

Advanced Transform Methods

Fast Fourier Transform (FFT)

Andy Watson

Fast Fourier Transform (FFT)

- What is the FFT?
 - A collection of “tricks” that exploit the symmetry of the DFT calculation to make its execution much faster
 - Speedup increases with DFT size
- This lecture: outline the basic workings of the simplest formulation, the radix-2 decimation-in-time algorithm

Introduction, continued

- Some dates:
 - ~1880 - algorithm first described by Gauss
 - 1965 - algorithm rediscovered (not for the first time) by Cooley and Tukey
- FFT Revolutionized digital signal processing from 1960s
- E.g. in 1967 8192-point DFT on mainframe IBM 7094:
 - ~30 minutes using conventional techniques
 - ~5 seconds using FFTs

Measures of computational efficiency

- Could consider
 - Number of additions
 - Number of multiplications
 - Amount of memory required
 - Scalability and regularity
- Focus most on number of multiplications
 - More costly than additions for fixed-point processors
 - Same cost as additions for floating-point processors, but number of operations is comparable

Comput. Cost of Discrete-Time Filtering

Convolution of an N -point input with an M -point unit sample response

- Direct convolution:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

- Number of multiplies $\approx MN$

Comput. Cost of Discrete-Time Filtering

Convolution of an N -point input with an M -point unit sample response

- Using transforms directly:

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

- Computation of N -point DFTs requires N^2 multiplies
- Each convolution (two direct transforms plus an inverse transform) requires three DFTs of length $N+M-1$

$$3(N + M - 1)^2 + (N + M - 1)$$

For $N \gg M$ the computation is $O(N^2)$

Cooley-Tukey decimation-in-time algorithm

- Consider DFT algorithm for an integer power of 2, $N = 2^V$

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N} \quad W_N = e^{-j2\pi/N}$$

- Create separate sums for even and odd values of n :

$$X[k] = \sum_{n \text{ even}} x[n]W_N^{nk} + \sum_{n \text{ odd}} x[n]W_N^{nk}$$

- Letting $n = 2r$ for n even and $n = 2r + 1$ for n odd, we get

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

Note different sign in twiddle factor
in this lecture – common in FFT texts

Cooley-Tukey decimation in time algorithm

- Splitting indices in time, we have obtained

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

- But $W_N^2 = e^{-j2\pi 2/N} = e^{-j2\pi/(N/2)} = W_{N/2}$
and $W_N^{2rk} W_N^k = W_N^k W_{N/2}^{rk}$

So:

$$X[k] = \underbrace{\sum_{r=0}^{(N/2)-1} x[2r]W_{N/2}^{rk}}_{\text{N/2-point DFT of } x[2r]} + W_N^k \underbrace{\sum_{r=0}^{(N/2)-1} x[2r+1]W_{N/2}^{rk}}_{\text{N/2-point DFT of } x[2r+1]}$$

N/2-point DFT of $x[2r]$

N/2-point DFT of $x[2r+1]$

Savings so far ...

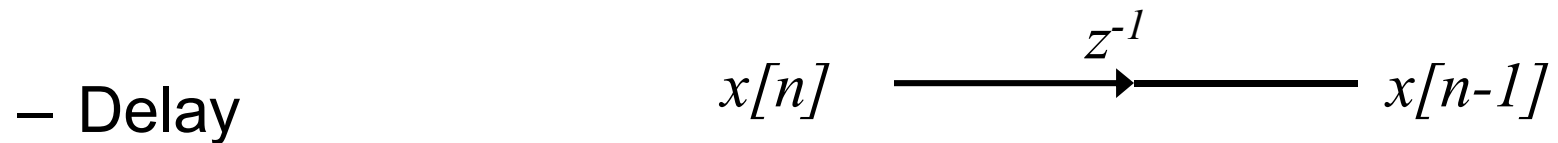
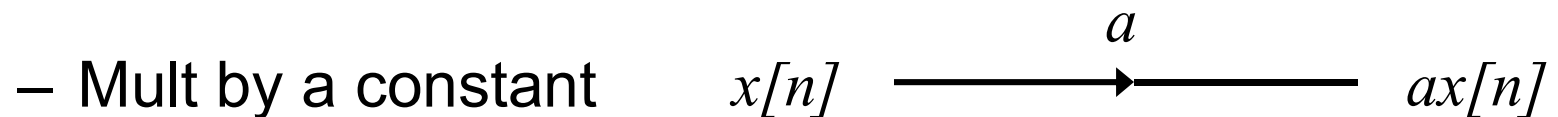
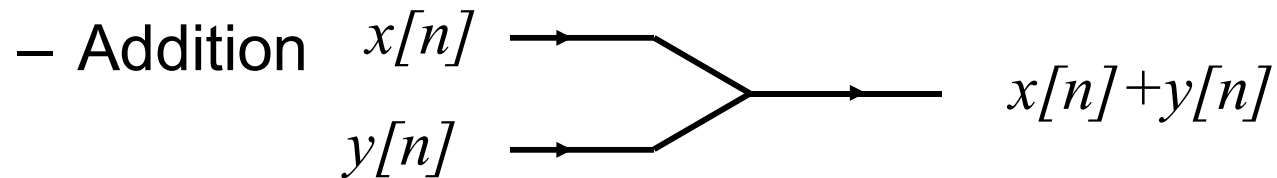
- We have split the DFT computation into two halves:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{nk} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} \end{aligned}$$

- Have we gained anything? Consider the nominal number of multiplications for $N = 8$
 - Original form produces $8^2 = 64$ multiplications
 - New form produces $2(4^2) + 8 = 40$ multiplications
 - So we're already ahead Let's keep going!!

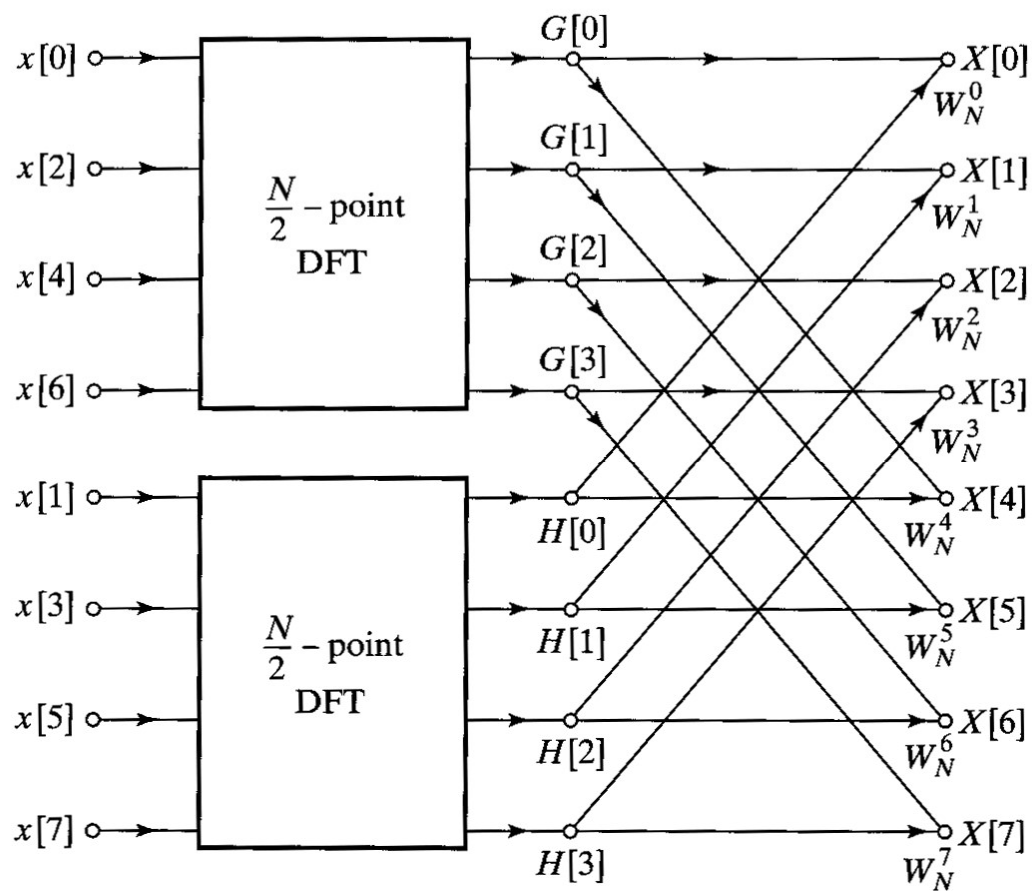
Signal flowgraph notation

- In generalizing this formulation, it is most convenient to adopt a graphic approach ...
- Signal flowgraph notation describes the three basic DSP operations:



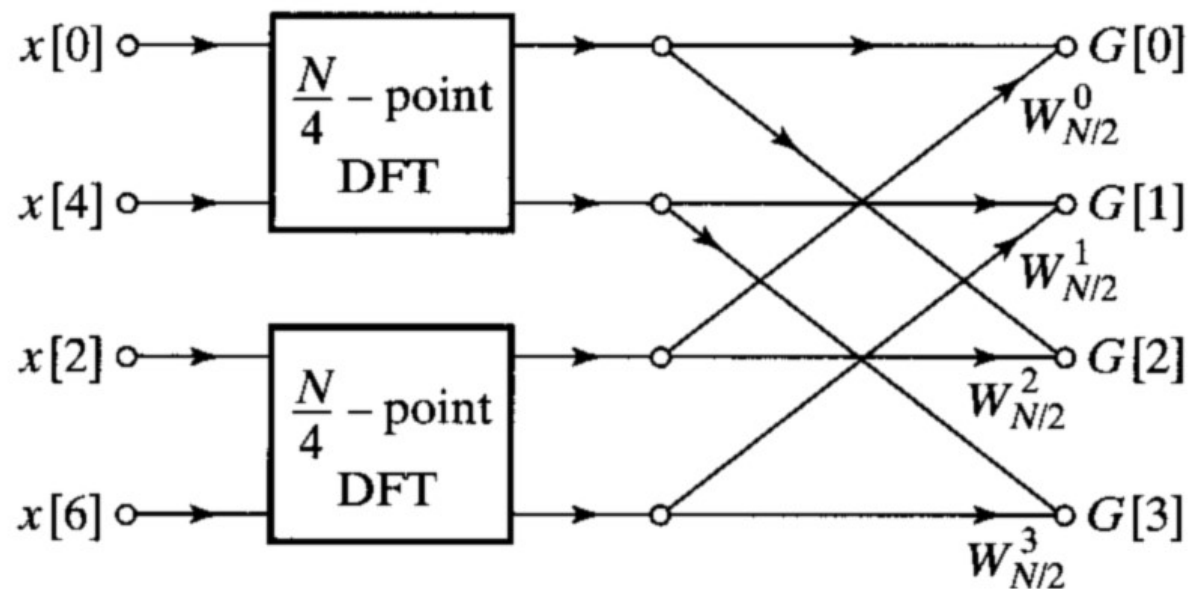
Signal flowgraph representation of 8-point DFT

- Recall that the DFT is now of the form $X[k] = G[k] + W_N^k H[k]$
- The DFT in (partial) flowgraph notation:

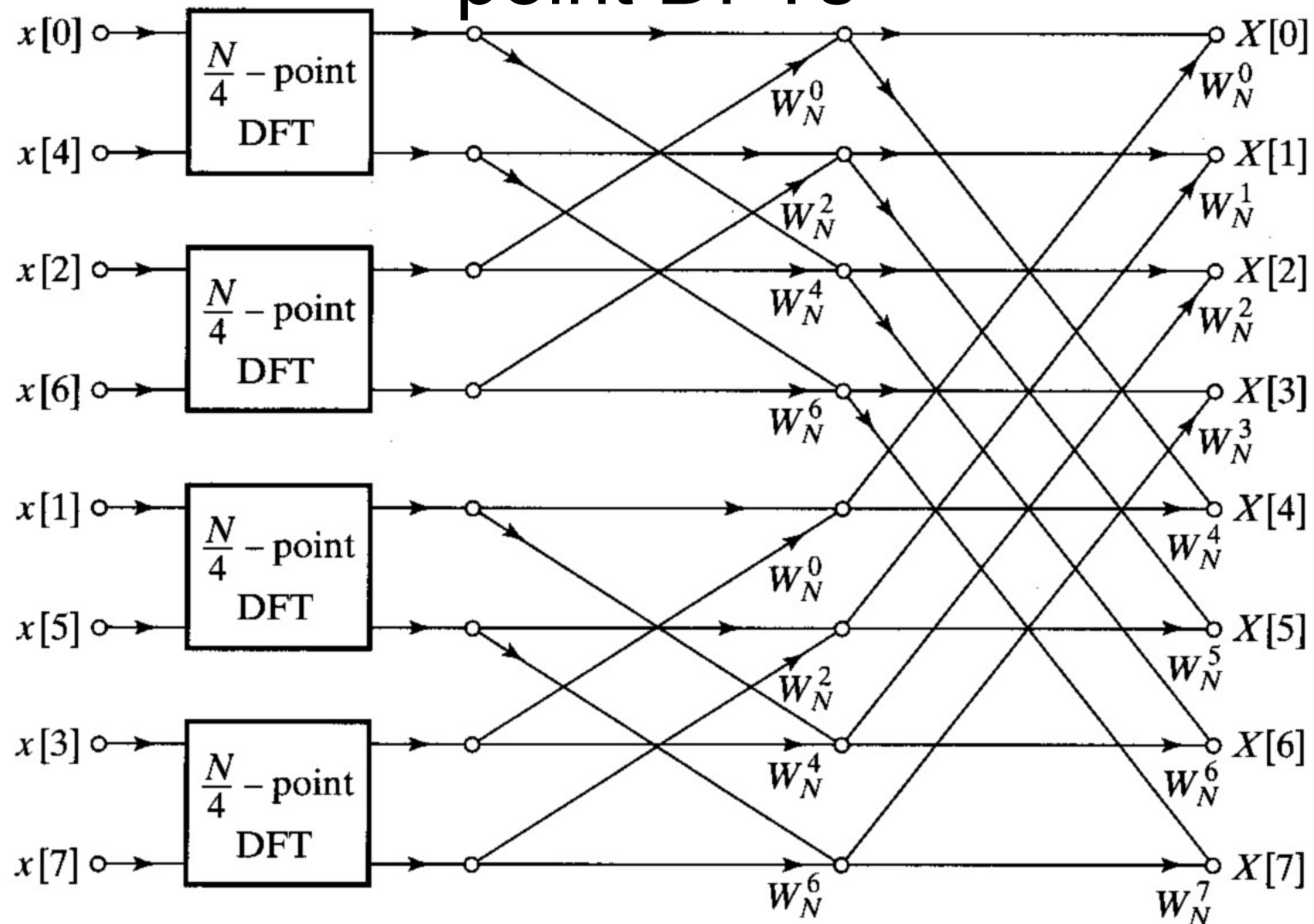


Continuing with the decomposition ...

- So why not break up into additional DFTs?
- Let's take the upper 4-point DFT and break it up into two 2-point DFTs:



The complete decomposition into 2-point DFTs



Now let's take a closer look at the 2-point DFT

- The expression for the 2-point DFT is:

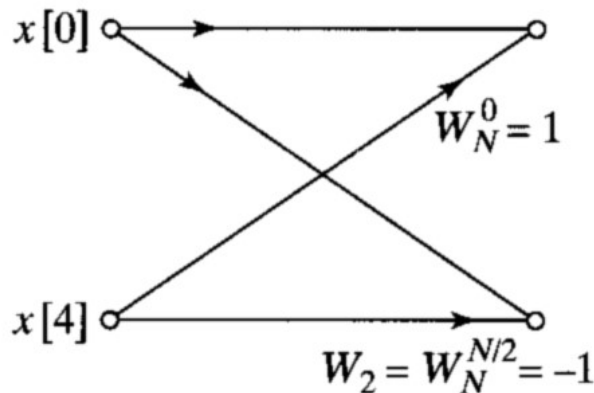
$$X[k] = \sum_{n=0}^1 x[n] W_2^{nk} = \sum_{n=0}^1 x[n] e^{-j2\pi nk/2}$$

- Evaluating for $k = 0, 1$ we obtain

$$X[0] = x[0] + x[1]$$

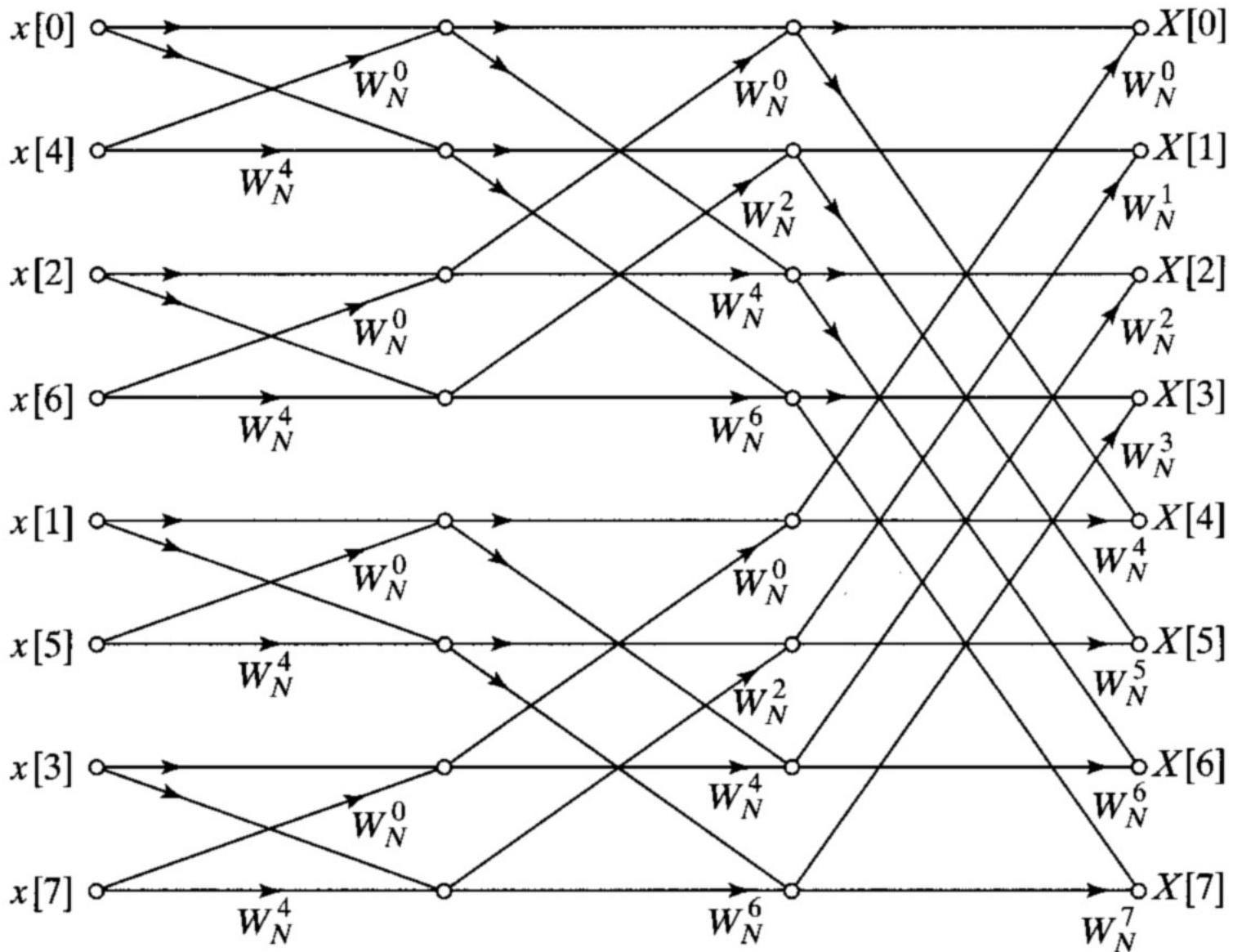
$$X[1] = x[0] + e^{-j2\pi 1/2} x[1] = x[0] - x[1]$$

which in signal flowgraph notation looks like ...

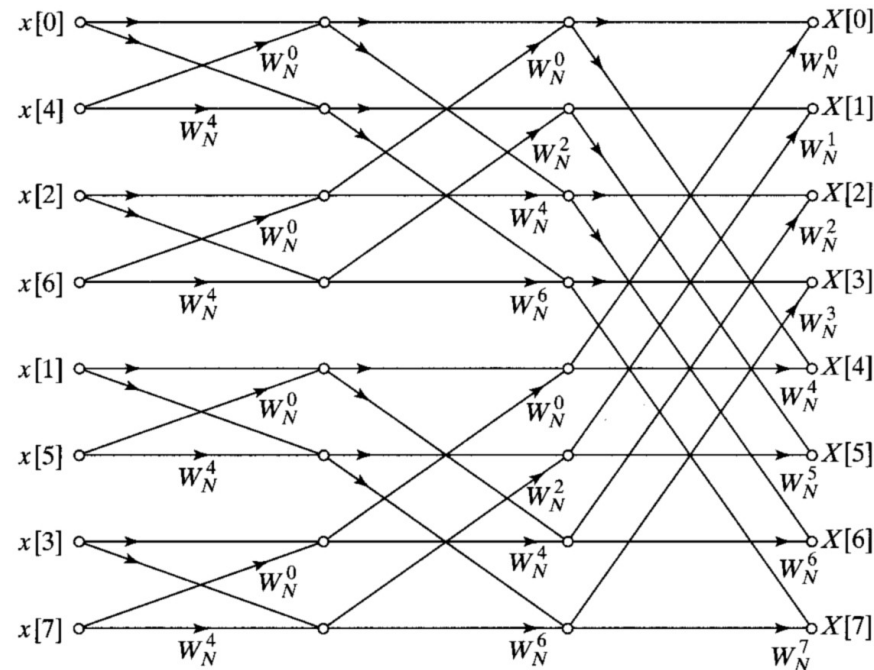


This topology is called the basic “butterfly”

The complete 8-point decimation-in-time FFT



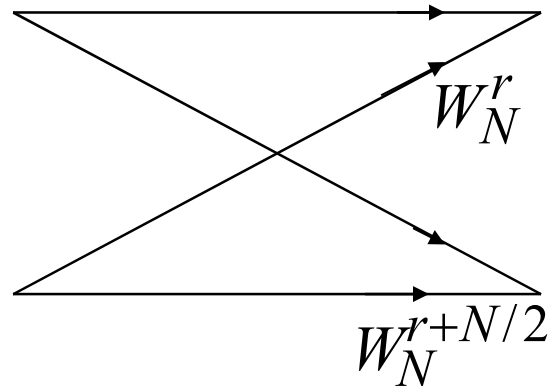
Number of multiplies for N-point FFTs



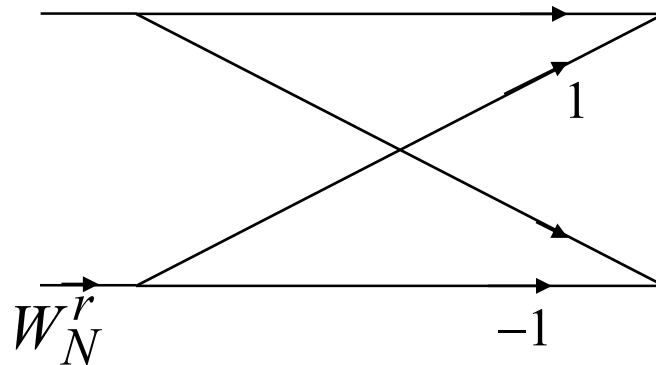
- Let $N = 2^\nu$ where $\nu = \log_2(N)$
- $(\log_2(N) \text{ columns})(N/2 \text{ butterflies/column})(2 \text{ mults/butterfly})$
or $\sim N \log_2(N)$ multiplies

Additional timesavers: reducing multiplications in the basic butterfly

- As we derived it, the basic butterfly is of the form

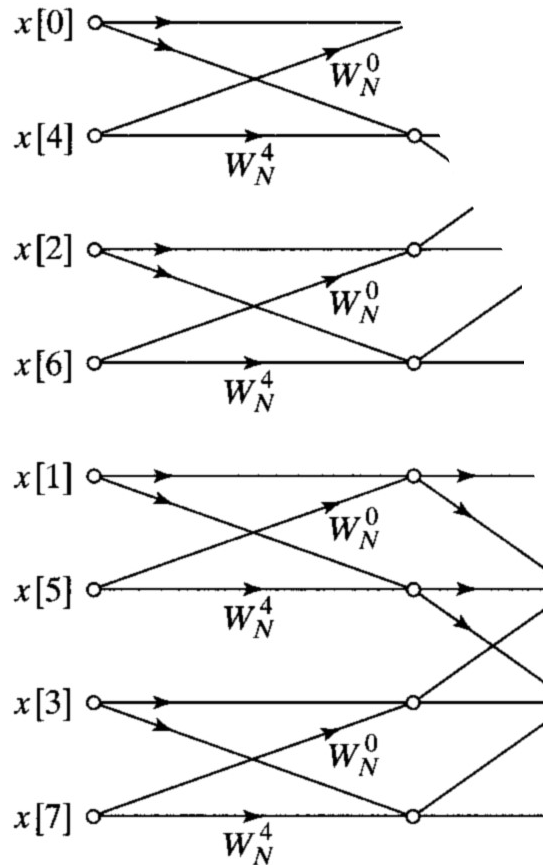


- Since $W_N^{N/2} = -1$ we can reduce computation by 2 by premultiplying by W_N^r



Bit reversal of the input

- Recall the first stages of the 8-point FFT:



Consider the binary representation of the indices of the input:

0	000
4	100
2	010
6	110
1	001
5	101
3	011
7	111

If these binary indices are time reversed, we get the binary sequence representing 0,1,2,3,4,5,6,7

Hence the indices of the FFT inputs are said to be in **bit-reversed order**

Some comments on bit reversal

- This implementation of FFT: input is bit reversed, output is in natural order
- Some other implementations: input in natural order, output bit reversed
- Sometimes convenient to implement filtering applications by
 - Use FFTs with input in natural order, output in bit-reversed order
 - Multiply frequency coefficients together (in bit-reversed order)
 - Use inverse FFTs with input in bit-reversed order, output in natural order
- Computing in this fashion means we never have to compute bit reversal explicitly