MONTE-CARLO ARITHMETIC

1. $n \log n$

1.1. **mul(b,a) and sqr(a).** Schoolbook, 2 and 3-way Toom-Cook up to 6000 bits. Then, the double precision complex FFT takes over until its lack of cache optimizations brings it behind the 50-bit finite field FFT. The complex FFT coefficients are now signed, which makes them extra secure$^{(\mathrm{TM})}$: The dot product $x_1y_1+\cdots+x_my_m$ has the *vast* majority of its mass centered around zero if the x_i and y_i are chosen uniformly from $[-2^{t-1}, 2^{t-1}]$ instead of from $[0, 2^t)$. The complex FFT has not failed yet, and even it if did it would have to get past several checks modulo p which further reduce the error rate by a factor of 2^{-64}. Thus, the complex FFT is less likely to fail in practice than the operating system is in loading the program.

1.2. **inv(a).** An n-bit inverse x_1 is extended to a $2n$-bit inverse x_2 via $x_2 = x_1 + x_1(1 - ax_1)$. Quadratic range: The middle n bits of ax_1 require n^2 bit multiplications, and the multiplication $x_1(1 - ax_1)$ requires $\frac{1}{2}n^2$ bit multiplications, for a total of $\frac{3}{2}n^2$. Newton iteration to a final precision of n bits therefore produces

$$\tfrac{3}{2}\left(\tfrac{n}{2}\right)^2 + \tfrac{3}{2}\left(\tfrac{n}{4}\right)^2 + \cdots = \tfrac{1}{2}n^2,$$

which is the same bit complexity as a `mulhi`, and this is a good algorithm at every precision, unless a is very short.

1.3. **div(b,a).** Unless a is much shorter than the target precision, Karp-Markstein is used. Quadratic range: Producing an $n/2$-bit inverse $x = a^{-1} + O(2^{-n/2})$ and then evaluating $y = bx + O(2^{-n/2})$ and $b/a = y + x(b - ay) + O(2^{-n})$ costs

$$\tfrac{1}{2}\left(\tfrac{n}{2}\right)^2 + 2\left(\tfrac{n}{2}\right)^2 = \tfrac{5}{8}n^2.$$

The (quadratic) schoolbook approach would have the cost $\frac{1}{2}n^2$, which might seem better. However, these $\frac{1}{2}n^2$ operations are `mpn_submul_1`, which runs at 2.0 cycles per limb. This loses slightly to $\frac{5}{8}n^2$ operations in `mpn_addmul_1`, which runs at 1.55 cycles per limb. Thus, this is a good algorithm at every precision.

1.4. **rsqrt(a).** An n-bit inverse x_1 is extended to a $2n$-bit inverse x_2 via $x_2 = x_1 + \frac{1}{2}x_1(1 - ax_1^2)$.

1.5. **divsqrt(b,a).** Start with a $n/2$-bit inverse $x = 1/\sqrt{a} + O(2^{-n/2})$, evaluate $y = bx + O(2^{-n/2})$, and finally evaluate $b/\sqrt{a} = bx + \frac{1}{2}y(1 - ax^2) + O(2^{-n})$. At the highest precision this saves only about 8% over the usual `mul(b,rsqrt(a))`, but there are more gains at lower precision in fusing this common combination.

1.6. **sqrtv1(a).** Karp-Markstein: Start with a $n/2$-bit inverse $x = 1/\sqrt{a} + O(2^{-n/2})$, evaluate $y = ax + O(2^{-n/2})$, and finally evaluate $\sqrt{a} = y + \frac{1}{2}x(a - y^2) + O(2^{-n})$.

1.7. **sqrtv2(a).** The usual Newton iteration: $x_2 = \frac{1}{2}(x_1 + \frac{a}{x_1})$. This formula cannot be implemented naively. Suppose s is fixed at 0 or 1 and we have an n-bit square root via the following information:

> Integers a_1, x_1, and z_1 with $2^{n-1} \le a_1, x_1 < 2^n$ and $2^{-s}\frac{a_1}{2^n} - \left(\frac{x_1}{2^n}\right)^2 = \frac{z_1}{2^{2n}}$, where either $-1 \le \frac{z_1}{x_1} \le 1$ (nearest square root, not used), or $0 \le \frac{z_1}{x_1} \le 2$ (floor square root, used here).

This n-bit square root can be extended to an $n+m$-bit square root (for $m \le n$, $0 \le a_2, x_2 < 2^m$) via

$$2^{-s}\left(\frac{a_1}{2^n} + \frac{a_2}{2^{n+m}}\right) - \left(\frac{x_1}{2^n} + \frac{x_2}{2^{n+m}}\right)^2 = \frac{2^{2m}z_1 + 2^{n+m-s}a_2 - 2^m 2x_1x_2 - x_2^2}{2^{2(m+n)}} = \frac{2^{m+1}r - x_2^2}{2^{2(m+n)}} =: \frac{z_2}{2^{2(m+n)}},$$

where $r := 2^{m-1}z_1 + 2^{n-1-s}a_2 - x_1x_2$. Calculating r and x_2 $(= q)$ is a division by x_1. We have

$$x_2 = (2x_1)^{-1}\left(2^m z_1 + 2^{n-s}a_2\right) + \epsilon \implies \left|\frac{z_2}{2^m x_1 + x_2} + 2\epsilon\right| \le 2^{3-n}\epsilon + 2^{3+m-n}.$$

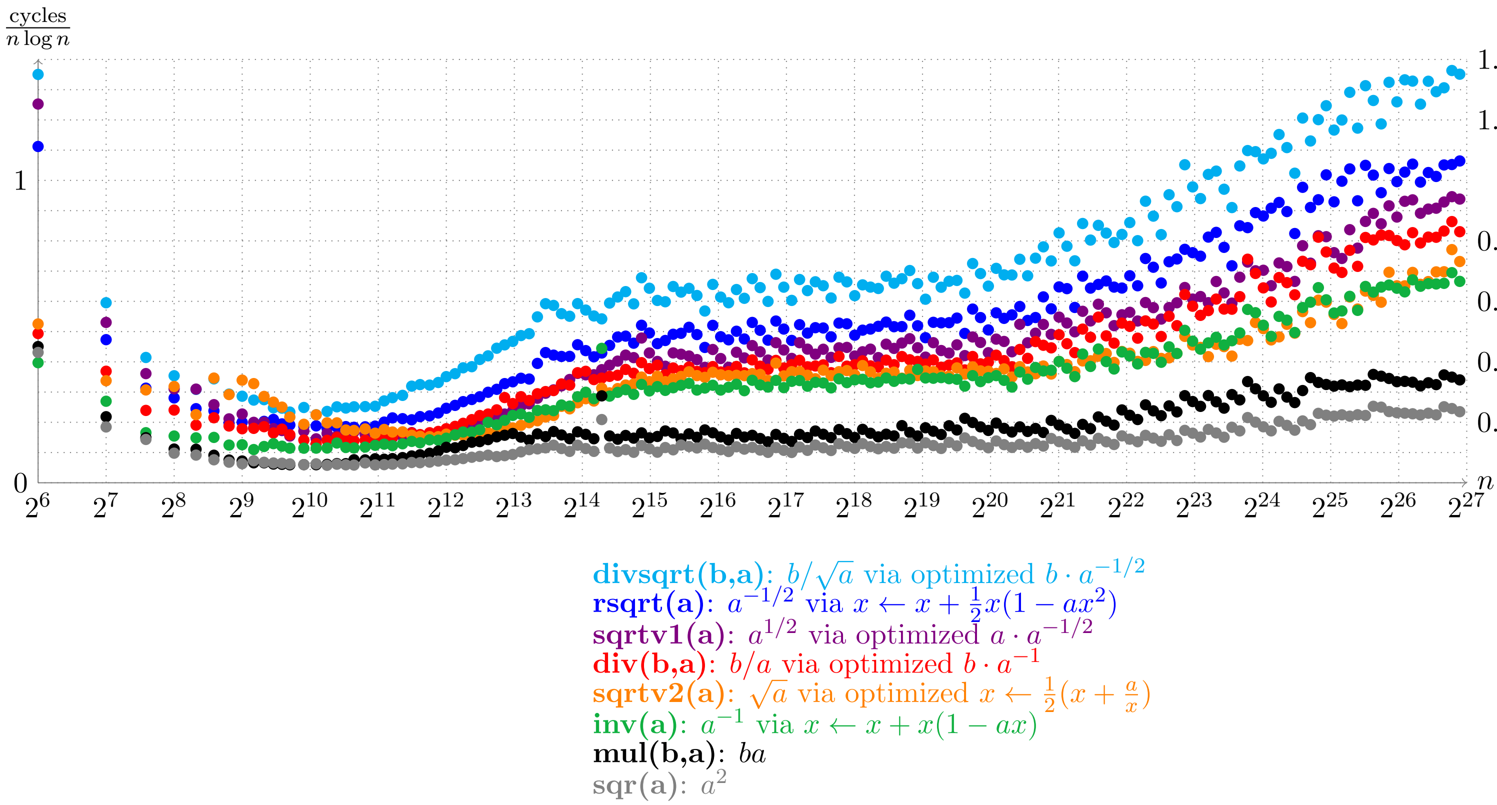
Hence, if $(2x_1)^{-1}$ to known precisely enough to ensure $|\epsilon| < 0.501$, at most one fixed up $'$ is required to obtain

$$0 \le \frac{z_2'}{2^m x_1' + x_2'} \le 2$$

in the case $n - m \ge 64$. For smaller values of $n - m$, at most 5 fixups are required. Since r is small, the calculation of r from x_2 can use multiplication modulo 2^{n+m+64} or $2^{n+m+64} - 1$, and, of course, the inverse $(2x)^{-1}$ need not be updated on the last iteration.

1.8. **timings.** Figures 1 and 2 show the timings of various operations performed with full n bit precision inputs/outputs. This means that outputs are computed with a guaranteed error of less than 0.5001 ulps. The timing are divided by the expected asymptotic complexity of the operation, and they are expressed in CPU cycles, where, on this machine, 4 cycles is the latency of a double precision floating point addition or multiplication. For example, In Figure 1, the orange dot above precision 2^8 is located at a height of approximately 0.3, indicating that `sqrtv2` operating at a precision of 256 bits runs in approximately $0.3 \cdot 256 \cdot \log 256 = 400$ cycles, or, approximately 100 times as slowly as a double precision multiplication runs.

FIGURE 1. $O(n \log n)$ operations



2. $n \log^2 n$

2.1. **log(a).** Up to 2^9 bits the region of interest is $1/\sqrt{2} \le a \le \sqrt{2}$, and a 5-stage reduction is used:

$$\log a = -\log(1 - \tfrac{s_1}{2^8}) - \log(1 - \tfrac{s_2}{2^{14}}) - \log(1 - \tfrac{s_3}{2^{20}}) - \log(1 - \tfrac{s_4}{2^{26}}) - \log(1 - \tfrac{s_5}{2^{32}}) + \log(1 + x),$$

where the s_i are any integers chooses successively so that $|x| < 2^{-32}$. To be fast in this range, it is important that no division is required for x. The final log is evaluated, for example, as

$$\log(1 + x) = x - \left(\tfrac{1}{2}x^2 - \tfrac{1}{3}x^3 + x^2(\tfrac{1}{4}x^2 - \tfrac{1}{5}x^3 + x^2(\tfrac{1}{6}x^2 - \tfrac{1}{7}x^3))\right) + O(x^8).$$

Up to 2^{16} bits, the interesting region is the same and another 5-stage reduction is used:

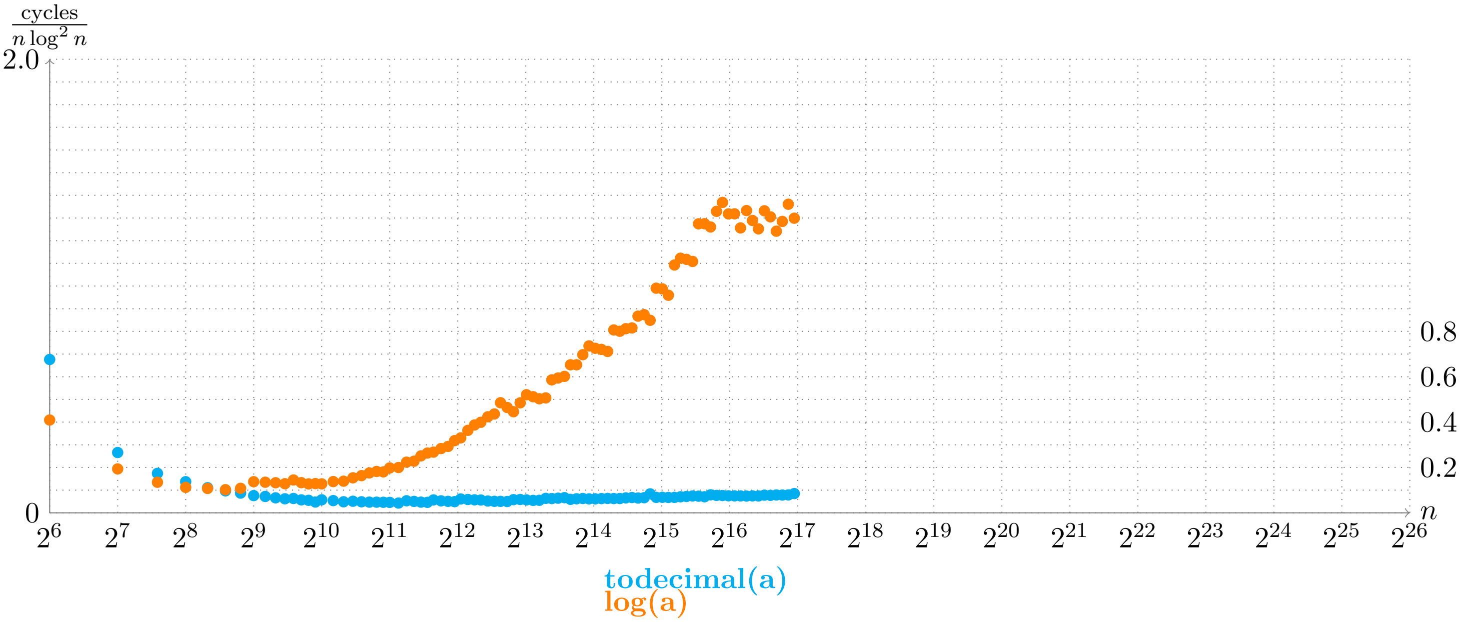
$$\log a = \pm\log(1 - \tfrac{r_1}{2^8}) \pm \log(1 - \tfrac{r_2}{2^{14}}) \pm \log(1 - \tfrac{r_3}{2^{20}}) \pm \log(1 - \tfrac{r_4}{2^{26}}) \pm \log(1 - \tfrac{r_5}{2^{31}}) + 2\tanh^{-1}(x),$$

where the r_i are non-negative integers and where $|x| < 2^{-32}$. Unlike the previous reduction, a division is required to compute x.

AGM iteration past 2^{16} bits.

2.2. **todecimal(a).**

Figure 2. $O(n \log^2 n)$ operations



2.3. timings.