Comparison of SS, MR, AKS and Trial Division for Primality

Kevin van As
4076311
MSc Applied Physics

Laurent Verweijen
4030281
MSc Media and Knowledge

ABSTRACT

In this report, we will study common primality tests. We will look at both deterministic primality tests and probablisic primality tests.

Keywords

randomized algorithms, primality, SS, MR, AKS

1. INTRODUCTION

Prime numbers are of interest to mathematicians and cryptographers. Many encryption algorithms, e.g. RSA, rely on the fact that factoring large numbers is difficult. However, since every number may be factorised using prime numbers, we can greatly reduce the computational effort if we know the prime numbers smaller than n, the number to be factorised.

Recently, Angrawal, Kayal and Saxena (AKS) [1] have proven that the primality problem can be solved deterministically in polynomial time. However, in practice randomized algorithms are preferred because of their speed and low error probability.

In this report, we will compare several simple deterministic algorithms (TD, WS) and randomized algorithms (SS, MR) with the AKS algorithm.

2. THEORY

There exist several algorithms to check whether a given integer, n, is a prime number. The easiest among them all is the "trial division" (TD) algorithm, described in Sec. 2.1. It is a deterministic algorithm with a $O(\sqrt{n})$ complexity. Several different algorithms have been devised, both deterministic and random, to beat this complexity. All randomized algorithms below are based on "Fermat's Theorem" for primality [2], which says that "for every prime number, n,

$$a^{n-1} \equiv 1 (\bmod n), \tag{1}$$

$\forall a \in Z_n^*$". Sadly, there does as well exist an infinite set of composite numbers which satisfy this criterion. They are the so-called "Carmichael numbers". Each of the randomized algorithms below deals with these numbers in their own way.

2.1 Trial Division

In the trial division (TD) algorithm, we start with the very definition of a prime number: it is only dividable by 1 and itself. To test this statement, we divide by every integer up to $\sqrt{n}$. If any of those divisions result in an integer, the number is not a prime. Otherwise, it must necessarily be a prime, from the very definition. TD is a deterministic algorithm with $O(\sqrt{n})$ complexity.

2.2 Wheel-Sieve

The Wheel-Sieve (WS) algorithm is an optimized version of TD. The algorithm takes the first k prime numbers (hard-coded, or with a cheap primality algorithm) $p_1 = 2, p_2, \ldots, p_k$ as initial prime numbers. First the algorithm uses trial division for these k numbers to see whether n is dividable by any of these numbers.

Since we only take the first k prime numbers, which may be very far off from n, further steps are required. Let, $m = \prod p_i$, then the algorithm will test divisibility of n by all numbers l, such that $\forall p_i : l \neq p_i (\mod m)$. Unfortunately, this algorithm gives only a linear speedup in comparison with trial division.

2.3 Solovay-Strassen

The Solovoy-Strassen (SS) algorithm is a randomized primality test based on Fermat's Theorem. The algorithm will never error for prime n, but it has a probability of at most $(\frac{1}{2})^k$ of incorrectly identifying a composite number as prime when the algorithm is repeated k times. Each trial of the algorithm has a runtime of $O((\log n)^3)$ as this is the time needed for modular exponentiation.

2.4 Miller-Rabin

The Miller-Rabin (MR) primality test is also based on Fermat's Theorem and will never error for prime n either. On composite numbers it will error with a probability smaller than $\frac{1}{4}$. [3] Again each trial of the algorithm has a runtime of $O((\log n)^3)$ as this is the time needed for modular exponentiation.

2.5 Angrawal-Kayal-Saxena

Angrawal-Kayal-Saxena (AKS) is a polynomial deterministic primality test, published in 2002. They were the first

Table 1: Execution time as a function of $\log n$ in milliseconds

$\log n$	SS	AKS	MR	TD	WS
1	8	1726	13	12	8
2	110	19326	128	8	8
3	110	32415	130	14	16
4	122	689248	140	20	14
5	156	582623	184	23	18
6	169	3735301	190	26	18
7	192	10045207	212	29	20
8	211	21503625	217	32	21
9	234	60103404	237	36	22
10	254	127910717	245	40	25
11	275	292019772	256	46	27
12	297	581977224	269	54	30
13	321	1014476731	278	64	34
14	726.0	-	292.0	78.0	39.0
15	1278.0	-	304.0	94.0	44.0
16	702.0	-	316.0	117.0	52.0
17	749.0	-	330.0	149.0	63.0
18	790.0	-	342.0	189.0	77.0
19	830.0	-	354.0	251.0	94.0
20	873.0	-	369.0	325.0	118.0
21	914.0	-	382.0	1499.0	154.0
22	954.0	-	409.0	1005.0	513.0
23	997.0	-	747.0	1332.0	727.0
24	1037.0	-	772.0	1798.0	624.0
25	1075.0	-	794.0	2399.0	828.0
26	1120.0	-	819.0	3252.0	1112.0
27	1157.0	-	844.0	4438.0	1512.0
28	1197.0	-	871.0	6130.0	2073.0
29	1238.0	-	897.0	7479.0	2841.0
30	1017.0	-	922.0	11676.0	3915.0
31	1318.0	-	947.0	14595.0	5036.

to show that primality is in P. The authors state that the runtime of this algorithm is at most $O((\log n)^{\frac{21}{2}})$ [1].

3. EXPERIMENTS

Experiments have been performed as a function of n in the form of intervals around n. These intervals have a variable length and are given by $[2^{k-1}, 2^k >$ for the k'th interval. For each interval, we sampled 1,000,000 random numbers, and for these numbers we measured the runtime and the fraction of samples that were predicted correctly. For AKS these experiments had to be aborted at a certain point, because our implementation of the algorithm was very slow.

4. RESULTS

Table 1 and Fig. 1 show the runtime as a function of the input size, $\log n$.

Evidently, TD and WS perform well for small n, whereas for big n they are exponential in the input size.

As n increases, SS and MR perform better than TD and WS in terms of execution time. MR seems to outperform SS both in runtime and in error rate. Then again, this conclusion is only valid for our implementation of the algorithms, since both algorithms have about the same time complexity.

Although AKS would theoretically run in polynomial time, it's the snail of the five algorithms, and therefore it has been decided to drop the results for integers above 2^{13}. In the runtime graph, the line for AKS is almost a straight vertical line. We performed an additional test of the AKS algorithm on numbers in the range 2 - 500, and noticed that for composite numbers the algorithm generally finished in less than a second, but when ran on both prime and composite numbers, the algorithm could last up to 10^9s (shown in Table 2). From this it follows that AKS is particularily slow when executed on prime n.

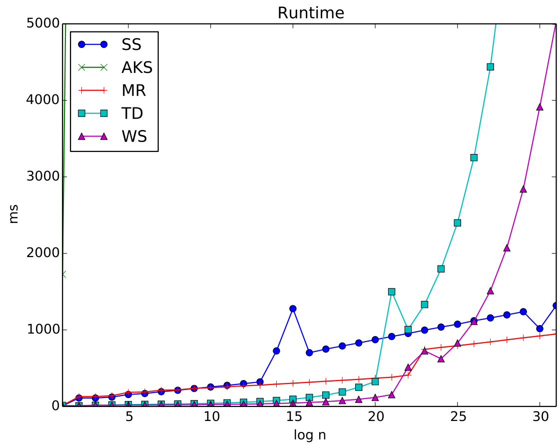

Figure 1: Execution time as function of $\log n$

Table 2: Execution time for AKS in the number range 2-500. Each datapoint consists of 10,000 samples.

Prime numbers included	$O(10^9)$s
Composite numbers only	$O(1)$s

Table 3 and Fig. 2 show the error as a function of the input size. From the figure it follows that for the probablistic methods the error is small for both small and large n. For small n, this is because there are many small primes numbers and for prime numbers these tests never fail. Consequently, the size of the set of numbers the algorithm may potentially fail on is extremely small. For large n, the error is small, because large numbers are more likely to have many divisors and therefore the algorithms are more likely to pick a random number that has factors in common with those big numbers. Evidently, the deterministic algorithms have a 0 error.

5. CONCLUSIONS

We have looked at the behaviour of different common primality tests as a function of the input size. Simple deterministic algorithms (e.g., trial division) work very well for small numbers but become slow when applied to large numbers.

Randomized algorithms scale well with their inputs and have small error percentages. While they are slower than the deterministic algorithms for small numbers, for numbers greater than $\approx 2^{13}$ they outperform the tested simple deterministic algorithms. They also have the nice property that they can be repeated multiple times when even smaller errors are required: pay time to gain a lower error.

AKS might be nice for theoretical reasons (i.e., it proves that primality testing is in P), but in practice the algorithm is rather slow. Maybe the algorithm may be optimized, but in our current implementation, it is more than a factor 10^9 slower than it can be, because it performs very poorly when it is testing a prime number. With our current implementation, it doesn't seem feasible for practical purposes.

Table 3: Error as a function of $\log n$

$\log n$	SS	AKS	MR	TD	WS
1	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0	1.0
4	0.951203	1.0	0.95053	1.0	1.0
5	0.978582	1.0	0.978538	1.0	1.0
6	0.983269	1.0	0.984793	1.0	1.0
7	0.985784	1.0	0.987807	1.0	1.0
8	0.991804	1.0	0.992822	1.0	1.0
9	0.993558	1.0	0.99467	1.0	1.0
10	0.995892	1.0	0.996872	1.0	1.0
11	0.996757	1.0	0.997659	1.0	1.0
12	0.997884	1.0	0.998644	1.0	1.0
13	0.998752	1.0	0.999046	1.0	1.0
14	0.999077	-	0.999402	1.0	1.0
15	0.999396	-	0.999603	1.0	1.0
16	0.999572	-	0.999767	1.0	1.0
17	0.999736	-	0.999844	1.0	1.0
18	0.999832	-	0.999898	1.0	1.0
19	0.999872	-	0.999928	1.0	1.0
20	0.999923	-	0.999947	1.0	1.0
21	0.99994	-	0.999969	1.0	1.0
22	0.999975	-	0.999988	1.0	1.0
23	0.999971	-	0.999994	1.0	1.0
24	0.999983	-	0.99999	1.0	1.0
25	0.999991	-	0.999992	1.0	1.0
26	0.99999	-	0.999997	1.0	1.0
27	0.999996	-	0.999997	1.0	1.0
28	0.999996	-	1.0	1.0	1.0
29	0.999997	-	1.0	1.0	1.0
30	0.999999	-	1.0	1.0	1.0
31	1.0	-	1.0	1.0	1.0

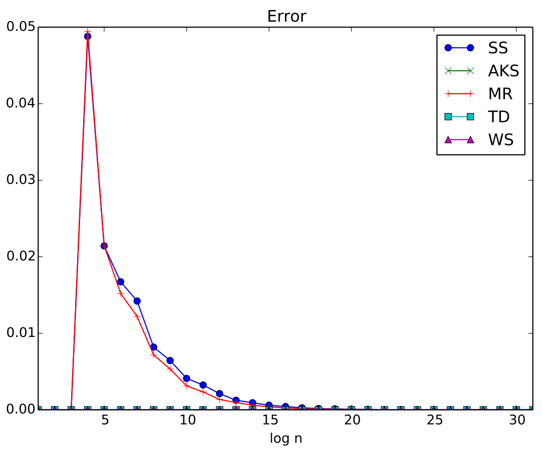

Figure 2: Execution time as function of $\log n$

6. REFERENCES

[1] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. *Ann. of Math*, 2:781–793, 2002.

[2] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 2007.

[3] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128 – 138, 1980.