1. NOTATION

$\mathbb{i}$	$\sqrt{-1}$
$\mathbb{e}$	$2.71\ldots$
π	$3.14\ldots$
$\mathrm{EvalPoly}(a, \vec{x})$	$\sum_{0\le i<n} x_i a^i$, $\vec{x} = (x_0, \ldots, x_{n-1})$
ω_b^a	$\mathbb{e}^{2a\pi\mathbb{i}/b}$ a b^{th} root of unity
$\bar{i}^k$	length k bit-reversal of i. Only defined for $0 \le i < 2^k$

2. FFT/IFFT

Unless the output of the fft is specifically needed to be in the usual order

$$\left\{\mathrm{EvalPoly}(\omega_{2^k}^i, \vec{x})\right\}_{0\le i<2^k}$$

there is no reason not to give the output in bit-reversed order

$$\left\{\mathrm{EvalPoly}(\omega_{2^k}^{\bar{i}^k}, \vec{x})\right\}_{0\le i<2^k}.$$

The reason being here is that the bit-reversed output is much simpler and faster because it groups similar outputs close together. For example, $\mathrm{EvalPoly}(1, \vec{x})$ and $\mathrm{EvalPoly}(-1, \vec{x})$ are very similar computationally and are right next to each other in the bit-reversed output, but are very far apart in the usual order. Therefore, we will restrict exclusively to bit-reversed outputs.

The usual sequence for calculating a length 16 fft follows the columns below and ends with the fft in the last column.

x_i	y_i	z_i	w_i	$\mathrm{fft}(\vec{x})$
x_0	$x_0 + x_8$	$y_0 + y_4$	$z_0 + z_2$	$w_0 + w_1$
x_1	$x_1 + x_9$	$y_1 + y_5$	$z_1 + z_3$	$(w_0 - w_1)$
x_2	$x_2 + x_{10}$	$y_2 + y_6$	$(z_0 - z_2)\omega_4^0$	$w_2 + w_3$
x_3	$x_3 + x_{11}$	$y_3 + y_7$	$(z_2 - z_3)\omega_4^1$	$(w_2 - w_3)$
x_4	$x_4 + x_{12}$	$(y_0 - y_4)\omega_8^0$	$z_4 + z_6$	$w_4 + w_5$
x_5	$x_5 + x_{13}$	$(y_1 - y_5)\omega_8^1$	$z_5 + z_7$	$(w_4 - w_5)$
x_6	$x_6 + x_{14}$	$(y_2 - y_6)\omega_8^2$	$(z_4 - z_6)\omega_4^0$	$w_6 + w_7$
x_7	$x_7 + x_{15}$	$(y_3 - y_7)\omega_8^3$	$(z_5 - z_7)\omega_4^1$	$(w_6 - w_7)$
x_8	$(x_0 - x_8)\omega_{16}^0$	$y_8 + y_{12}$	$z_8 + z_{10}$	$w_8 + w_9$
x_9	$(x_1 - x_9)\omega_{16}^1$	$y_9 + y_{13}$	$z_9 + z_{11}$	$(w_8 - w_9)$
x_{10}	$(x_2 - x_{10})\omega_{16}^2$	$y_{10} + y_{14}$	$(z_8 - z_{10})\omega_4^0$	$w_{10} + w_{11}$
x_{11}	$(x_3 - x_{11})\omega_{16}^3$	$y_{11} + y_{15}$	$(z_9 - z_{11})\omega_4^1$	$(w_{10} - w_{11})$
x_{12}	$(x_4 - x_{12})\omega_{16}^4$	$(y_8 - y_{12})\omega_8^0$	$z_{12} + z_{14}$	$w_{12} + w_{13}$
x_{13}	$(x_5 - x_{13})\omega_{16}^5$	$(y_9 - y_{13})\omega_8^1$	$z_{13} + z_{15}$	$(w_{12} - w_{13})$
x_{14}	$(x_6 - x_{14})\omega_{16}^6$	$(y_{10} - y_{14})\omega_8^2$	$(z_{12} - z_{14})\omega_4^0$	$w_{14} + w_{15}$
x_{15}	$(x_7 - x_{15})\omega_{16}^7$	$(y_{11} - y_{15})\omega_8^3$	$(z_{13} - z_{15})\omega_4^1$	$(w_{14} - w_{15})$

One problem with this approach is that each column accesses many different *twiddle factors* ω_b^a. In the case of a Schönhage–Strassen fft where the base ring is $\mathbb{Z}/(2^m + 1)\mathbb{Z}$, this doesn't matter because each twiddle factor is a power of two and implemented via bit shifts. In other cases, these twiddle factors have to be either computed on the fly or precomputed and then retrieved from storage. In the case of precomputation, we have to have in memory the table

$$1, \omega_2^1, 1, \omega_4^1, 1, \omega_8^1, \omega_8^2, \omega_8^3, 1, \omega_{16}^1, \omega_{16}^2, \omega_{16}^3, \omega_{16}^4, \omega_{16}^5, \omega_{16}^6, \omega_{16}^7, 1, \ldots$$

so that the columns can access this table sequentially. However, such a table is nice because once it is extended to accomodate an fft of a certain length, it can be reused for all ffts of smaller length.

By rearranging the twiddle factors as ($\omega = \omega_{16}$)

x_i	y_i	z_i	w_i	$\mathrm{fft}(\vec{x})$
x_0	$x_0 + \omega^0 x_8$	$y_0 + \omega^0 y_4$	$z_0 + \omega^0 z_2$	$w_0 + \omega^0 w_1$
x_1	$x_1 + \omega^0 x_9$	$y_1 + \omega^0 y_5$	$z_1 + \omega^0 z_3$	$w_0 + \omega^8 w_1$
x_2	$x_2 + \omega^0 x_{10}$	$y_2 + \omega^0 y_6$	$z_0 + \omega^8 z_2$	$w_2 + \omega^4 w_3$
x_3	$x_3 + \omega^0 x_{11}$	$y_3 + \omega^0 y_7$	$z_1 + \omega^8 z_3$	$w_2 + \omega^{12} w_3$
x_4	$x_4 + \omega^0 x_{12}$	$y_0 + \omega^8 y_4$	$z_4 + \omega^4 z_6$	$w_4 + \omega^2 w_5$
x_5	$x_5 + \omega^0 x_{13}$	$y_1 + \omega^8 y_5$	$z_5 + \omega^4 z_7$	$w_4 + \omega^{10} w_5$
x_6	$x_6 + \omega^0 x_{14}$	$y_2 + \omega^8 y_6$	$z_4 + \omega^{12} z_6$	$w_6 + \omega^6 w_7$
x_7	$x_7 + \omega^0 x_{15}$	$y_3 + \omega^8 y_7$	$z_5 + \omega^{12} z_7$	$w_6 + \omega^{14} w_7)$
x_8	$x_0 + \omega^8 x_8$	$y_8 + \omega^4 y_{12}$	$z_8 + \omega^2 z_{10}$	$w_8 + \omega^1 w_9$
x_9	$x_1 + \omega^8 x_9$	$y_9 + \omega^4 y_{13}$	$z_9 + \omega^2 z_{11}$	$w_8 + \omega^9 w_9$
x_{10}	$x_2 + \omega^8 x_{10}$	$y_{10} + \omega^4 y_{14}$	$z_8 + \omega^{10} z_{10}$	$w_{10} + \omega^5 w_{11}$
x_{11}	$x_3 + \omega^8 x_{11}$	$y_{11} + \omega^4 y_{15}$	$z_9 + \omega^{10} z_{11}$	$w_{10} + \omega^{13} w_{11}$
x_{12}	$x_4 + \omega^8 x_{12}$	$y_8 + \omega^{12} y_{12}$	$z_{12} + \omega^6 z_{14}$	$w_{12} + \omega^3 w_{13}$
x_{13}	$x_5 + \omega^8 x_{13}$	$y_9 + \omega^{12} y_{13}$	$z_{13} + \omega^6 z_{15}$	$w_{12} + \omega^{11} w_{13}$
x_{14}	$x_6 + \omega^8 x_{14}$	$y_{10} + \omega^{12} y_{14}$	$z_{12} + \omega^{14} z_{14}$	$w_{14} + \omega^7 w_{15}$
x_{15}	$x_7 + \omega^8 x_{15}$	$y_{11} + \omega^{12} y_{15}$	$z_{13} + \omega^{14} z_{15}$	$w_{14} + \omega^{15} w_{15}$

There are still the same number of twiddle factor multiplications, but each twiddle factor itself can be reused in each column. Also, as with the previous method, there is a universal (bit-reversed) table

$$1, \quad \omega_2^1, \quad \omega_4^1, \omega_4^3, \quad \omega_8^1, \omega_8^5, \omega_8^3, \omega_8^7, \quad \omega_{16}^1, \omega_{16}^9, \omega_{16}^5, \omega_{16}^{13}, \omega_{16}^3, \omega_{16}^{11}, \omega_{16}^7, \omega_{16}^{15}, \quad \ldots$$

that can be reused. The difference here is that the portion that is used for a specific fft is now half the size (making note of $\omega_4^3 = -\omega_4^1$, $\omega_8^5 = -\omega_8^1$, $\omega_8^7 = -\omega_8^3$, etc.).

Since the output of the fft is given in bit-reversed order, the inverse operation cannot simply replace ω by ω^{-1} and use the same calculation sequence. Thus the ifft has to invert the left-to-right operation by working from the right to the left and inverting each basic operation. This gives slightly different data access patterns but involves essentially the same calculations. For example, the operation $w_8 = z_8 + \omega^2 z_{10}, w_{10} = z_8 - \omega^2 z_{10}$ is inverted by $2z_8 = w_8 + w_{10}, 2z_{10} = \omega^{-2}(w_8 - w_{10})$ and the negative power $\omega^{-\bar{i}k}$ can be looked up in the table by flipping some of the lower bits of i.

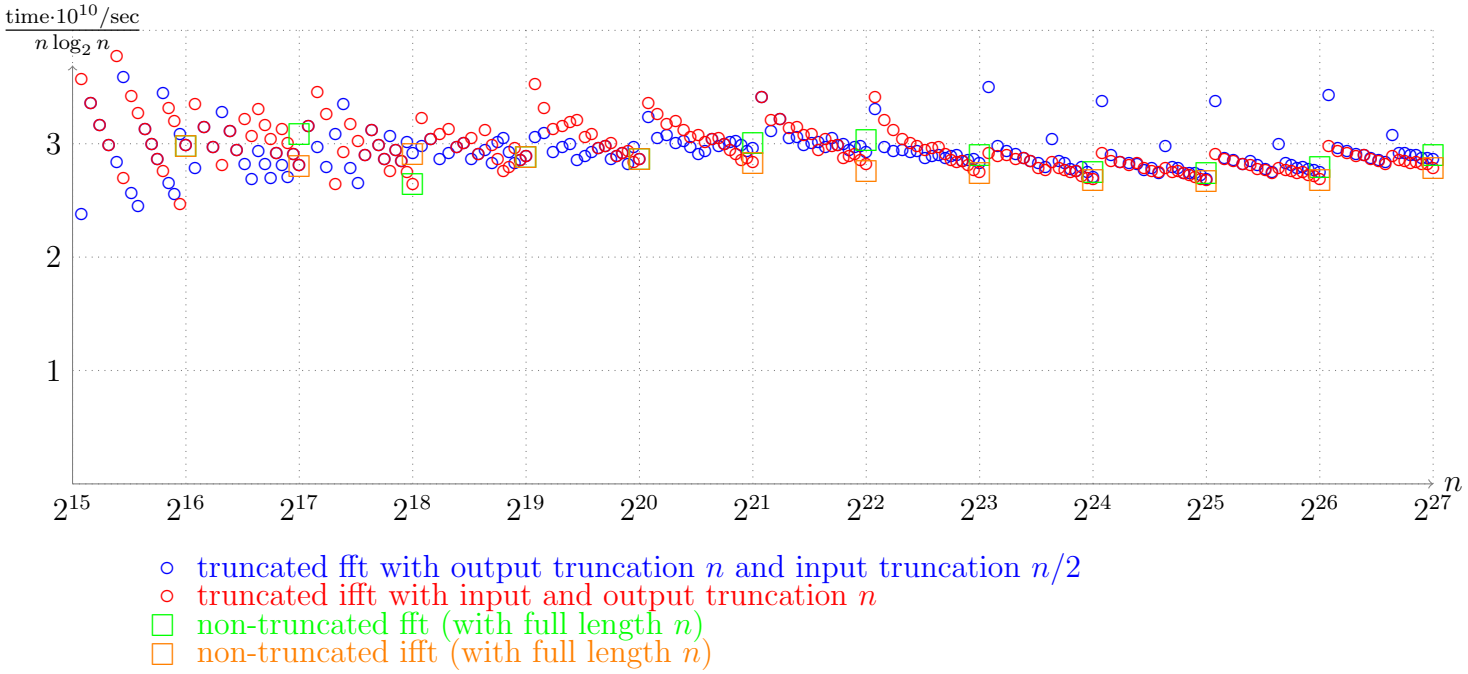
3. TRUNCATION

Since zero padding the input and output data to the next power of two obviously leads to ugly performance jumps at powers of two, a *truncated fft* is necessary, which assumes certain portions of the input and output are zero. If the desired output length is denoted by n we expect

$$\text{runtime} \approx n \log n,$$

so the more constant the ratio is, the better the truncation is working. Figure 1 show a graph of such a ratio when the fft is performed over $\mathbb{F}_p$ for a 50 bit prime p and the input is further truncated to length $n/2$, which corresponds to assuming the top half of the n inputs are zero. As expected, the truncated fft/ifft performs worst right after a power of two, but slightly unexpected is how bad the fft is there after $n > 2^{23}$ and how well the ifft is doing there. There are performance bumps at $3 \cdot 2^n$, and, rather surprisingly, the non-truncated ifft is eventually beating the non-truncated fft by about 7%. This is surprising for

FIGURE 1. Truncation effectiveness with 50 bit coefficients



the non-truncated version because they are performing the exact same calculations, just in a different order. The result of optimizing only the length 16 basecases for the non-truncated fft and ifft is shown in Figure 2, where, unfortunatey, the fft is lagging even further behind the ifft on almost all sizes. The small sizes where also given more stable timings in this graph. Finally, the corresponding graph for FLINT's Schönhage–Strassen fft is shown in Figure 3, where very large coefficients had to be used to ensure that the graph can contiune to $n = 2^{18}$ while keeping the coefficient size constant.

FIGURE 2. Truncation effectiveness with 50 bit coefficients and optimized length 16 basecase

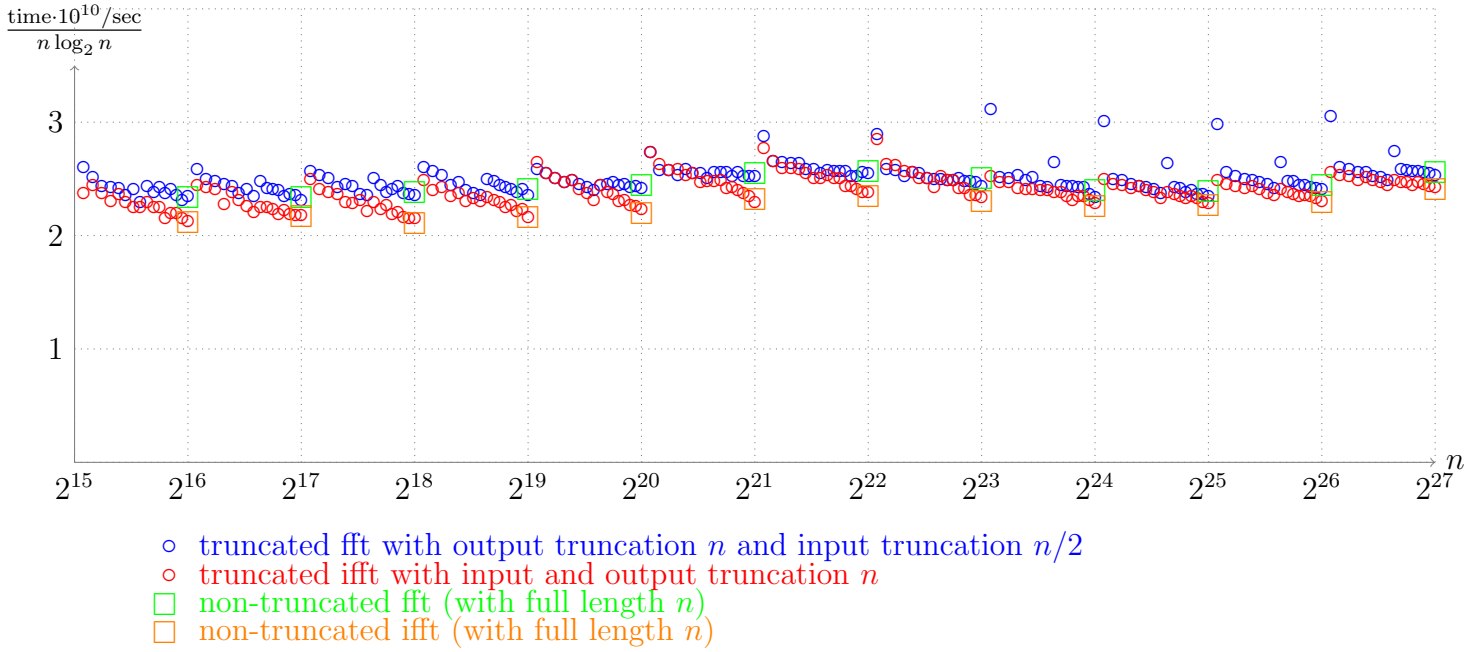


FIGURE 3. Truncation effectiveness of {i}`fft_mfa_truncate_sqrt2` with 2^{16} bit coefficients