Sviluppo del Crivello di Eratostene

Claudio Mirolo

Dipartimento di Matematica e Informatica,
Università di Udine, via delle Scienze 206 – Udine

claudio.mirolo@uniud.it

Corso di Programmazione

www.dimi.uniud.it/claudio

```
boolean[] sieve = new boolean[ n+1 ];


for ( int k=2; k<=n; k=k+1 ) {

  sieve[k] = true;
}

Vector<Integer> primes = new Vector<Integer>();


int p = 2;

while ( p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );

    for ( int k=2*p; k<=n; k=k+p ) {
      sieve[k] = false;
  }}
  p = p + 1;
}
```

```
return primes;
```

```
  boolean[] sieve = new boolean[ n+1 ];
  sieve[2] = true;

  for ( int k=3; k<=n; k=k+1 ) {

    sieve[k] = ( k % 2 > 0 );
  }

  Vector<Integer> primes = new Vector<Integer>();
  primes.add( 2 );

  int p = 3;

  while ( p <= n ) {

    if ( sieve[p] ) {

      primes.add( p );

      for ( int k=2*p; k<=n; k=k+p ) {
        sieve[k] = false;
    }}
    p = p + 2;
  }









  return primes;
```

```
boolean[] sieve = new boolean[ n+1 ];
sieve[2] = true;

for ( int k=3; k<=n; k=k+1 ) {

  sieve[k] = ( k % 2 > 0 );
}

Vector<Integer> primes = new Vector<Integer>();
primes.add( 2 );

int p = 3;

while ( p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );

    for ( int k=2*p; k<=n; k=k+p ) {
      sieve[k] = false;
  }}
  p = p + 2;
}
```

```
return primes;
```

```
  boolean[] sieve = new boolean[ n+1 ];
  sieve[2] = true;

  for ( int k=3; k<=n; k=k+1 ) {

    sieve[k] = ( k % 2 > 0 );
  }

  Vector<Integer> primes = new Vector<Integer>();
  primes.add( 2 );

  int p = 3;

| while ( p*p <= n ) {

    if ( sieve[p] ) {

      primes.add( p );

      for ( int k=2*p; k<=n; k=k+p ) {
        sieve[k] = false;
    }}
    p = p + 2;
  }

| while ( p <= n ) {
|
|   if ( sieve[p] ) {
|
|     primes.add( p );
|   }
|   p = p + 2;
| }

  return primes;
```

```
boolean[] sieve = new boolean[ n+1 ];
sieve[2] = true;

for ( int k=3; k<=n; k=k+1 ) {

  sieve[k] = ( k % 2 > 0 );
}

Vector<Integer> primes = new Vector<Integer>();
primes.add( 2 );

int p = 3;

while ( p*p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );

    for ( int k=2*p; k<=n; k=k+p ) {
      sieve[k] = false;
  }}
  p = p + 2;
}

while ( p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );
  }
  p = p + 2;
}

return primes;
```

```
  boolean[] sieve = new boolean[ n+1 ];
|
|
|  for ( int k=3; k<=n; k=k+2 ) {
|
|    sieve[k] = true;
|  }

  Vector<Integer> primes = new Vector<Integer>();
  primes.add( 2 );

  int p = 3;

  while ( p*p <= n ) {

    if ( sieve[p] ) {

      primes.add( p );

|     for ( int k=p*p; k<=n; k=k+2*p ) {
        sieve[k] = false;
    }}
    p = p + 2;
  }

  while ( p <= n ) {

    if ( sieve[p] ) {

      primes.add( p );
    }
    p = p + 2;
  }

  return primes;
```

```
boolean[] sieve = new boolean[ n+1 ];


for ( int k=3; k<=n; k=k+2 ) {

  sieve[k] = true;
}

Vector<Integer> primes = new Vector<Integer>();
primes.add( 2 );

int p = 3;

while ( p*p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );

    for ( int k=p*p; k<=n; k=k+2*p ) {
      sieve[k] = false;
  }}
  p = p + 2;
}

while ( p <= n ) {

  if ( sieve[p] ) {

    primes.add( p );
  }
  p = p + 2;
}

return primes;
```

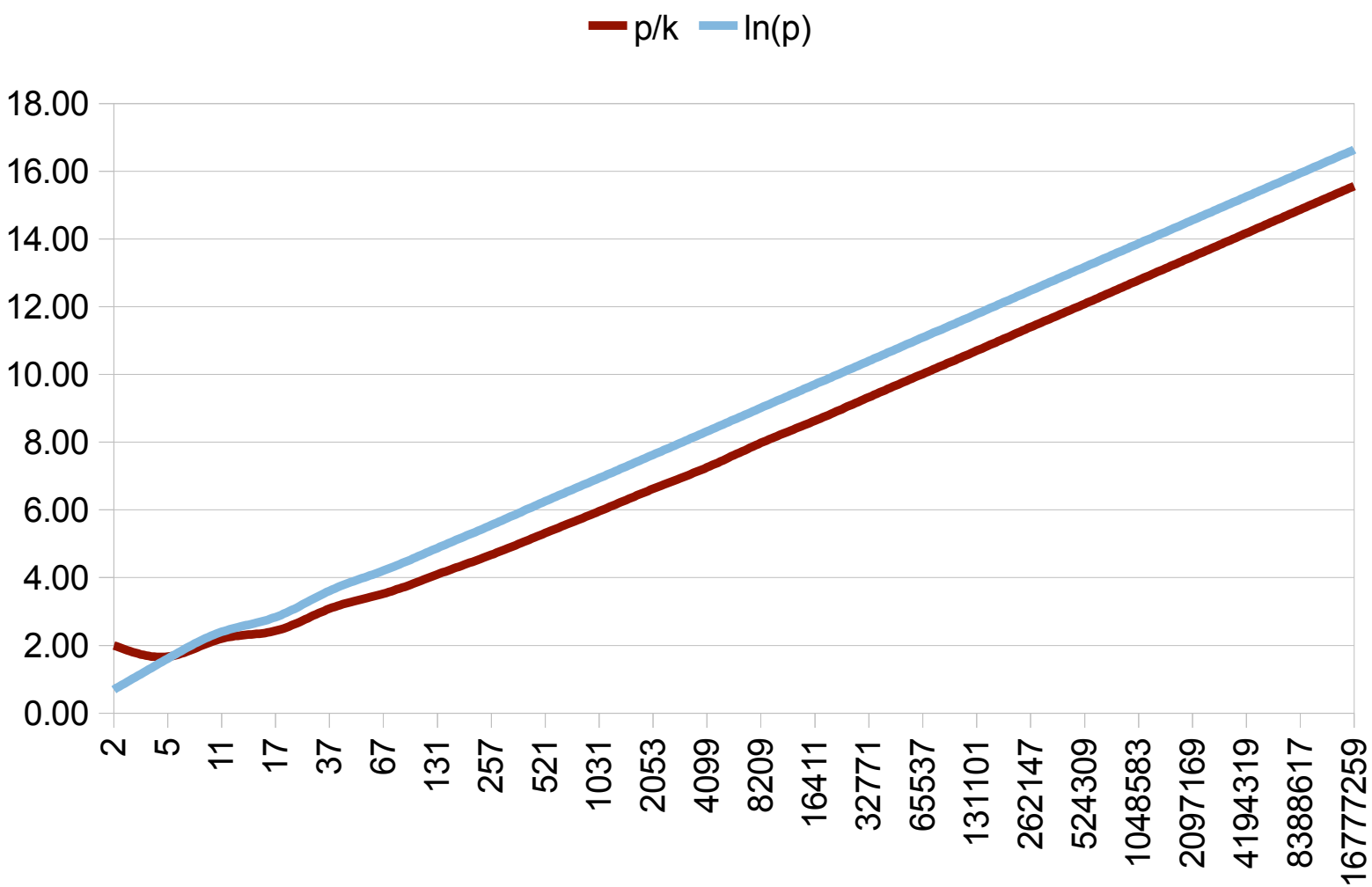
```
|  boolean[] sieve = new boolean[ n/2+1 ];


   for ( int k=3; k<=n; k=k+2 ) {

|    sieve[k/2] = true;
   }

   Vector<Integer> primes = new Vector<Integer>();
   primes.add( 2 );

   int p = 3;

   while ( p*p <= n ) {

|    if ( sieve[p/2] ) {

       primes.add( p );

       for ( int k=p*p; k<=n; k=k+2*p ) {
|        sieve[k/2] = false;
     }}
     p = p + 2;
   }

   while ( p <= n ) {

|    if ( sieve[p/2] ) {

       primes.add( p );
     }
     p = p + 2;
   }

   return primes;
```

```
boolean[] sieve = new boolean[ n/2+1 ];


for ( int k=3; k<=n; k=k+2 ) {

  sieve[k/2] = true;
}

Vector<Integer> primes = new Vector<Integer>();
primes.add( 2 );

int p = 3;

while ( p*p <= n ) {

  if ( sieve[p/2] ) {

    primes.add( p );

    for ( int k=p*p; k<=n; k=k+2*p ) {
      sieve[k/2] = false;
  }}
  p = p + 2;
}

while ( p <= n ) {

  if ( sieve[p/2] ) {

    primes.add( p );
  }
  p = p + 2;
}

return primes;
```

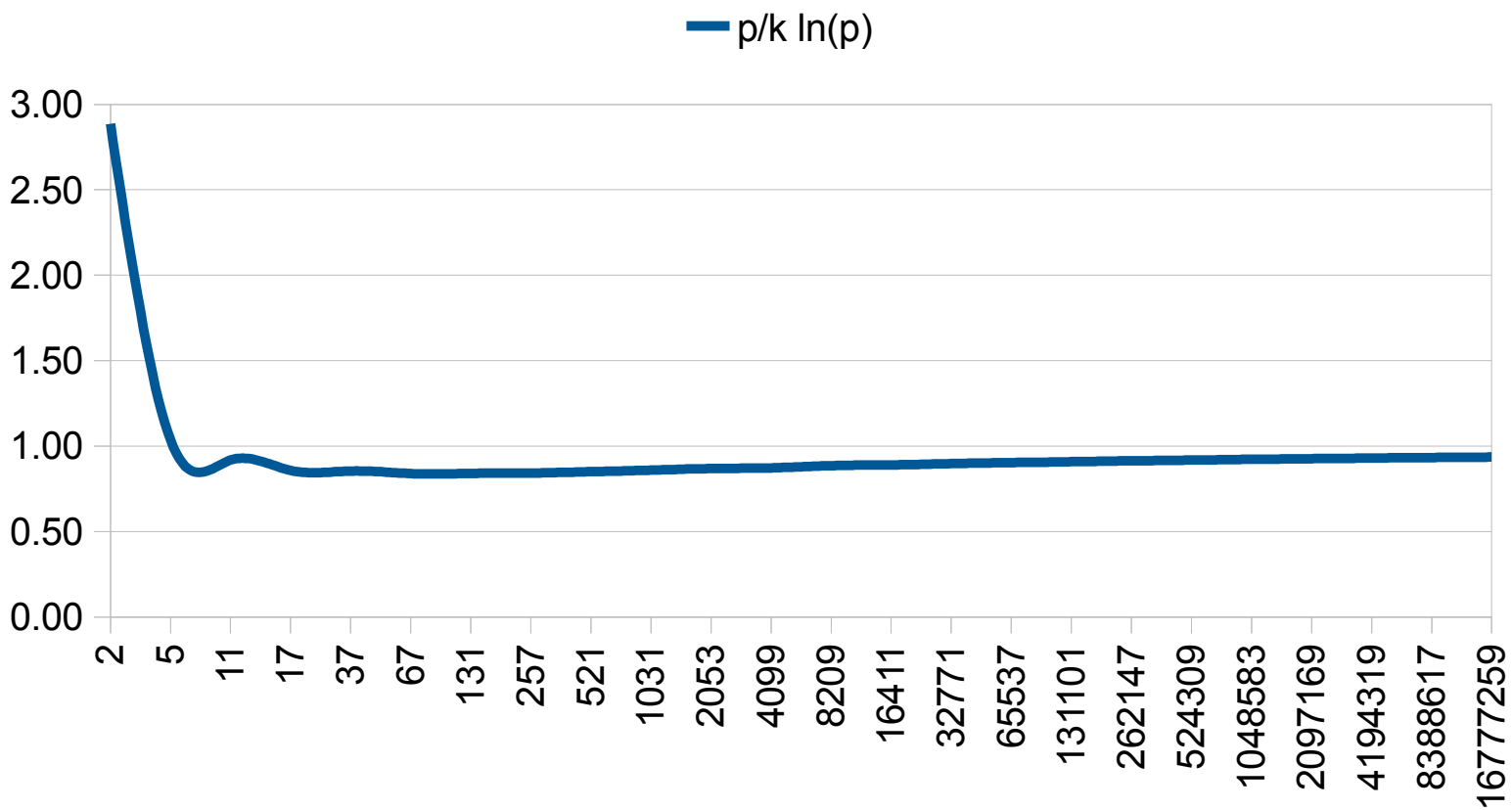
```
|   int m = n / 2;
|   boolean[] sieve = new boolean[ m+1 ];
|
|   for ( int i=1; i<=m; i=i+1 ) {
|
|     sieve[i] = true;
|   }

    Vector<Integer> primes = new Vector<Integer>();
    primes.add( 2 );

    int p = 3;

    while ( p*p <= n ) {

      if ( sieve[p/2] ) {

        primes.add( p );

|       for ( int i=p*p/2; i<=m; i=i+p ) {
|         sieve[i] = false;
|     }}
      p = p + 2;
    }

    while ( p <= n ) {

      if ( sieve[p/2] ) {

        primes.add( p );
      }
      p = p + 2;
    }

    return primes;
```

```
int m = n / 2;
boolean[] sieve = new boolean[ m+1 ];

for ( int i=1; i<=m; i=i+1 ) {

  sieve[i] = true;
}

Vector<Integer> primes = new Vector<Integer>();
primes.add( 2 );

int p = 3;

while ( p*p <= n ) {

  if ( sieve[p/2] ) {

    primes.add( p );

    for ( int i=p*p/2; i<=m; i=i+p ) {
      sieve[i] = false;
  }}
  p = p + 2;
}

while ( p <= n ) {

  if ( sieve[p/2] ) {

    primes.add( p );
  }
  p = p + 2;
}

return primes;
```

Densità dei numeri primi: n / primi $\leq n$



Trend asintotico: inverso densità / ln *n*



Numeri primi gemelli: occorrenze

primo p	**gemelli** $\leq p$
2	0
5	1
11	2
17	3
37	5
67	7
131	10
257	17
521	24
1031	36
2053	62
4099	107
8209	177
16411	290
32771	505
65537	860
131101	1526
262147	2679
524309	4750
1048583	8535
2097169	15500
4194319	27995
8388617	50638
16777259	92246