18-491/691 Lecture #15
INTRODUCTION TO THE FAST FOURIER TRANSFORM ALGORITHM

Richard M. Stern

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Phone: +1 (412) 268-2535
FAX: +1 (412) 268-3890
rms@cs.cmu.edu
http://www.ece.cmu.edu/~rms

March 22, 2021
[OSYP 9.2]

Introduction

- **Today we will begin our discussion of the family of algorithms known as "Fast Fourier Transforms," which have revolutionized digital signal processing**
- **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
- **Today - will outline the basic workings of the simplest formulation, the radix-2 decimation-in-time algorithm**
- **Wednesday - will discuss some of the variations and extensions**
  - Alternate structures
  - Non-radix 2 formulations

Introduction, continued

- **Some dates:**
  - ~1880 - algorithm first described by Gauss
  - 1965 - algorithm rediscovered (not for the first time) by Cooley and Tukey

- **In 1967 (spring of my freshman year), calculation of a 8192-point DFT on the top-of-the line IBM 7094 took ….**
  - ~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

- **For the present discussion we'll focus most on number of multiplications as a measure of computational complexity**
  - More costly than additions for fixed-point processors
  - Same cost as additions for floating-point processors, but number of operations is comparable

Computational 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 multiplys ≈ *MN*
  - For *M* ≈ *N*, the number of multiplys is $O(N^2)$

Computational 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 multiplys
  - Each convolution requires three DFTs of length *N+M-1* plus an additional *N+M-1* complex multiplys or

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

  - For $N >> M$, for example, the computation is $O(N^2)$

The Cooley-Tukey decimation-in-time algorithm

- **Consider the DFT algorithm for an integer power of 2,** $N = 2^{\nu}$

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

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

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

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

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

The 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 …

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

G[k] *H[k]*

Savings so far …

- **We have split the DFT computation into two halves:**

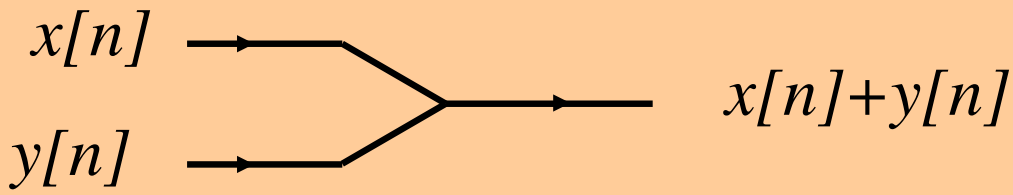
$$X[k] = \sum_{k=0}^{N-1} x[n] W_N^{nk}$$

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

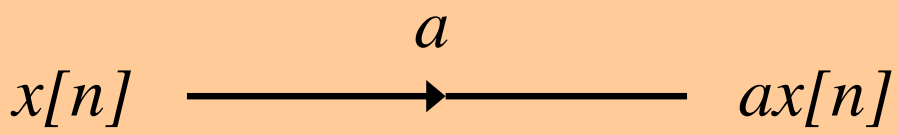
- **Have we gained anything? Consider the nominal number of multiplications for** $N = 8$
  - Original form produces $8^2 = 64$ multiplications
  - New form produces $2(4^2) + 8 = 40$ multiplications
  - So we're already ahead ….. Let's keep going!!

Signal flowgraph notation

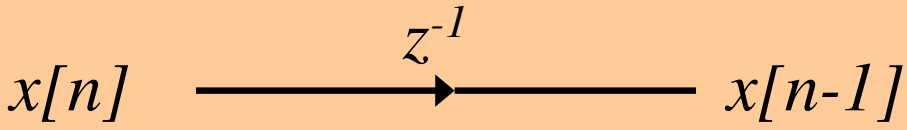
- **In generalizing this formulation, it is most convenient to adopt a graphic approach …**
- **Signal flowgraph notation describes the three basic DSP operations:**
  - Addition



  - Multiplication by a constant

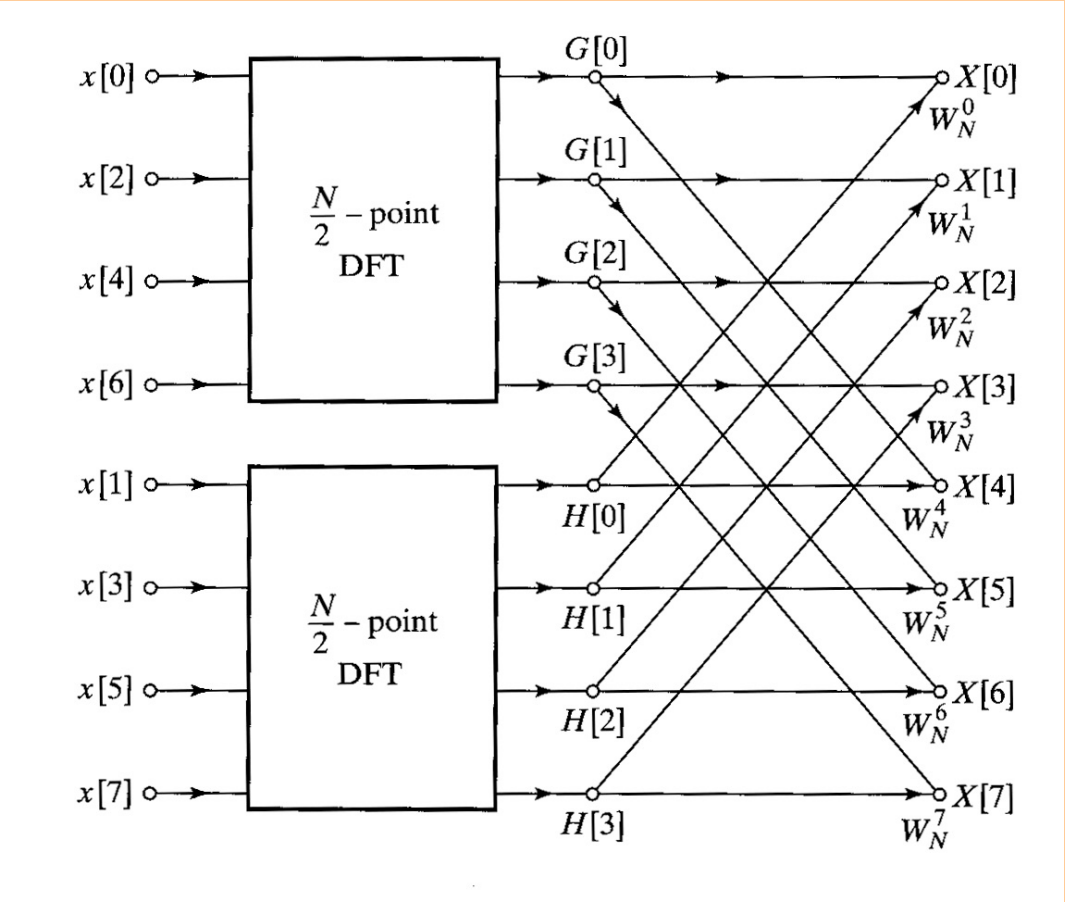


  - Delay



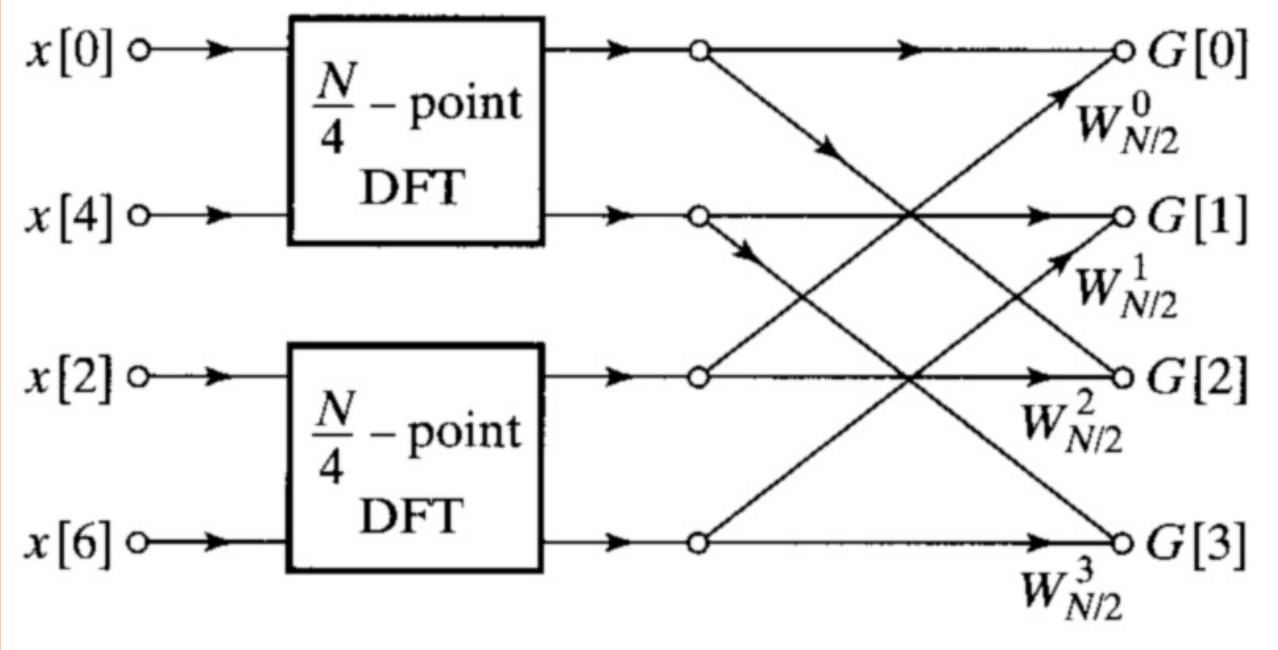
Signal flowgraph representation of 8-point DFT

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

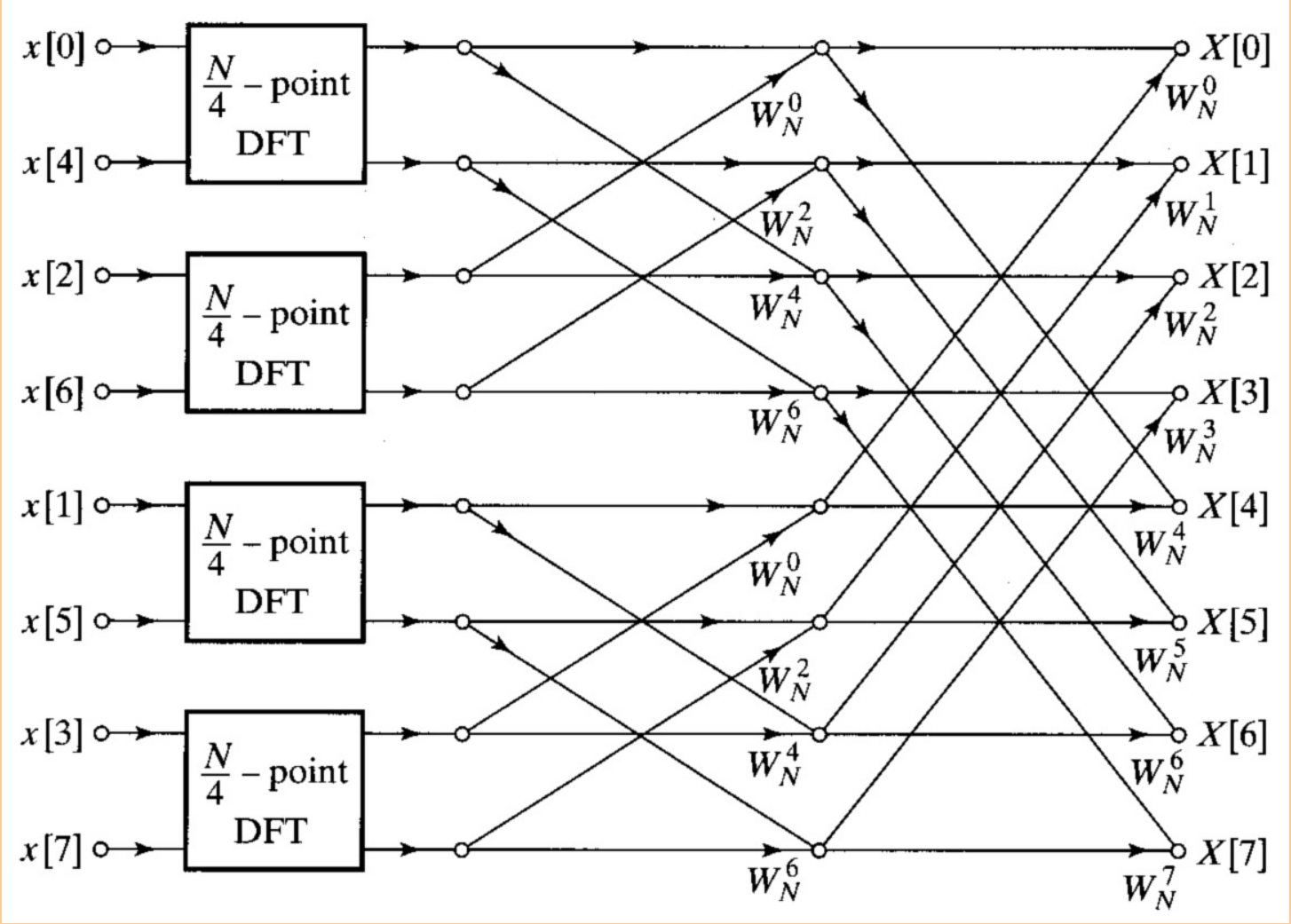


Continuing with the decomposition …

- 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:**

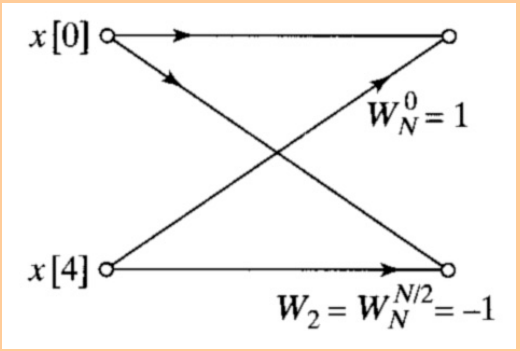
$$X[k] = \sum_{n=0}^{1} x[n] W_2^{nk} = \sum_{n=0}^{1} x[n] e^{-j2\pi nk/2}$$

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

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

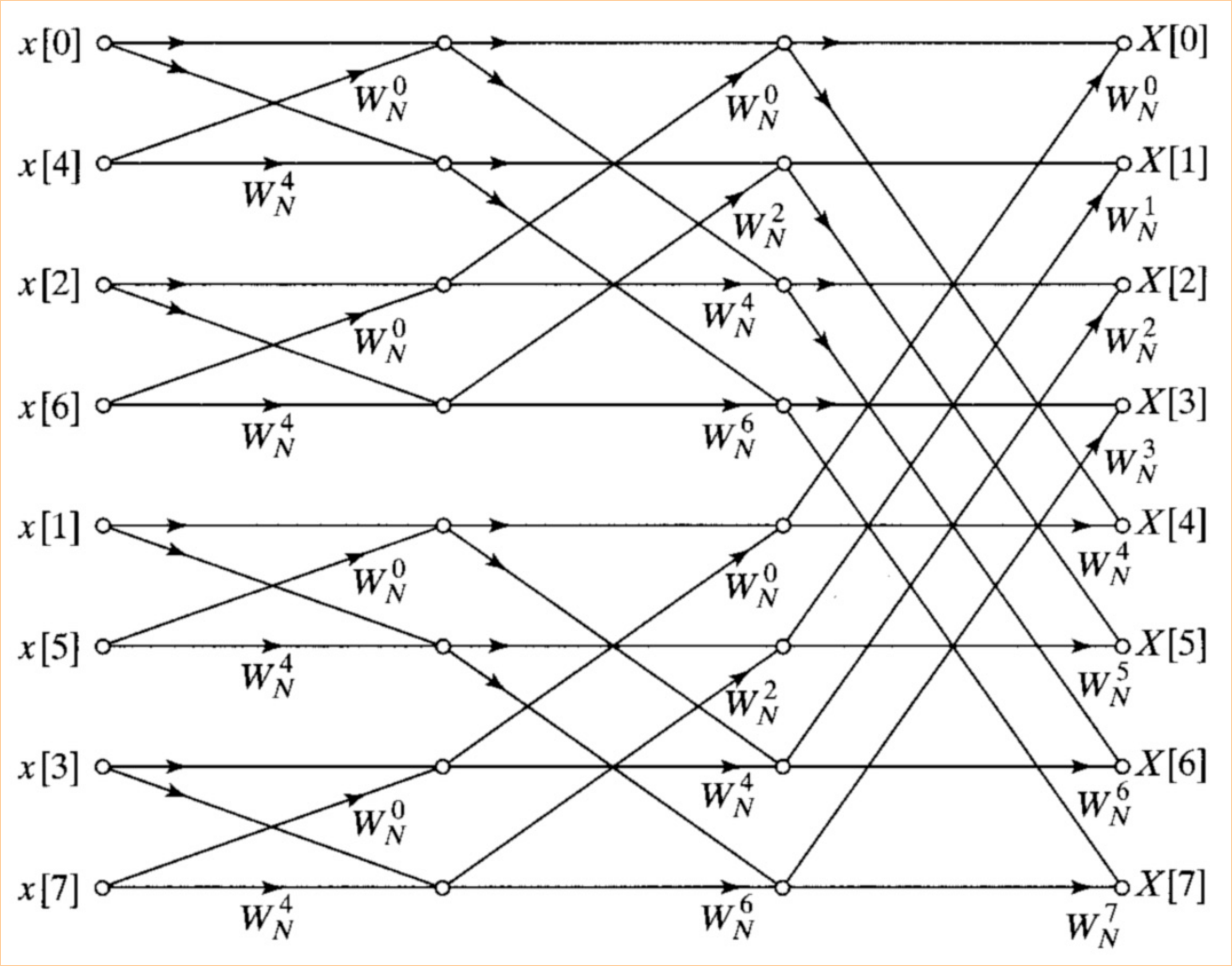
$$X[1] = x[0] + e^{-j2\pi 1/2} x[1] = x[0] - x[1]$$

which in signal flowgraph notation looks like ...

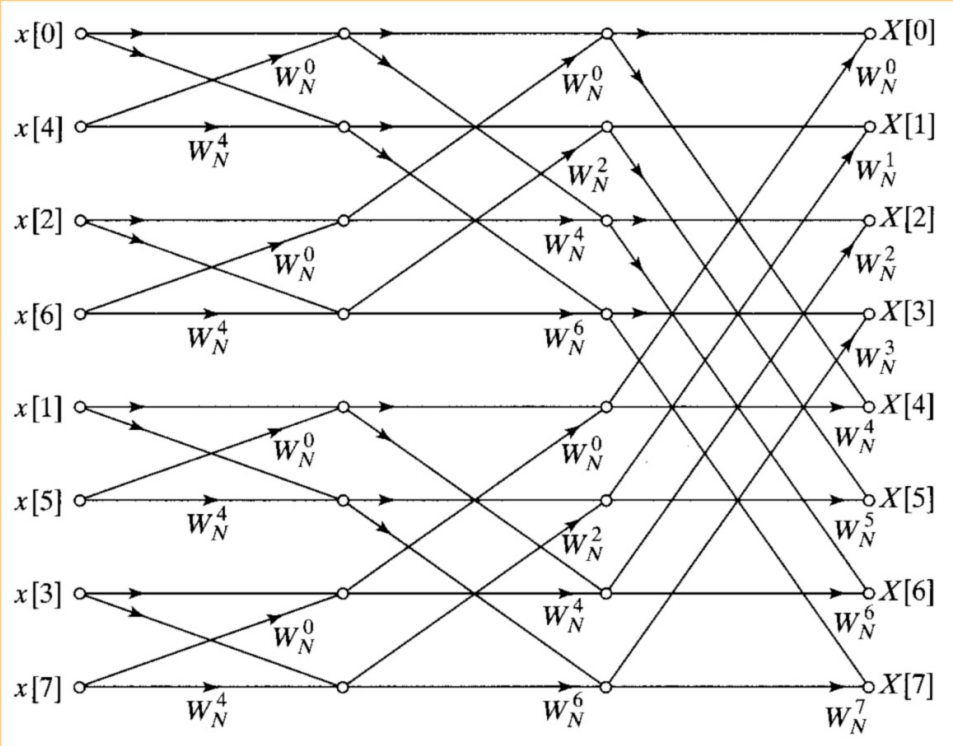


This topology is referred to as the basic butterfly

The complete 8-point decimation-in-time FFT



Number of multiplys for N-point FFTs



- **Let** $N = 2^{\nu}$ where $\nu = \log_2(N)$
- **($\log_2(N)$ columns)($N/2$ butterflys/column)(2 mults/butterfly)**

  or ~ $N\log_2(N)$ **multiplys**

Comparing processing with and without FFTs

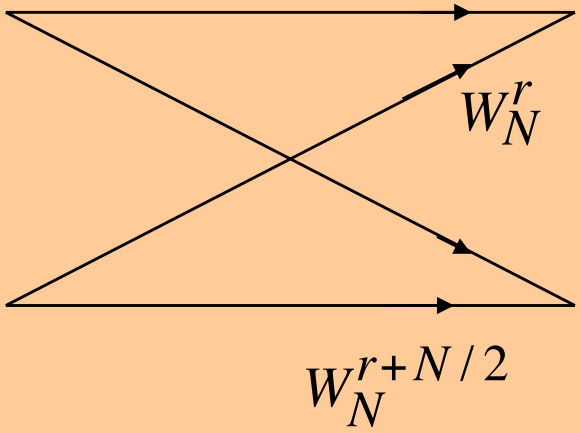
- "Slow" DFT requires N^2 mults; FFT requires $N \log_2(N)$ mults
- Filtering using FFTs requires $3(N \log_2(N))+N$ mults
- Let $\alpha_1 = N\log_2(N)/N^2;\ \ \alpha_2 = [3(N\log_2(N))+N]/N^2$

N	α_1	α_2
16	.25	.8124
32	.156	.50
64	.0935	.297
128	.055	.171
256	.031	.097
1024	.0097	.0302

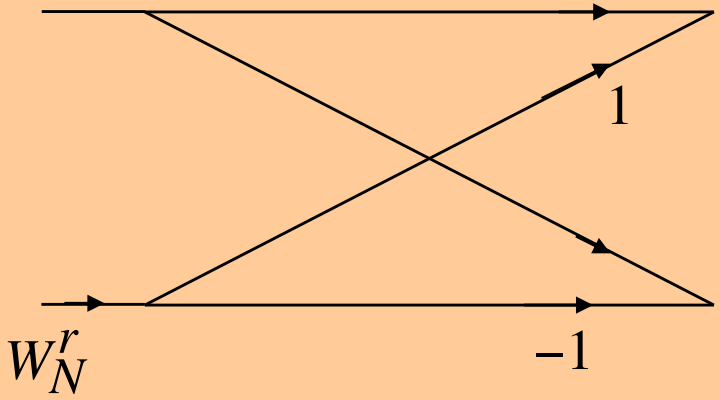
Note: 1024-point FFTs accomplish speedups of 100 for DFTs, 30 for filtering!

Additional timesavers: reducing multiplications in the basic butterfly

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

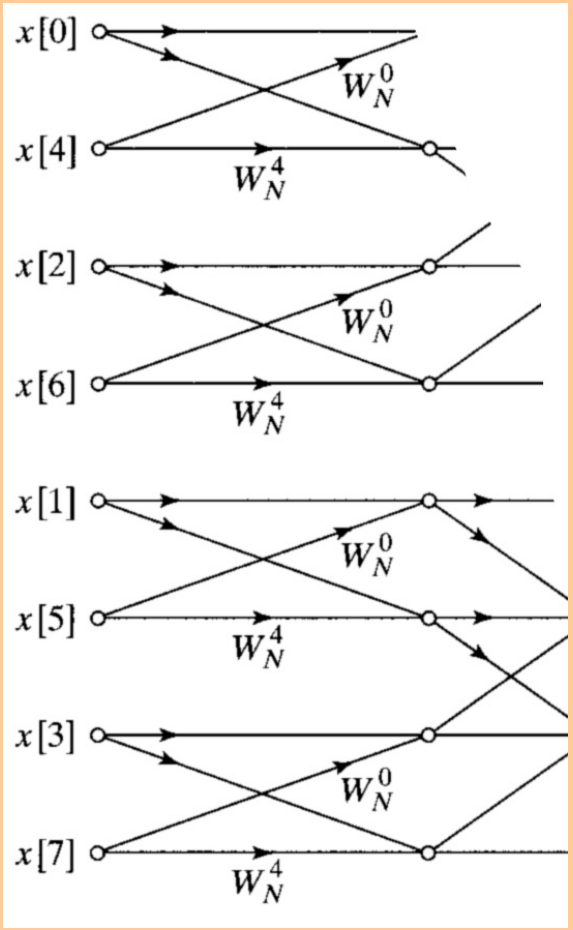


- Since $W_N^{N/2} = -1$ we can reduce computation by 2 by premultiplying by W_N^r



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

- **In the implementation of the FFT that we discussed, the input is bit reversed and the output is developed in natural order**
- **Some other implementations of the FFT have the input in natural order and the output bit reversed (to be described Wednesday)**
- **In some situations it is 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**

Using FFTs for inverse DFTs

- **We've always been talking about forward DFTs in our discussion about FFTs …. what about the inverse FFT?**

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]W_N^{-kn}; \quad X[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$

- **One way to modify FFT algorithm for the inverse DFT computation is:**
  - Replace W_N^k by W_N^{-k} wherever it appears
  - Multiply final output by $1/N$
- **This method has the disadvantage that it requires modifying the internal code in the FFT subroutine**

A better way to modify FFT code for inverse DFTs

- **Taking the complex conjugate of both sides of the IDFT equation and multiplying by *N*:**

$$Nx^*[n] = \sum_{k=0}^{N-1} X^*[k] W_N^{nk}; \text{ or } x[n] = \frac{1}{N}\left[\sum_{k=0}^{N-1} X^*[k] W_N^{nk}\right]^*$$

- **This suggests that we can modify the FFT algorithm for the inverse DFT computation by the following:**
  - Complex conjugate the input DFT coefficients
  - Compute the *forward* FFT
  - Complex conjugate the output of the FFT and multiply by $1/N$
- **This method has the advantage that the internal FFT code is undisturbed; it is widely used.**

Summary

- We developed the structure of the basic decimation-in-time FFT
- Use of the FFT algorithm reduces the number of multiplys required to perform the DFT by a factor of more than 100 for 1024-point DFTs, with the advantage increasing with increasing DFT size
- We can use the same structure to compute inverse FFTs
- On Wednesday we will consider alternate forms of the FFT, and FFTs for values of DFT sizes that are not an integer power of 2