COM303: Digital Signal Processing

Lecture 8: Numerical Fourier Analysis

Overview:

- the short-time Fourier transform (aka the "spectrogram")
- the Fast Fourier Transform algorithm

the short-time Fourier transform (STFT)

Overview:

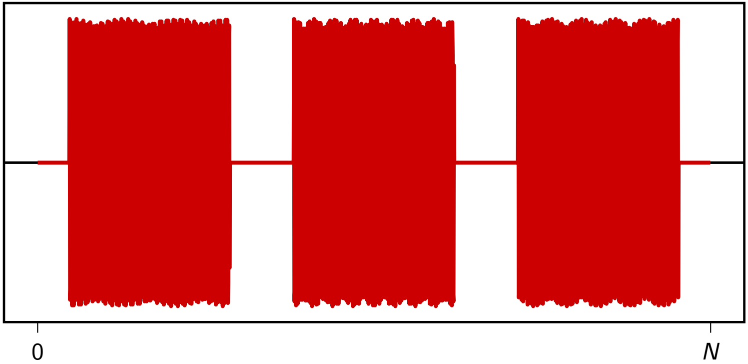
- Time vs frequency representations
- The STFT and the spectrogram
- Time-Frequency tilings

Dual-Tone Multi Frequency dialing

DTMF signaling

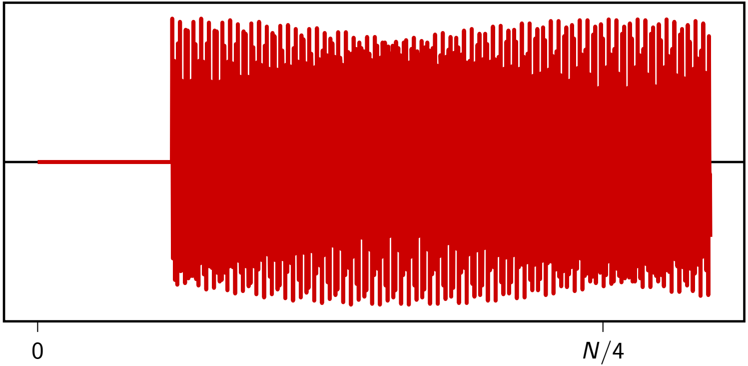
	1209Hz	1336Hz	1477Hz
697Hz	1	2	3
770Hz	4	5	6
852Hz	7	8	9
941Hz	*	0	#

1-5-9 in time

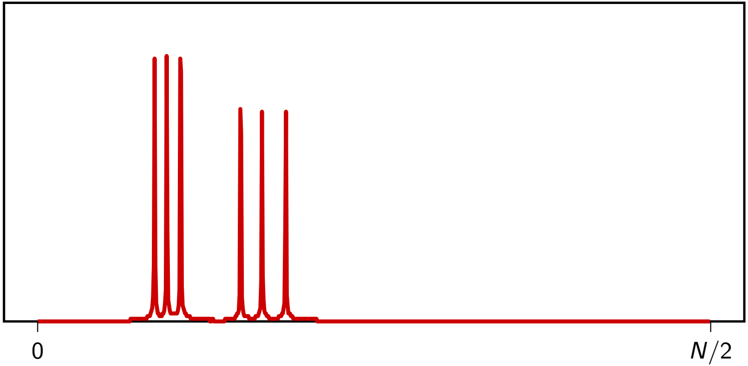


Play

1-5-9 in time (detail)



1-5-9 in frequency (magnitude)



The fundamental tradeoff

- time representation obfuscates frequency
- frequency representation obfuscates time

Short-Time Fourier Transform

Idea:

- take small signal pieces of length L
- look at the DFT of each piece:

$$X[m;k] = \sum_{n=0}^{L-1} x[m+n]\, e^{-j\frac{2\pi}{L}nk}$$

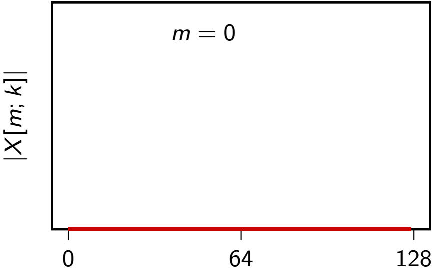
Short-Time Fourier Transform

Idea:

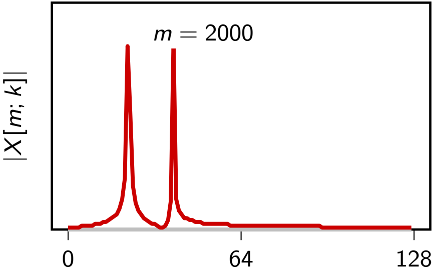
- take small signal pieces of length L
- look at the DFT of each piece:

$$X[m;k] = \sum_{n=0}^{L-1} x[m+n]\, e^{-j\frac{2\pi}{L}nk}$$

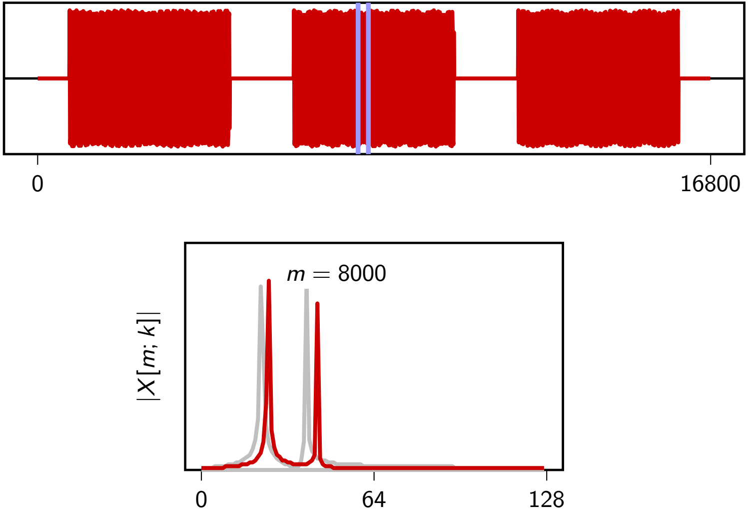
Short-Time Fourier Transform ($L = 256$)



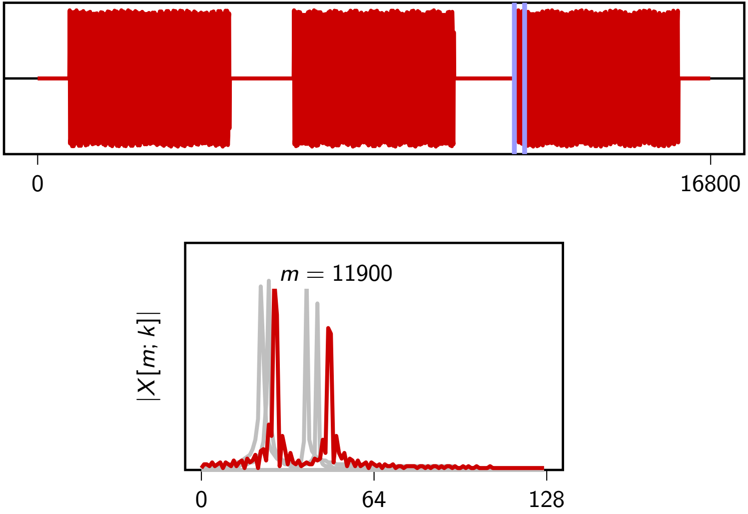
Short-Time Fourier Transform ($L = 256$)



Short-Time Fourier Transform ($L = 256$)



Short-Time Fourier Transform ($L = 256$)



The Spectrogram

Idea:

- color-code the magnitude: dark is small, white is large
- use $10\log_{10}(|X[m;k]|)$ to see better (power in dBs)
- plot spectral slices one after another

The Spectrogram

Idea:

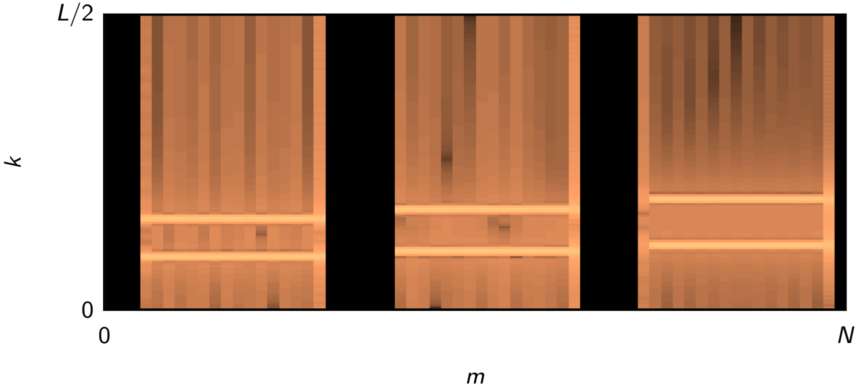
- color-code the magnitude: dark is small, white is large
- use $10\log_{10}(|X[m;k]|)$ to see better (power in dBs)
- plot spectral slices one after another

The Spectrogram

Idea:

- color-code the magnitude: dark is small, white is large
- use $10\log_{10}(|X[m;k]|)$ to see better (power in dBs)
- plot spectral slices one after another

DTMF spectrogram



Labeling the Spectrogram

If we know the "system clock" $F_s = 1/T_s$ we can label the axis

- highest positive frequency: $F_s/2$ Hz
- frequency resolution: F_s/L Hz
- width of time slices: LT_s seconds

Labeling the Spectrogram

If we know the "system clock" $F_s = 1/T_s$ we can label the axis

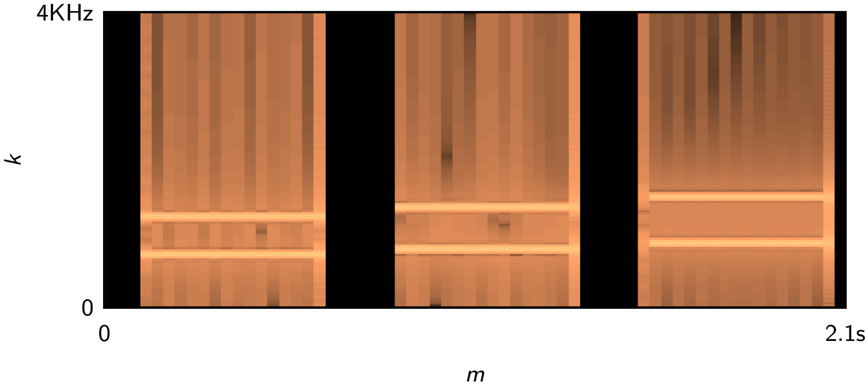
- highest positive frequency: $F_s/2$ Hz
- frequency resolution: F_s/L Hz
- width of time slices: LT_s seconds

Labeling the Spectrogram

If we know the "system clock" $F_s = 1/T_s$ we can label the axis

- highest positive frequency: $F_s/2$ Hz
- frequency resolution: F_s/L Hz
- width of time slices: LT_s seconds

DTMF spectrogram ($F_s = 8000$)



The Spectrogram

Questions:

- width of the analysis window?
- position of the windows (overlapping?)
- shape of the window (weighing the samples)

The Spectrogram

Questions:

- width of the analysis window?
- position of the windows (overlapping?)
- shape of the window (weighing the samples)

The Spectrogram

Questions:

- width of the analysis window?
- position of the windows (overlapping?)
- shape of the window (weighing the samples)

Wideband vs Narrowband

Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

Wideband vs Narrowband

Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

Wideband vs Narrowband

Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

Wideband vs Narrowband

Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

Wideband vs Narrowband

Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

Wideband vs Narrowband

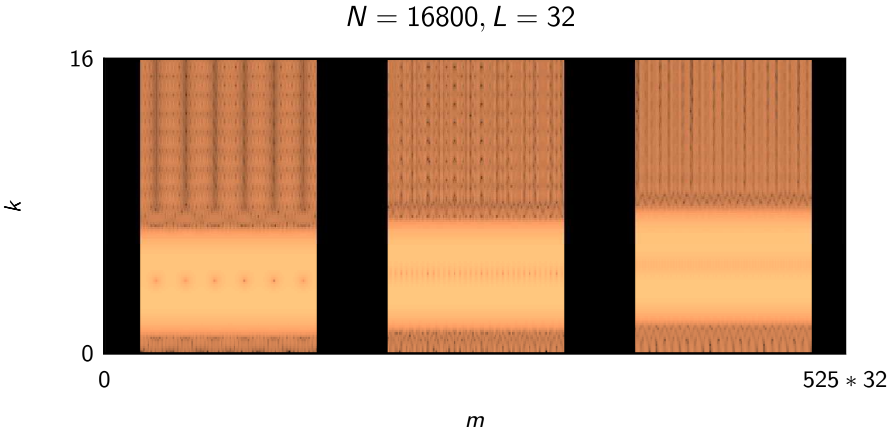
Long window: narrowband spectrogram

- long window $\Rightarrow$ more DFT points $\Rightarrow$ more frequency resolution
- long window $\Rightarrow$ more "things can happen" $\Rightarrow$ less precision in time

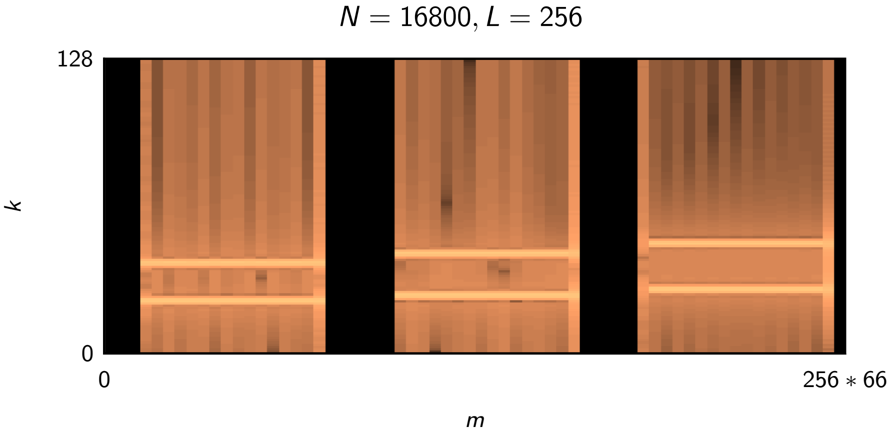
Short window: wideband spectrogram

- short window $\Rightarrow$ many time slices $\Rightarrow$ precise location of transitions
- short window $\Rightarrow$ fewer DFT points $\Rightarrow$ poor frequency resolution

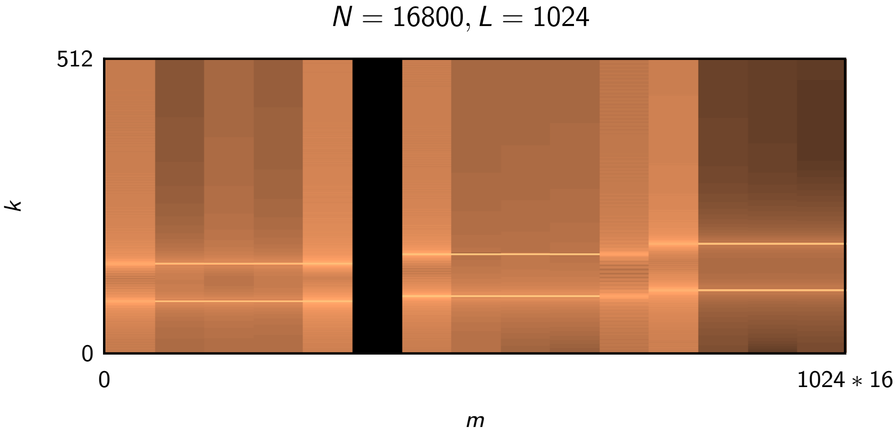
DTMF spectrogram (wideband)



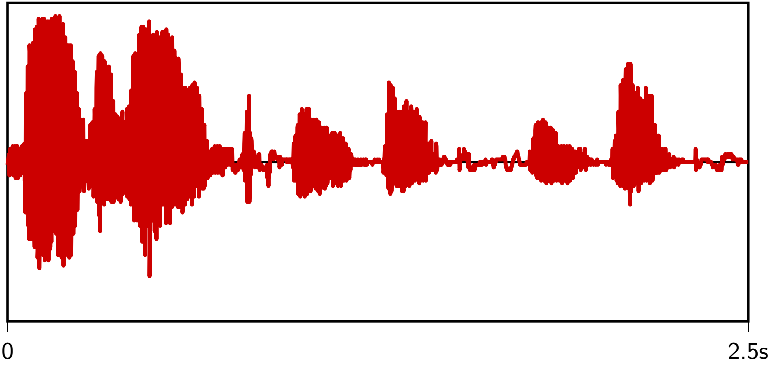
DTMF spectrogram



DTMF spectrogram (narrowband)



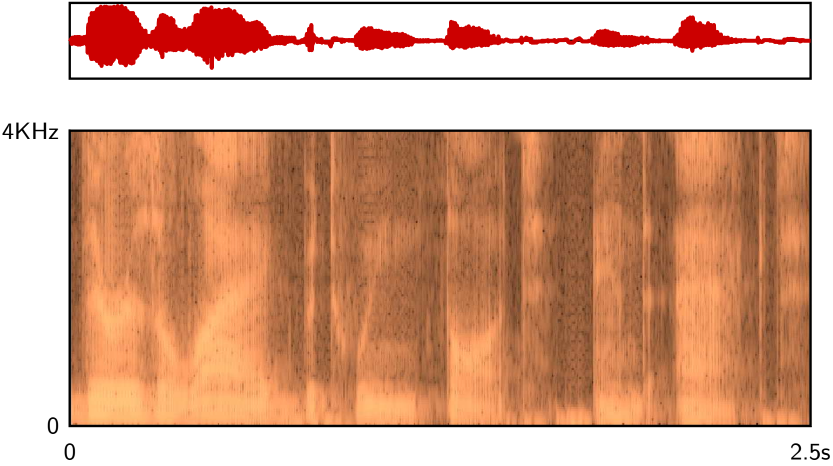
Speech analysis



Play

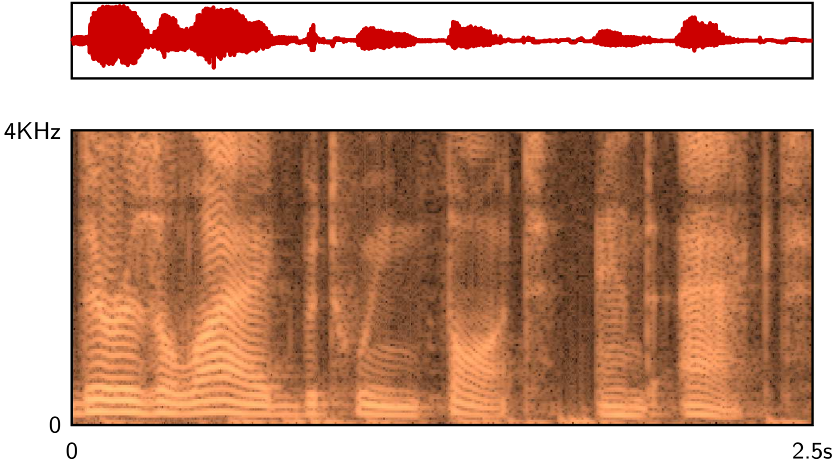
Speech analysis

8ms analysis window (125Hz frequency bins) , 4ms shifts



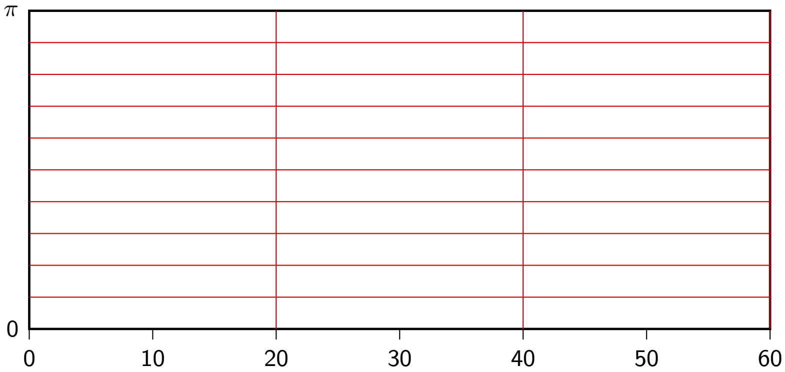
Speech analysis

32ms analysis window (31Hz frequency bins), 4ms shifts



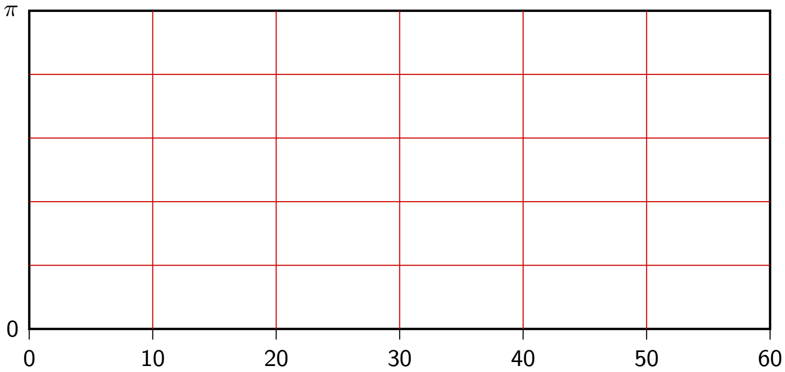
Time-Frequency tiling

$L = 20$

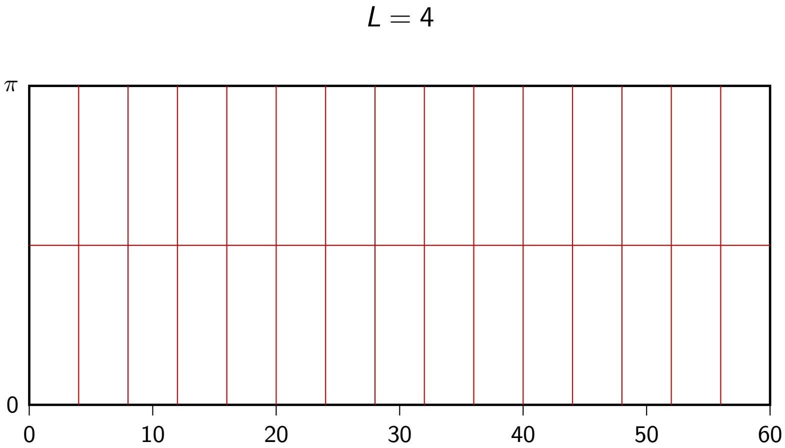


Time-Frequency tiling

$L = 10$



Time-Frequency tiling



Food for thought

- time "resolution" $\Delta t = L$
- frequency "resolution" $\Delta f = 2\pi/L$
- $\Delta t \Delta f = 2\pi$

uncertainty principle!

Food for thought

- time “resolution” $\Delta t = L$
- frequency “resolution” $\Delta f = 2\pi/L$
- $\Delta t \Delta f = 2\pi$

uncertainty principle!

Food for thought

- time “resolution” $\Delta t = L$
- frequency “resolution” $\Delta f = 2\pi/L$
- $\Delta t \Delta f = 2\pi$

uncertainty principle!

Food for thought

- time "resolution" $\Delta t = L$
- frequency "resolution" $\Delta f = 2\pi / L$
- $\Delta t \Delta f = 2\pi$

uncertainty principle!

Even more food for thought

more sophisticated tilings of the time-frequency planes
can be obtained with the *wavelet* transform

the Fast Fourier Transform (FFT)

Overview

- A bit of history: From Gauss to the fastest FFT in the west
- Small DFT matrices
- The Cooley-Tukey FFT
- Decimation-in-Time FFT for length 2^N FFTs
- Conclusions: There are FFTs for any length!

Fourier had the Fourier transform

But Gauss had the FFT all along ;)

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

History

- Gauss computes trigonometric series efficiently in 1805
- Fourier invents Fourier series in 1807
- People start computing Fourier series, and develop tricks
- Good comes up with an algorithm in 1958
- Cooley and Tukey (re)-discover the fast Fourier transform algorithm in 1965 for N a power of a prime
- Winograd combines all methods to give the most efficient FFTs in 1978
- Frigo and Johnson develop the FFTW in 1999

The DFT matrix

- $W_N = e^{-j\frac{2\pi}{N}}$ (or simply W when N is clear from the context)
- powers of N can be taken modulo N, since $W^N = 1$.
- DFT Matrix of size N by N:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{2(N-1)} \\ & & & \dots & & \\ 1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix}$$

The DFT matrix

- $W_N = e^{-j\frac{2\pi}{N}}$ (or simply W when N is clear from the context)
- powers of N can be taken modulo N, since $W^N = 1$.
- DFT Matrix of size N by N:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{2(N-1)} \\ & & & \dots & & \\ 1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix}$$

The DFT matrix

- $W_N = e^{-j\frac{2\pi}{N}}$ (or simply W when N is clear from the context)
- powers of N can be taken modulo N, since $W^N = 1$.
- DFT Matrix of size N by N:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{2(N-1)} \\ & & & \dots & & \\ 1 & W^{N-1} & W^{2(N-1)} & W^{3(N-1)} & \dots & W^{(N-1)^2} \end{bmatrix}$$

Small DFT matrices: $N = 2$

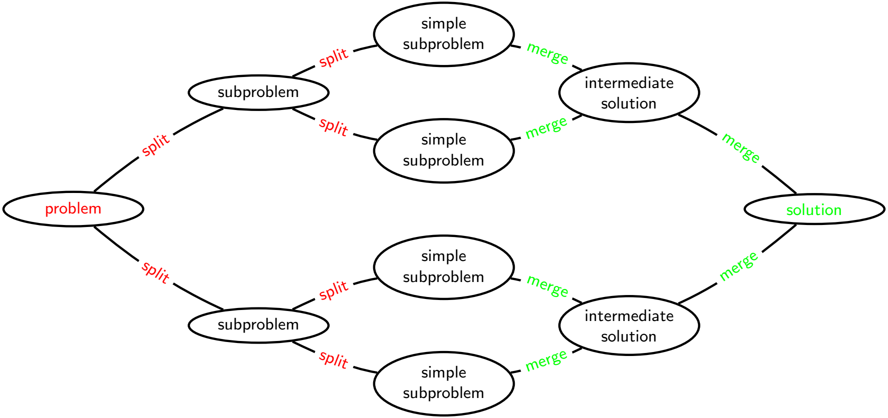
$$\mathbf{W}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Small DFT matrices: $N = 4$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & 1 & W^2 \\ 1 & W^3 & W^2 & W \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Divide et impera - Divide and Conquer (Julius Caesar)

Divide and conquer is a standard attack for developing fast algorithms.



Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N \ \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Assume N even
- Split the problem into two subproblems of size $N/2$; cost is $N^2/4$ each
- *If* the cost to recover the full solution is linear N ...
- ... the divide-and-conquer solution costs $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2

Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N \ \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Assume N even
- Split the problem into two subproblems of size $N/2$; cost is $N^2/4$ each
- *If* the cost to recover the full solution is linear N ...
- ... the divide-and-conquer solution costs $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2

Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N\ \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Assume N even
- Split the problem into two subproblems of size $N/2$; cost is $N^2/4$ each
- *If* the cost to recover the full solution is linear N ...
- ... the divide-and-conquer solution costs $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2

Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N \ \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Assume N even
- Split the problem into two subproblems of size $N/2$; cost is $N^2/4$ each
- *If* the cost to recover the full solution is linear N ...
- ... the divide-and-conquer solution costs $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2

Divide and Conquer for DFT - One step

Recall: computing $\mathbf{X} = \mathbf{W}_N\ \mathbf{x}$ has complexity $O(N^2)$.

Idea:

- Assume N even
- Split the problem into two subproblems of size $N/2$; cost is $N^2/4$ each
- *If* the cost to recover the full solution is linear N ...
- ... the divide-and-conquer solution costs $N^2/2 + N$ for one step
- For $N \geq 4$ this is better than N^2

Divide and Conquer for DFT - One step

Graphically

- Split DFT input into 2 pieces of size $N/2$

Divide and Conquer for DFT - One step

Graphically

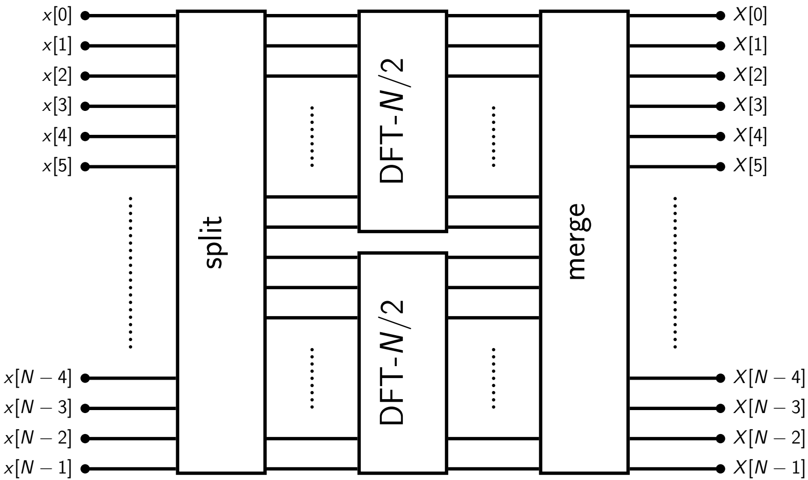
- Split DFT input into 2 pieces of size $N/2$
- Compute two DFT's of size $N/2$

Divide and Conquer for DFT - One step

Graphically

- Split DFT input into 2 pieces of size $N/2$
- Compute two DFT's of size $N/2$
- Merge the two results

Divide and Conquer for DFT - One step



Divide and Conquer for DFT - Multiple steps

Idea: if $N = 2^K$, divide and conquer can be reapplied!

- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- Assume complexity to recover the full solution still linear, e.g. N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Divide and Conquer for DFT - Multiple steps

Idea: if $N = 2^K$, divide and conquer can be reapplied!

- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- Assume complexity to recover the full solution still linear, e.g. N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Divide and Conquer for DFT - Multiple steps

Idea: if $N = 2^K$, divide and conquer can be reapplied!

- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- Assume complexity to recover the full solution still linear, e.g. N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Divide and Conquer for DFT - Multiple steps

Idea: if $N = 2^K$, divide and conquer can be reapplied!

- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- Assume complexity to recover the full solution still linear, e.g. N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Divide and Conquer for DFT - Multiple steps

Idea: if $N = 2^K$, divide and conquer can be reapplied!

- Cut the two problems of size $N/2$ into 4 problems of size $N/4$
- Assume complexity to recover the full solution still linear, e.g. N at each step
- You can do this $\log_2 N - 1 = K - 1$ times, until problem of size 2 is obtained
- The divide-and-conquer solution has therefore complexity of order $N \log_2 N$
- For $N \geq 4$ this is much better than N^2!

Divide and Conquer for DFT - Multiple steps

N	N^2	$N \log N$
10	100	10
100	10,000	200
1000	1M	3000
10,000	100M (10^8)	40,000 $(4 \cdot 10^4)$
100,000	10B (10^{10})	500,000 $(5 \cdot 10^5)$

Divide and Conquer for DFT - Multiple steps

Graphically

- Split DFT input into 2, 4 and 8 pieces of sizes $N/2$, $N/4$ and $N/8$, respectively
- Compute 8 DFT's of size $N/8$
- Merge the results successively into DFT's of size $N/4$, $N/2$ and finally N

Divide and Conquer for DFT - Multiple steps

Graphically

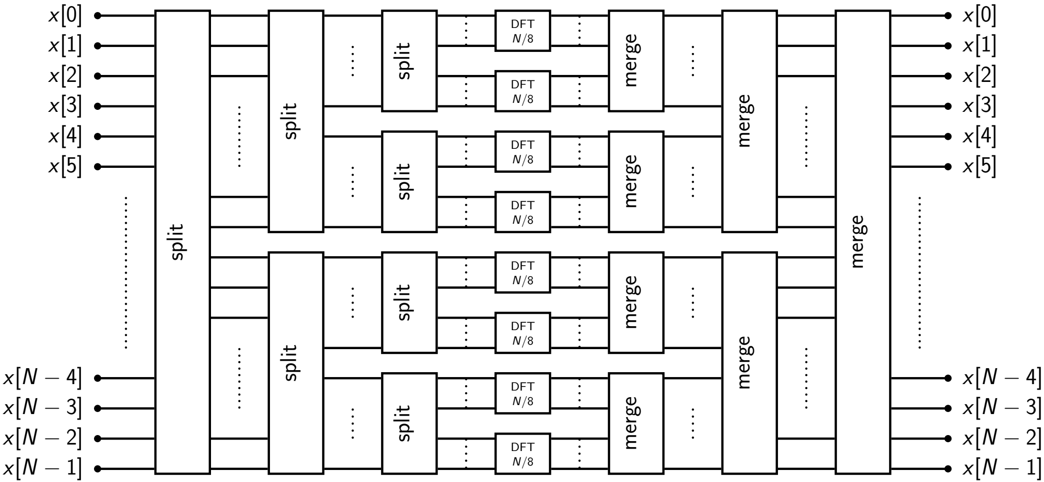
- Split DFT input into 2, 4 and 8 pieces of sizes $N/2$, $N/4$ and $N/8$, respectively
- Compute 8 DFT's of size $N/8$
- Merge the results successively into DFT's of size $N/4$, $N/2$ and finally N

Divide and Conquer for DFT - Multiple steps

Graphically

- Split DFT input into 2, 4 and 8 pieces of sizes $N/2$, $N/4$ and $N/8$, respectively
- Compute 8 DFT's of size $N/8$
- Merge the results successively into DFT's of size $N/4$, $N/2$ and finally N

Divide and Conquer for DFT - Multiple steps



Divide and Conquer for DFT- Analysis of DIT

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk}, \qquad k = 0, 1, \dots, N-1, \quad W_N = e^{-j\frac{2\pi}{N}}$$

Idea (a good guess is half of the answer!):

- break input into even and odd indexed terms (so-called "decimation in time"):

$$x[n], \quad n = 0, 1, \dots, N-1 \;\longrightarrow\; x[2n] \text{ and } x[2n+1], \quad n = 0, \dots, N/2-1$$

- break output into first and second half

$$X[k], \quad k = 0, 1, \dots, N-1 \;\longrightarrow\; X[k] \text{ and } X[k+N/2], \quad k = 0, \dots, N/2-1$$

Divide and Conquer for DFT- Analysis of DIT

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk}, \qquad k = 0, 1, \ldots, N-1, \quad W_N = e^{-j\frac{2\pi}{N}}$$

Idea (a good guess is half of the answer!):

- break input into even and odd indexed terms (so-called "decimation in time"):

$$x[n], \quad n = 0, 1, \ldots, N-1 \;\longrightarrow\; x[2n] \text{ and } x[2n+1], \quad n = 0, \ldots, N/2-1$$

- break output into first and second half

$$X[k], \quad k = 0, 1, \ldots, N-1 \;\longrightarrow\; X[k] \text{ and } X[k+N/2], \quad k = 0, \ldots, N/2-1$$

Divide and Conquer for DFT- Analysis of DIT

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{nk}, \qquad k = 0, 1, \dots, N-1, \quad W_N = e^{-j\frac{2\pi}{N}}$$

Idea (a good guess is half of the answer!):

- break input into even and odd indexed terms (so-called "decimation in time"):

$$x[n], \quad n = 0, 1, \dots, N-1 \;\longrightarrow\; x[2n] \text{ and } x[2n+1], \quad n = 0, \dots, N/2-1$$

- break output into first and second half

$$X[k], \quad k = 0, 1, \dots, N-1 \;\longrightarrow\; X[k] \text{ and } X[k+N/2], \quad k = 0, \dots, N/2-1$$

Divide and Conquer for DFT- Analysis of DIT

Consider even and odd inputs separately:

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

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{2nk+k}$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X_A[k] + W_N^k\, X_B[k], \qquad k = 0, 1, \ldots, N-1$$

Divide and Conquer for DFT- Analysis of DIT

Consider even and odd inputs separately:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{(2n+1)k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{2nk+k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X_A[k] + W_N^k X_B[k], \qquad k = 0, 1, \ldots, N-1
\end{aligned}
$$

Divide and Conquer for DFT- Analysis of DIT

Consider even and odd inputs separately:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{(2n+1)k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{2nk+k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X_A[k] + W_N^k X_B[k], \qquad k = 0, 1, \ldots, N-1
\end{aligned}
$$

Divide and Conquer for DFT- Analysis of DIT

Consider even and odd inputs separately:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{(2n+1)k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{2nk+k} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X_A[k] + W_N^k\, X_B[k], \qquad k = 0, 1, \ldots, N-1
\end{aligned}
$$

Divide and Conquer for DFT- Analysis of DIT

hmmm, we haven't gained much so far:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $N(N/2 + N/2 + 1) \approx N^2$
- but here comes the trick!

Divide and Conquer for DFT- Analysis of DIT

hmmm, we haven't gained much so far:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $N(N/2 + N/2 + 1) \approx N^2$
- but here comes the trick!

Divide and Conquer for DFT- Analysis of DIT

hmmm, we haven't gained much so far:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $N(N/2 + N/2 + 1) \approx N^2$
- but here comes the trick!

Divide and Conquer for DFT- Analysis of DIT

hmmm, we haven't gained much so far:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $N(N/2 + N/2 + 1) \approx N^2$
- but here comes the trick!

Divide and Conquer for DFT- Analysis of DIT

Consider now the first and second half of the outputs separately:

$$X[k] = X_A[k] + W^k\, X_B[k]$$

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

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

$$= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk}$$

$$= X_A[k] - W_N^k\, X_B[k], \qquad k = 0, 1, \ldots, N/2-1$$

Divide and Conquer for DFT- Analysis of DIT

Consider now the first and second half of the outputs separately:

$$
\begin{aligned}
X[k] &= X_A[k] + W^k X_B[k] \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
X[k+N/2] &= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{n(k+N/2)} + W_N^{k+N/2} \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{n(k+N/2)} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X_A[k] - W_N^k X_B[k], \qquad k = 0, 1, \ldots, N/2-1
\end{aligned}
$$

Divide and Conquer for DFT- Analysis of DIT

Consider now the first and second half of the outputs separately:

$$
\begin{aligned}
X[k] &= X_A[k] + W^k\, X_B[k] \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} + W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
X[k+N/2] &= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{n(k+N/2)} + W_N^{k+N/2} \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{n(k+N/2)} \\
&= \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{nk} - W^k \sum_{n=0}^{N/2-1} x[2n+1]\, W_{N/2}^{nk} \\
&= X_A[k] - W_N^k\, X_B[k], \qquad k = 0, 1, \ldots, N/2-1
\end{aligned}
$$

Divide and Conquer for DFT- Analysis of DIT

so the trick is that we only need to compute for half the range of k:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $(N/2)(N/2 + N/2 + 1) \approx N^2/2$
- the rest is just sums and differences

Divide and Conquer for DFT- Analysis of DIT

so the trick is that we only need to compute for half the range of k:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $(N/2)(N/2 + N/2 + 1) \approx N^2/2$
- the rest is just sums and differences

Divide and Conquer for DFT- Analysis of DIT

so the trick is that we only need to compute for half the range of k:

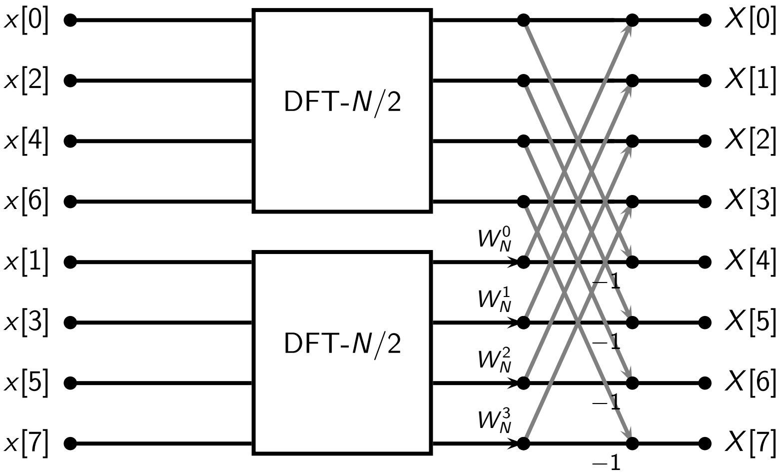
- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $(N/2)(N/2 + N/2 + 1) \approx N^2/2$
- the rest is just sums and differences

Divide and Conquer for DFT- Analysis of DIT

so the trick is that we only need to compute for half the range of k:

- both $X_A[k]$ and $X_B[k]$ require $N/2$ multiplications
- multiplying the second DFT by W_N^k requires another multiplication
- to compute for all k we need $(N/2)(N/2 + N/2 + 1) \approx N^2/2$
- the rest is just sums and differences

Divide and Conquer for DFT- Analysis of DIT



Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what is the complexity now?

- Split DFT input into 2 pieces of size N/2: free!
- Compute 2 DFT-N/2: twice $(N/2)^2$, or $N^2/2$
- Merge the two results: multiplication by $N/2$ complex numbers W^k
- Total: $N^2/2 + N/2$ which is indeed smaller than N^2 for any $N \geq 4$,
- In general, about half the complexity of the initial problem!

Divide and Conquer for DFT- Analysis of DIT

So, what if we repeat the process?

- Go until DFT-2, since that is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $(N/2)(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Divide and Conquer for DFT- Analysis of DIT

So, what if we repeat the process?

- Go until DFT-2, since that is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $(N/2)(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Divide and Conquer for DFT- Analysis of DIT

So, what if we repeat the process?

- Go until DFT-2, since that is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $(N/2)(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Divide and Conquer for DFT- Analysis of DIT

So, what if we repeat the process?

- Go until DFT-2, since that is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $(N/2)(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Divide and Conquer for DFT- Analysis of DIT

So, what if we repeat the process?

- Go until DFT-2, since that is trivial (sum and difference)
- Requires $\log_2 N - 1$ steps
- Each step requires a merger of order $N/2$ multiplications and N additions
- Total: $(N/2)(\log_2 N - 1)$ multiplications and $N \log_2 N$ additions

Key Result: **A DFT of size N requires order $N \log_2 N$ operations!**

Matrix factorization view of DFT, N $= 4$

- Separate even and odd samples
- Compute two DFT's of size 2 having output $X'[k]$ and $X''[k]$
- Compute sum and difference of $X'[k]$ and $W^k X''[k]$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This uses 8 additions and no multiplications!

Matrix factorization view of DFT, N $= 4$

- Separate even and odd samples
- Compute two DFT's of size 2 having output $X'[k]$ and $X"[k]$
- Compute sum and difference of $X'[k]$ and $W^k X"[k]$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This uses 8 additions and no multiplications!

Matrix factorization view of DFT, N = 4

- Separate even and odd samples
- Compute two DFT's of size 2 having output $X'[k]$ and $X"[k]$
- Compute sum and difference of $X'[k]$ and $W^k X"[k]$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This uses 8 additions and no multiplications!

Matrix factorization view of DFT, N $= 4$

- Separate even and odd samples
- Compute two DFT's of size 2 having output $X'[k]$ and $X"[k]$
- Compute sum and difference of $X'[k]$ and $W^k X"[k]$

$$\mathbf{W}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -j \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This uses 8 additions and no multiplications!

Matrix factorization view of DFT, N $= 8$, $1/8$

Now this is going to be big...

Too big for a single slide!

$$\mathbf{W}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & W^3 & \dots & W^7 \\ 1 & W^2 & W^4 & W^6 & \dots & W^{14} \\ & & & \dots & & \\ 1 & W^7 & W^{14} & W^{21} & \dots & W^{49} \end{bmatrix} = \dots$$

Matrix factorization view of DFT, N $= 8$, $2/8$

Step 1: separate even from odd indexed samples
Call this $\mathbf{D}_8$ for decimation of size 8

$$\mathbf{D}_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This requires no arithmetic operations!

Matrix factorization view of DFT, $N = 8$, 3/8

Step 2: Compute two DFTs of size $N/2$ on the even and on the odd indexed samples
Each submatrix is $\mathbf{W}_4$, and the matrix is block diagonal, where $\mathbf{0_4}$ stands for a matrix of 0's

$$\begin{bmatrix} \mathbf{W_4} & \mathbf{0_4} \\ \mathbf{0_4} & \mathbf{W_4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & & & & \\ 1 & j & -1 & -j & & & & \\ 1 & -1 & 1 & -1 & & & & \\ 1 & -j & -1 & j & & & & \\ & & & & 1 & 1 & 1 & 1 \\ & & & & 1 & j & -1 & -j \\ & & & & 1 & -1 & 1 & -1 \\ & & & & 1 & -j & -1 & j \end{bmatrix}$$

This requires two DFT-4, or a total of 16 additions!

Matrix factorization view of DFT, N $= 8$, $4/8$

Step 3: Multiply output of second DFT of size 4 by W^k
This is a diagonal matrix, with $\mathbf{I_4}$ for the identity of size 4,

$$\begin{bmatrix} \mathbf{I_4} & \mathbf{0_4} \\ \mathbf{0_4} & \boldsymbol{\Lambda}_4 \end{bmatrix} = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & W & & \\ & & & & & & W^2 & \\ & & & & & & & W^3 \end{bmatrix} \text{ where } \quad \boldsymbol{\Lambda}_4 = \begin{bmatrix} 1 & & & \\ & W & & \\ & & W^2 & \\ & & & W^3 \end{bmatrix}$$

This requires 2 multiplications ($W^2 = -j$ is free)

Matrix factorization view of DFT, N $= 8$, $5/8$

Step 4: Recombine final output $X[k]$ and $X[k+N/2]$ by sum and difference, $\mathbf{S}_8$

$$\mathbf{S}_8 = \begin{bmatrix} \mathbf{I_4} & \mathbf{I_4} \\ \mathbf{I_4} & -\mathbf{I_4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}$$

This requires 8 additions!

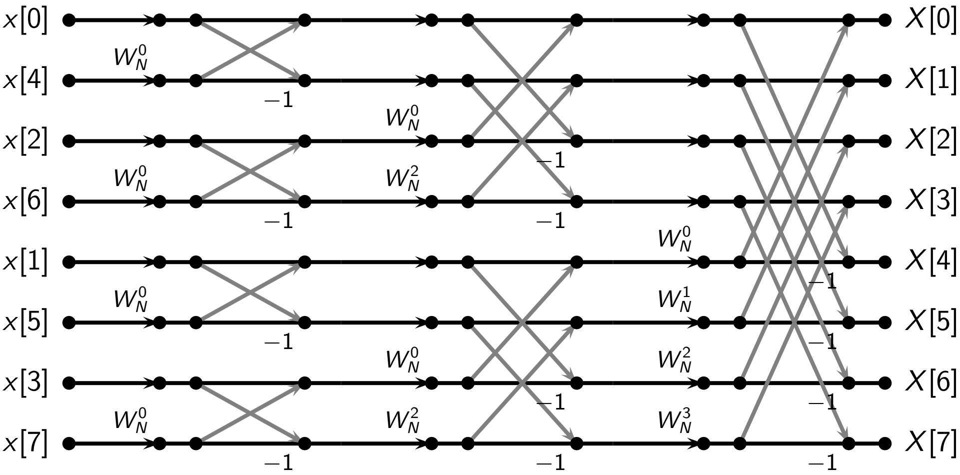
Matrix factorization view of DFT, $N = 8$, 6/8

In total:
Product of 4 matrices

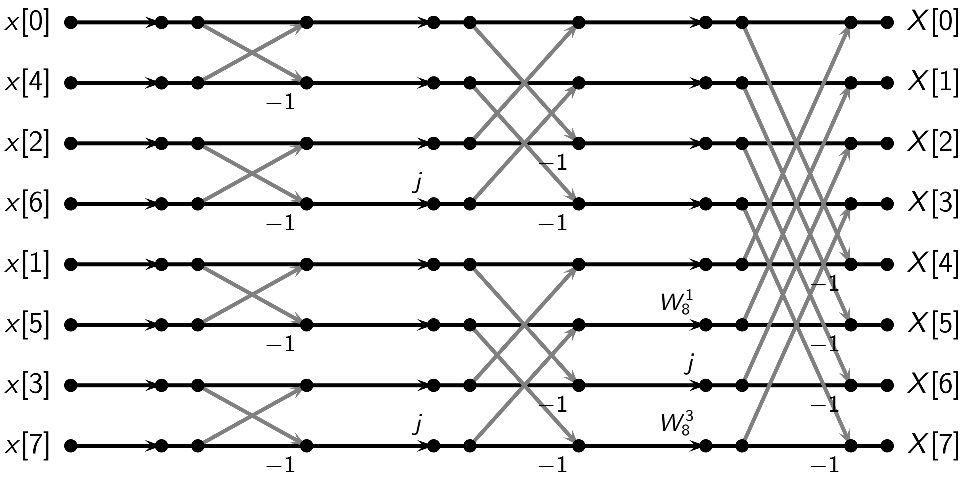
$$\mathbf{W}_8 = \begin{bmatrix} \mathbf{I}_4 & \mathbf{I}_4 \\ \mathbf{I}_4 & -\mathbf{I}_4 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{\Lambda}_4 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{W}_4 & \mathbf{0}_4 \\ \mathbf{0}_4 & \mathbf{W}_4 \end{bmatrix} \cdot \mathbf{D}_8$$

This requires 24 additions and 2 multiplications!

Flowgraph view of FFT, $N = 8$



Flowgraph view of FFT, $N = 8$



Matrix factorization view of DFT, $N = 8$, 8/8

Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

Matrix factorization view of DFT, $N = 8$, 8/8

Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

Matrix factorization view of DFT, $N = 8$, 8/8

Is this a big deal?

- In image processing (e.g. digital photography) one takes block of 8 by 8 pixels
- One computes a transform (called DCT) similar to a DFT
- It has a fast algorithm inspired by what we just saw

Some examples

image processing (JPEG compression)

- image is divided into 8×8-pixel blocks
- DFT performed on rows and columns: 16 8-point DFTs
- direct computation: $16 \times 8^2 = 1024$ multiplications
- FFT: $16 \times 2 = 32$ multiplications

Some examples

image processing (JPEG compression)

- image is divided into 8×8-pixel blocks
- DFT performed on rows and columns: 16 8-point DFTs
- direct computation: $16 \times 8^2 = 1024$ multiplications
- FFT: $16 \times 2 = 32$ multiplications

Some examples

image processing (JPEG compression)

- image is divided into 8×8-pixel blocks
- DFT performed on rows and columns: 16 8-point DFTs
- direct computation: $16 \times 8^2 = 1024$ multiplications
- FFT: $16 \times 2 = 32$ multiplications

Some examples

image processing (JPEG compression)

- image is divided into 8×8-pixel blocks
- DFT performed on rows and columns: 16 8-point DFTs
- direct computation: $16 \times 8^2 = 1024$ multiplications
- FFT: $16 \times 2 = 32$ multiplications

Some examples

audio processing (MP3 compression)

- audio is split into 1152-point frames
- direct DFT computation: $1.3 \cdot 10^6$ multiplications
- FFT: 3500 multiplications

Some examples

audio processing (MP3 compression)

- audio is split into 1152-point frames
- direct DFT computation: $1.3 \cdot 10^6$ multiplications
- FFT: 3500 multiplications

Some examples

audio processing (MP3 compression)

- audio is split into 1152-point frames
- direct DFT computation: $1.3 \cdot 10^6$ multiplications
- FFT: 3500 multiplications

Conclusions

Don't worry, be happy!

- The Cooley-Tukey is the most popular algorithm, mostly for $N = 2^N$
- Note that there is always a good FFT algorithm around the corner
- Do not zero-pad to lengthen a vector to have a size equal to a power of 2
- There are good packages out there (e.g. Fastest Fourier Transform in the West, SPIRAL)
- It does make a BIG difference!