FAST FOURIER TRANSFORM

The time taken to evaluate a DFT on a digital computer depends principally on the number of multiplications involved, since these are the slowest operations. With the DFT, this number is directly related to N^2 (matrix multiplication of a vector), where N is the length of the transform. For most problems, N is chosen to be at least 256 in order to get a reasonable approximation for the spectrum of the sequence under consideration – hence computational speed becomes a major consideration.

Highly efficient computer algorithms for estimating Discrete Fourier Transforms have been developed since the mid-60's. These are known as Fast Fourier Transform (FFT) algorithms and they rely on the fact that the standard DFT involves a lot of redundant calculations:

$$\text{Re-writing} \quad F[n] = \sum_{k=0}^{N-1} f[k] e^{-j\frac{2\pi}{N}nk} \text{ as} F[n] = \sum_{k=0}^{N-1} f[k] W_N^{nk}$$

it is easy to realise that the same values of W_N^{nk} are calculated many times as the computation proceeds. Firstly, the integer product nk repeats for different combinations of k and n; secondly, W_N^{nk} is a periodic function with only N distinct values.

For example, consider $N = 8$ (the FFT is simplest by far if N is an integral power of 2)

$$W_8^1 = e^{-j\frac{2\pi}{8}} = e^{-j45^o} = \frac{1-j}{\sqrt{2}} = a, \text{ say.}$$

Then $a^2 = -j \qquad a^3 = -ja = -a^* \qquad a^4 = -1$

$$a^5 = -a \qquad a^6 = j \qquad a^7 = ja = a^* \qquad a^8 = 1$$

From the above, it can be seen that:

$$W_8^4 = -W_8^0$$
$$W_8^5 = -W_8^1$$
$$W_8^6 = -W_8^2$$
$$W_8^7 = -W_8^3$$

Also, if nk falls outside the range 0-7, we still get one of the above values:

eg. if $n = 5$ and $k = 7$, $\quad W_8^{35} = a^{35} = (a^8)^4 \cdot a^3 = a^3$

DECIMATION IN TIME ALGORITHM

Let us begin by splitting the single summation over N samples into 2 summations, each with $\frac{N}{2}$ samples, one for k even and the other for k odd.
Substitute $m = \frac{k}{2}$ for k even and $m = \frac{k-1}{2}$ for k odd and write:

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_N^{(2m+1)n}$$

Note that $\quad W_N^{2mn} = e^{-j\frac{2\pi}{N}(2mn)} = e^{-j\frac{2\pi}{\frac{N}{2}}mn} = W_{\frac{N}{2}}^{mn}$

Therefore $\quad F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_{\frac{N}{2}}^{mn} + W_N^m \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_{\frac{N}{2}}^{mn}$

$$\text{ie. } F[n] = G[n] + W_N^m H[n]$$

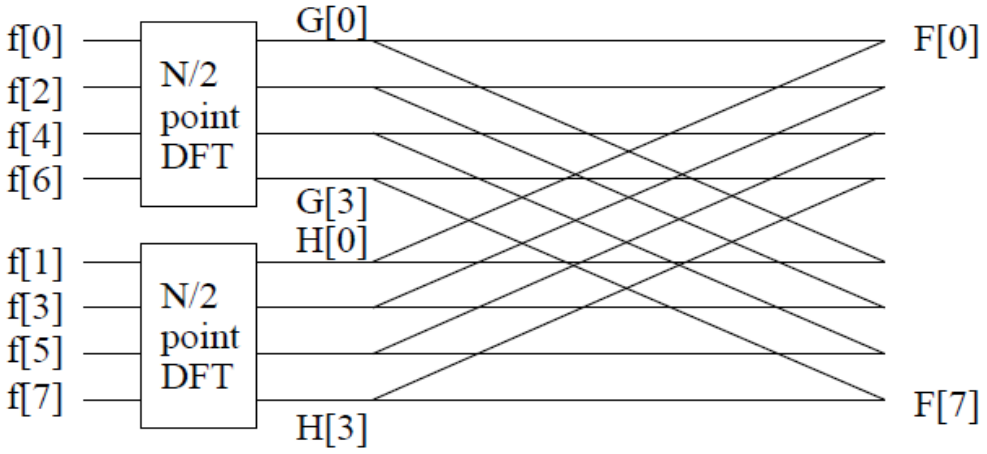
Thus the N-point DFT $F[n]$ can be obtained from two $\frac{N}{2}$-point transforms, one on even input data, $G[n]$, and one on odd input data, $H[n]$. Although the frequency index n ranges over N values, only $\frac{N}{2}$ values of $G[n]$ and $H[n]$ need to be computed since $G[n]$ and $H[n]$ are periodic in n with period $\frac{N}{2}$.

For example, for $N = 8$:

- even input data $f[0]f[2]f[4]f[6]$
- odd input data $f[1]f[3]f[5]f[7]$

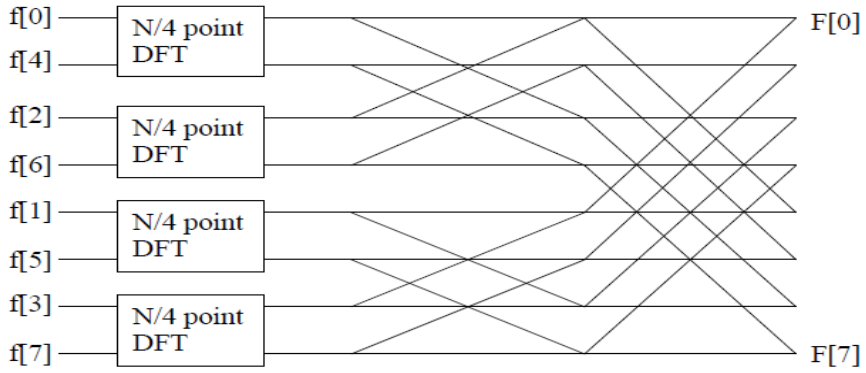
$$\begin{aligned}
F[0] &= G[0] + W_8^0 H[0] \\
F[1] &= G[1] + W_8^1 H[1] \\
F[2] &= G[2] + W_8^2 H[2] \\
F[3] &= G[3] + W_8^3 H[3] \\
F[4] &= G[0] + W_8^4 H[0] = G[0] - W_8^0 H[0] \\
F[5] &= G[1] + W_8^5 H[1] = G[1] - W_8^1 H[1] \\
F[6] &= G[2] + W_8^6 H[2] = G[2] - W_8^2 H[2] \\
F[7] &= G[3] + W_8^7 H[3] = G[3] - W_8^3 H[3]
\end{aligned}$$

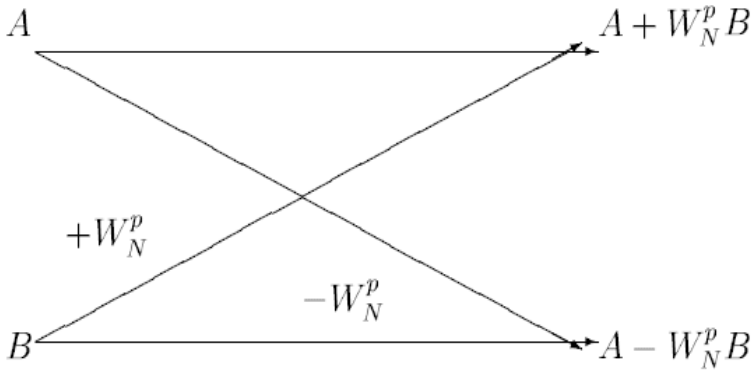
This can be graphically shown as



Assuming than N is a power of 2, we can repeat the above process on the two $\frac{N}{2}$-point transforms, breaking them down to $\frac{N}{4}$-point transforms, etc…, until we come down to 2-point transforms. For $N = 8$, only one further stage is needed

Thus the FFT is computed by dividing up, or *decimating*, the sample sequence $f[k]$ into sub-sequences until only 2-point DFT's remain. Since it is the input, or time, samples which are divided up, this algorithm is known as the *decimation-in-time* (DIT) algorithm. (An equivalent algorithm exists for which the output, or frequency, points are sub-divided – the decimation-in-frequency algorithm.)





where A and B are complex numbers. Thus a butterfly computation requires one complex multiplication and 2 complex additions.

Note also, that the input samples are "bit-reversed" (see table below) because at each stage of decimation the sequence input samples is separated into even- and odd- indexed samples.

(*NB*: the bit-reversal algorithm only applies if N is an integral power of 2).

Computational Speed of FFT

The DFT requires N^2 complex multiplications. At each stage of the FFT (i.e. each halving) $\frac{N}{2}$ complex multiplications are required to combine the results of the previous stage. Since there are $(log_2 N)$ stages, the number of complex multiplications required to evaluate an N-point DFT with the FFT is approximately $N/2log_2 N$ (approximately because multiplications by factors such as W_N^0, $W_N^{\frac{N}{2}}$, $W_N^{\frac{N}{4}}$ and $W_N^{\frac{3N}{4}}$ are really just complex additions and subtractions).

N	N^2 (DFT)	$\frac{N}{2}log_2 N$ (FFT)	saving
32	1,024	80	92%
256	65,536	1,024	98%
1,024	1,048,576	5,120	99.5%