codility

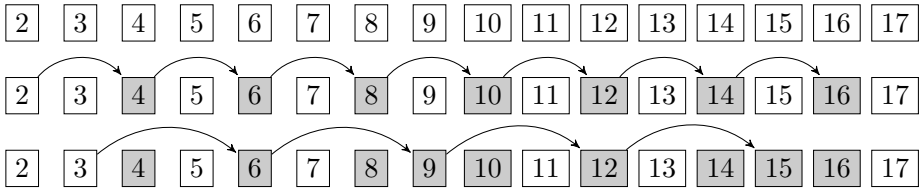
WE TEST CODERS

# Chapter 11

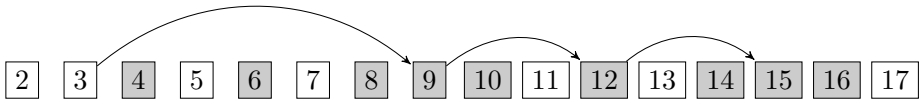
# Sieve of Eratosthenes

The Sieve of Eratosthenes is a very simple and popular technique for finding all the prime numbers in the range from 2 to a given number $n$. The algorithm takes its name from the process of sieving—in a simple way we remove multiples of consecutive numbers.

Initially, we have the set of all the numbers $\{2, 3, \ldots, n\}$. At each step we choose the smallest number in the set and remove all its multiples. Notice that every composite number has a divisor of at most $\sqrt{n}$. In particular, it has a divisor which is a prime number. It is sufficient to remove only multiples of prime numbers not exceeding $\sqrt{n}$. In this way, all composite numbers will be removed.




The above illustration shows steps of sieving for $n = 17$. The elements of the processed set are in white, and removed composite numbers are in gray. First, we remove multiples of the smallest element in the set, which is 2. The next element remaining in the set is 3, and we also remove its multiples, and so on.




The above algorithm can be slightly improved. Notice that we needn't cross out multiples of $i$ which are less than $i^2$. Such multiples are of the form $k \cdot i$, where $k < i$. These have already been removed by one of the prime divisors of $k$. After this improvement, we obtain the following implementation:

**11.1: Sieve of Eratosthenes.**

```
1 def sieve(n):
2     sieve = [True] * (n + 1)
3     sieve[0] = sieve[1] = False
4     i = 2
5     while (i * i <= n):
6         if (sieve[i]):
```

```
            k = i * i
            while (k <= n):
                sieve[k] = False
                k += i
        i += 1
    return sieve
```

Let's analyse the time complexity of the above algorithm. For each prime number $p_j \leqslant \sqrt{n}$ we cross out at most $\frac{n}{p_j}$ numbers, so we get the following number of operations:

$$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \ldots = \sum_{p_j \leqslant \sqrt{n}} \frac{n}{p_j} = n \cdot \sum_{p_j \leqslant \sqrt{n}} \frac{1}{p_j} \tag{11.1}$$

The sum of the reciprocals of the primes $p_j \leqslant n$ equals asymptotically $O(\log \log n)$. So the overall time complexity of this algorithm is $O(n \log \log n)$. The proof is not trivial, and is beyond the scope of this article. An example proof can be found here.

## 11.1. Factorization

Factorization is the process of decomposition into prime factors. More precisely, for a given number $x$ we want to find primes $p_1, p_2, \ldots, p_k$ whose product equals $x$.

Use of the sieve enables fast factorization. Let's modify the sieve algorithm slightly. For every crossed number we will remember the smallest prime that divides this number.

**11.2: Preparing the array $F$ for factorization.**

```
def arrayF(n):
    F = [0] * (n + 1)
    i = 2
    while (i * i <= n):
        if (F[i] == 0):
            k = i * i
            while (k <= n):
                if (F[k] == 0):
                    F[k] = i;
                k += i
        i += 1
    return F
```

For example, take an array $F$ with a value of $n = 20$:

| 0 | 0 | 2 | 0 | 2 | 0 | 2 | 3 | 2 | 0 | 2 | 0 | 2 | 3 | 2 | 0 | 2 | 0 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

With this approach we can factorize numbers very quickly. If we know that one of the prime factors of $x$ is $p$, then all the prime factors of $x$ are $p$ plus the decomposition of $\frac{x}{p}$.

**11.3: Factorization of $x$ — $O(\log x)$.**

```
def factorization(x, F):
    primeFactors = []
    while (F[x] > 0):
        primeFactors += [F[x]]
        x /= F[x]
    primeFactors += [x]
    return primeFactors
```

Number $x$ cannot have more than $\log x$ prime factors, because every prime factor is $\geqslant 2$. Factorization by the above method works in $O(\log x)$ time complexity. Note that consecutive factors will be presented in non-decreasing order.

Every lesson will provide you with programming tasks at `http://codility.com/programmers`.