Computer Science Stack Exchange is a question and answer site for students, researchers and practitioners of computer science. It only takes a minute to sign up.

Sign up to join this community

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

When is the AKS primality test actually faster than other tests?

Asked 6 years, 1 month ago Active 1 year, 3 months ago Viewed 9k times

25

13

I am trying to get an idea of how the AKS primality test should be interpreted as I learn about it, e.g. a corollary for proving that PRIMES ⊆ P, or an actually practical algorithm for primality testing on computers.

The test has polynomial runtime but with high degree and possible high constants. So, in practive, at which n does it surpass other primality tests? Here, n is the number of digits of the prime, and "surpass" refers to the approximate runtime of the tests on typical computer architectures.

I am interested in functionally comparable algorithms, that is deterministic ones that do not need conjectures for correctness.

Additionally, is using such a test over the others practical given the test's memory requirements?

algorithms efficiency primes

edited Mar 31 '14 at 7:48
Raphael ♦
67.5k 26 152 338

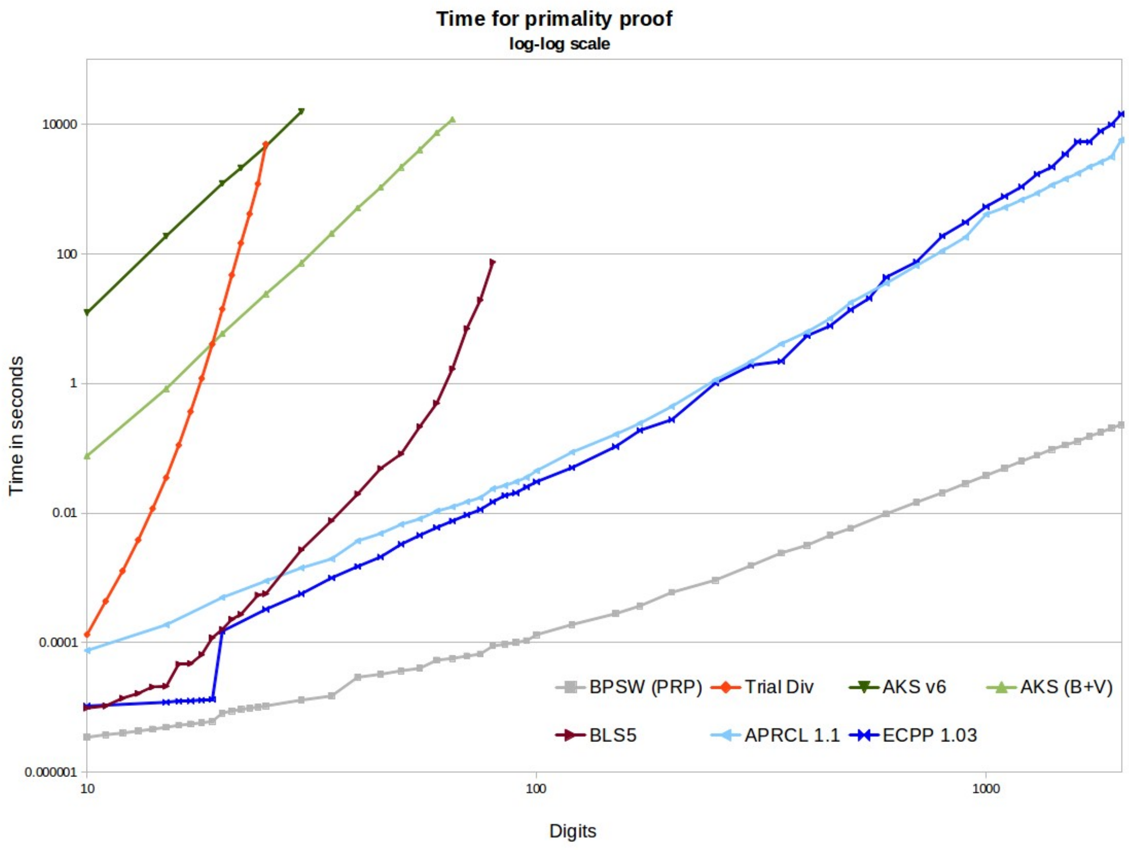
asked Mar 30 '14 at 13:40
Vortico
351 3 5

3 Answers

Active Oldest Votes

24

Quick answer: Never, for practical purposes. It is not currently of any practical use.



First, let's separate out "practical" compositeness testing from primality proofs. The former is good enough for almost all purposes, though there are different levels of testing people feel is adequate. For numbers under 2^64, no more than 7 Miller-Rabin tests, or one BPSW test is required for a deterministic answer. This will be vastly faster than AKS and be just as correct in all cases. For numbers over 2^64, BPSW is a good choice, with some additional random-base Miller-Rabin tests adding some extra confidence for very little cost. Almost all of the proof methods will start out (or they should) with a test like this because it is cheap and means we only do the hard work on numbers which are almost certainly prime.

Moving on to proofs. In each case the resulting proof requires no conjectures, so these may be functionally compared. The "gotcha" of APR-CL is that it isn't quite polynomial, and the "gotcha" of ECPP/fastECPP is that there may exist numbers that take longer than expected.

In the graph, we see two open source AKS implementations -- the first being from the v6 paper, the second including improvements from Bernstein and Voloch and a nice r/s heuristic from Bornemann. These use binary segmentation in GMP for the polynomial multiplies so are pretty efficient, and memory use is a non-issue for the sizes considered here. They produce nice straight lines with a slope of ~6.4 on the log-log graph, which is great. But extrapolating out to 1000 digits arrives at estimated times in the hundreds of thousands to millions of years, vs. a few minutes for APR-CL and

ECPP. There are further optimizations which could be done from the 2002 Bernstein paper, but I don't think this will materially change the situation (though until implemented this isn't proven).

Eventually AKS beats trial division. The BLS75 theorem 5 (e.g. n-1 proof) method requires partial factoring of n-1. This works great at small sizes, and also when we're lucky and n-1 is easy to factor, but eventually we'll get stuck having to factor some large semi-prime. There are more efficient implementations, but it really doesn't scale past 100 digits regardless. We can see that AKS will pass this method. So if you asked the question in 1975 (and had the AKS algorithm back then) we could calculate the crossover for where AKS was the most practical algorithm. But by the late 1980s, APR-CL and other cyclotomic methods was the correct comparison, and by the mid 1990s we'd have to include ECPP.

The APR-CL and ECPP methods are both open source implementations. Primo (free but not open source ECPP) will be faster for larger digit sizes and I'm sure has a nicer curve (I haven't done new benchmarking yet). APR-CL is non-polynomial but the exponent has a factor $\log\log\log n$ which as someone quipped "goes to infinity but has never been observed to do so". This leads us to believe that in theory the lines would not cross for any value of n where AKS would finish before our sun burned out. ECPP is a Las Vegas algorithm, in that when we get an answer it is 100% correct, we expect a result in conjectured $O(\log^{5+\epsilon}(n))$ (ECPP) or $O(\log^{4+\epsilon}(n))$ ("fastECPP") time, but there may be numbers that take longer. So our expectation is that standard AKS will always be slower than ECPP for almost all numbers (it certainly has shown itself so for numbers up to 25k digits).

AKS may have more improvements waiting to be discovered that makes it practical. Bernstein's Quartic paper discusses an AKS-based randomized $O(\log^{4+\epsilon}(n))$ algorithm, and Morain's fastECPP paper references other non-deterministic AKS-based methods. This is a fundamental change, but shows how AKS opened up some new research areas. However, almost 10 years later I have not seen anyone use this method (or even any implementations). He writes in the introduction, "Is the $(\lg n)^{4+o(1)}$ time for the new algorithm smaller than the $(\lg n)^{4+o(1)}$ time to find elliptic-curve certificates? My current impression is that the answer is no, but that further results [...] could change the answer."

Some of these algorithms can be easily parallelized or distributed. AKS very easily (each 's' test is independent). ECPP isn't too hard. I'm not sure about APR-CL.

ECPP and the BLS75 methods produce certificates which can be independently and quickly verified. This is a huge advantage over AKS and APR-CL, where we just have to trust the implementation and computer that produced it.

edited Jan 6 '19 at 21:32
Glorfindel
288 1 4 11

answered Apr 2 '14 at 18:51
DanaJ
544 2 9

18

The (asymptotically) most efficient deterministic primality testing algorithm is due to Lenstra and Pomerance, running in time $\tilde{O}(\log^6 n)$. If you believe the Extended Riemann Hypothesis, then Miller's algorithm runs in time $\tilde{O}(\log^4 n)$. There are many other deterministic primality testing algorithms, for example Miller's paper has an $\tilde{O}(n^{1/7})$ algorithm, and another well-known algorithm is Adleman–Pomerance–Rumley, running in time $O(\log n^{O(\log\log\log n)})$.

In reality, no one uses these algorithms, since they are too slow. Instead, probabilistic primality testing algorithms are used, mainly Miller–Rabin, which is a modification of Miller's algorithm mentioned above (another important algorithm is Solovay–Strassen). Each iteration of Miller–Rabin runs in time $\tilde{O}(\log^2 n)$, and so for a constant error probability (say 2^{-80}) the entire algorithm runs in time $\tilde{O}(\log^2 n)$, which is much faster than Lenstra–Pomerance.

In all of these tests, memory is not an issue.

In their comment, jbapple raises the issue of deciding which primality test to use in practice. This is a question of implementation and benchmarking: implement and optimize a few algorithms, and experimentally determine which is fastest in which range. For the curious, the coders of PARI did just that, and they came up with a deterministic function `isprime` and a probabilistic function `ispseudoprime` , both of which can be found here. The probabilistic test used is Miller–Rabin. The deterministic one is BPSW.

Here is more information from Dana Jacobsen:

Pari since version 2.3 uses an APR-CL primality proof for `isprime(x)` , and BPSW probable prime test (with "almost extra strong" Lucas test) for `ispseudoprime(x)` .

They do take arguments which change the behavior:

- `isprime(x,0)` (default.) Uses combination (BPSW, quick Pocklington or BLS75 theorem 5, APR-CL).
- `isprime(x,1)` Uses Pocklington–Lehmer test (simple $n-1$).
- `isprime(x,2)` Uses APR-CL.
- `ispseudoprime(x,0)` (default.) Uses BPSW (M-R with base 2, "almost extra strong" Lucas).
- `ispseudoprime(x,k)` (for $k \geq 1$.) Does k M-R tests with random bases. The RNG is seeded identically in each Pari run (so the sequence is deterministic) but is not reseeded between calls like GMP does (GMP's random bases are in fact the same bases every call so if `mpz_is_probab_prime_p(x,k)` is wrong once it will always be wrong).

Pari 2.1.7 used a much worse setup. `isprime(x)` was just M-R tests (default 10), which led to fun things like `isprime(9)` returning true quite often. Using `isprime(x,1)` would do a Pocklington proof, which was fine for about 80 digits and then became too slow to be generally useful.

You also write *In reality, no one uses these algorithms, since they are too slow.* I believe I know what you mean, but I think this is too strong depending on your audience. AKS in of course, stupendously slow, but APR-CL and ECPP are fast enough that some people use them. They are useful for paranoid crypto, and useful for people doing things like `primegaps` or `factordb` where one has enough time to want proven primes.

[My comment on that: when looking for a prime number in a specific range, we use some sieving approach followed by some relatively quick probabilistic tests. Only then, if at all, we run a deterministic test.]

In all of these tests, memory is not an issue. It is an issue for AKS. See, for instance, this eprint. Some of this depends on the implementation. If one implements what numberphile's video calls AKS (which is actually a generalization of Fermat's Little Theorem), memory use will be extremely high. Using an NTL implementation of the v1 or v6 algorithm like the referenced paper will result in stupid large amounts of memory. A good v6 GMP implementation will still use ~2GB for a 1024-bit prime, which is a *lot* of memory for such a small number. Using some of the Bernstein improvements and GMP binary segmentation leads to much better growth (e.g. ~120MB for 1024-bits). This is still much larger than other methods need, and no surprise, will be millions of times slower than APR-CL or ECPP.

edited Apr 2 '14 at 22:41

answered Mar 30 '14 at 15:24
Yuval Filmus
223k 16 217 390

2 I do not believe this answers the question as posed, which would require calculation of the constants of these tests. – jbapple Mar 30 '14 at 15:32

1 *Use your downvotes whenever you encounter an egregiously sloppy, no-effort-expended post, or an answer that is clearly and perhaps dangerously incorrect.* — I can't see how the person downvoting this answer justifies the voting. – Pål GD Mar 30 '14 at 15:43

2 @PålGD: Probably because it just does not answer the question (before the edit, that is). Also, do you use the same n as the OP, Yuval? – Raphael ♦ Mar 31 '14 at 7:52

@Raphael You're right, their n is my $\log n$. – Yuval Filmus Mar 31 '14 at 14:30

Good post, but your definition of "no one" is ever so slightly off. Out of curiosity, I tested how long it takes to verify a 2048 bit DSA probable prime generated with OpenSSL (using `openssl pkeyparam -text` to extract the hex string) using PARI's `isprime` (APR-CL as stated): about 80s on a fast notebook. For reference, Chromium needs slightly over 0.25s for each iteration of my JavaScript demo implementaton of the Frobenius test (which is much stronger than MR), so APR-CL is certainly paranoid but doable. – Arne Vogel May 16 '19 at 11:32

2

this is a complicated question because of what are known as "large/galactic constants" associated with the inflections between efficiencies of different algorithms. in other words the $O(f(n))$ associated with each different algorithm can hide very large constants such that a more efficient $O(f(n))$ over another $O(g(n))$ based on asymptotic/function complexity only "kicks in" for very large n . my understanding is that AKS is "more efficient" (than competing algorithms) only for "much larger n " out of range of current practical use (and that precise n is actually very difficult to calculate exactly), but theoretical improvements on the algorithm implementation (actively sought by some) could change that in the future.

saw this recent paper on arxiv which analyzes this topic in depth/detail, not sure what people think of it, have not heard reactions so far, it seems maybe a student-created thesis, but possibly one of the most detailed/comprehensive analyses of *practical* usage of the algorithm available.

- Deterministic Primality Testing - understanding the AKS algorithm Vijay Menon
- Powerful algorithms too complex to implement tcs.se

edited Apr 13 '17 at 12:32
Community ♦
1

answered Mar 30 '14 at 16:39
vzn
10.5k 1 20 44

AKS is more efficient than what? What is the competition? – Yuval Filmus Mar 30 '14 at 17:41

all other algorithms. mainly probabilistc? details in the paper – vzn Mar 30 '14 at 20:44