FFT*

*Steven W. Smith „The Scientist and Engineer's Guide to Digital Signal Processing" (http://www.dspguide.com/ch12/2.htm)

Transformata Fouriera (DFT)

$$\phi_k = \sum_{j=0}^{N-1} \Phi_j \, e^{\frac{i2\pi kj}{N}} \tag{1}$$

i odwrotna

$$\Phi_j = \frac{1}{N} \sum_{k=0}^{N-1} \phi_k \, e^{\frac{-i2\pi kj}{N}} \tag{2}$$

Można to policzyć wprost – wymaga N^2 operacji (sin i cos)

– *koncepcja Danielson&Lanczos (1942)*
też Gauss (1805) i Cooley&Tukey (1965)

Jeśli podzielić zbiór pomiarów $\{\Phi_j\}$ na pół – na elementy parzyste

i nieparzyste $\{\Phi_j\} = \{\Psi_j\} + \{\Omega_j\}$

$$\begin{cases} \Phi_{2j} = \Psi_j \\ \Phi_{2j+1} = \Omega_j \end{cases} \tag{3}$$

to

$$\phi_k = \sum_{j=0}^{N-1} \Phi_j \, e^{\frac{i2\pi kj}{N}} = \tag{4}$$

$$= \sum_{j=0}^{N/2-1} \Phi_{2j} \, e^{\frac{i2\pi kj}{N/2}} + e^{\frac{i2\pi k}{N}} \sum_{j=0}^{N/2-1} \Phi_{2j+1} \, e^{\frac{i2\pi kj}{N/2}} = \tag{5}$$

$$= \sum_{j=0}^{N/2-1} \Psi_j \, e^{\frac{i2\pi kj}{N/2}} + e^{\frac{i2\pi k}{N}} \sum_{j=0}^{N/2-1} \Omega_j \, e^{\frac{i2\pi kj}{N/2}} = \tag{6}$$

$$= \begin{cases} \psi_k + e^{\frac{i2\pi k}{N}} \omega_k \ , \ k = 0, ..., N/2 - 1 \\ \psi_k - e^{\frac{i2\pi k}{N}} \omega_k \ , \ k = N/2, ..., N - 1 \end{cases} \tag{7}$$

A jak wygląda tranformata Fouriera dla $N = 1$?

$$\phi_k = \sum_{j=0}^{0} \Phi_j \, e^{i2\pi kj} = \Phi_k \tag{8}$$

Finalnie mamy rekurencyjny przepis na trasformatę Fouriera

- dzielimy zbiór pomiarów na pół (parzyste i nieparzyste) – aż do uzyskania zbiorów jednoelementowych

- dla zbiorów jednolementowych tranformata jest identyczna z pomiarem

- ze zbiorów współczynników Fouriera M elementowych liczymy zbiór współczynników $2M$ elementowy – aż do uzyskania zbioru N elementowego

Uwaga: zbiór pomiarów musi mieć $N = 2^n$ elementów!

Na obrazkach to (algorytm Cooley-Tukey 1965) wygląda tak:

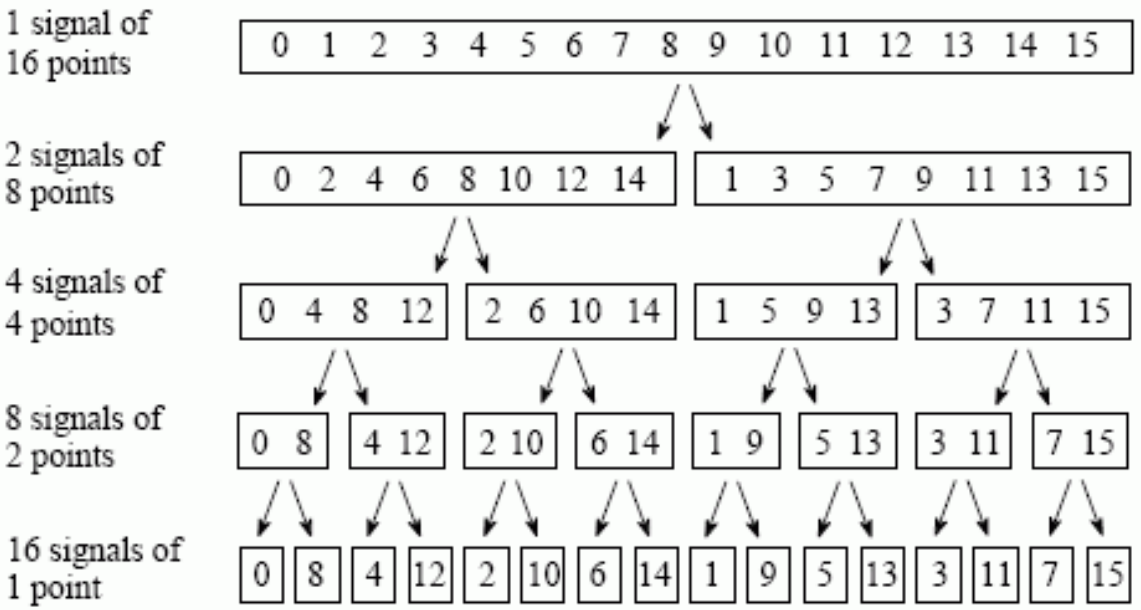


FIGURE 12-2
The FFT decomposition. An N point signal is decomposed into N signals each containing a single point. Each stage uses an *interlace decomposition*, separating the even and odd numbered samples.

Sample numbers in normal order			Sample numbers after bit reversal	
Decimal	*Binary*		*Decimal*	*Binary*
0	0000		0	0000
1	0001		8	1000
2	0010		4	0100
3	0011		12	1100
4	0100		2	0010
5	0101		10	1010
6	0110	⇨	6	0100
7	0111		14	1110
8	1000		1	0001
9	1001		9	1001
10	1010		5	0101
11	1011		13	1101
12	1100		3	0011
13	1101		11	1011
14	1110		7	0111
15	1111		15	1111

FIGURE 12-3
The FFT bit reversal sorting. The FFT time domain decomposition can be implemented by sorting the samples according to bit reversed order.

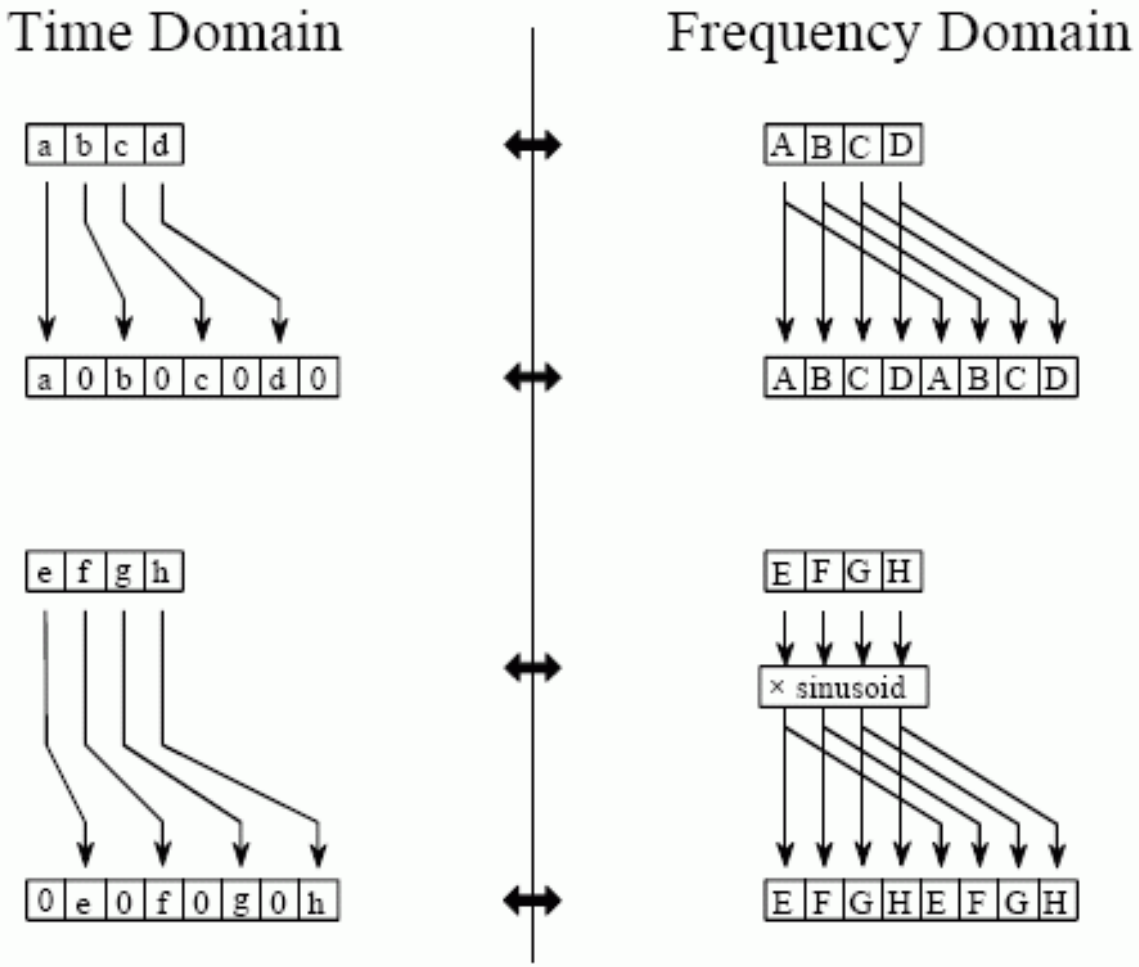
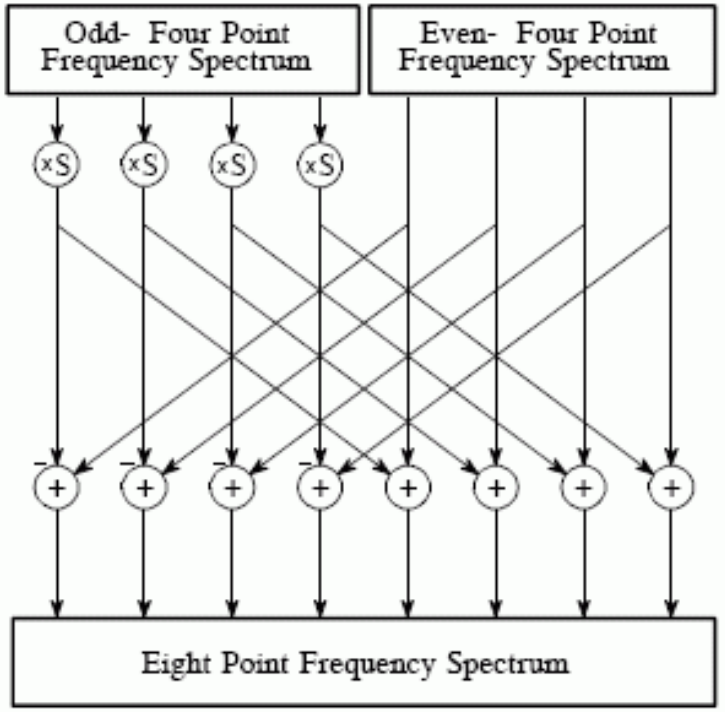


FIGURE 12-4
The FFT synthesis. When a time domain signal is diluted with zeros, the frequency domain is duplicated. If the time domain signal is also shifted by one sample during the dilution, the spectrum will additionally be multiplied by a sinusoid.

FIGURE 12-5
FFT synthesis flow diagram. This shows the method of combining two 4 point frequency spectra into a single 8 point frequency spectrum. The ×S operation means that the signal is multiplied by a sinusoid with an appropriately selected frequency.



FFT wymaga $N \log_2 N$ operacji

Sam algorytm FFT (Numerical Recipes r. 12.2):

```
/*************************************************************/
double sqr(double arg) { return arg*arg; }

void swap(double *a, double *b) { double t=*a; *a=*b; *b=t; }

void fft( double data[], int N, int isign ) {
  int n = N << 1;
  int i, j, m;
  double theta, wr, wpr, wpi, wi, wtemp;
  double tempr, tempi, sqrtN;
  int mmax,istep;
  j = 0;                          /* bit reversal section */
  for( i = 0; i < n; i += 2 ) {
    if( j > i ) {
      swap( data+j, data+i);
      swap( data+j+1, data+i+1 );
    }
    m = N;
    while( m >= 2 && j >= m ) {
      j -= m;
      m >>= 1;
    }
    j += m;
  }
```

```
  for( mmax = 2; n > mmax; ) {   /* Daniel-Lanczos section */
    istep = mmax << 1;
    theta = isign * ( 2.0*M_PI / mmax );
    wpr = -2.0 * sqr( sin( 0.5*theta ) );
    wpi = sin( theta );
    wr = 1.0; wi = 0.0;
    for( m = 0; m < mmax; m += 2 ) {
      for( i = m; i < n; i += istep ) {
        j = i + mmax;
        tempr = wr*data[j] - wi*data[j+1];
        tempi = wr*data[j+1] + wi*data[j];
        data[j] = data[i] - tempr;
        data[j+1] = data[i+1] - tempi;
        data[i] += tempr;
        data[i+1] += tempi;
      }
      wr = (wtemp=wr)*wpr - wi*wpi + wr;
      wi = wi*wpr + wtemp*wpi + wi;
    }
    mmax = istep;
  }
  sqrtN = sqrt( N );              /* normalisation section */
  for( i = 0; i < n; i++ )
    data[i] /= sqrtN;
}
```

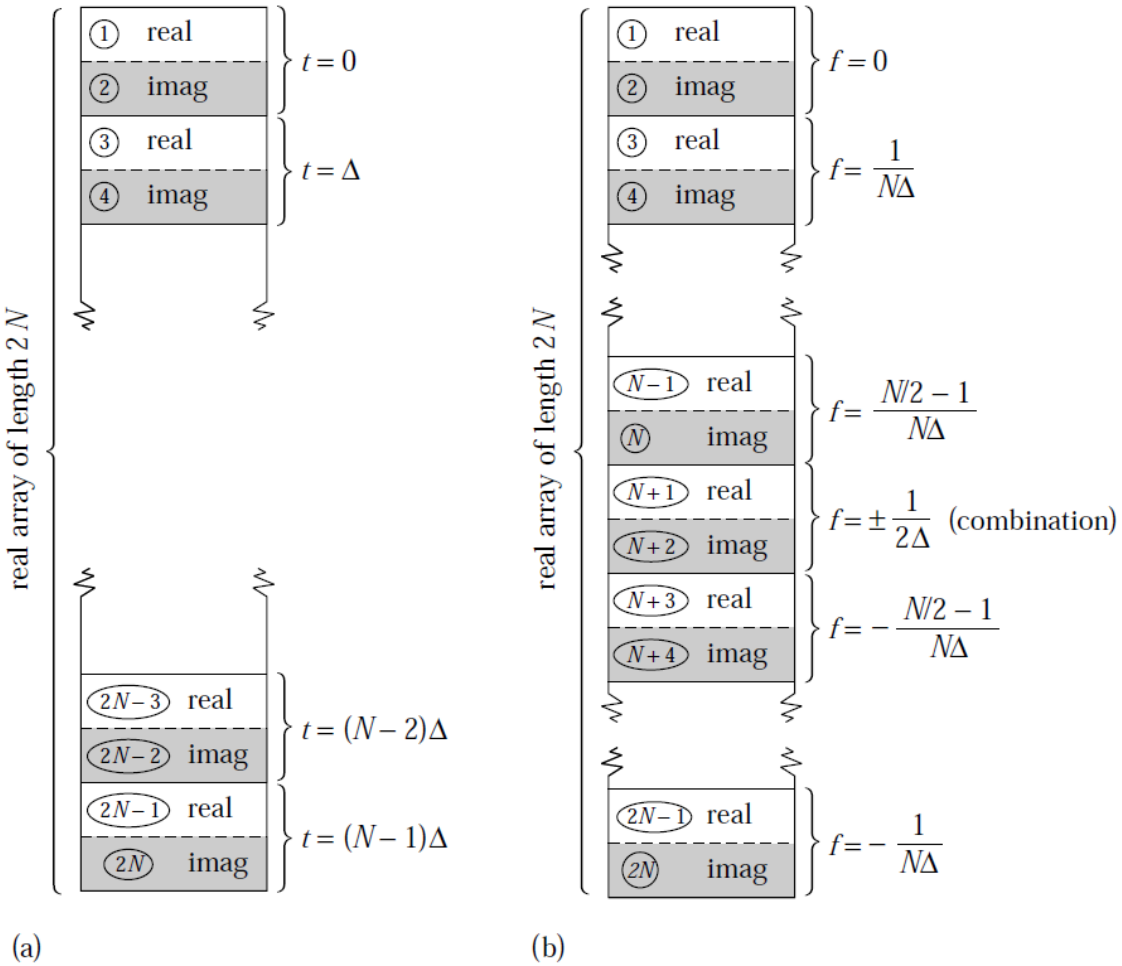


Figure 12.2.2. Input and output arrays for FFT. (a) The input array contains N (a power of 2) complex time samples in a real array of length $2N$, with real and imaginary parts alternating. (b) The output array contains the complex Fourier spectrum at N values of frequency. Real and imaginary parts again alternate. The array starts with zero frequency, works up to the most positive frequency (which is ambiguous with the most negative frequency). Negative frequencies follow, from the second-most negative up to the frequency just below zero.