Design and Implementation of a Perl number theory module

FOSDEM 2015

Dana Jacobsen, 1 Feb 2015

Intro

Number Theory

Primes, primality, factoring

Why do we care?

What does Perl/ntheory offer

Simple Example (1a)

(2012) Send your resume to:

```
{ the first 7-digit palindromic prime found in
consecutive digits of pi } @scoopshot.com
```

Simple Example (1b)

```
# Show all 7 digit prime palindromes in Pi(20k)

my $pi = "".Pi(20000);  # 20,000 digits of pi
for my $pos (2 .. length($pi)-7) {
  my $s = substr($pi, $pos, 7);
  say "$s at $pos"
      if $s eq reverse($s) && is_prime($s);
}
```

Output: 9149419 at 13902 / 1532351 at 17088

Simple Example

Project Euler 211: sum n < 64M where the sum of the squares of divisors is a perfect square

```
my $sum = 0;
for my $i (0..64_000_000-1) {
  $sum += $i if is_power(divisor_sum($i, 2), 2);
}
say $sum;
```

History of Perl/ntheory

2012 May:

dissatisfaction with existing modules

simple sieve and ~10 functions for native ints

released as Math::Prime::Util

2012 - 2015:

bigints, more speed, more features

> 120 functions

Why Perl

- I was already using Perl

- Contrast with Pari/GP:
  Pari/GP has great math, adding language
  Perl has great language, adding math

- Math::Pari, Pari/GP, SymPy, FLINT, SAGE, Mathematica, ...

Design (initial)

functional vs. OO

input validation

faster than existing modules

one module vs. many

Perl with C code included (XS)

Design (current prescriptive)

Correct

Useful

Lightweight

Fast

Portable

Design: Correct

> 7000 tests in install suite

Many additional tests

Test all configurations (PP, XS, GMP, MPFR)

TravisCI

Compare to other software

Found issues in Crypt::Primes, Math::Pari, Math::Big, Math::BigInt::GMP, FLINT, Perl6, ...

Design: Useful

Features of 10+ other modules

Used to make new Crypt:: modules

Extended some OEIS sequences

factordb.com uses ECPP verifier (elliptic curve primality proofs)

RosettaCode, Project Euler, StackOverflow

Standalone C versions of many parts

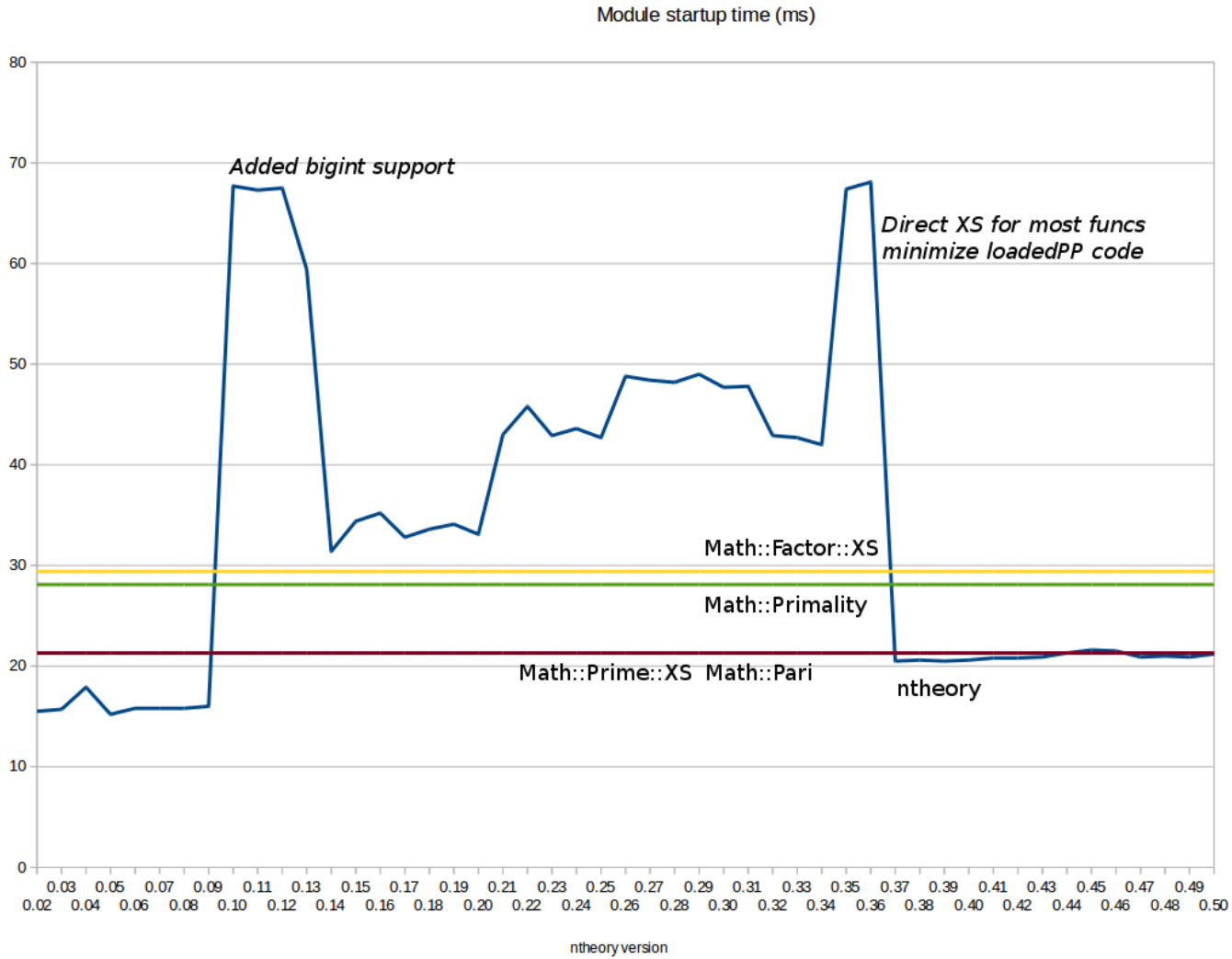
Design: Lightweight

Startup cost and memory use

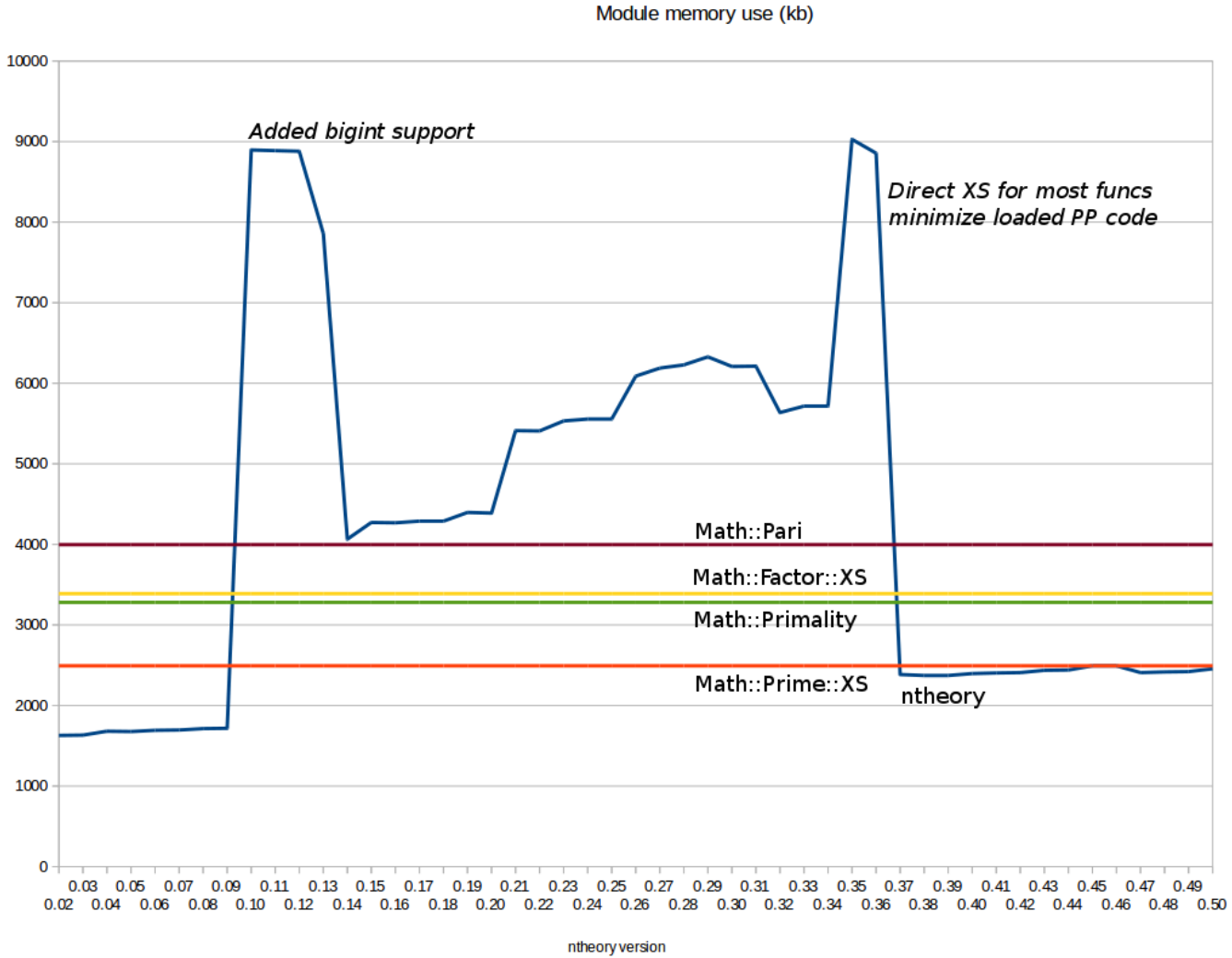
Important for scripts

Less than most other modules

CPU


Module startup time (ms)
80
70
60
50
40
30
20
10
0
Added bigint support
Direct XS for most funcs
minimize loadedPP code
Math::Factor::XS
Math::Primality
Math::Prime::XS
Math::Pari
ntheory
0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.50
ntheory version

Memory


Module memory use (kb)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Added bigint support
Direct XS for most funcs
minimize loaded PP code
Math::Pari
Math::Factor::XS
Math::Primality
Math::Prime::XS
ntheory
0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45 0.46 0.47 0.48 0.49 0.50
ntheory version

Fast - Prime generation

Much faster than other Perl modules

faster than Pari/GP

about as fast at Bernstein’s primegen

2x slower than primesieve and yafu

Fast - primality testing (64-bit)

25x faster than Math::Pari

5 - 24,000x faster than Math::Prime::XS

2000x faster than Perl6 native is-prime

2x faster than Pari/GP 2.8

Fast - primality testing (100 digits)

4x faster than Math::Pari's weak test

5x faster than Math::Primality

10x faster than Perl6 native is-prime

2x faster than Pari/GP 2.8

20x faster than OpenPFGW's weak test

OpenPFGW faster for ~5000 digits, much faster for 50k+,
Relatively weak test (great pre-test)

Fast - Primality Proofs

Primality proofs:

- 100 digit proofs with certificates in milliseconds
- Fastest AKS implementation
- Fastest open source ECPP

Good alternatives: Primo, mpz_aprcl, Pari/GP

Fast - Factoring

Much faster than other modules for 64-bit

Faster than Pari/GP for 64-bit

Similar to Pari/GP for 20+ digits

State of the art:

yafu, msieve, gmp-ecm, CADO-NFS, etc.

Fast - Examples

OEIS A066265: Added new terms to semiprime counts

OEIS A181671: Added more Ramanujan primes

OEIS A067836: Frank Buss conjecture, many more terms

Lucas pseudoprimes to 10^14

Prime gaps: 48% of all record prime gaps

Not fastest at everything: partitions, RiemannZeta

Portability

32-bit, 64-bit, big/little endian

Linux, Solaris, *BSD, Win32, Cygwin, AIX

Perl back to 5.6 (ouch)

Can run entirely in pure Perl

Total non-CORE chain: 4 modules

Tries to use GMP, Math::BigInt::{GMP,Pari}, Math::MPFR

Issues: threading

All functions should be thread-safe

Internals are not multi-threaded

Win32…. <argh>

Issues: bigints

CORE Math::BigInt -- very slow

Math::BigInt::GMP -- better

Math::GMPz, Math::GMP, Math::Pari, …

bugs in modules, including core modules

C+GMP -- very fast

Ordering (Perl, XS, GMP)

Perl	Perl -> {XS,GMP}	XS -> {GMP,Perl}
Input Validation (Perl) Native (Perl) BigInt (Perl)	Input Validation (Perl) Call native XS if possible Call bigint GMP if possible Native (Perl) BigInt (Perl)	Input Validation (XS) Native (XS) if possible call direct to GMP if possible load and call Perl code: Native (Perl) BigInt (Perl)
Simple but slow	overhead is still very large	Tricky XS code Lowest overhead by far Front end for Pure Perl

Optimizing at different sizes

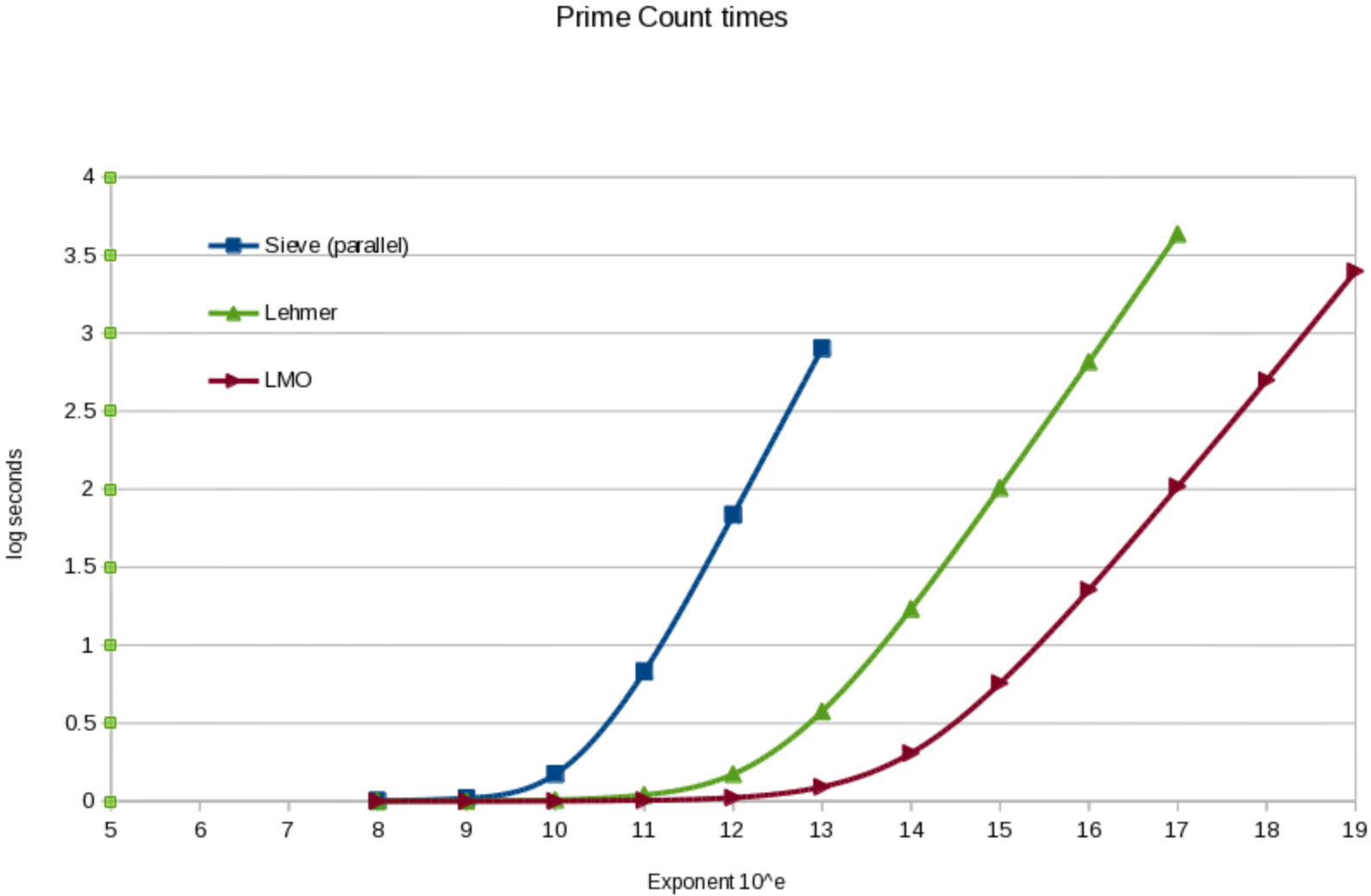
What is a big number?

primes [2,n] vs [a,b]

nth_prime and nth_prime_approx

Primes: Count 1000M primes

TIME	MODULE	NOTES
~500s	*Python SymPy primepi*	uses ~3 GB
90s	ntheory (PP)	PP string sieve
71s	Bit::Vector	XS sieve
46s	Math::Prime::XS	XS sieve
10.3s	Math::Prime::FastSieve	XS sieve
2.8s	*Pari/GP primepi*	(no tables)
0.5s	ntheory (private function)	mod-30 sieve (no tables)
0.24	*primesieve*	fastest sieve software
0.07s	ntheory (PP)	PP Lehmer
0.001s	ntheory	LMO


Prime Count times
Sieve (parallel)
Lehmer
LMO
log seconds
Exponent 10^e
0
0.5
1
1.5
2
2.5
3
3.5
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Lists, blocks, Iterators

```
my $aref = primes(2**27, 2**28);

forprimes { … } 1e9

forcomposites { … } 10**22,10**22+10000

my $it = prime_iterator;

my $itobj = prime_iterator_object;

tie my @pr, 'Math::Prime::Util::PrimeArray';
```

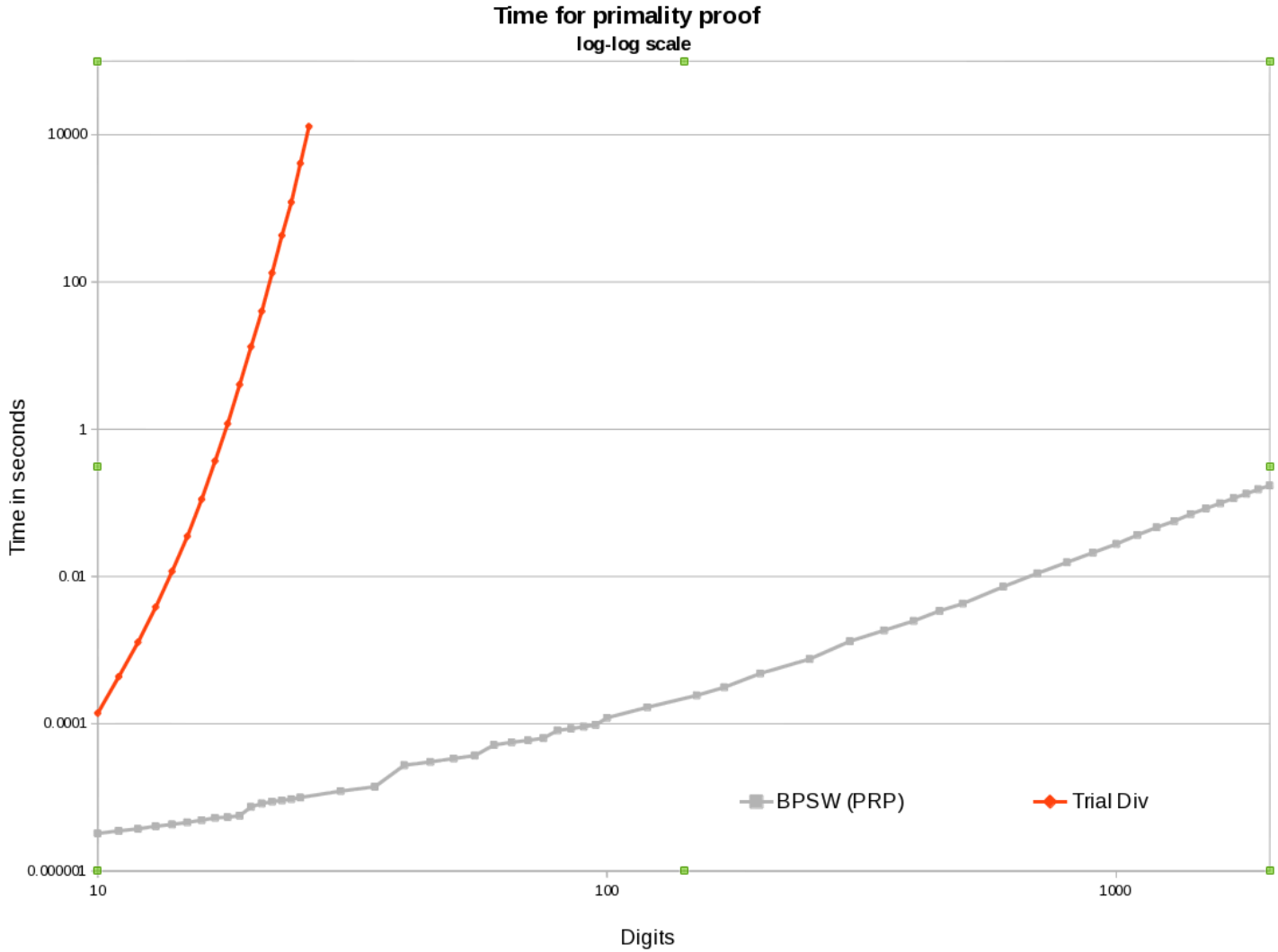
Primality

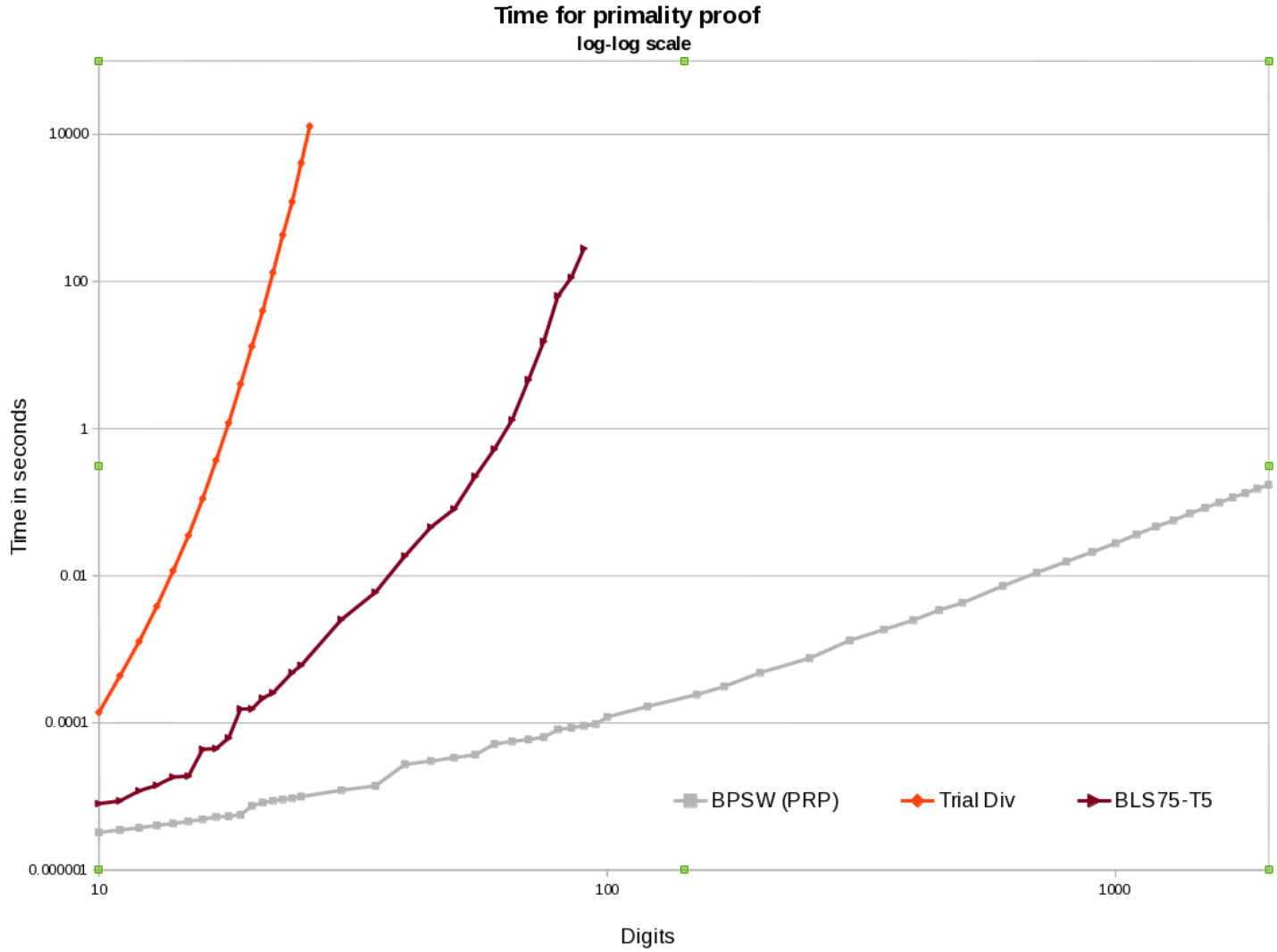
11 compositeness tests

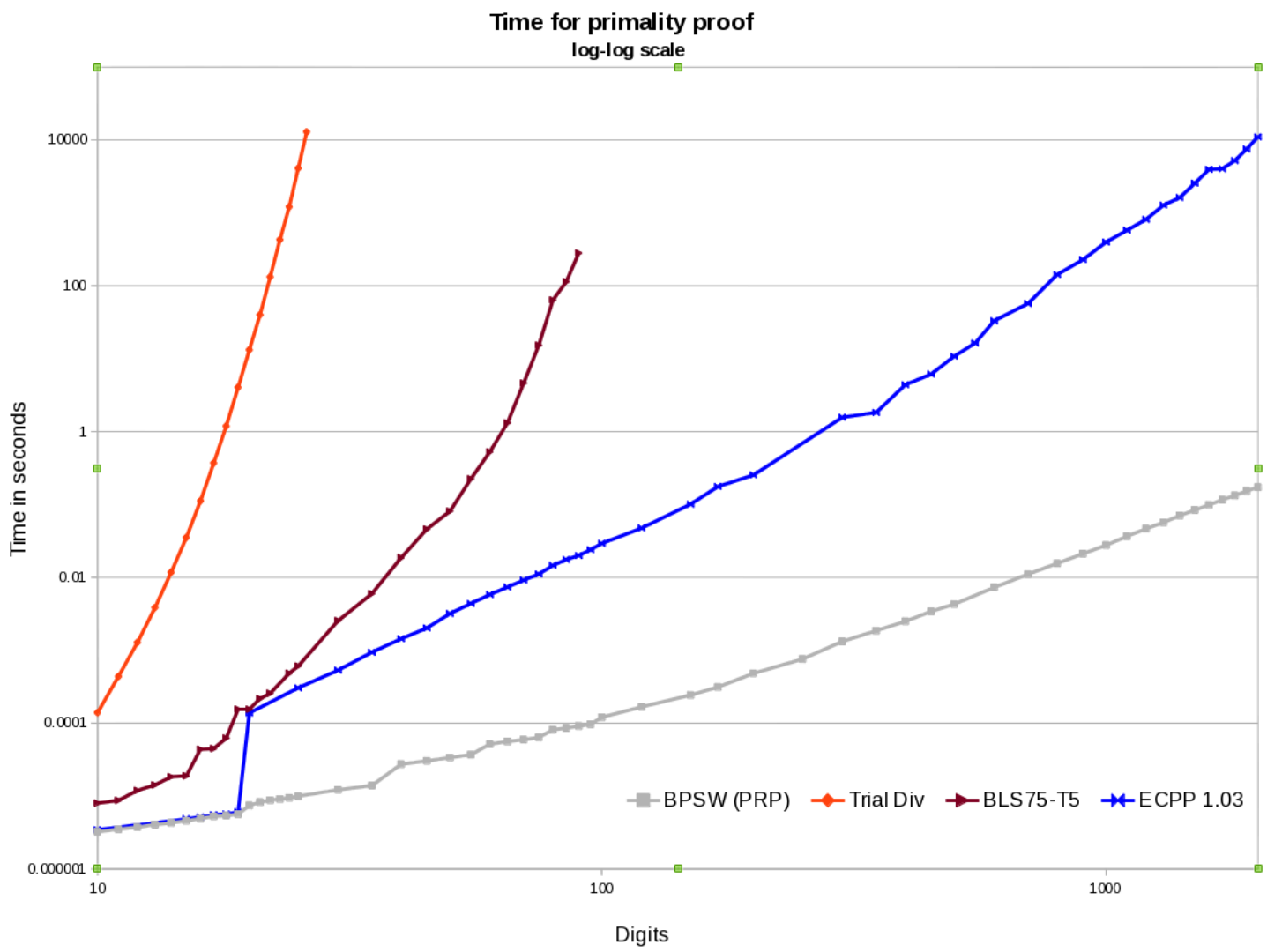
3 proof methods (BLS75, ECPP, AKS)

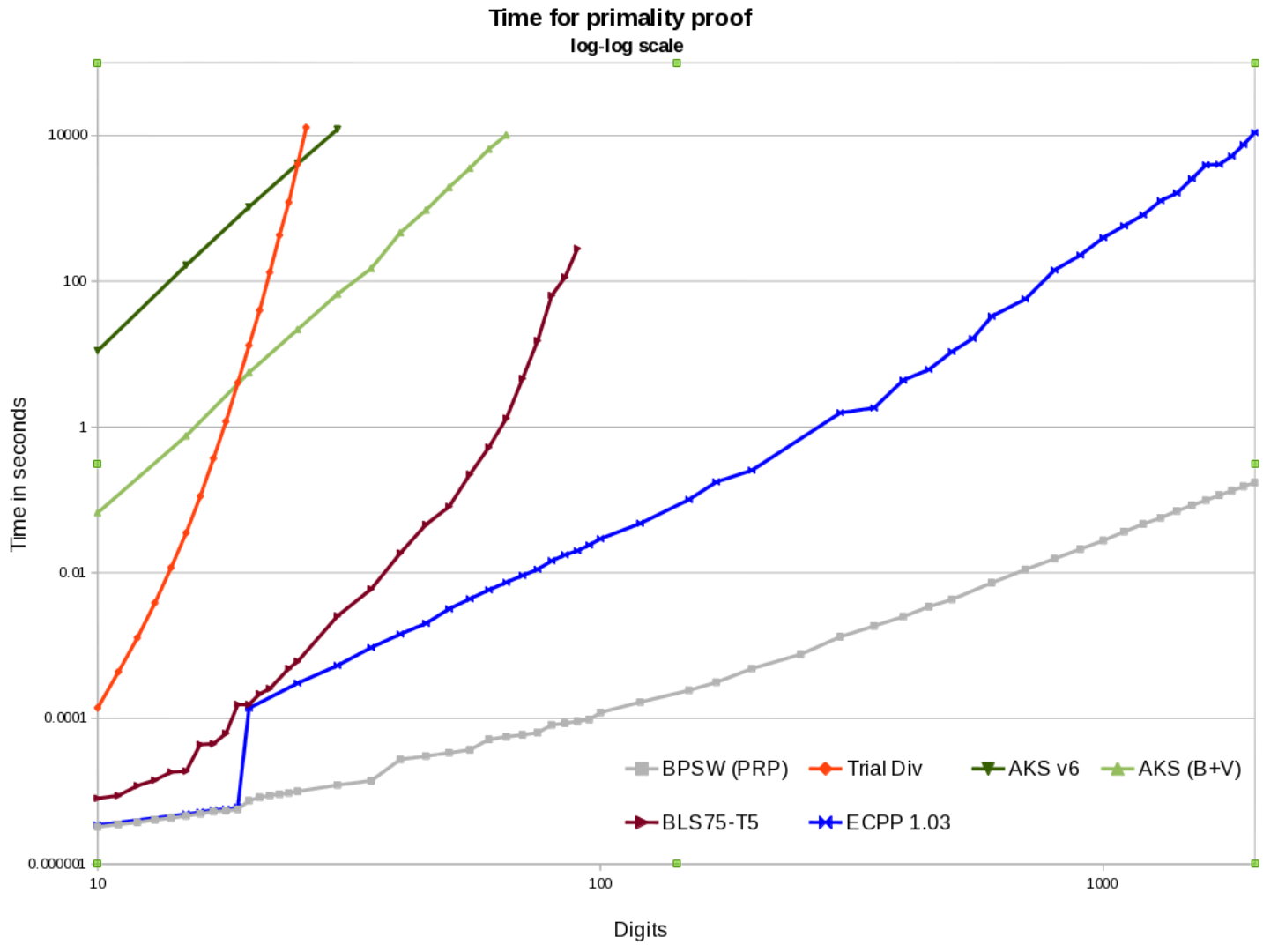
provable prime or *probable* prime

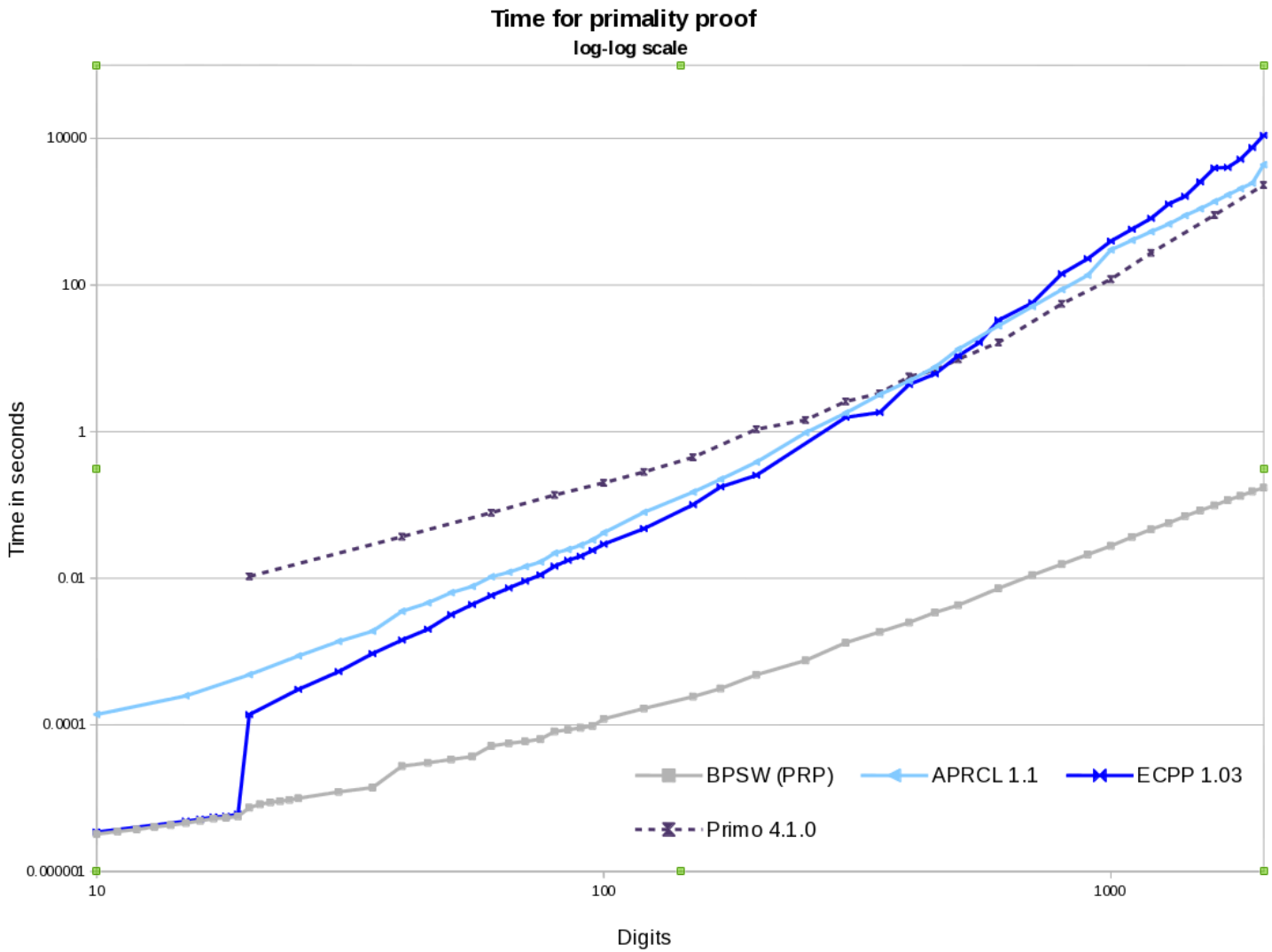
What should is_prime() do?


Time for primality proof
log-log scale
Time in seconds
10000
100
1
0.01
0.0001
0.000001
10
100
1000
Digits
BPSW (PRP)
Trial Div


Time for primality proof
log-log scale
Time in seconds
10000
100
1
0.01
0.0001
0.000001
10
100
1000
Digits
BPSW (PRP)
Trial Div
BLS75-T5


Time for primality proof
log-log scale
Time in seconds
10000
100
1
0.01
0.0001
0.000001
10
100
1000
Digits
BPSW (PRP)
Trial Div
BLS75-T5
ECPP 1.03


Time for primality proof
log-log scale
Time in seconds
10000
100
1
0.01
0.0001
0.000001
10
100
1000
Digits
BPSW (PRP)
Trial Div
AKS v6
AKS (B+V)
BLS75-T5
ECPP 1.03


Time for primality proof
log-log scale
Time in seconds
10000
100
1
0.01
0.0001
0.000001
10
100
1000
Digits
BPSW (PRP)
APRCL 1.1
ECPP 1.03
Primo 4.1.0

Factoring

methods:

- Pollard Rho
- p-1, p+1
- Fermat, Hart's OLF
- SQUFOF
- ECM
- Quadratic Sieve

also znlog(a,g,p) for solving $a = g^k \bmod p$

Combinatorics

forpart { … } $n	integer partitions
forcomb { … } $n,$k	combinations
forperm { … } $n	permutations

Misc

vecsum	vecprod	vecmin	vecmax	vecextract
vecreduce	is_power	gcd	lcm	gcdext
chinese	primorial	pn_primorial	factorial	binomial
partitions	sqrtint	valuation	hammingweight	kronecker
invmod	moebius	mertens	euler_phi	jordan_totient
carmichael_lambda	exp_mangoldt	liouville	znorder	znprimroot
chebyshev_theta	chebyshev_psi	consecutive_integer_lcm	lucasu	lucasv
lucas_sequence	bernfrac	bernreal	harmfrac	harmreal
stirling		ExponentialIntegral	LogarithmicIntegral	RiemannZeta
RiemannR	LambertW	Pi		

Examples

```
forprimes { say } 1e5;

forprimes { say if is_prime(2*$_+1) } 1e5;

forcomposites {
  say if divisor_sum($_)+6 == divisor_sum($_+6)
} 9,1e7;
```

Examples

```
say vecsum( @{primes(2_000_000)} );

say join “ “, factor(82374872347);

for (1..100) { say “F$_=”, lucasu($_,1,-1) }

say “Bernoulli(56) = “, join(“/”,bernfrac(56));
say “Harmonic(56) = “, join(“/”,harmfrac(56));
```

Examples

```
use bigint;

say “10^14th prime is “, nth_prime(10**14);

say “10^100th ~ “, nth_prime_approx(10**100);

say “π(10^14) = “, prime_count(10**14);

say “π(10^100) ~”, prime_count_approx(10**100);

say “π2(10^14) = “, twin_prime_count(10**14);
```

Examples

```
say random_ndigit_prime(100);
say random_nbit_prime(32);
say random_proven_prime(128);
my ($n, $cert) =
     random_proven_prime_with_certificate(2048);
die unless verify_prime($cert);
```

Examples (Lucas-Carmichael #s)

```
# positive composite, odd, square-free,
# and p+1 divides n+1 for all prime factors of n.

use ntheory qw/:all/;
foroddcomposites {
  my $n = $_;
  say unless moebius($n) == 0
    || vecsum(map { ($n+1) % ($_+1) != 0 } factor($n));
} 1e10;
```

Thank you, and Questions

Random Primes

uniformity

custom RNG if desired

use in modules

random proven primes