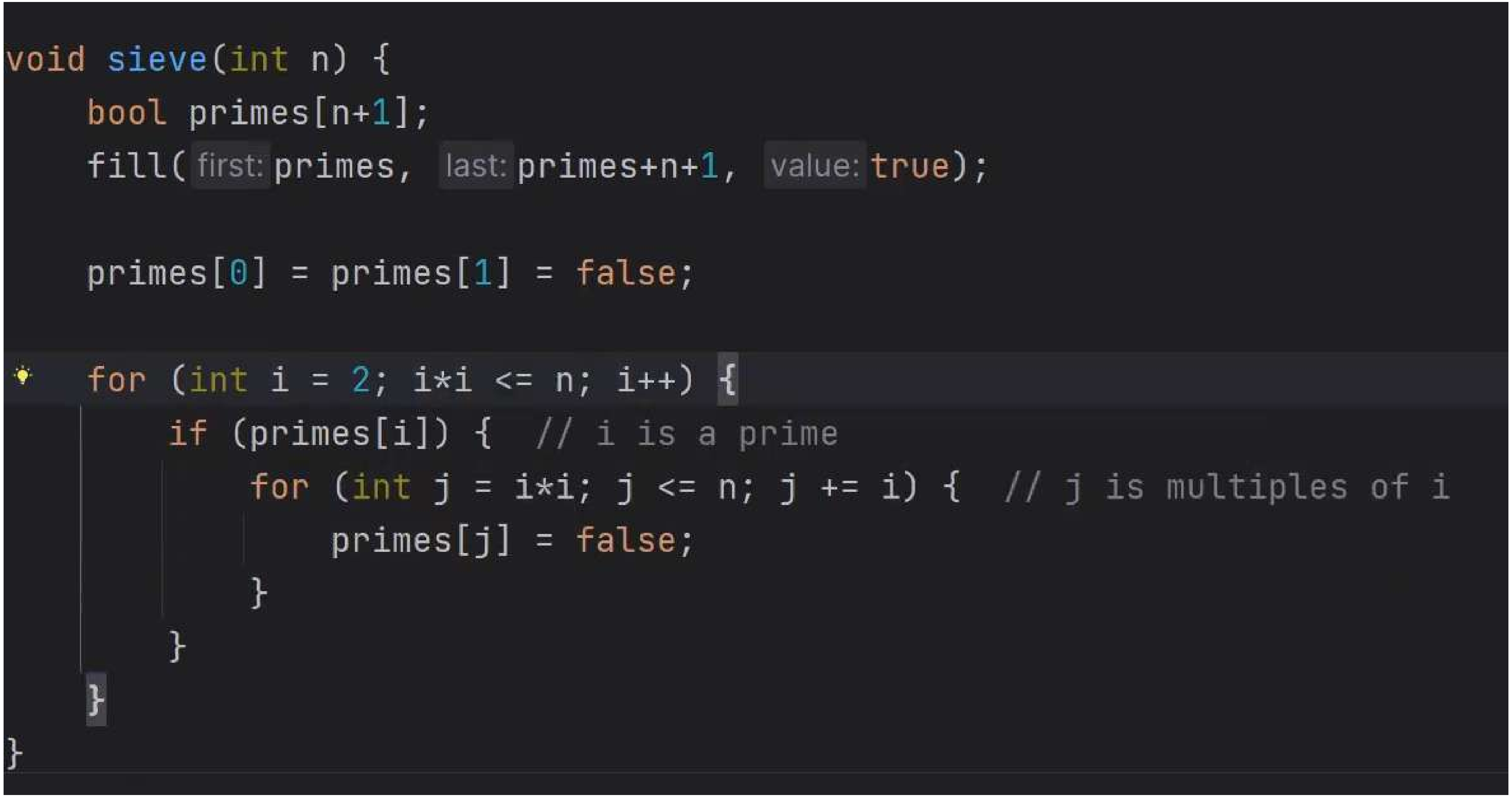
Sieve of Eratosthenes – Time Complexity

The Prime Harmonic Series is the infinite sum of the reciprocals of all prime numbers. https://bit.ly/3uTfpr0

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \dots$$

The above sum comes out as $\log \log N$ when the primes are less than N.

The sieve takes approximately $2N$ iterations for $N = 10^7$.



```
void sieve(int n) {
    bool primes[n+1];
    fill( first: primes,  last: primes+n+1,  value: true);

    primes[0] = primes[1] = false;

    for (int i = 2; i*i <= n; i++) {
        if (primes[i]) {  // i is a prime
            for (int j = i*i; j <= n; j += i) {  // j is multiples of 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
- Wheel Factorization
- Sieve method (precomputation)

Prime Factorization – Trial Division

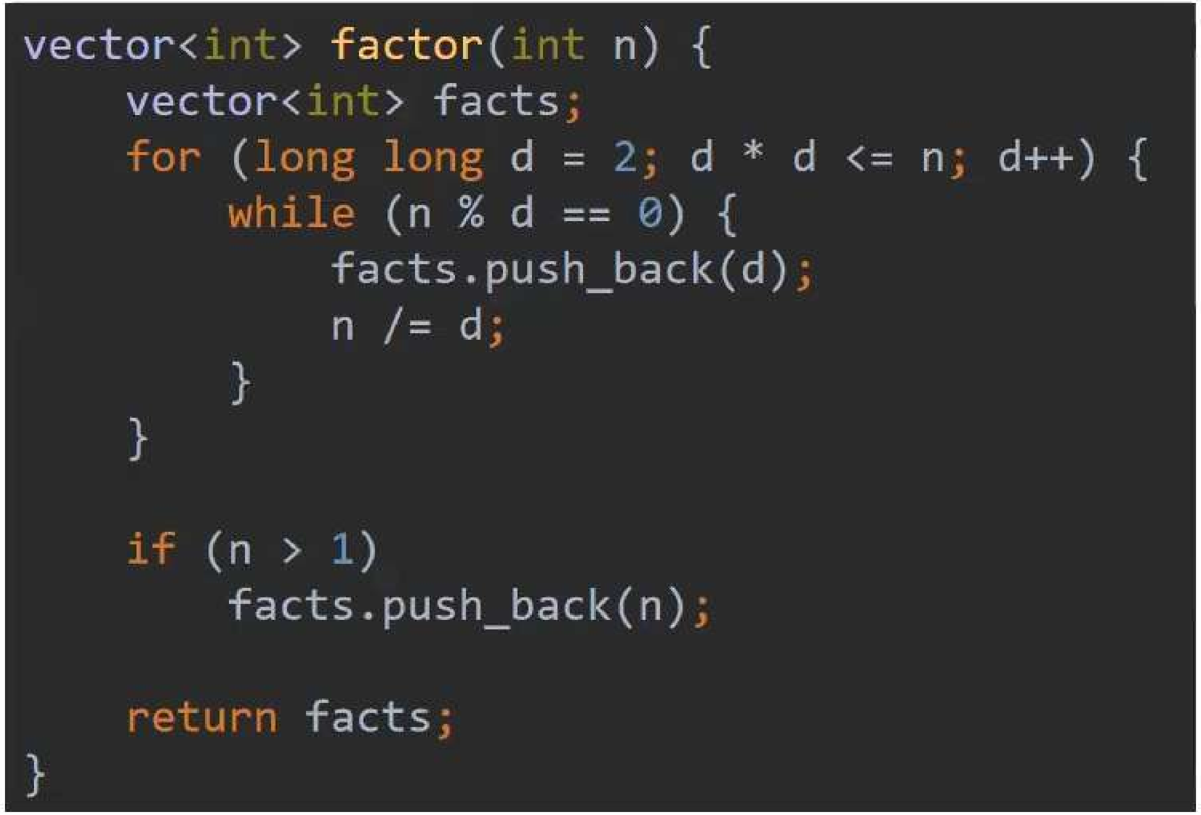
Trial division is the most basic method of prime factorization.

We will use the fact that the smallest divisor of any number N is prime, and it will be less than $\sqrt{N}$.

N can be represented as $p_0^{q_0} * p_1^{q_1} * p_2^{q_2} * \ldots * p_k^{q_k}$ where p_i is prime. Let us assume that the prime array is sorted.

We will iterate through the range $[2, \sqrt{N}]$ to find p_0 first, then p_1, etc.

Prime Factorization – Trial Division Code



```cpp
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;
}
```

Modular Arithmetic

Modular Arithmetic is arithmetic operations involving taking the modulo with some modulus. It is usually given in problem statements so that the solution doesn’t overflow in case the answer is huge.

A few operations involving modulo are:

- Addition
- Subtraction
- Mutliplication
- Division

Modular Arithmetic

Modular addition:

$(A + B) \,\%\, M = (A \,\%\, M + B \,\%\, M) \,\%\, M.$

Modular subtraction (be very careful when using C++):

$(A - B) \,\%\, M = (A \,\%\, M - B \,\%\, M) \,\%\, M.$

$(A - B) \,\%\, M = (((A \,\%\, M - B \,\%\, M) \,\%\, M) + M) \,\%\, M.$ (in C++)

Modular multiplication:

$(A * B) \,\%\, M = ((A \,\%\, M) * (B \,\%\, M)) \,\%\, M.$

Binary Exponentiation

The idea of binary exponentiation is as follows:

When B is even: $A^B = A^{\frac{B}{2}} \times A^{\frac{B}{2}}$.
When B is odd: $A^B = A^{\frac{B}{2}} \times A^{\frac{B}{2}} \times A$.
(Assuming division is floored)

We can do the above using a recursive function (or iteratively).

Problems

- https://codeforces.com/problemset/problem/472/A
- https://codeforces.com/problemset/problem/1475/A
- https://codeforces.com/problemset/problem/1679/A
- https://codeforces.com/problemset/problem/1165/D

Resources

- https://en.wikipedia.org/wiki/Primality_test
- https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes
- https://cp-algorithms.com/algebra/factorization.html (advanced)
- https://bit.ly/3cl0Vdj (modular arithmetic basics)