

Primes v.1.0.

Get it (for free) at www.scyrus.de

Some “rules” for the generation of primes

- Primes are **NEVER** even numbers.
⇒ We have to **check odd numbers only** (this saves lots of time).

- **Every number** that is **NOT a prime** can be written as the product of two or more primes (**prime factorization**).
Look at some examples...

- 5 is a prime number and can't be factorized.
- $10 = 2 \cdot 5$ (2 and 5 are primes)
- $21 = 3 \cdot 7$ (3 and 7 are primes)
- $100 = 2 \cdot 2 \cdot 5 \cdot 5$ (2 and 5 are primes)
- 101 is a prime number and can't be factorized.

⇒ If any integer [Number] **can be divided by a smaller prime** then [Number] is **not prime**.

⇒ We **don't have to check all numbers** smaller than [Number], we **only have to check the primes**.

⇒ And we only have to **check the primes smaller than $\sqrt{[Number]}$** .

Some “rules” for the generation of primes

Question: Is 1013 a prime? It is.

$$\sqrt{1013} = 31.82766\dots \approx 32$$

All primes smaller than 32:

1 : 2
2 : 3
3 : 5
4 : 7
5 : 11
6 : 13
7 : 17
8 : 19
9 : 23
10 : 29
11 : 31
12 : 37

To test 1013, **only 11 primes** have to be checked.

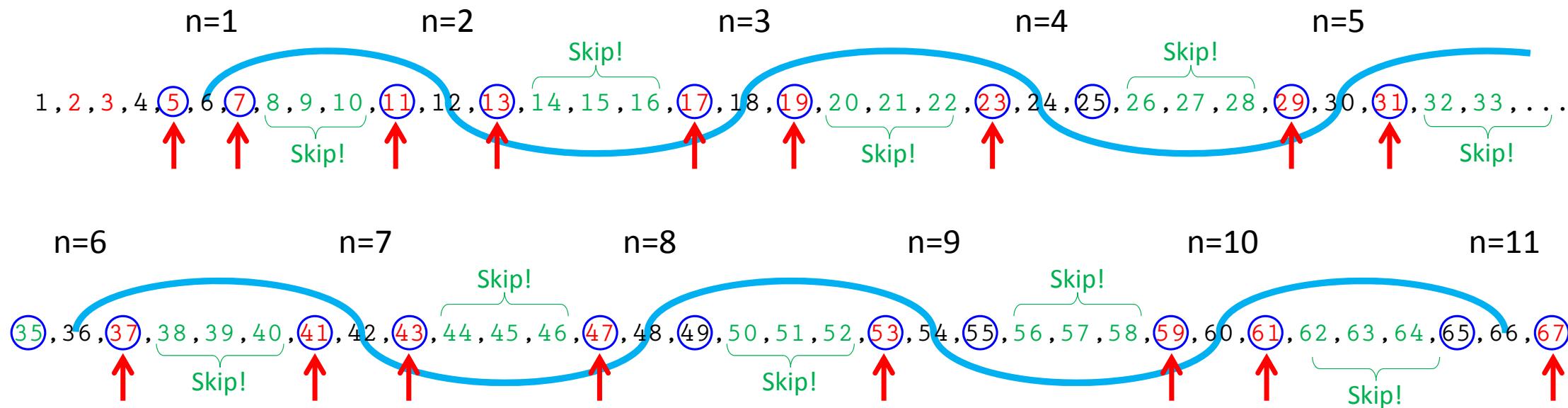
Because primes are never even,
in this case **only 10 primes** have to be tested.

⇒ **Already calculated primes can be used to calculate more primes.**

Another interesting thing about primes...

Primes bigger than 3 are always multiples of 6 either +1 or -1. Thus, **all primes** are like $n*6 \pm 1$.

Not every number that satisfies this criterion is a prime number,
but **every prime number satisfies this criterion**. Take a look at this...



⇒ ALL primes >3 are adjacent to a multiple of 6.

⇒ Primes >3 divided by 6 always leave 5 or 1 as remainder.

⇒ You can use this to **optimize your code** to generate primes.

Basic idea of the “Primes”-macro

First: Generate an array containing only odd numbers.

```
upperlimit=100;
```

This is the arbitrary upper limit. We want to know all primes smaller than 100.

```
large=newArray(round(-1+upperlimit/2));
```

This creates an array with $(\text{upperlimit}/2) - 1$ elements.

```
for (i = 0; i < large.length; i++) {large[i]=3+2*i;}
```

This fills our array with odd numbers starting at 3.

2 is also a prime and even the most annoying and ridiculous of them all, but it's added to our array at the end of this computation.

Result:

```
large=[3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,  
69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99]
```

“large” is just the name of our array, containing all odd numbers > 2 and < 100 .

Basic idea of the “Primes”-macro

First step:

Take the first element (here, 3) and delete all multiples of that number (in fact, they are replaced by zero). You don't have to start at the very beginning. Save time and start at 3^2 .

The increment of this operation is 2×3 , because every second increment of an odd number is an even number, but there are no even numbers in this array. Your result should look like this.

```
large=[3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99]
```

Now, take the next element of this array (here, 5) and do the same: start at 5^2 and use 2×5 as increment.

```
large=[3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99].
```

And 7, starting at 7^2 , with an increment of 2×7 .

```
large=[3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,99].
```

Done! 11^2 is 121, already bigger than your upper limit. Now, only primes remain here.

Don't forget adding the 2 to get the complete set of primes below your upper limit of 100.

The complete macro can be downloaded at www.scyrus.de

The according code that eliminates all none-primes is quite simple:

```
for (p = 0; p<large.length && large[p]<sqrt(upperlimit); p++) {  
    stp=large[p];  
    strt=((large[p]*large[p])-3)/2-stp;  
    n=1;  
    if (large[p]>2) { while(strt+n*stp<large.length) {large[strt+n*stp]=0; n++;} }  
}
```

If you code something like this:

Start with a big array to store your primes and delete the empty elements (zeros) later;
don't add results (step by step) to an already existing array by using commands
like "Array.concat(array,result)"!

This will become extremely slow and boring if looped for millions of times!

After starting “Primes” in ImageJ...

1 : 2

2 : 3

3 : 5

4 : 7

5 : 11

6 : 13

7 : 17

8 : 19

9 : 23

[...]

19 : 67

20 : 71

21 : 73

22 : 79

23 : 83

24 : 89

25 : 97

⇒ We found 25 primes smaller than 100.

How fast are we?

Range [N]	Calc. time [sec]	Output time [sec]	Overall time [sec]	Number of primes found [N]	Primes generated per second
1,000,000	0.535	11.096	11.631	78,498	146,725.2336
2,000,000	1.116	19.561	20.677	148,933	133,452.5090
3,000,000	1.814	28.108	29.922	216,816	119,523.7045
4,000,000	2.364	37.411	39.775	283,146	119,774.1117
5,000,000	2.99	44.2015	47.1915	348,513	116,559.5318
6,000,000	3.588	51.356	54.944	412,849	115,063.8239
7,000,000	4.096	59.184	63.28	476,648	116,369.1406
8,000,000	4.794	66.558	71.352	539,777	112,594.2845
9,000,000	5.622	74.444	80.066	602,489	107,166.3109
10,000,000	6.086	81.584	87.67	664,579	109,197.9954
20,000,000	12.22	134.489	146.709	1,270,607	103,977.6596
30,000,000	18.272	196.413	214.685	1,857,859	101,67.9225
40,000,000	24.668	247.497	272.165	2,433,654	98,656.3159
50,000,000	31.217	287.012	318.229	3,001,134	96,137.8095
60,000,000	37.852	337.553	375.405	3,562,115	94,106.3880
70,000,000	46.457	374.372	420.829	4,118,064	88,642.4866
100,000,000	64.401	497.247	561.648	5,761,455	89,462.1978
120,000,000	78.91	579.739	658.649	6,841,648	86,701.9136
150,000,000	98.454	687.524	785.978	8,444,396	85,769.9636
200,000,000	157.298	865.207	1022.505	11,078,937	70,432.7900

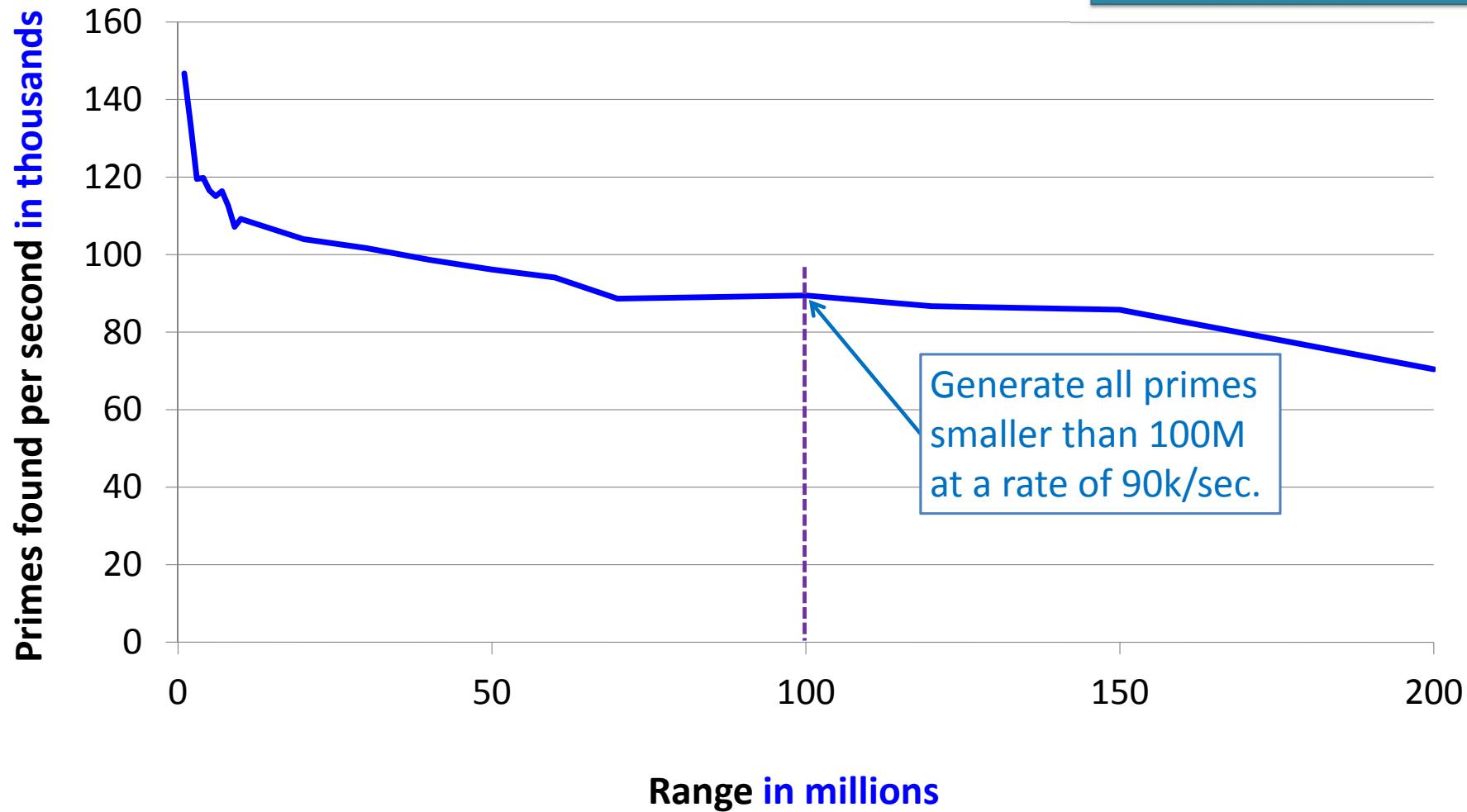
Used machine:
i7-4710HQ (2.5 GHz)
16 GB-RAM

Above a range 200M you may get problems with only 16 GB of RAM.

Also:
ImageJ does not free RAM after completing a computation! Repeated computations may occupy the RAM completely.

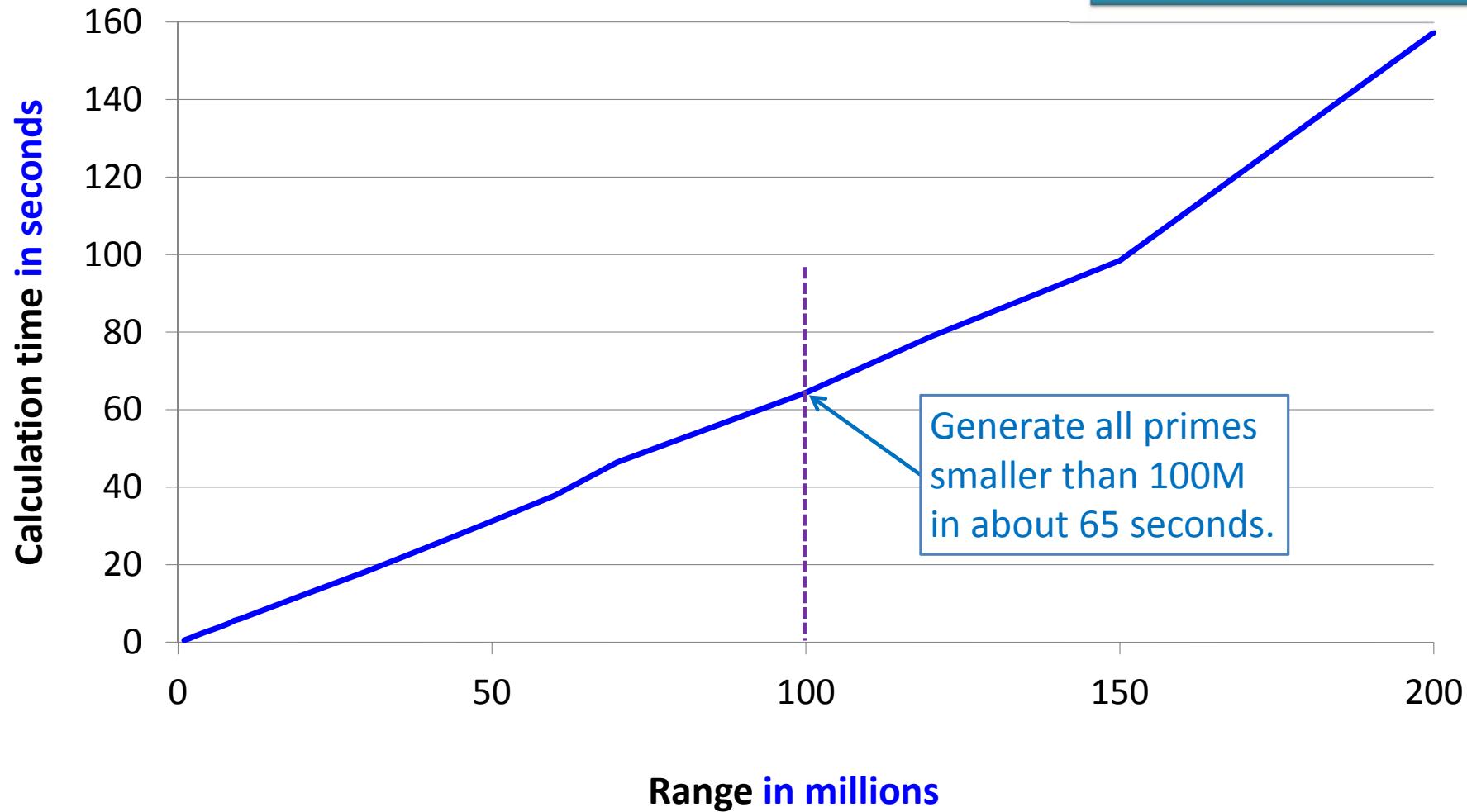
Some plots from the table...

Used machine:
i7-4710HQ (2.5 GHZ), 16 GB-RAM



Some plots from the table...

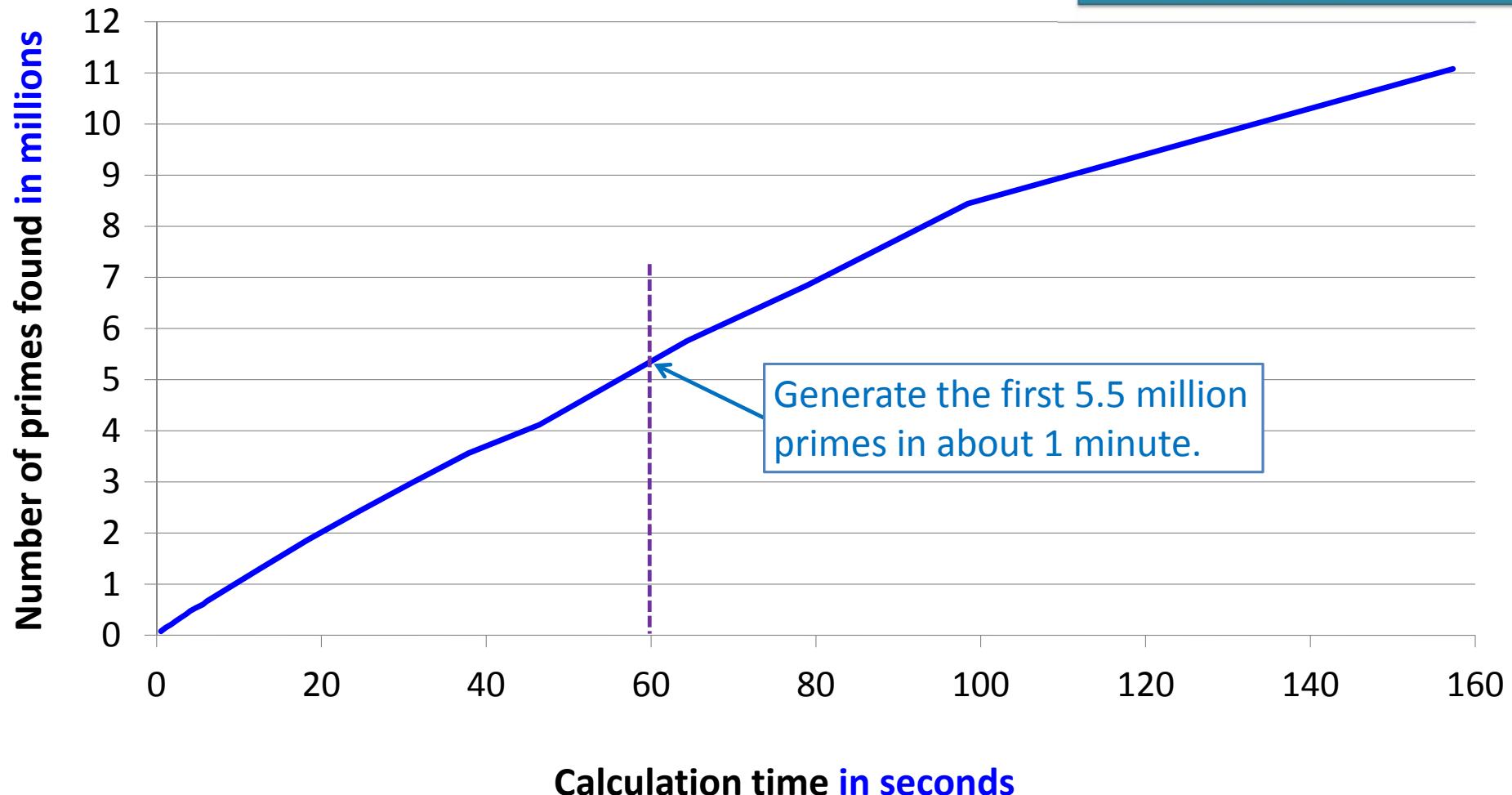
Used machine:
i7-4710HQ (2.5 GHZ), 16 GB-RAM



Some plots from the table...

Used machine:

i7-4710HQ (2.5 GHZ), 16 GB-RAM



The output you get

Prime-#; Prime; [Distance to next prime]; prime/#; #/prime; twin?

1: 2; [1]; 2; 0.5;

2: 3; [2]; 1.5; 0.6667; YES (1.)

3: 5; [2]; 1.6667; 0.6; YES (2.)

4: 7; [4]; 1.75; 0.5714;

5: 11; [2]; 2.2; 0.4545; YES (3.)

6: 13; [4]; 2.1667; 0.4615;

7: 17; [2]; 2.4286; 0.4118; YES (4.)

8: 19; [4]; 2.375; 0.4211;

9: 23; [6]; 2.5556; 0.3913;

10: 29; [2]; 2.9; 0.3448; YES (5.)

11: 31; [6]; 2.8182; 0.3548;

12: 37; [4]; 3.0833; 0.3243;

[...]

78491: 999907; [10]; 12.7391; 0.0785;

78492: 999917; [14]; 12.7391; 0.0785;

78493: 999931; [22]; 12.7391; 0.0785;

78494: 999953; [6]; 12.7392; 0.0785;

78495: 999959; [2]; 12.7391; 0.0785; YES (8169.)

78496: 999961; [18]; 12.739; 0.0785;

78497: 999979; [4]; 12.7391; 0.0785;

78498: 999983; N/A; 12.739; 0.0785

Range: 1,000,000

Calculation time: 0.552 seconds.

Printing time: 10.608 seconds.

Overall time: 11.16 seconds.

Number of primes found: 78,498

Generated primes per second (calculation time): 142,206.522

Highest gap between two primes: 114 (between 492,113 and 492,227)

Average distance between two primes: 12.7391 +/- 10.2825

Twin primes found: 8,169

Gap Toplist:

Gap #1: 114 (between 492,113 and 492,227)

Gap #2: 112 (between 370,261 and 370,373)

Gap #3: 100 (between 838,249 and 838,349)

Gap #4: 100 (between 396,733 and 396,833)

Gap #5: 98 (between 604,073 and 604,171)

Gap #6: 96 (between 860,143 and 860,239)

Gap #7: 96 (between 360,653 and 360,749)

Gap #8: 92 (between 927,869 and 927,961)

Gap #9: 90 (between 843,911 and 844,001)

Gap #10: 90 (between 818,723 and 818,813)

Gap #11: 90 (between 576,791 and 576,881)

If you just want to print the primes and nothing else, this is your code...

```
1 upperlimit=100000000;
2
3 time=getTime();
4
5 large=newArray(round(-1+upperlimit/2));
6
7 for (i = 0; i < large.length; i++) {large[i]=3+2*i;}
8
9 for (p = 0; p<large.length && large[p]<sqrt(upperlimit); p++) {stp=large[p]; strt=((large[p]*large[p])-3)/2-stp; n=1;
10 if(large[p]>2){while(strt+n*stp<large.length){large[strt+n*stp]=0; n++;}}}
11
12 large=Array.concat(large,2);
13 large=Array.sort(large);
14
15 z=0;
16 while (large[z]<1) {z=z+1;}
17
18 large=Array.slice(large,z,large.length);
19
20 calctime=(getTime()-time)/1000;
21 Array.print(large);
22 print("");
23 print("First " + large.length + " primes, calculated in " + calctime + " seconds.");
```

Here, with 16GB of RAM your **upper limit** can be set to **300,000,000**.

And this is what you get...

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, [...]