La Transformada Rápida de Fourier

Cuando la función f(t) está dada por una lista de N valores $f(t_1)$, $f(t_2)$, ...$f(t_N)$ se dice que está ***discretizada o muestreada***, entonces la integral que define la Transformada de Fourier:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

Se convierte en la sumatoria

$$F(n) = \sum_{k=1}^{N} f(t_k) e^{-j\frac{2\pi n}{N}(k-1)}, \quad \text{para } 1 \le n \le N$$

(Donde k es la frecuencia discreta)
Llamada ***Transformada Discreta de Fourier***

La Transformada Rápida de Fourier

La Transformada Discreta de Fourier (DFT) requiere el cálculo de N funciones exponenciales para obtener F(n), lo cual resulta un esfuerzo de cálculo enorme para N grande.

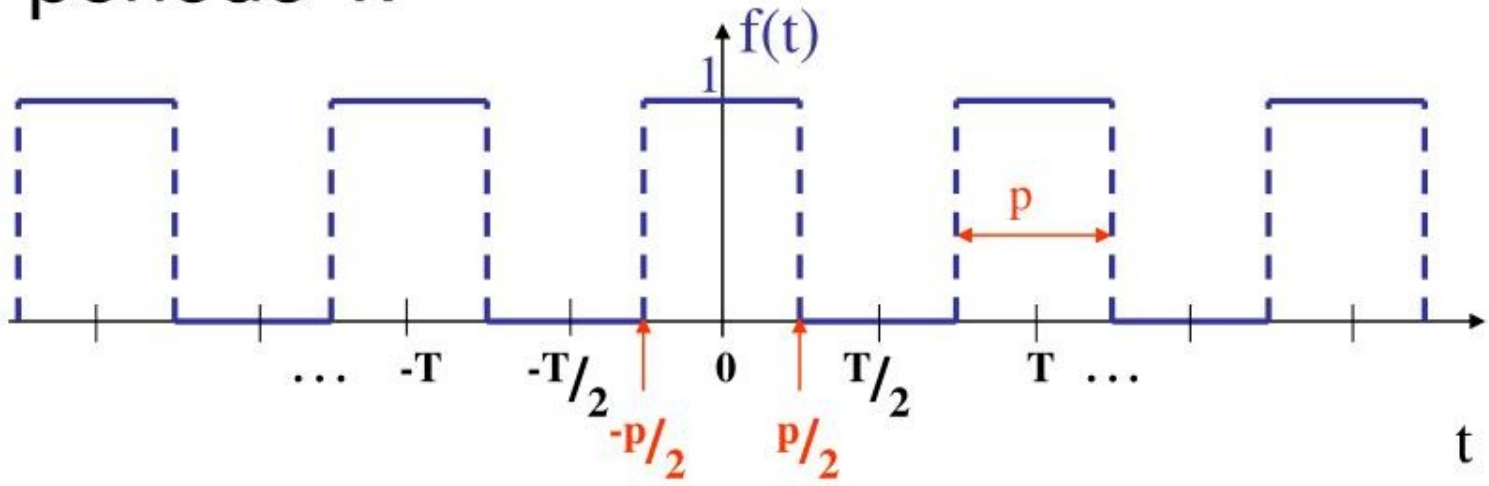
Se han desarrollado métodos que permiten ahorrar cálculos y evaluar de manera rápida la Transformada discreta, a estos métodos se les llama

Transformada Rápida de Fourier (FFT)

La FFT y la Serie de Fourier

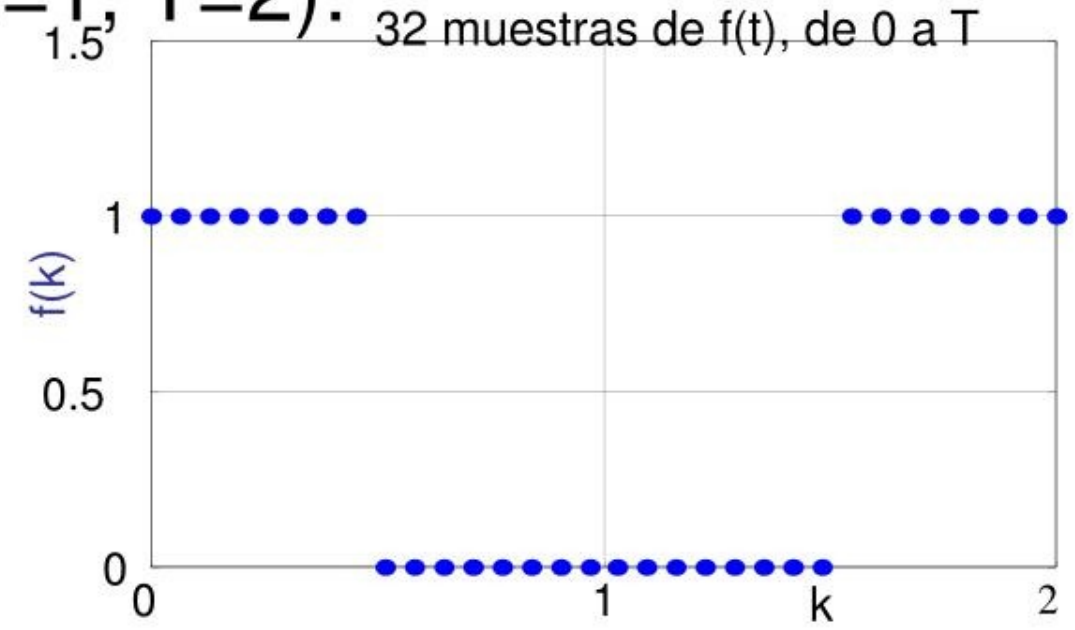
Podemos hacer uso de la FFT para calcular los coeficientes c_n y c_{-n} de la Serie compleja de Fourier como sigue:

Ejemplo: Sea f(t) el tren de pulsos de ancho p y periodo T.



La FFT y la Serie de Fourier

La versión muestreada f(k) de f(t) sólo puede tomar un número finito de puntos. Tomemos por ejemplo N=32 puntos cuidando que cubran el intervalo de 0 a T (con p=1, T=2):



La FFT y la Serie de Fourier

Para obtener estas 32 muestras usando Matlab se puede hacer lo siguiente:

```
k=0:31
f=[(k<8)|(k>23)]
Plot(k,f,'o')
```

La FFT y la Serie de Fourier

Con los 32 puntos f(k) calculamos F(n) mediante la FFT, por ejemplo, en Matlab:

```
F=fft(f)/N;
```

Con lo que obtenemos 32 valores complejos de F(n). Estos valores son los coeficientes de la serie compleja ordenados como sigue:

n	1	2	3	4	...	16	17	18	19	...	32
F(n)	c_0	c_1	c_2	c_3	...	c_{15}	c_{-16}	c_{-15}	c_{-14}	...	c_{-1}

La FFT y la Serie de Fourier

Podemos graficar el espectro de amplitud reordenando previamente F(n) como sigue

```
aux=F;
F(1:16)=aux(17:32);
F(17:32)=aux(1:16);
```

F(n) queda:

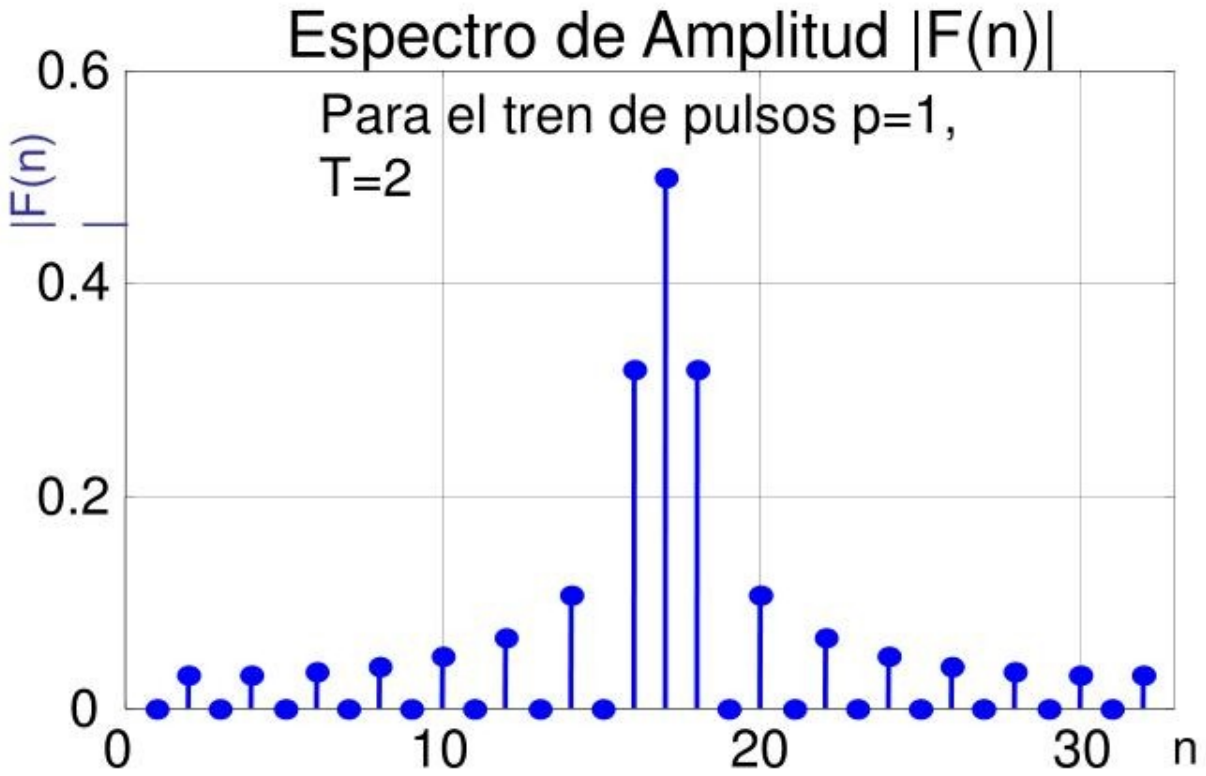
n	1	...	13	14	15	16	17	18	19	...	32
F(n)	c_{-16}	...	c_{-3}	c_{-2}	c_{-1}	c_0	c_1	c_2	c_3	...	c_{15}

Y para graficar el espectro de amplitud:

```
stem(abs(F))
```

Obteniéndose:

La FFT y la Serie de Fourier

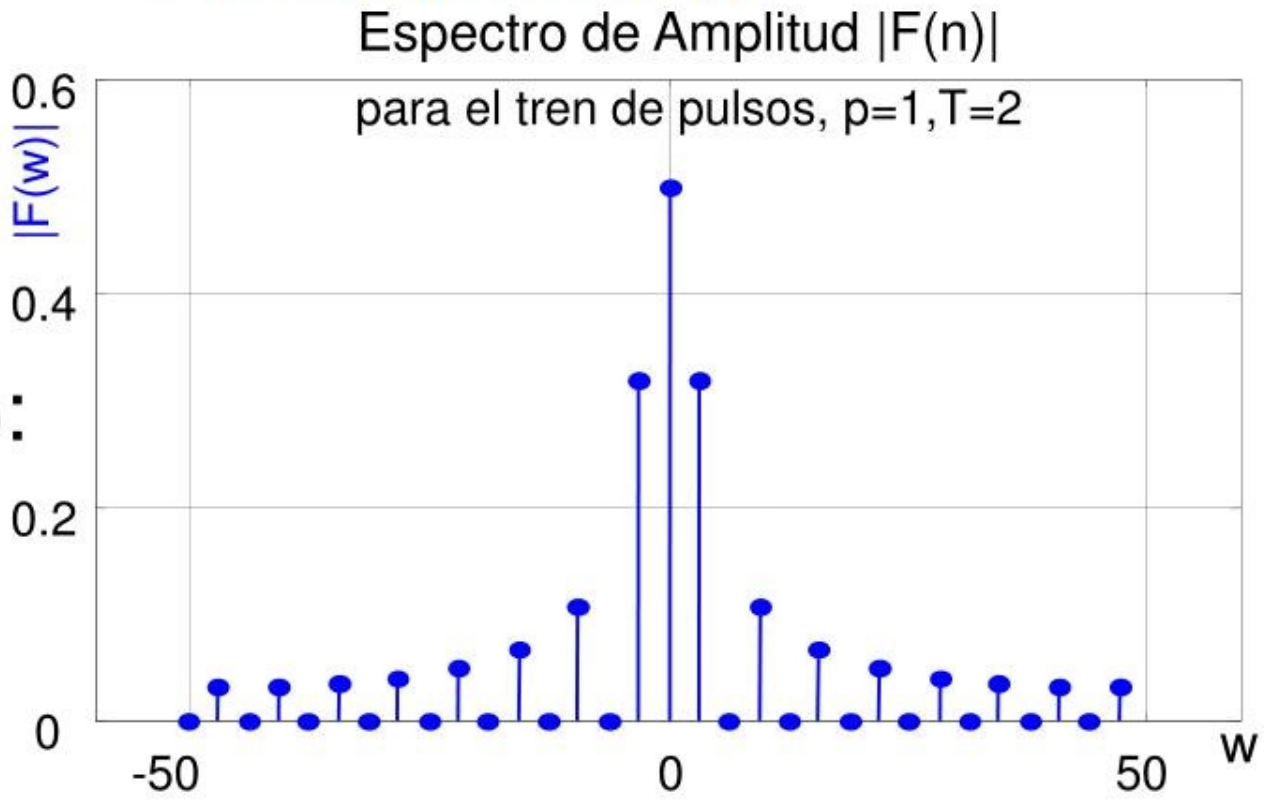


Si deseamos una escala horizontal en unidades de frecuencia (rad/seg):

La FFT y la Serie de Fourier

```
w0=2*pi/T;
n=-16:15;
w=n*w0;
Stem(w,abs(F))
```

Obteniendo:



La FFT y la Serie de Fourier

También podemos obtener los coeficientes de la forma trigonométrica, recordando que:

$$c_n = \frac{1}{2}(a_n - jb_n), \quad c_{-n} = \frac{1}{2}(a_n + jb_n)$$

Podemos obtener

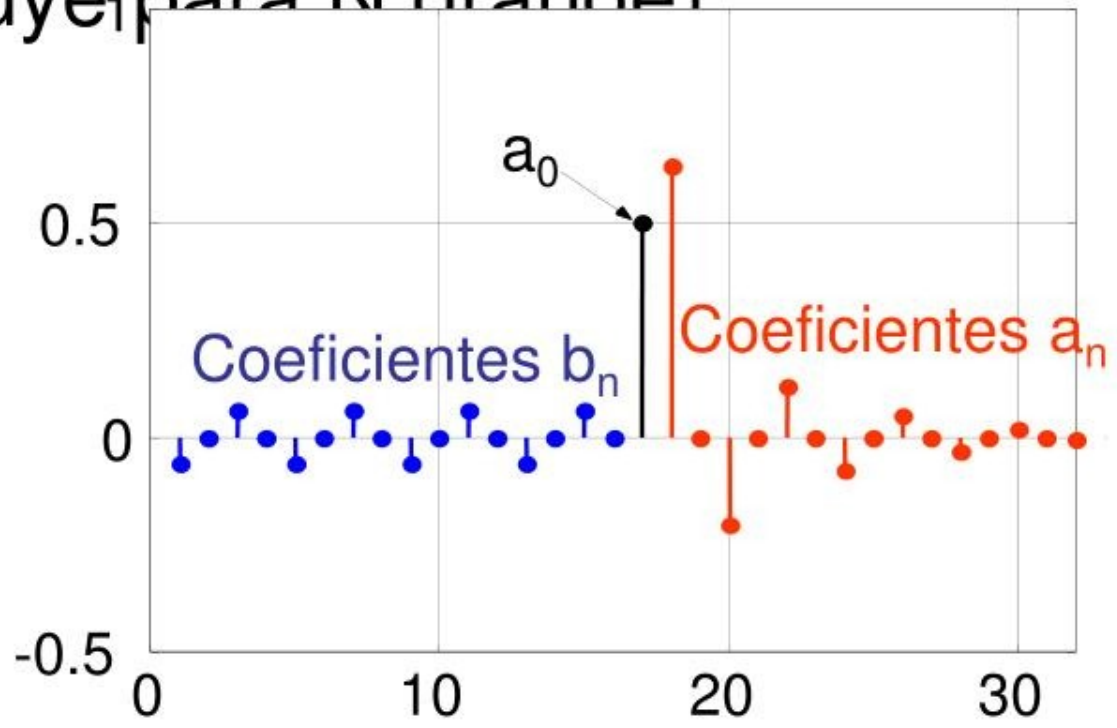
$$a_0 = c_0, \quad a_n = 2\,\mathrm{Re}(c_n), \quad b = -2\,\mathrm{Im}(c_n)$$

Para el ejemplo se obtiene: a_0=0.5, a_n=b_n=0 (para n par), además para n impar:

n	1	3	5	7	9	11	13	15
a_n	0.6346	-0.2060	0.1169	-0.0762	0.0513	-0.0334	0.0190	-0.0062
b_n	-0.0625	0.0625	-0.0625	0.0625	-0.0625	0.0625	-0.0625	0.0625

La FFT y la Serie de Fourier

Como el tren de pulsos es una función par, se esperaba que $b_n=0$; (el resultado obtenido es erróneo para b_n, pero el error disminuye para N grande):



La FFT y la Serie de Fourier

Tarea: Usar el siguiente código para generar 128 puntos de una función periódica con frecuencia fundamental $\omega_0=120\pi$ (60 hertz) y dos armónicos impares en el intervalo [0,T]:

```
N=128;
w0=120*pi;
T=1/60;
t=0:T/(N-1):T;
f=sin(w0*t)+0.2*sin(3*w0*t)+0.1*sin(11*w0*t);
```

Usando una función periódica diferente a la subrayada:

a) Graficar la función.

b) Obtener y graficar el espectro de amplitud de la señal usando la función FFT

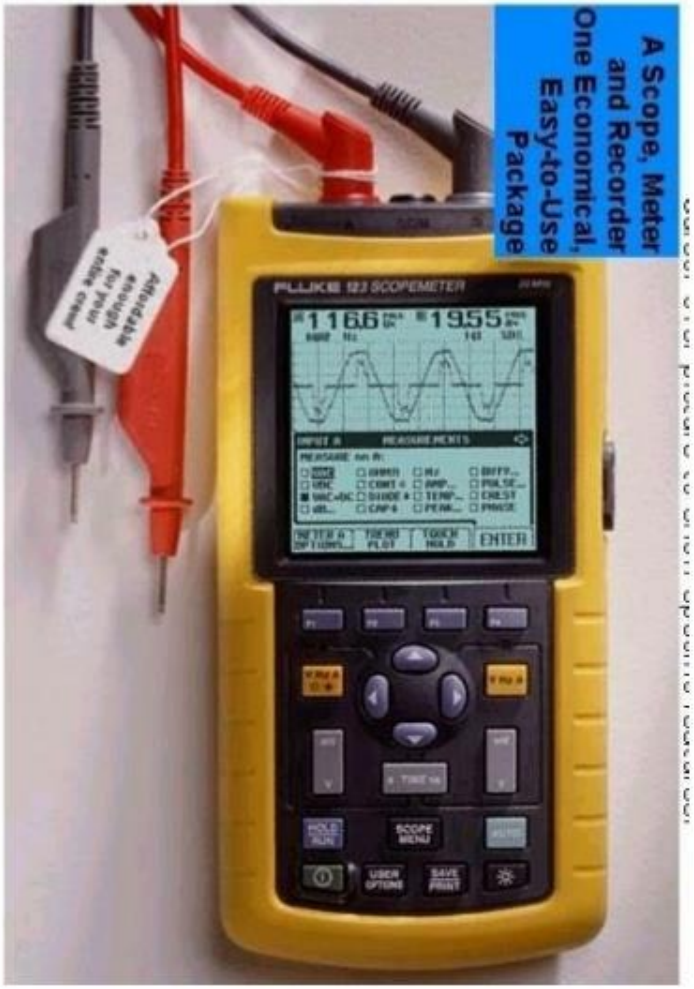
Medidores Digitales

La FFT ha hecho posible el desarrollo de equipo electrónico digital con la capacidad de cálculo de espectros de frecuencia para señales del mundo real, por ejemplo:

1) Osciloscopio digital Fuke 123 (**$ 18,600.00 M.N.**)
2) Osc. digital Tektronix THS720P (**$3,796 dls**)
3) Power Platform PP-4300

Medidores Digitales

El Fluke 123 scope meter



Medidores Digitales

Tektronix THS720P (osciloscopio digital)

Medidores Digitales

Analizador de potencia PP-4300

Es un equipo especializado en monitoreo de la calidad de la energía: permite medición de 4 señales simultáneas (para sistemas trifásicos)

Fast Fourier Transform (FFT), I

- DFT appears to be an $O(N^2)$ process.
- Danielson and Lanczos; DFT of length N can be rewritten as the sum of two DFT of length $N/2$.

$$
\begin{aligned}
H_k &= \sum_{j=0}^{N-1} e^{2\pi i jk/N} h_j \\
&= \sum_{j=0}^{N/2-1} e^{2\pi i(2j)k/N} h_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i(2j+1)k/N} h_{2j+1} \\
&= \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} h_{2j} + e^{2\pi ik/N} \sum_{k=0}^{N/2-1} e^{2\pi ijk/(N/2)} h_{2k+1} \\
&= H_k^0 + e^{2\pi ik/N} H_k^1 = H_k^0 + W^k H_k^1, \quad (W \equiv e^{2\pi i/N})
\end{aligned}
$$

- We can do the same reduction of H_k^0 to the transform of its $N/4$ even-numbered input data and $N/4$ odd-numbered data.
- For $N = 2^R$, we can continue applying the reduction until we subdivide the data into the transforms of length 1.
- For every pattern of $\log_2 N$ number of 0's and 1's, there is one-point transformation that is just one of the input number h_n

$$H_k^{1001\ldots001} = h_n \qquad \text{for some } n$$

Fast Fourier Transform (FFT), II

- For N=8

$$H_k = \sum_{j=0}^{7} W^{jk} h_j = h_0 + W^k h_1 + W^{2k} h_2 + W^{3k} h_3 + W^{4k} h_4 + W^{5k} h_5 + W^{6k} h_6 + W^{7k} h_7$$

$$= (h_0 + W^{2k} h_2 + W^{4k} h_4 + W^{6k} h_6) + W^k (h_1 + W^{2k} h_3 + W^{4k} h_5 + W^{6k} h_7)$$

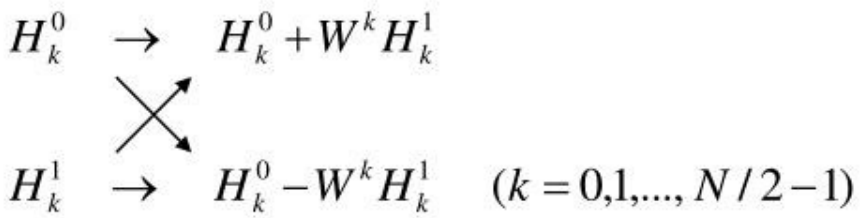
$$= (h_0 + W^{4k} h_4) + W^{2k} (h_2 + W^{4k} h_6) + W^k [(h_1 + W^{4k} h_5) + W^{2k} (h_3 + W^{4k} h_7)]$$

$$= (h_0) + W^{4k} (h_4) + W^{2k} [h_2 + W^{4k} (h_6)] + W^k [(h_1) + W^{4k} (h_5) + W^{2k} [h_3 + W^{4k} (h_7)]]$$

- Since $W^{N/2} = -1$, H_k^0 and H_k^1 have period $N/2$,

$$H_k = H_k^0 + W^k H_k^1, \quad H_{k+N/2} = H_k^0 - W^k H_k^1 \quad \text{for } k = 0,1,..., N/2-1$$

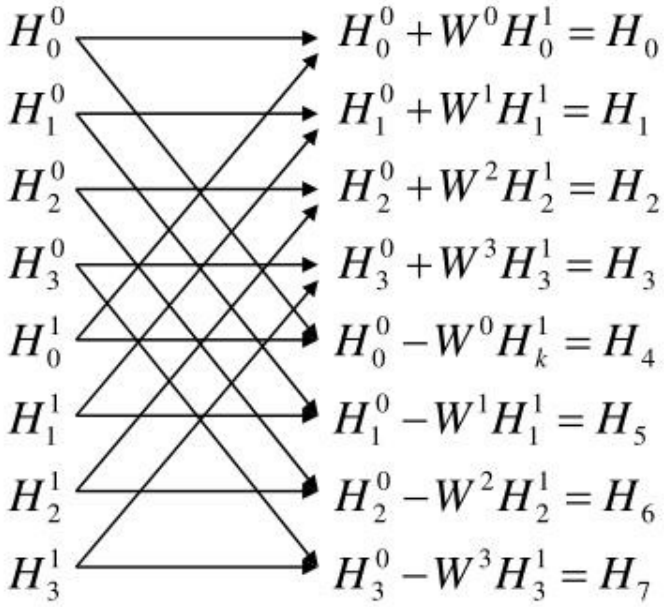
- Diagrammatically (butterfly),



- There are N/2 butterflies for this stage of the FFT, and each butterfly requires one multiplication

Fast Fourier Transform (FFT), III

- So far,

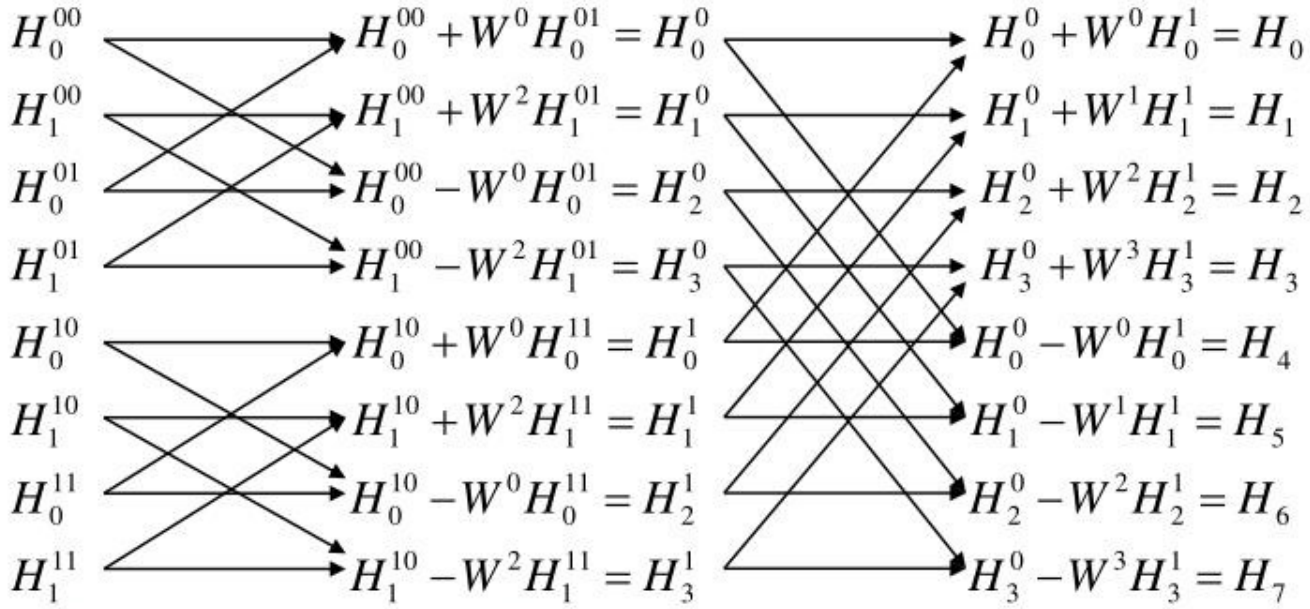


- The splitting of $\{H_k\}$ into two half-size DFTs can be repeated on H_k^0 and H_k^1 themselves,

$$H_k^0 = H_k^{00} + W^{2k} H_k^{01}, \quad H_{k+N/4}^0 = H_k^{00} - W^{2k} H_k^{01}$$
$$H_k^1 = H_k^{10} + W^{2k} H_k^{11}, \quad H_{k+N/4}^1 = H_k^{10} - W^{2k} H_k^{11} \quad \text{for } k = 0,1,\ldots, N/4-1$$

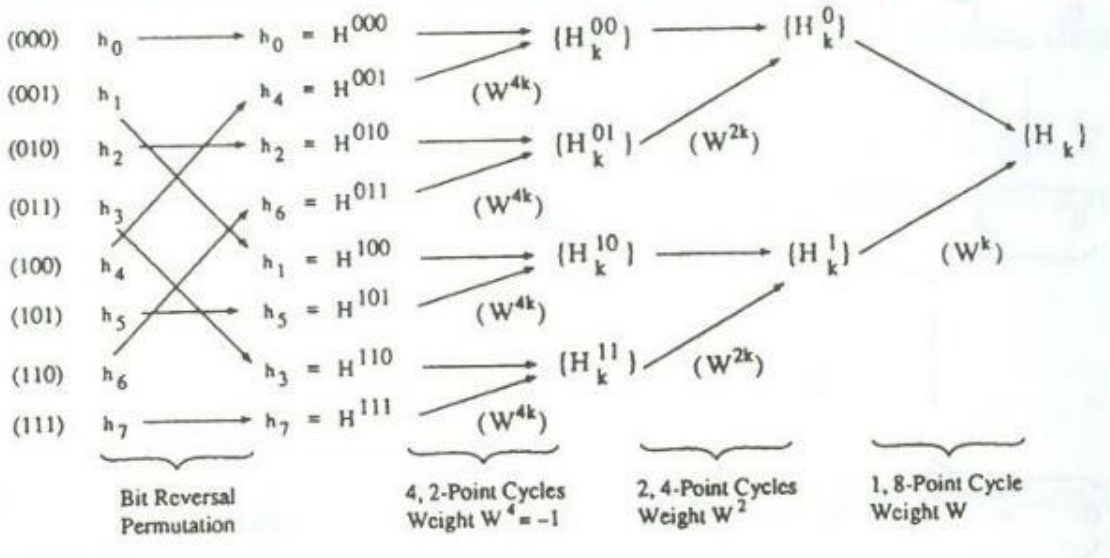
Fast Fourier Transform (FFT), III

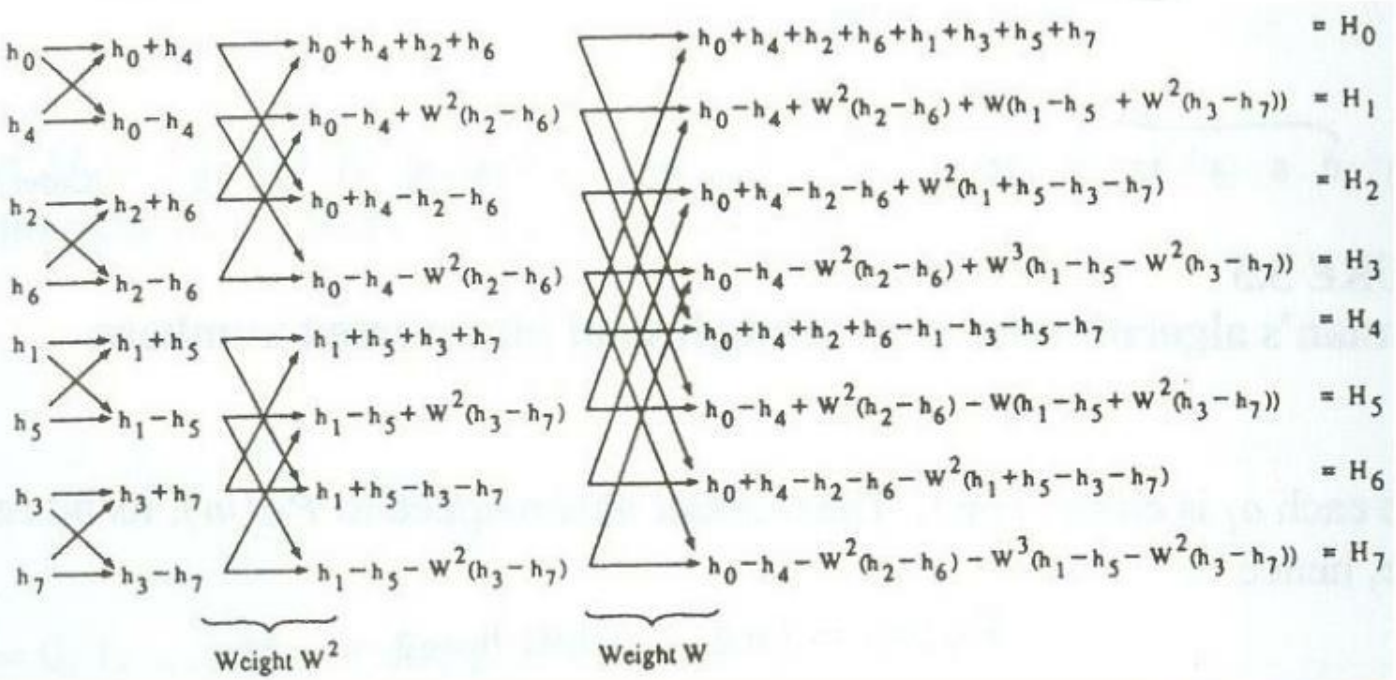
- So far,



  - $\{H_k^{00}\}$ is the $N/4$-point DFT of $\{h_0, h_4, \ldots, h_{N-4}\}$,
  - $\{H_k^{01}\}$ is the $N/4$-point DFT of $\{h_2, h_6, \ldots, h_{N-2}\}$,
  - $\{H_k^{10}\}$ is the $N/4$-point DFT of $\{h_1, h_5, \ldots, h_{N-3}\}$,
  - $\{H_k^{11}\}$ is the $N/4$-point DFT of $\{h_3, h_7, \ldots, h_{N-1}\}$,
  - Note that there is a reversal of the last two digits in the binary expansions of the indices j in $\{h_j\}$.

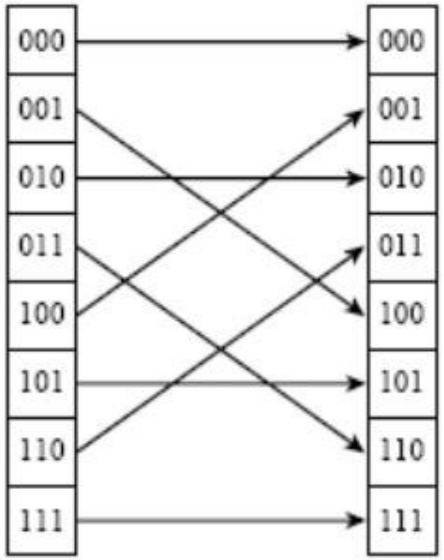
Fast Fourier Transform (FFT), IV





Fast Fourier Transform (FFT), V

- If we continue with this process of halving the order of the DFTS, then after $R=\log_2 N$ stages, we reach where we are performing N one-point DFTs.
  - One-point DFT of the number h_j is just the identity $h_j \rightarrow h_j$
  - Since the reversal of the order of the bits will continue, all bits in the binary expansion of j will be arranged in reverse order.
  - Therefore, to begin the FFT, one must first rearrange $\{h_j\}$ so it is listed in bit reverse order.
- For each of the $\log_2 N$ stages, there are $N/2$ multiplications, hence there are $(N/2)\log_2 N$ multiplications needed for FFT.
  - Much less time than the $(N-1)2$ multiplications needed for a direct DFT calculation.
  - When N=1024, FFT=5120 multiplication, DFT=1,046,529 → savings by a factor of almost 200.

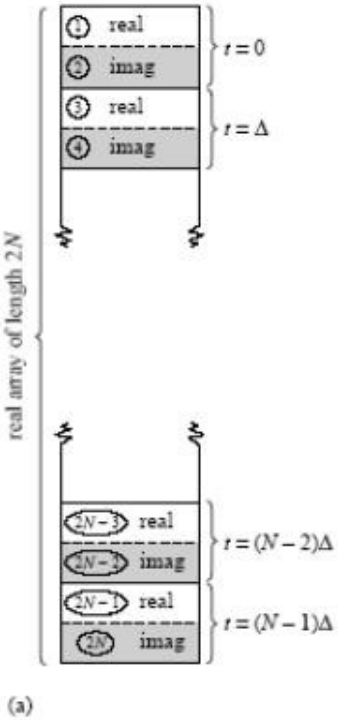


Fast Fourier Transform (FFT), VI

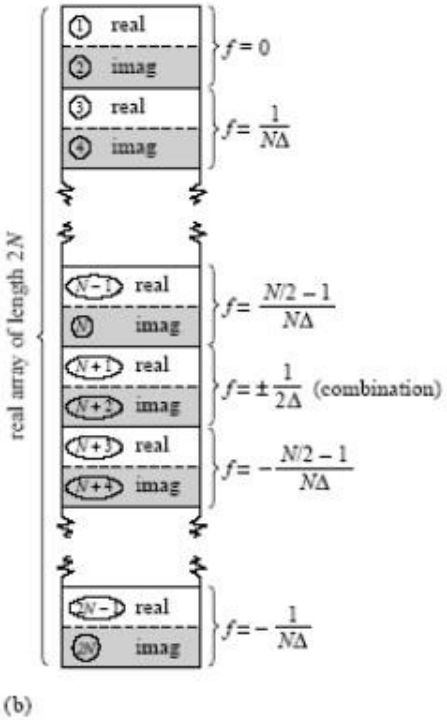
```
#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(float data[], unsigned long nn, int isign)
Replaces data[1..2*nn] by its discrete Fourier transform, if isign is input as 1; or replaces
data[1..2*nn] by nn times its inverse discrete Fourier transform, if isign is input as -1.
data is a complex array of length nn or, equivalently, a real array of length 2*nn. nn MUST
be an integer power of 2 (this is not checked for!).
{
    unsigned long n,mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta;          Double precision for the trigonomet-
    float tempr,tempi;                             ric recurrences.

    n=nn << 1;
    j=1;
    for (i=1;i<n;i+=2) {                       This is the bit-reversal section of the
        if (j > i) {                               routine.
            SWAP(data[j],data[i]);             Exchange the two complex numbers.
            SWAP(data[j+1],data[i+1]);
        }
        m=nn;
        while (m >= 2 && j > m) {
            j -= m;
            m >>= 1;
        }
        j += m;
    }
    Here begins the Danielson-Lanczos section of the routine.
    mmax=2;
    while (n > mmax) {                         Outer loop executed log2 nn times.
        istep=mmax << 1;
        theta=isign*(6.28318530717959/mmax);   Initialize the trigonometric recurrence.
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0;
        wi=0.0;
        for (m=1;m<mmax;m+=2) {                Here are the two nested inner loops.
            for (i=m;i<=n;i+=istep) {
                j=i+mmax;                      This is the Danielson-Lanczos for-
                tempr=wr*data[j]-wi*data[j+1];     mula:
                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;       Trigonometric recurrence.
            wi=wi*wpr+wtemp*wpi+wi;
        }
        mmax=istep;
    }
}
```



(a)



(b)