Sieve of Eratosthenes

Need :

Sieve of Eratosthenes is used for generating primes in a range **[L , R]** in $O(n \log \log n)$. It is a much efficient algorithm compared to generating all primes using brute force .

Concept :

The algorithm works as follows :

- We make a Boolean array of size of upper limit of the range of given range . In this case **R** . Let's name it **is_prime** .
- We mark all the elements of the **is_prime** array [2 , R] initially as true.
- Then we traverse from 2 to R , marking all the multiples of the current number (which is true) as **false** i.e. not prime .
- Now we can store all the numbers marked true inside a array **prime**.

Example :

For example:

Let's take the range from [2 , 49] .

- **Step 1 :**
    - Making a array is_prime and marking all its values as true .

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

- **Step 2 :**
  - Traverse the array and for all the values that are true , mark all it's multiples as false .
    - First we get 2 →

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

    - Next we get 3 as true →

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

    - Next as we see 4 is marked false , we jump to the next true number i.e 5 →

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

    - Then we go to next true value i.e. 7

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

Here we can see that all the numbers that are not prime are marked false and all the primes are true . The idea behind is this: A number is prime, if none of the smaller prime numbers divides it. Since we iterate over the prime numbers in order, we already marked all numbers, who are divisible by at least one of the prime numbers, as divisible. Hence if we reach a cell and it is not marked, then it isn't divisible by any smaller prime number and therefore has to be prime.

Implimentation :

```
int n;
vector<char> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

Optimizations :

1. Sieving till root :

- To find all the primes , it is enough to sieve till the root . Using this we can deduce :

```
int n;
vector<char> is_prime(n+1, true);
is_prime[0] = is_prime[1] = false;
for (int i = 2; i * i <= n; i++) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j += i)
            is_prime[j] = false;
    }
}
```

2. Sieving for odd numbers only :

- Since all even numbers (except 22) are composite, we can stop checking even numbers at all. Instead, we need to operate with odd numbers only.
- First, it will allow us to half the needed memory. Second, it will reduce the number of operations performing by algorithm approximately in half.