Fourier Transforms – The *Fast* Fourier Transform, cont'd

CS370 Lecture 23 – March 8, 2017

Making Fourier Transforms Fast

We've examined the discrete Fourier transform and its inverse.

Next, we want to determine a more *efficient* way to compute them.

Questions:

- What is the complexity of the naïve method?

  Answer: $\boldsymbol{O(N^2)}$

- What properties of the DFT allow it to be sped up?

  Answer: We can write a length-N DFT as 2 length-N/2 DFT's.

- What is the complexity of the new method?

A Faster Fourier Transform

Design a *divide and conquer* strategy.

We'll:

1. Split the full DFT into *two* DFT's of *half* the length.
2. Repeat recursively.
3. Finish at the base case: the DFT of individual pairs of numbers.

(If $N \neq 2^m$ for some m, we can pad our initial data with zeros.)

Key question: *How* can we split up the DFT?

Dividing it up

The usual DFT of the sequence f_n is:

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$$

It can be split into two separate DFT's of two arrays of ***half the length*** (N/2):

$$g_n = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$

$$h_n = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right) W^{-n}$$

The "W-factor" (or sometimes twiddle factor).

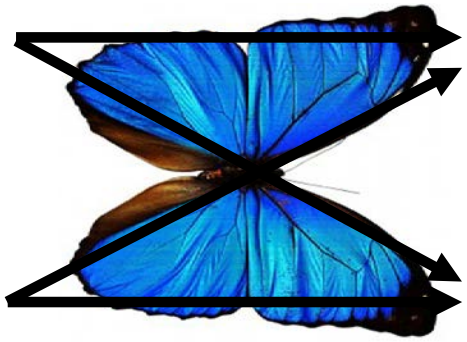
where $n \in \left[0, \frac{N}{2} - 1\right]$. Then $F_{even} = DFT(g)$ and $F_{odd} = DFT(h)$.

Visualizing – A Butterfly operation

We can think of each step as taking a pair of numbers and producing two outputs:

f_n

$f_{n+\frac{N}{2}}$



g_n

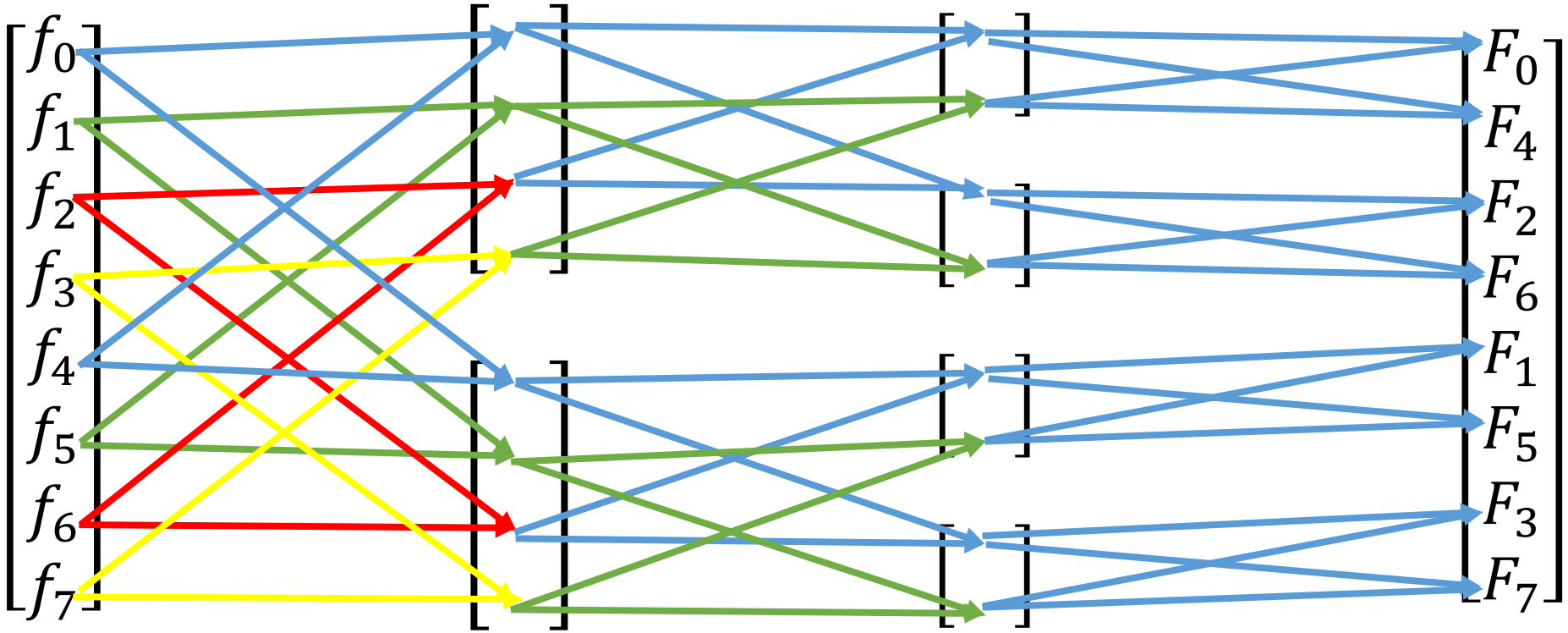
$$\frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$

$$\frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right)W^{-n}$$

h_n

The (admittedly mild) resemblance to a butterfly gives its name.

Big Picture – Recursive Butterfly FFT alg'm

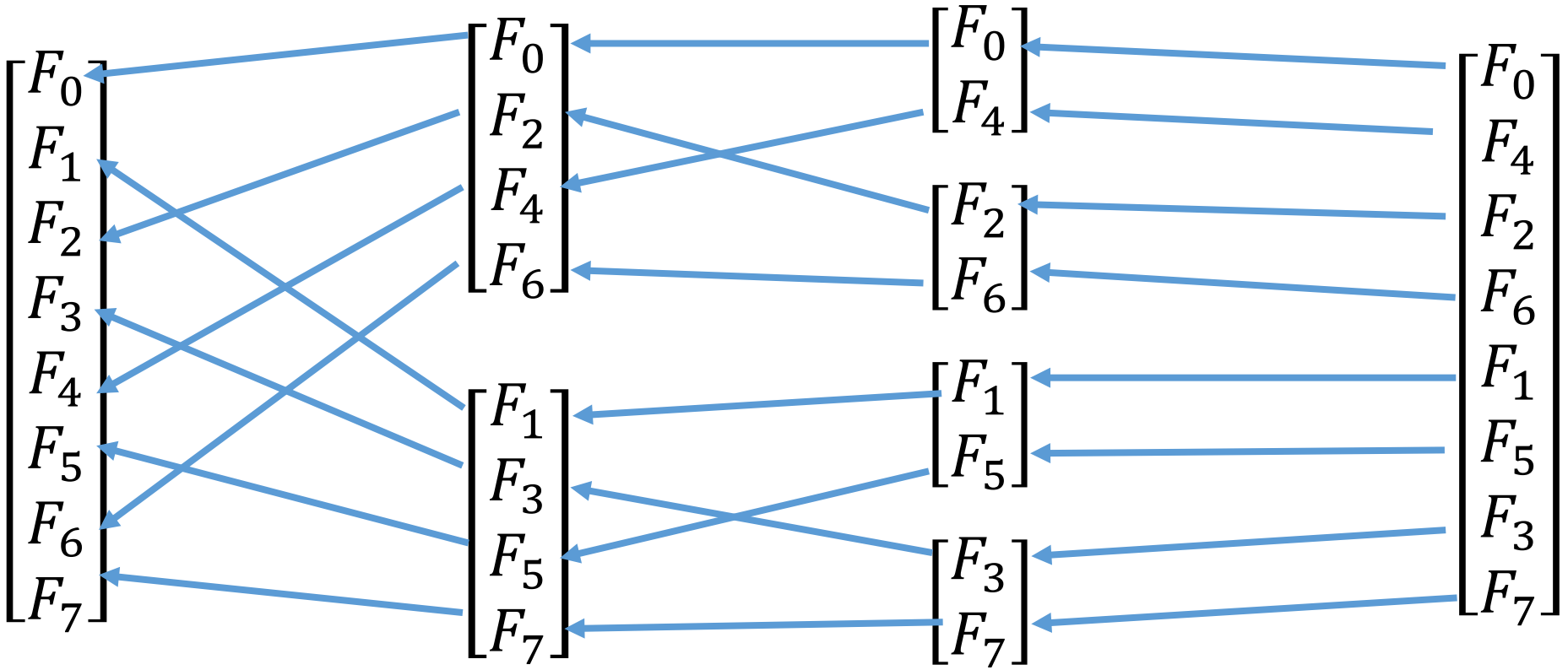


$N = 8 = 2^3$, so we have 3 recursive stages.

Coefficient output order is "bit-reversed"!

Reassembling The Result Vector

Each step applies an even-odd index splitting for the result location. So, the output ends up in “bit-reversed” positions, which we must undo.



Bit-Reversed Output

Consider the binary indices of the data: output indices have bits in reverse order!

So we must “unscramble” the coefficients as a final step. Just reorder the result vector.

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} f_{000} \\ f_{001} \\ f_{010} \\ f_{011} \\ f_{100} \\ f_{101} \\ f_{110} \\ f_{111} \end{bmatrix}$$

Perform All Butterfly Stages

$$\begin{bmatrix} F_0 \\ F_4 \\ F_2 \\ F_6 \\ F_1 \\ F_5 \\ F_3 \\ F_7 \end{bmatrix} = \begin{bmatrix} F_{000} \\ F_{100} \\ F_{010} \\ F_{110} \\ F_{001} \\ F_{101} \\ F_{011} \\ F_{111} \end{bmatrix}$$

Is it actually faster?

Naïve algorithm: $\mathrm{O}(N^2)$ complex FP operations.

FFT algorithm:

- We spend $\mathrm{O}(N)$ operations to split the arrays in two, at each “stage”.
- Data has (or was padded to) length $N = 2^m$, so we need $m = \log_2 N$ stages.
- Total cost $\mathrm{O}(N \log_2 N)$ operations. Cheaper!

Inverse Fast Fourier Transform

The DFT and IDFT have similar forms.

So, the *same core butterfly algorithm* can perform either, with minor changes.

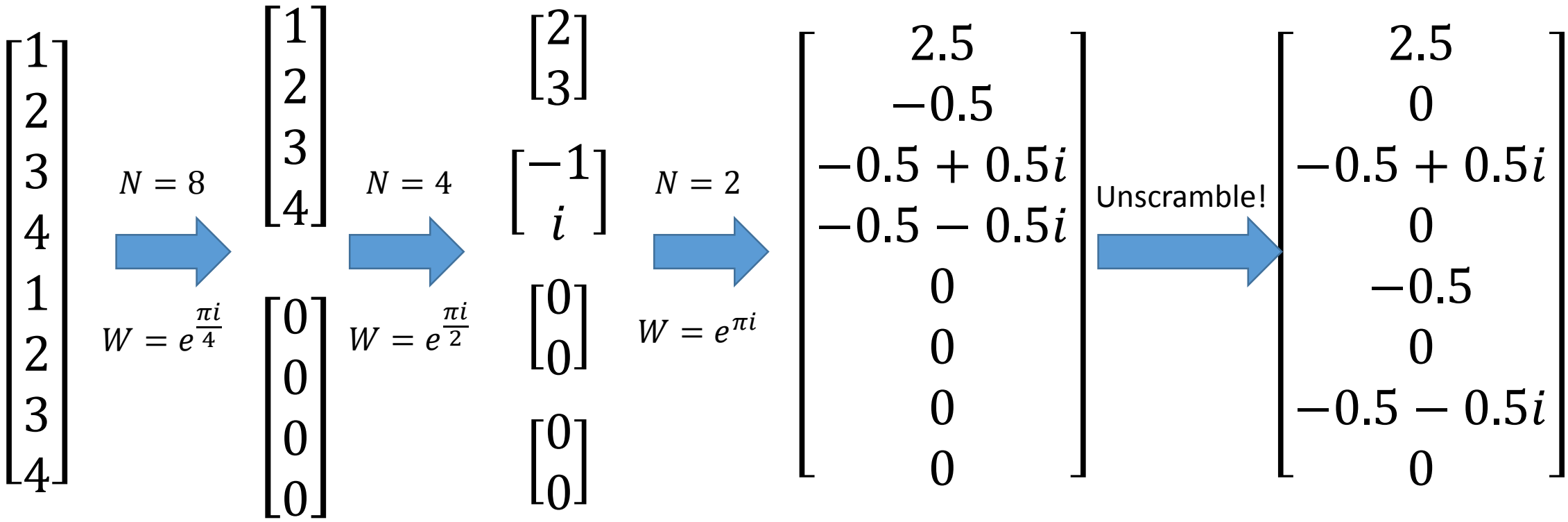
Consider the generic expression:

$$f'_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n U^{nk}$$

If $U = W^{-1}$ this is our forward DFT.

If $U = W$, and we post-multiply by N, this is the reverse (inverse) FFT.

A Complete Butterfly example (Ex 5.4)



$$g_n = \frac{1}{2}\left(f_n + f_{n+\frac{N}{2}}\right)$$

$$h_n = \frac{1}{2}\left(f_n - f_{n+\frac{N}{2}}\right)W^{-n}$$

Use the "current" N at each stage to determine W.

Another real FFT example:

Find the sequence of vectors and the final result of applying the butterfly FFT approach to the data:

$$f_n = \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \\ 4 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

A smaller, complex FFT example:

Find the sequence of vectors and the final result of applying the butterfly FFT approach to the data:

$$f_n = \begin{bmatrix} 2+i \\ 3 \\ -2 \\ -2i \end{bmatrix}$$