ORIGINAL PAPER

EVOLUTION OF ALGORITHMS: A CASE STUDY OF THREE PRIME GENERATING SIEVES

NEERAJ ANANT PANDE [1]

Manuscript received: 03.08.2013. Accepted paper: 02.09.2013;

Published online: 15.09.2013.

Abstract. *In this paper, the algorithmic analysis is done with a view of inspecting the evolution of better and better algorithms. This has been with reference to three prime generating sieves. The Three sieves, starting with the simplest one to a nice property user one, have been converted into computer algorithms and using Java programming language, the runtimes of the corresponding sieves have been obtained, compared and contrasted. The third algorithm's performance is surprisingly superior than the first two. The theoretical properties help boost the speed and this is where the mathematics' smartness becomes noticeably appreciable.*

Keywords: *Algorithm, Sieve, Prime Number*

1. INTRODUCTION

Algorithm is a systematic process to get a particular task done (see [5]). In fact everything that we do is always through such algorithms; only thing is that every time we don't deeply realize those explicitly. But in research, the algorithms are intentionally identified, studied and refined for efficiency of the outcome.

Algorithms evolve in due course of time and sometimes reach so great heights that are unparalleled. We present here a case study of the evolution of algorithms from domain of mathematics.

The mathematics community has been curious about the prime numbers from antiquity. Primes have never lost their charm; rather have become more and more appealing with the advances in studies. There are various reasons for that. They are quite irregularly spaced. In one place, we have theoretically proved that there are arbitrarily large gaps between successive primes (see [1]) and at the same time we have a strong conjecture predicting that the number of twin primes in infinite. We concentrate here on the other interesting aspect of the primes; their determination that has been an unending challenge.

What we will be dealing with here is a very preliminary idea, introduced as early as in school days, viz., prime numbers (See [6]).

Definition (*Prime Number or Prime*) :

A positive integer $p > 1$ is said to be prime number or prime if, and only if, it has only two positive divisors, viz., 1 and itself.

[1] Department of Mathematics & Statistics, Yeshwant Mahavidyalaya (College), Nanded, Maharashtra, INDIA.
E-mail: napande@gmail.com

A positive integer larger than 1 which is not a prime number is called composite number.

1 is the only positive integer which is neither prime nor composite.

The Fundamental Theorem of Aritmetic gives the primes the status of being the building blocks of the natural number. Prime numbers enjoy many unique properties. There are many unsettled open problems about these primes and they make primes all-time a mysterious domain till now.

2. DETERMINATION OF PRIMALITY

For a certain positive integer to be prime, by definition, it should have only two positive divisors, and no others. Now both practically as well as theoretically it is impossible to check all infinite integers and then deny the existence of others as divisors!

Many composite numbers can be readily and easily confirmed to be composite, e.g., those (except 2) that are even are always composite (being divisible by 2), those (except 3) whose sum of the digits is perfectly divisible by 3 are composite (being divisible by 3), those (except 5) whose units place digit is 5 or 0 are composite (being divisible by 5) etc. But this is not the case for prime numbers. It is, in general, a tedious task to determine whether a particular number is prime or not.

3. THE THREE SIEVES

Our quest for primes has been in either of the two ways : whether a given particular number is prime or not and to find all prime numbers in a given range of integers.

The methods that generate all primes in a specific range of integers are called as sieves.

3.1. THE DUMB SIMPLEST SIEVE-1

One knows that any positive integer cannot have divisors larger than itself. This immediately hints that while deciding on primality, the task of finding divisors of a positive integer n reduces to inspection of at most the finite set of numbers from 2 to $n-1$. To conclude that n is prime, one needs to check these $n-2$ numbers for divisibility with n and on finding that none of these succeeds in perfectly dividing n, it can be ascertained that n is prime number.

An algorithm implementing this simplest sieve would be like :

```
Take an integer n larger than 1.
For values of integer d from 2 to n–1
     If d divides n perfectly,
          Stop checking and declare that n is not prime
     Else
          Continue checking
If checks do not stop for any value of d till n–1, n is prime.
```

This algorithm is very simple looking. It is based upon only one property that all positive divisors of every positive integer are always less than or equal to itself. If we are able to find even single such divisor between 1 and the number itself except these two, then the number is not prime, else it is prime.

While when a number being tested for primality in this algorithm is composite, the process of determination will terminate at the instance we find first divisor. Now for a composite number, the first divisor may occur as early as 2, which actually happens in case of even numbers, or it may be far ahead of 2 for odd numbers.

But if the number being tested is prime, then the conclusion will be drawn only after all $n - 2$ testing's. Naturally as the value of n being tested for primality increases, the number of testing's significantly increase.

The author has implemented this algorithm in Java programming language (see [3]) and has determined all primes in the range of 1 to 100000 (10^5) and recorded the testing times taken by an electronic computer. The summary of the time requirements for testing all numbers in the ranges of increasing powers of 10 are given in the following table.

Table 1. Runtimes for Sieve 1

All numbers in the Range	The Number of Primes Found	Time Taken in Seconds
1 – 10	4	0.000009799
1 – 100	25	0.000301895
1 – 1000	168	0.010925123
1 – 10000	1229	0.424310380
1 – 100000	9592	32.155770011

The data has been generated in all multiples of 1000 and detail table will follow later. These values will be used in graphs being plotted later during comparison with performance of other algorithms.

For this algorithm, the time requirement grows very rapidly with increasing range. Technically speaking, for a prime number n, the determination that it is prime requires time of order $n - 2$ (denoted as $O(n - 2)$).

As an approximate estimate, it can be seen that in the last range shown in the Table 1, the number of primes increased by about 8 times than those in the previous range but the time required shoots up by about 80 times. The algorithm works correctly, but not effectively.

3.2. AN IMPROVED SIEVE-2

One can make use of one more property that any divisor of any positive integer cannot exceed half of that number. This immediately hints that while deciding on primality, the task of finding divisors of a positive integer n reduces to inspection of the smaller finite set of numbers from 2 to $n/2$. To conclude that n is prime, one needs to check these $n/2 - 1$ numbers for divisibility with n and on finding that none of these succeeds in perfectly dividing n, it can be ascertained that n is prime number.

An algorithm implementing this simplest sieve would be like :

```
Take an integer n larger than 1.
For values of integer d from 2 to n/2
     If d divides n perfectly,
          Stop checking and declare that n is not prime
     Else
          Continue checking
If checks do not stop for any value of d till n/2, n is prime.
```

This algorithm was implemented in Java programming language ([3]) for determining all primes in the range of 1 to 100000 (10^5) and recording the testing times taken by an electronic computer. The summary of the time requirements for testing all numbers in the ranges of increasing powers of 10 are given in the following table.

Table 2. Runtimes for Sieve 2

All numbers in the Range	The Number of Primes Found	Time Taken in Seconds
1 – 10	4	0.000013065
1 – 100	25	0.000216505
1 – 1000	168	0.029273951
1 – 10000	1229	0.264684804
1 – 100000	9592	16.516622393

Here too the complete data is in multiples of 1000 and will be used in comparative plotting.

This algorithm requires almost half testing's as compared to previous algorithm. Technically for a prime number n, the determination that it is prime requires time of order $n/2 - 1$ (denoted as $O(n/2 - 1)$). But there is an extra requirement of determining the half value of the integer being tested at which the testing process will terminate. That is why the first four primes have taken larger time than in the previous algorithm. But later onwards, its better performance over earlier algorithm is clear.

Still in the last range executed, the number of primes increased by about 8 times than those in the previous range but the time required shoots up by about same 80 times!

3.3. FURTHER IMPROVED SIEVE-3

Now we make use of another nice property that any divisor of any positive integer cannot exceed square root of that number. This immediately hints that while deciding on primality, the task of finding divisors of a positive integer *n* reduces to inspection of at most the smaller finite set of numbers from 2 to $\sqrt{n}$. To conclude that *n* is prime, one needs to check these $\sqrt{n} - 1$ numbers for divisibility with *n* and on finding that none of these succeeds in perfectly dividing *n*, it can be ascertained that *n* is prime number.

An algorithm implementing this improved sieve would be like :

```
Take an integer n larger than 1.
For values of integer d from 2 to √n
     If d divides n perfectly,
          Stop checking and declare that n is not prime
     Else
          Continue checking
If checks do not stop for any value of d till √n, n is prime.
```

This algorithm was implemented in Java programming language using ([3]) for determining all primes in the range of 1 to 100000 (10^5) and recording the testing times taken by an electronic computer. The summary of the time requirements for testing all numbers in the ranges of increasing powers of 10 are given in the following table.

Table 3. Runtimes for Sieve 3

All numbers in the Range	The Number of Primes Found	Time Taken in Seconds
1 – 10	4	0.000155847
1 – 100	25	0.000320559
1 – 1000	168	0.001467944
1 – 10000	1229	0.038082080
1 – 100000	9592	0.324932087

Again here also the complete data is in multiples of 1000 and will be used in comparative plotting.

This algorithm requires very few testing's as compared to previous both algorithms. Here the determination that a number *n* it is prime requires time of order $\sqrt{n} - 1$ (denoted as $O(\sqrt{n} - 1)$). But there is an extra requirement of determining the square root of the integer being tested at which the testing process will terminate. That is why the first two ranges of 4 primes and 25 primes have taken larger time than in the previous algorithms. But later onwards, its superior performance over both the earlier algorithms is amazing.

In the last range executed, the number of primes increased by about 8 times than those in the previous range but the time required raised only by about 10 times as compared to 80 times in both previous algorithms! This time is 100 times less than that in first algorithm and 50 times less than in the second one.

4. COMPARATIVE ANALYSIS

Table 4 shows the exhaustive data of runtimes of all these algorithms for ranges of numbers 1 – 10, 1 – 100 and then 1 to all successive multiples of 1000 till 100000.

Table 4. Comparative Run Times for the Three Sieves

Numbers Range	Number of Primes	Time Taken by Sieve-1 in seconds	Time Taken by Sieve-2 in seconds	Time Taken by Sieve-3 in seconds
1-10	4	0.000009799	0.000013065	0.000155847
1-100	25	0.000301895	0.000216505	0.000320559
1-1000	168	0.010925123	0.029273951	0.001467944
1-2000	303	0.036073340	0.036801712	0.003046474
1-3000	430	0.057996841	0.049050130	0.028546513
1-4000	550	0.088084554	0.065033732	0.030299553
1-5000	669	0.125700494	0.085017083	0.031401211
1-6000	783	0.169405181	0.108890065	0.032561661
1-7000	900	0.224389143	0.136230399	0.033995542
1-8000	1007	0.282516633	0.166020417	0.035329570
1-9000	1117	0.349458784	0.199808319	0.036684128
1-10000	1229	0.424310380	0.264684804	0.038082080
1-11000	1335	0.503088469	0.305576321	0.039513628
1-12000	1438	0.587039822	0.348869928	0.040997903
1-13000	1547	0.682914368	0.399121569	0.043155024
1-14000	1652	0.784114760	0.451219571	0.045359739
1-15000	1754	0.918186638	0.513004788	0.047533191
1-16000	1862	1.037553416	0.573774203	0.049419214
1-17000	1960	1.153286263	0.632894631	0.051040204
1-18000	2064	1.281861518	0.699644541	0.053353638
1-19000	2158	1.406150994	0.763103009	0.054983027
1-20000	2262	1.551533410	0.837870149	0.056760798
1-21000	2360	1.695732979	0.911304662	0.058634222
1-22000	2464	1.860656405	0.991781198	0.060436722
1-23000	2564	2.020725724	1.074077965	0.062278884
1-24000	2668	2.195218780	1.164734452	0.064138311
1-25000	2762	2.357384095	1.247637808	0.065985139
1-26000	2860	2.533877025	1.338314359	0.068471218
1-27000	2961	2.721674428	1.434371348	0.070437964
1-28000	3055	2.905076866	1.528329075	0.072826989
1-29000	3153	3.100691326	1.629600391	0.075416654
1-30000	3245	3.300967650	1.727991813	0.079422937
1-31000	3340	3.505987495	1.832609630	0.082030800

Numbers Range	Number of Primes	Time Taken by Sieve-1 in seconds	Time Taken by Sieve-2 in seconds	Time Taken by Sieve-3 in seconds
1-32000	3432	3.716433506	1.941605148	0.084048406
1-33000	3538	3.959300261	2.066486280	0.090329395
1-34000	3638	4.205028246	2.188217352	0.092433324
1-35000	3732	4.433633177	2.305173642	0.095165305
1-36000	3824	4.683053885	2.423181662	0.097254302
1-37000	3923	4.937176119	2.553160002	0.099435687
1-38000	4017	5.185780733	2.681144533	0.101601207
1-39000	4107	5.429387992	2.808020874	0.103732198
1-40000	4203	5.695611662	2.945391659	0.105962110
1-41000	4291	5.946170890	3.073694416	0.108119698
1-42000	4392	6.241444074	3.226797302	0.110436865
1-43000	4494	6.546142240	3.381837538	0.112783895
1-44000	4579	6.808270973	3.517972750	0.116959556
1-45000	4675	7.108639499	3.673172098	0.119287455
1-46000	4761	7.385341282	3.816891530	0.121669481
1-47000	4851	7.679012140	3.968133123	0.123979182
1-48000	4946	7.996065577	4.130157990	0.126365407
1-49000	5035	8.314989172	4.286519655	0.128643846
1-50000	5133	8.656017778	4.462643966	0.131100062
1-51000	5222	8.972996091	4.624688896	0.133474155
1-52000	5319	9.325705721	4.804673442	0.135955568
1-53000	5408	9.656100834	4.972209863	0.138487841
1-54000	5500	10.002780043	5.151245798	0.140959921
1-55000	5590	10.347700146	5.327421902	0.143415671
1-56000	5683	10.714098141	5.512943199	0.146550799
1-57000	5782	11.107297540	5.713492021	0.159018055
1-58000	5873	11.476036966	5.902593123	0.161499934
1-59000	5963	11.847037099	6.094521859	0.163981347
1-60000	6057	12.244367363	6.298696213	0.166549549
1-61000	6145	12.620211804	6.490205003	0.169738337
1-62000	6232	12.996333876	6.683727667	0.172297206
1-63000	6320	13.382392798	6.882349405	0.175528456
1-64000	6413	13.797418643	7.094285290	0.178182513
1-65000	6493	14.157816079	7.279560219	0.181277513
1-66000	6591	14.607762279	7.511887269	0.184584819
1-67000	6675	14.999396680	7.714644540	0.187114292
1-68000	6774	15.468882956	7.958584023	0.191984263
1-69000	6854	15.852152504	8.155694889	0.194461476
1-70000	6935	16.263466962	8.356804572	0.197096403
1-71000	7033	16.747744660	8.605458179	0.200581486

Numbers Range	Number of Primes	Time Taken by Sieve-1 in seconds	Time Taken by Sieve-2 in seconds	Time Taken by Sieve-3 in seconds
1-72000	7128	17.224277625	8.850794215	0.204044172
1-73000	7218	17.683680537	9.086019357	0.207384607
1-74000	7301	18.113278500	9.306093323	0.212236380
1-75000	7393	18.594797621	9.553675135	0.218546765
1-76000	7484	19.081081727	9.800540240	0.229417760
1-77000	7567	19.528563779	10.044869340	0.234111820
1-78000	7662	20.045324281	10.310952562	0.240660640
1-79000	7746	20.510162690	10.549022604	0.244078532
1-80000	7837	21.018556006	10.809555543	0.250687545
1-81000	7925	21.519495751	11.063898015	0.254228621
1-82000	8017	22.046802023	11.334497054	0.258572259
1-83000	8106	22.562145907	11.597799302	0.264022221
1-84000	8190	23.059121365	11.850656100	0.267532968
1-85000	8277	23.574294471	12.116306312	0.270768884
1-86000	8362	24.086909641	12.381037776	0.273658577
1-87000	8450	24.620299061	12.653995977	0.277159991
1-88000	8543	25.191369279	12.948167970	0.280083280
1-89000	8619	25.663514278	13.189677367	0.286502384
1-90000	8713	26.253592173	13.492730746	0.290187641
1-91000	8802	26.819974863	13.797192343	0.293107664
1-92000	8887	27.369111781	14.077469732	0.296063149
1-93000	8984	28.000216976	14.399042252	0.299830529
1-94000	9070	28.564943213	14.688365738	0.303390736
1-95000	9157	29.141564180	14.985249180	0.306962608
1-96000	9252	29.779486960	15.312227334	0.310729988
1-97000	9336	30.353537860	15.604210943	0.315606491
1-98000	9418	30.917208634	15.893170476	0.318798079
1-99000	9505	31.544728891	16.202090497	0.321881414
1-100000	9592	32.155770011	16.516622393	0.324932087

The graphical comparison of the times in seconds taken by all the three algorithms for the range 1 to 10000 is shown in the next figure.

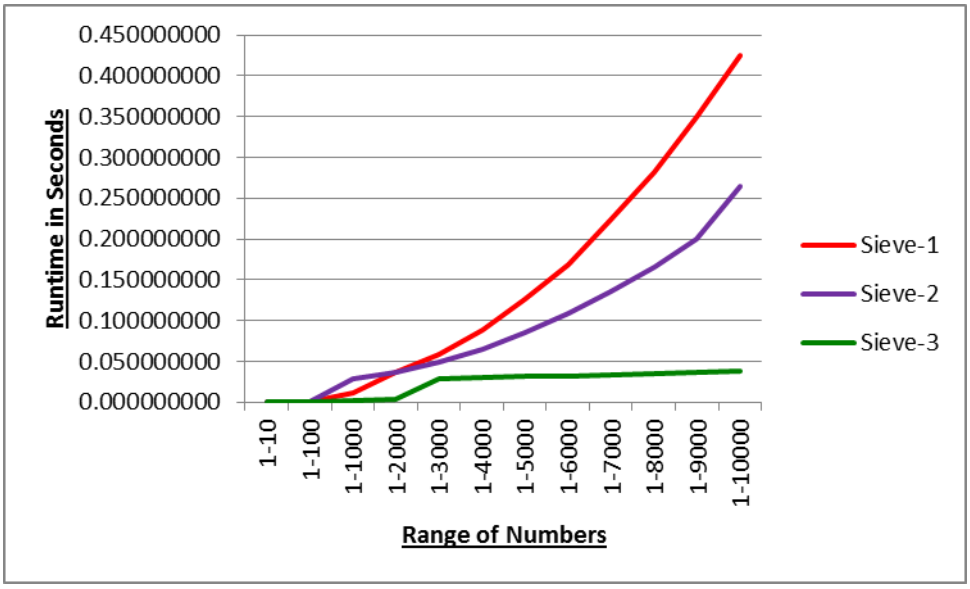

Fig. 1. Runtime Comparison of Three Sieves for Initial Ranges

For the higher range of 1 to 100000, the time required by the third algorithm is so small compared to the other two that its graph looks almost parallel and touching(!) to the horizontal axis.

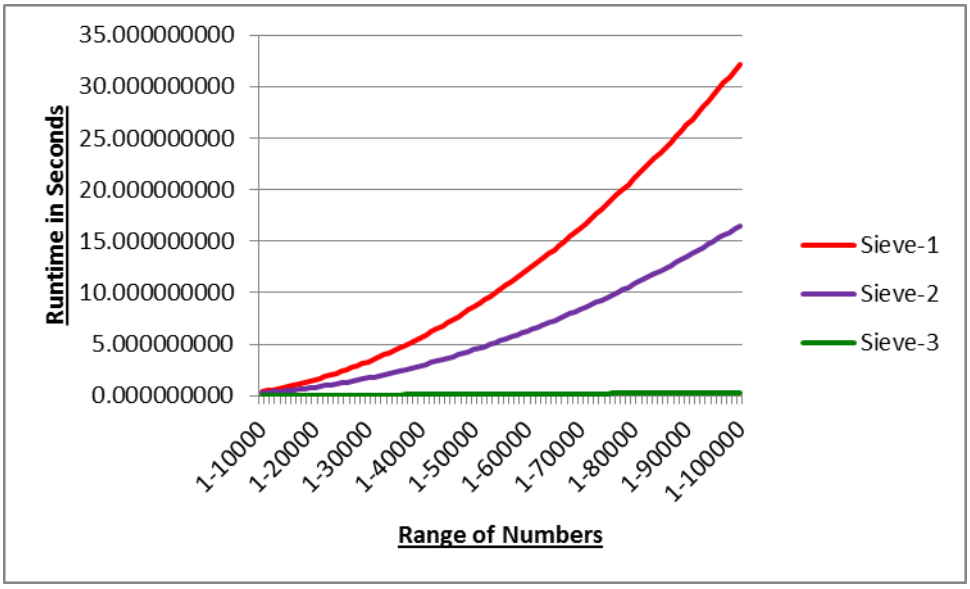

Fig. 2. Runtime Comparison of Three Sieves for Higher Ranges

It is worthwhile to mention that the third algorithm is far better than the first two and its efficiency shows more significant difference for higher ranges of numbers.

Acknowledgements

The author acknowledges the Java (7 Update 25) Programming Language Development Team and the NetBeans IDE 7.3.1 Development Team for the use of their software in implementing the algorithms on their platforms.

Thanks will also be due to University Grants Commission (U.G.C.), New Delhi for funding awaited for this work under a proposed Research Project.

REFERENCES

[1] Brent Richard P., *The First Occurrence of Large Gaps Between Successive Primes*, Mathematics of Computation, 27 (1973), 959-963.

[2] Green, Ben; Tao, Terence, The *Primes Contain Arbitrarily Long Arithmetic Progressions*, Annals of Mathematics, 167 (2), (2008), 481–547.

[3] Herbert Schildt, *Java* : *The Complete Reference*, 7th Edition, Tata McGraw - Hill Education, 2006

[4] Knuth Donald E., *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[5] Kolmogorov Andrei N.; Uspensky Vladimir A., *On the Definition of Algorithm*, Uspekhi Mat. Nauk 13:4 (1958), 3-28, Russian; translated into English in AMS Translations 29 (1963), 217-245.

[6] Niven Evan, Zuckerman Herbert S., Montgomery Huge L., An Introduction to the Theory of Numbers, John Wiley & Sons Inc., U.K., 2008.