Improved Arithmetic in the Ideal Class Group of Imaginary Quadratic Number Fields

With an Application to Integer Factoring

Maxwell Sayles

Department of Computer Science

University of Calgary

May 22, 2013

Major Goals

- Improve the performance of arithmetic in the ideal class group.
- SuperSPAR integer factoring algorithm.

Arithmetic in the Ideal Class Group

Based on NUCOMP, NUDUPL, and NUCUBE
(Shanks, Jacobson, Williams, Imbert, Schmidt).

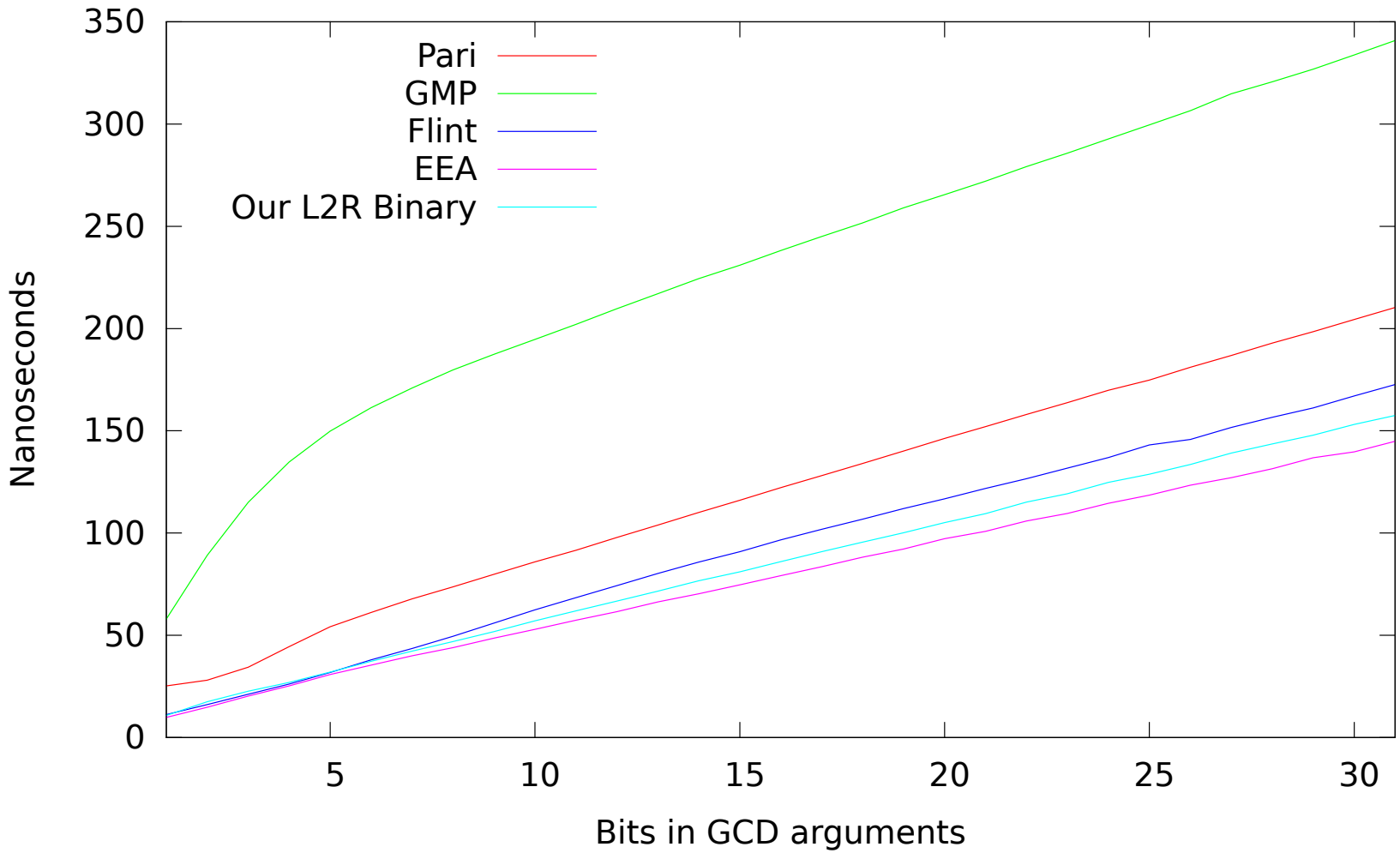
- Implementations for 64-bit and 128-bit arithmetic.
  - A reference implementation using GMP.
- Full/Partial extended greatest common divisor.

XGCD Experiments

- 32-bit, 64-bit, 128-bit implementations.
- Pari, GMP, and Flint as reference implementations.
- 1,000,000 pseudorandom positive integer pairs (a, b) for each bit size.

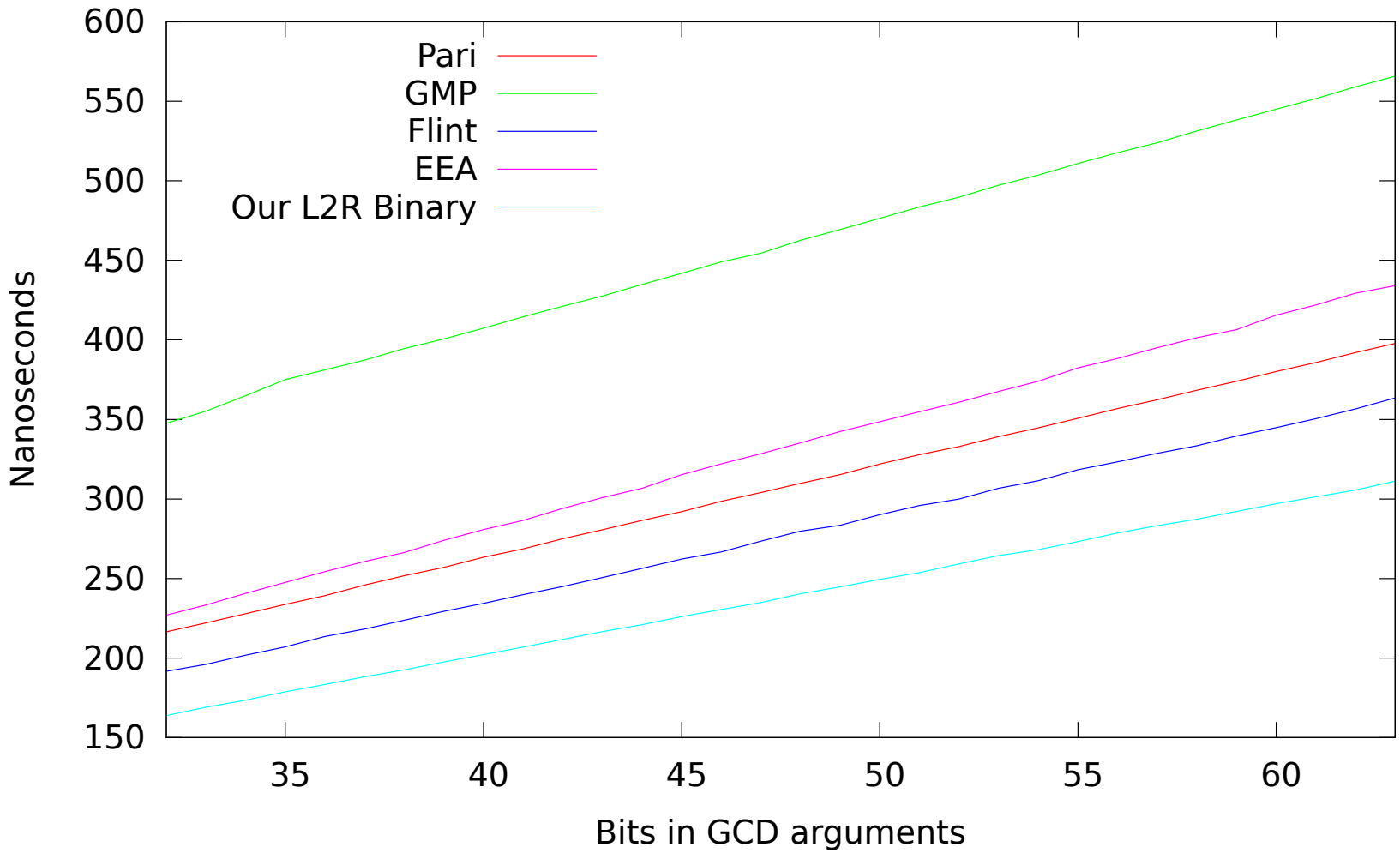
XGCD Results

32-bit



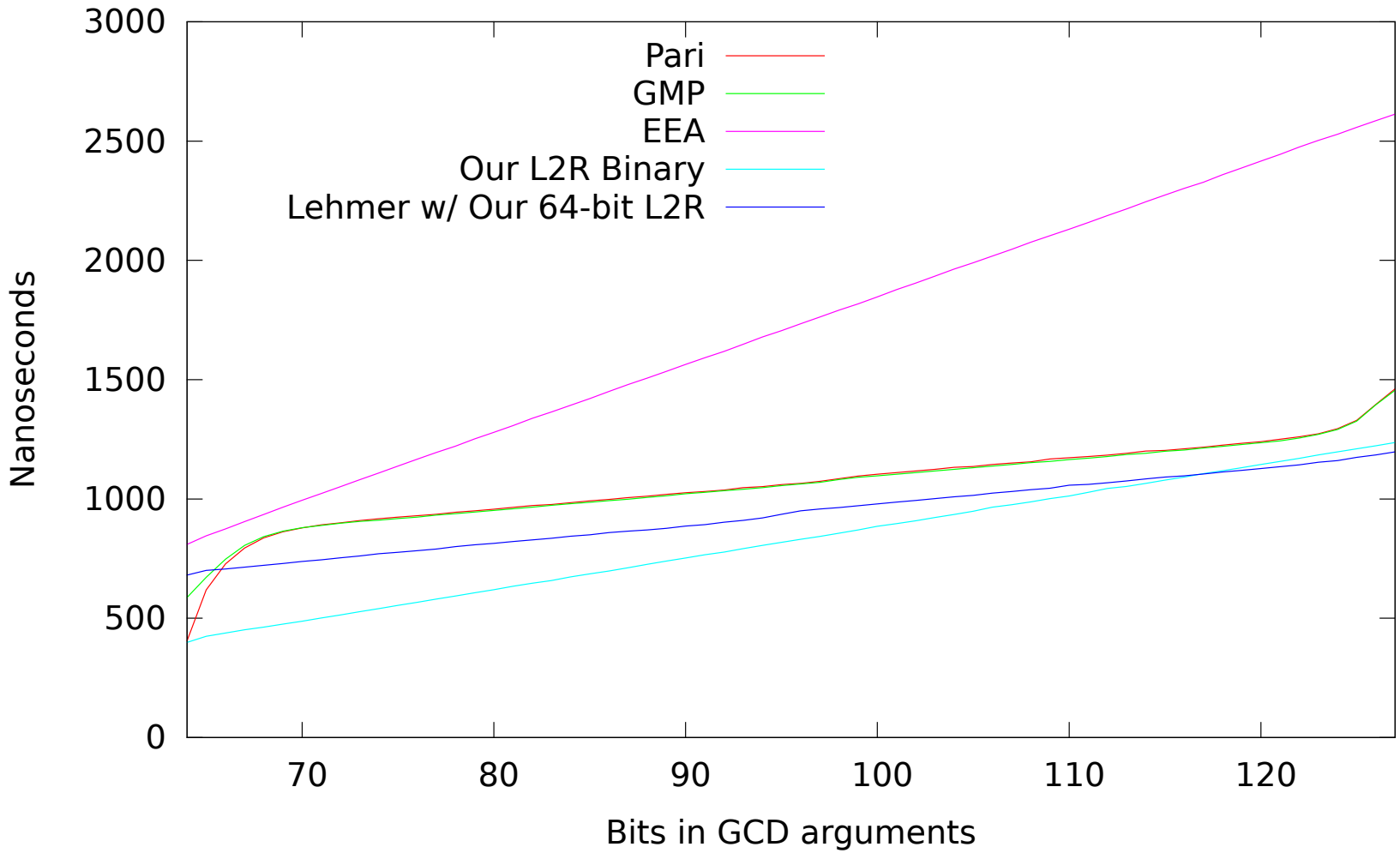
XGCD Results

64-bit



XGCD Results

128-bit



XGCD Improvement

How much faster than the reference implementations?

Bit Range	Algorithm	GMP	Pari	Flint
1 – 31	EEA	3.44	1.60	1.17
32 – 63	Our L2R Binary	1.94	1.29	1.16
64 – 118	Our L2R Binary	1.38	1.38	–
119 – 127	Lehmer w/ Our 64-bit L2R	1.12	1.13	–

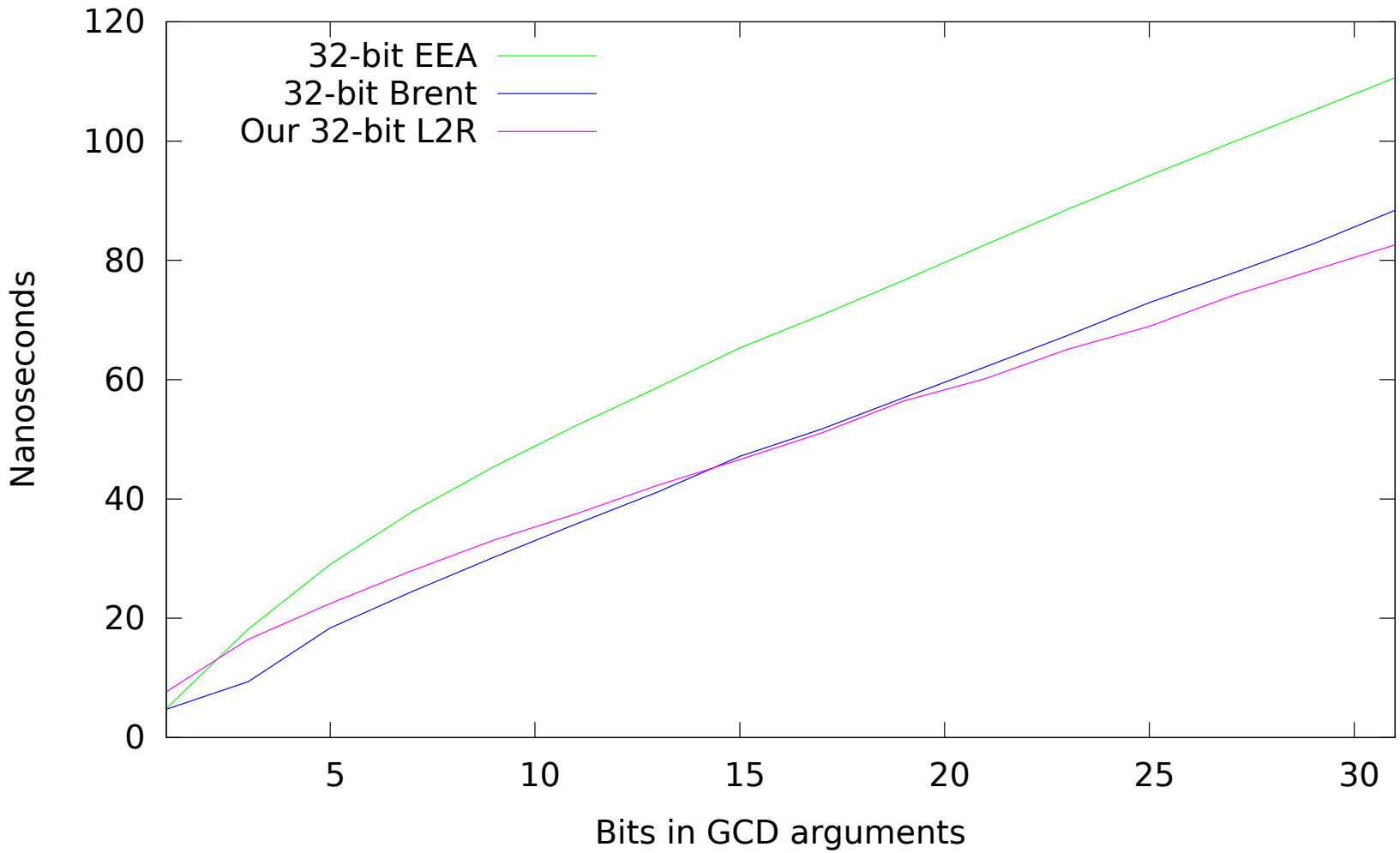
All times are based on the average. The measurement of improvement is an average over the bit range.

Partial XGCD Experiments

- Operations must be unimodular.
- 32-bit, 64-bit, 128-bit implementations.
- 1,000,000 pseudorandom positive integer triples (a, b, T) with $0 \leq b \leq a$ and $0 \leq T \leq \sqrt{a}$.

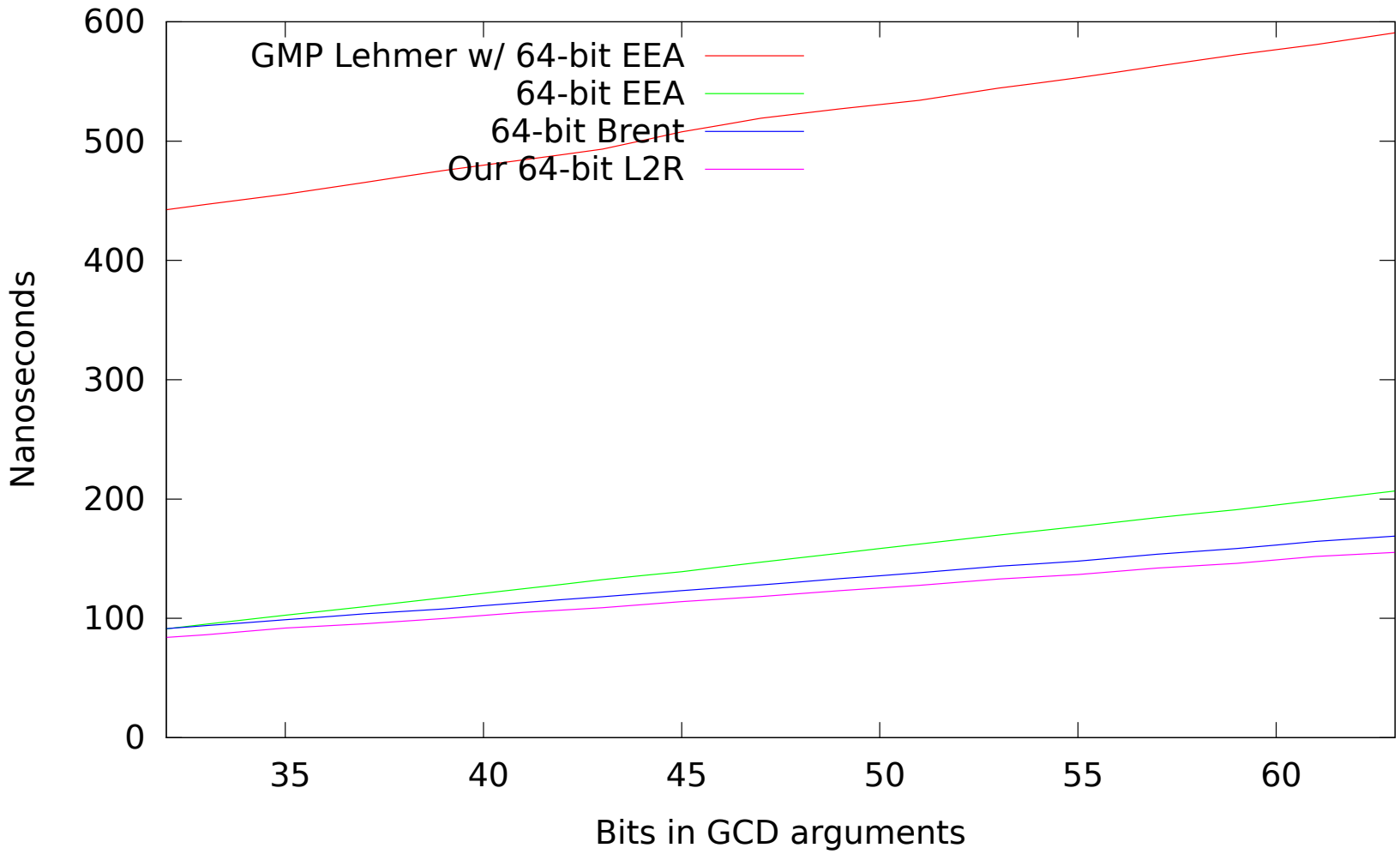
Partial XGCD Results

32-bits



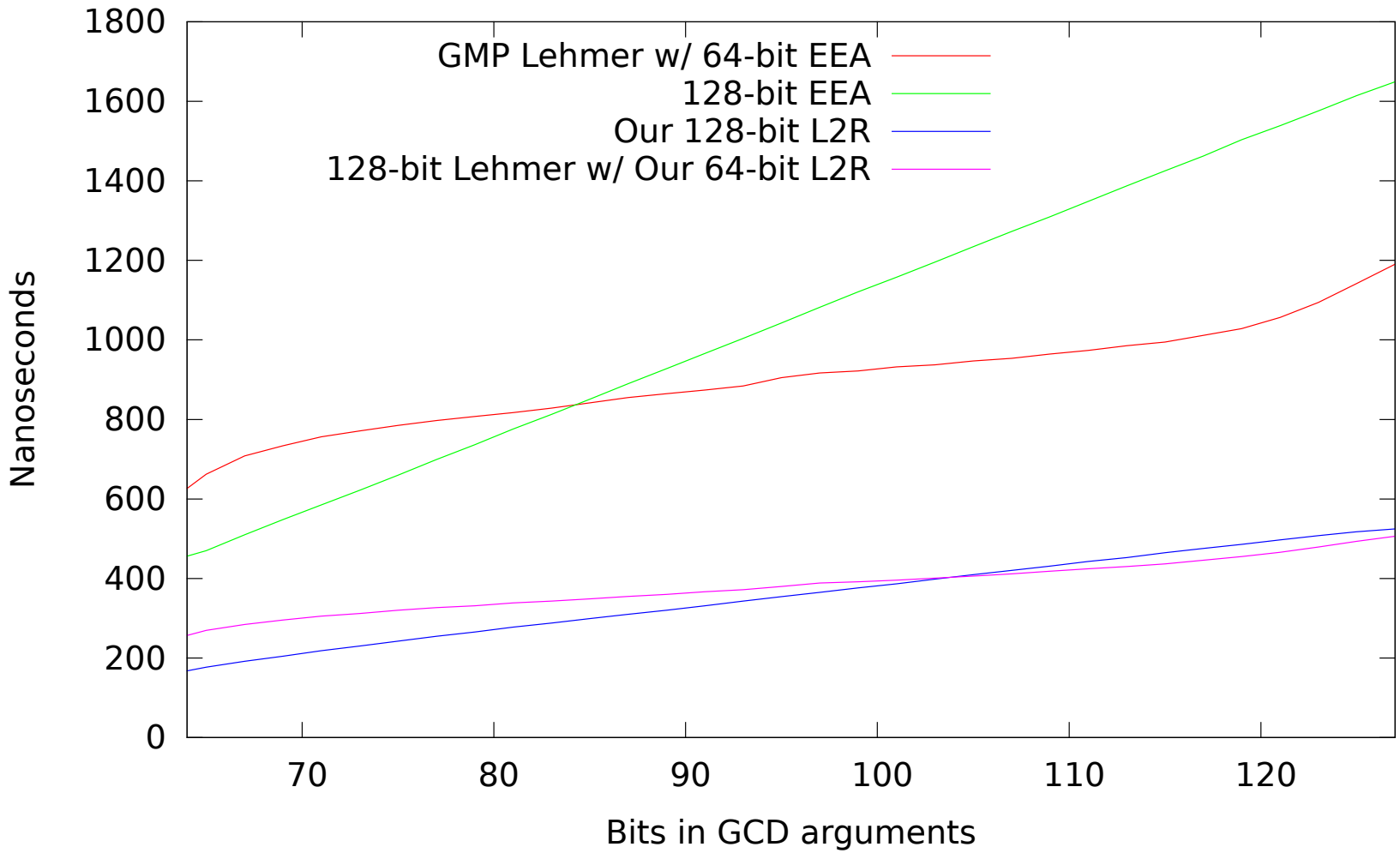
Partial XGCD Results

64-bits



Partial XGCD Results

128-bits



Partial XGCD Improvement

How much faster than the reference implementation?

Bit Range	Algorithm	GMP Lehmer w/ 64-bit EEA
1 – 15	Brent	11.12
16 – 31	Our L2R Binary	6.14
32 – 63	Our L2R Binary	4.37
64 – 103	Our L2R Binary	3.18
104 – 127	Lehmer w/ Our 64-bit L2R	2.31

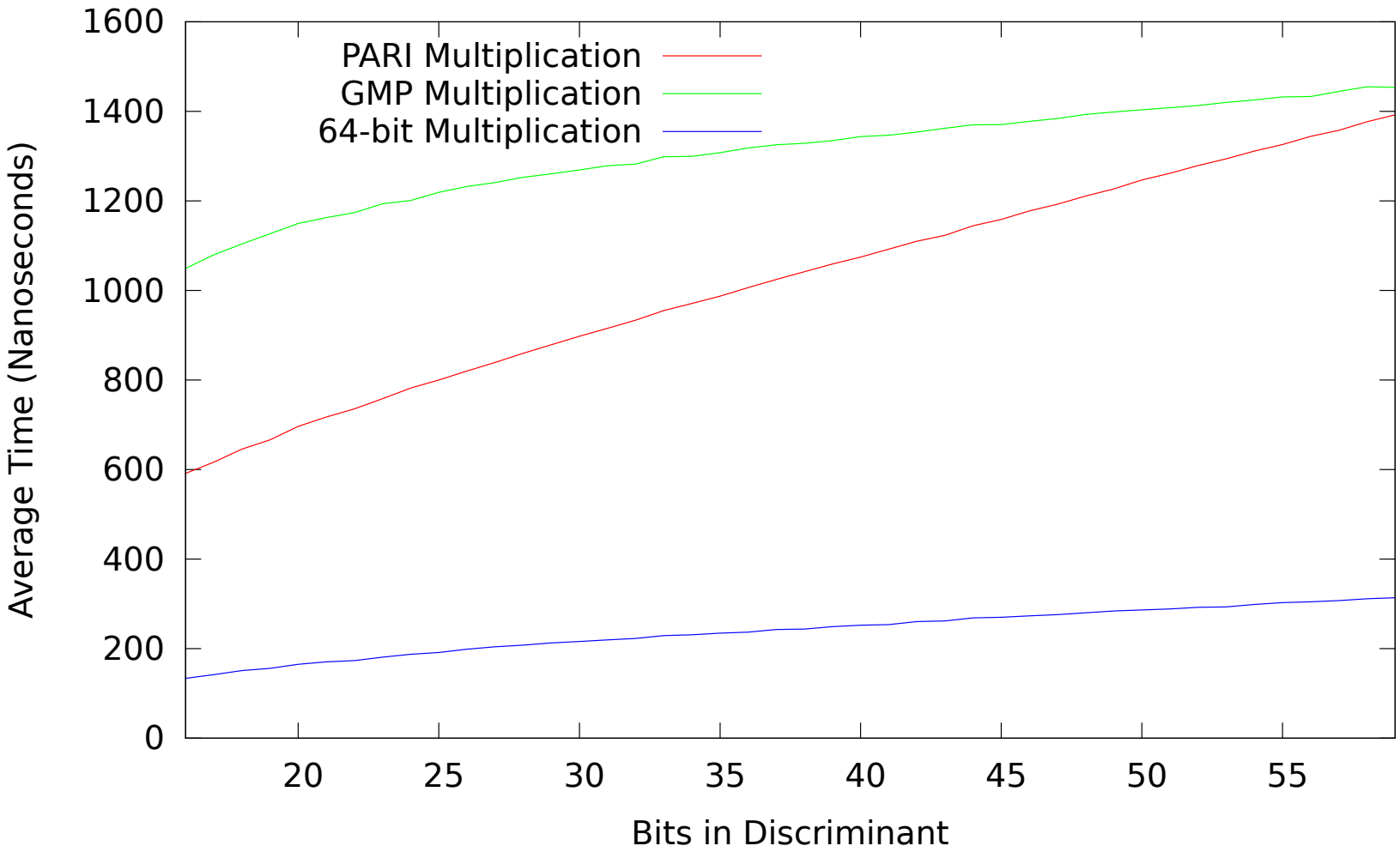
All times are based on the average. The measurement of improvement is an average over the bit range.

Ideal Class Group Experiments

- 64-bit and 128-bit (GMP and Pari for comparison).
- Negative discriminants from 16-bits to 118-bits of size.
- 10,000 groups.
- 1000 multiplications, squares, and cubes.
- About one CPU day.

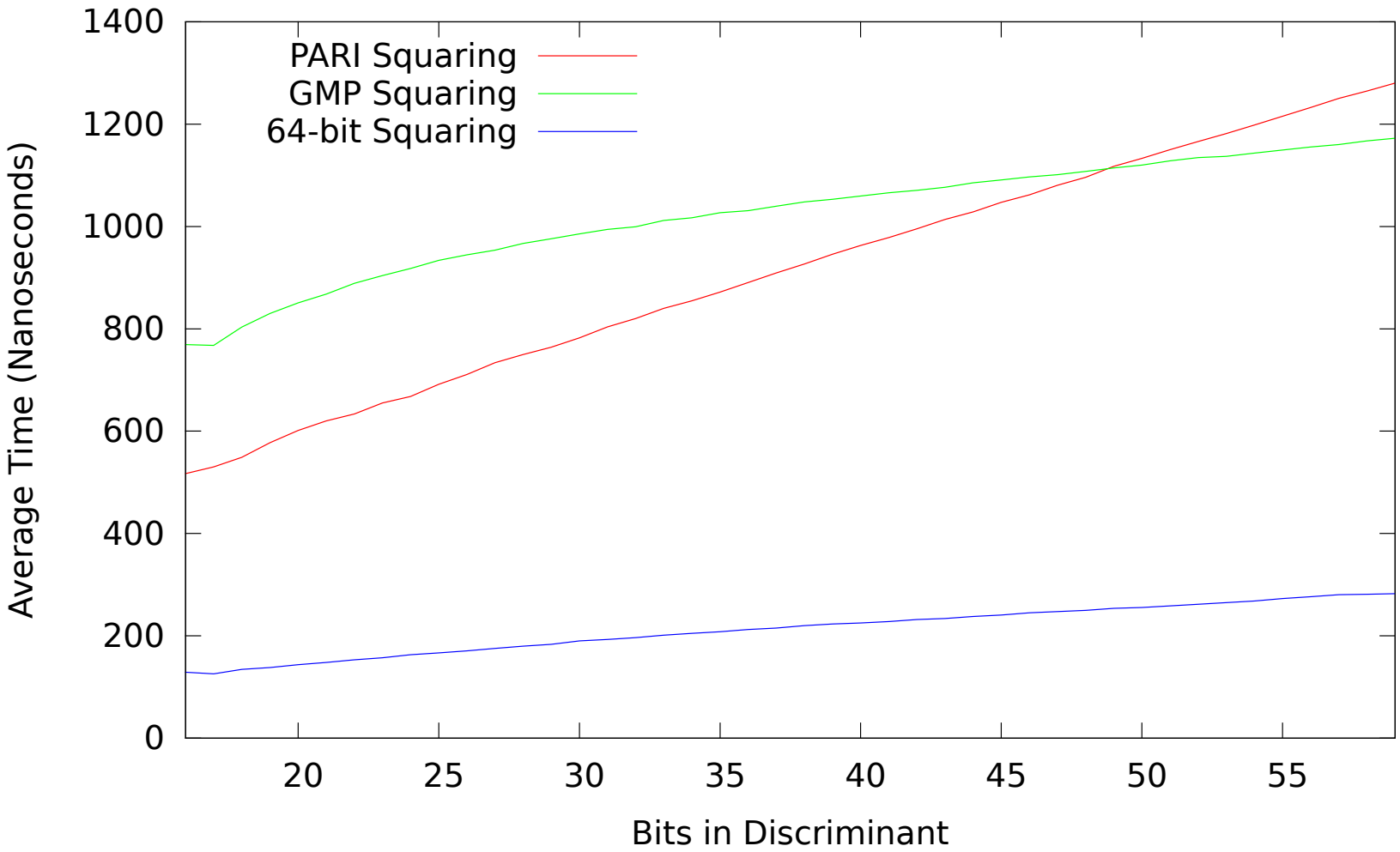
Ideal Class Group Results

64-bit multiplication



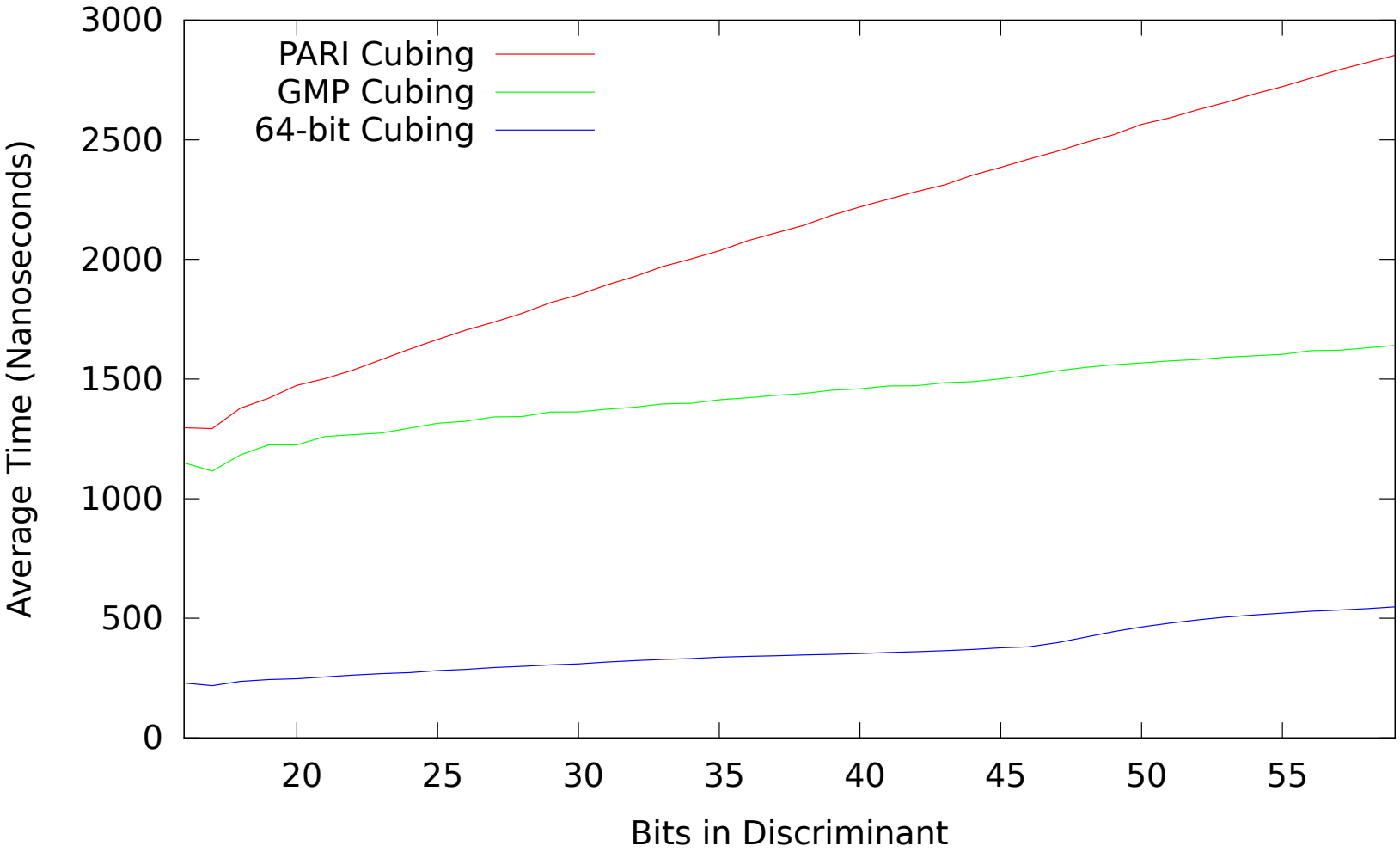
Ideal Class Group Results

64-bit squaring



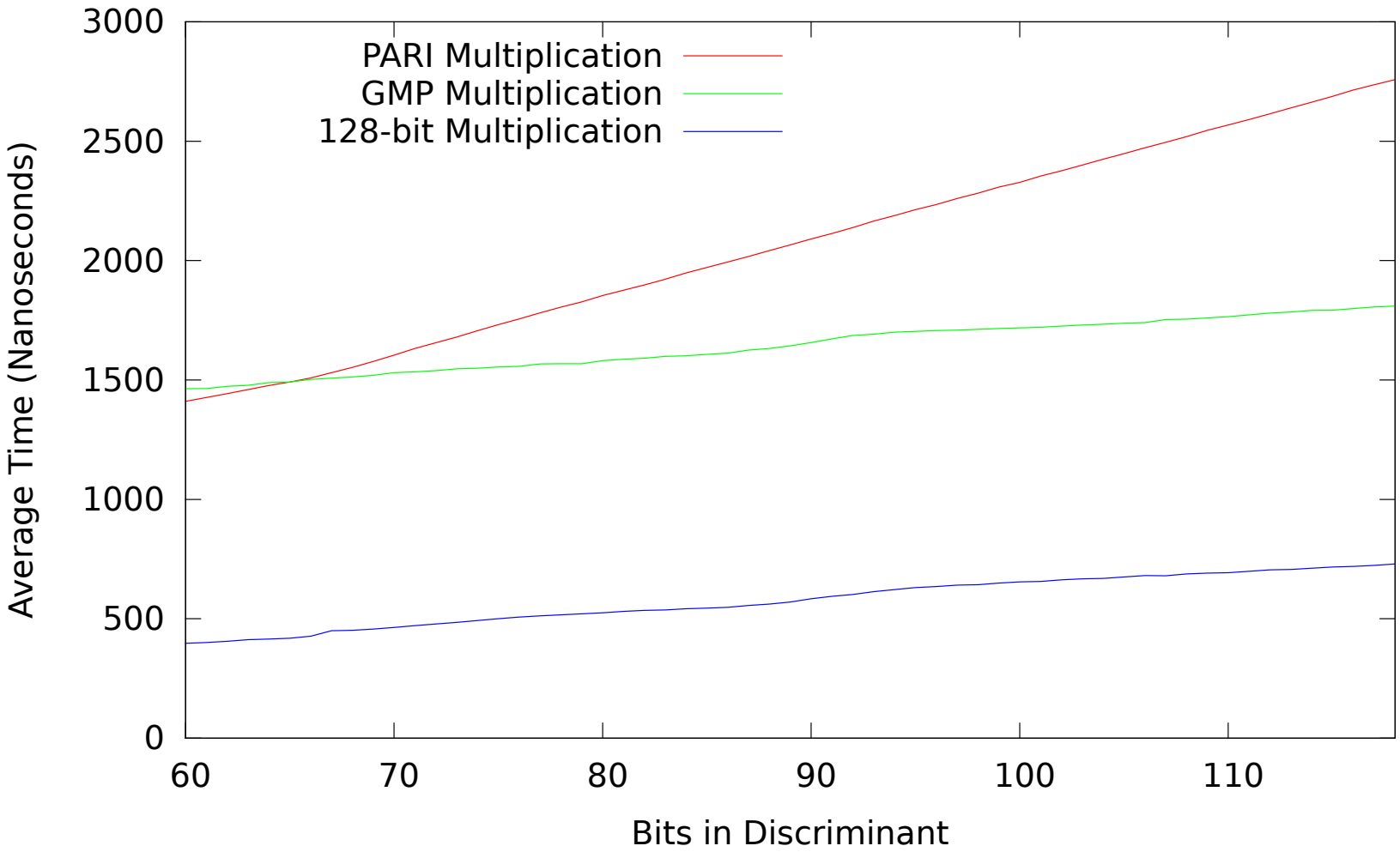
Ideal Class Group Results

64-bit cubing



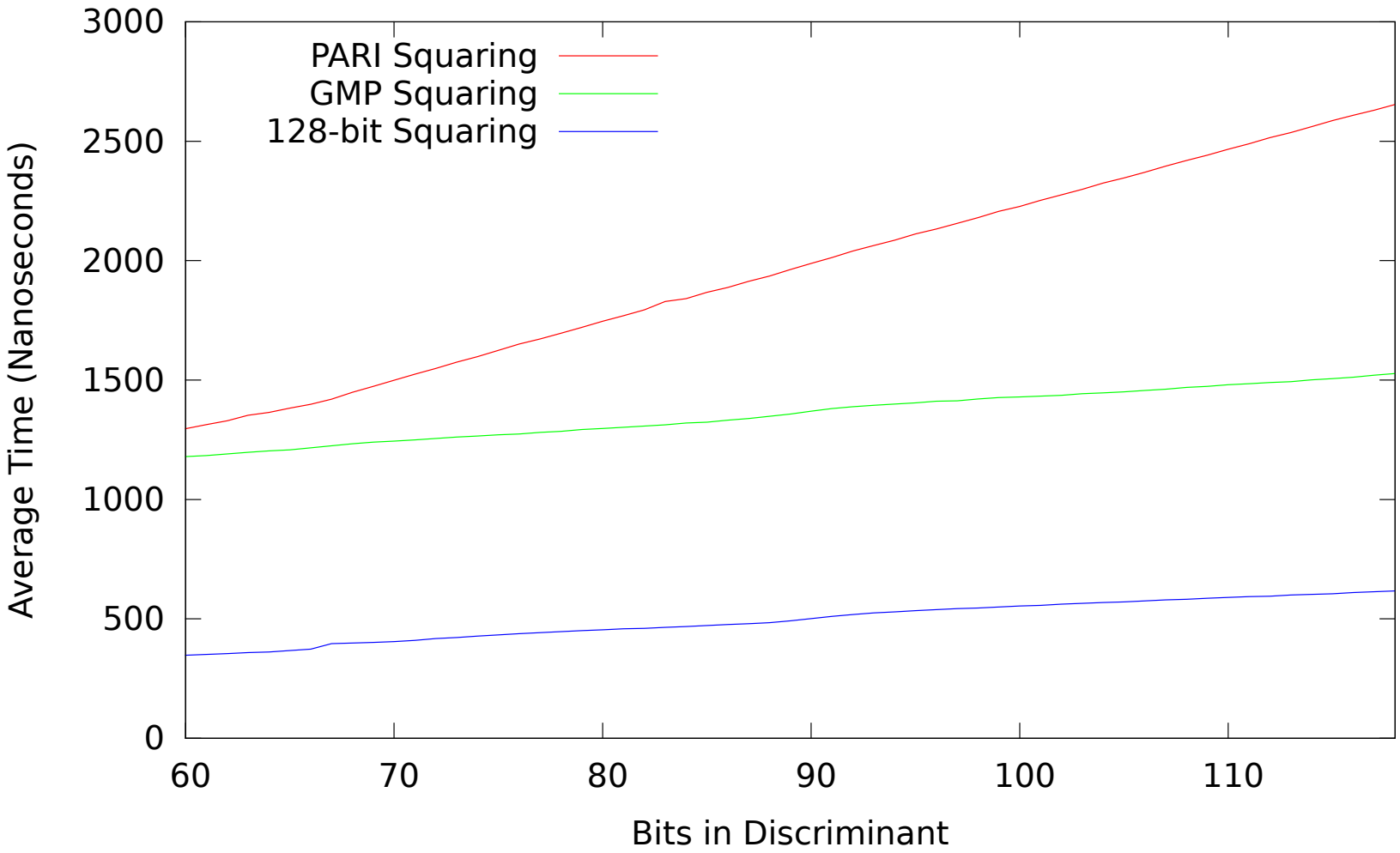
Ideal Class Group Results

128-bit multiplication



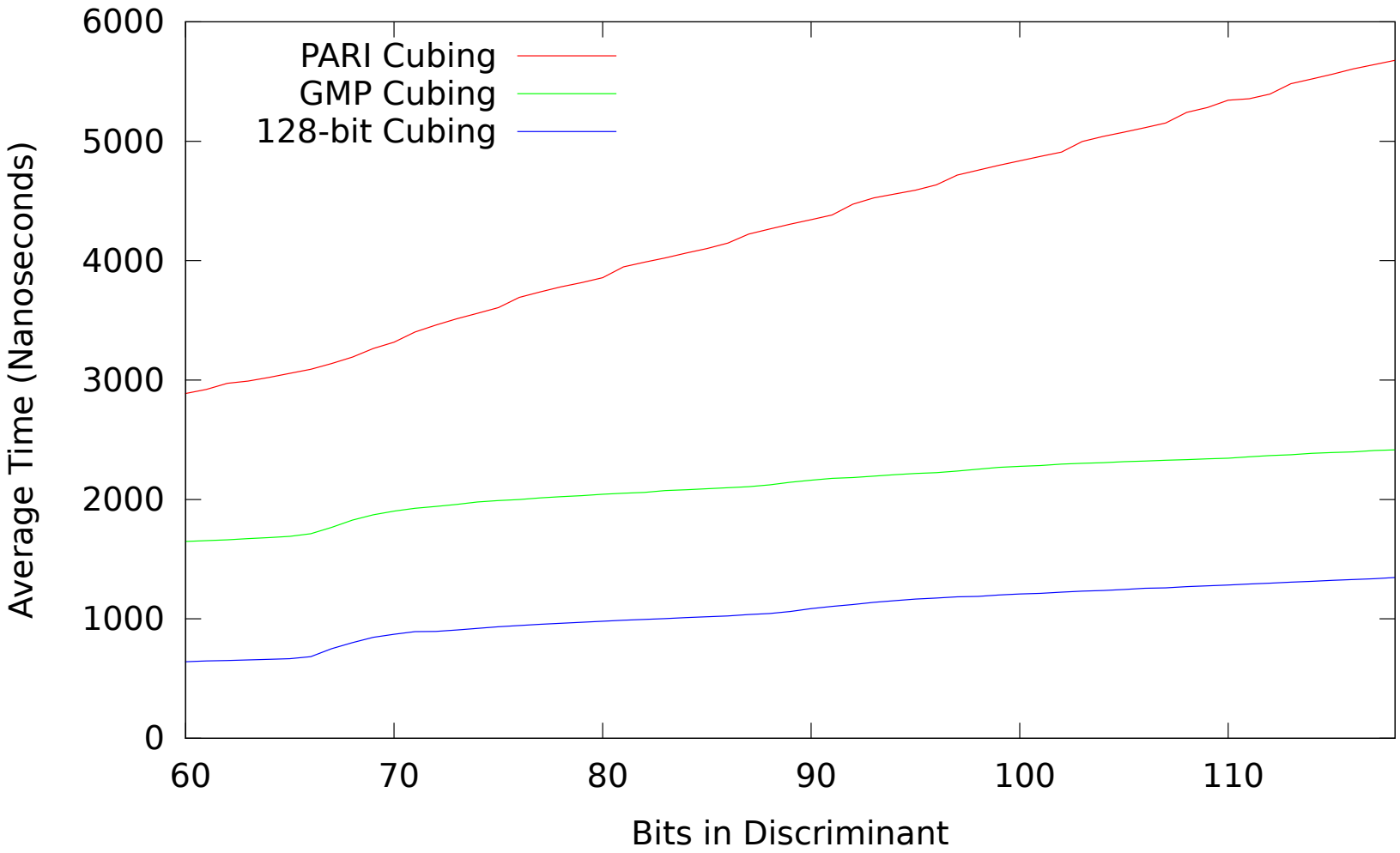
Ideal Class Group Results

128-bit squaring



Ideal Class Group Results

128-bit cubing



Ideal Class Group Improvement

How much faster than the reference implementations?

Operation	Bit Range	Pari	GMP
Multiplication	1 – 59	4.27	5.66
Squaring	1 – 59	4.27	4.95
Cubing	1 – 59	5.86	4.10
Multiplication	59 – 118	3.60	2.74
Squaring	59 – 118	4.02	2.66
Cubing	59 – 118	3.82	1.83

All times are based on the average. The measurement of improvement is an average over the bit range.

SPAR/SuperSPAR

SPAR operates in two stages:

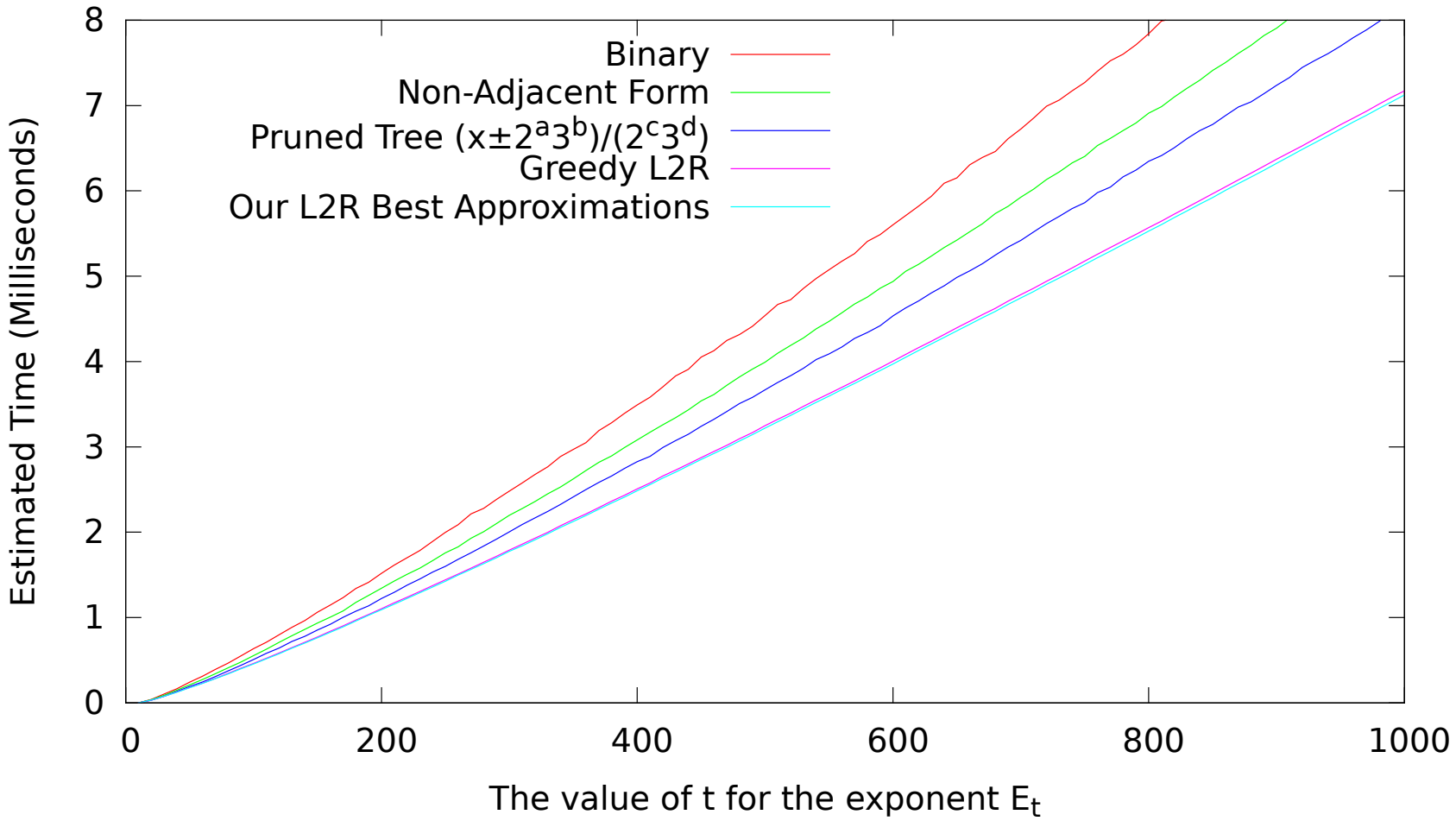
- Exponentiation stage.
- Search stage.

SuperSPAR improves upon each stage.

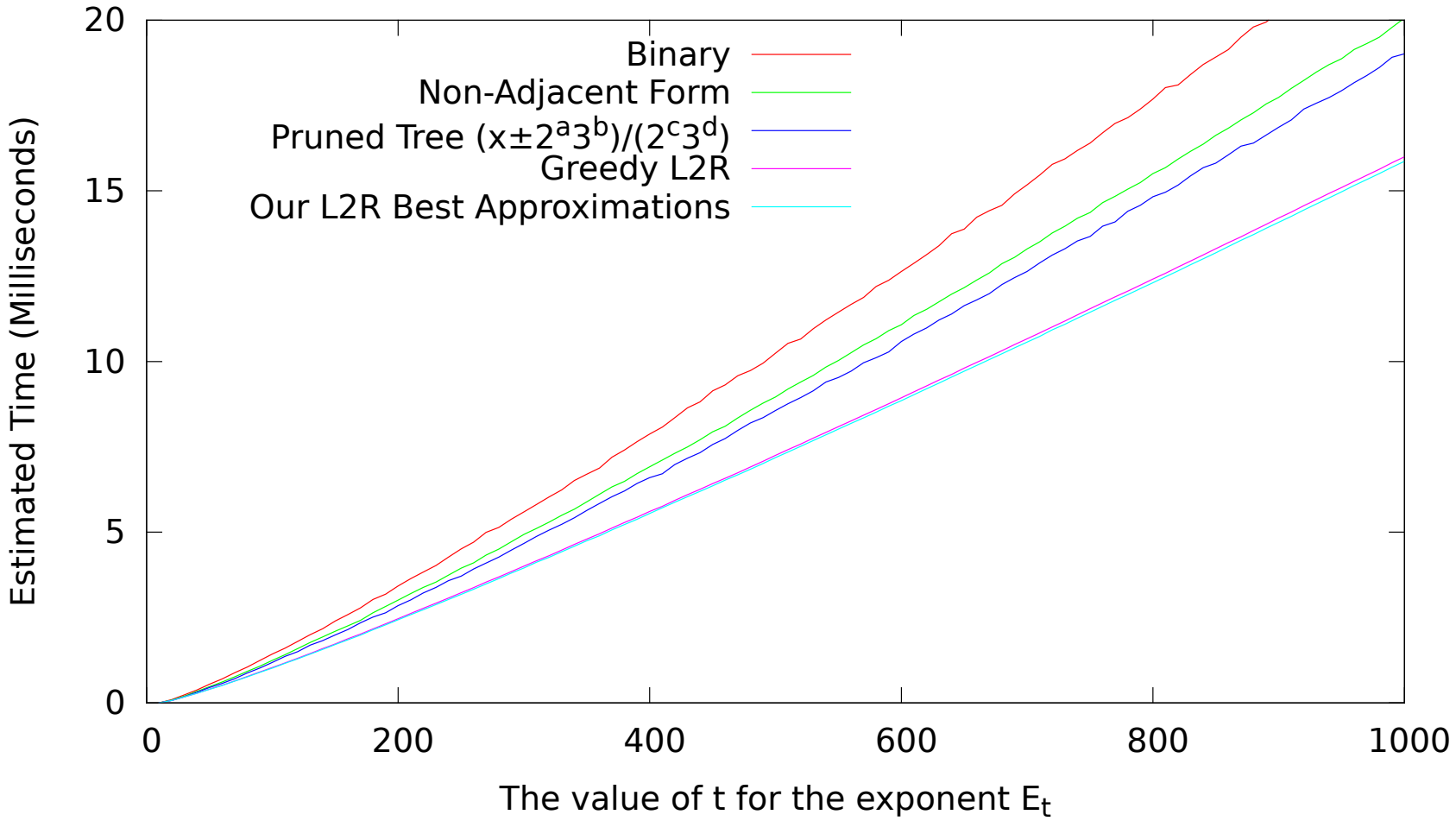
Power Primorial Exponentiation Experiments

- Use average time of class group operations.
- Compute 2,3 representations in advance for each technique.
- Estimate the time to exponentiate given a representation and the average cost of group operations.

Exponentiation Results (59-bit Discriminants)



Exponentiation Results (118-bit Discriminants)



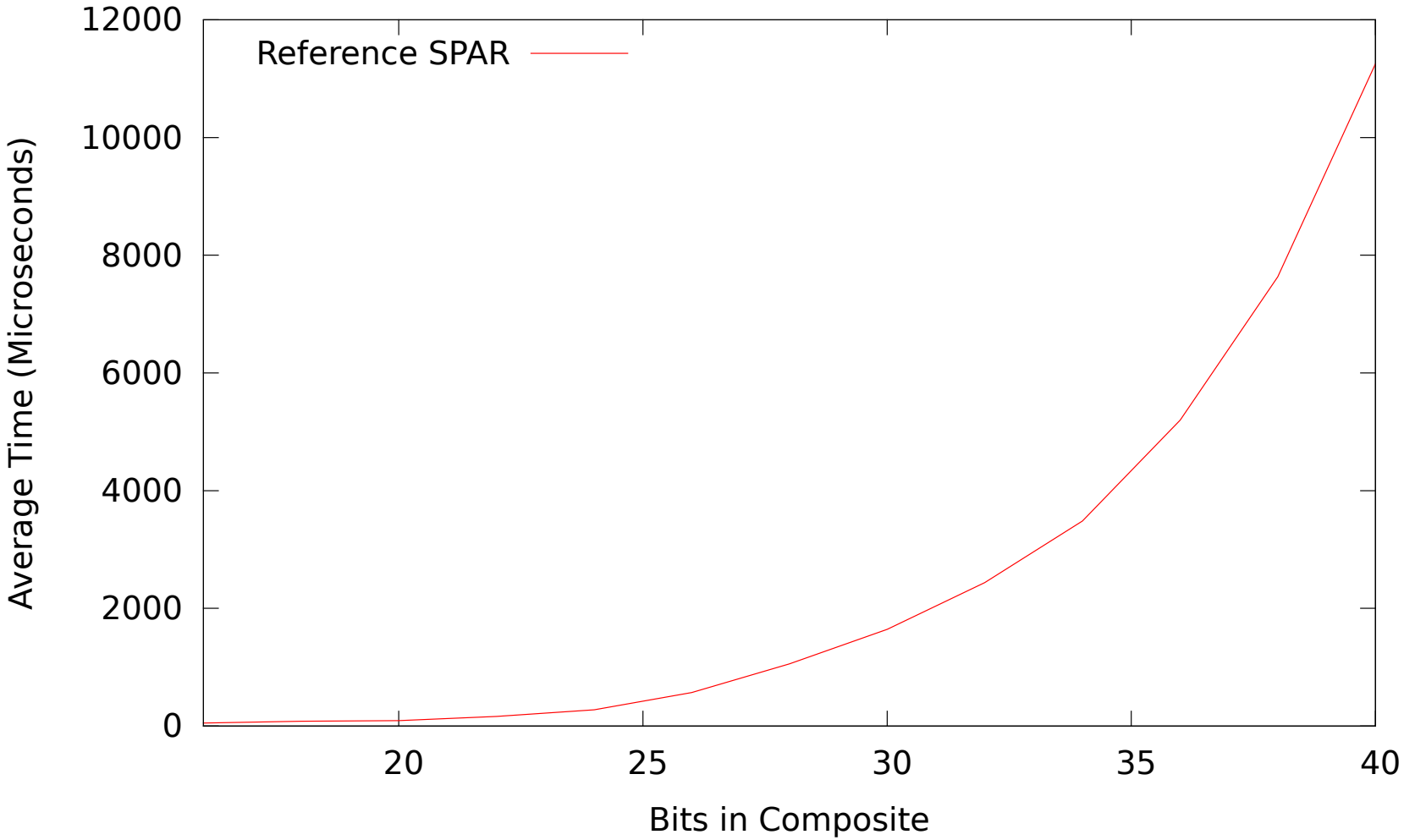
Exponentiation Improvement

How much faster than the reference implementations?

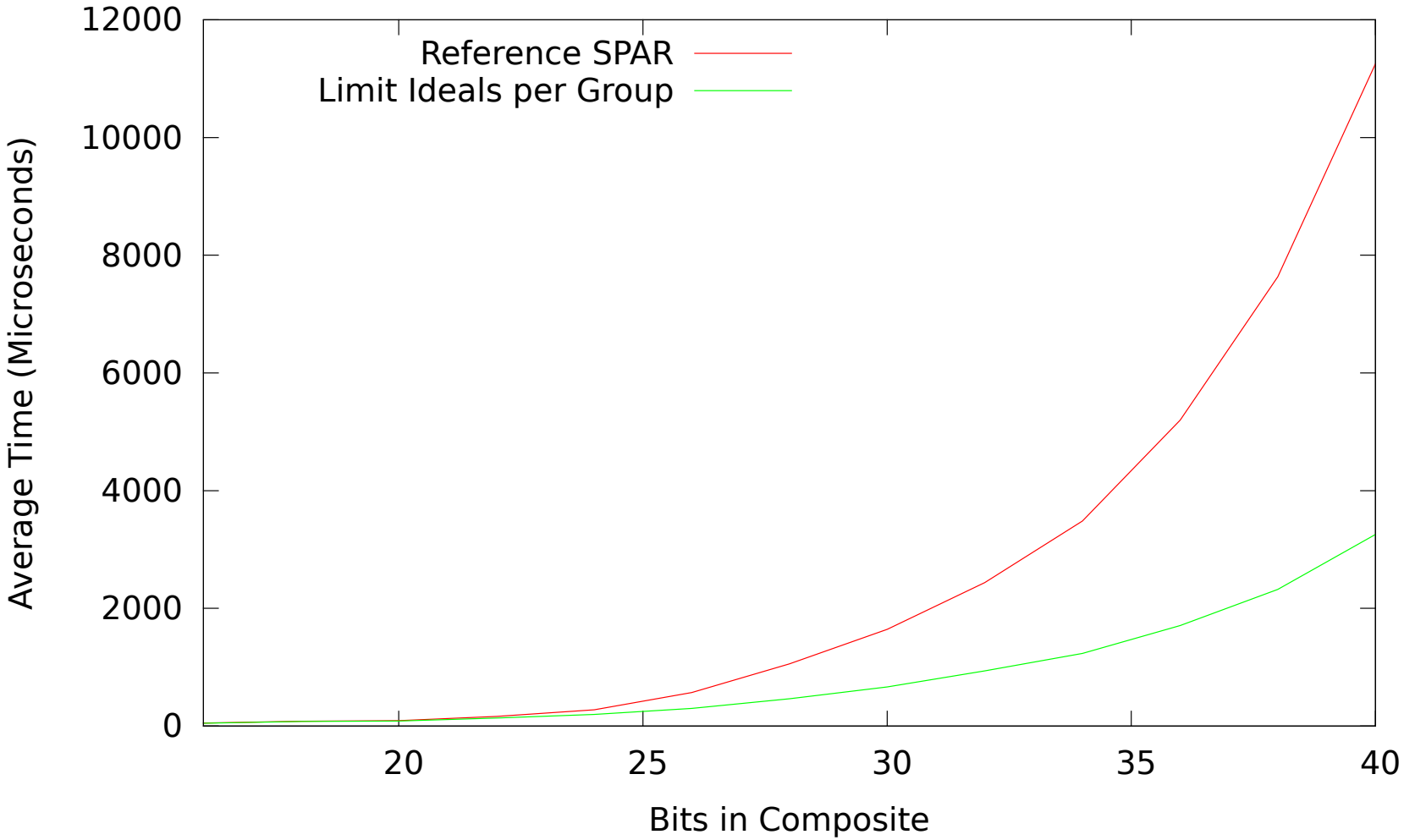
	Binary	NAF	Pruned Tree	Greedy Left-to-Right
Left-to-Right Best Approximations (64-bit)	1.40	1.25	1.13	1.009
Left-to-Right Best Approximations (128-bit)	1.42	1.25	1.18	1.010

All times are based on the average. The measurement of improvement is an average over the bit range.

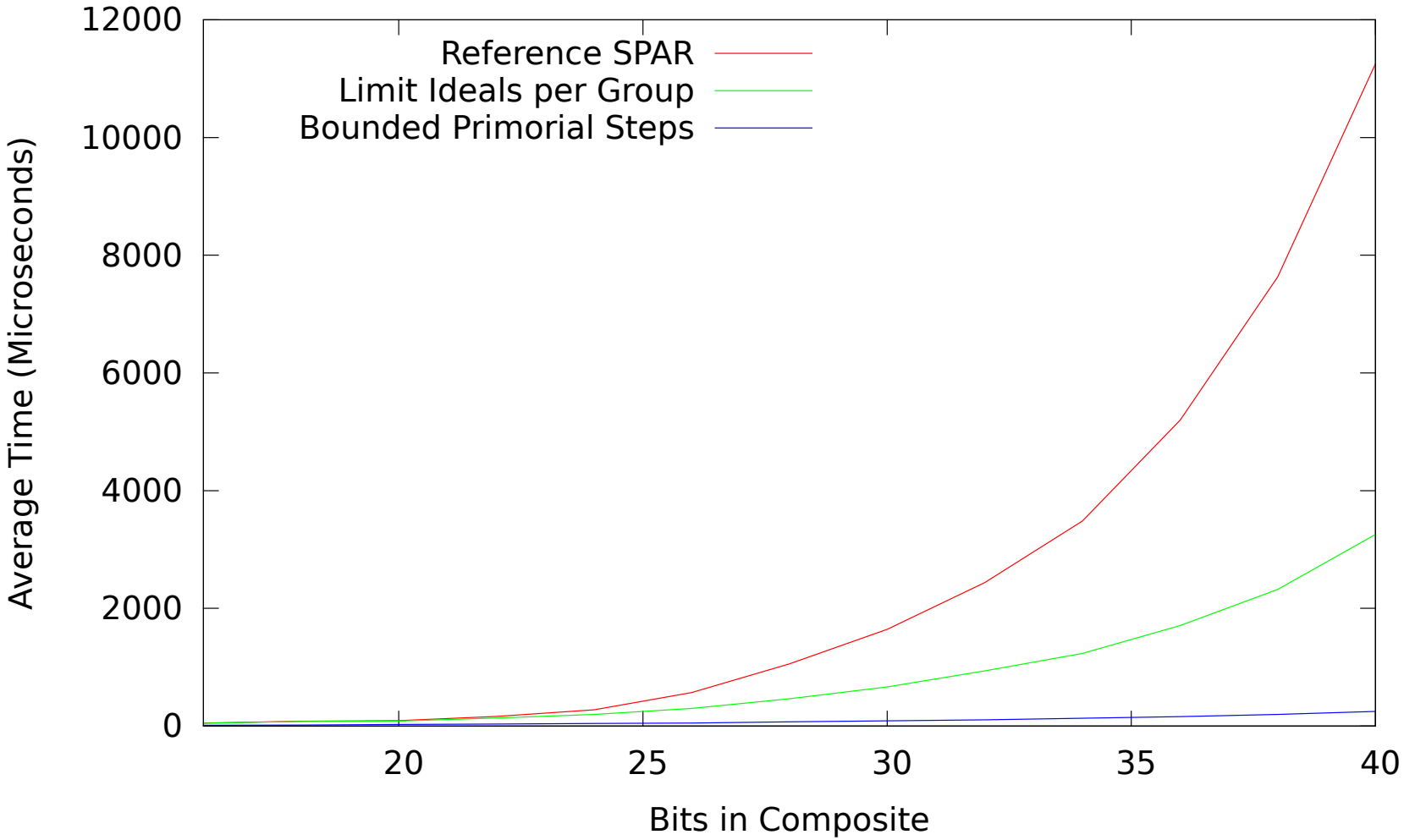
Reference Implementation of SPAR



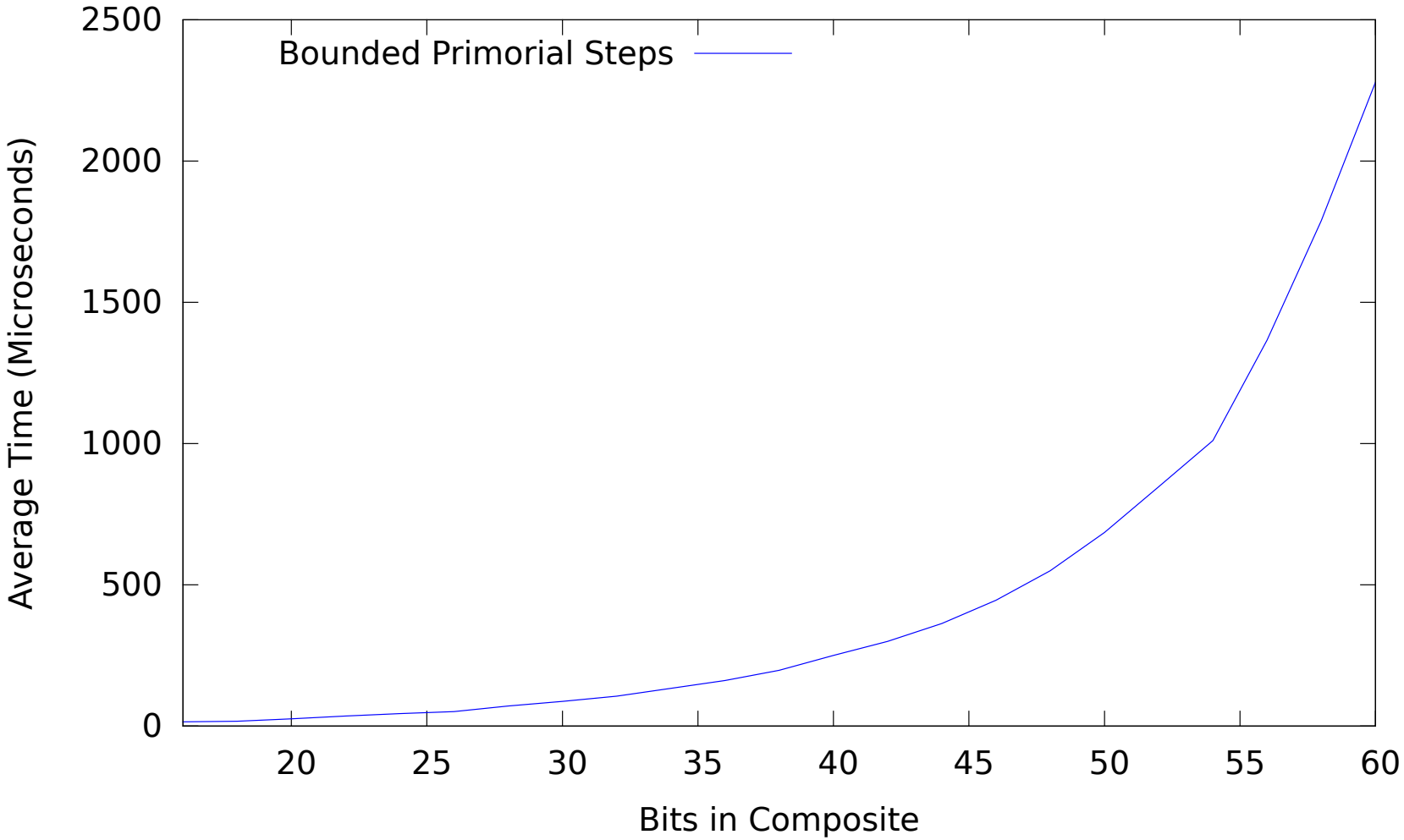
Bound the Number of Ideals per Group



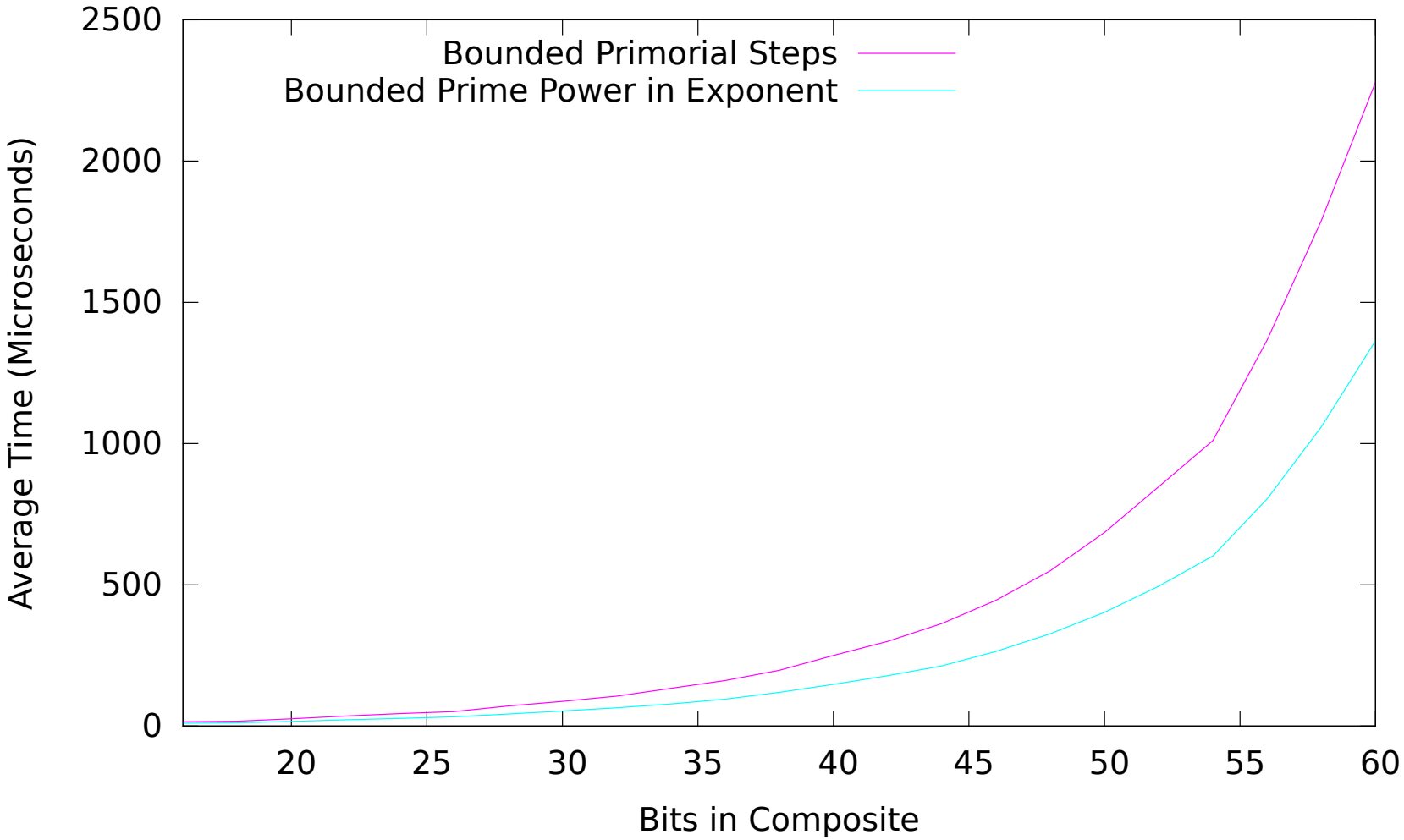
Bounded Primorial Steps Search



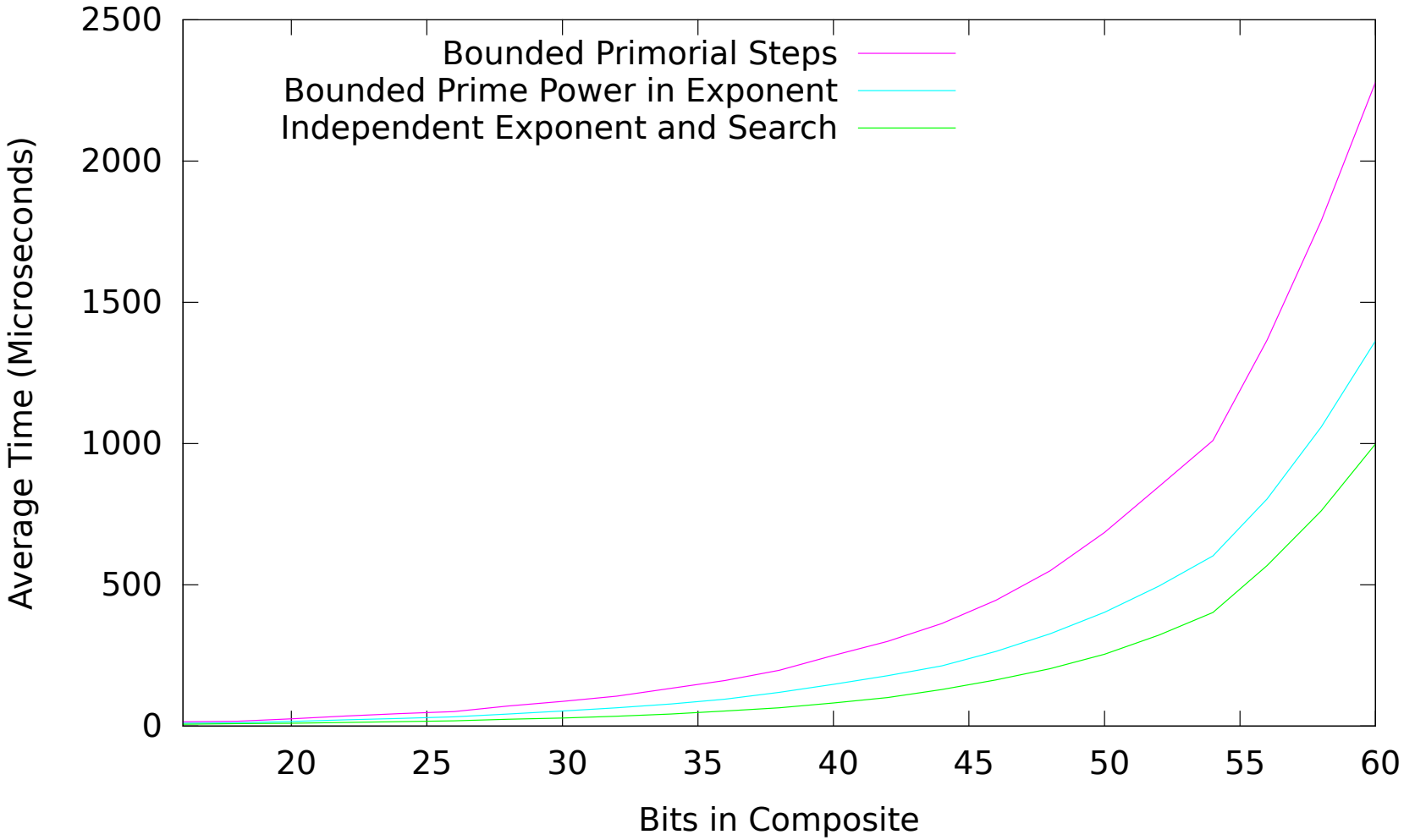
Bounded Primorial Steps Search



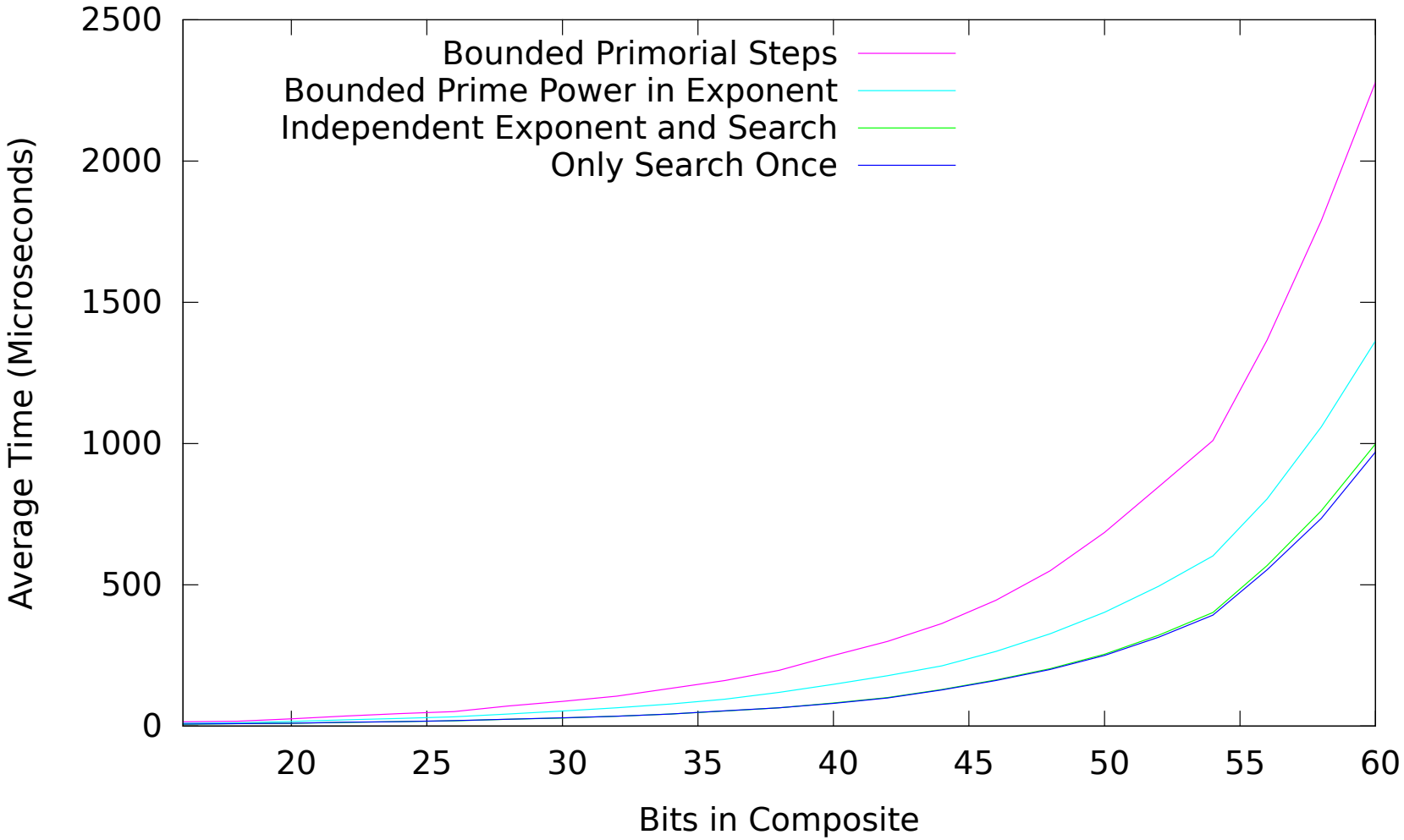
Bound e_i in Exponent $E = \prod p_i^{e_i}$



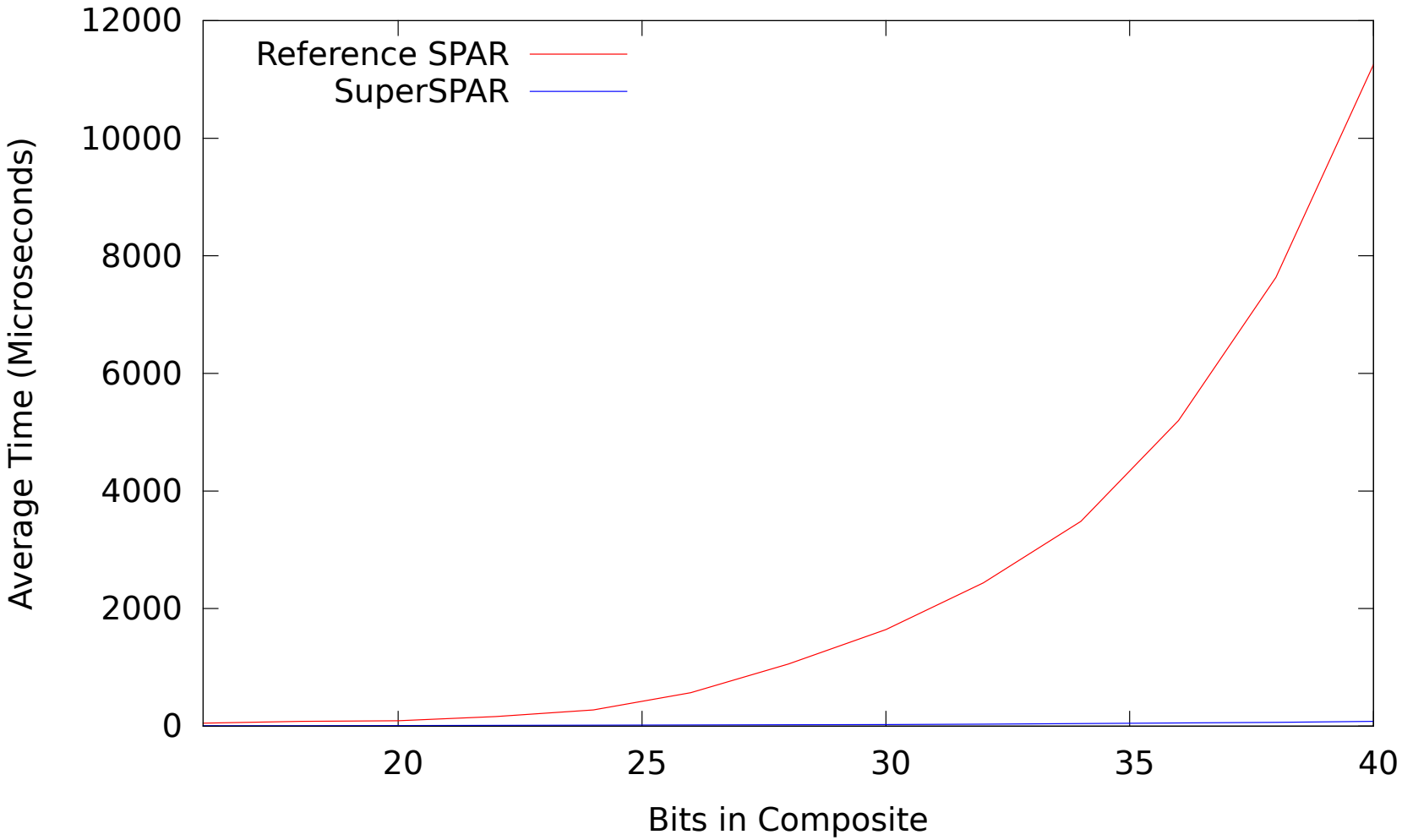
Independent Exponent E and Search Bound mP_w



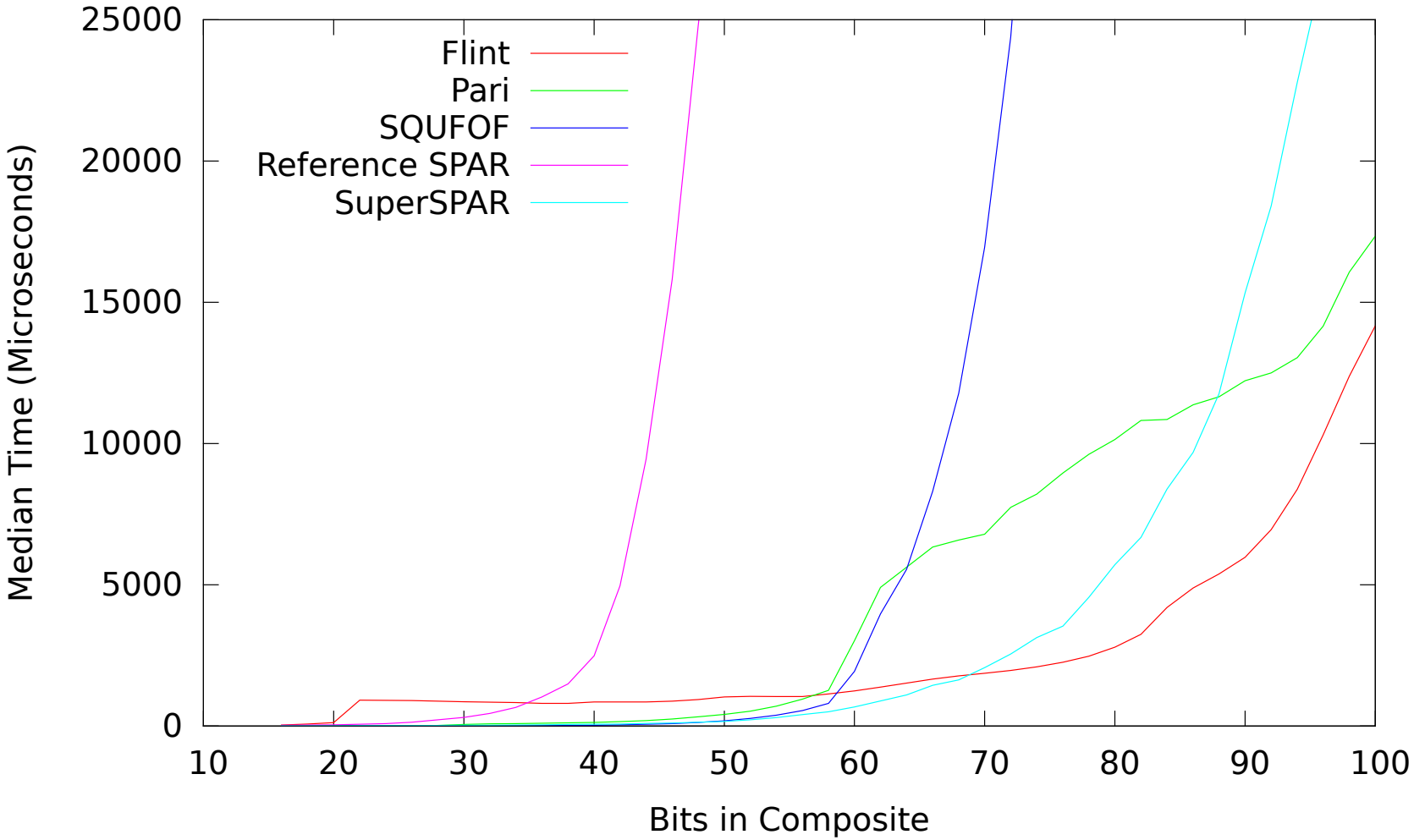
Perform a Single Search



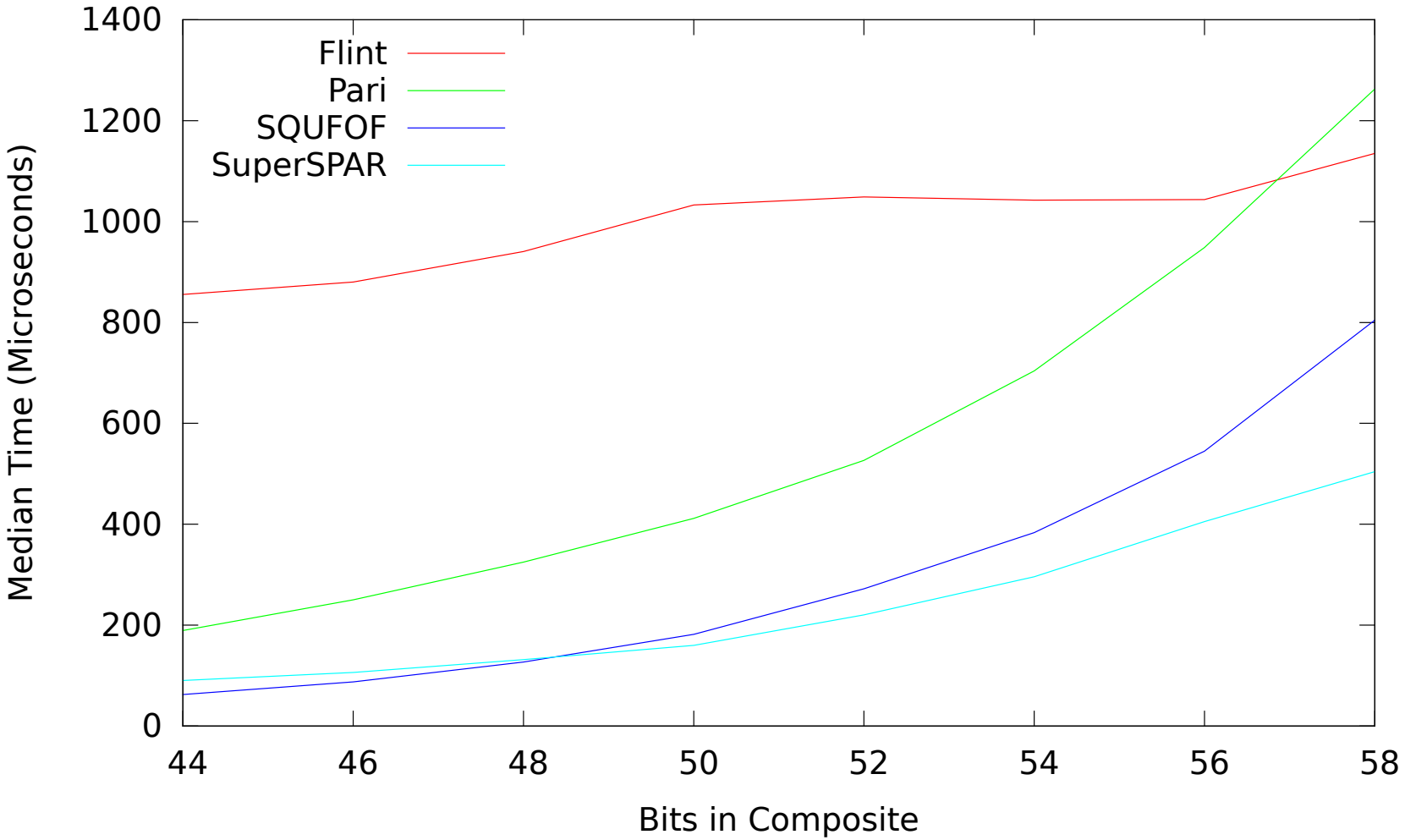
SPAR vs SuperSPAR



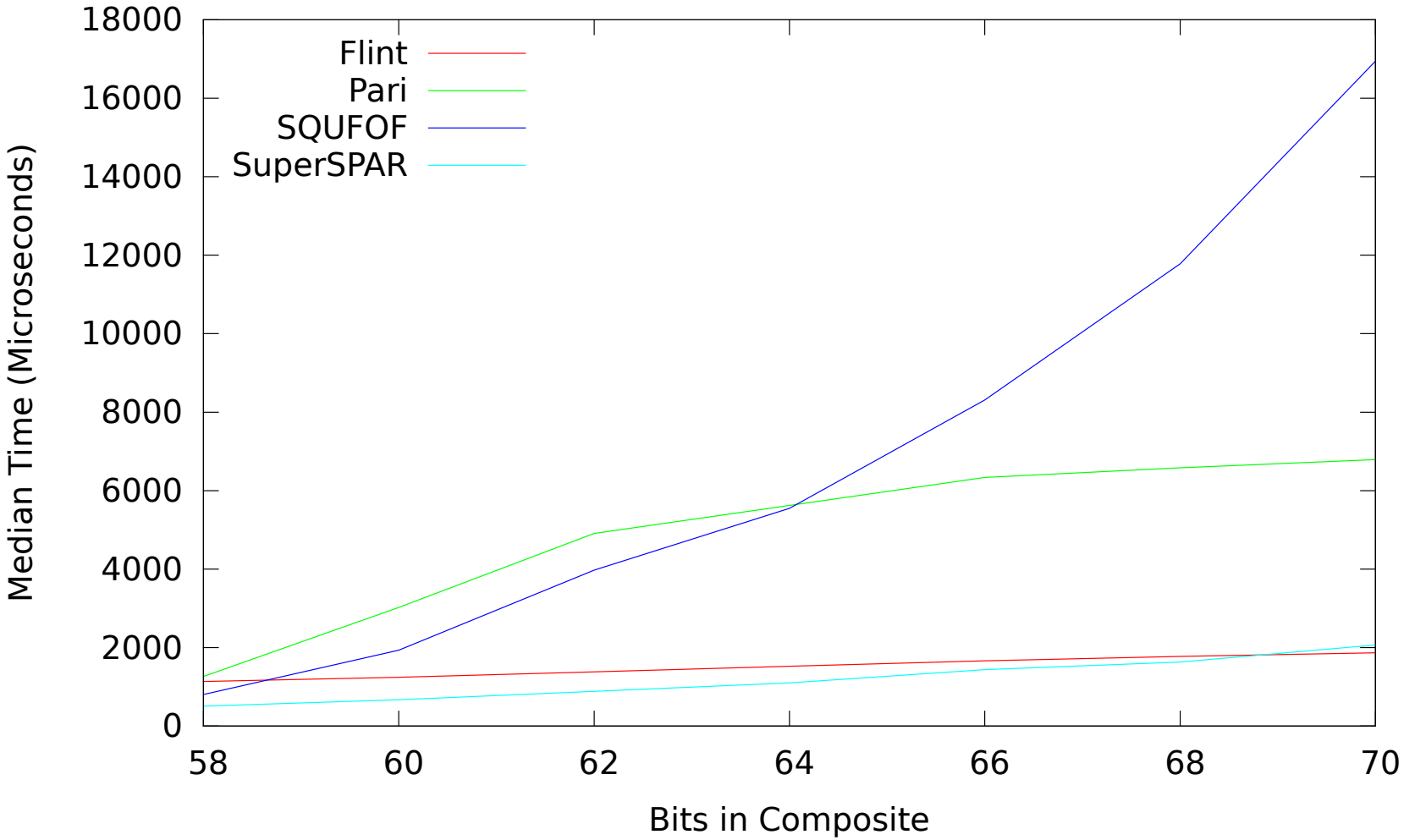
Median Integer Factoring Times



Median Integer Factoring Times (Zoomed Left)



Median Integer Factoring Times (Zoomed Right)



Future Work

Some work that is eagerly awaiting:

- Special case for quotients $q \leq 4$ in XGCD.
- Real quadratic fields and hyperelliptic curves.
- Improvements to left-to-right best approximations algorithm.

Thank You

Summary of performance improvements:

- Arithmetic in the ideal class group for discriminants smaller than 118-bits.
- The extended GCD for inputs smaller than 128-bits.
- Integer factoring for integers 49-bits to 68-bits of size (median case).
- 2,3 representations for exponentiating by power primorials.

Some Possible Applications

These are just a few possible applications.

Ideal class group:

- Class group tabulation for larger discriminants (Ramachandran2006).

Integer Factoring:

- Factoring left-overs after sieving in algorithms like the number field sieve using multiple large primes.

Contributions

Contributions to ideal class group arithmetic:

- Optimized implementations using 64 and 128-bit arithmetic.
- Optimized XGCD for 32, 64, and 128-bits.
  - A simplified left-to-right binary XGCD (and partial XGCD).

Contributions to integer factoring:

- SuperSPAR integer factoring algorithm.
- Left-to-right best approximations method for computing 2,3 representations of power primorials.

Extended Greatest Common Divisor (XGCD)

Extended greatest common divisor solves

$$s = Ua + Vb = \gcd(a, b)$$

for the largest s dividing both a and b.

Extended Euclidean Algorithm

1: **procedure** $\mathrm{EEA}(a, b)$ $\qquad \{a, b \in \mathbb{Z}_{\geq 0}\}$

2: $\quad \begin{bmatrix} s & U & V \\ t & X & Y \end{bmatrix} \leftarrow \begin{bmatrix} a & 1 & 0 \\ b & 0 & 1 \end{bmatrix}$

3: $\quad$ **while** $t > 0$ **do**

4: $\quad\quad q \leftarrow \lfloor s/t \rfloor$

5: $\quad\quad \begin{bmatrix} s & U & V \\ t & X & Y \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -q \end{bmatrix} \cdot \begin{bmatrix} a & 1 & 0 \\ b & 0 & 1 \end{bmatrix}$

6: $\quad$ **return** (s, U, V)

Our Left-to-Right Binary XGCD

1: **procedure** OUR_L2R_XGCD(a, b) $\{a, b \in \mathbb{Z}_{\geq 0}\}$

2: $\begin{bmatrix} s & U & V \\ t & X & Y \end{bmatrix} \leftarrow \begin{bmatrix} a & 1 & 0 \\ b & 0 & 1 \end{bmatrix}$

3: **while** $t > 0$ **do**

4: $k \leftarrow$ NUMBITS(s) - NUMBITS(t)

5: subtract 2^k times the second row from the first

6: **if** $s < 0$ **then** negate the first row

7: **if** $s < t$ **then** swap the rows of the matrix

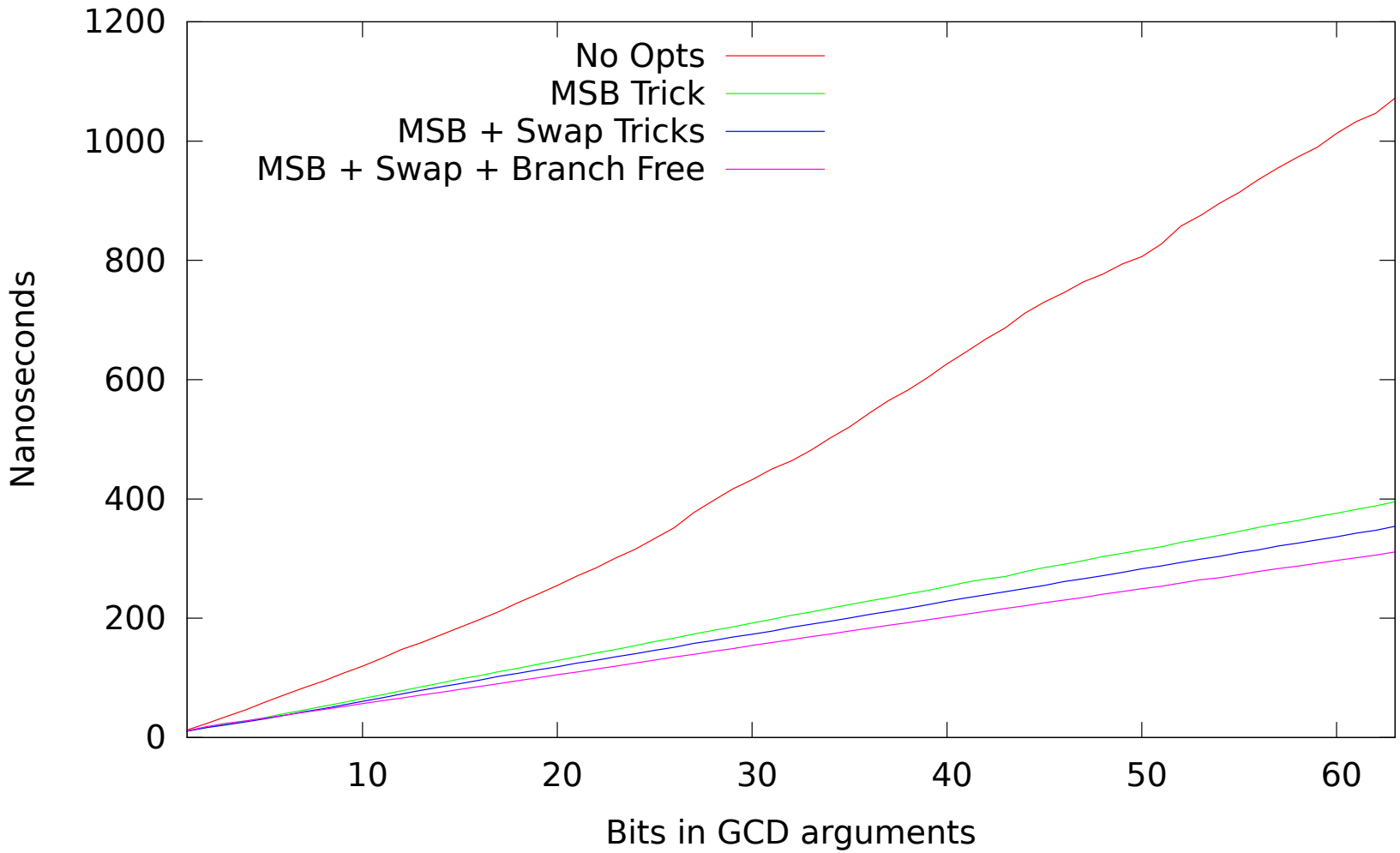
8: **return** (s, U, V)

Our Left-to-Right Binary PXGCD

1: **procedure** OUR_L2R_PXGCD(a, b, T) $\quad \{a, b, T \in \mathbb{Z}_{\geq 0}\}$
2: $\quad \begin{bmatrix} r_1 & c_1 \\ r_0 & c_0 \end{bmatrix} \leftarrow \begin{bmatrix} a & 0 \\ b & -1 \end{bmatrix}$
3: $\quad z \leftarrow 0, s_1 \leftarrow 1, s_0 \leftarrow 1$
4: $\quad$ **while** $r_0 \neq 0$ and $r_0 > T$ **do**
5: $\quad\quad k \leftarrow$ NUMBITS(r_1) - NUMBITS(r_0)
6: $\quad\quad$ subtract 2^k times the second row from the first
7: $\quad\quad$ **if** $r_1 < 0$ **then** $r_1 \leftarrow -r_1, c_1 \leftarrow -c_1, s_1 \leftarrow -s_1$
8: $\quad\quad$ **if** $r_1 < r_0$ **then**
9: $\quad\quad\quad$ swap the rows of the matrix and s_1 with s_0
10: $\quad\quad\quad z \leftarrow z + 1$
11: $\quad$ **return** $(s_1 \cdot r_1, s_1 \cdot c_1, s_0 \cdot r_0, s_0 \cdot c_0, z)$

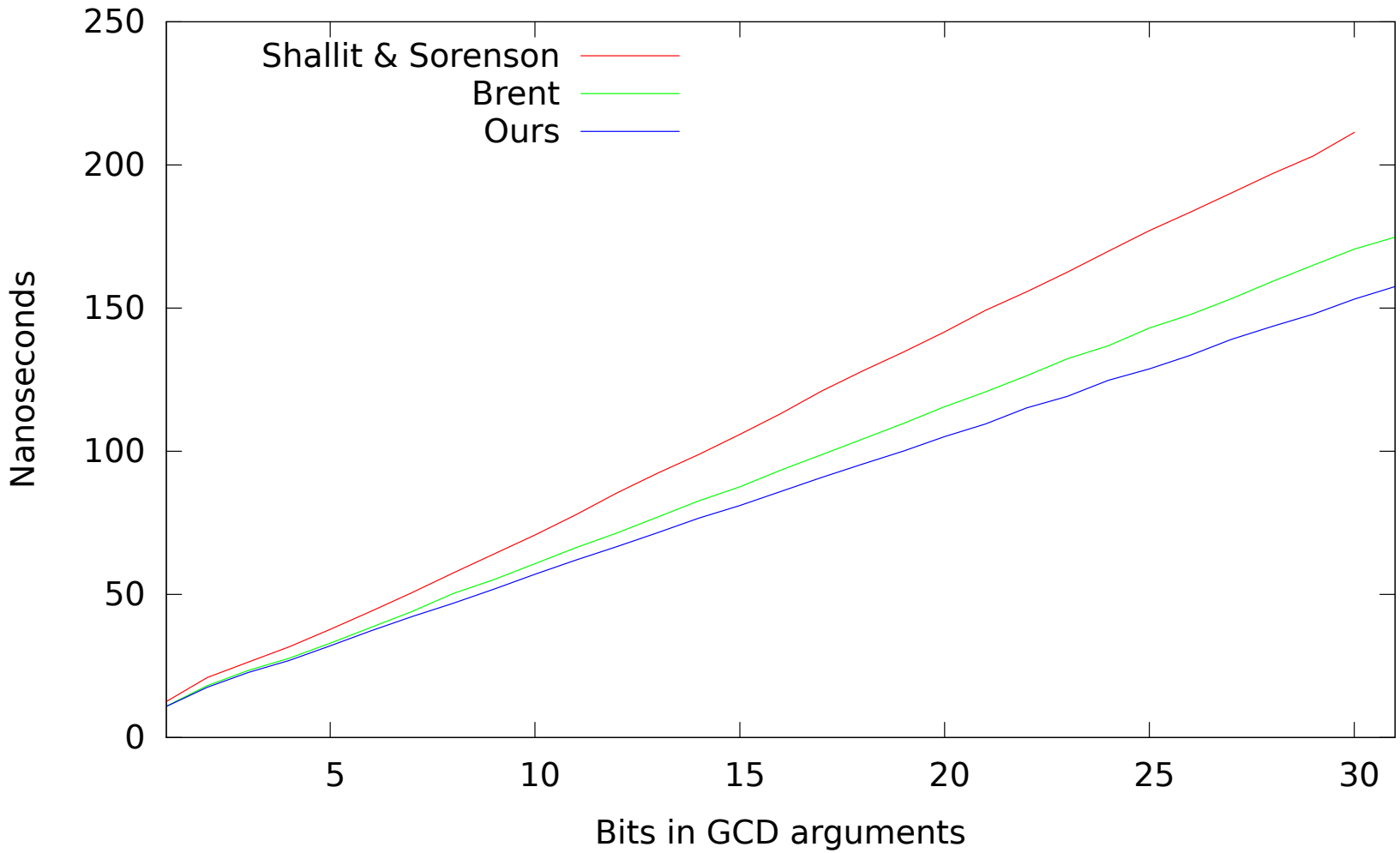
Binary XGCD Optimizations

64-bit simplified left-to-right binary XGCD



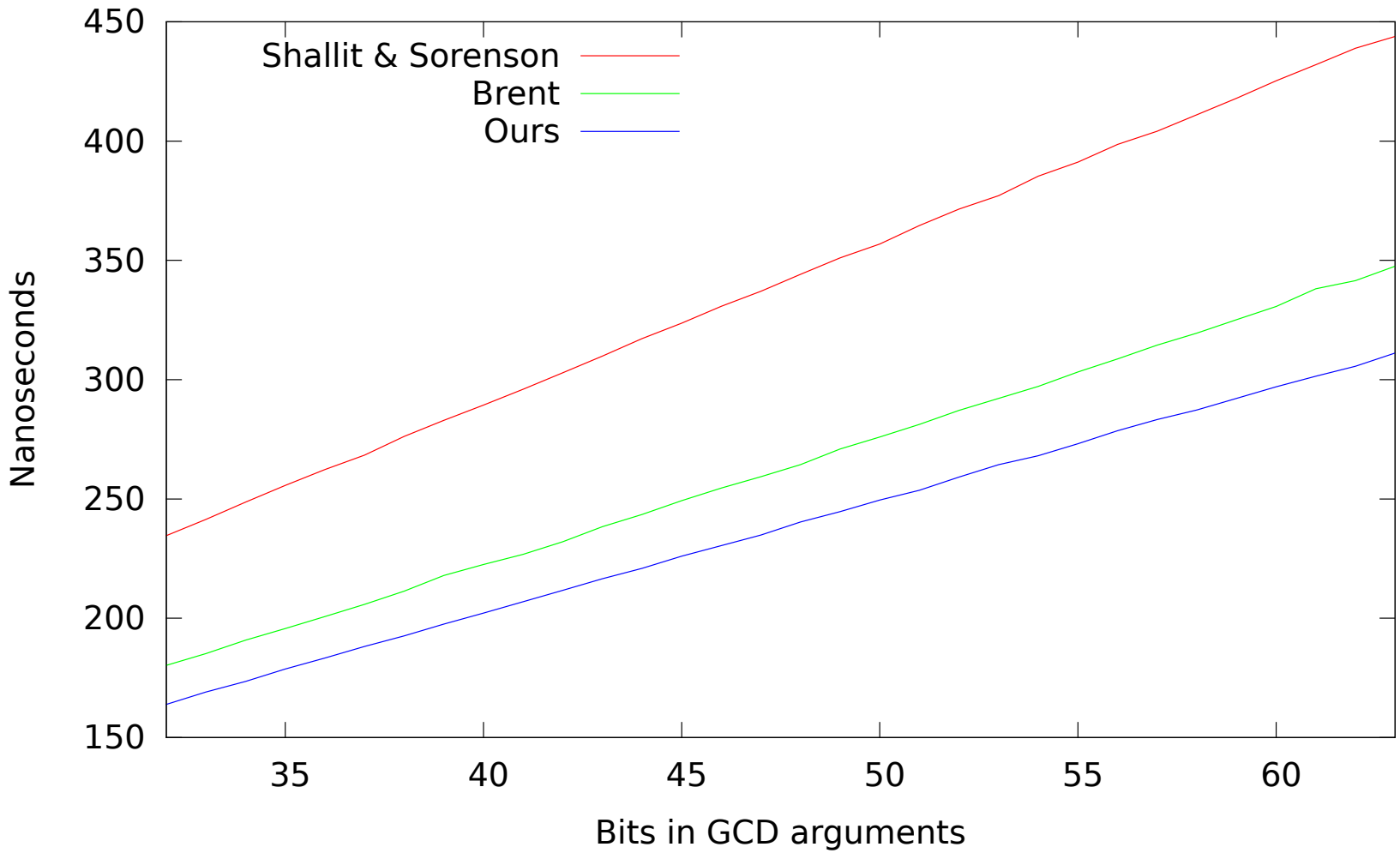
Left-to-Right Binary XGCDs

32-bit



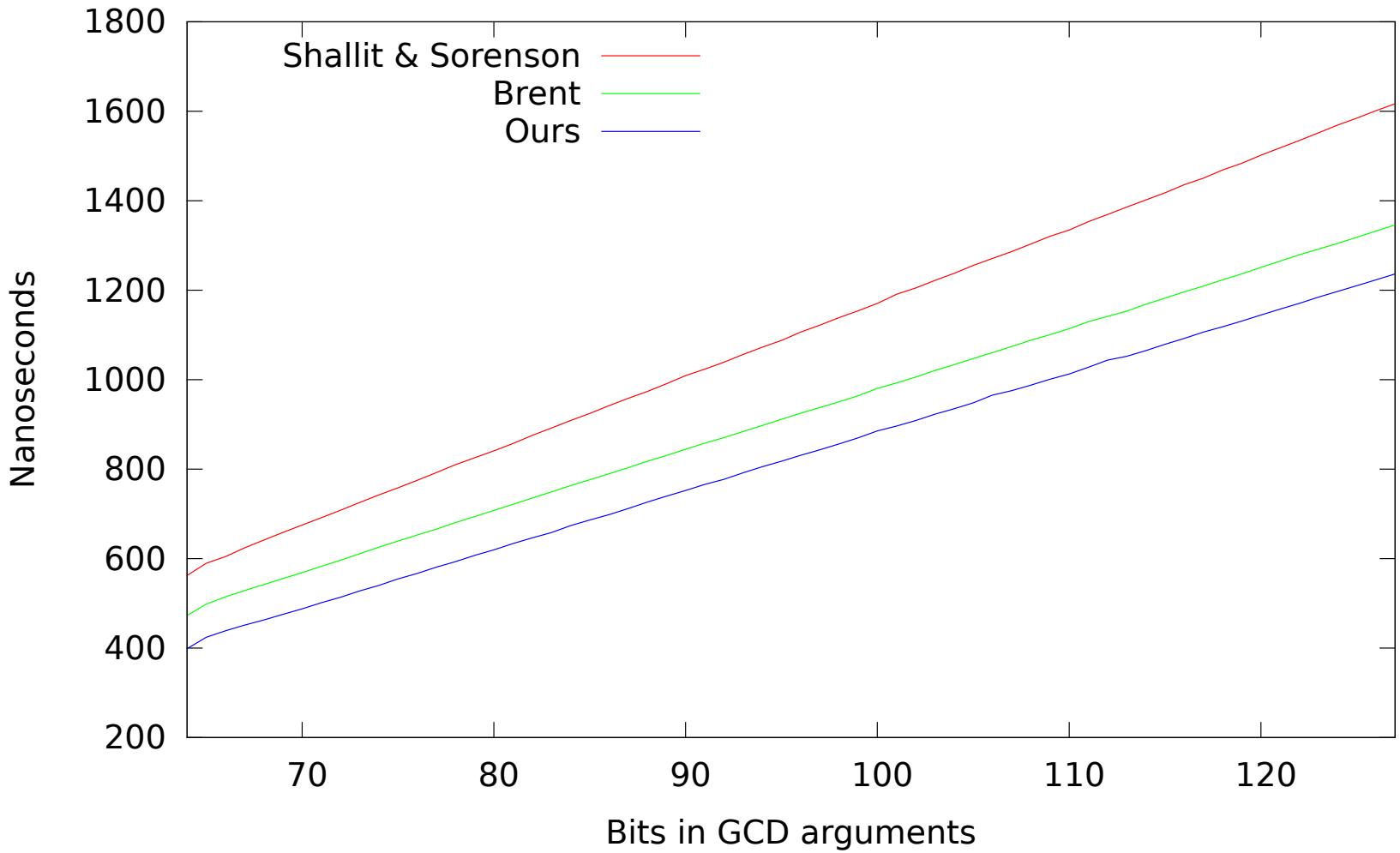
Left-to-Right Binary XGCDs

64-bit



Left-to-Right Binary XGCDs

128-bit



Approximate Square Root

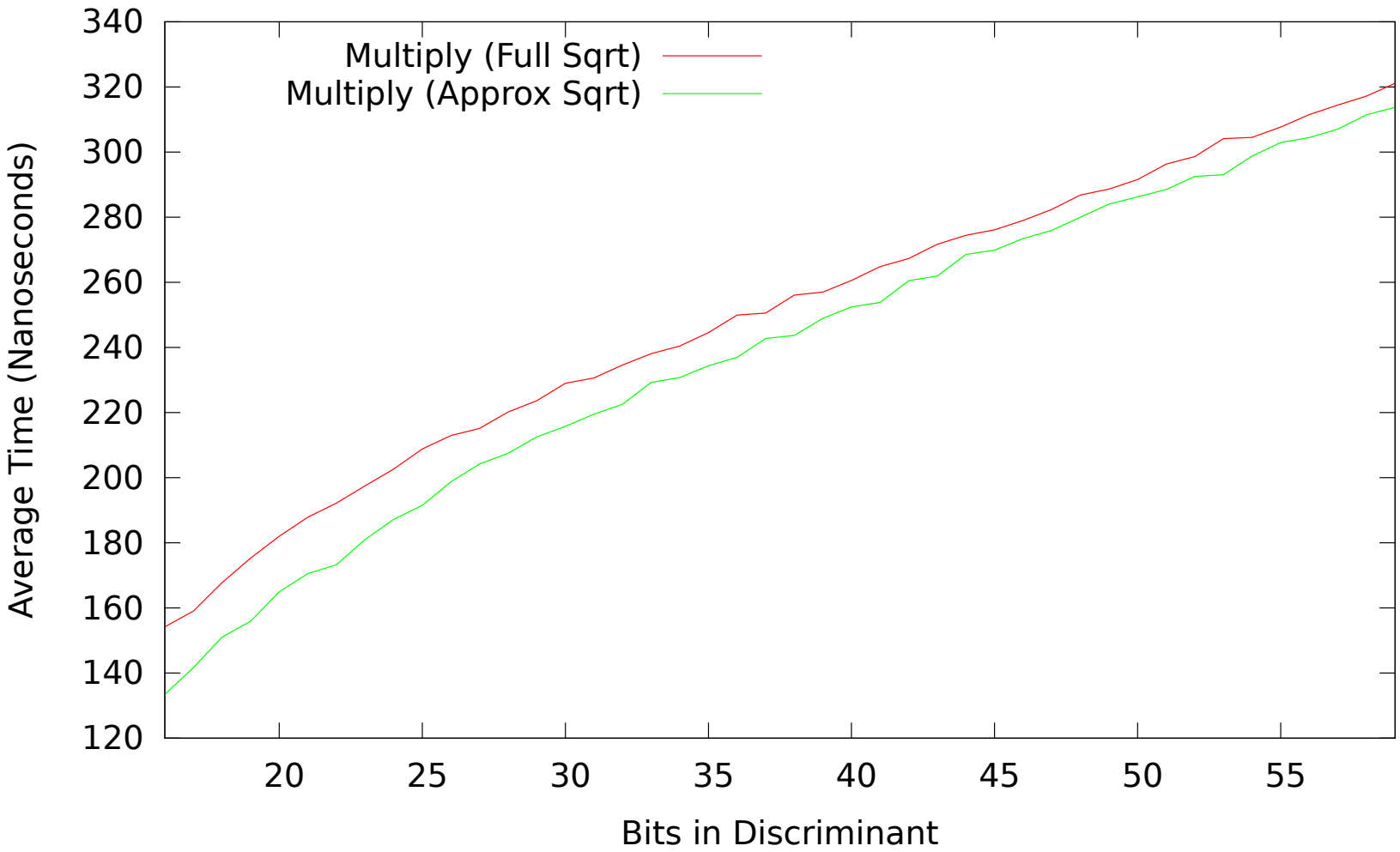
Notice that

$$\begin{aligned} x^{1/2} &= 2^{(\log_2 x)/2} &&\approx 2^{\lfloor \lfloor \log_2 x + 1 \rfloor / 2 \rfloor} \\ \Rightarrow \quad x/x^{1/2} &= x/2^{(\log_2 x)/2} &&\approx x/2^{\lfloor \lfloor \log_2 x + 1 \rfloor / 2 \rfloor}, \end{aligned}$$

which is approximated by shifting x right by $\lfloor \lfloor \log_2 x + 1 \rfloor / 2 \rfloor$ bits.

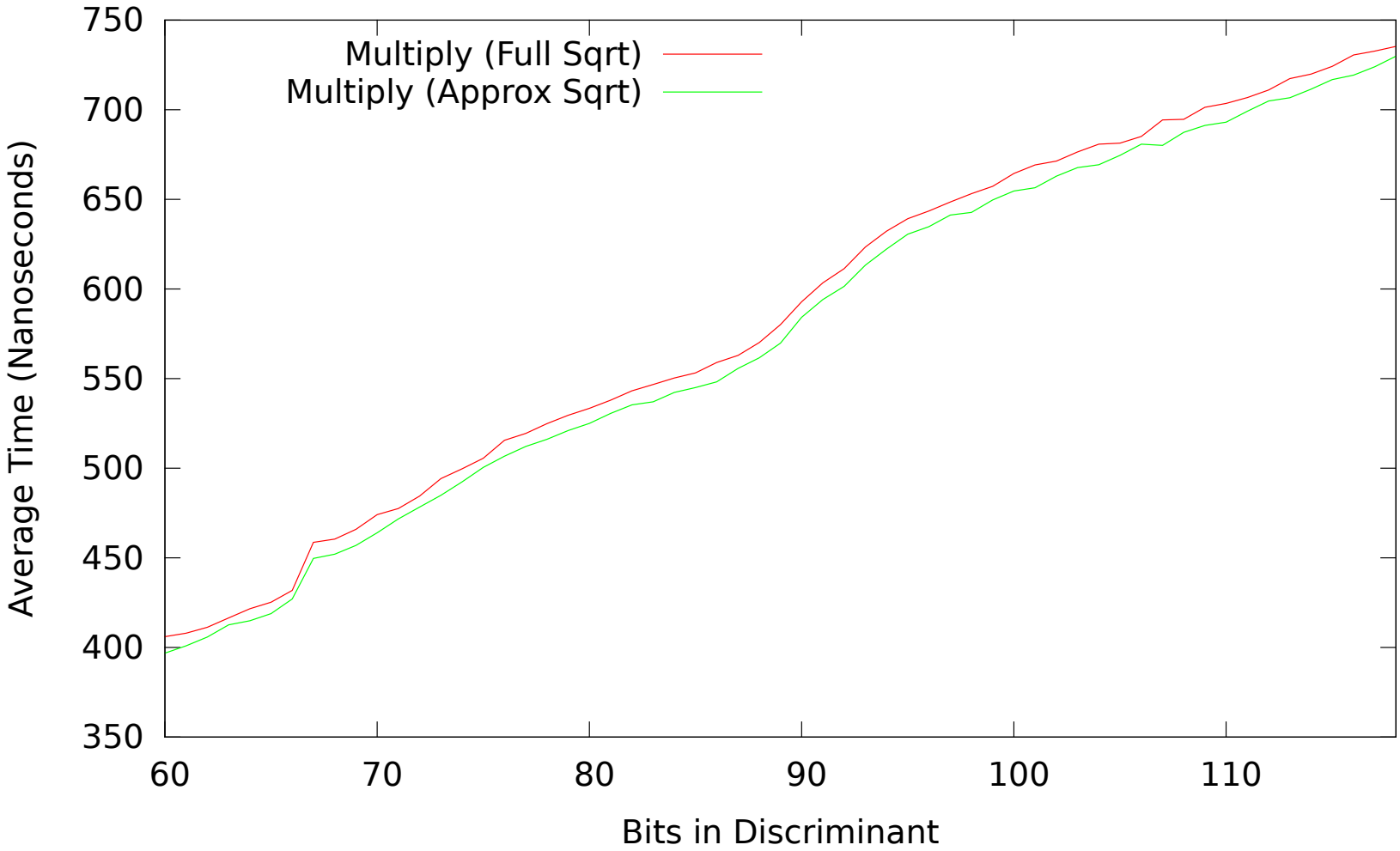
Square Root Approximation

64-bit ideal multiplication



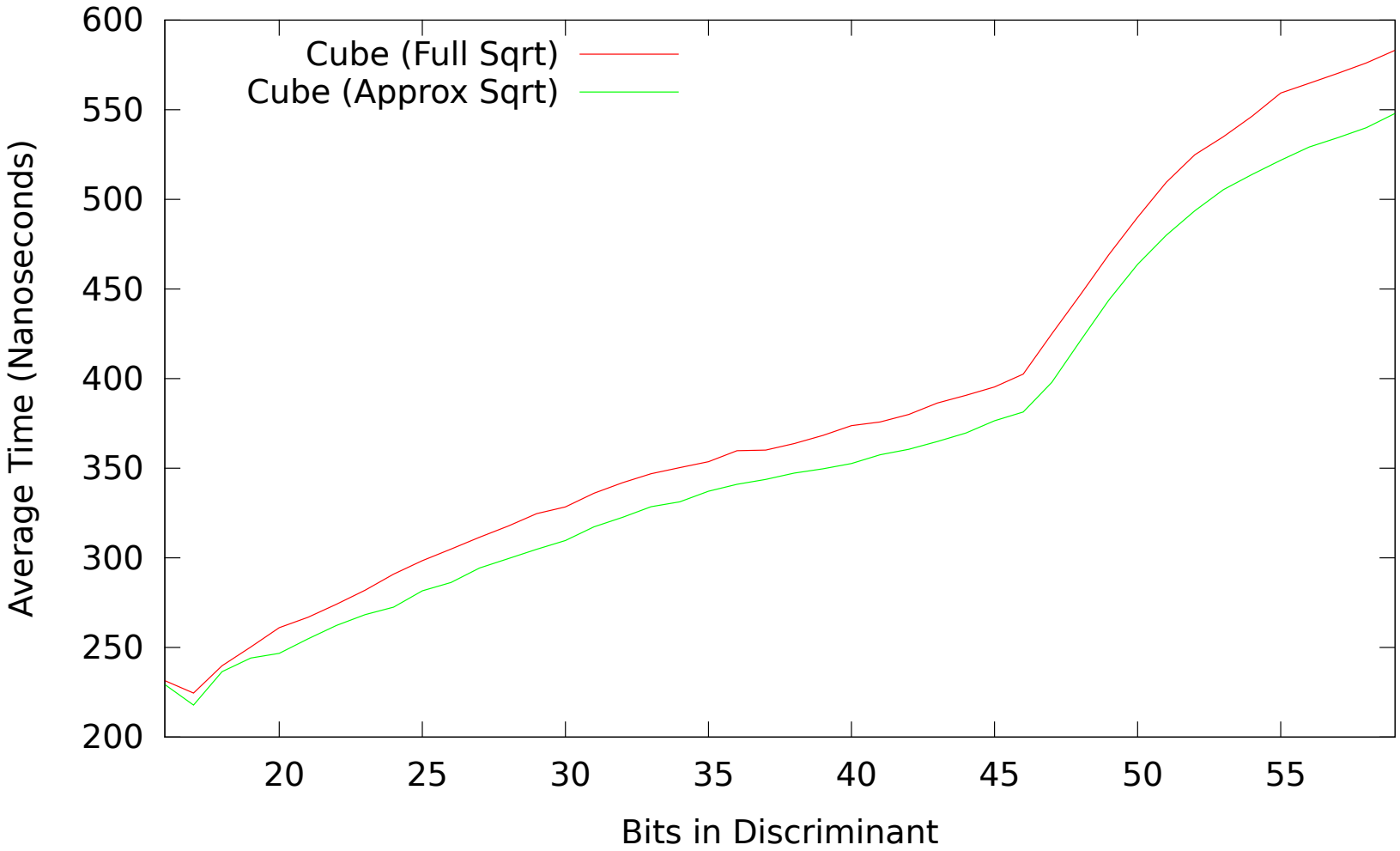
Square Root Approximation

128-bit ideal multiplication



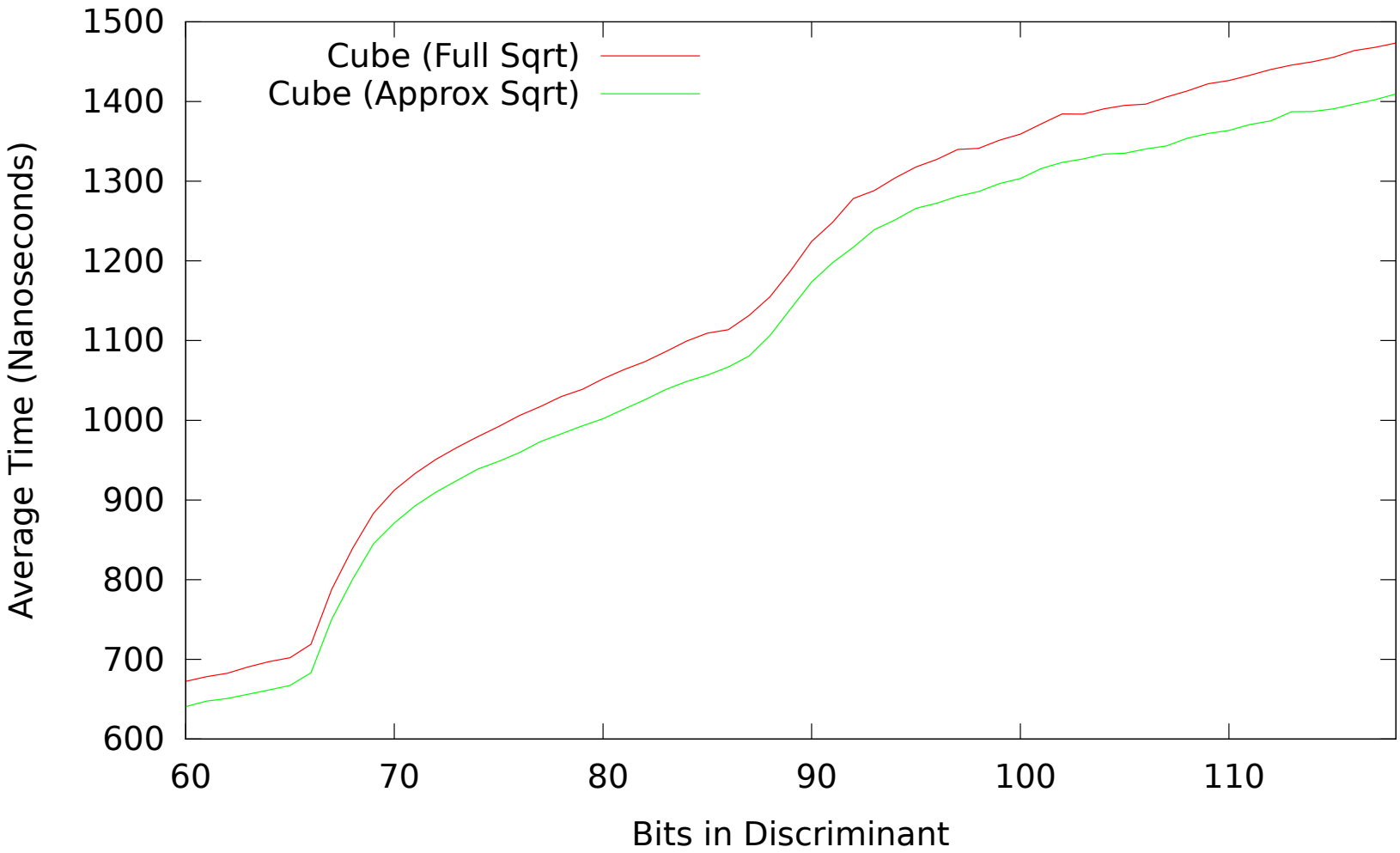
Square Root Approximation

64-bit ideal cubing



Square Root Approximation

128-bit ideal cubing



Large Intermediates During Cubing

Ideal Cubing (Algorithm 2.4, Page 19).

$$U = Y'c_1(Y'(b_1 - Y'c_1a_1) - 2) \bmod {a_1}^2 \qquad \{\text{line } 4\}$$

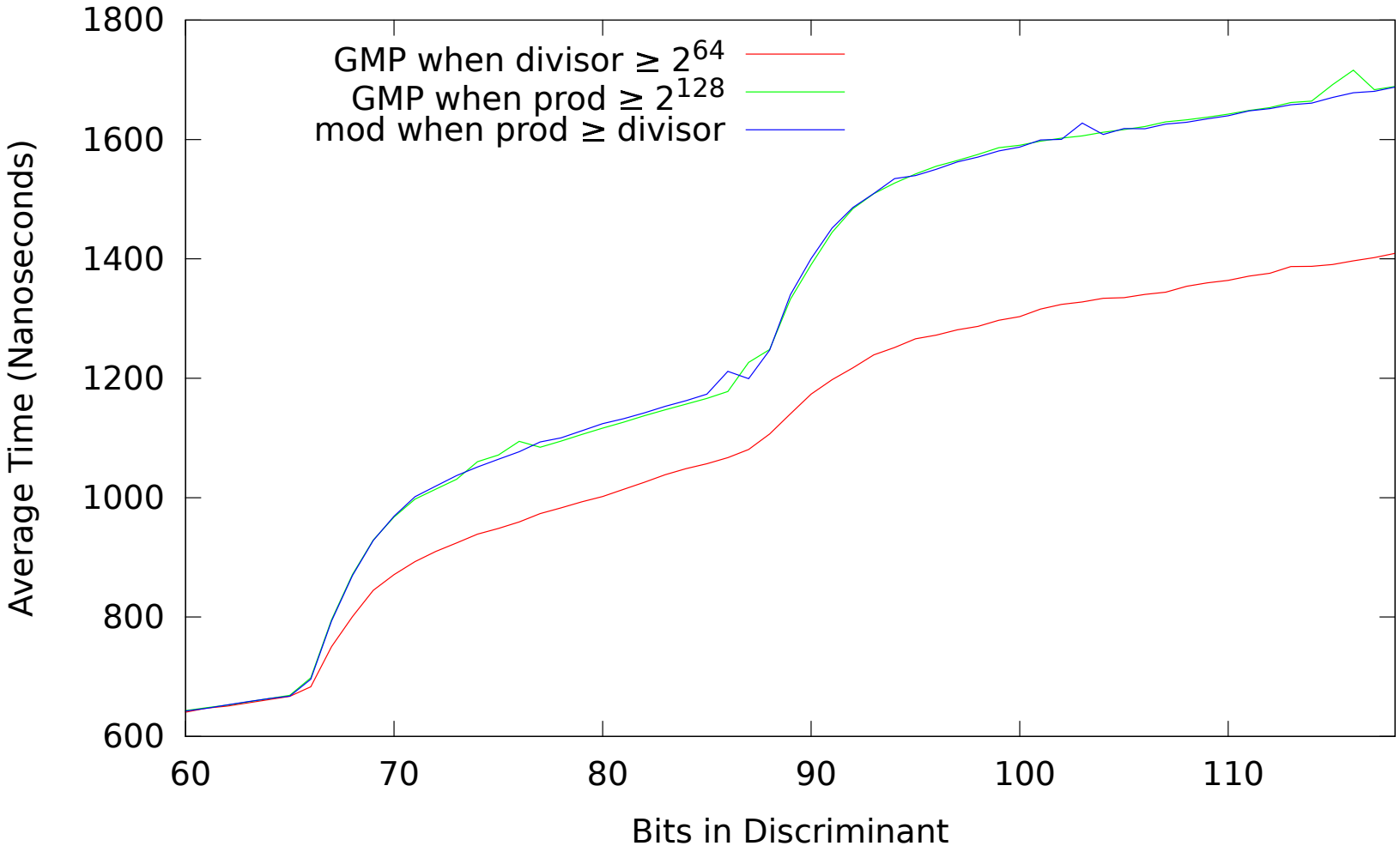
$$U = -c_1(XY'a_1 + Yb_1) \bmod {a_1}^2/s \qquad \{\text{line } 7\}$$

$$M_1 = \frac{(R_ia_1 + C_iUa_1)}{{a_1}^2} \qquad \{\text{line } 19\}$$

$$M_2 = \frac{R_i(b_1 + Ua_1) - sC_ic_1}{{a_1}^2} \qquad \{\text{line } 20\}$$

The problem is $a_1 \leq \sqrt{|\Delta|/3}$ so ${a_1}^2 \leq |\Delta|/3$.

GMP with 128-bit Cubing



Exponentiation by Odd Power Primorials

The goal is to compute

$$[\mathfrak{a}]^E$$

for some ideal class $[\mathfrak{a}]$ and exponent

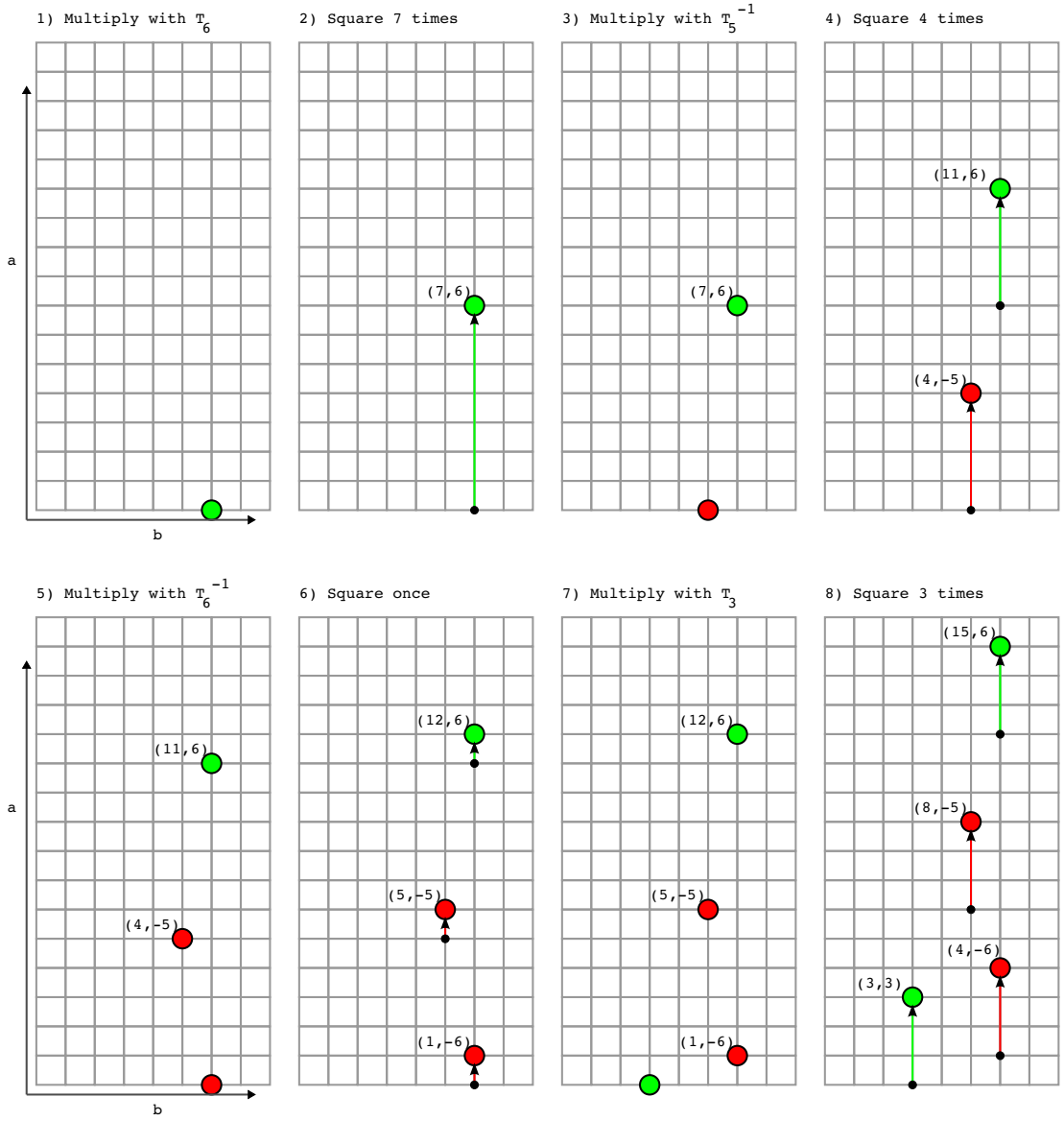
$$E = \prod_{i=2}^{k} {p_i}^{e_i}$$

where p_i is the i^{th} prime.

Representations Tested

Representations tested:

- Binary, Non-Adjacent Form (for reference)
- Right-to-Left and $2^2 3^2$ Windowed Right-to-Left (Ciet et al.).
- Left-to-Right (Berthé and Imbert).
- Pruned Trees (Doche and Habsieger).
- Our Left-to-Right Best Approximations
  (combines Left-to-Right and Pruned Trees)


1) Multiply with T_6
2) Square 7 times
3) Multiply with T_5^{-1}
4) Square 4 times
a
b
(7,6)
(7,6)
(11,6)
(4,-5)
5) Multiply with T_6^{-1}
6) Square once
7) Multiply with T_3
8) Square 3 times
a
b
(11,6)
(4,-5)
(12,6)
(5,-5)
(1,-6)
(12,6)
(5,-5)
(1,-6)
(15,6)
(8,-5)
(4,-6)
(3,3)

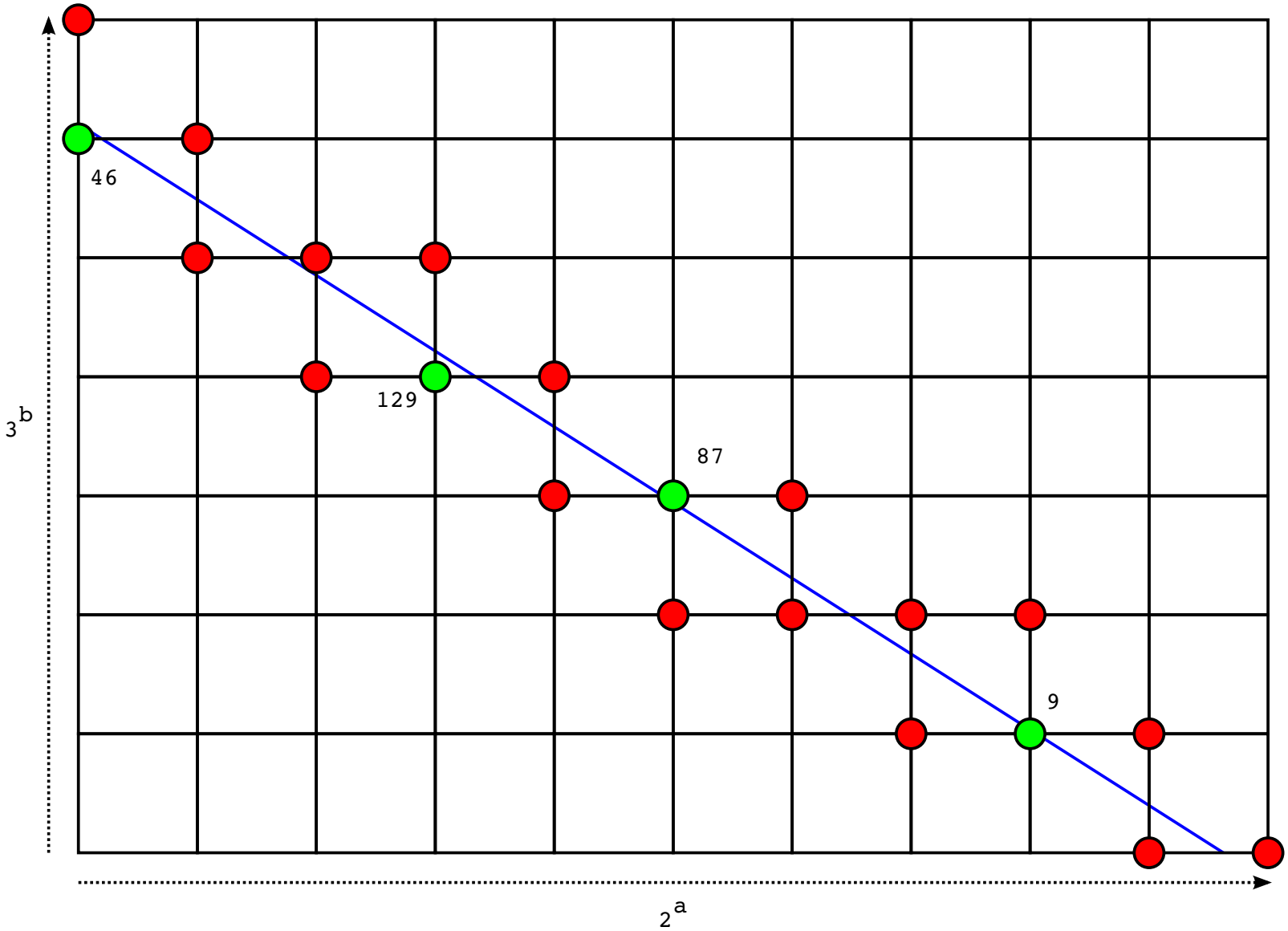
Left-to-Right Best Approximations

Left-to-Right best approximations is a technique combining the greedy approach of Berthé and Imbert with the tree based approach of Doche and Habsieger.

- Iterate on a set of partial representations $x + \sum s_i 2^{a_i} 3^{b_i}$.
- Keep the best $2^a 3^b$ approximations for each x.
- Squares and cubes are bound $a \leq A$, $b \leq B$, and the bounds A and B are iterated.

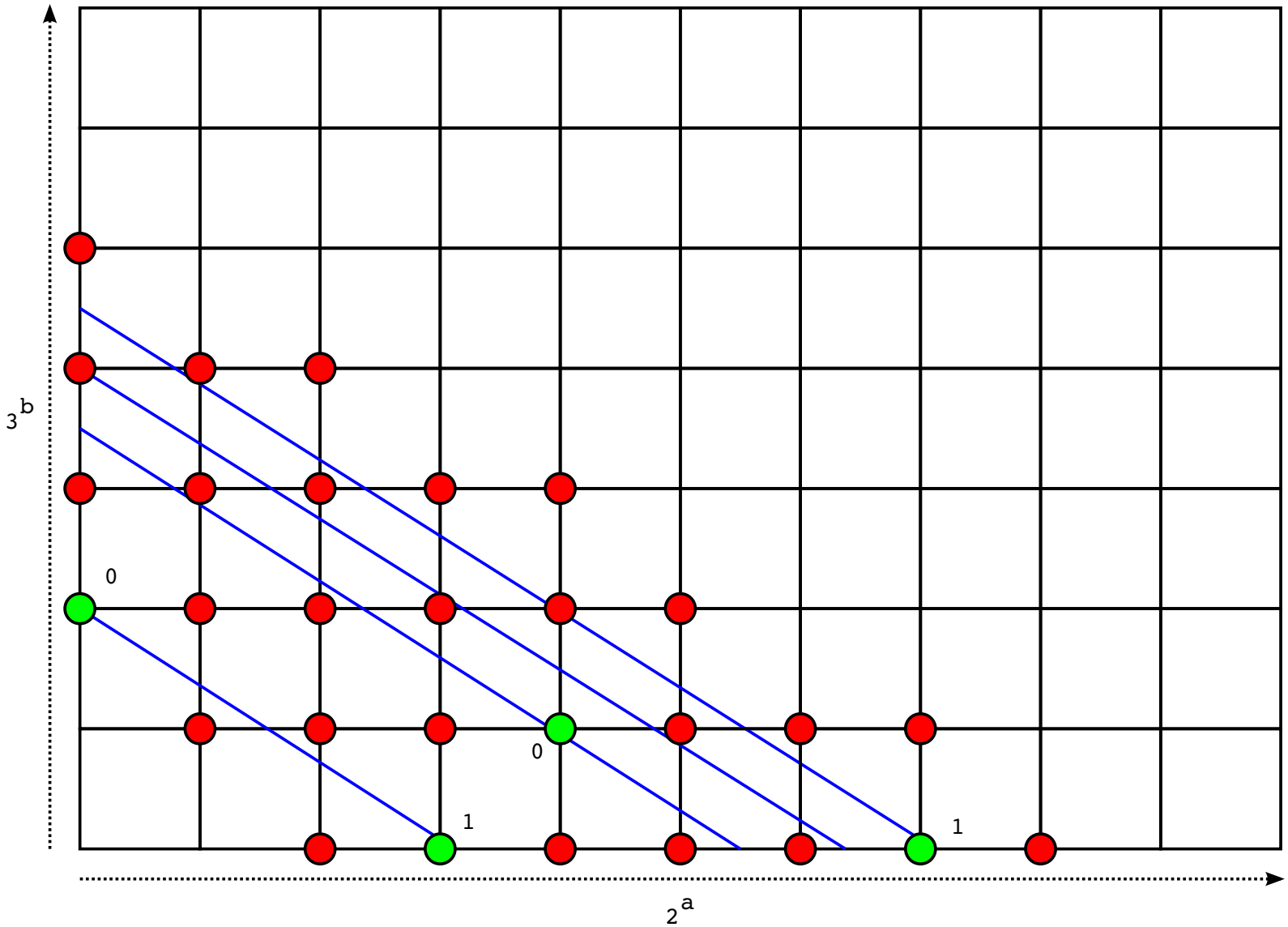
Left-to-Right Best Approximations

Example – First Iteration on 777



Left-to-Right Best Approximations

Example – Second Iteration on 9, 48, 87, and 129

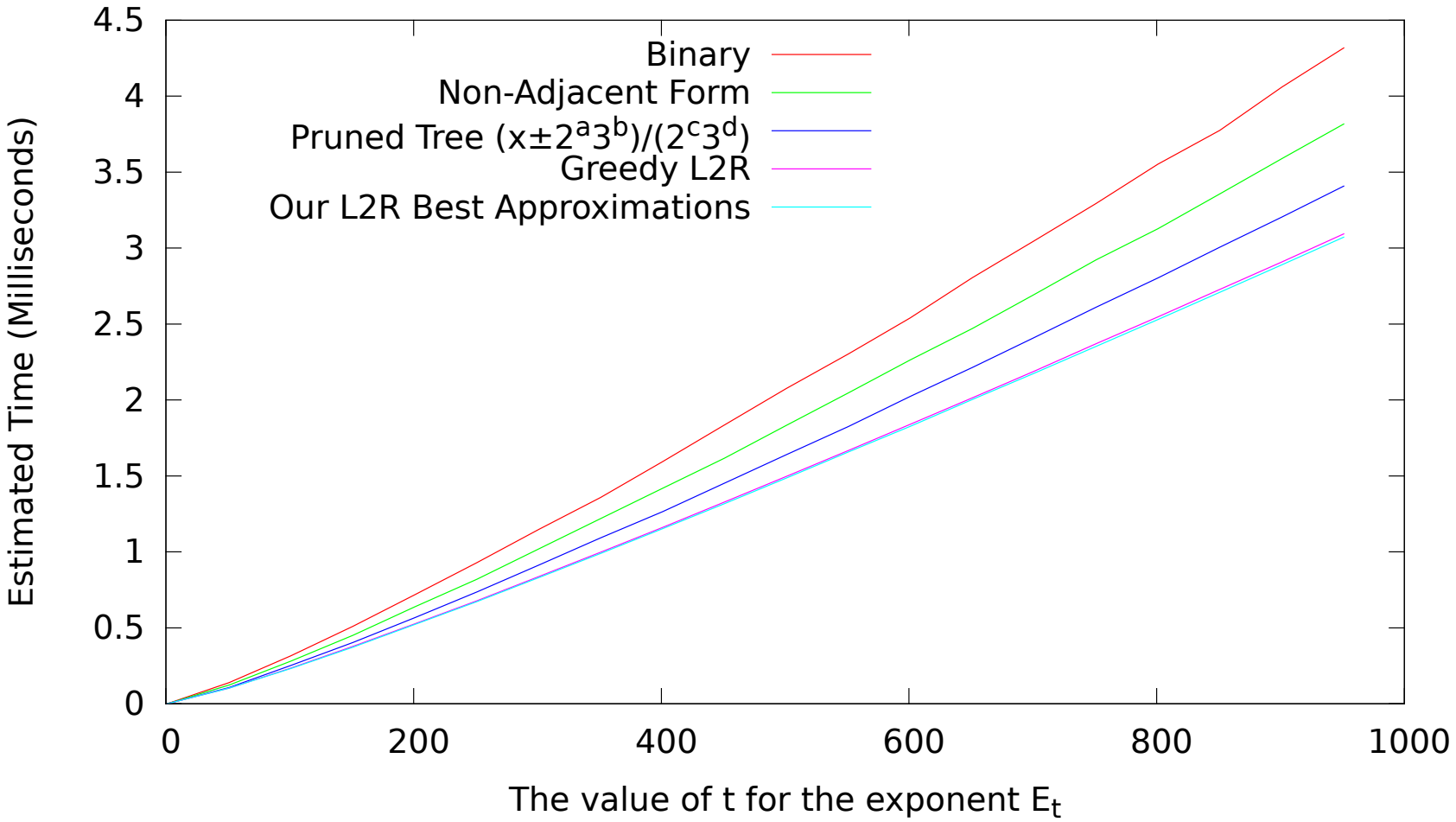


2,3 Representations for $E = 777$:

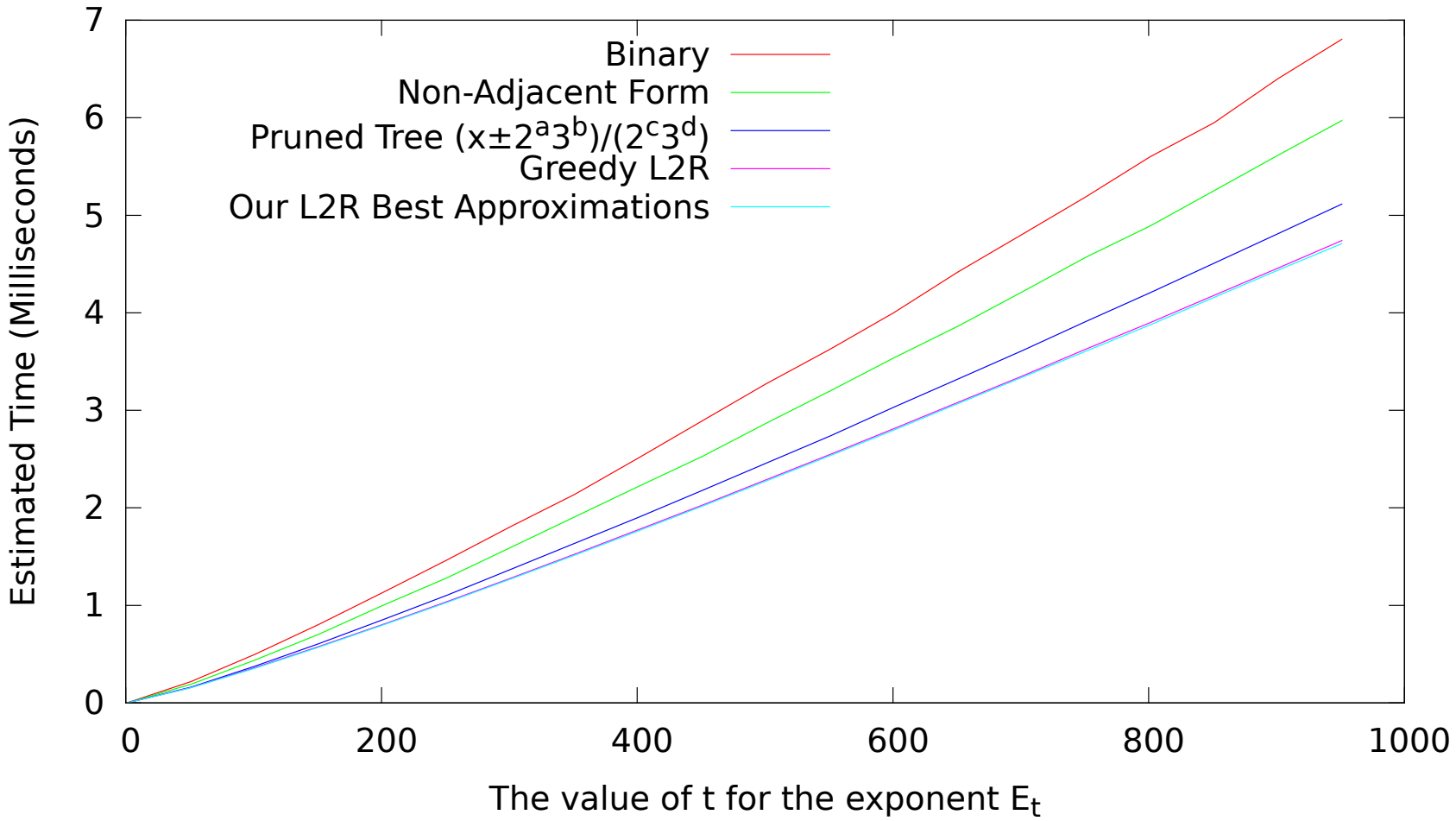
$$\begin{aligned} 2^8 3^1 + 2^0 3^2 &\quad 8 \text{ squares, } 2 \text{ cubes, } 1 \text{ multiplication} \\ 2^0 3^6 + 2^4 3^1 &\quad 4 \text{ squares, } 6 \text{ cubes, } 1 \text{ multiplication} \\ 2^8 3^1 + 2^3 3^0 + 2^0 3^0 &\quad 8 \text{ squares, } 1 \text{ cube, } 2 \text{ multiplications} \\ 2^3 3^4 + 2^7 3^0 + 2^0 3^0 &\quad 7 \text{ squares, } 4 \text{ cubes, } 2 \text{ multiplications} \end{aligned}$$

$2^8 3^1 + 2^3 3^0 + 2^0 3^0$ possibly preferred since multiplication is cheaper than squaring.

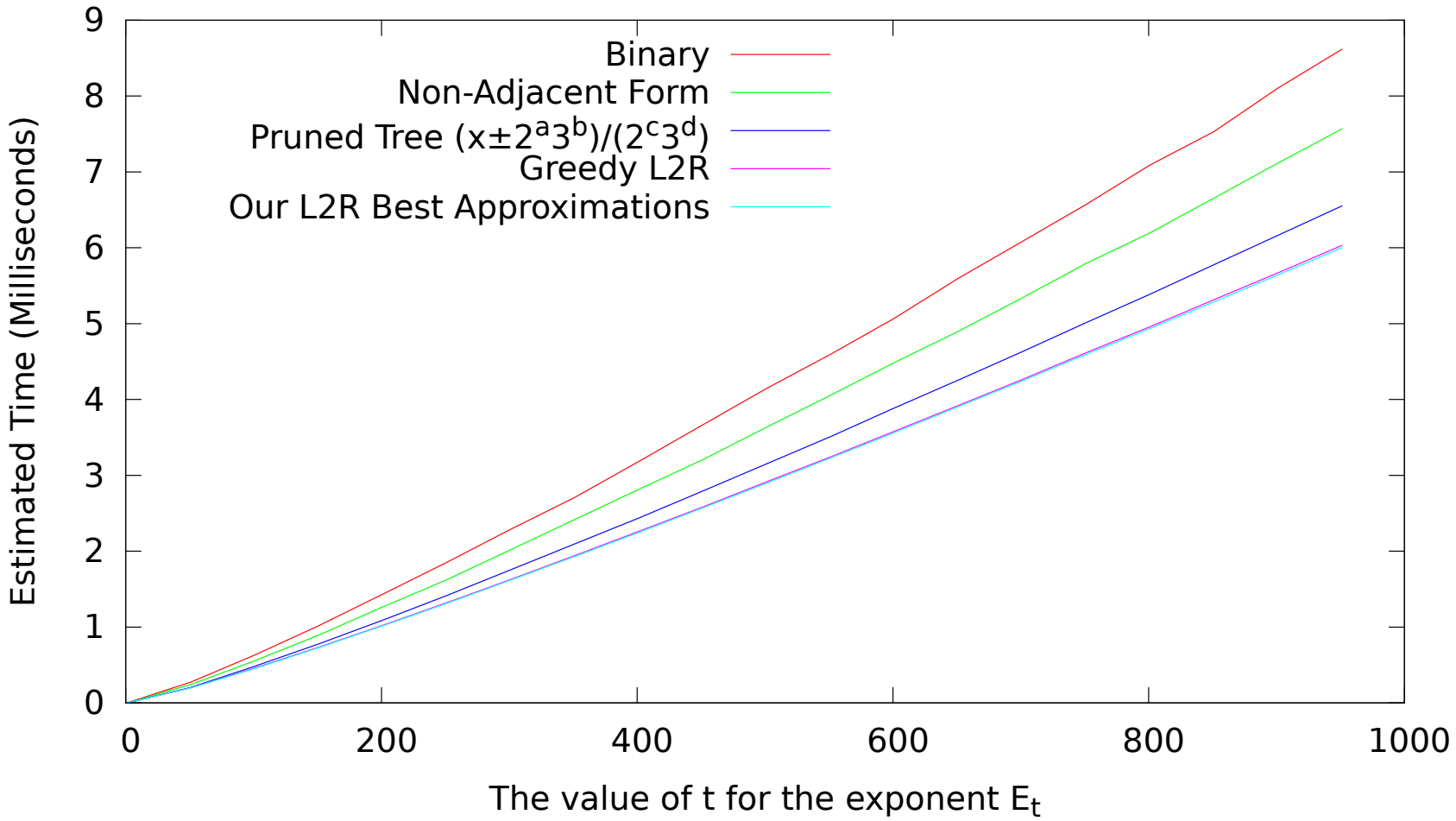
Exponentiation Results (16-bit Discriminants)



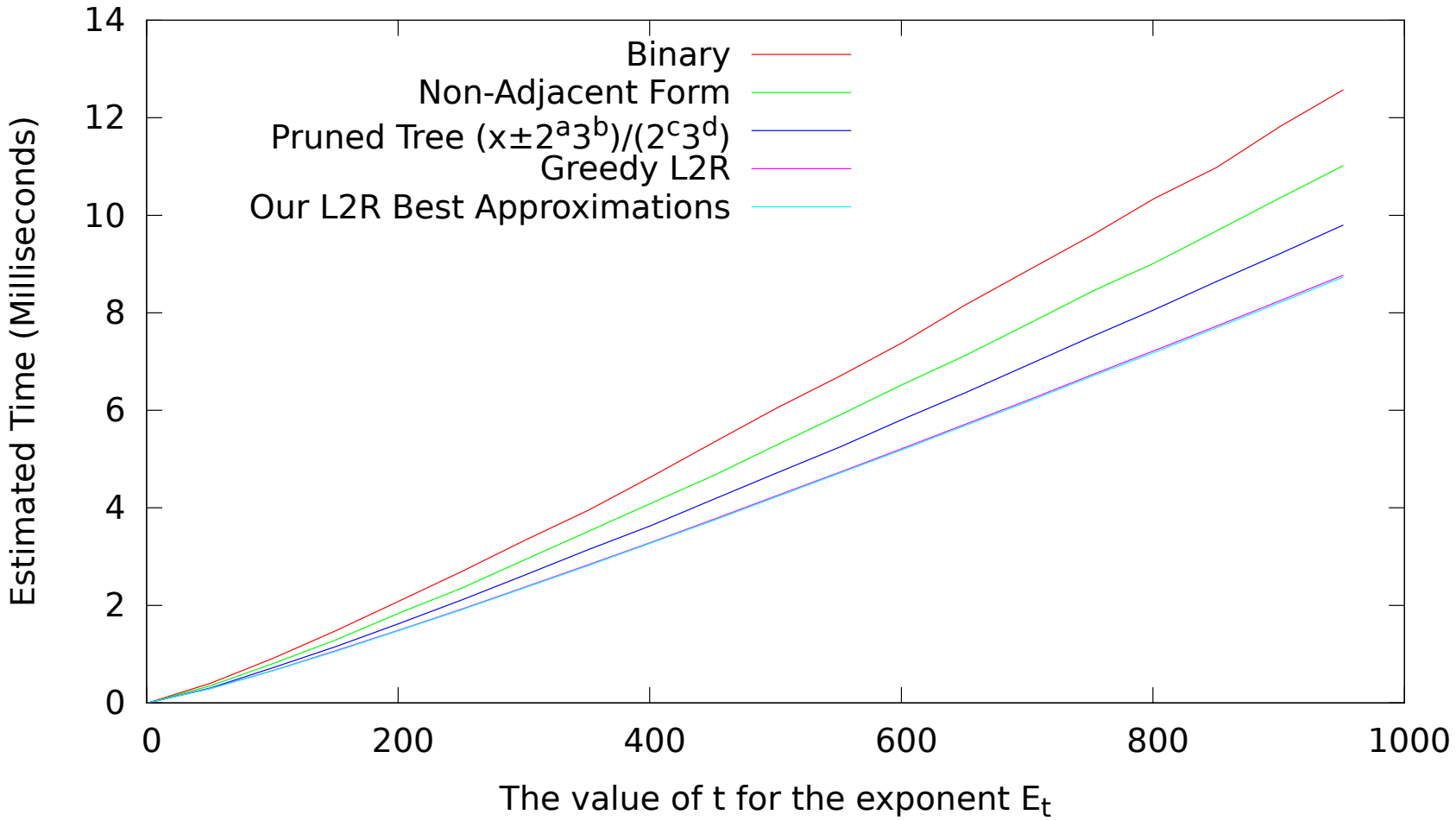
Exponentiation Results (32-bit Discriminants)



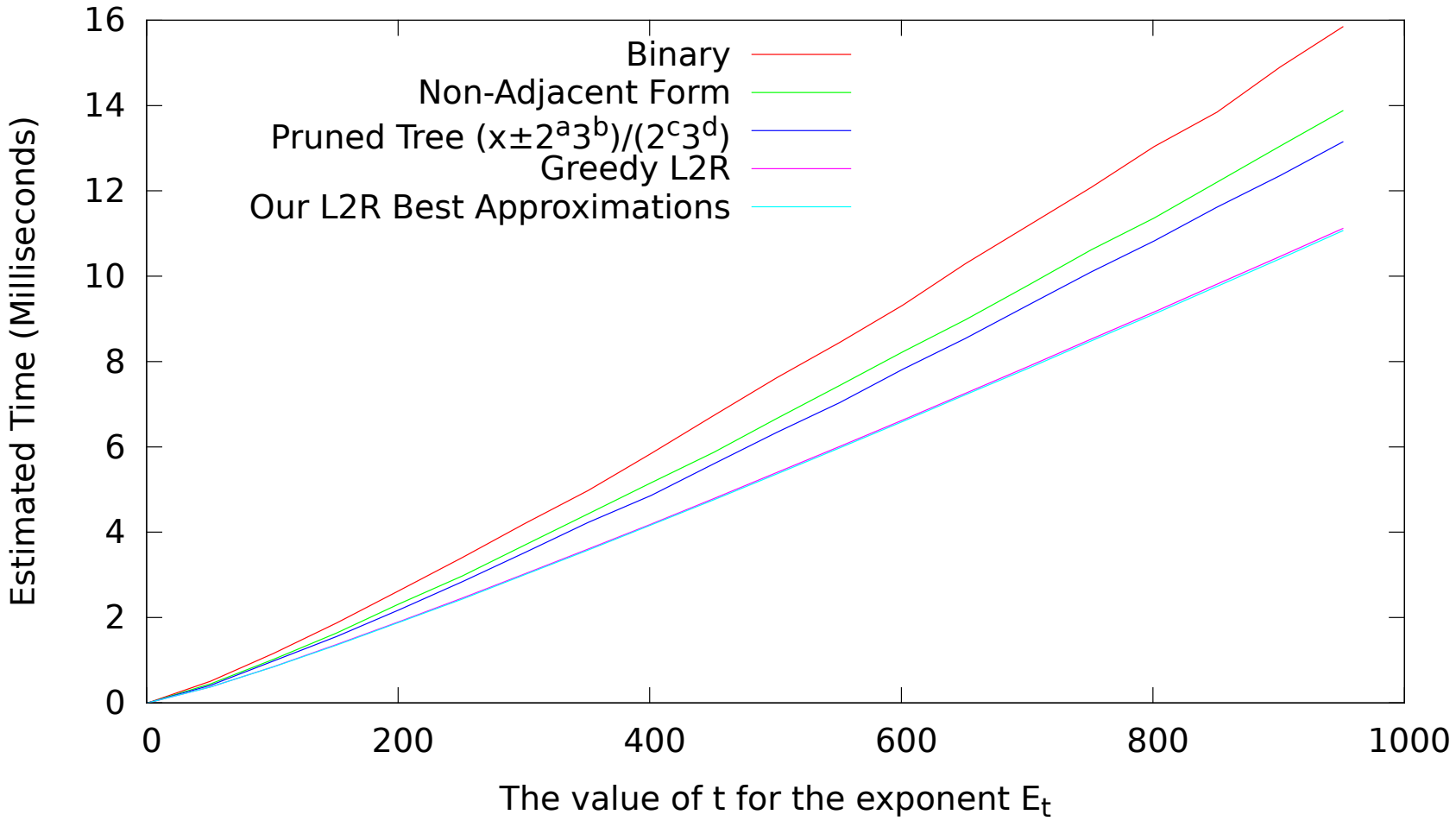
Exponentiation Results (48-bit Discriminants)



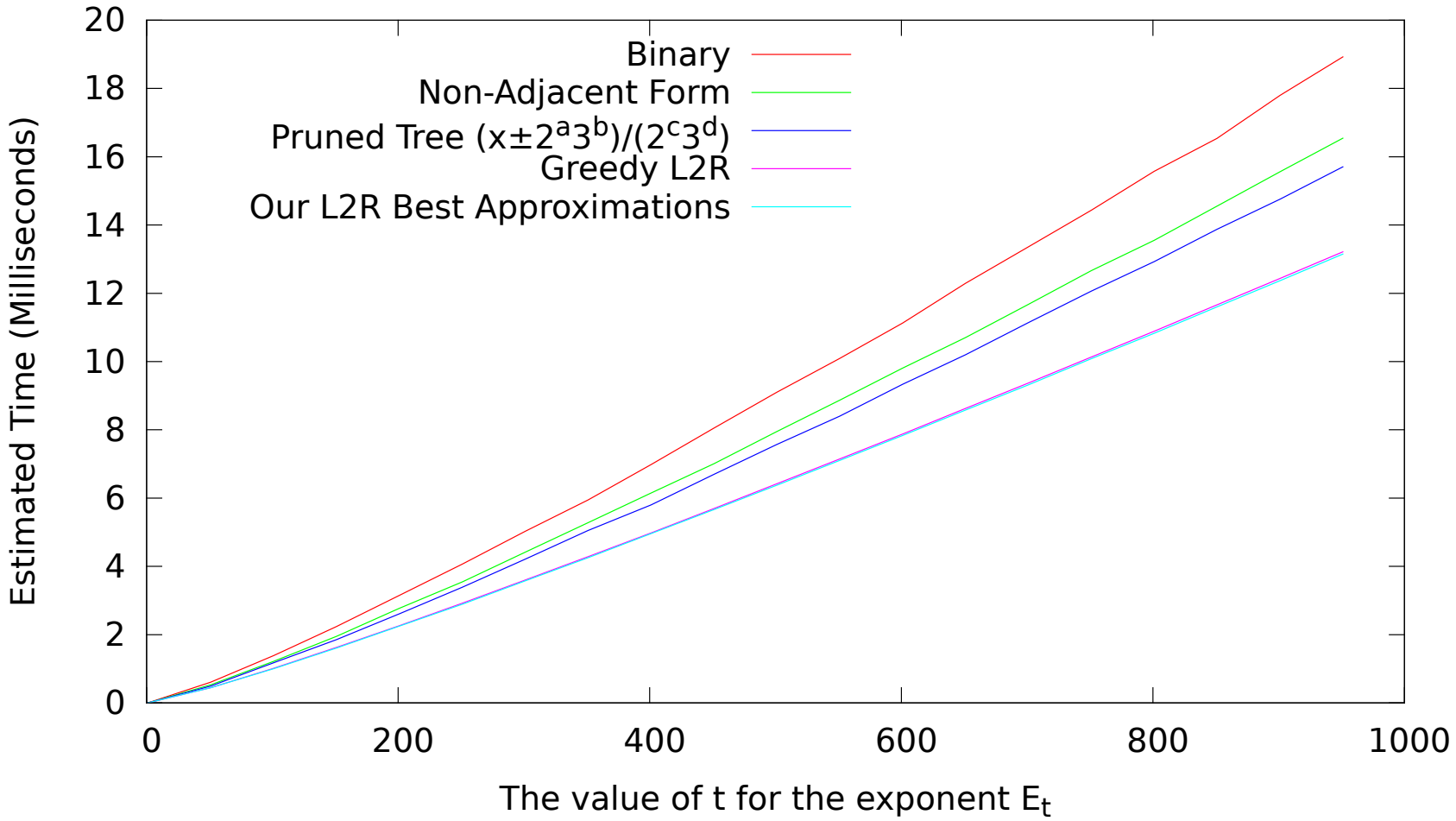
Exponentiation Results (64-bit Discriminants)



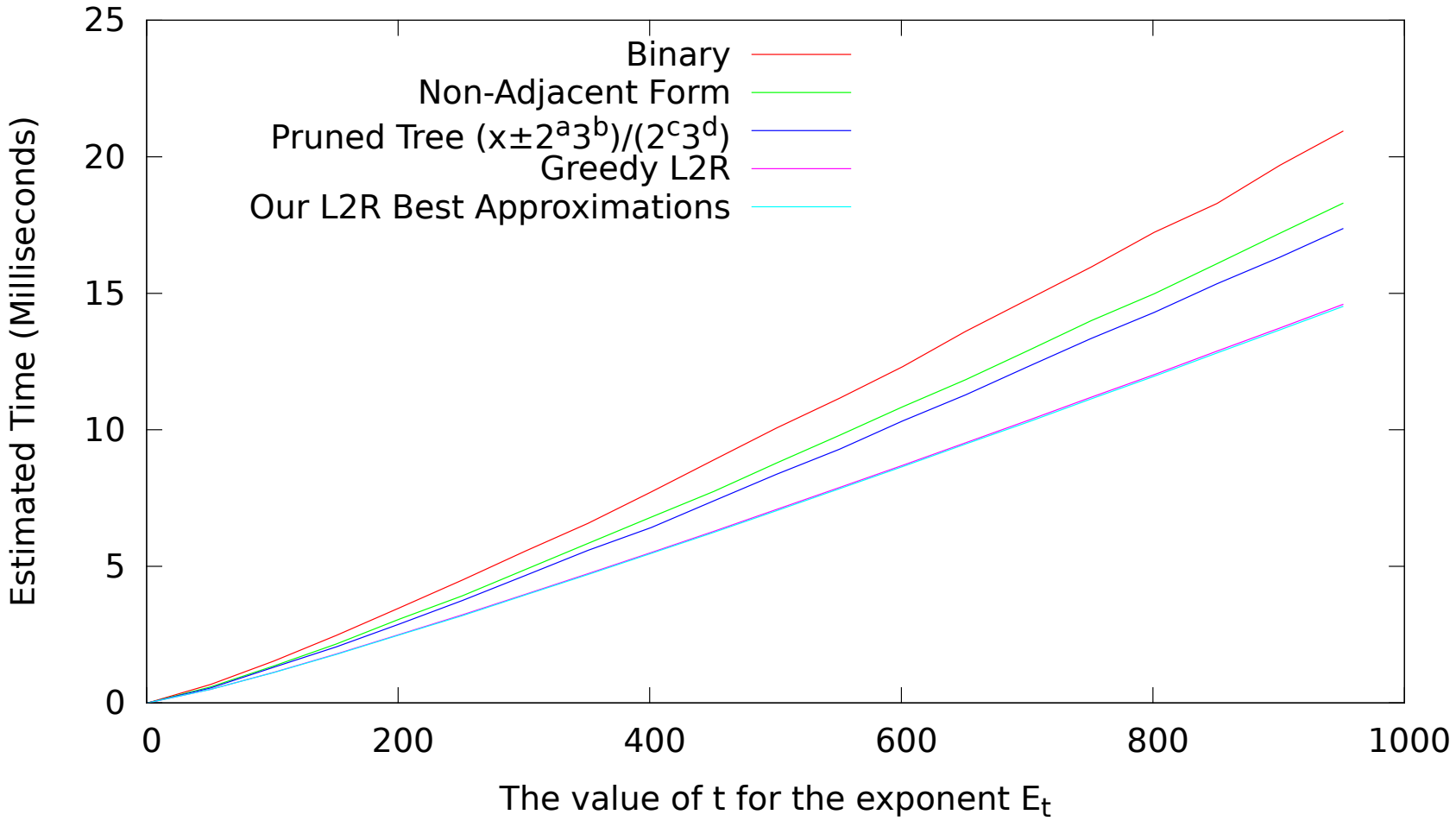
Exponentiation Results (80-bit Discriminants)



Exponentiation Results (96-bit Discriminants)



Exponentiation Results (112-bit Discriminants)



SuperSPAR

SuperSPAR is an integer factoring algorithm based on arithmetic in the ideal class group of imaginary quadratic integers.

- Extends SPAR using a bounded primorial steps search.
- Uses improvements to ideal class group arithmetic and 2,3 representations of power primorials.
- Fastest median time for integers 49-bits to 68-bits of size.
- Fastest average time for integers 49-bits to 64-bits of size.

Median times do not appear in thesis.

Theoretically Optimal Parameters for Exponentiation Stage

$\log_2 N$	$r = \sqrt{\ln N / \ln \ln N}$	$\lfloor N^{1/2r} \rfloor$	p_t	t
16	2.14693	13	13	6
32	2.67523	63	61	18
48	3.08112	221	211	47
64	3.42016	655	653	119
80	3.71609	1738	1733	270
96	3.98139	4258	4253	583
112	4.22355	9802	9791	1208
128	4.44745	21473	21467	2407

Theoretically Optimal Search Bounds

$\log_2 N$	$\lfloor N^{1/2r} \rfloor$	$2m\phi(P_w)$	$m^2\phi(P_w)P_w$	m	w
16	13	16	480	1	3
32	63	64	7680	4	3
48	221	224	94080	14	3
64	655	656	806880	41	3
80	1738	1824	7277760	19	4
96	4258	4320	40824000	45	4
112	9802	10080	222264000	105	4
128	21473	22080	1173110400	23	5

Factorization of the Order

Example #1

$N = 92233754336196605 27, k = 1, \Delta = -kN$

- $\mathrm{ord}([\mathfrak{p}_{19}]) = 2^2 \cdot 13 \cdot 2770667$
- $\mathrm{ord}([\mathfrak{p}_{37}]) = 2^3 \cdot 3 \cdot 13 \cdot 2770667$
- $\mathrm{ord}([\mathfrak{p}_{43}]) = 2^4 \cdot 3 \cdot 13 \cdot 2770667$
- $\mathrm{ord}([\mathfrak{p}_{47}]) = 2^3 \cdot 3 \cdot 13 \cdot 2770667$
- $\mathrm{ord}([\mathfrak{p}_{59}]) = 2^4 \cdot 3 \cdot 13 \cdot 2770667$

where $\mathfrak{p}_p = [p, (b + \sqrt{\Delta})/2]$.

Each prime ideal split N.

Factorization of the Order

Example #2

$N = 182782835644284671 83, k = 3, \Delta = -4kN$

- $\text{ord}([\mathfrak{p}_{11}]) = 2 \cdot 3 \cdot 59 \cdot 157 \cdot 1451$
- $\text{ord}([\mathfrak{p}_{17}]) = 2 \cdot 5^2 \cdot 59 \cdot 157 \cdot 1451$
- $\text{ord}([\mathfrak{p}_{23}]) = 2 \cdot 3 \cdot 5^2 \cdot 59 \cdot 157 \cdot 1451$
- $\text{ord}([\mathfrak{p}_{29}]) = 2 \cdot 3 \cdot 59 \cdot 157 \cdot 1451$
- $\text{ord}([\mathfrak{p}_{31}]) = 2 \cdot 5 \cdot 59 \cdot 157 \cdot 1451$

where $\mathfrak{p}_p = [p, (b + \sqrt{\Delta})/2]$.

Only $[\mathfrak{p}_{23}]$ and $[\mathfrak{p}_{29}]$ split N.

Reusing the Order

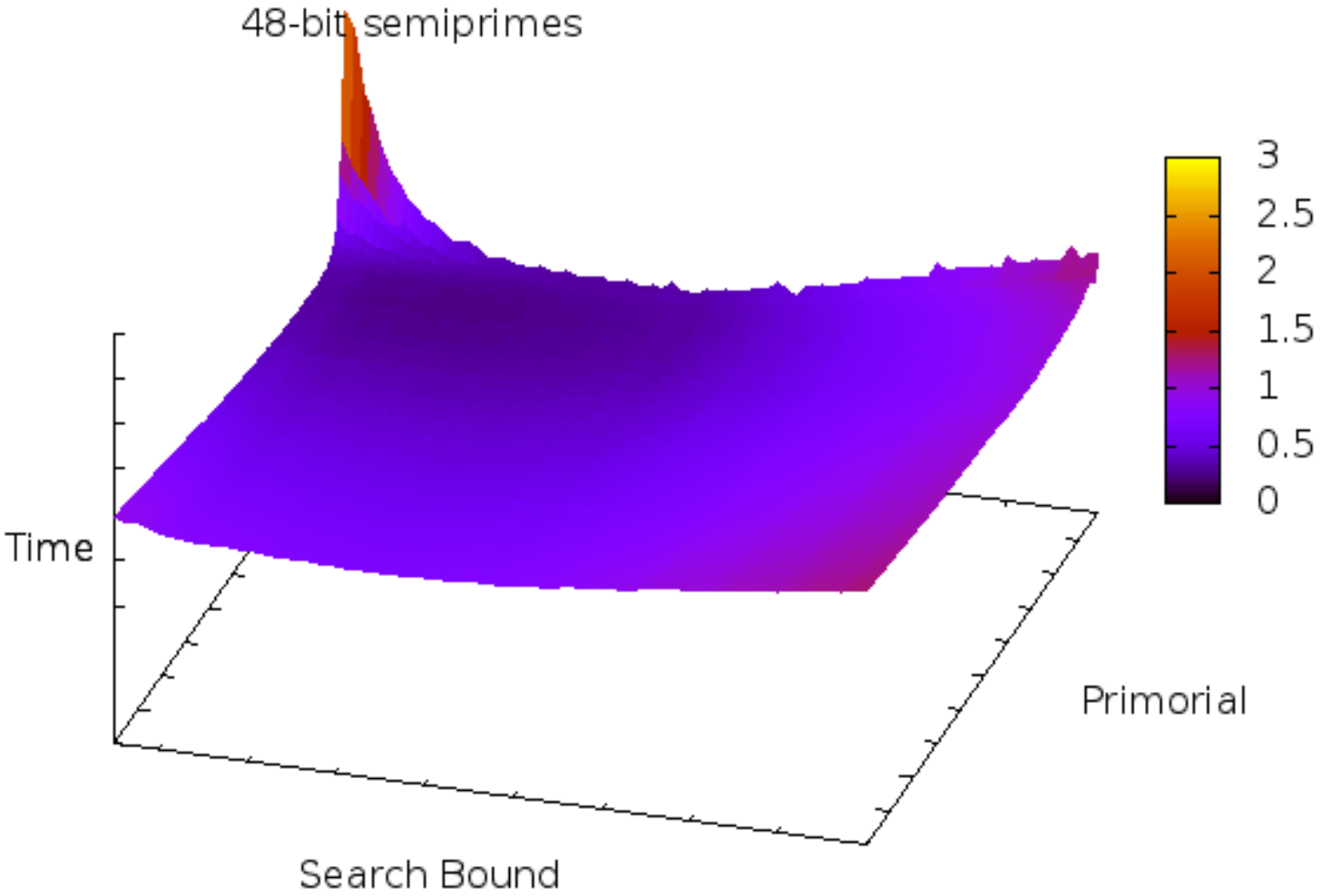
Once the order of an ideal class is known, we skip the search phase.

- The exponentiation stage removed all small primes $\Rightarrow$ search stage not necessary.
- The exponentiation stage did not remove all small primes $\Rightarrow$ stepping coprime will never find the order.

For $N \leq 2^{80}$, this is expected to work better than 97.7% of the time.

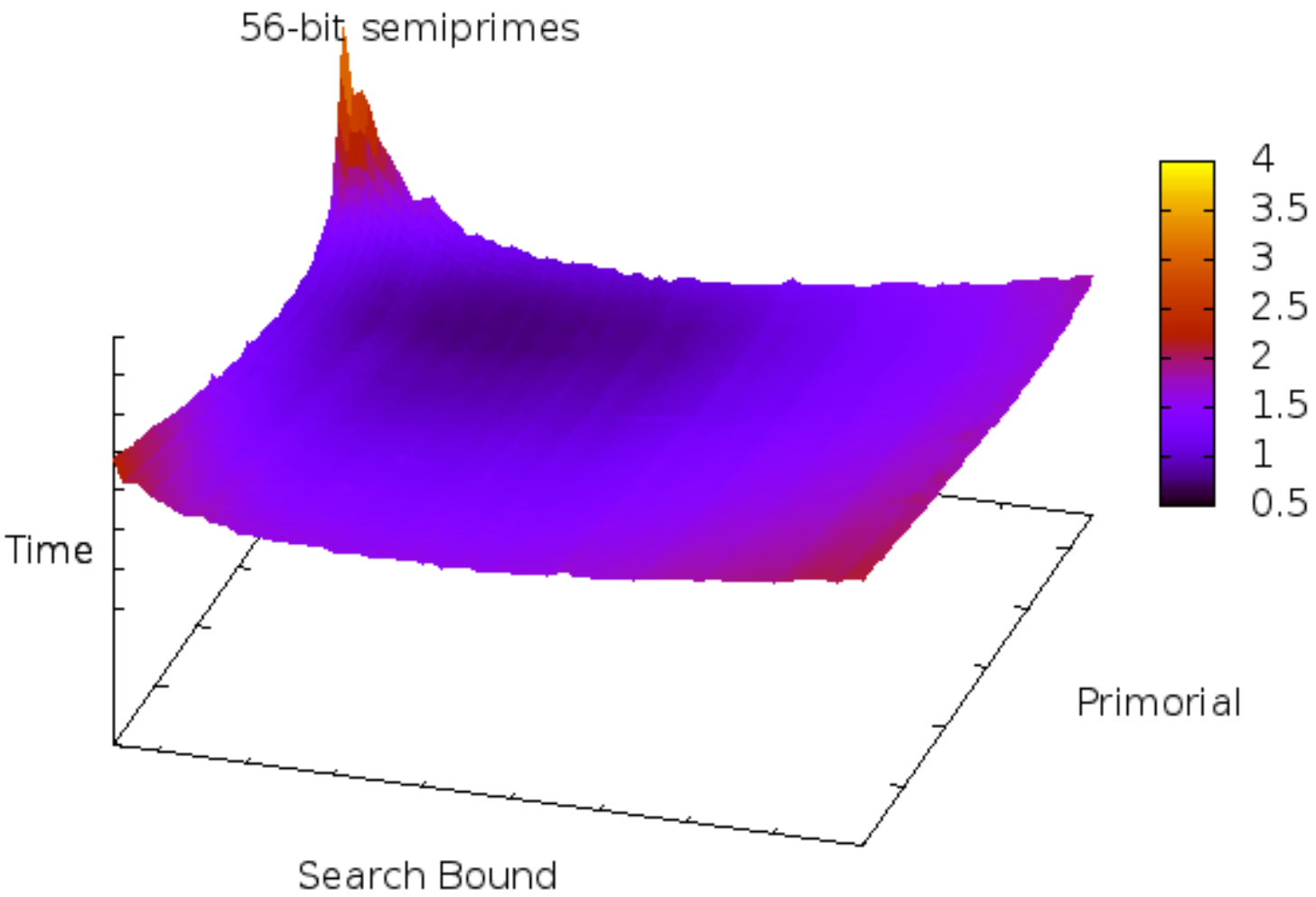
SuperSPAR Search Space

48-bit semiprimes



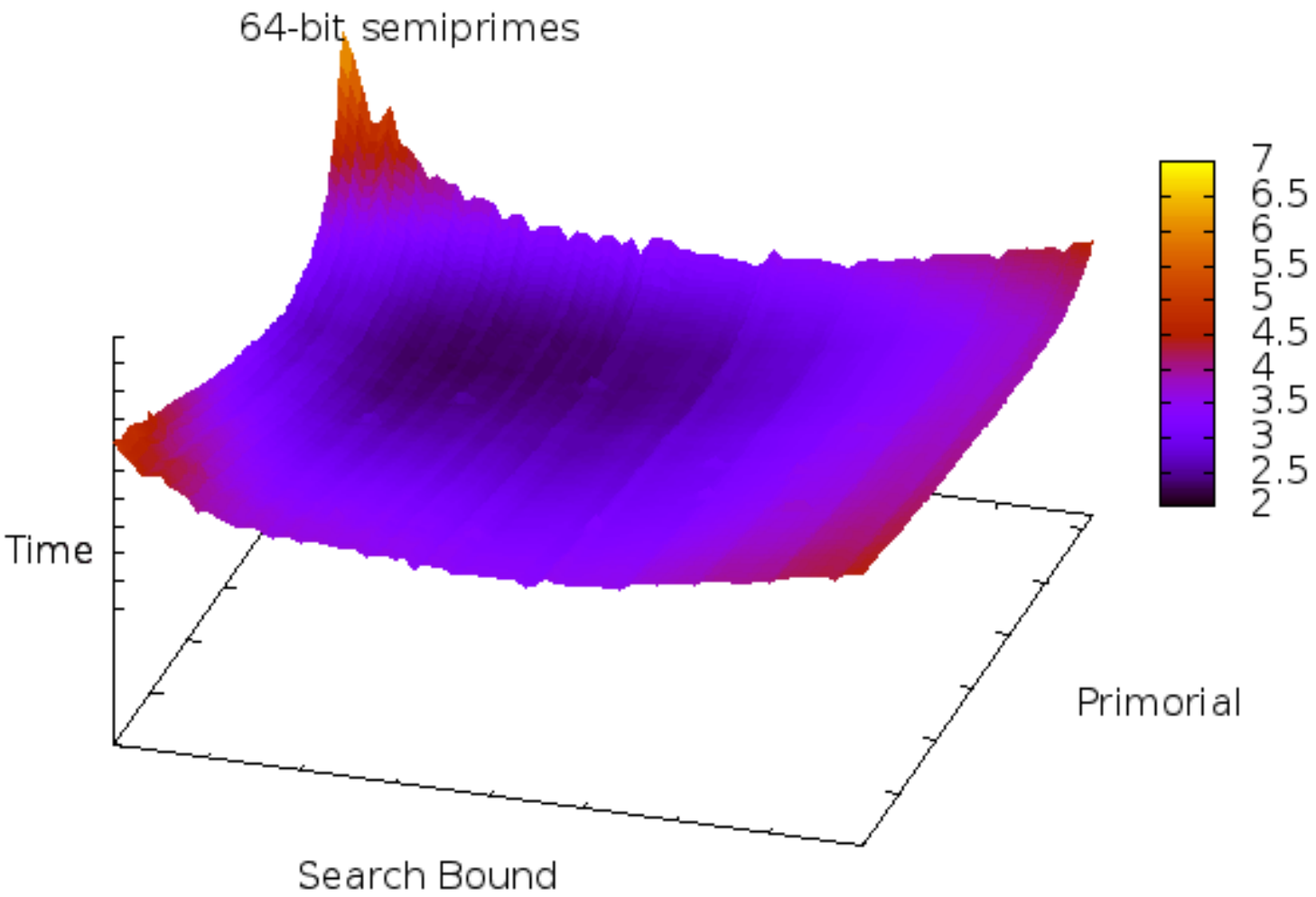
SuperSPAR Search Space

56-bit semiprimes



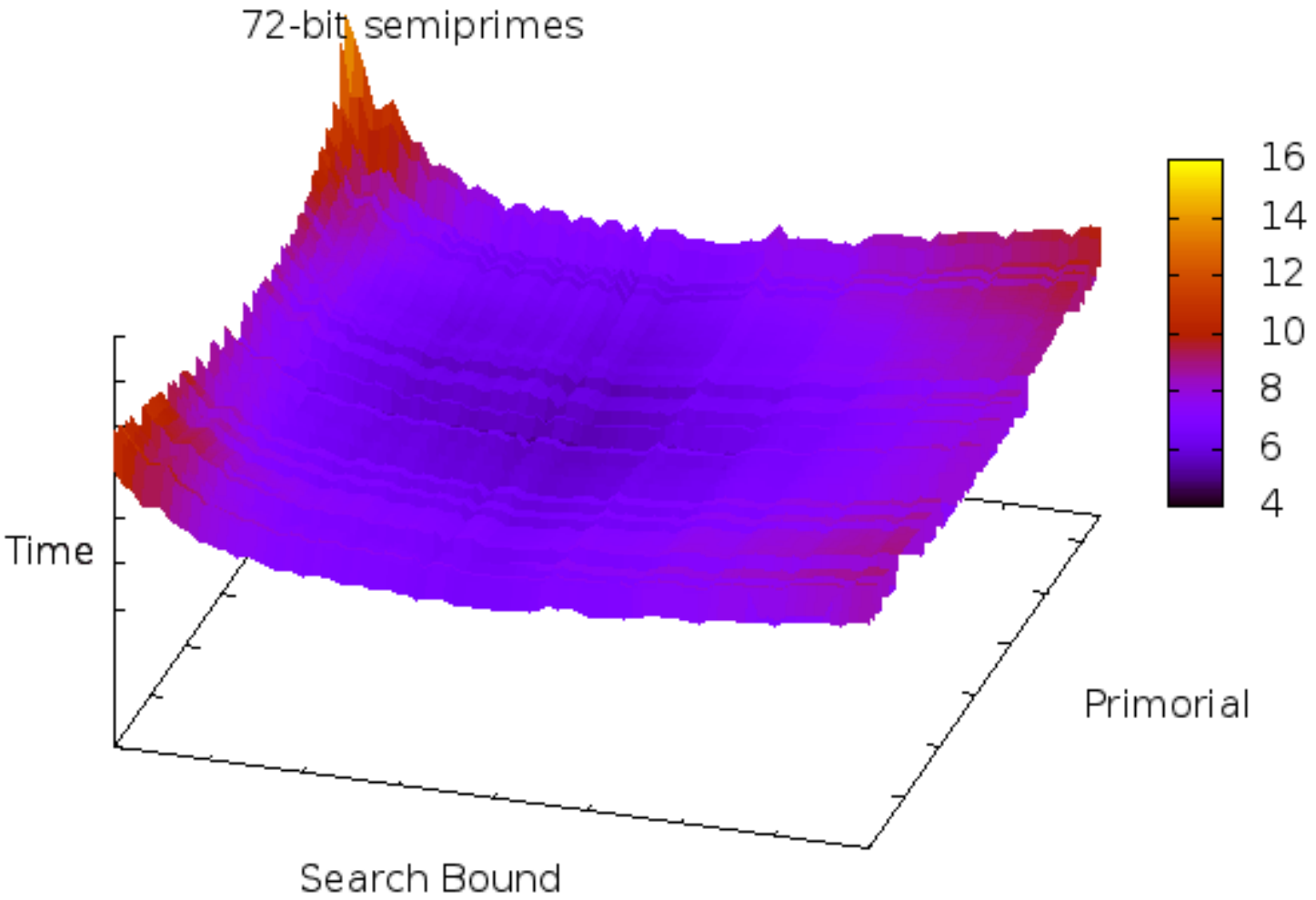
SuperSPAR Search Space

64-bit semiprimes



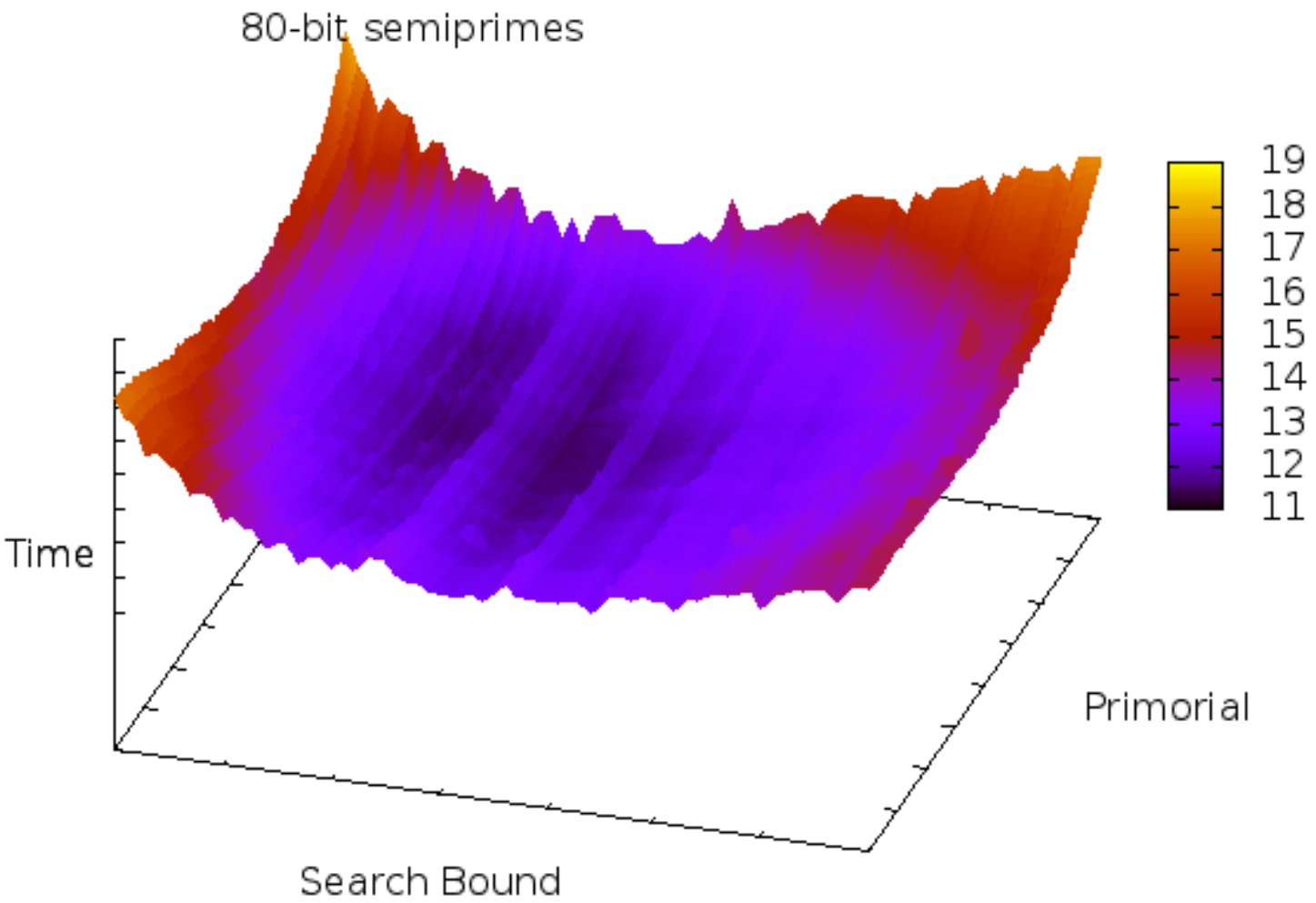
SuperSPAR Search Space

72-bit semiprimes



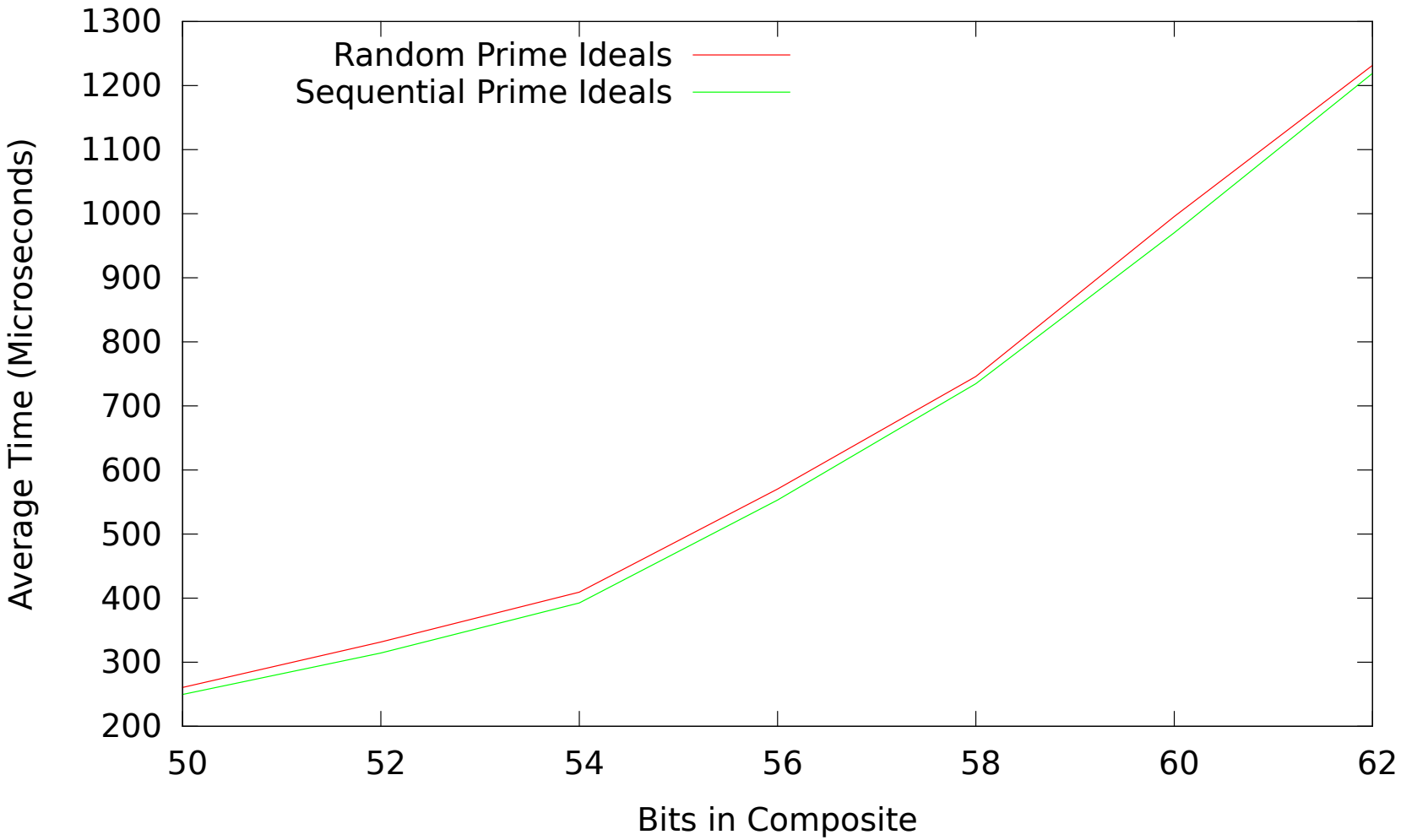
SuperSPAR Search Space

80-bit semiprimes



SuperSPAR Prime Ideals

Sequential vs Random

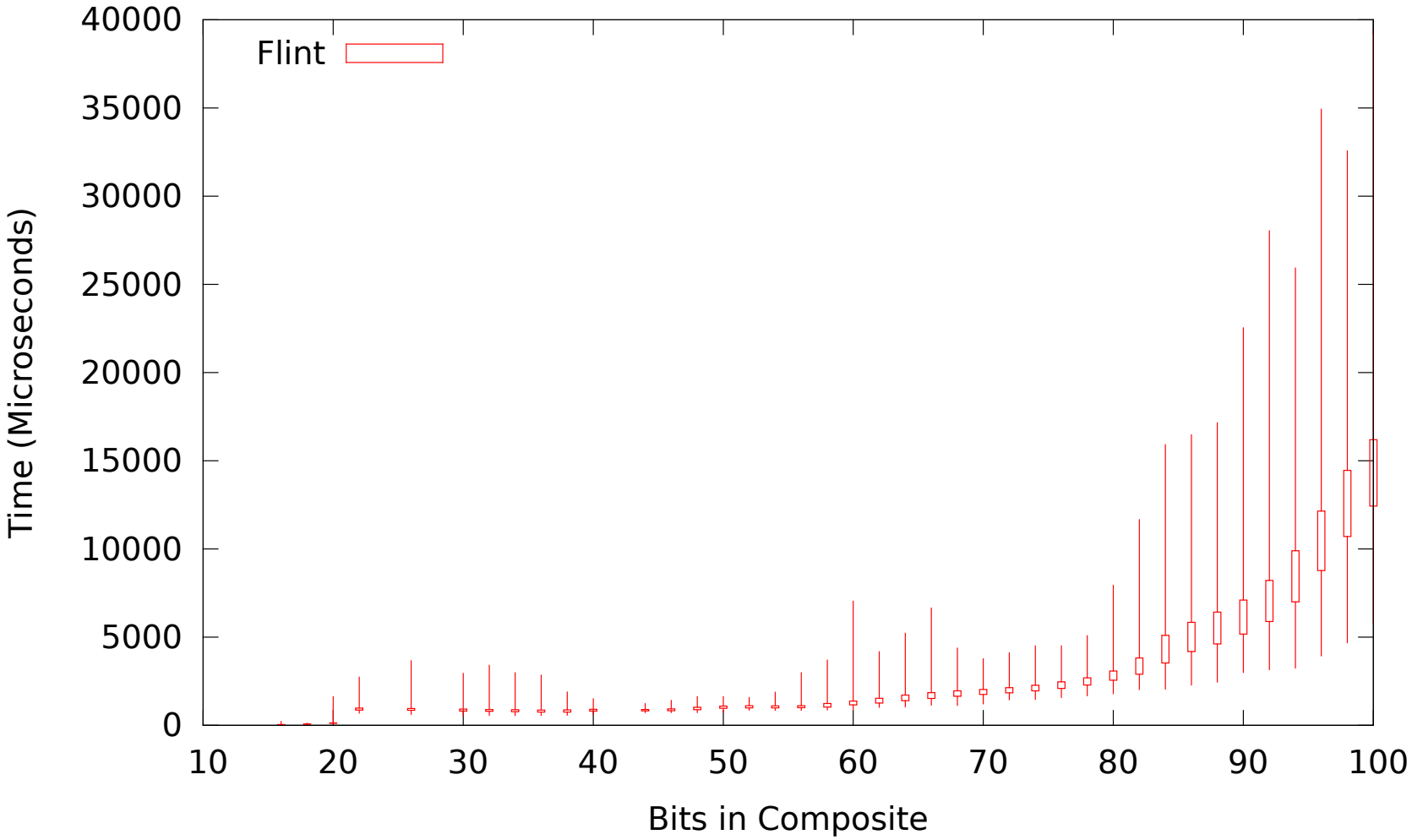


SPAR – Expanding the 2-Sylow Group

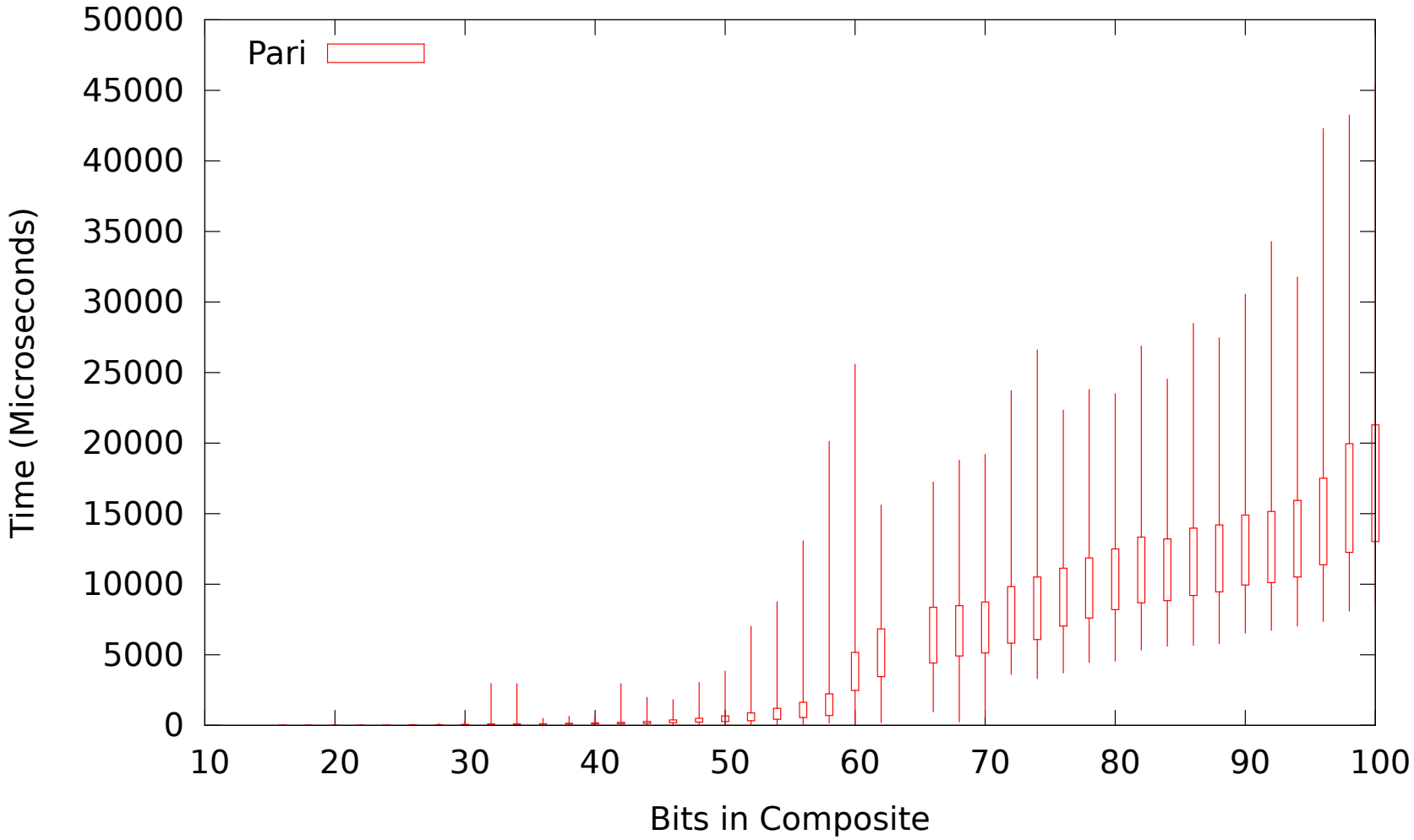
Bits	With 2-Sylow Group	Without 2-Sylow Group
16	48.05208	48.02160
20	84.20318	84.09526
24	197.02479	196.84587
28	463.70674	463.60538
32	937.57875	935.81785
36	1709.55629	1706.32960
40	3255.31447	3248.67998

Table: Average time in microseconds to factor using SPAR when the number of ideals per class group is bound by the size of the integer to factor.

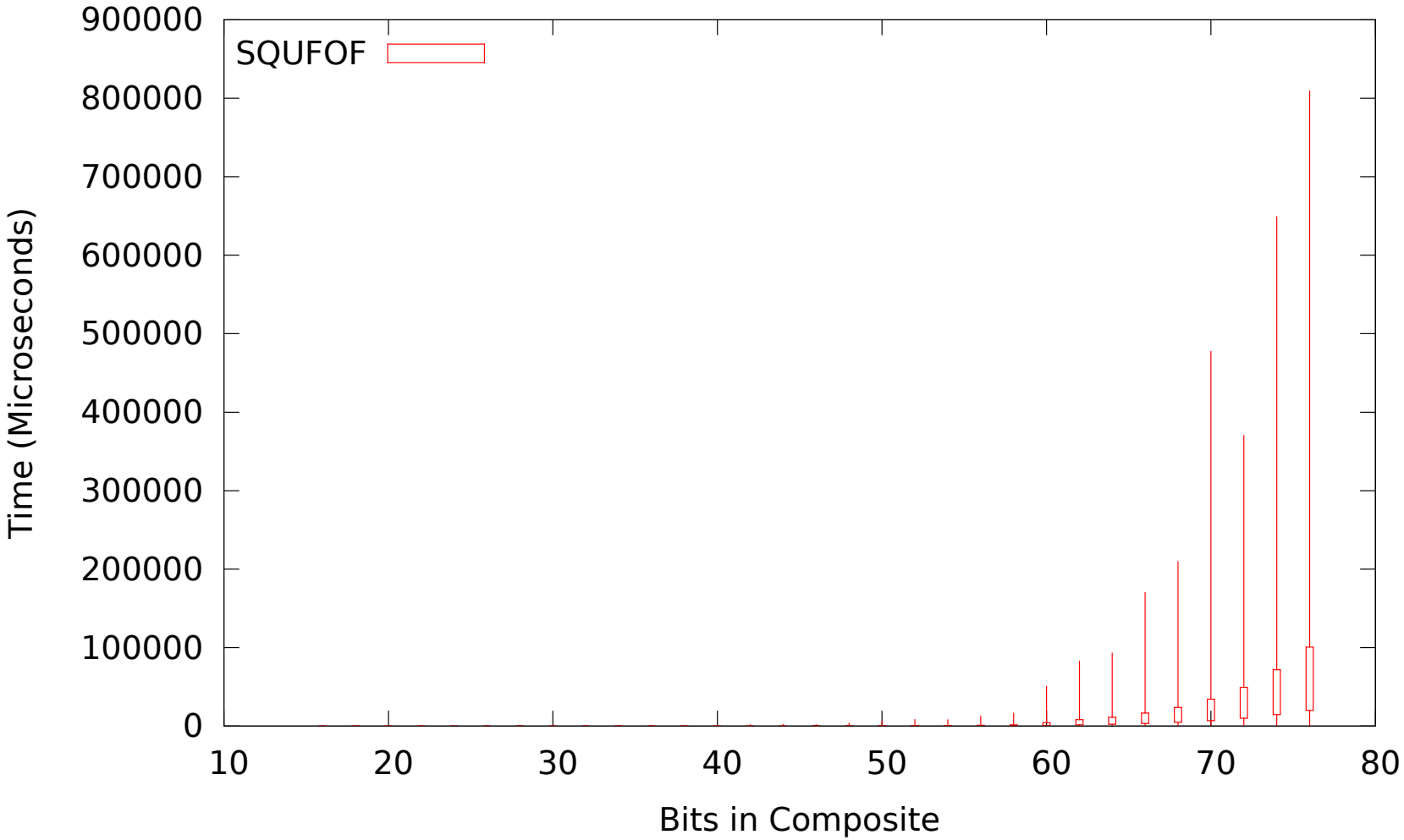
Flint


Flint
Time (Microseconds)
Bits in Composite
0
5000
10000
15000
20000
25000
30000
35000
40000
10
20
30
40
50
60
70
80
90
100

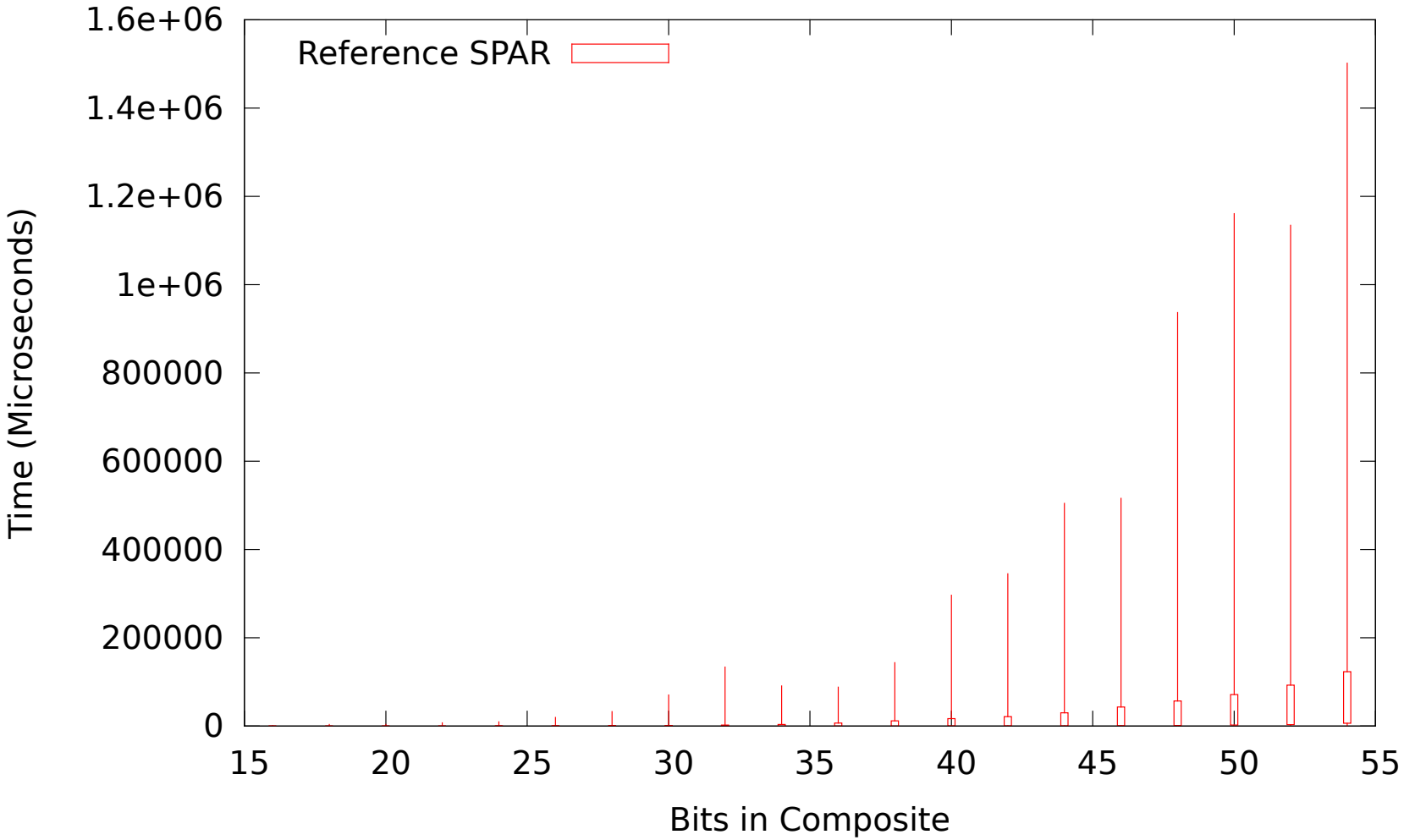
Pari


Pari
Time (Microseconds)
Bits in Composite
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
10
20
30
40
50
60
70
80
90
100

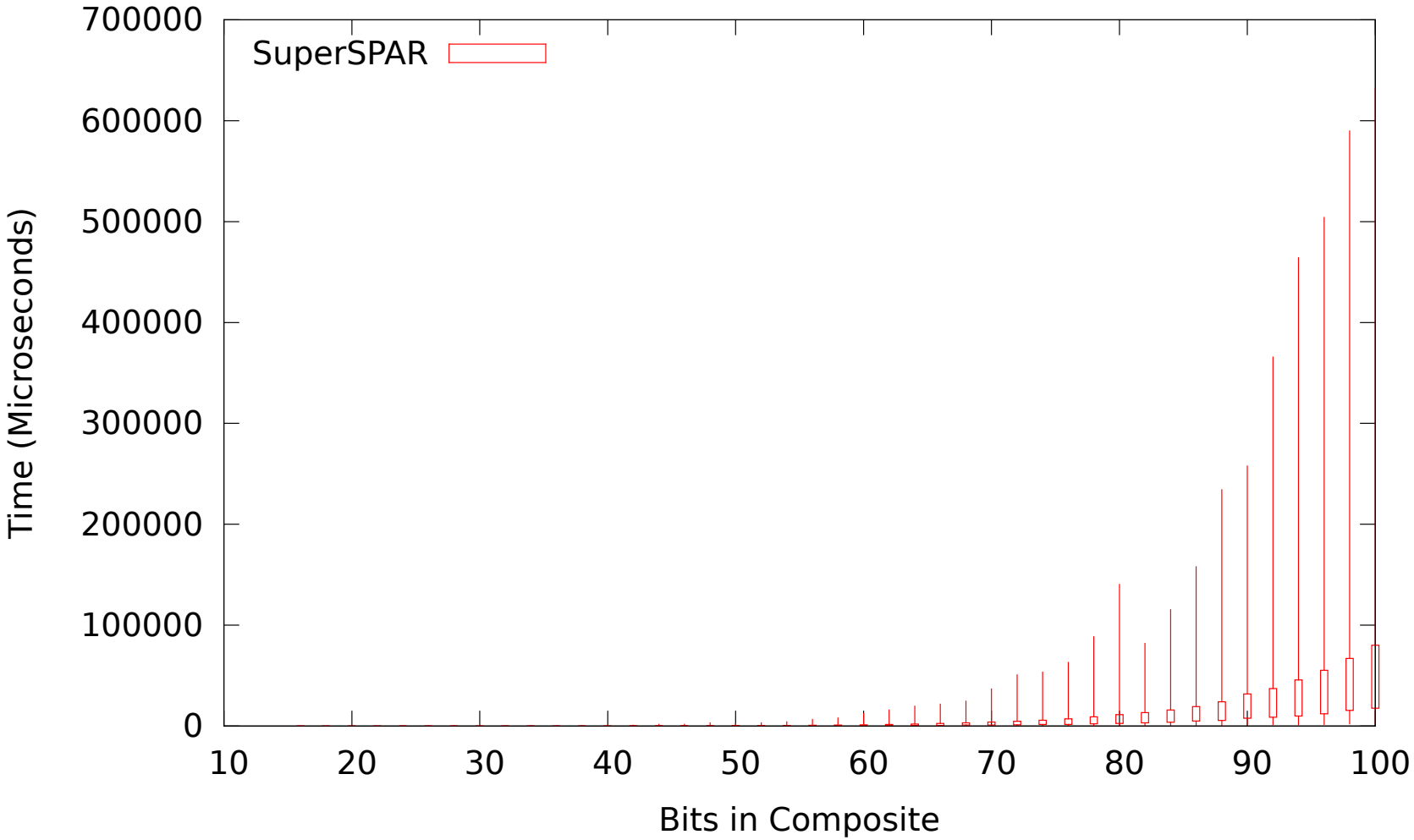
SQUFOF (Optimized from Pari)



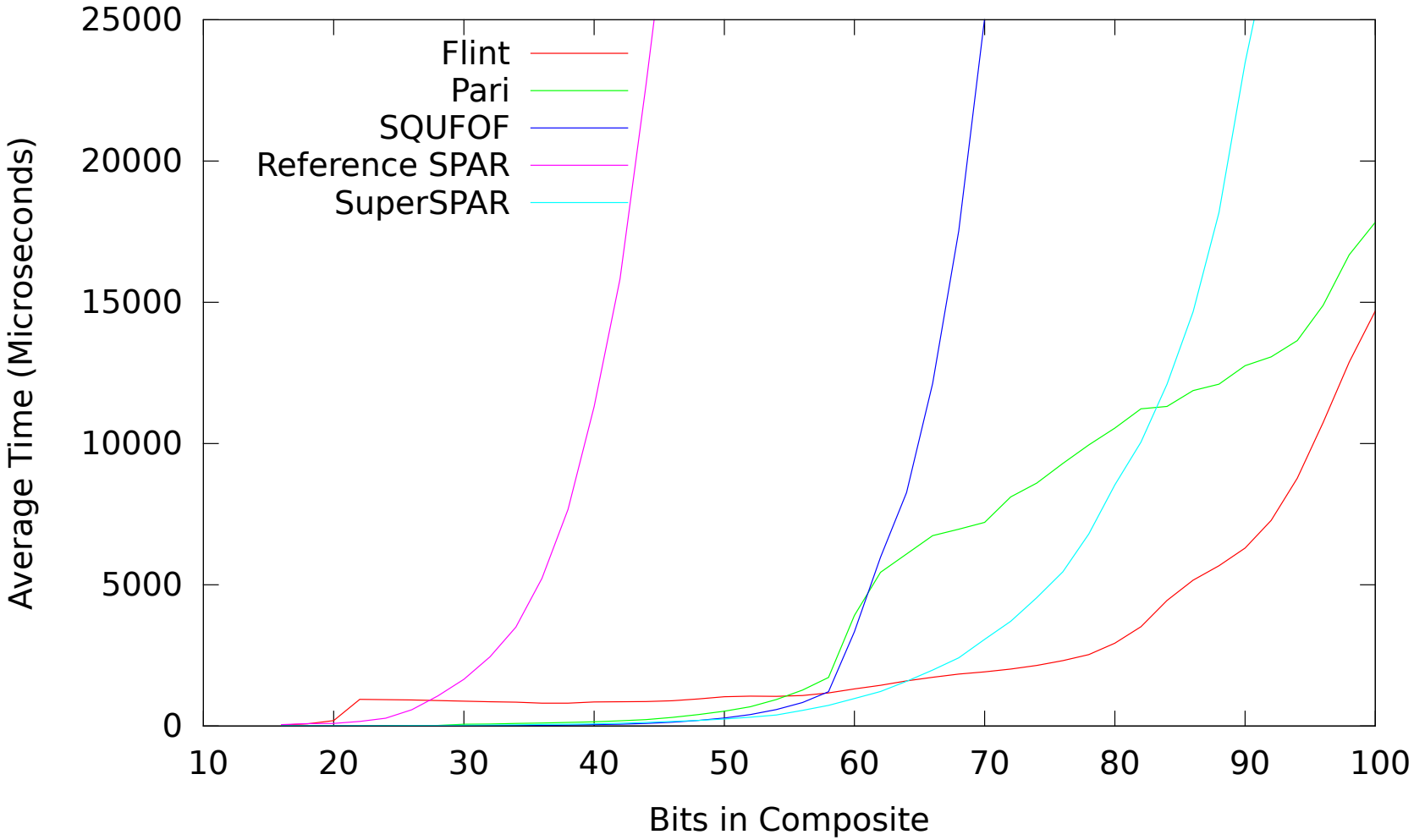
Vanilla SPAR



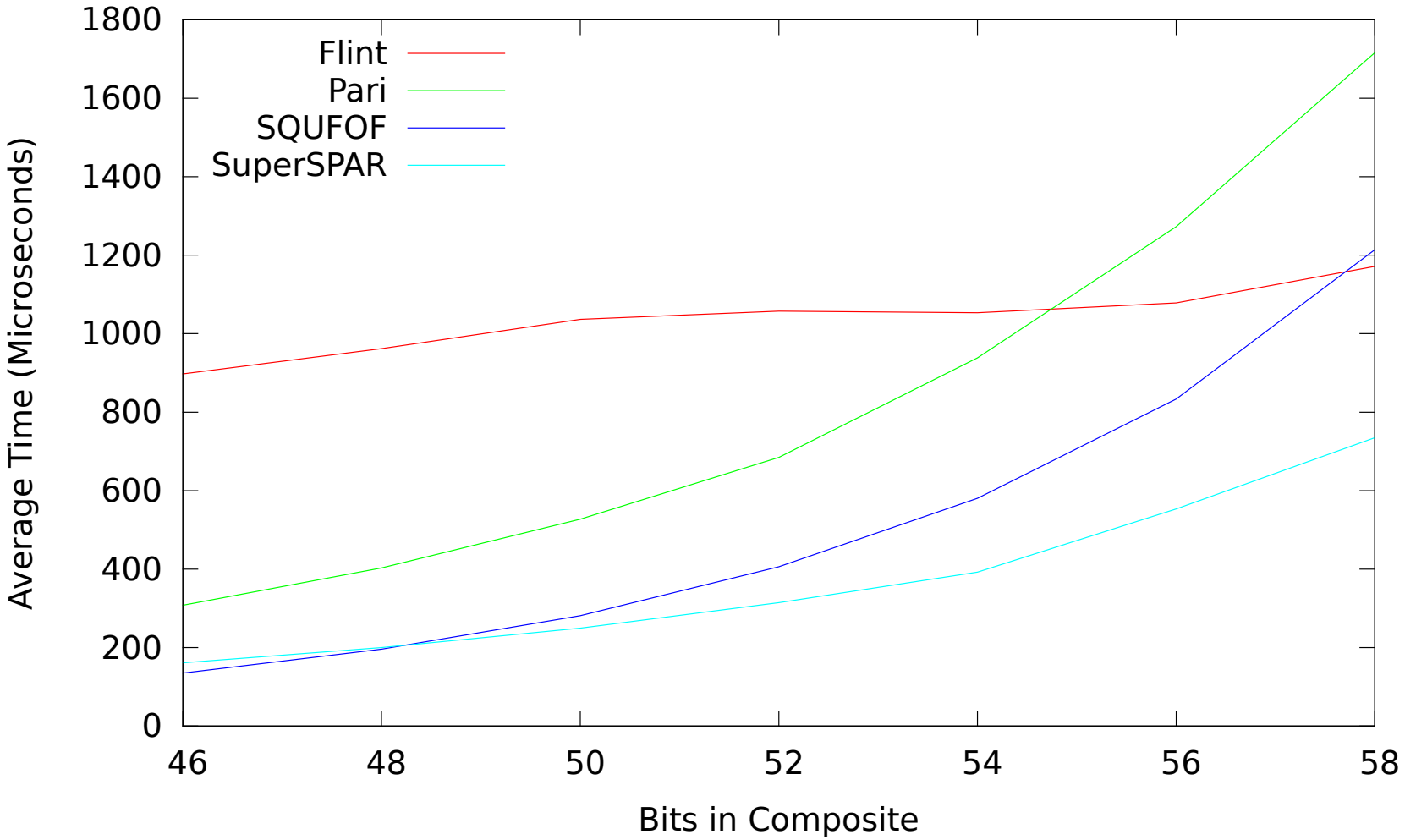
SuperSPAR


SuperSPAR
Time (Microseconds)
700000
600000
500000
400000
300000
200000
100000
0
10
20
30
40
50
60
70
80
90
100
Bits in Composite

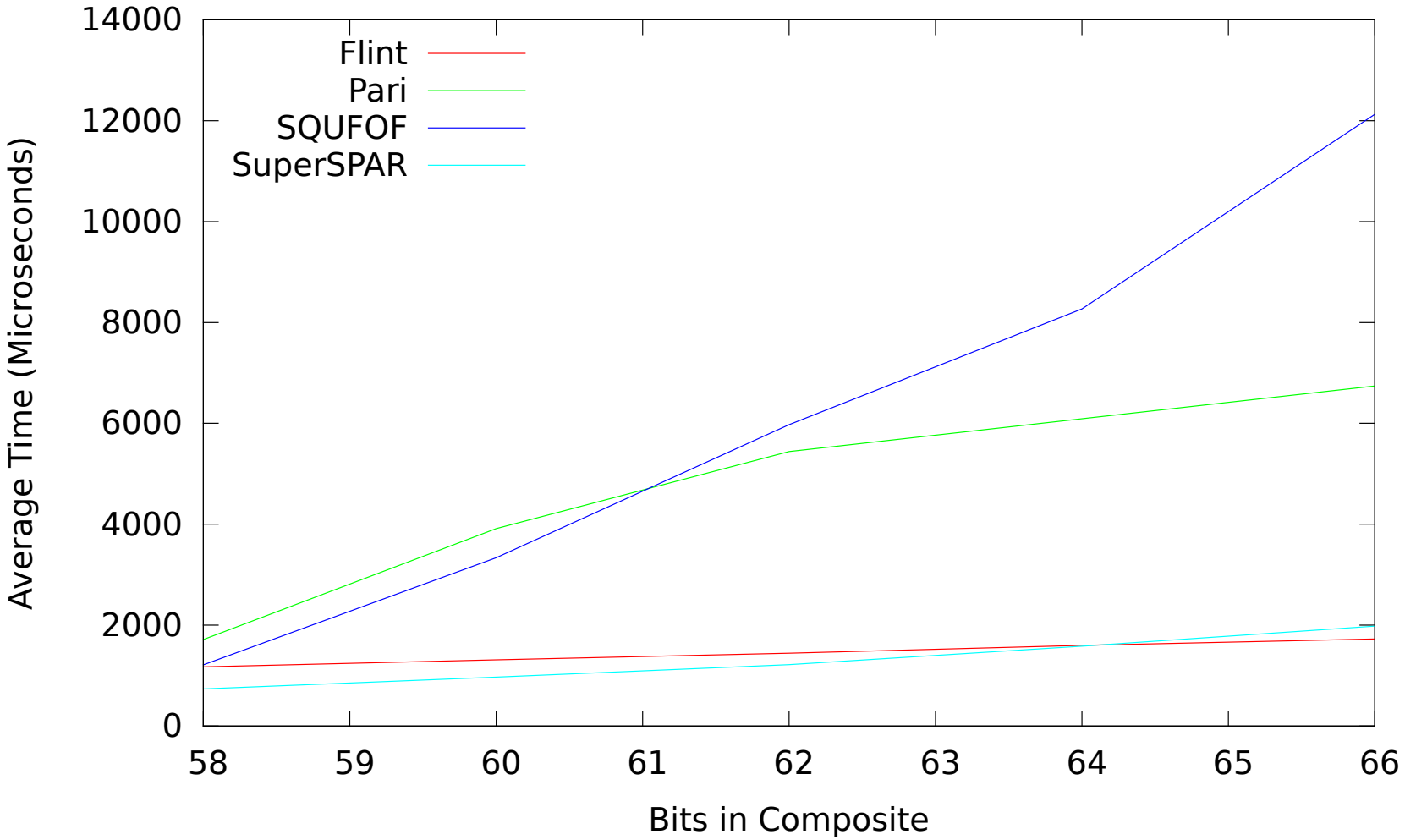
Average Integer Factoring Times



Average Integer Factoring Times (Zoomed Left)



Average Integer Factoring Times (Zoomed Right)



Simple Continued Fraction Expansion of K/L

Recurrences:

$$\begin{aligned} R_i &= R_{i-2} - q_i R_{i-1} \\ C_i &= C_{i-2} - q_i C_{i-1} \\ A_i &= A_{i-2} + q_i A_{i-1} \\ B_i &= B_{i-2} + q_i B_{i-1} \end{aligned}$$

Invariants:

$$\begin{aligned} C_i &= (-1)^{i+1} B_i \\ L &= R_i B_{i-1} + B_i R_{i-1} \\ (-1)^{i-1} &= A_i B_{i-1} - B_i A_{i-1} \\ (-1)^{i+1} R_i &= L A_i - K B_i \end{aligned}$$