Digital Signal Processing

Class 19
04/01/2025

ENGR 71

- Class Overview
  - Circulant Matrix
  - Fast Fourier Transform
- Assignments
  - Reading:

    Chapter 8: The Fast Fourier Transform
  - Problems:

    Chapter 7: 7.8, 7.9, 7.11(b), 7.14, 7.18, 7.25

    Pick one symmetry property from Table 7.1 and one property from Table 7.2 to prove. (Next class, say which ones.)

    Due: Friday, April 4

Project Ideas

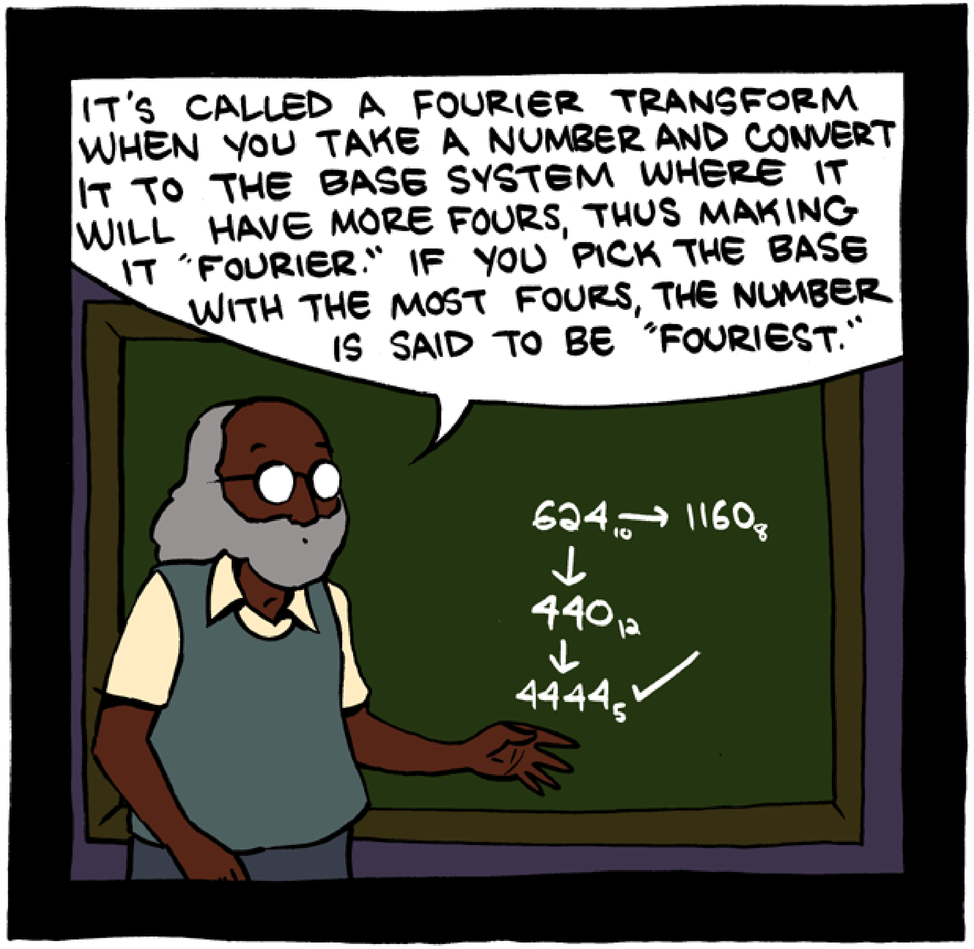
- Project Ideas
  - Speech recognition (more complex than Lab 2)
    - Classifier for multiple words
      - I can provide a dataset with multiple instances of several different words
  - Musical instrument tone recognition
    - Using recordings of musical instruments, determine note being played
    - Determine if instrument is in tune, sharp, or flat.
  - Identification of musical instruments
    - I have a dataset of recordings for several different instruments
  - Identification of music genre
    - From frequency characteristics, can you determine a type of music
      - Classical, rock, etc.

Project ideas

- Filtering
  - Filtering to isolate sounds
  - Equalizer
- Noise reduction
- Audio effects processing
  - Reverb, echo, distortion
- Echo cancellation
- Several possibilities if you are interested in 2-D signal processing for image data

- Hardware projects
  - Link to site with collection of Arduino-based projects
- Theoretical research topics are also welcome
  - Paper on some interesting topic

Fourier Transform



Teaching math was way more fun after tenure.

Fourier Series

- **Fourier series for periodic signals:**

$$x(t) = x(t + T_0) \qquad f_0 = \frac{1}{T_0} \qquad \omega_0 = 2\pi f_0 = \frac{2\pi}{T_0}$$

$$x(t) = \sum_{k=-\infty}^{+\infty} X_k e^{jk\omega_0 t} \qquad \text{(Synthesis Eq.)}$$

$$X_k = \frac{1}{T_0}\int_{T_0} x(t) e^{-jk\omega_0 t} dt \qquad \text{(Analysis Eq.)}$$

Fourier Transform

- **Fourier transform aperiodic signals:**

$$X(\Omega) = \mathscr{F}[x(t)] = \int_{-\infty}^{+\infty} x(t) e^{-j\Omega t} dt \qquad \text{(Analysis Equation)}$$

$$x(t) = \mathscr{F}^{-1}[X(\Omega)] = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\Omega) e^{+j\Omega t} d\omega \qquad \text{(Synthesis Equation)}$$

Time and frequency are continous variables

$-\infty < t < \infty$

$-\infty < \Omega < \infty$

Using Ω to distinguish it from dicrete time case where frequency is between -π and π

Discrete-Time Fourier Transform

- **Discrete-time Fourier transform**

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \quad -\pi \le \omega < \pi \quad \text{(Analysis equation)}$$

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}d\omega \qquad \text{(Synthesis equation)}$$

Time, labeled by the integer index n, is discrete $(t = nT_s)$

$-\infty < n < \infty$

$-\pi < \omega < \pi$

Limits on ω are imposed by the Nyquist condition

π represents maximum positive frequency $f_{Nyquist} = \dfrac{f_s}{2} = \dfrac{1}{2T_s}$

(where T_s is the sampling interval or alternatively, f_s is the sampling frequency)

Discrete Fourier Series

- **Discrete Fourier series for a periodic sequence with period *N***

$$x_p[n+mN] = x_p[n] \quad m = \dots, -1, 0, 1, \dots$$

$$x[n] = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N}$$

where

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}$$

Discrete Fourier Transform

- **Discrete Fourier Transform**

Discrete Fourier Transform (DFT) Analysis Equation

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \quad , \quad k = 0, 1, 2, \ldots, N\text{-}1$$

Inverse Discrete Fourier Transform (IDFT) Synthesis Equation

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N} \quad , \quad n = 0, 1, 2, \ldots, N\text{-}1$$

Circular Convolution

- Circular Convolution

$$x_3 = x_1 \odot x_2 = \sum_{n=0}^{N-1} x_1[n] x_2[((m-n))_N]$$

Circular Convolution

– Example with N=4 length sequence

$$x_3[0]=\sum_{n=0}^{3}x_1[n]x_2[((0-n))_4]=x_1[0]x_2[((0-0))_4]+x_1[1]x_2[((0-1))_4]+x_1[2]x_2[((0-2))_4]+x_1[3]x_2[((0-3))_4]$$

$$x_3[0]=x_1[0]x_2[0]+x_1[1]x_2[3]+x_1[2]x_2[2]+x_1[2]x_2[1]$$

$$x_3[1]=\sum_{n=0}^{3}x_1[n]x_2[((1-n))_4]=x_1[0]x_2[((1-0))_4]+x_1[1]x_2[((1-1))_4]+x_1[2]x_2[((1-2))_4]+x_1[3]x_2[((1-3))_4]$$

$$x_3[1]=x_1[0]x_2[1]+x_1[1]x_2[0]+x_1[2]x_2[3]+x_1[3]x_2[2]$$

$$x_3[2]=\sum_{n=0}^{3}x_1[n]x_2[((2-n))_4]=x_1[0]x_2[((2-0))_4]+x_1[1]x_2[((2-1))_4]+x_1[2]x_2[((2-2))_4]+x_1[3]x_2[((2-3))_4]$$

$$x_3[2]=x_1[0]x_2[2]+x_1[1]x_2[1]+x_1[2]x_2[0]+x_1[3]x_2[3]$$

$$x_3[3]=\sum_{n=0}^{3}x_1[n]x_2[((3-n))_4]=x_1[0]x_2[((3-0))_4]+x_1[1]x_2[((3-1))_4]+x_1[2]x_2[((3-2))_4]+x_1[3]x_2[((3-3))_4]$$

$$x_3[3]=x_1[0]x_2[3]+x_1[1]x_2[2]+x_1[2]x_2[1]+x_1[3]x_2[0]$$

Circular Convolution

Re-written as:

$$x_3[0] = x_2[0]x_1[0] + x_2[3]x_1[1] + x_2[2]x_1[2] + x_2[1]x_1[3]$$
$$x_3[1] = x_2[1]x_1[0] + x_2[0]x_1[1] + x_2[3]x_1[2] + x_2[2]x_1[3]$$
$$x_3[2] = x_2[2]x_1[0] + x_2[1]x_1[1] + x_2[0]x_1[2] + x_2[3]x_1[3]$$
$$x_3[3] = x_2[3]x_1[0] + x_2[2]x_1[1] + x_2[1]x_1[2] + x_2[0]x_1[3]$$

This can be written in matrix form as:

$$\begin{pmatrix} x_3[0] \\ x_3[1] \\ x_3[2] \\ x_3[3] \end{pmatrix} = \begin{pmatrix} x_2[0] & x_2[3] & x_2[2] & x_2[1] \\ x_2[1] & x_2[0] & x_2[3] & x_2[2] \\ x_2[2] & x_2[1] & x_2[0] & x_2[3] \\ x_2[3] & x_2[2] & x_2[1] & x_2[0] \end{pmatrix} \begin{pmatrix} x_1[0] \\ x_1[1] \\ x_1[2] \\ x_1[3] \end{pmatrix} \quad \text{or} \quad \mathbf{x_3} = \mathbf{C}_4^{x_2}\ \mathbf{x_1}$$

where $\mathbf{x_1}$ and $\mathbf{x_3}$ are length 4 sequences

and $\mathbf{C}_4^{x_2}$ is the 4×4 circulant matrix formed from the elements of sequence $x_2[n]$

Circulant Matrix

Notice that the circulant matrix consists of the columns of sequence $x_2(n)$ cyclically permuted.
The first column is the sequence;
column 2 has the last element of column 1 first, followed by the remaining elements of column 1;
column 3 has the last element of column 2 first, followed by the remaining elements of column 2;
column 4 has the last element of column 3 first, followed by the remaining elements of column 3;

This definition of the circulant matrix can be generalized for any N-length sequence.

Circulant Matrix

– Example 7.2.1 from book:

$$x_1[n] = \{2,1,2,1\}; \qquad x_2[n] = \{1,2,3,4\}$$

Find the circular convolution of $x_1[n]$ and $x_2[n]$: $\quad x_3[n] = x_1[n] \odot x_2[n]$

In matrix form:

$$\begin{pmatrix} x_3[0] \\ x_3[1] \\ x_3[2] \\ x_3[3] \end{pmatrix} = \begin{pmatrix} 1 & 4 & 3 & 2 \\ 2 & 1 & 4 & 3 \\ 3 & 2 & 1 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 14 \\ 16 \\ 14 \\ 16 \end{pmatrix}$$

```
> x1 = [2,1,2,1];
> x2 = [1,2,3,4];
> c = [1 4 3 2; 2 1 4 3; 3 2 1 4; 4 3 2 1]
> x3 = c*x1'
```

In Matlab, you can also use the command `toeplitz`
to construct a ciculant matrix of any size based on a sequence $x[n]$:

```
c = toeplitz([x(1) fliplr(x(2:end))], x)’
```

Circulant Matrix

- Circulant Matrix
  - The circulant matrix has a lot of interesting properties
    - The eigenvalues are the elements of the DFT of the sequence in the first column
    - The matrix formed, using the eigenvectors as columns, is the matrix with elements W_N^{kn} which is used to find the DFT: $\mathbf{X} = \mathbf{W}_N \mathbf{x}$

    - This can be used to show that circular convolution in the time-domain is the product of the DFT's of the convolved sequences

      If, $y[n] = x_1[n] \odot x_2[n]$, then $Y[k] = X_1[k] \cdot X_2[k]$

$$\left(W_N \equiv e^{-j2\pi/N}\right)$$

Circulant Matrix

Start with the expression for circular convolution in the time-domain: $y[n]=x_1[n]\odot x_2[n]$

In matrix notation: $\vec{\mathbf{y}}=\mathbf{C}_N^{x_2}\,\vec{\mathbf{x}}_1$ where $\mathbf{C}_N^{x_2}$ is the circulant matrix formed using sequence $x_2[n]$,

and $\vec{\mathbf{x}}_1$ is a sequence (represented as a column vector).

$\mathbf{C}_N^{x_2}$ is diagnoalized by $\mathbf{W}_N$ with eigenvalues: $X_2[k]$ which is the DFT of $x_2[n]$, so $\mathbf{C}_N^{x_2}=\mathbf{W}_N^{-1}diag\left(\vec{\mathbf{X}}_2\right)\mathbf{W}_N$.

$\left(diag\left(\vec{\mathbf{X}}_2\right)\right.$ **is a diagonal matrix** with $X_2[k]$ as the diagonal elements.$\left.\right)$

Substituting this in the the matrix equation for convolution:

$$\vec{\mathbf{y}}=\left(\mathbf{W}_N^{-1}diag\left(\vec{\mathbf{X}}_2\right)\mathbf{W}_N\right)\vec{\mathbf{x}}_1 \quad\Rightarrow\quad \mathbf{W}_N\vec{\mathbf{y}}=\left(\mathbf{W}_N\mathbf{W}_N^{-1}\right)diag\left(\vec{\mathbf{X}}_2\right)\left(\mathbf{W}_N\vec{\mathbf{x}}_1\right) \quad\Rightarrow\quad \mathbf{W}_N\vec{\mathbf{y}}=diag\left(\vec{\mathbf{X}}_2\right)\left(\mathbf{W}_N\vec{\mathbf{x}}_1\right)$$

The DFT's of $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{y}}$ are: $\vec{\mathbf{X}}_1=\mathbf{W}_N\vec{\mathbf{x}}_1$ and $\vec{\mathbf{Y}}=\mathbf{W}_N\vec{\mathbf{y}}$, showing that $\vec{\mathbf{Y}}=\vec{\mathbf{X}}_2^T\vec{\mathbf{X}}_1$.

Note that this is the element-by-element product: $\vec{\mathbf{X}}_2^T\vec{\mathbf{X}}_1$: $Y[k]=X_2[k]X_1[k]$ for each $k=0,1,\ldots,N-1$,

since in the matrix equation, $diag\left(\vec{\mathbf{X}}_2\right)$ is a diagonal matrix.

$$\boxed{\text{Therefore for } y[n]=x_1[n]\odot x_2[n],\quad Y[k]=X_1[k]\cdot X_2[k]}$$

Fast Fourier Transform

- Fast Fourier Transform
  - Cooley and Tukey (1965)
  - Actually, Gauss knew about this algorithm in 1805

Fast Fourier Transform

- Fast Fourier Transform
  - Recall equations for DFT and IDFT

Discrete Fourier Transform (DFT)

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-j2\pi kn/N}\ ,\qquad k=0,1,2,\ldots,N\text{-}1$$

Inverse Discrete Fourier Transform (IDFT)

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)e^{j2\pi kn/N}\ ,\quad n=0,1,2,\ldots,N\text{-}1$$

Fast Fourier Transform

- Define W_N as: $W_N = e^{-j2\pi/N}$ which is the N'th primative root of unity.
  - Then:

$$W_N^{kn} = e^{-j2\pi kn/N}, \quad W_N^{-nk} = e^{j2\pi nk/N}$$

- The DFT and IDFT in terms of W_N are:

Discrete Fourier Transform (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0, 1, 2, \ldots, N\text{-}1$$

Inverse Discrete Fourier Transform (IDFT)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad n = 0, 1, 2, \ldots, N\text{-}1$$

Fast Fourier Transform

- Algorithms for DFT and IDFT can be made more efficient by exploiting symmetry and periodicity properties of W_N
  - Symmetry property: $W_N^{k+N/2} = -W_N^k$

$$W_N^{k+N/2} = e^{-j2\pi(k+N/2)n/N}$$

$$= e^{-j2\pi kn/N} e^{-j2\pi(N/2)n/N}$$

$$= e^{-j2\pi kn/N} e^{-j\pi n}$$

$$= -e^{-j2\pi kn/N}$$

$$\boxed{W_N^{k+N/2} = -W_N^k}$$

Fast Fourier Transform

- Algorithms for DFT and IDFT can be made more efficient by exploiting symmetry and periodicity properties of W_N
  - Periodicity property: $W_N^{(k+N)n} = W_N^{k(n+N)} = W_N^{kn}$

$$
\begin{aligned}
W_N^{(k+N)n} &= e^{-j2\pi(k+N)n/N} \\
&= e^{-j2\pi kn/N} e^{-j2\pi(N)n/N} \\
&= e^{-j2\pi kn/N} e^{-j2\pi n} \\
&= e^{-j2\pi kn/N}
\end{aligned}
$$

$$
\begin{aligned}
W_N^{k(n+N)n} &= e^{-j2\pi k(n+N)/N} \\
&= e^{-j2\pi kn/N} e^{-j2\pi k(N)/N} \\
&= e^{-j2\pi kn/N} e^{-j2\pi k} \\
&= e^{-j2\pi kn/N}
\end{aligned}
$$

$$\boxed{W_N^{(k+N)n} = W_N^{k(n+N)} = W_N^{kn}}$$

Fast Fourier Transform

- Algorithms for DFT and IDFT can be made more efficient by exploiting symmetry and periodicity properties of W_N
  - Complex conjugate symmetry: $W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^*$

$$
\begin{aligned}
W_N^{k(N-n)} &= e^{-j2\pi k(N-n)/N} \\
&= e^{-j2\pi kN/N} e^{+j2\pi kn/N} \\
&= e^{-j2\pi k} e^{+j2\pi n/N} \\
&= e^{+j2\pi kn/N} \\
&= W_N^{-kn} = \left(W_N^{kn}\right)^*
\end{aligned}
$$

$$\boxed{W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^*}$$

Fast Fourier Transform

– Other relationships for W_N

- If N can be factored into a product of integers: $N = LM$

$$W_N^{mqL} = e^{-j2\pi mqL/N} = e^{-j2\pi mq/(N/L)}$$

$$W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}$$

$$W_N^{Mpl} = e^{-j2\pi plM/N} = e^{-j2\pi pl/(N/M)}$$

$$W_N^{Mpl} = W_{N/M}^{pl} = W_L^{pl}$$

For N factored as $N = ML$

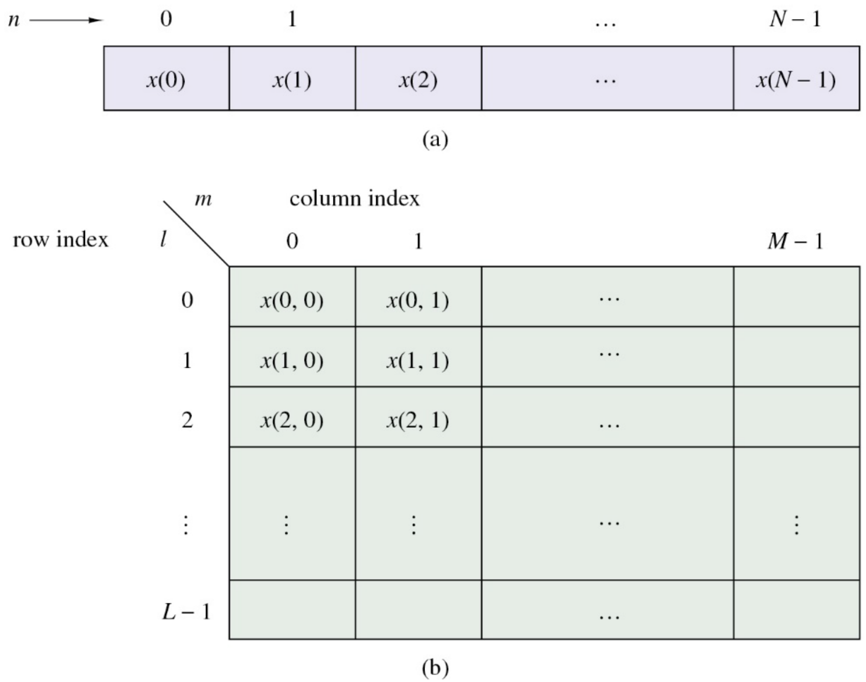
$$W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}$$

$$W_N^{Mpl} = W_{N/M}^{pl} = W_L^{mq}$$

for integers: $m,\ q,\ p,$ and l

Fast Fourier Transform

- Divide and Conquer Algorithms
  - Consider the case where $N=ML$
  - Represent the input sequence $x[n]$ and output DFT $X[k]$ as 2-D arrays rather than linear sequences



Fast Fourier Transform

Map the sequence into $x(n) \rightarrow x(l,m): \quad n = l + mL$

Column-wise $\quad n = l + mL$

$l \backslash m$	0	1	2		$M-1$
0	$x(0)$	$x(L)$	$x(2L)$	$\cdots$	$x((M-1)L)$
1	$x(1)$	$x(L+1)$	$x(2L+1)$	$\cdots$	$x((M-1)L+1)$
2	$x(2)$	$x(L+2)$	$x(2L+2)$	$\cdots$	$x((M-1)L+2)$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$L-1$	$x(L-1)$	$x(2L-1)$	$x(3L-1)$	$\cdots$	$x(LM-1)$

$x(0,0)$ $x(2,0)$ $x(2,1)$

(b)

Map the DFT into $X(k) \rightarrow X(p,q): \quad k = Mp + q$

Row-wise $\quad k = Mp + q$

$p \backslash q$	0	1	2		$M-1$
0	$X(0)$	$X(1)$	$X(2)$	$\cdots$	$X(M\text{-}1)$
1	$X(M)$	$X(M+1)$	$X(M+2)$	$\cdots$	$X(2M\text{-}1)$
2	$X(2M)$	$X(2M+1)$	$X(2M+2)$	$\cdots$	$X(3M\text{-}1)$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$L-1$	$X((L\text{-}1)M)$	$X((L\text{-}1)M+1)$	$X((L\text{-}1)M+2)$	$\cdots$	$X(LM\text{-}1)$

$X(0,2)$ $X(0,0)$ $X(2,1)$

(a)

Fast Fourier Transform

Transform with 1-D n , k:

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{kn}$$

Transform with 2-D indices: $n=l+mL$; $k=Mq+p$

$$X(p,q)=\sum_{m=0}^{M-1}\sum_{l=0}^{L-1}x(l,m)W_N^{(Mp+q)(mL+l)}$$

$$W_N^{(Mp+q)(mL+l)}=W_N^{MLmp+mqL+Mpl+lq}=W_N^{MLmp}W_N^{mqL}W_N^{Mpl}W_N^{lq}$$

Take advantage of symmetry properties:

$$W_N^{MLmp}=W_N^{Nmp}=1;\quad W_N^{mLq}=W_{N/L}^{mq}=W_M^{mq};\quad W_N^{Mpl}=W_{N/M}^{pl}=W_L^{pl}$$

Fast Fourier Transform

Using $W_N^{MLmp} = 1; \quad W_N^{mLq} = W_M^{mq}; \quad W_N^{Mpl} = W_L^{pl}$

$$X(p,q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)} = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_M^{mq} W_L^{pl} W_N^{lq}$$

Rearranging:

$$X(p,q) = \sum_{l=0}^{L-1} W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq} \right] W_L^{pl}$$

Fast Fourier Transform

(Repeating the last equation from previous slide)

$$X(p,q) = \sum_{l=0}^{L-1} W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m) W_M^{mq} \right] W_L^{pl}$$

The innermost term is an M-point DFT over index m

$$F(l,q) = \sum_{m=0}^{M-1} x(l,m) W_M^{mq}, \quad 0 \le q \le M-1 \text{ for each row } l$$

The remaining sum is :

$$X(p,q) = \sum_{l=0}^{L-1} \left[W_N^{lq} F(l,q) \right] W_L^{pl}$$

is an $L-$point DFT over new array: $G(l,q) = W_N^{lq} F(l,q)$; $0 \le l \le L$ -1; $0 \le q \le M$ -1

Fast Fourier Transform

The final DFT is

$$X(p,q)=\sum_{l=0}^{L-1}G(l,q)W_L^{pl}$$

The linear index for the DFT is: $k=qL+p$

Fast Fourier Transform

- Have we done any good here?

- Remember, an N-point DFT requires N^2 multiplication and $N(N-1)$ additions

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{kn}$$

- Step 1 of the new algorithm finds L, m-point DFT's

$$F(l,q)=\sum_{m=0}^{M-1}x(l,m)W_M^{mq}$$

Number of multiplications: LM^2

Number of additions: $LM(M-1)$

Fast Fourier Transform

- Step 2: (getting an $L \times M$ array)

  $G(l,q) = W_N^{lq} F(l,q)$; $0 \le l \le L$ -1; $0 \le q \le M$ -1

  Number of multiplications: LM

  Number of additions: none

- Step 3 (Finds M, L-point DFT's)

  $$X(p,q) = \sum_{l=0}^{L-1} G(l,q) W_L^{pl}$$

  Number of multiplications: ML^2

  Number of additions: $ML(L-1)$

Fast Fourier Transform

- Total number of multiplications and additions:

Number of multiplications: $LM^2 + LM + ML^2 = ML\left(M + L + 1\right) = N\left(M + L + 1\right)$

Number of additions: $LM\left(M - 1\right) + 0 + ML\left(L - 1\right) = ML\left(M + L - 2\right) = N\left(M + L - 2\right)$

Comparison :

Multiplications: $N^2 \rightarrow N\left(M + L + 1\right)$

Additions: $\mathrm{N}\left(N - 1\right) \rightarrow N\left(M + L + 2\right)$

Comparison : Example for $N = 1000,\ M = 500,\ L = 2$

Multiplications: $N^2 \rightarrow N\left(M + L + 1\right)$: $1{,}000{,}000 \rightarrow 503{,}000$

Additions: $\mathrm{N}\left(N - 1\right) \rightarrow N\left(M + L - 2\right)$: $999{,}000 \rightarrow 500{,}000$

Fast Fourier Transform

- Previous algorithm shows how divide and conquer works, but it is not how the usual FFT works
  - Rather than a single factorization of N, factorize it many times and repeat the procedure.
  - Factors of: $N = r_1 r_2 \cdots r_\nu$
  - Radix algorithms for when N is a power of some value, $N = r^\nu$
  - Most common one is when N is a power of 2.
    - You can always pad a sequence to make this the case
      $N = 2^\nu$
    - Divide and conquer by recursively splitting sequence into 2 equal parts.
    - This will reduce computational complexity from $N^2 \rightarrow N \log_2 N$

Fast Fourier Transform

- Radix-2 FFT (decimation in time) Algorithm
  - For this algorithm, number of samples is a power of 2: $N = 2^{\nu}$
    - If it isn't you could pad it.
  - Start by dividing the sequence in 2: M=N/2; L=2

Separate sum into separate sums of even and odd elements of original sequence:

$$X(k) = \sum_{n=even} x(n) W_N^{kn} + \sum_{n=odd} x(n) W_N^{kn}$$

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_N^{k(2r)} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{k(2r+1)}$$

$$\text{even indices}: \; n = 2r$$

$$\text{odd indices}: \; n = 2r+1$$

$$r = 0, 1, \ldots, \frac{N}{2} - 1$$

Fast Fourier Transform

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)W_N^{k(2r)} + \sum_{r=0}^{N/2-1} x(2r+1)W_N^{k(2r+1)}$$

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)W_N^{k(2r)} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)W_N^{k(2r)}$$

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)W_{N/2}^{kr} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)W_{N/2}^{kr}$$

N/2 DFT of even samples

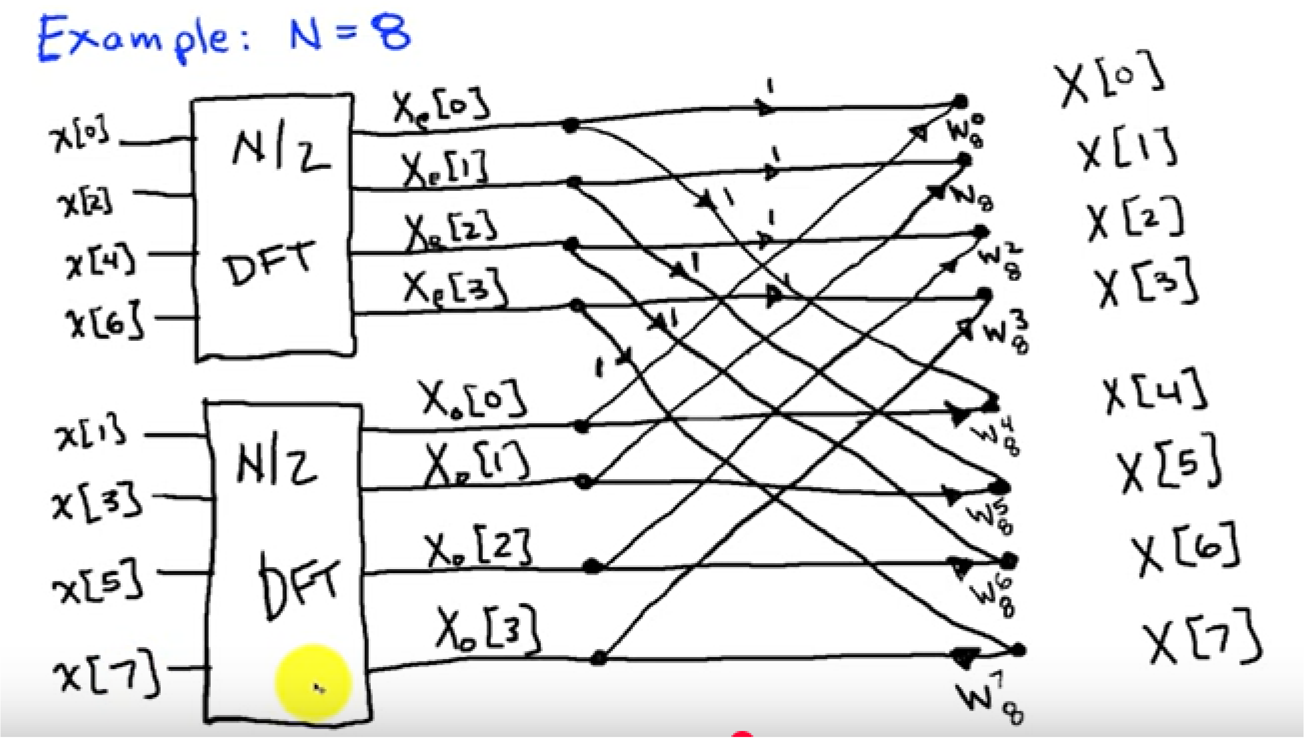
N/2 DFT of odd samples

$$\boxed{X(k) = X_e(k) + W_N^k X_o(k)}$$

Sum of 2 *N*/2 point DFT's

Fast Fourier Transform

$$\boxed{X(k) = X_e(k) + W_N^k X_o(k)}$$



Operation count:

$$2 \times \left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N \quad \text{multiplies}$$

Original without splitting would be N^2 multiplies

Fast Fourier Transform

You can keep splitting each of the N/2 sub-DFT's:

$$\text{Split } \frac{N}{2} \text{ DFT's } \rightarrow 2\times\frac{N}{4}$$

$$\text{Split } \frac{N}{4} \text{ DFT's } \rightarrow 2\times\frac{N}{8}$$

etc.

How many times can you split for $N = 2^p$?

$$\frac{N}{2},\frac{N}{4},\frac{N}{8},\cdots\frac{N}{2^{p-1}},\frac{N}{2^p}$$

$p = \log_2 N$ splits

Fast Fourier Transform

$$\frac{N}{2},\frac{N}{4},\frac{N}{8},\cdots\frac{N}{2^{p-1}},\frac{N}{2^p}$$

$p=\log_2 N$ splits

Operation count for multiplies:

$$1:\frac{N}{2}DFT's:\ 2\left(\frac{N}{2}\right)^2+N=\frac{N^2}{2}+N=\frac{N^2}{2^1}+N$$

$$2:\frac{N}{4}DFT's:\ 2\left[2\left(\frac{N}{4}\right)^2+\frac{N}{2}\right]+N=\frac{N^2}{4}+2N=\frac{N^2}{2^2}+2N$$

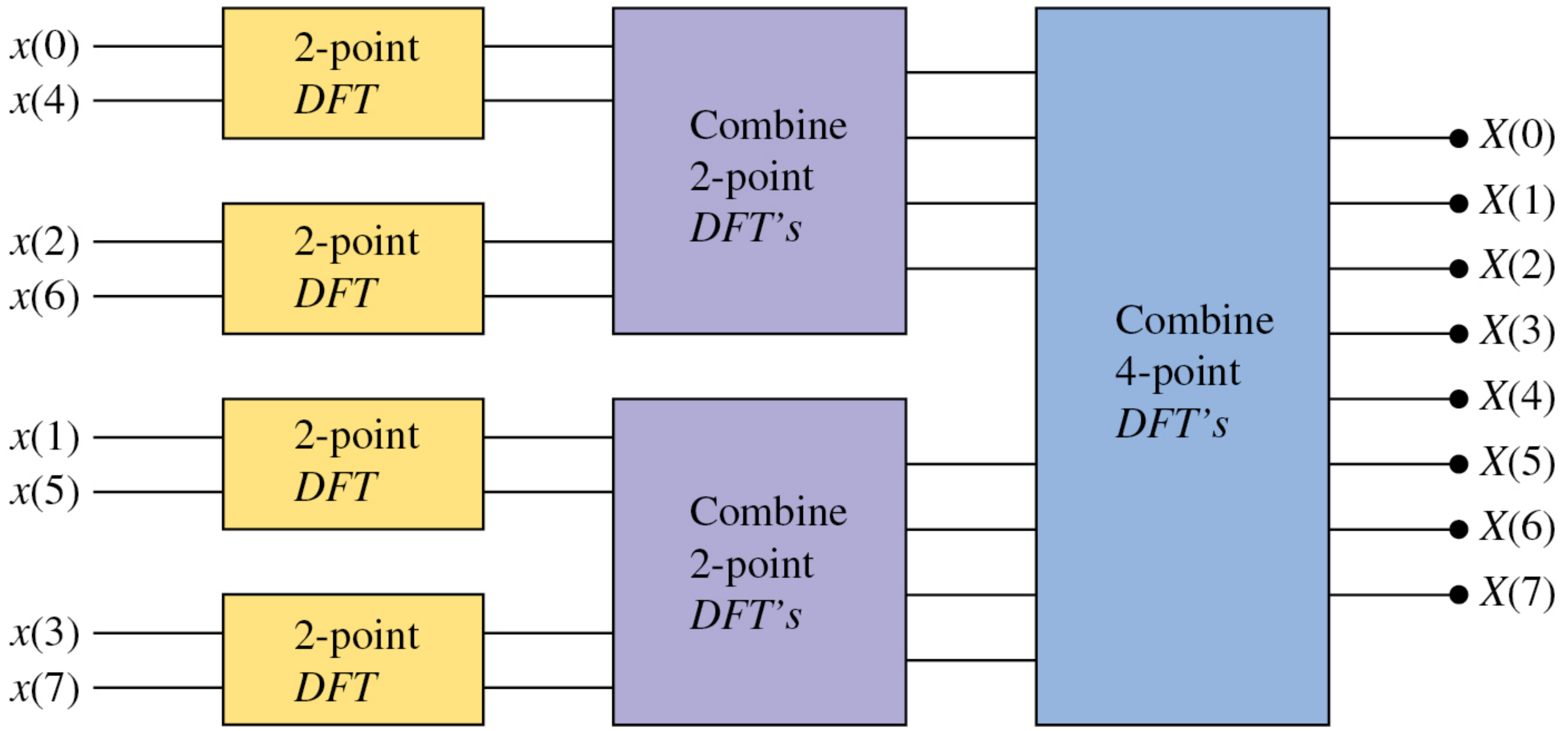
$$3:\frac{N}{4}DFT's:\ 2\left\{2\left[2\left(\frac{N}{8}\right)^2+\frac{N}{4}\right]+\frac{N}{2}\right\}+N=\frac{N^2}{8}+3N=\frac{N^2}{2^3}+3N$$

$$p:\frac{N}{2^p}DFT's\rightarrow\frac{N^2}{2^p}+3N=\frac{N^2}{N}+pN=N+N\log_2 N$$

$$\boxed{\sim O\left(N\log_2 N\right)\ \text{for large N}}$$

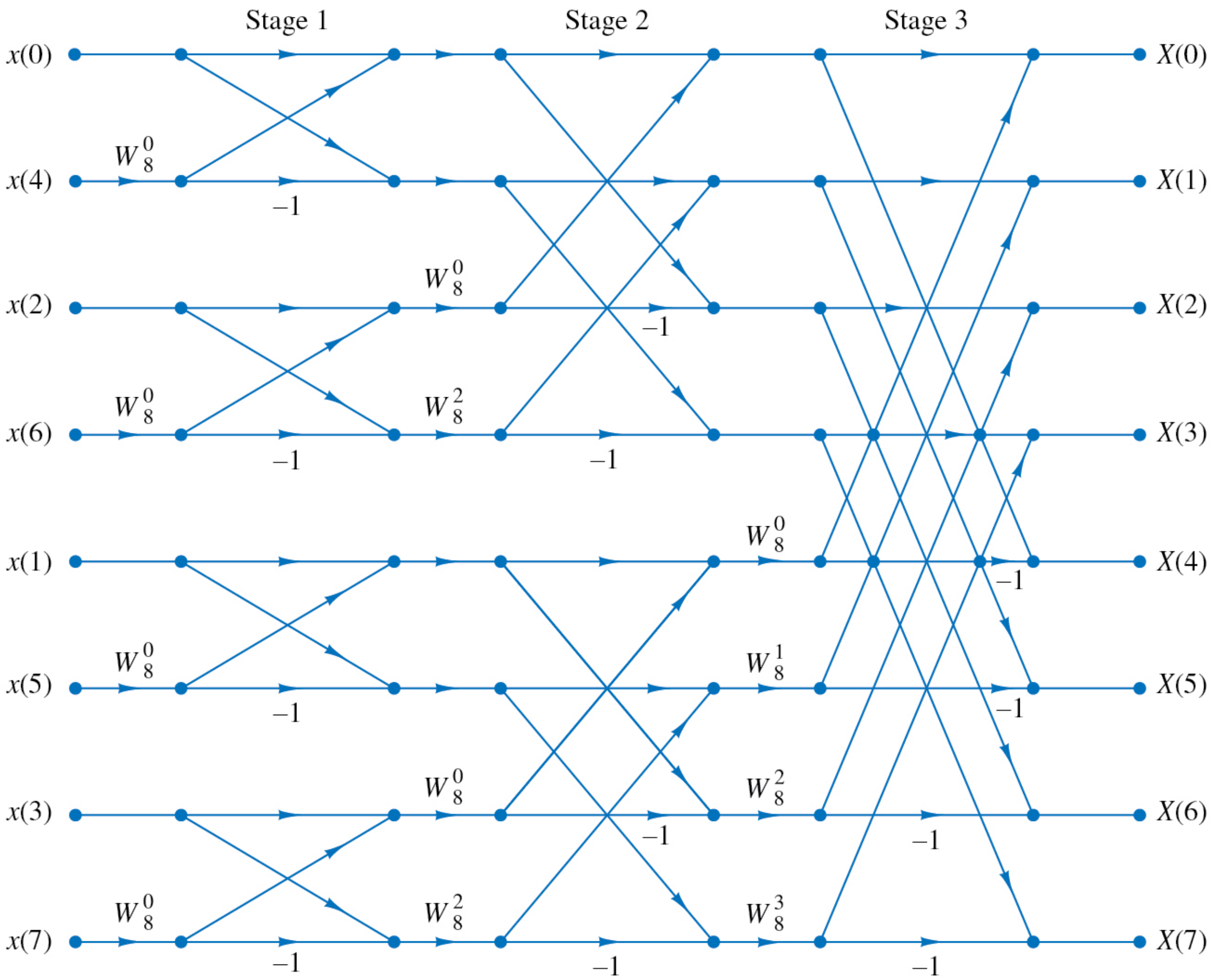
Fast Fourier Transform

Diagram for 3 stages of $N = 8$ point DFT



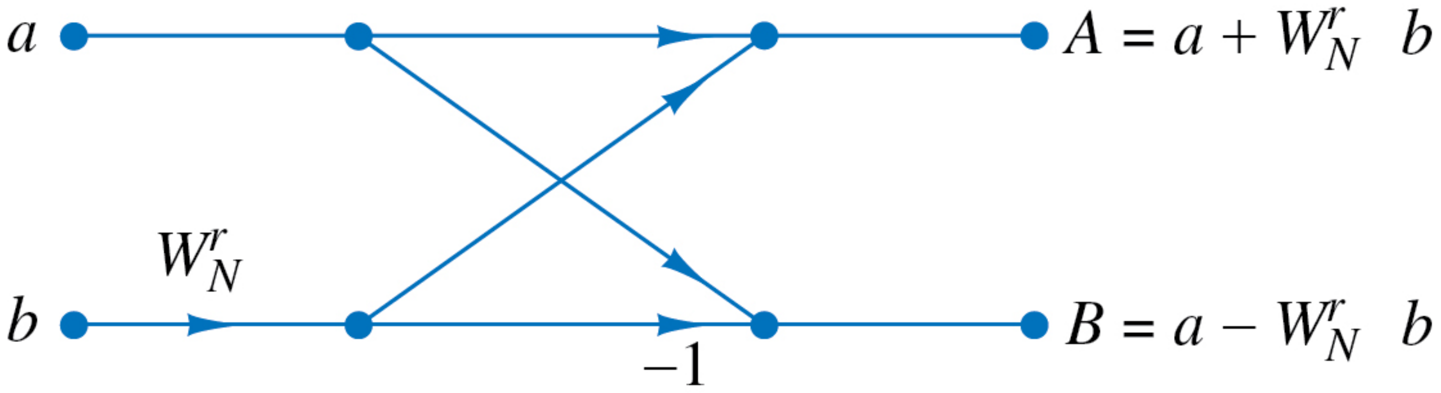
Fast Fourier Transform

Signal Flow Graph for $N = 8$ point DFT



Fast Fourier Transform

The basic operation that takes place at each stage is a "butterfly"



Fast Fourier Transform

A careful operation count shows that there are $N/2$ butterflies at each stage and $\log_2 N$ stages

Multiplies: $\frac{N}{2}\log_2 N$

Additions: $N\log_2 N$

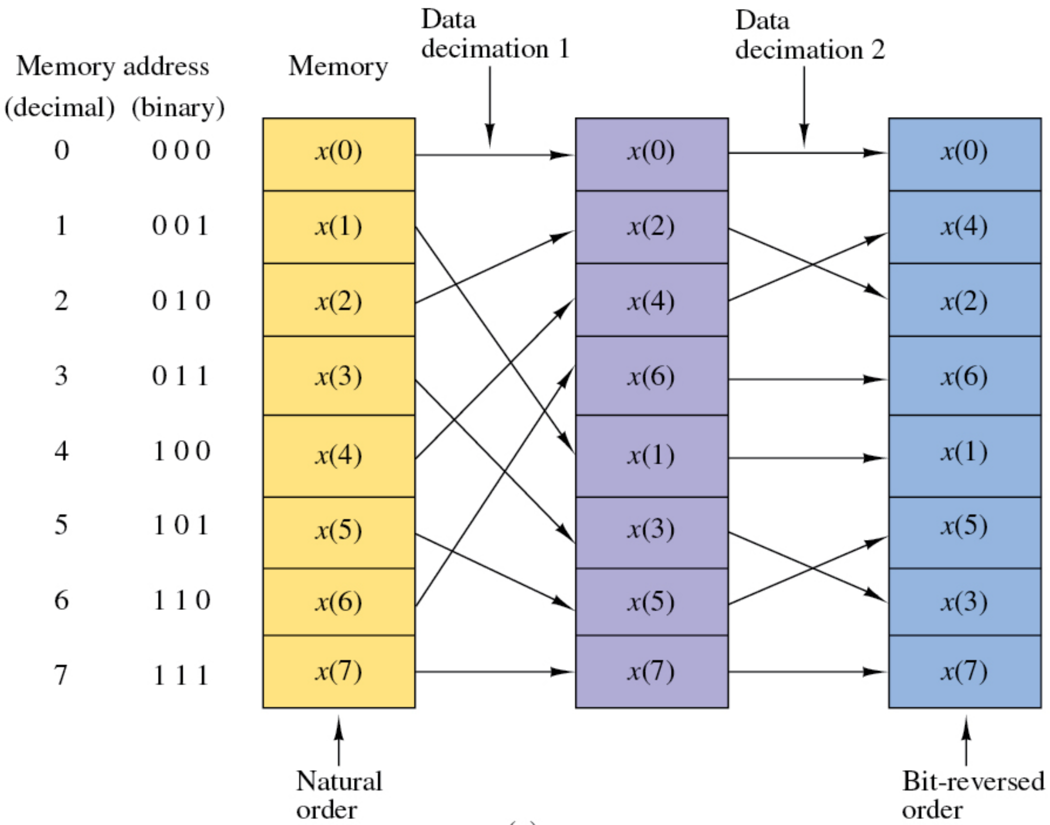
Number of Points, N	Complex Multiplications in Direct Computation, N^2	Complex Multiplications in FFT Algorithm, $(N/2)\log_2 N$	Speed Improvement Factor
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1,024	80	12.8
64	4,096	192	21.3
128	16,384	448	36.6
256	65,536	1,024	64.0
512	262,144	2,304	113.8
1,024	1,048,576	5,120	204.8

If you had a signal with 10^9 samples, and it took 1 nsec per multiply.

DFTwould take 31 years. FFT would take 30 sec.

Fast Fourier Transform

Notice that the order of the inputs is "weird" because of all of the sub-DFT suffling: $\{x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7)\}$



(a)

$(n_2 n_1 n_0)$	$\rightarrow$	$(n_1 n_0 n_2)$	$\rightarrow$	$(n_0 n_1 n_2)$
(0 0 0)	$\rightarrow$	(0 0 0)	$\rightarrow$	(0 0 0)
(0 0 1)	$\rightarrow$	(0 1 0)	$\rightarrow$	(1 0 0)
(0 1 0)	$\rightarrow$	(1 0 0)	$\rightarrow$	(0 1 0)
(0 1 1)	$\rightarrow$	(1 1 0)	$\rightarrow$	(1 1 0)
(1 0 0)	$\rightarrow$	(0 0 1)	$\rightarrow$	(0 0 1)
(1 0 1)	$\rightarrow$	(0 1 1)	$\rightarrow$	(1 0 1)
(1 1 0)	$\rightarrow$	(1 0 1)	$\rightarrow$	(0 1 1)
(1 1 1)	$\rightarrow$	(1 1 1)	$\rightarrow$	(1 1 1)

(b)

Fast Fourier Transform

- If you are into matrix stuff, another way to think of the radix-2 fft is as factorization of the W_N matrix
  - The fft becomes a product of $\log_2 N$ matrices where each matrix has only N non-zero elements
  - For an 8-point fft:

$$W_8 = \left(B_1 B_2 B_3\right) P_8$$

where P_8 reorders the inputs (bit-reversal)

Each B represents a butterfly

In general, FFTwould be:

$$\mathbf{X} = \left(\mathbf{B}_{\log_2 \mathbf{N}} \mathbf{L} \mathbf{B}_3 \mathbf{B}_2 \mathbf{B}_1\right) \mathbf{P}_\mathbf{N} \mathbf{x}$$

Fast Fourier Transform

For $N = 8$, the **bit-reversal permutation matrix** is:

$$P_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

First stage (Grouping pairs of 2)

$$B_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

Second stage (Grouping pairs of 4)

$$B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & W_8^1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & W_8^3 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & W_8^1 & 0 & 0 & 0 & -W_8^1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & W_8^3 & 0 & 0 & 0 & -W_8^3 \end{bmatrix}$$

Third stage (Final Combination)

$$B_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & W_8^1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & W_8^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & W_8^3 \end{bmatrix}$$

Fast Fourier Transform

- There are several other FFT algorithms
  - The book covers a couple of others
  - The Matlab fft uses radix-2 if signal is power of 2
    - If not, and has a few small factors, uses a mixed radix algorithm
    - If the sequence has a prime number number of samples, it uses the Bluestein (or chirp-z) algorithm

- A good video on fft for reference:
  - https://www.youtube.com/watch?v=lGCAlv3G8Oc