

Sieve of Eratosthenes

Finding primes in Range [1:n] without using Sieve of Eratosthenes

We can check if each number is a prime or not. To check if the number 'i' is prime we will traverse all the numbers till $[2, \sqrt{i}]$ can check if they divide n or not. Similarly, we do this for all the n numbers.

Time Complexity: $O(n \sqrt{n})$

Space Complexity: $O(1)$

```
bool check_prime(int n) {
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0)
            return false;
    }
    return true;
}

void get_primes_till_n(int n) {
    for (int i = 2; i <= n; i++) {
        if (check_prime(i)) {
            cout << i << " ";
        }
    }
}
```

15.1 Sieve of Eratosthenes | Challenge | C++ Placement Course

Prime Factorization Question

	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

Sieve of Eratosthenes

Algorithm: We start from 2, and on each encounter of a prime number, we mark its multiples as composite.

Time Complexity: $O(n \log \log n)$

Space Complexity: $O(n)$

```
void primeSieve(int n) {
    int prime[n+1] = {0};
    for (int i = 2; i <= n; i++) {
        if (prime[i] == 0) {
            for (int j = i * i; j <= n; j += i) {
                prime[j] = 1;
            }
        }
    }

    for (int i = 2; i <= n; i++) {
        if (prime[i] == 0) {
            cout << i << " ";
        }
    } cout << endl;
}
```

15.1 Sieve of Eratosthenes | Challenge | C++ Placement Course

Firstly we pick a number ($i=2$ to $i \leq n$, with $i++$) and then we will cross out all of its multiple for being a prime as they can't be. For crossing out of a number we will use that i and from that we will start another loop from ($j=i*i$ to $\leq n$, with $j+=i$)

	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

Prime Factorization using Sieve

Explanation:

while(num != 1):

We keep on dividing it with its smallest prime factor.

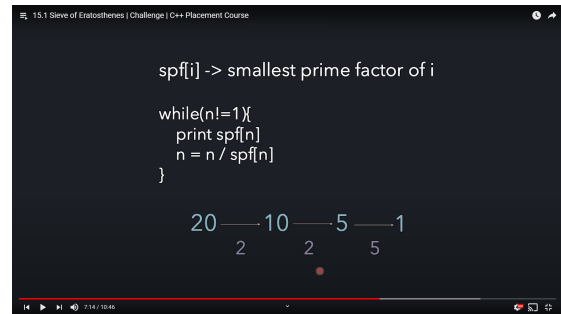
The smallest prime factor is pre-calculated using a slightly modified prime sieve.

Since we start from 2 and go on, we mark the first multiple as the spf.

Preprocessing for Sieve: $O(n \log \log n)$

Time Complexity for factorization: $O(\log n)$

Space Complexity: $O(n)$



```
void primefactor(int n) {

    int spf[n+1] = {0};
    for (int i = 2; i <= n; i++) {
        spf[i] = i;
    }
    for (int i = 2; i <= n; i++) {
        if (spf[i] == i) {
            for (int j = i * i; j <= n; j += i) {
                if (spf[j] == j) {
                    spf[j] = i;
                }
            }
        }
    }
    while (n != 1) {
        cout << spf[n] << " ";
        n = n / spf[n];
    }
}
```

Additional Question:

[Find primes in the given range](#)