Example: Sieve of Erathostenes

- Algorithm to compute all prime numbers less than or equal to some given number *N*
  - invented by the Greek mathematician Erathostenes (ca. 276 – 195 BC)
- A prime number is a natural number that is evenly divisible by only 1 and itself
  - there are infinitely many prime numbers
  - the largest known prime number so far is the Mersenne prime $2^{74202281}$-1, which has 22,338,618 decimal digits
- Prime numbers have very important applications in public-key cryptography
  - the keys are based on very large prime numbers (with 100 or 200 decimal numbers)

Sequential algorithm

- Create a list of marks for the natural numbers 2, 3, ..., N
  - initially all numbers are unmarked

  $k = 2$

  repeat

  mark all multiples of k between k^2 and N

  set k to the next unmarked number

  until $k^2 > N$

  All unmarked numbers are prime

- The complexity of the algorithm is $\Theta(N \ln \ln N)$
  - N grows exponentially with the number of digits
  - not a practical algorithm for very large N

Illustration, N=30

- k = 2
  - mark all multiples of 2 between 4 and 30

k

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

- k = 3
  - mark all multiples of 3 between 9 and 30

k

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

- k = 5
  - mark all multiples of 5 between 25 and 30

k

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

- k = 6
  - k^2 > 30, the algorith terminates. The unmarked numbers are prime.

Implementation

- We need one mark for each number in the range [2, *N*]
  - allocate a byte array of size *N+1*
- Initially all numbers in [0, *N*] are initialized to unmarked
  - define constants to denote marked and unmarked numbers

```
const char unmarked = (char)0;
const char   marked = (char)1;
```

  - the numbers 0 and 1 are initially set to marked, since they are not prime
- Loop through unmarked numbers and mark all multiples

```
for (i = 2; i <= (int)sqrt(N); i++)
    if (prime[i]==unmarked)
        for (k=i*i; k<=N; k+=i)
            prime[k] = marked;
```

- Finally count how many unmarked positions there are
  - print out the number of primes ≤ N and the largest of these

Improvement

- All multiples of 2 are immediately marked in the array
  - we use *N*/2 positions to store the fact that 2 is a prime number
- Store only marks for odd numbers ≥ 3 in the array *prime*
  - we need (*N*-3)/2+1 positions for the marks (position 0 is not used)

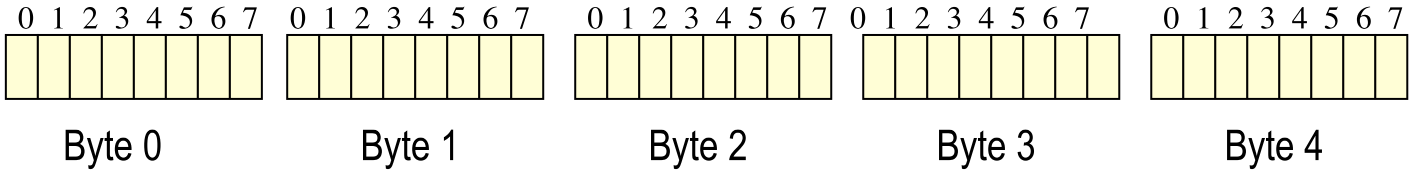
3	5	7	9	11	13	15	17	19	21	23	25	27	29
1	2	3	4	5	6	7	8	9	10	11	12	13	14

- The number represented by the mark in position *i* is 2**i*+1
- The mark for the odd number *k* is stored in position *k*/2

- This improvement makes the program twice as fast, since we go through a loop that is half the size

Further improvement

- The value of *N* is limited by the amount of memory
  - we use one byte for each mark that we store
- We don't actually need a byte to store a binary mark
  - one bit is enough – can store 8 times more information



- Assuming we store a mark for all natural numbers from 0 to *N*, the mark for the number *i* will be stored in byte *i*/8 bit *i*%8