Sieve of Eratosthenes - Algorithms for Competitive Programming

https://cp-algorithms.com/algebra/sieve-of-eratosthenes.html

Given a number n, print all primes smaller than or equal to n. It is also given that n is a small number.

Example:

👉 Input : n =10

👉 Output : 2 3 5 7

👉 Input : n = 20

👉 Output: 2 3 5 7 11 13 17 19

Following is the algorithm to find all the prime numbers less than or equal to a given integer n by the Eratosthene's method:

When the algorithm terminates, all the numbers in the list that are not marked are prime.

Explanation with Example:

Let us take an example when n = 100. So, we need to print all prime numbers smaller than or equal to 100.

We create a list of all numbers from 2 to 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

According to the algorithm we will mark all the numbers which are divisible by 2 and are greater than or equal to the square of it.

	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

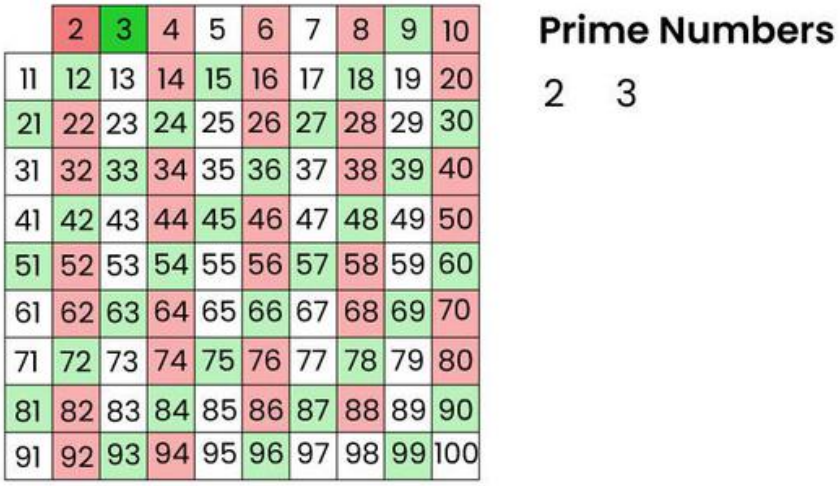
Prime Numbers

2

Sieve of Eratosthenes

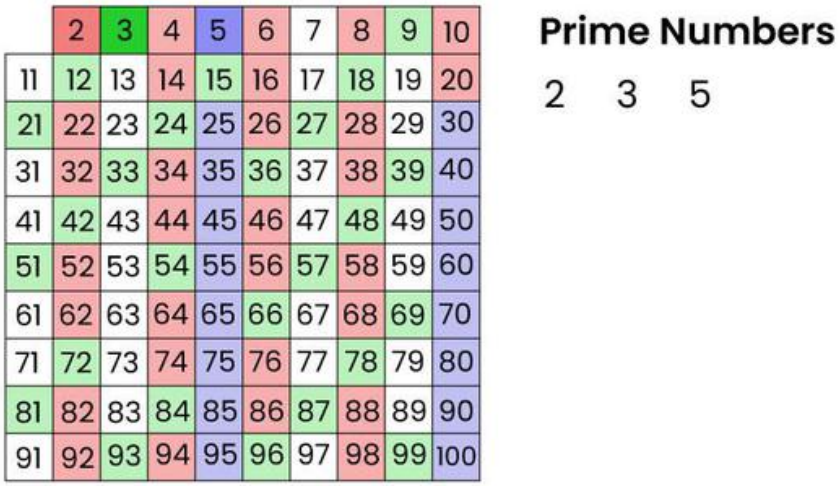
Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.

0 seconds of 0 secondsVolume 0%



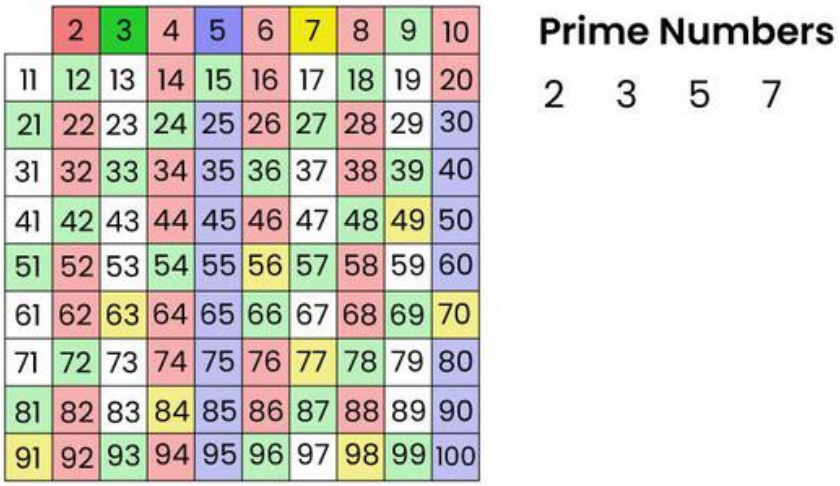
Sieve of Eratosthenes

We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.



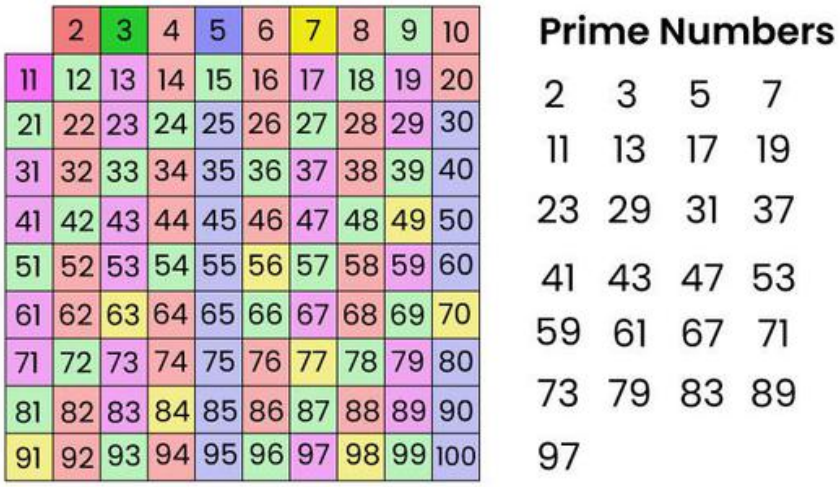
Sieve of Eratosthenes

We move to our next unmarked number 7 and mark all multiples of 7 and are greater than or equal to the square of it.



Sieve of Eratosthenes

We continue this process, and our final table will look like below:



Sieve of Eratosthenes

So, the prime numbers are the unmarked ones: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 and 97.

Implementation:

Following is the implementation of the above algorithm. In the following implementation, a boolean array arr[] of size n is used to mark multiples of prime numbers.

```
// C++ program to print all primes smaller than or equal to
// n using Sieve of Eratosthenes
#include <bits/stdc++.h>
using namespace std;

void SieveOfEratosthenes(int 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.
    bool prime[n + 1];
    memset(prime, true, sizeof(prime));

    for (int p = 2; p * p <= n; p++) {
        // If prime[p] is not changed, then it is a prime
        if (prime[p] == true) {
            // 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.
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }

    // Print all prime numbers
    for (int p = 2; p <= n; p++)
        if (prime[p])
            cout << p << " ";
}

// Driver Code
int main()
{
```

```
    int n = 30;
    cout << "Following are the prime numbers smaller "
        << " than or equal to " << n << endl;
    SieveOfEratosthenes(n);
    return 0;
}
```

Output

```
Following are the prime numbers smaller  than or equal to 30
2 3 5 7 11 13 17 19 23 29
```

Time Complexity: O(n*log(log(n)))

Auxiliary Space: O(n)

Optimizing by removing evens :

```
// the following implementation
// stores only halves of odd numbers
// the algorithm is a faster by some constant factors

#include <bitset>
#include <iostream>
using namespace std;

bitset<500001> Primes;
void SieveOfEratosthenes(int n)
{
    Primes[0] = 1;
    for (int i = 3; i*i <= n; i += 2) {
        if (Primes[i / 2] == 0) {
```

```
            for (int j = 3 * i; j <= n; j += 2 * i)
                Primes[j / 2] = 1;
        }
    }
}

int main()
{
    int n = 100;
    SieveOfEratosthenes(n);
    for (int i = 1; i <= n; i++) {
        if (i == 2)
            cout << i << ' ';
        else if (i % 2 == 1 && Primes[i / 2] == 0)
            cout << i << ' ';
    }
    return 0;
}
```

Output

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Time Complexity: O(n*log(log(n)))

Auxiliary Space: O(n)

Practice Problems

- Leetcode - Four Divisors
- Leetcode - Count Primes
- SPOJ - Printing Some Primes
- SPOJ - A Conjecture of Paul Erdos
- SPOJ - Primal Fear

- SPOJ - Primes Triangle (I)
- Codeforces - Almost Prime
- Codeforces - Sherlock And His Girlfriend
- SPOJ - Namit in Trouble
- SPOJ - Bazinga!
- Project Euler - Prime pair connection
- SPOJ - N-Factorful
- SPOJ - Binary Sequence of Prime Numbers
- UVA 11353 - A Different Kind of Sorting
- SPOJ - Prime Generator
- SPOJ - Printing some primes (hard)
- Codeforces - Nodbach Problem
- Codeforces - Colliders