LEVEL 2

WEEK #4

SEASON 2025

HELWAN ICPC
COMMUNITY

Overview

- NUMBER THEORY?
- MODULAR ARITHMENTIC
- SIEVE OF ERATOSTHENES
- PRIME FACTORIZATION
- GCD
- LCM
- ADDITIONAL RESOURCES

What is Number Theory?

Number theory is a branch of pure mathematics that deals with the properties and relationships of numbers, particularly integers.

Number theory often involves solving problems related to properties of integers, such as divisibility and prime numbers

Any operation or analysis done between or on integers is considered as a Number Theory.

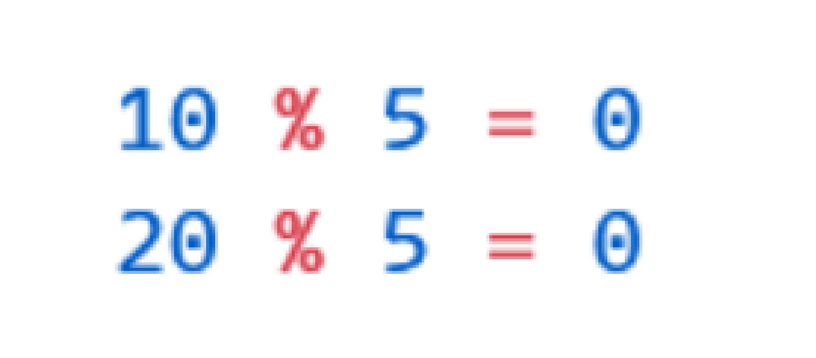
Modular Arithmetic

Modular arithmetic is the branch of arithmetic mathematics related with the “mod” functionality.

In some problems, you are asked to print the answer modulo some integer number n, for example : 1e9 + 7.

Modular arithmetic is used to map the answer into a specific range that lies between 0 and n-1.

This results in congruence between some numbers, that are mapped to the same number.

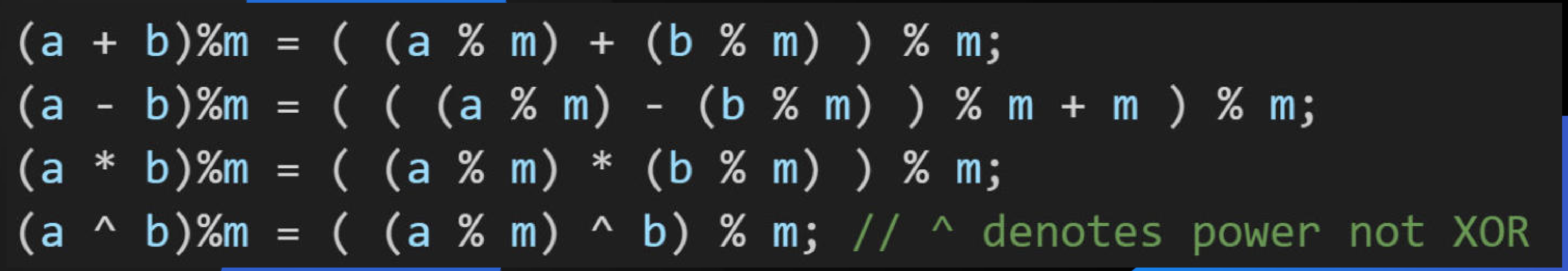


```
10 % 5 = 0
20 % 5 = 0
```

In the previous example, assuming that n = 5. In the domain of 5, 0, 5, 10, 15, 20, 25, … are all congruent and return the same answer (0).

Modular Arithmetic

Based on the previous, some helpful operations came to hand.
Assuming that in **m** domain, **a ≅ a%m.**



```
(a + b)%m = ( (a % m) + (b % m) ) % m;
(a - b)%m = ( ( (a % m) - (b % m) ) % m + m ) % m;
(a * b)%m = ( (a % m) * (b % m) ) % m;
(a ^ b)%m = ( (a % m) ^ b) % m; // ^ denotes power not XOR
```

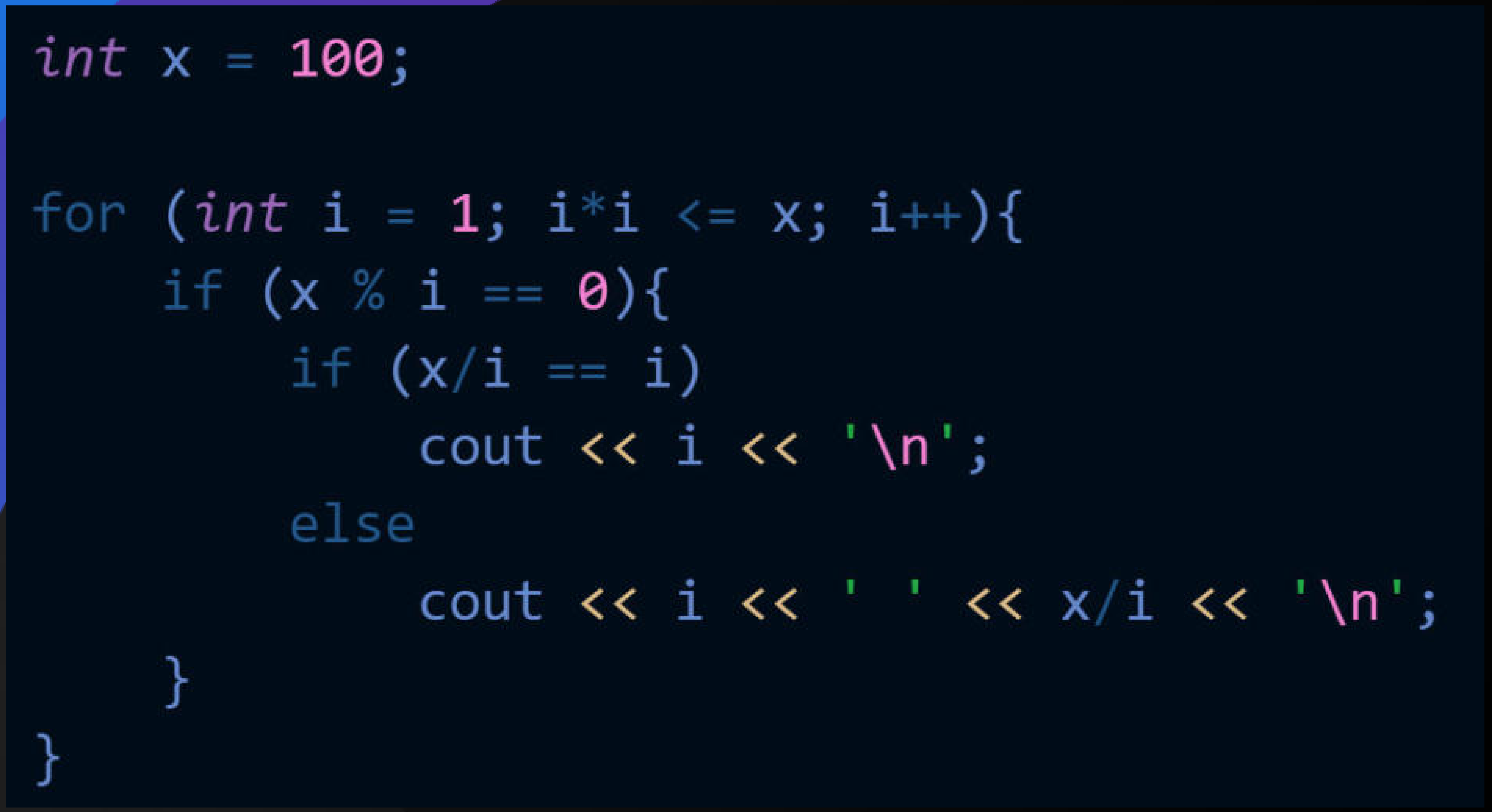
In the shown operations, instead of operating on large numbers, modular arithmetic can be used to reduce the values and keep the right answer under a certain domain m.

Division operation doesn’t always work.

Divisors

Integer x is a divisor of y, if x can divide y without any reminders. The Notation of such relation is **(x | y)**.

To find all divisors of y, we can make a simple loop from 1 to sqrt(y).



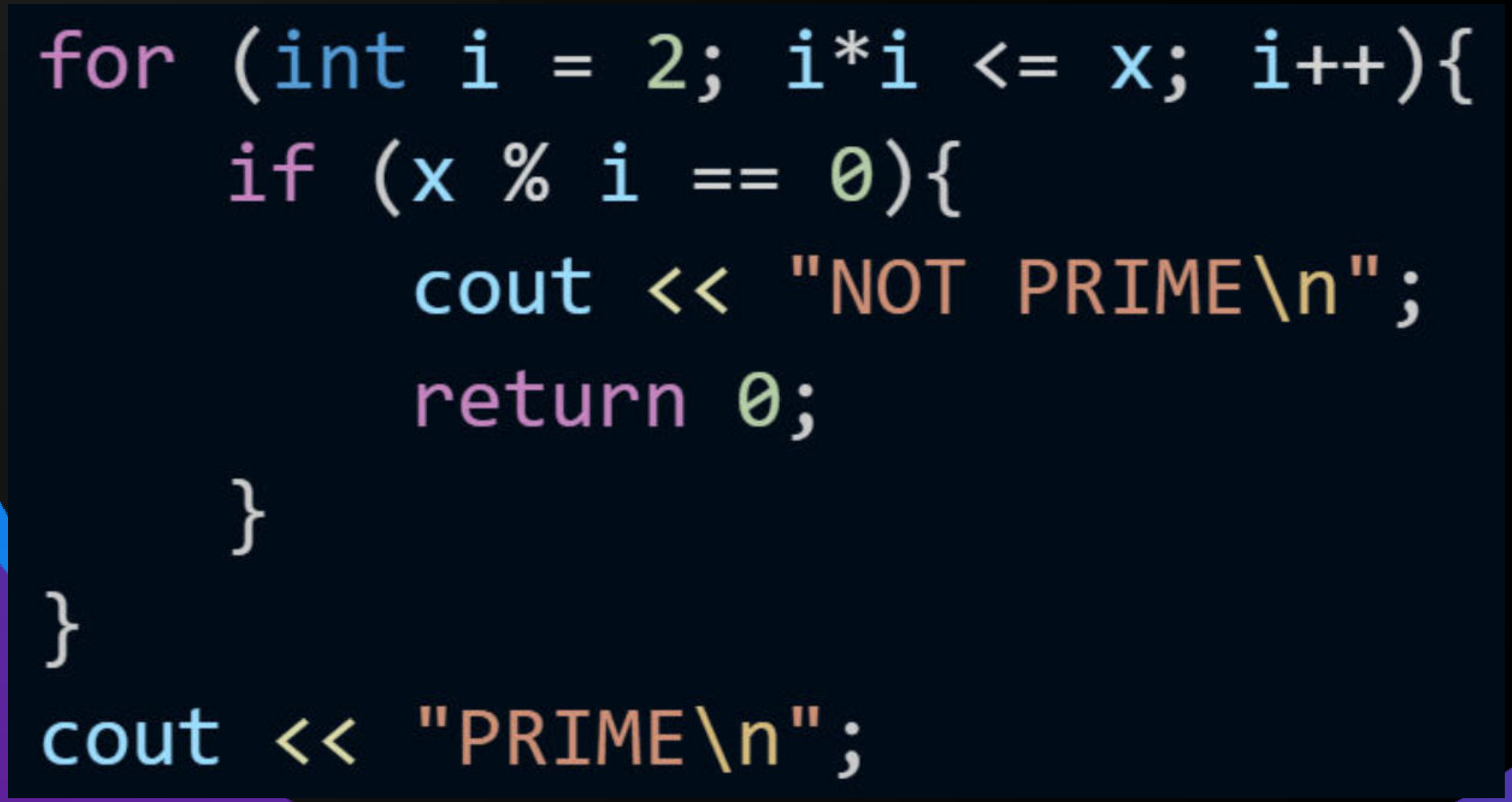
```
int x = 100;

for (int i = 1; i*i <= x; i++){
    if (x % i == 0){
        if (x/i == i)
            cout << i << '\n';
        else
            cout << i << ' ' << x/i << '\n';
    }
}
```

Primes

A prime number is a number where it’s only divisors are 1 and itself.

To determine if x is a prime, make a simple loop from 2 to the sqrt(x).



```
for (int i = 2; i*i <= x; i++){
    if (x % i == 0){
        cout << "NOT PRIME\n";
        return 0;
    }
}
cout << "PRIME\n";
```

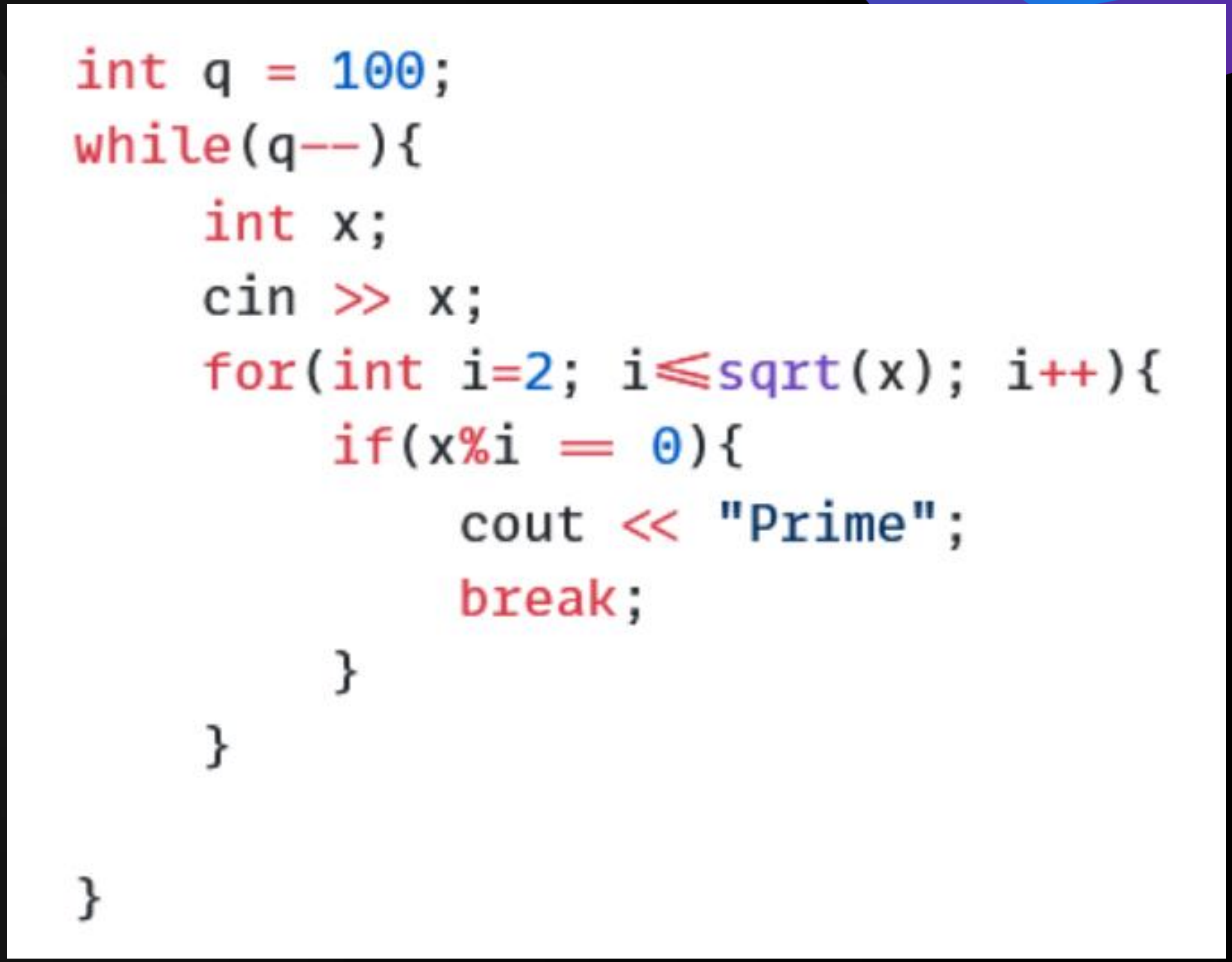
WHY SIEVE?

Motivation

Assume you are given q queries, where in each, an integer n is given. Determine whether n is prime or not.

A simple solution is to iterate for each n, from 1 to sqrt(n).
Such solution has complexity of O(q*sqrt(n)) which is efficient.

Now assume that n can reach 10^7 or higher, such algorithm wont be efficient.

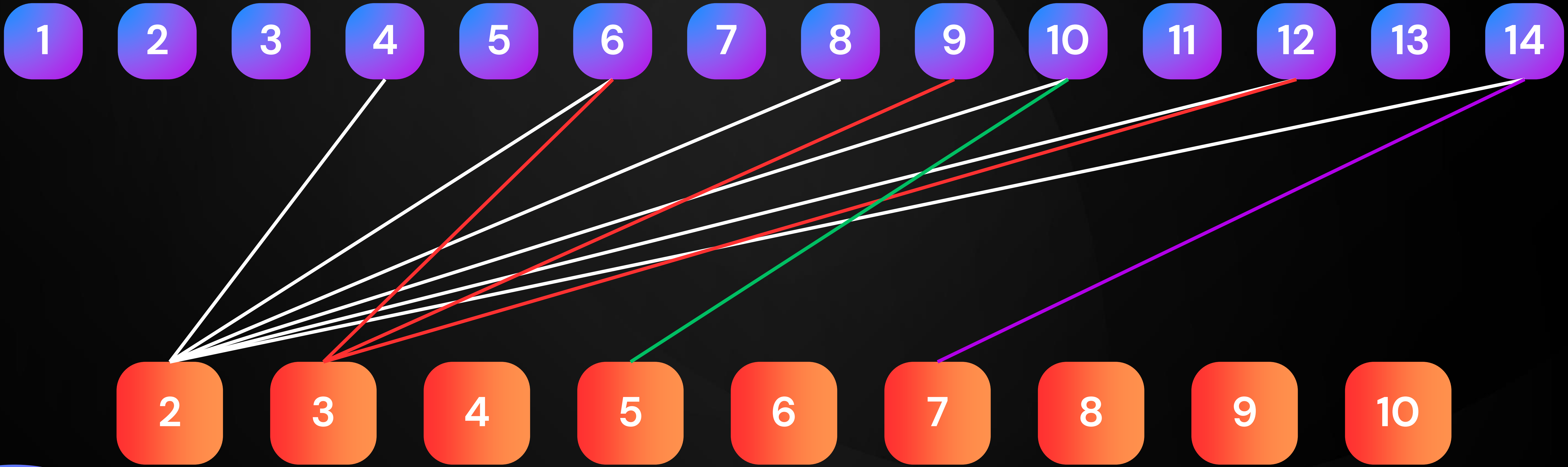


```
int q = 100;
while(q--){
    int x;
    cin >> x;
    for(int i=2; i<=sqrt(x); i++){
        if(x%i == 0){
            cout << "Prime";
            break;
        }
    }

}
```

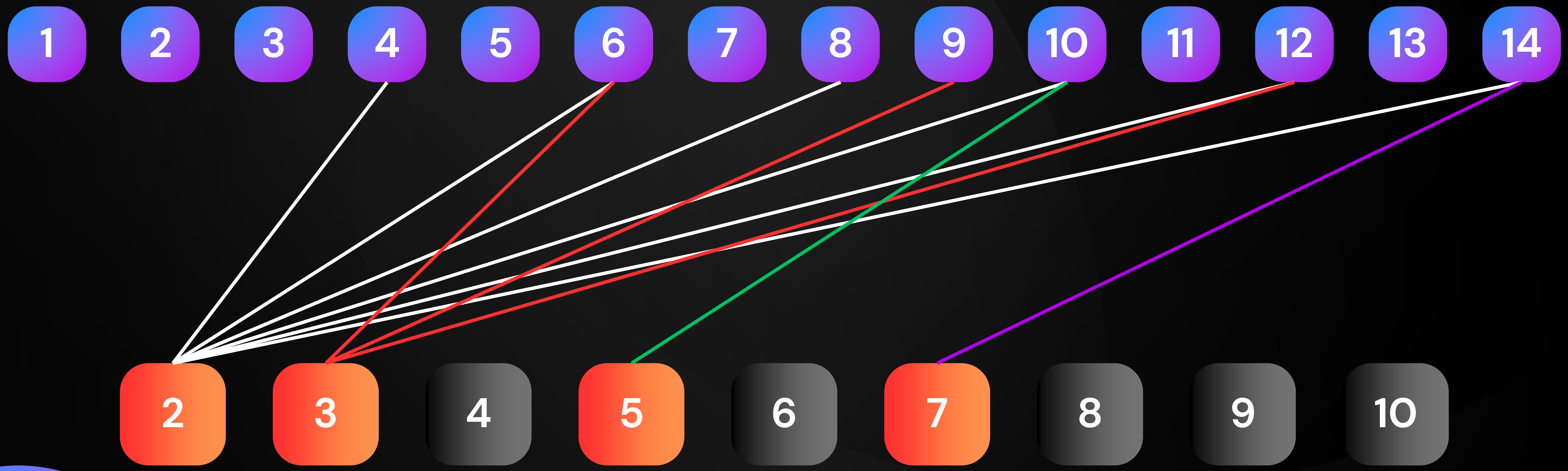
Sieve of Eratosthenes

The idea of sieve is instead of checking the divisors for each n, check the multiple of each number within the range.



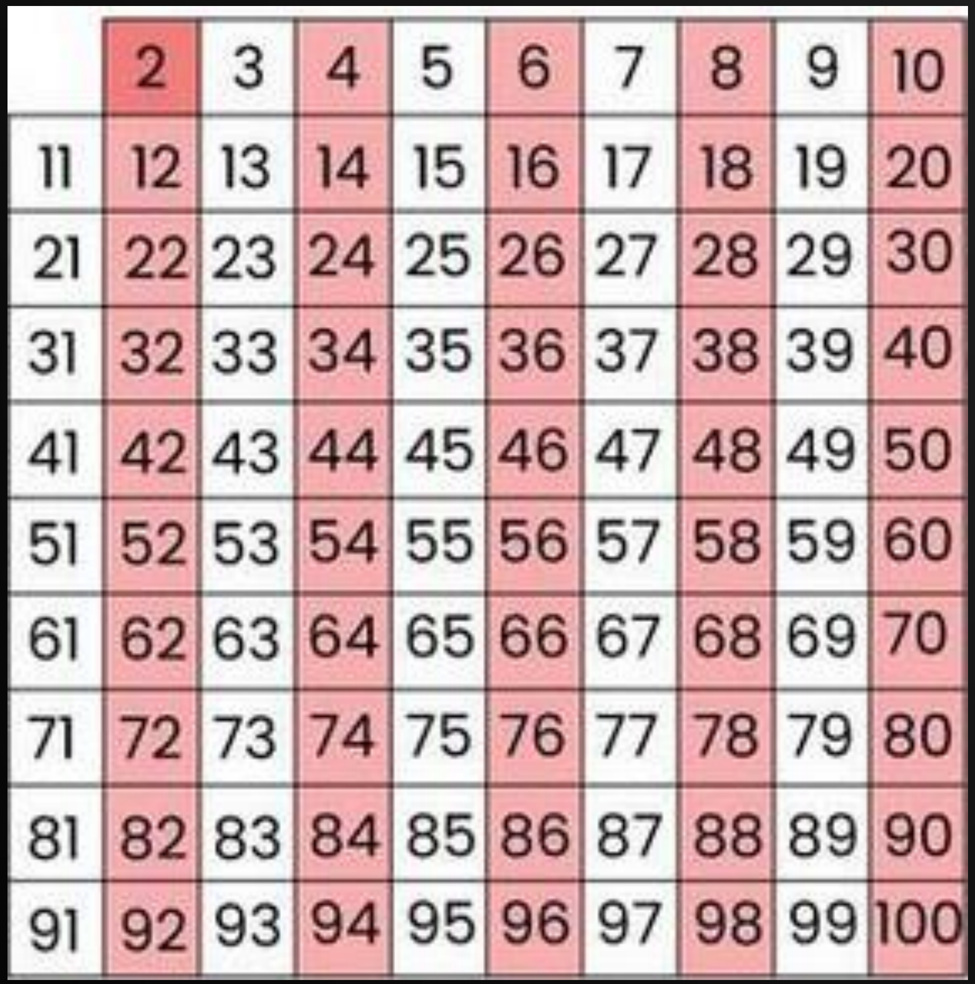
Sieve of Eratosthenes

The idea of sieve is instead of checking the divisors for each n, check the multiple of each number within the range.

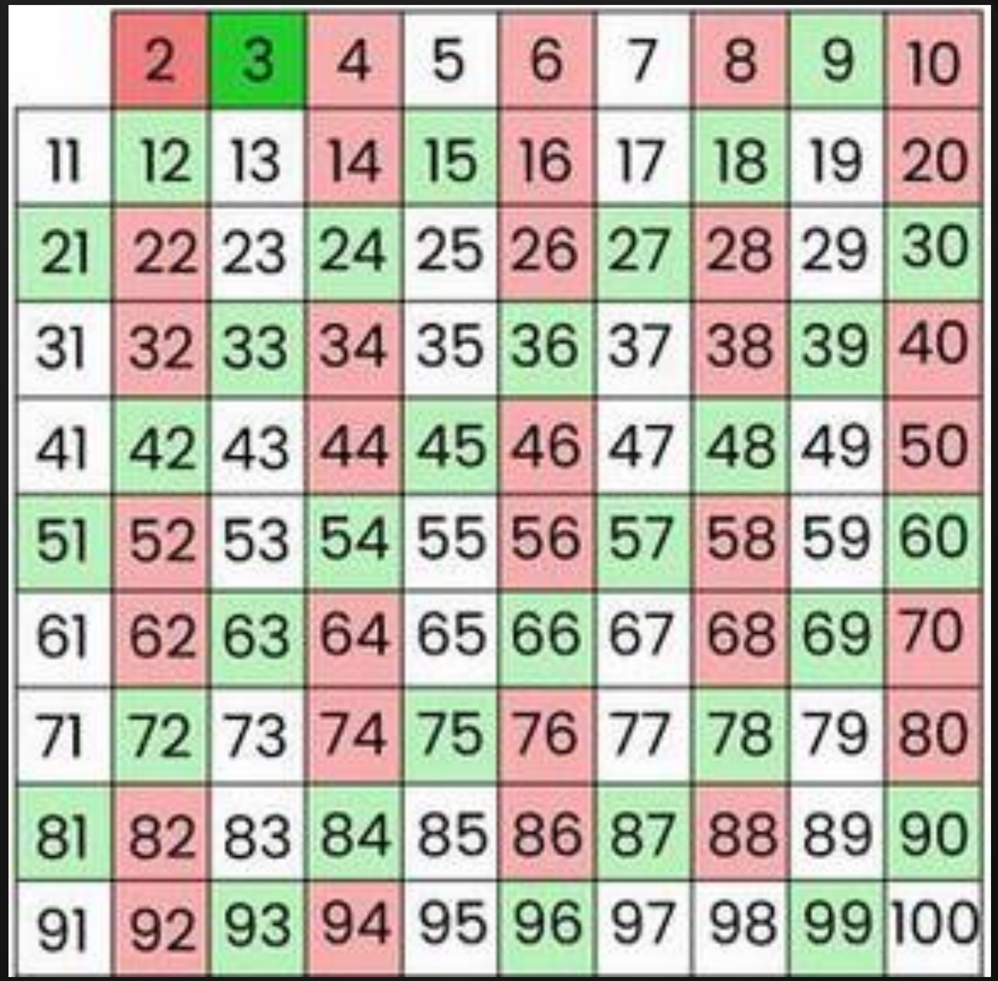


Sieve of Eratosthenes

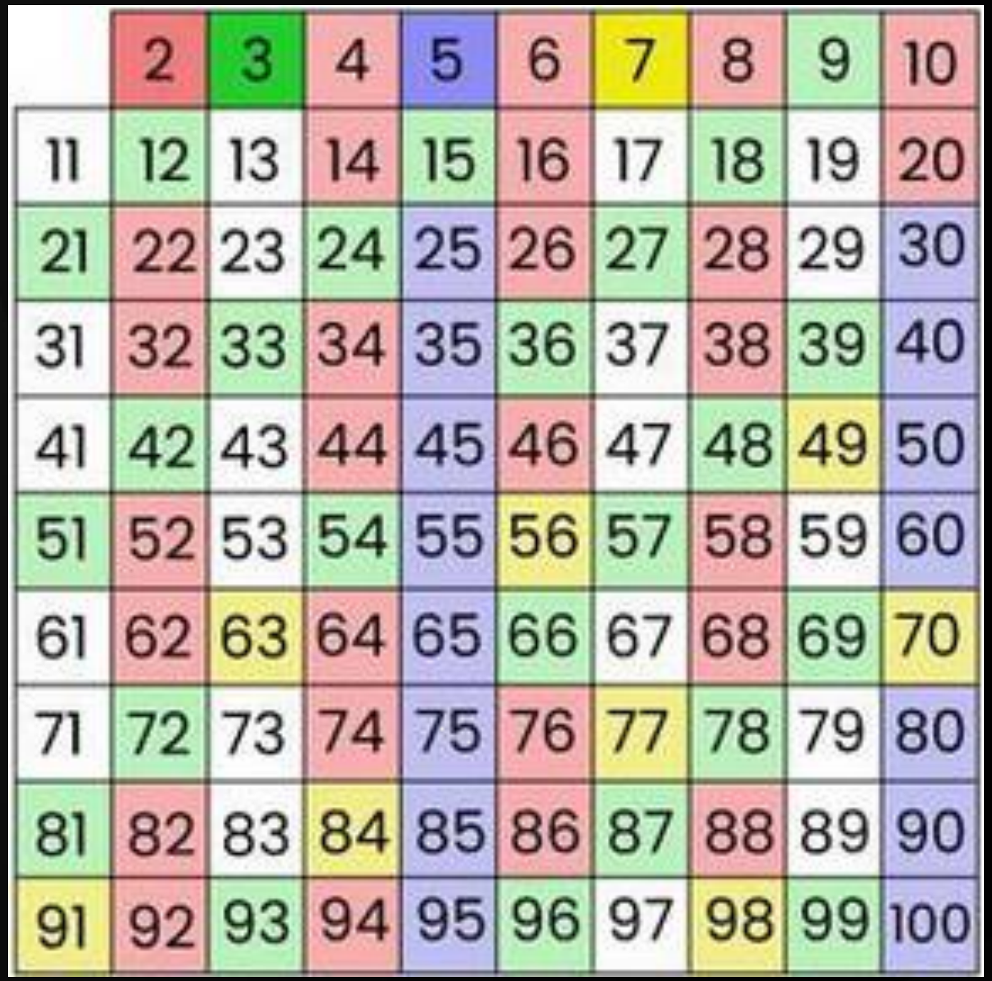
Complexity of such algorithm is O(n*log(log n)).



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

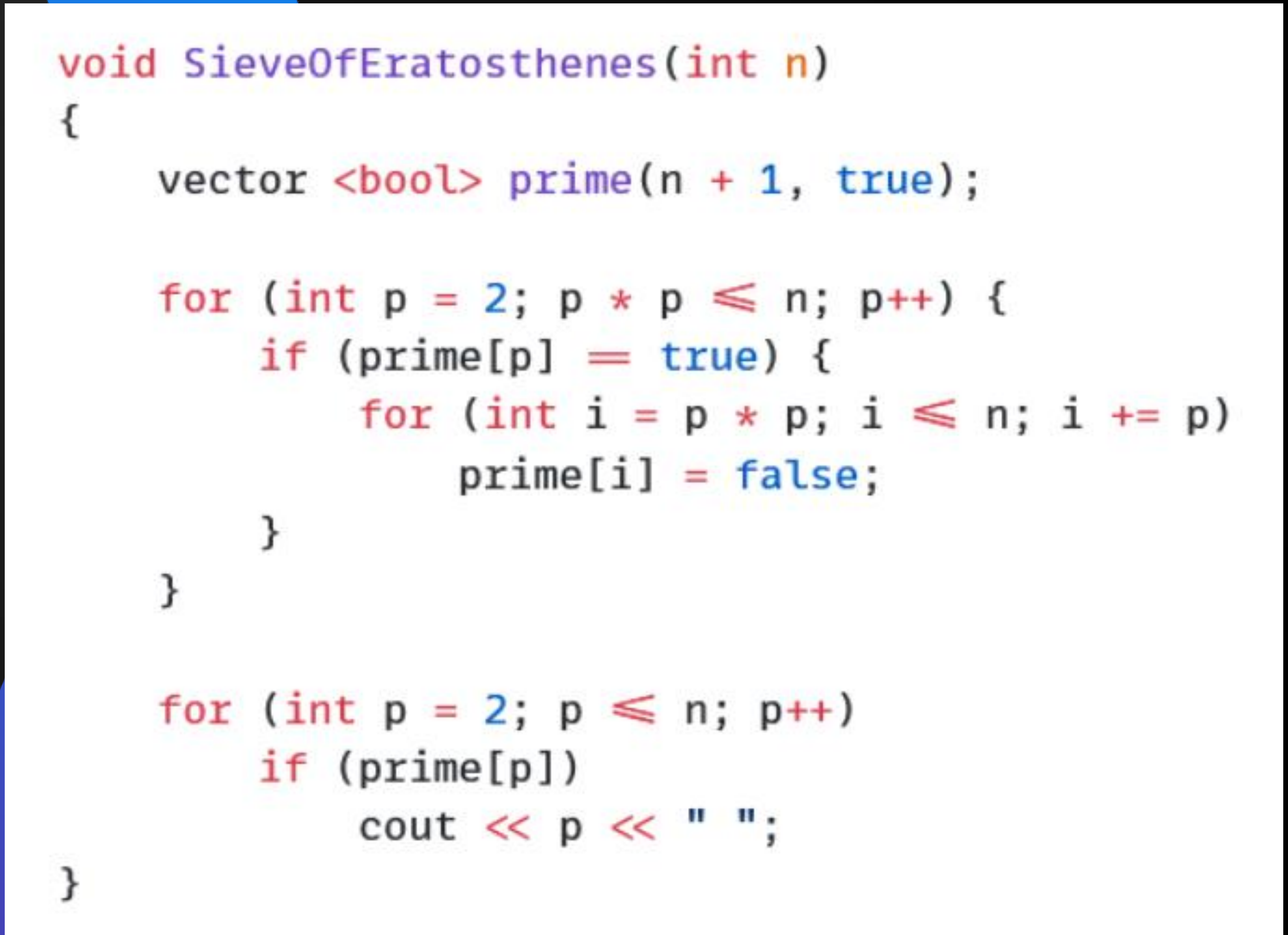
Sieve of Eratosthenes

Complexity of such algorithm is O(n*log(log n)).

Create a boolean array "prime[0..n]" and initialize all entries it as true. A value in prime[i] will finally be false if i is Not a prime, else true.

If prime[p] is not changed, then it is a prime

Update all multiples of p greater than or equal to the square of it numbers which are multiple of p and are less than p^2 are already been marked.



```
void SieveOfEratosthenes(int n)
{
    vector <bool> prime(n + 1, true);

    for (int p = 2; p * p <= n; p++) {
        if (prime[p] == true) {
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }

    for (int p = 2; p <= n; p++)
        if (prime[p])
            cout << p << " ";
}
```

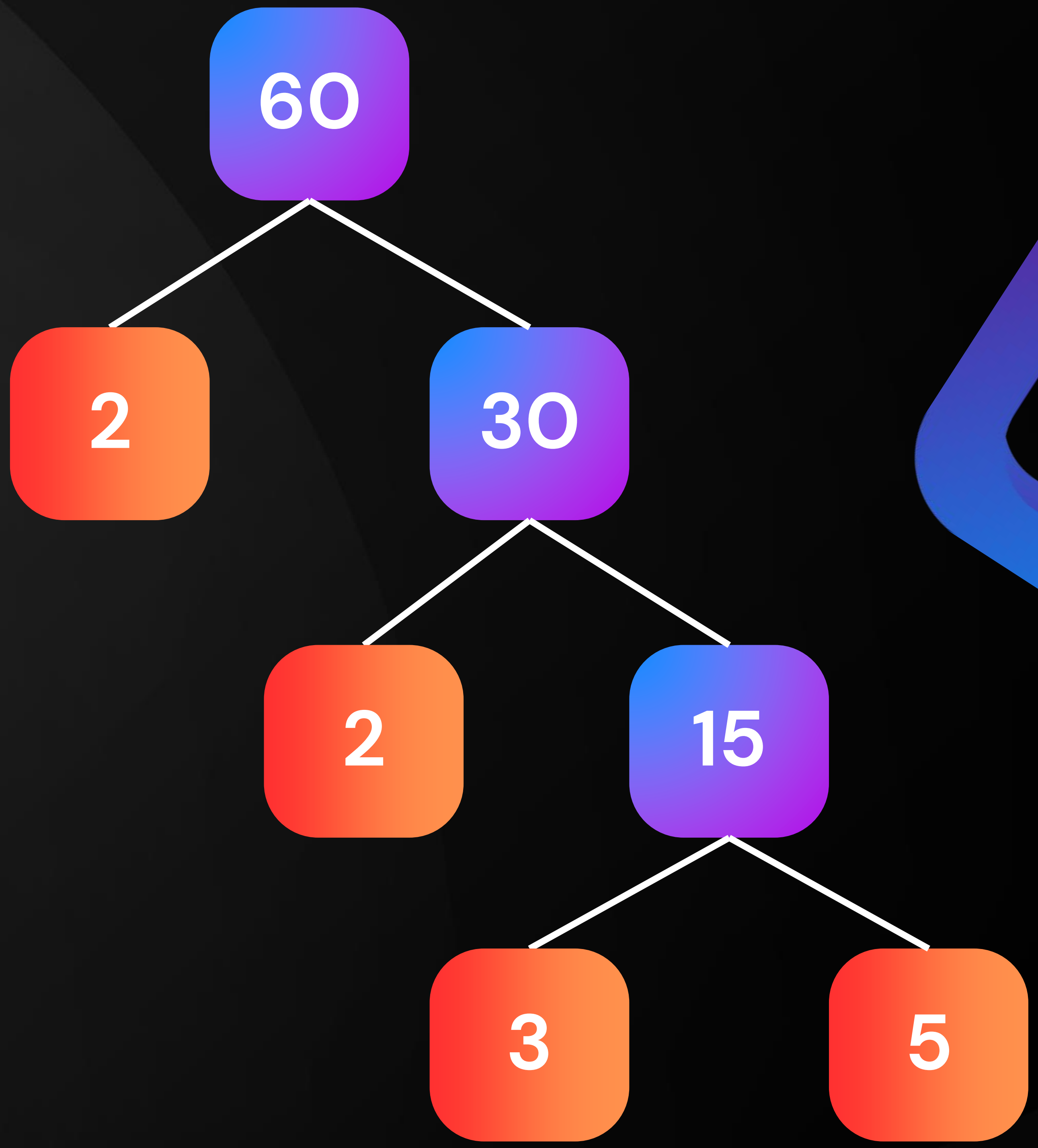
LET’S DO SOME CODE

Prime Factorization

Based on Sieve, we can say for sure that any number can be represented as product of some prime numbers.

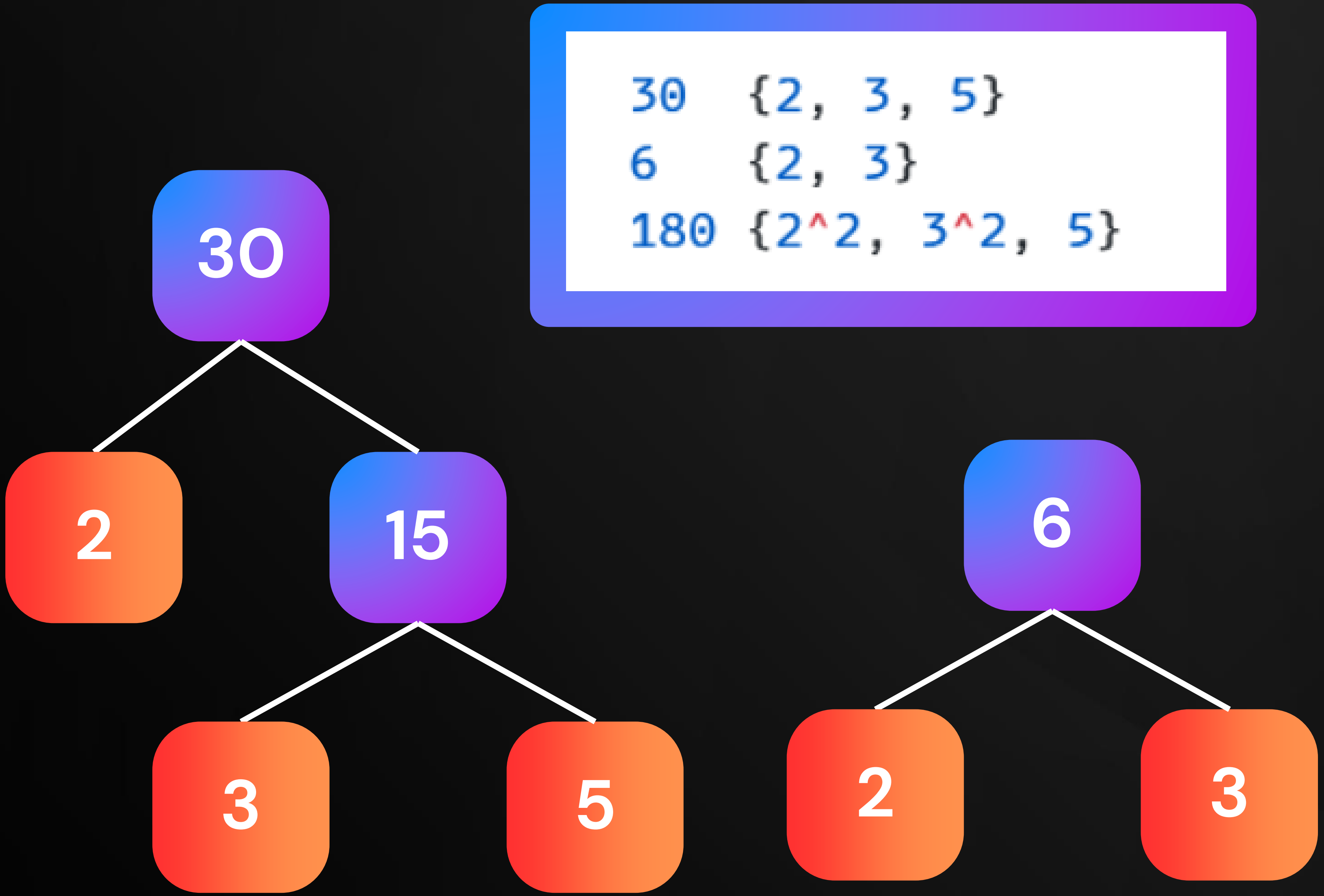
There is only one (unique!) set of prime factors for any number

In order to maintain this property of unique prime factorization, it is necessary that the number one, 1, be categorized as non prime.

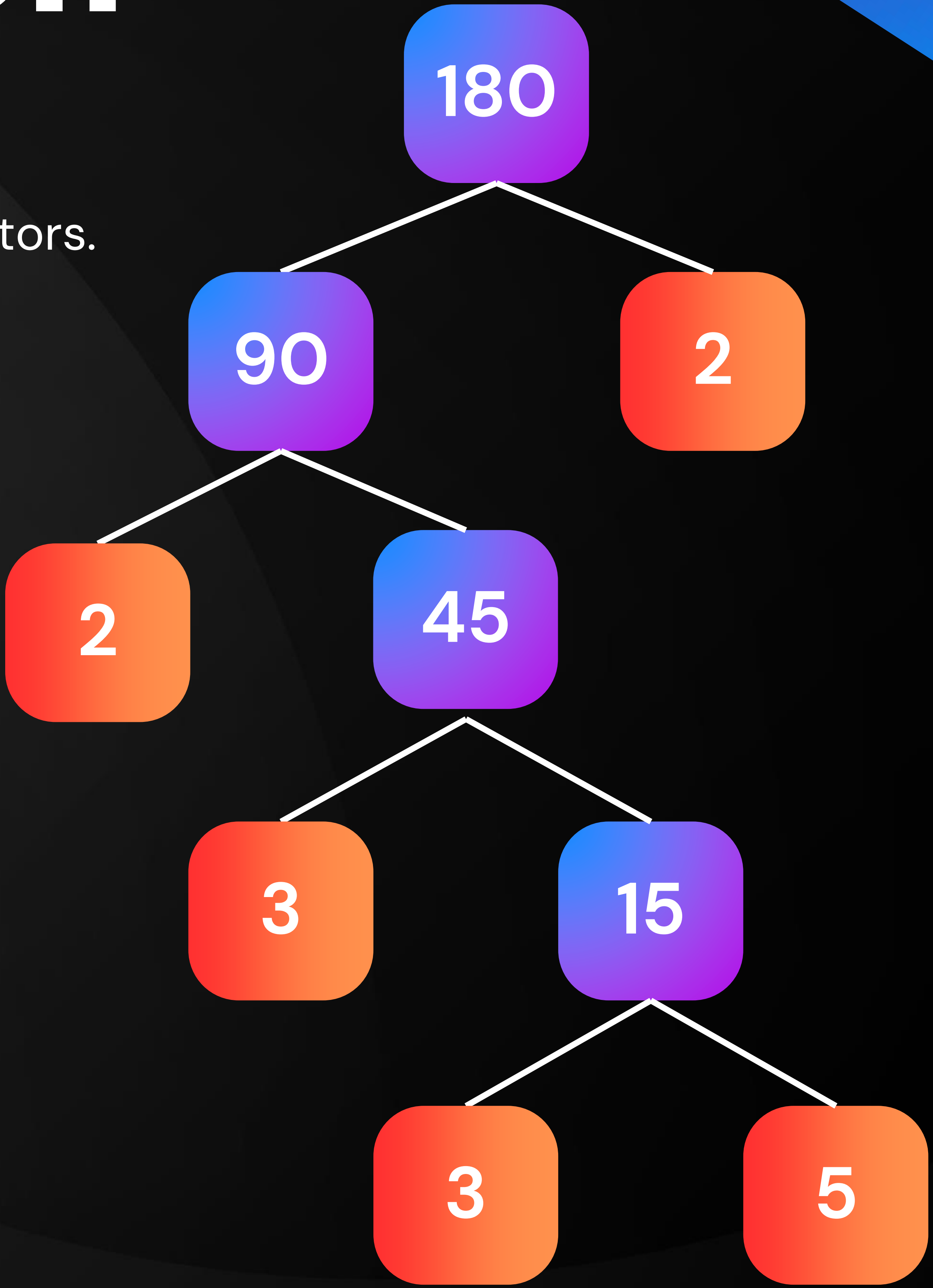


Prime Factorization

We can get the product of elements by multiplying their prime factors.



```
30  {2, 3, 5}
6   {2, 3}
180 {2^2, 3^2, 5}
```



Prime Factorization

Divisibility can be determined using the prime factors.

30 ~~2~~ ~~3~~ 5

6 ~~2~~ ~~3~~

30 ~~2~~ ~~3~~ 5

12 ~~2~~ 2 ~~3~~

3o is divisible by 6 and the result is 5

3o is not divisible by 12 and the result is 5/2

Prime Factorization

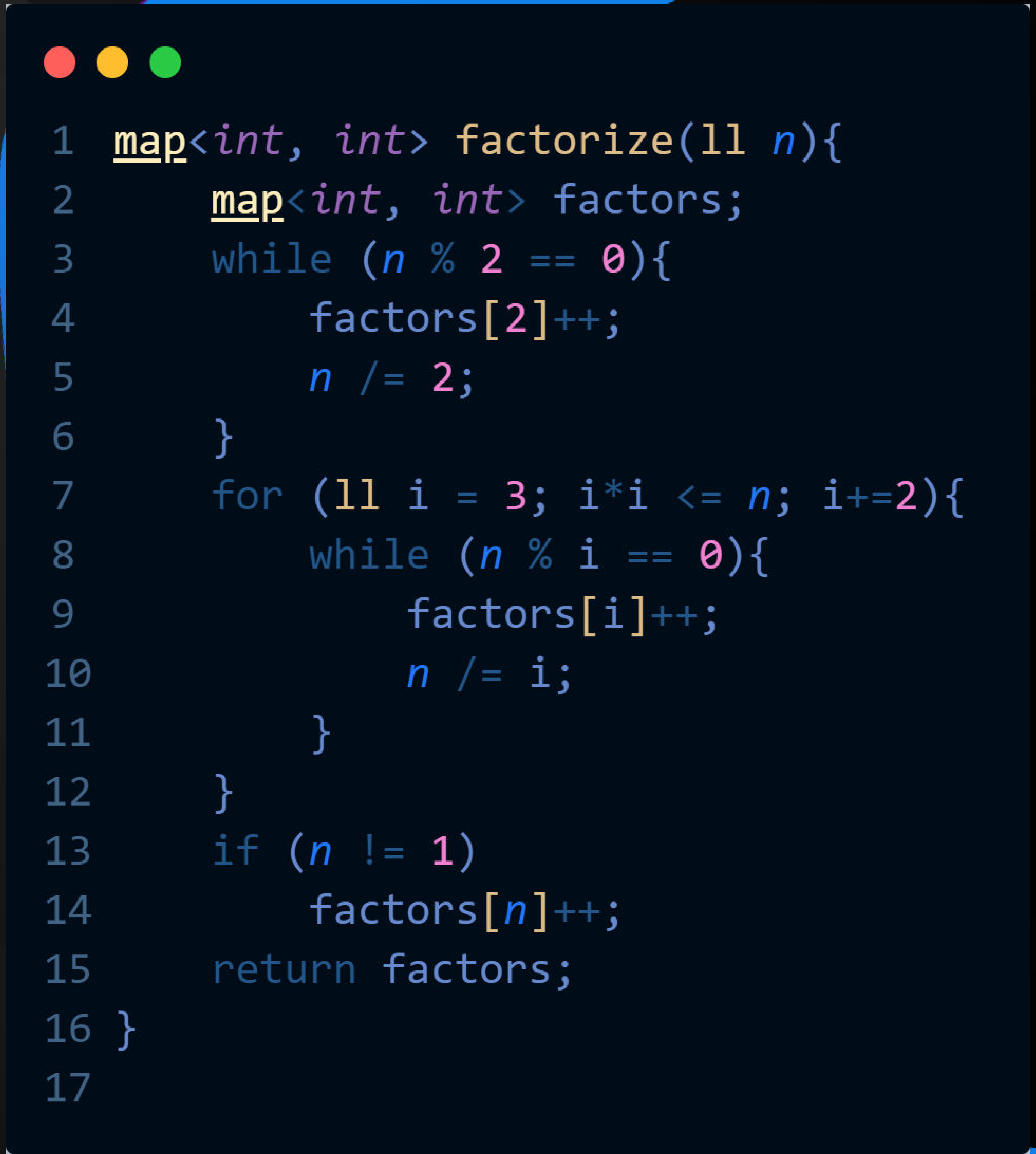
The following code shows how the prime factorization does work.

Start by dividing the number by 2s .

The number n must be odd at this point. So we can skip one element (Note i = i + 2)

While i divides n, print i and divide n .

The last condition is to handle the case when n is a prime number greater than 2.



```cpp
map<int, int> factorize(ll n){
    map<int, int> factors;
    while (n % 2 == 0){
        factors[2]++;
        n /= 2;
    }
    for (ll i = 3; i*i <= n; i+=2){
        while (n % i == 0){
            factors[i]++;
            n /= i;
        }
    }
    if (n != 1)
        factors[n]++;
    return factors;
}

```

LET'S DO SOME CODE

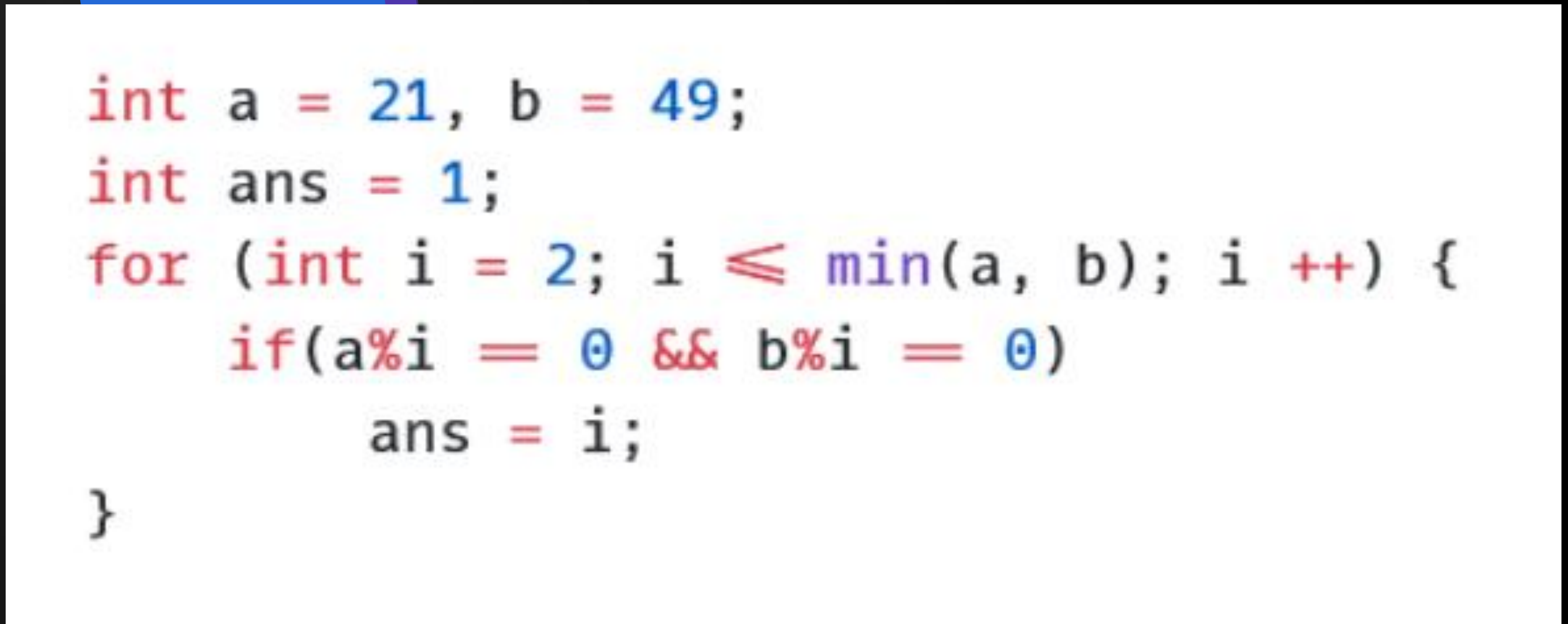
GCD

GCD stands for greatest common divisor.

Given two numbers, a and b, what can be done to find gcd(a, b), the greatest integer g, where g | a and g | b.

Such a problem can be solved using a simple loop from 1 to min(a, b).

Such solution takes O(min(a, b)).



```
int a = 21, b = 49;
int ans = 1;
for (int i = 2; i <= min(a, b); i ++) {
    if(a%i == 0 && b%i == 0)
        ans = i;
}
```

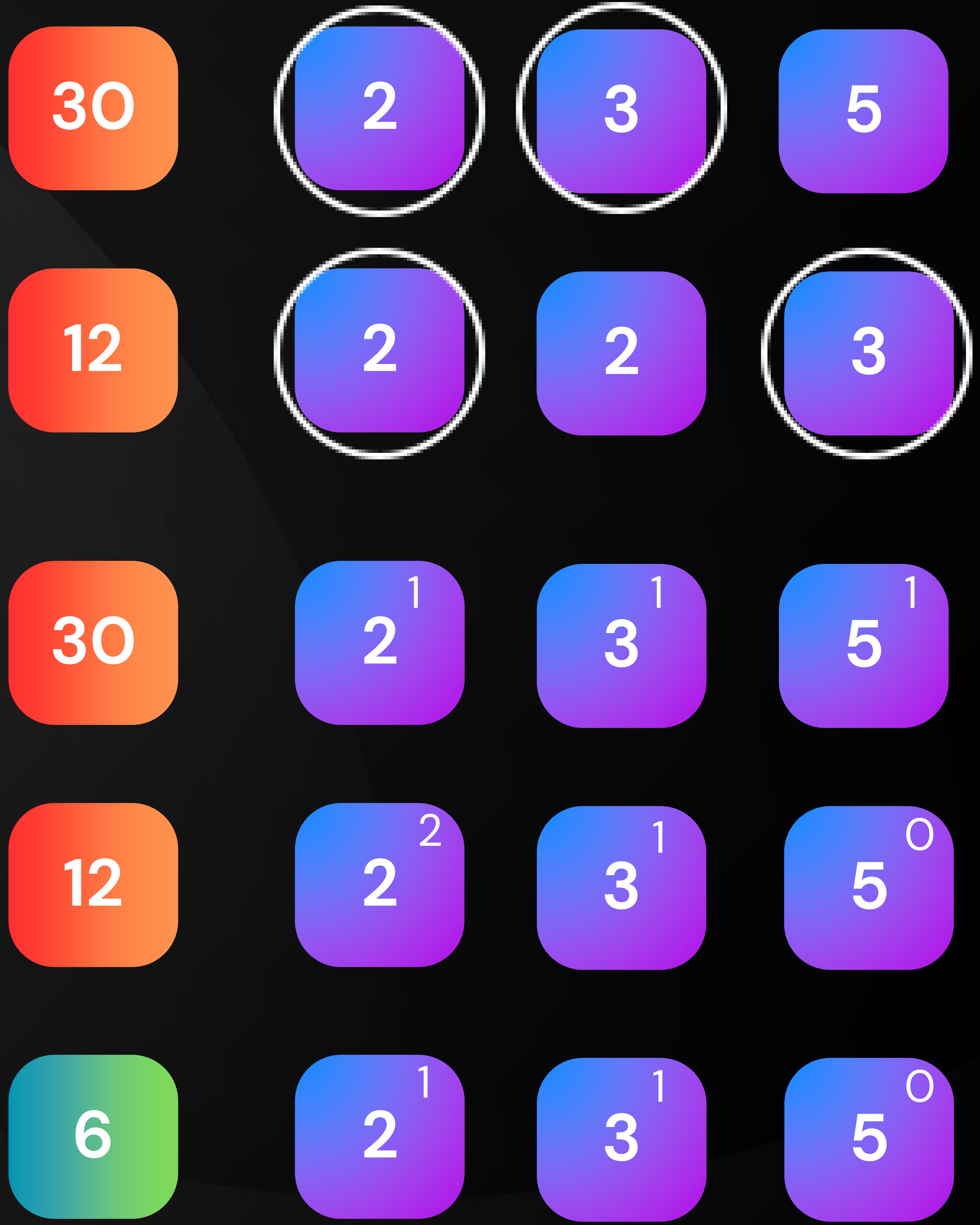
GCD

Another way of getting the GCD is by using prime factors.

GCD can be calculated by choosing the common factors from the both numbers' factorization.

By other words, choosing the min power for each factor in both factorizations.

Such solution takes O(sqrt(max(a, b)).



30 2 3 5

12 2 2 3

30 2^1 3^1 5^1

12 2^2 3^1 5^0

6 2^1 3^1 5^0

LCM

LCM stands for least common multiplier.

Given two numbers, a and b, what can be done to find lcm(a, b), the smallest integer g, where a | g and b | g.

Such a problem can be solved using a multiple solutions.

Finding LCM using Listing Method

List the multiple of 6 and 8

- Multiples of 6 = 6, 12, 18, 24, 50, ...
- Multiples of 8 = 8, 16, 24, 32, 40, ...

The least common multiple of 6 and 8 is, 24

Thus, the LCM of 6 and 8 is 24

Finding LCM using Formula

LCM (a, b) = (a × b) ÷ GCD(a, b)

LCM

Finding LCM using Listing Method

LCM can be calculated by choosing the maximum factors from the both numbers' factorization.

Choosing the maximum will allow both a and b to be divisible by the LCM result.

Such solution takes O(sqrt(max(a, b)).

30	2^1	3^1	5^1
12	2^2	3^1	5^0
60	2^2	3^1	5^1

LET'S DO SOME CODE

HELWAN ICPC
COMMUNITY

THANK YOU

For attending

For more about Number Theory, you may watch these:

https://www.youtube.com/watch?v=nUFsapuZQ-c&t=3s
https://www.youtube.com/watch?v=jEDIio9-IAc
https://www.youtube.com/watch?v=DlzXoNbpgJg