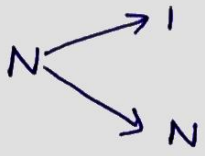
Sieve of Eratosthenes

⟹ Fastest way of Finding Prime numbers in Competitive Coding.

def^n :- A prime number is a number which has only two factors 1 & itself.



Brute force approach: Check all numbers between ~~1 to N~~ 2 to N-1 to check if it is prime

if a number has a divisor in this range then it is not prime.

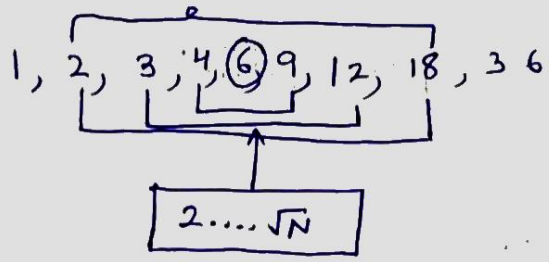
Time complexity

$O(N)$ for 1 Number

$O(N \cdot N) = O(N^2)$ for N numbers

A slightly better approach

iterate from 2 to $\sqrt{N}$



for N numbers Time complexity = $O(N \cdot \sqrt{N})$

if $N = 10^6$ then $\sqrt{N} = 10^3$

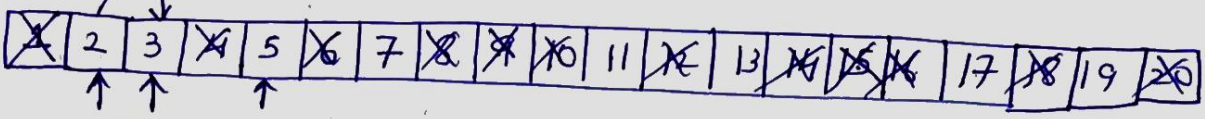
$\therefore$ $N \cdot \sqrt{N} = 10^9$ → we will be getting a TLE (Time limit Exceeded)

Sieve Approach

→ In sieve approach we will be directly generating an array containing prime Numbers

Prime Sieve

Suppose N=20



Mark multiples of 2 as not prime

Now Since 3 does not have a factor smaller than it which can be a multiple of therefore 3 is prime

Now mark multiples of 3 as not prime

Similarly 5 is prime & mark its multiples as not prime

5 → 10, 15, 20, 25, 30, ...

Not prime.

Chance of another optimization

Suppose you are at a number 5 in an array and you want to mark all the multiples of 5 as not prime

```
for (j = 2*no; j<= MaxRange; j+= no)
{
    primes(j) = false;
}
```

* In order to further improve the time complexity instead of starting with a 2*no. we can start the loop from $(no.)^2$

Eg for no = 5 → ~~10, 15, 20~~ 25, 30, ...

i.e we can skip this part which is going to help in reducing the number of iterations

* The justification of starting with no*no instead of 2*no.

Note: ~~In order to~~ Take into consideration the divisors/factors of a number.

i.e. for a no. n

1, ..., $\sqrt{n}$, ... n

$1 \ldots\ldots, \sqrt{n}, \ldots\ldots n$

Some of the factors will be less than $\sqrt{n}$ & some will be greater than $\sqrt{n}$ (to be precise factors comes in pairs so half of the factors of n would be less than $\sqrt{n}$ & half would be more than $\sqrt{n}$)

for $(n-1)$ there would be some factors which are less than or equal to $\sqrt{n-1}$

Now for n ∵ we already have ~~traversed~~ traversed from 1 to $\sqrt{n-1}$ ∴ This part would be taken care of ∵ we actually have covered one factor of it which lies w/in 1 to $\sqrt{n}$ part hence that factor would make mark the numbers in range from $\sqrt{n}$ to n not prime.

Eg

$1 \ldots\ldots 5 \ldots\ldots\ldots 25$

p 10, 15, 20 p^2, p^2+p, p^2+2p, p^2+3p, ...

Then no. will have atleast 1 divisor which will be less than 5
hence we can avoid marking these no. as not prime ∵ they will be taken care of by no. lower than 5

hence start the loop from $(n \times n)$

Another optimization can be that all even numbers are not prime, 2 is the only even prime no number.

∴ all even no. can be skipped ∴ we check only the odd numbers.

Time complexity

$$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{7} + \cdots\cdots$$

∵ we are marking all multiples of 2 as not prime

marking all multiples of 3 as not prime

$$= N\left[\frac{1}{2} + \frac{1}{3} + \cdots\cdots\right]$$

$$T(n) = O(N \log\log N)$$

if N is a large no. i.e $N = 10^{18}$

$$\log_2 10^{18} = 60 \text{ & } \log_2 60 \simeq 7$$

$$\therefore O(N * 7) = O(N)$$

∴ The time complexity is approximately linear.

which is a great improvement.