# 离散傅里叶变换和快速傅里叶变换

Xin Li (李新)
Email: lixustc@ustc.edu.cn
Phone: 0551-63607202

# 为什么用离散傅里叶变换

- 实际的信号是离散的

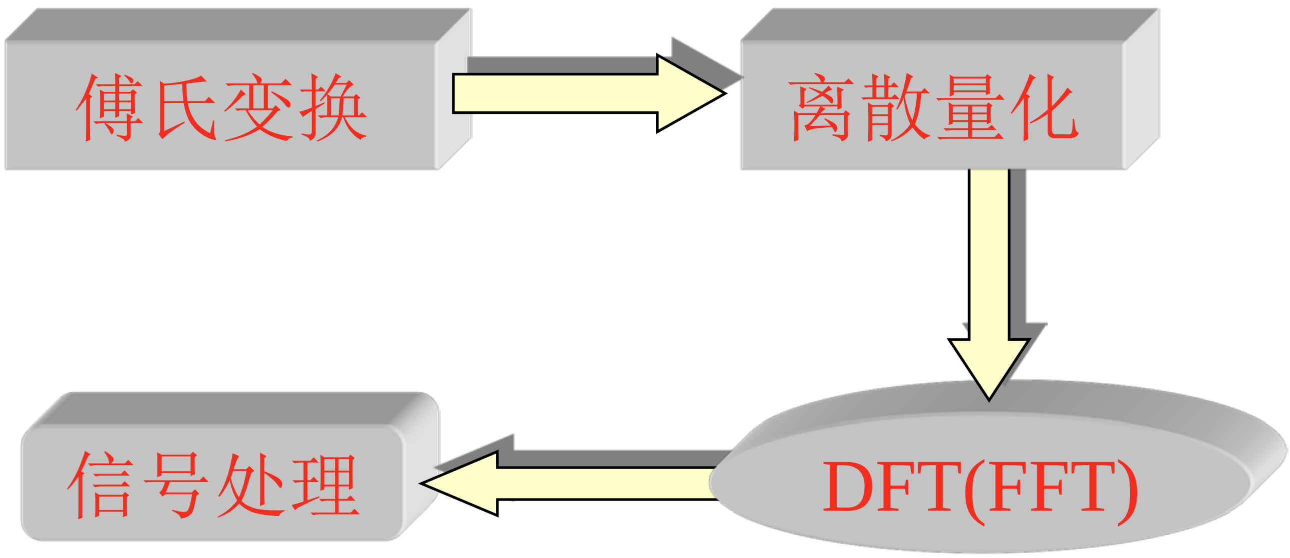
- 分析有限长序列的有用工具;
- 在信号处理的理论上有重要意义;
- 在运算方法上起核心作用
  - 谱分析、卷积等

# 为什么研究连续傅里叶变换

- 连续傅里叶变换------离散傅里叶变换
  - 采样
  - 可以有理论保证
- 离散傅里叶变换------连续傅里叶变换
  - 预测
  - 没有保证

# DFT

- 离散与量化，
- 快速运算。



# 离散时间信号

- 对模拟信号f(t)进行等间隔采样，采样间隔为T;

$$x_j = f(jT), -\infty < j < \infty$$

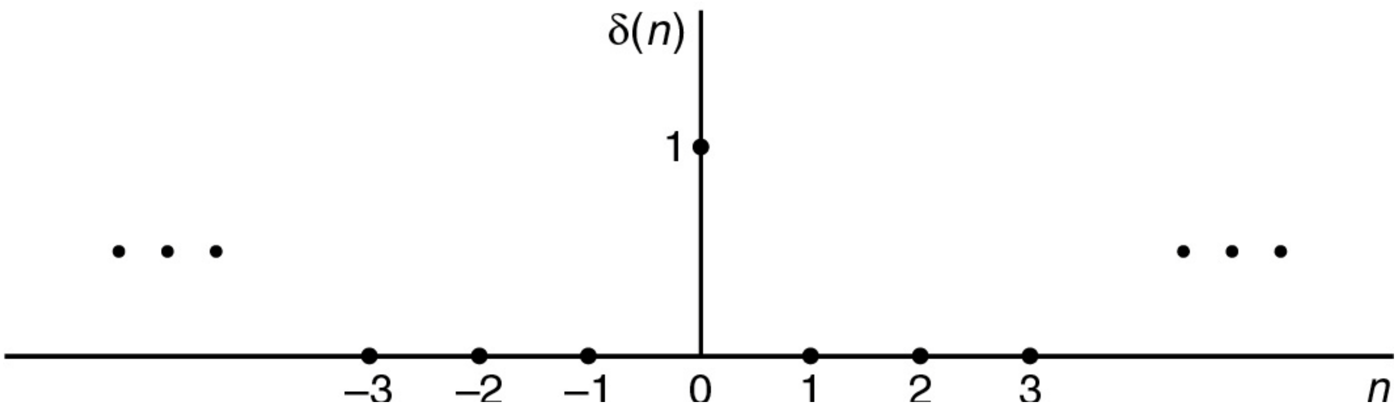
- 周期信号:

$$x_j = x_{j+n}, \exists n$$

- 公式表示、图形表示、集合符号表示

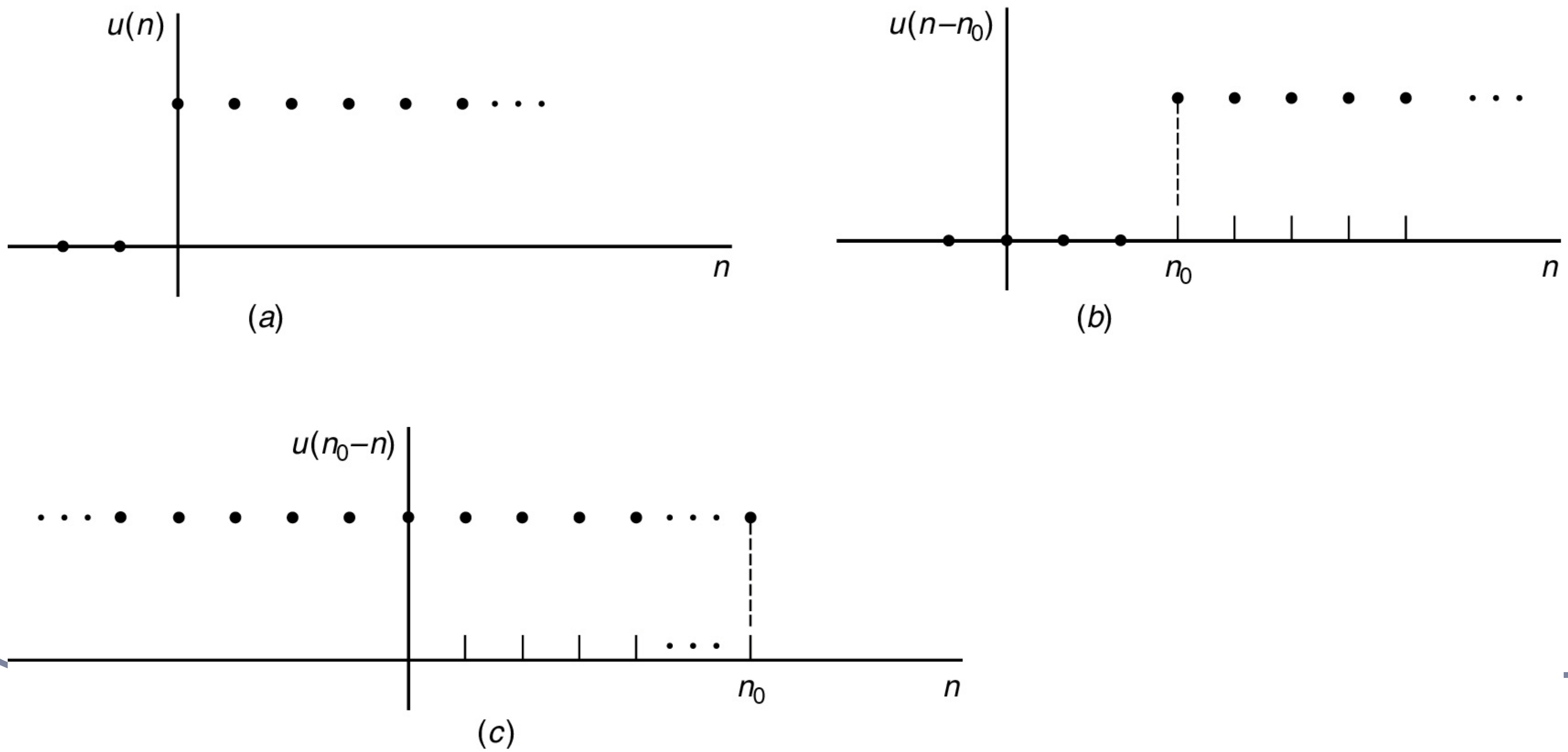
# 单位脉冲序列

$$\delta(j)=\begin{cases}1, & j=0\\ 0, & j\neq 0\end{cases}$$



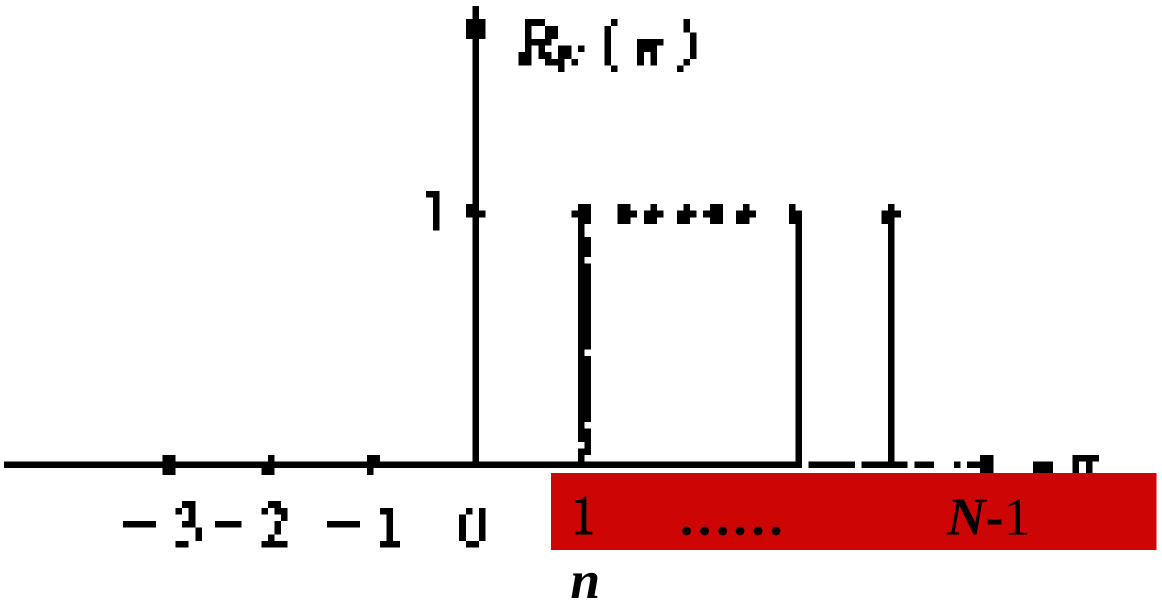
# 单位阶跃序列

$$u(j)=\begin{cases}1, & j\ge 0\\ 0, & j<0\end{cases}$$



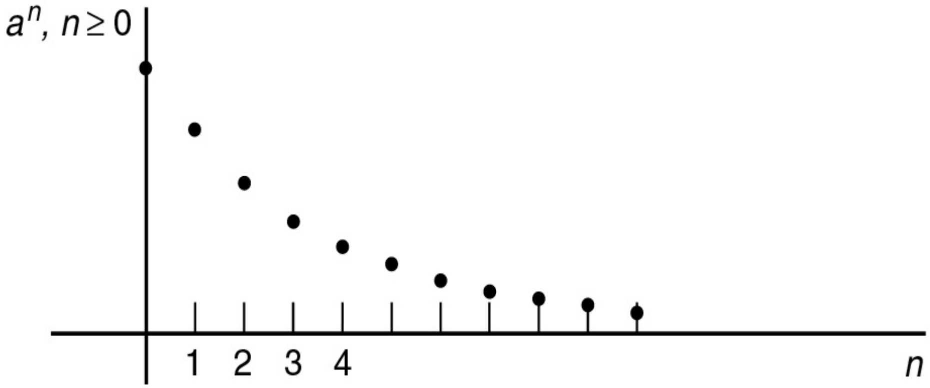
# 矩形序列

$$R_n(j)=\begin{cases}1, & 0\le j\le n-1\\ 0, & j<0,\ j\ge n\end{cases}$$

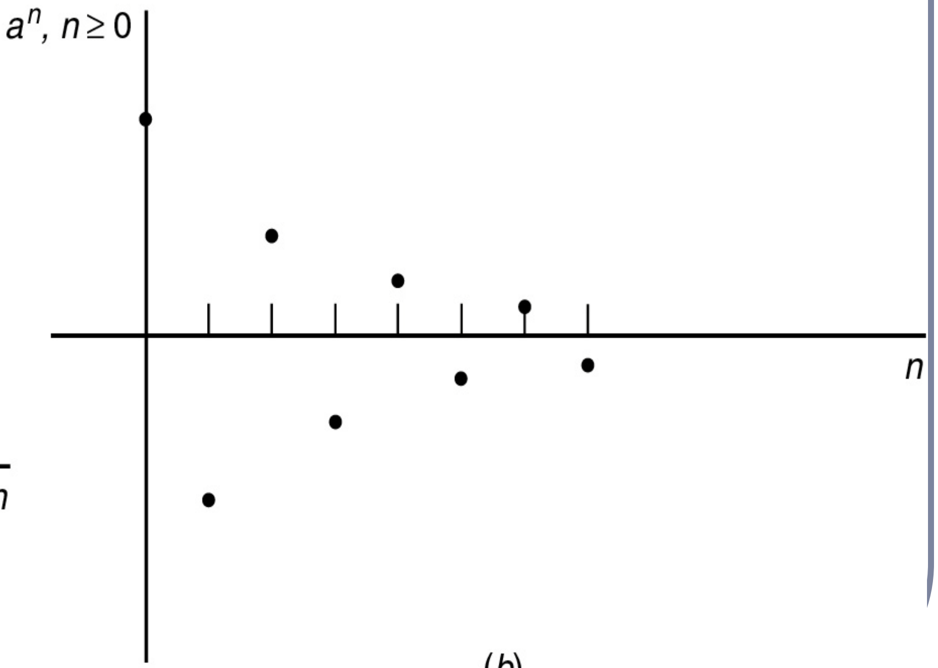


# 实指数序列

$$x(n) = a^n u(n)$$



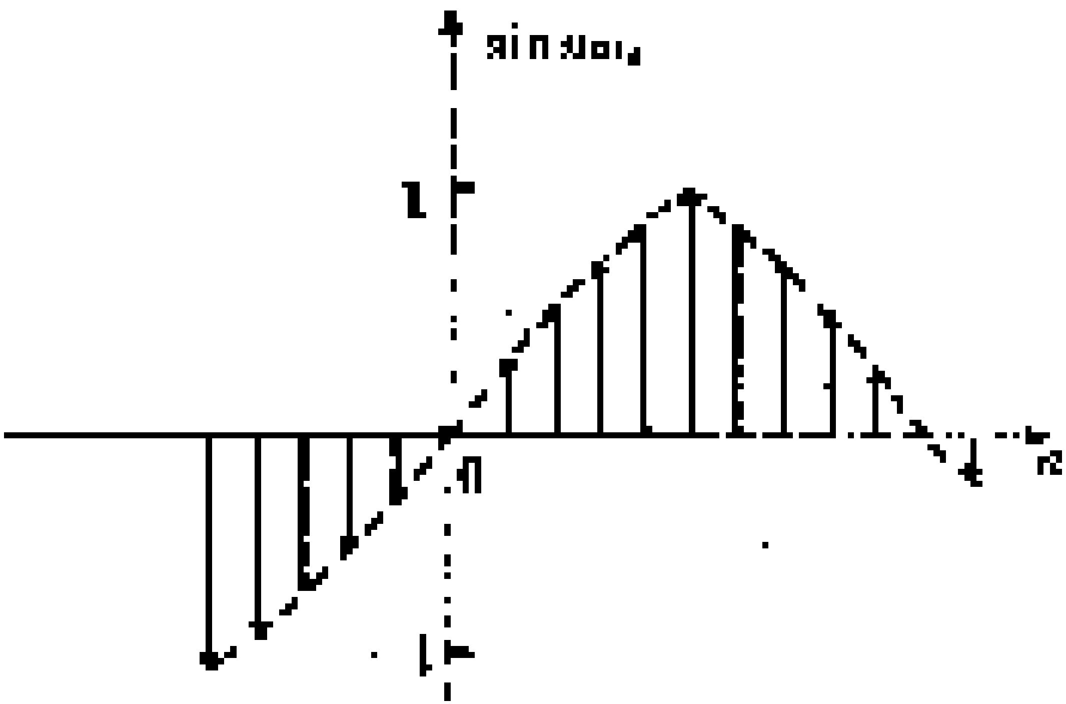

(a)




(b)

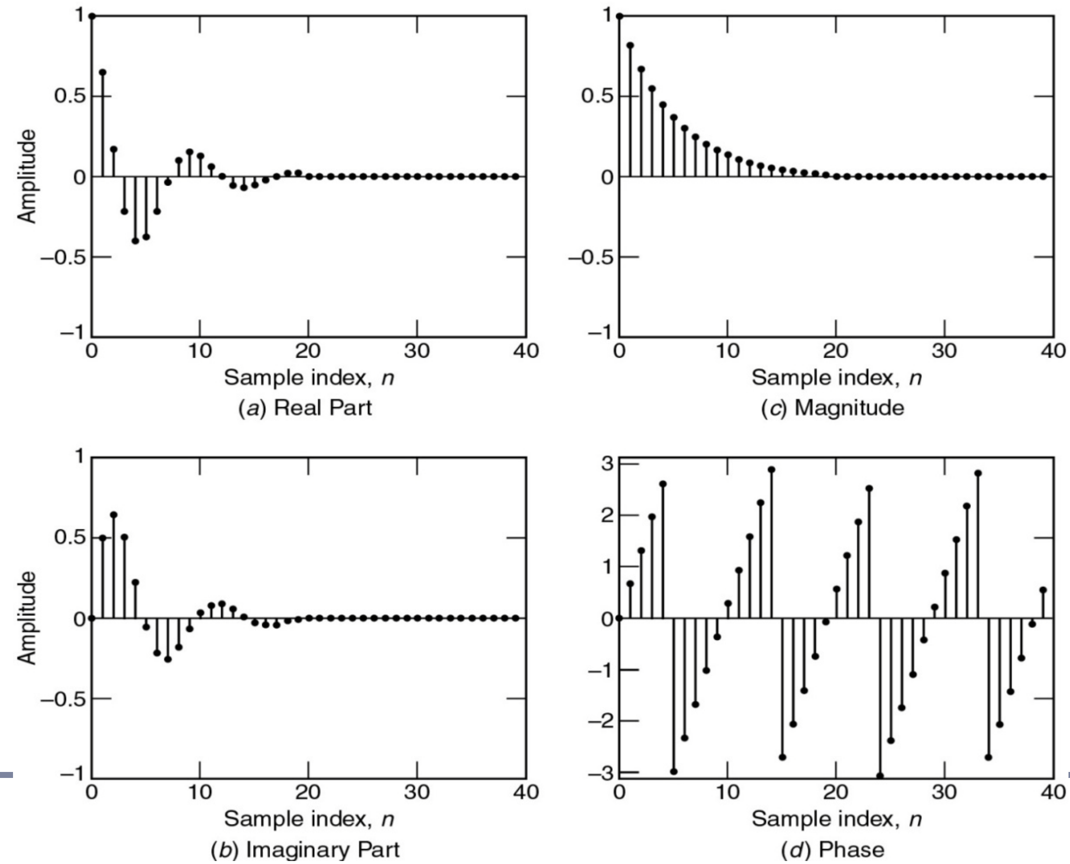
# 正弦序列

$$x(j)=\sin(j\lambda_0)$$



# 复指数序列

$$x(j) = Ae^{(\alpha+i\lambda_0)j} = Ae^{\alpha n}(\cos\lambda_0 j + i\sin\lambda_0 j)$$



# 系列的运算

- 序列的相加：　z(j)=x(j)+y(j)
- 序列的相乘：　f(j)=x(j)*y(j)
- 序列的移位：　y(j)=x(j-j0)
- 序列的能量：　$S=\sum_{j=-\infty}^{\infty}|x(j)|^2$
- 能量有限：　$\sum_{j=-\infty}^{\infty}|x(j)|^2<\infty$
- 序列的单位脉冲序列表示：

$$x(j)=\sum_{k=-\infty}^{\infty}x(k)\delta(j-k)$$

# 周期系列的DFT

- 周期为n的系列：$y=\{y_j\}_{-\infty}^{\infty}, y_j = y_{j+n}$
- DFT：

$$F_n(y)=\bar{y}=\{\bar{y}_j\}$$

$$\bar{y}_j=\sum_{k=0}^{n-1} y_k \overline{\omega}^{jk}, \omega = e^{\frac{2\pi i}{n}}$$

- IDFT:

$$y_j=\frac{1}{n}\sum_{k=0}^{n-1}\bar{y}_k \omega^{jk},$$

# 快速傅里叶变换（FFT）

虽然频谱分析和DFT运算很重要，但在很长一段时间里，由于DFT运算复杂，并没有得到真正的运用，而频谱分析仍大多采用模拟信号滤波的方法解决，直到1965年首次提出DFT运算的一种快速算法以后，情况才发生了根本变化，人们开始认识到DFT运算的一些内在规律，从而很快地发展和完善了一套高速有效的运算方法——快速付里变换(FFT)算法。FFT的出现，使DFT的运算大大简化，运算时间缩短一～二个数量级，使DFT的运算在实际中得到广泛应用。

# 直接计算量

- $4n^2$实数相乘和$n(4n-2)$次实数相加;
- 计算n=10点的DFT，需要100次复数相乘，而n=1024点时，需要1048576（一百多万）次复数乘法
- 反变换IDFT与DFT的运算结构相同，只是多乘一个常数1/n，所以二者的计算量相同。

# 基本思想

- 系数 $\omega_n^{jk} = e^{-i\frac{2\pi}{n}jk}$ 是一个周期函数
- 长度为n点的大点数的DFT运算依次分解为若干个小点数的DFT。因为DFT的计算量正比于$n^2$，n小，计算量也就小。

$n=2^m$，m：正整数

首先将序列x(j)分解为两组，一组为偶数项，一组为奇数项，

$$\begin{cases} x(2r) = x_1(r) \\ x(2r+1) = x_2(r) \end{cases}$$

r=0,1，…，n/2-1

$$X(k) = F_n(x) = \sum_{j=0}^{n-1} x(j)\overline{\omega}_n^{jk} = \sum_{r=0}^{n/2-1} x(2r)\overline{\omega}_n^{2rk} + \sum_{r=0}^{n/2-1} x(2r+1)\overline{\omega}_n^{(2r+1)k}$$

$$= \sum_{r=0}^{n/2-1} x(2r)\overline{\omega}_n^{2rk} + \overline{\omega}_n^{k} \sum_{r=0}^{n/2-1} x(2r+1)\overline{\omega}_n^{2rk}$$

$$\overline{\omega}_n^{2j} = e^{-i\frac{2\pi}{n}2j} = e^{-i\frac{2\pi}{n/2}j} = \overline{\omega}_{n/2}^{j}$$

$$X(k) = \sum_{r=0}^{n/2-1} x(2r)\overline{\omega}_{\frac{n}{2}}^{rk} + \overline{\omega}_n^{k} \sum_{r=0}^{n/2-1} x(2r+1)\overline{\omega}_{\frac{n}{2}}^{rk}$$

$$= G(k) + \overline{\omega}_n^{k} H(k)$$

$$G(k) = \sum_{r=0}^{n/2-1} x(2r)\overline{\omega}_{\frac{n}{2}}^{rk}$$

$$H(k) = \sum_{r=0}^{n/2-1} x(2r+1)\overline{\omega}_{\frac{n}{2}}^{rk}$$

$$\overline{\omega}_{n/2}^{r(n/2+k)} = \overline{\omega}_{n/2}^{rk} \qquad \overline{\omega}_{n}^{(k+\frac{n}{2})} = -\overline{\omega}_{n}^{k}$$

$$X(k+\frac{n}{2}) = G(k) - \overline{\omega}_{n}^{k} H(k), \qquad k = 0,1,\cdots\frac{n}{2}-1$$

可见，一个n点的DFT被分解为两个n/2点的DFT，这两个n/2点的DFT再合成为一个n点DFT.

$$X(k) = G(k) + \overline{\omega}_n^k H(k), \qquad k = 0,1,\cdots,\frac{n}{2}-1$$

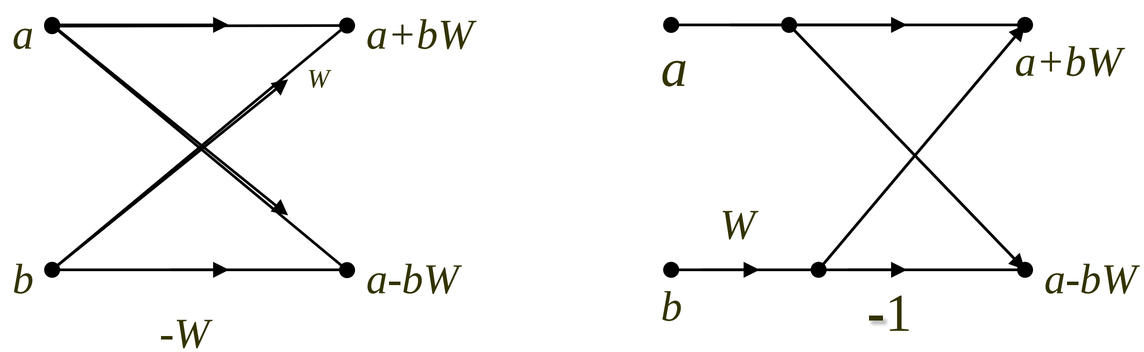
$$X(k+\frac{n}{2}) = G(k) - \overline{\omega}_n^k H(k), \qquad k = 0,1,\cdots\frac{n}{2}-1$$

依此类推，G(k)和H(k)可以继续分下去，这种算法是在输入序列分成越来越小的子序列上执行DFT运算，最后再合成为*n*点的DFT。

# 蝶形信号流图

将$G(k)$和$H(k)$ 合成$X(k)$运算可归结为：

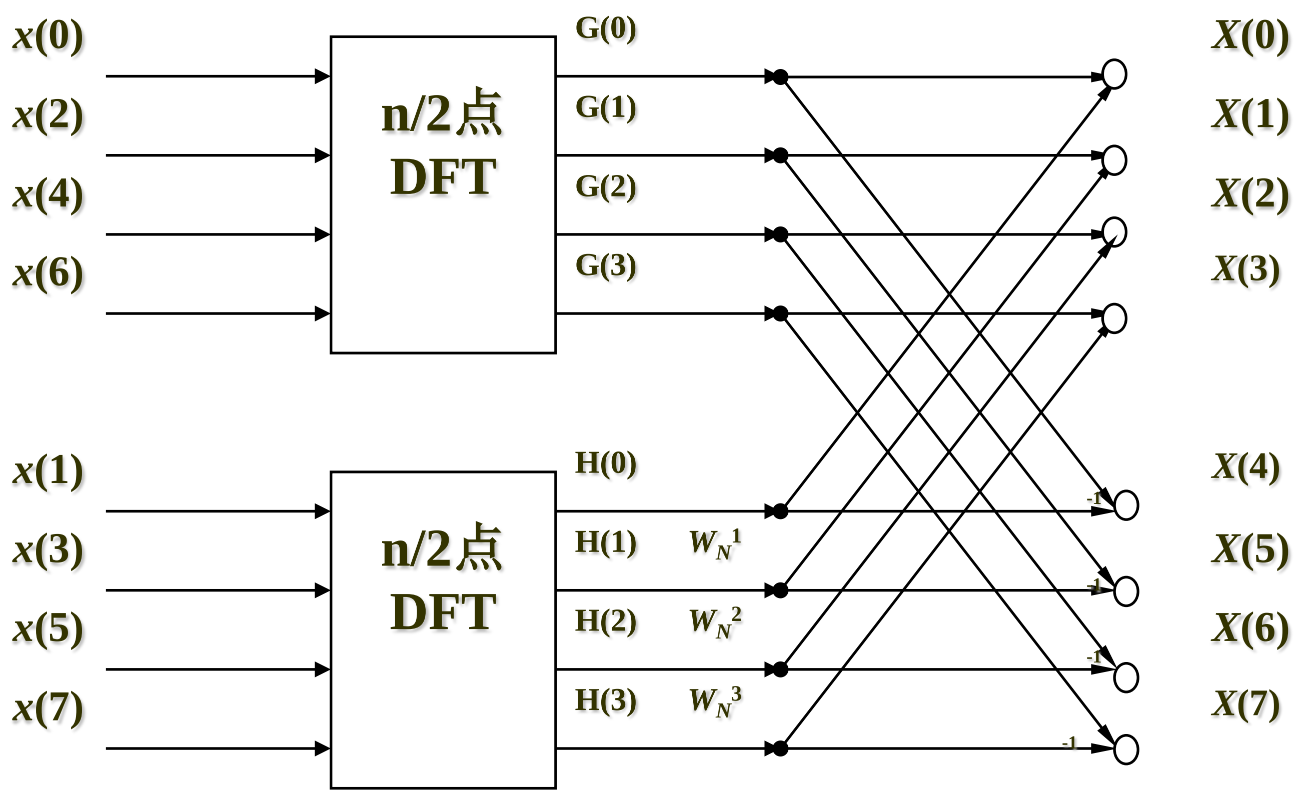
$$\begin{cases} a - bW \\ a + bW \end{cases}$$




(a) 蝶形运算的简化 (b)

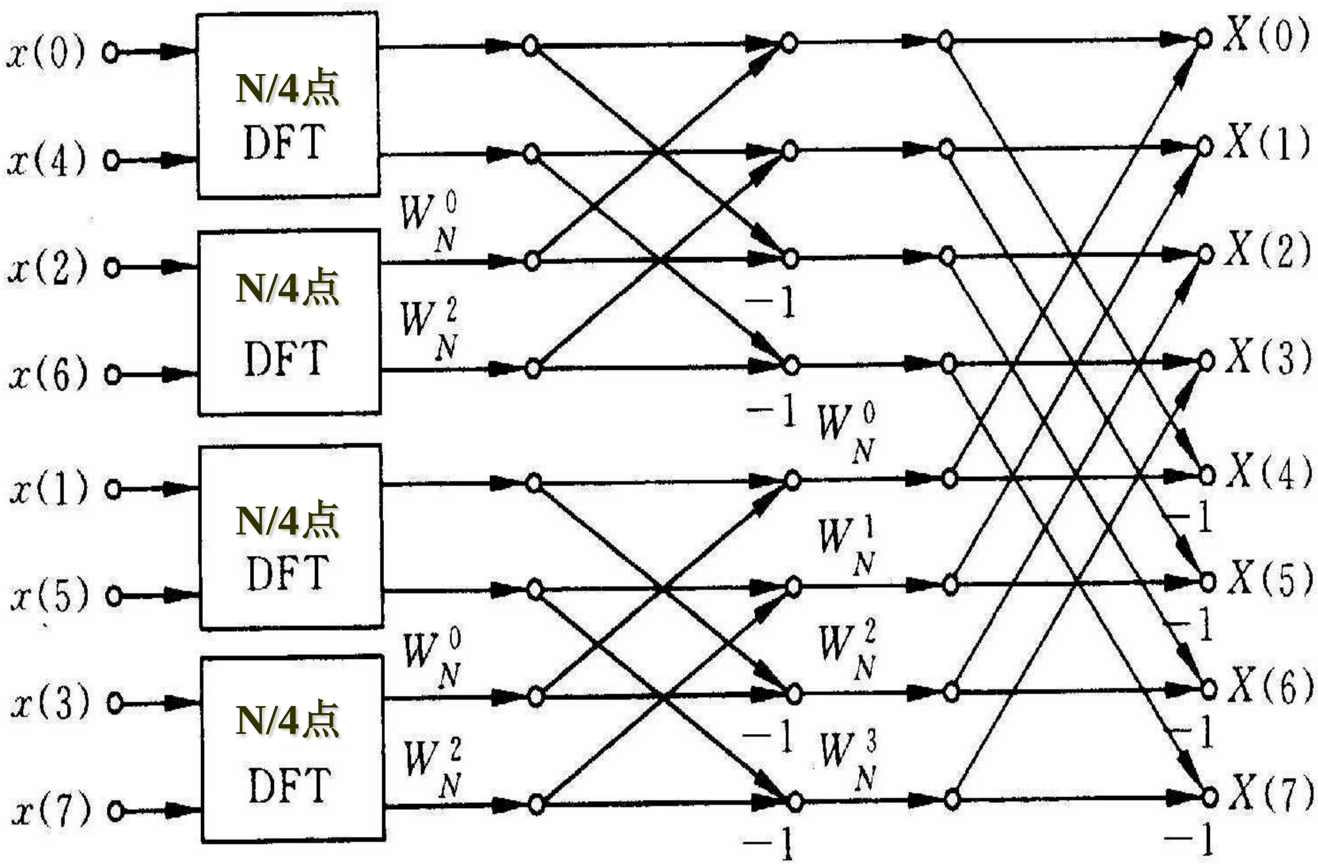
图 (a)为实现这一运算的一般方法，它需要两次乘法、两次加减法。考虑到-*bW*和*bW*两个乘法仅相差一负号，可将图 (a)简化成图 2.7(b)，此时仅需一次乘法、两次加减法。图 (b)的运算结构像一蝴蝶通常称作蝶形运算结构简称蝶形结，采用这种表示法，就可以将以上所讨论的分解过程用流图表示。

n=$2^3$=8 的例子。

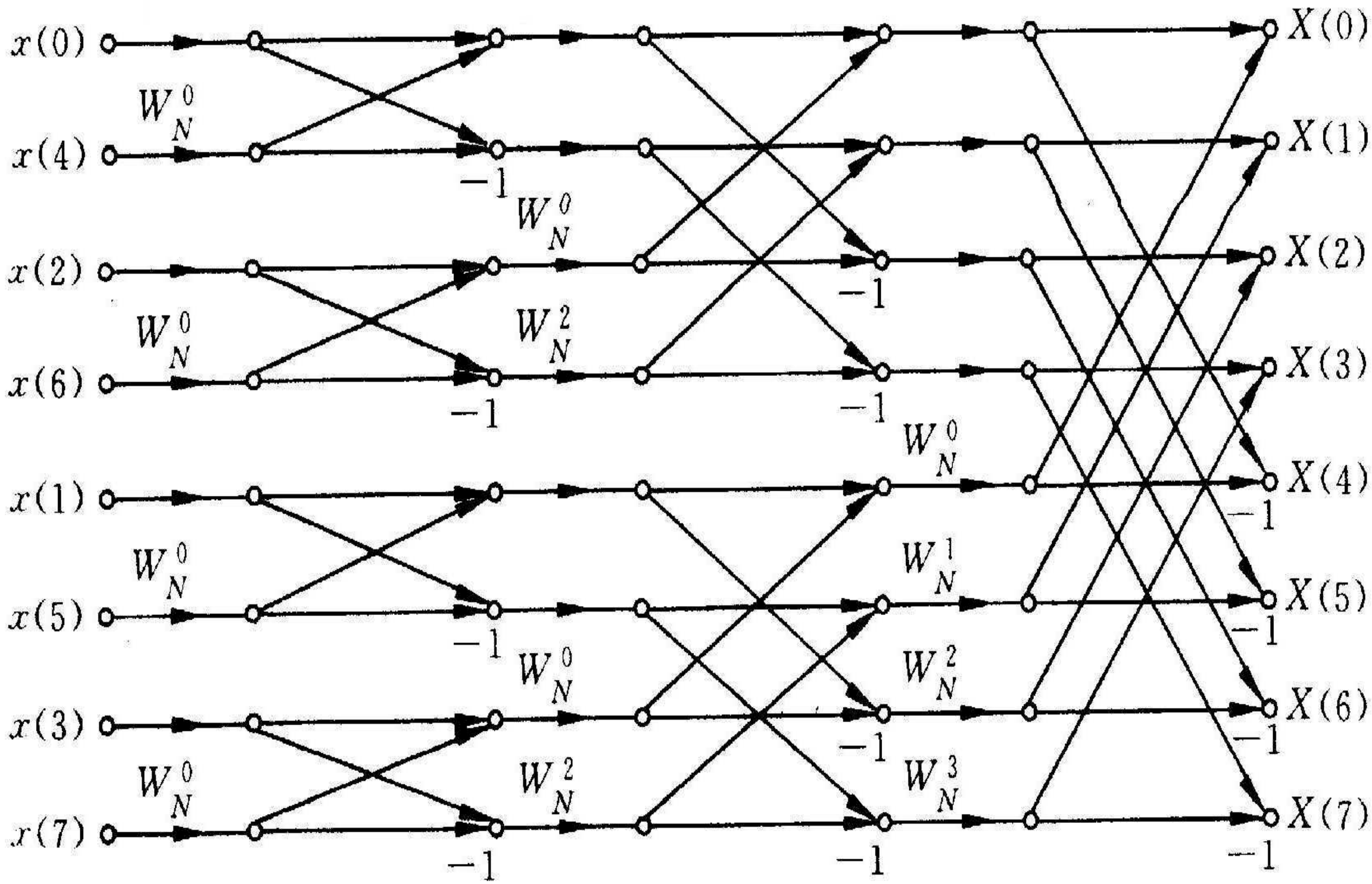


x(0)
x(2)
x(4)
x(6)
x(1)
x(3)
x(5)
x(7)
n/2点
DFT
n/2点
DFT
G(0)
G(1)
G(2)
G(3)
H(0)
H(1)
H(2)
H(3)
$W_N^1$
$W_N^2$
$W_N^3$
-1
-1
-1
-1
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)

按照这个办法，继续把n/2用 2 除，由于$n=2^m$，仍然是偶数，可以被 2 整除，因此可以对两个n/2点的DFT再分别作进一步的分解。即对$\{G(k)\}$和$\{H(k)\}$的计算，又可以分别通过计算两个长度为n/4=2点的DFT，进一步节省计算量。这样，一个 8 点的DFT就可以分解为四个 2 点的DFT。



x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
N/4点 DFT
N/4点 DFT
N/4点 DFT
N/4点 DFT
$W_N^0$
$W_N^2$
$W_N^0$
$W_N^2$
−1
−1
−1
−1
$W_N^0$
$W_N^1$
$W_N^2$
$W_N^3$
−1
−1
−1
−1
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)



x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
W N 0
W N 2
W N 1
W N 3
−1
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)

# 计算量

对于$n=2^m$，总是可以通过m次分解最后成为2点的DFT运算。这样构成从x(n)到X(k)的m级运算过程。

每一级运算都由n/2个蝶形运算构成。因此每一级运算都需要n/2次复乘和n次复加。

复乘 $\frac{n}{2}\bullet m=\frac{n}{2}\log_2 n$　复加 $n\bullet m=n\log_2 n$

而直接运算时则与$n^2$成正比。

例 n=2048，$n^2=4194304$，$(n/2)\log_2 n=11264$，$n^2/[(n/2)\log_2 n]=392.4$。FFT显然要比直接法快得多。