Sieve of Eratosthenes

This is a simple algorithm to find all the prime numbers from 2 to N. The algorithm basically un-marks all the numbers from 2 to N initially, Un-marking a number denotes that the number is prime while on the other hand if a number is marked it means that the number is composite, so initially, we consider all the numbers till N are prime.

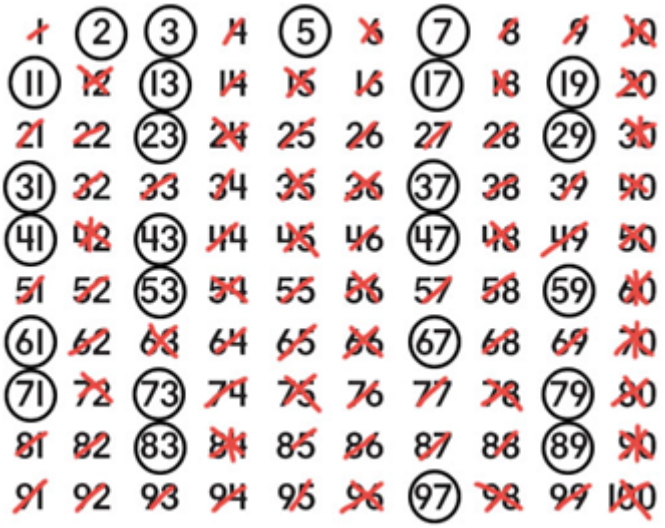
Now, starting from 2, we take a number that is unmarked and mark all the numbers till N which are multiples of the current unmarked number. In other words, we mark those numbers as composites or not prime.
In this way, all the unmarked numbers will be the prime numbers till N.

Since the prime factors of a number are bounded by sqrt(N), we can limit our iterations till sqrt(N) starting from 2, and then for each unmarked number, we mark the multiples of that number till N, similar to the procedure described above.

For Example: We want to find all the prime numbers till 100. So our algorithm will start from 2 [an unmarked(denoted by circles) number] and start marking(denoted by crosses) all the multiples of the un-marked number till 100. We mark 1 as it is not a prime number initially.

Step 1: Start with 2 and mark all multiples of 2 till 100 - {4,6,8,10,...,100}

Step 2: Start with 3 (next un-marked number) and mark all multiples of 3 till 100 - {3,6,9,12, ..., 99}.
...
Step n: Start with 7 the un-marked number before sqrt(100) i.e 10 and mark all multiples of 7 till 100 - {7,14,21, ..., 98}.



Hence all the un-marked numbers from 2 to 100 are the prime numbers from 2 to 100.

Pseudocode:

```
/* Input n is a positive integer, finds all the prime numbers from 2 to n.*/
function sieveOfEratosthenes(n)

        /*
                Declaring a sieve array whose value for the ith index is false if the ith
                value is composite, and is true if the ith value is prime
        */

        bool  sieve[n + 1]
        //   Initializing sieve array to be true
        for i = 1 to n
                sieve[i] = true
        sieve[1] = false

        /*
                Now iterating from 2 and for every prime value marking the multiples of
                that prime up to n as composites i.e false
        */
        for i = 2; i * i<=n; i++
                if sieve[i] equals true
                        //   Marking all multiples of i till n to be false(composites)
                        for j = i * i; j<=n; j += i
                                sieve[j] = false

        //   Iterating from 2 to n to print all primes
        print("The prime numbers smaller than or equal to n are")
        for i = 2 to n
                if sieve[i] equals true
                        print(i)
```

Time complexity: $O(n*\log_2(\log_2 n))$, where n is the number up to which the primes are to be found.