Linear Sieve - Algorithms for Competitive Programming

https://cp-algorithms.com/algebra/prime-sieve-linear.html

The **classical Sieve of Eratosthenes algorithm** takes O(N log (log N)) time to find all prime numbers less than N. In this article, a modified Sieve is discussed that works in O(N) time.

Example :

```
Given a number N, print all prime
numbers smaller than N

Input :  int N = 15
Output : 2 3 5 7 11 13

Input : int N = 20
Output : 2 3 5 7 11 13 17 19
```

Manipulated Sieve of Eratosthenes algorithm works as follows:

```
For every number i where i varies from 2 to N-1:
    Check if the number is prime. If the number
    is prime, store it in prime array.

For every prime numbers j less than or equal to the smallest
prime factor p of i:
    Mark all numbers i*p as non_prime.
    Mark smallest prime factor of i*p as j
```

Below is the implementation of the above idea.

```
// C++ program to generate all prime numbers
// less than N in O(N)
#include<bits/stdc++.h>
using namespace std;
const long long MAX_SIZE = 1000001;

// isPrime[] : isPrime[i] is true if number is prime
// prime[] : stores all prime number less than N
// SPF[] that store smallest prime factor of number
// [for Exp : smallest prime factor of '8' and '16'
// is '2' so we put SPF[8] = 2 , SPF[16] = 2 ]
vector<long long >isprime(MAX_SIZE , true);
vector<long long >prime;
vector<long long >SPF(MAX_SIZE);

// function generate all prime number less than N in O(n)
void manipulated_seive(int N)
{
    // 0 and 1 are not prime
    isprime[0] = isprime[1] = false ;

    // Fill rest of the entries
    for (long long int i=2; i<N ; i++)
    {
        // If isPrime[i] == True then i is
        // prime number
        if (isprime[i])
        {
            // put i into prime[] vector
            prime.push_back(i);

            // A prime number is its own smallest
            // prime factor
            SPF[i] = i;
        }
        // Remove all multiples of i*prime[j] which are
        // not prime by making isPrime[i*prime[j]] = false
        // and put smallest prime factor of i*Prime[j] as prime[j]
        // [ for exp :let i = 5 , j = 0 , prime[j] = 2 [ i*prime[j] = 10 ]
        // so smallest prime factor of '10' is '2' that is prime[j] ]
        // this loop run only one time for number which are not prime
        for (long long int j=0;
            j < (int)prime.size() &&
            i*prime[j] < N && prime[j] <= SPF[i];
            j++)
        {
            isprime[i*prime[j]]=false;

            // put smallest prime factor of i*prime[j]
            SPF[i*prime[j]] = prime[j] ;
        }
    }
}

// driver program to test above function
int main()
{
    int N = 13 ; // Must be less than MAX_SIZE

    manipulated_seive(N);

    // print all prime number less than N
    for (int i=0; i<prime.size() && prime[i] <= N ; i++)
        cout << prime[i] << " ";
    return 0;
}
```

Output :

```
2 3 5 7 11
```

Auxiliary Space: O(1)

Illustration:

```
isPrime[0] = isPrime[1] = 0

After i = 2 iteration :
isPrime[]   [F, F, T, T, F, T, T, T]
SPF[]       [0, 0, 2, 0, 2, 0, 0, 0]
     index   0  1  2  3  4  5  6  7

After i = 3 iteration :
isPrime[]  [F, F, T, T, F, T, F, T, T, F ]
SPF[]      [0, 0, 2, 3, 2, 0, 2, 0, 0, 3 ]
  index     0  1  2  3  4  5  6  7  8  9

After i = 4 iteration :
isPrime[]  [F, F, T, T, F, T, F, T, F, F]
SPF[]      [0, 0, 2, 3, 2, 0, 2, 0, 2, 3]
  index     0  1  2  3  4  5  6  7  8  9
```