Overview

The Fast Fourier Transform

- First used by Carl Friedrich Gauss but went unnoticed.
- James Cooley and John Tukey independently discovered the Cooley-Tukey algorithm in 1965.
- Improves the $O(N^2)$ running time it takes to perform DFT to $O(N \log_2 N)$ (regarding complex addition and multiplication).
-

	n=1000	n=10^6	n=10^9
n^2	10^6	10^{12}	10^{18}
$n \log_2 n$	~10^4	~$2 \cdot 10^7$	~$3 \cdot 10^{10}$

The Fast Fourier Transform

- First used by Carl Friedrich Gauss but went unnoticed.
- James Cooley and John Tukey independently discovered the Cooley-Tukey algorithm in 1965.
- Improves the $O(N^2)$ running time it takes to perform DFT to $O(N \log_2 N)$ (regarding complex addition and multiplication).
-

	n=1000	n=10^6	n=10^9
n^2	10^6	10^{12}	10^{18}
$n \log_2 n$	~10^4	~$2 \cdot 10^7$	~$3 \cdot 10^{10}$

The Fast Fourier Transform

- First used by Carl Friedrich Gauss but went unnoticed.
- James Cooley and John Tukey independently discovered the Cooley-Tukey algorithm in 1965.
- Improves the $O(N^2)$ running time it takes to perform DFT to $O(N \log_2 N)$ (regarding complex addition and multiplication).
-

	n=1000	n=10^6	n=10^9
n^2	10^6	10^{12}	10^{18}
$n \log_2 n$	~10^4	~$2 \cdot 10^7$	~$3 \cdot 10^{10}$

The Fast Fourier Transform

- First used by Carl Friedrich Gauss but went unnoticed.
- James Cooley and John Tukey independently discovered the Cooley-Tukey algorithm in 1965.
- Improves the $O(N^2)$ running time it takes to perform DFT to $O(N \log_2 N)$ (regarding complex addition and multiplication).
-

	n=1000	n=10^6	n=10^9
n^2	10^6	10^{12}	10^{18}
$n \log_2 n$	~10^4	~$2 \cdot 10^7$	~$3 \cdot 10^{10}$

OVERVIEW

A degree $N-1$ polynomial is given by
$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$.

Other representations:

- **Coefficient vector:** A vector $(a_0, a_1, \cdots, a_{N-1})$ in a space of monomials.
- **Roots:** Given roots of a polynomial, we can write $P(x) = k \prod_{n=0}^{N-1} (x - r_n)$ for some constant k.
- **Samples:** We can also take N distinct pairs of $(x_n, y_n = P(x_n))$ to uniquely determine the polynomial $P(x)$ by Lagrange polynomial interpolation.

A degree $N-1$ polynomial is given by
$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$.

Other representations:

- **Coefficient vector:** A vector $(a_0, a_1, \cdots, a_{N-1})$ in a space of monomials.
- **Roots:** Given roots of a polynomial, we can write $P(x) = k \prod_{n=0}^{N-1} (x - r_n)$ for some constant k.
- **Samples:** We can also take N distinct pairs of $(x_n, y_n = P(x_n))$ to uniquely determine the polynomial $P(x)$ by Lagrange polynomial interpolation.

A degree $N-1$ polynomial is given by
$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$.

Other representations:

- **Coefficient vector:** A vector $(a_0, a_1, \cdots, a_{N-1})$ in a space of monomials.
- **Roots:** Given roots of a polynomial, we can write $P(x) = k \prod_{n=0}^{N-1} (x - r_n)$ for some constant k.
- **Samples:** We can also take N distinct pairs of $(x_n, y_n = P(x_n))$ to uniquely determine the polynomial $P(x)$ by Lagrange polynomial interpolation.

There are three main things we want to do with polynomials, namely:

- **Evaluation:** Evalutae the polynomial at a point in its domain.
- **Addition:** Add two polynomials together.
- **Multiplication:** Multiply two polynomials together. Note that polynomial multiplication in the coefficient vector form actually corresponds to the finite discrete convolution of two vectors in their inner product space.

There are three main things we want to do with polynomials, namely:

- **Evaluation:** Evalutae the polynomial at a point in its domain.
- **Addition:** Add two polynomials together.
- **Multiplication:** Multiply two polynomials together. Note that polynomial multiplication in the coefficient vector form actually corresponds to the finite discrete convolution of two vectors in their inner product space.

There are three main things we want to do with polynomials, namely:

- **Evaluation:** Evalutae the polynomial at a point in its domain.
- **Addition:** Add two polynomials together.
- **Multiplication:** Multiply two polynomials together. Note that polynomial multiplication in the coefficient vector form actually corresponds to the finite discrete convolution of two vectors in their inner product space.

Running-time for each operation on degree $N-1$ polynomials			
	Coeffcients	Roots	Samples
Evaluation	$O(N)$	$O(N)$	$O(N^2)$
Addition	$O(N)$	N/A	$O(N)$
Multiplication	$O(N^2)$	$O(N)$	$O(N)$

Objective: convert coefficients to samples efficiently as to avoid $O(N^2)$ running-time.

OVERVIEW

We can sample $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$ by doing

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix}.$$

This calculates N samples at N distinct points in the set $X = \{x_0, \cdots, x_{N-1}\}$ with a running-time of $O(N^2)$...

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Divide and conquer! (this is where we assume N is a power of 2)

To evaluate our polynomial $P(x)$ at the point x, we can define the following:

- $P_{\text{even}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k}x^k$
- $P_{\text{odd}}(x) = \sum_{k=0}^{[(N-1)/2]} a_{2k+1}x^k$
- $P(x) = P_{\text{even}}(x^2) + x \cdot P_{\text{odd}}(x^2)$

Do this for every x we want to sample... N of them.

Let T measure the running-time of calculating $P(x)$ for every x in X, then we have that

$$\begin{aligned} T(N,|X|) &= 2T(\frac{N}{2},|X|) + O(|X|) \\ &= 2\left[2T(\frac{N}{4},|X|) + O(|X|)\right] + O(|X|) \\ &\vdots \\ &= 2^{\log_2 N} T(1,|\mathbf{X}|) + \sum_{l=0}^{\log_2 N} 2^l O(|X|) \\ &= \mathbf{N^2} + O(|X|\log_2 N). \end{aligned}$$

$O(N^2)$ again!

Root cause? This algorithm chokes when it has to evaluate monomials **for every element in the set**.

Let T measure the running-time of calculating $P(x)$ for every x in X, then we have that

$$\begin{aligned} T(N, |X|) &= 2T(\frac{N}{2}, |X|) + O(|X|) \\ &= 2\left[2T(\frac{N}{4}, |X|) + O(|X|)\right] + O(|X|) \\ &\vdots \\ &= 2^{\log_2 N} T(1, |\mathbf{X}|) + \sum_{l=0}^{\log_2 N} 2^l O(|X|) \\ &= \mathbf{N^2} + O(|X| \log_2 N). \end{aligned}$$

$O(N^2)$ again!

Root cause? This algorithm chokes when it has to evaluate monomials **for every element in the set**.

Let T measure the running-time of calculating $P(x)$ for every x in X, then we have that

$$\begin{aligned} T(N,|X|) &= 2T(\frac{N}{2},|X|)+O(|X|) \\ &= 2\left[2T(\frac{N}{4},|X|)+O(|X|)\right]+O(|X|) \\ &\vdots \\ &= 2^{\log_2 N}T(1,|\mathbf{X}|)+\sum_{l=0}^{\log_2 N}2^l O(|X|) \\ &= \mathbf{N^2}+O(|X|\log_2 N). \end{aligned}$$

$O(N^2)$ again!

Root cause? This algorithm chokes when it has to evaluate monomials **for every element in the set**.

OVERVIEW

What if there is a set that shrinks in size whenever we square each element?

$$
\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}
$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.
Key property: any such root computes to 1 when raised to the Nth power.

What if there is a set that shrinks in size whenever we square each element?

$$
\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}
$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.
Key property: any such root computes to 1 when raised to the Nth power.

What if there is a set that shrinks in size whenever we square each element?

$$
\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}
$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.
Key property: any such root computes to 1 when raised to the Nth power.

What if there is a set that shrinks in size whenever we square each element?

$$\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.
Key property: any such root computes to 1 when raised to the Nth power.

What if there is a set that shrinks in size whenever we square each element?

$$\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.

Key property: any such root computes to 1 when raised to the Nth power.

What if there is a set that shrinks in size whenever we square each element?

$$
\begin{aligned}
\{1\} &= \{e^{2\pi i \frac{1}{1}}\}, N = 1 \\
\{-1, 1\} &= \{e^{2\pi i \frac{1}{2}}, e^{2\pi i \frac{2}{2}}\}, N = 2 \\
\{i, -1, -i, 1\} &= \{e^{2\pi i \frac{1}{4}}, e^{2\pi i \frac{2}{4}}, e^{2\pi i \frac{3}{4}}, e^{2\pi i \frac{4}{4}}\}, N = 4 \\
&\vdots
\end{aligned}
$$

Eventually end up with **the Nth roots of unity** $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$.
Key property: any such root computes to 1 when raised to the Nth power.

OVERVIEW

Why not sample our polynomial at the Nth roots of unity?

$$\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + O(N) \\
&= 2\left[2T(\frac{N}{4}) + O(\frac{N}{2})\right] + O(N) \\
&\vdots \\
&= 2^{\log_2 N} T(1) + \sum_{l=0}^{\log_2 N} 2^l O(\frac{N}{2^l}) \\
&= N + O(N \log_2 N)
\end{aligned}$$

Hello, $O(N \log_2 N)$!

But what does converting polynomial forms have to do with FFT?

Why not sample our polynomial at the Nth roots of unity?

$$
\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + O(N) \\
&= 2\left[2T(\frac{N}{4}) + O(\frac{N}{2})\right] + O(N) \\
&\vdots \\
&= 2^{\log_2 N} T(1) + \sum_{l=0}^{\log_2 N} 2^l O(\frac{N}{2^l}) \\
&= N + O(N \log_2 N)
\end{aligned}
$$

Hello, $O(N \log_2 N)$!

But what does converting polynomial forms have to do with FFT?

Why not sample our polynomial at the Nth roots of unity?

$$\begin{aligned} T(N) &= 2T(\frac{N}{2}) + O(N) \\ &= 2\left[2T(\frac{N}{4}) + O(\frac{N}{2})\right] + O(N) \\ &\vdots \\ &= 2^{\log_2 N} T(1) + \sum_{l=0}^{\log_2 N} 2^l O(\frac{N}{2^l}) \\ &= N + O(N \log_2 N) \end{aligned}$$

Hello, $O(N \log_2 N)$!

But what does converting polynomial forms have to do with FFT?

Why not sample our polynomial at the Nth roots of unity?

$$
\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + O(N) \\
&= 2\left[2T(\frac{N}{4}) + O(\frac{N}{2})\right] + O(N) \\
&\vdots \\
&= 2^{\log_2 N} T(1) + \sum_{l=0}^{\log_2 N} 2^l O(\frac{N}{2^l}) \\
&= N + O(N \log_2 N)
\end{aligned}
$$

Hello, $O(N \log_2 N)$!
But what does converting polynomial forms have to do with FFT?

Overview

Recall our choice to sample the polynomial on $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$, this gives us the following transformation:

$$\begin{pmatrix} 1 & (e^{2\pi i \frac{0}{N}})^1 & (e^{2\pi i \frac{0}{N}})^2 & \cdots & (e^{2\pi i \frac{0}{N}})^{N-1} \\ 1 & (e^{2\pi i \frac{1}{N}})^1 & (e^{2\pi i \frac{1}{N}})^1 & \cdots & (e^{2\pi i \frac{1}{N}})^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (e^{2\pi i \frac{N-1}{N}})^1 & (e^{2\pi i \frac{N-1}{N}})^2 & \cdots & (e^{2\pi i \frac{N-1}{N}})^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}.$$

Now recall again the discrete Fourier coefficients $A_n = \dfrac{1}{N}\displaystyle\sum_{k=0}^{N-1} F(k)(e^{-2\pi i \frac{n}{N}})^k$ for n running from 0 to $N-1$.

Replace a_k with $\frac{1}{N}F(k)$ and take the conjugate of the DFT matrix then we have Cooley and Tukey's radix-2 FFT.

Recall our choice to sample the polynomial on $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$, this gives us the following transformation:

$$\begin{pmatrix} 1 & (e^{2\pi i \frac{0}{N}})^1 & (e^{2\pi i \frac{0}{N}})^2 & \cdots & (e^{2\pi i \frac{0}{N}})^{N-1} \\ 1 & (e^{2\pi i \frac{1}{N}})^1 & (e^{2\pi i \frac{1}{N}})^1 & \cdots & (e^{2\pi i \frac{1}{N}})^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (e^{2\pi i \frac{N-1}{N}})^1 & (e^{2\pi i \frac{N-1}{N}})^2 & \cdots & (e^{2\pi i \frac{N-1}{N}})^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}.$$

Now recall again the discrete Fourier coefficients

$A_n = \dfrac{1}{N} \displaystyle\sum_{k=0}^{N-1} F(k)(e^{-2\pi i \frac{n}{N}})^k$ for n running from 0 to $N-1$.

Replace a_k with $\frac{1}{N}F(k)$ and take the conjugate of the DFT matrix then we have Cooley and Tukey's radix-2 FFT.

Recall our choice to sample the polynomial on $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$, this gives us the following transformation:

$$\begin{pmatrix} 1 & (e^{2\pi i \frac{0}{N}})^1 & (e^{2\pi i \frac{0}{N}})^2 & \cdots & (e^{2\pi i \frac{0}{N}})^{N-1} \\ 1 & (e^{2\pi i \frac{1}{N}})^1 & (e^{2\pi i \frac{1}{N}})^1 & \cdots & (e^{2\pi i \frac{1}{N}})^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (e^{2\pi i \frac{N-1}{N}})^1 & (e^{2\pi i \frac{N-1}{N}})^2 & \cdots & (e^{2\pi i \frac{N-1}{N}})^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}.$$

Now recall again the discrete Fourier coefficients
$A_n = \dfrac{1}{N} \displaystyle\sum_{k=0}^{N-1} F(k)(e^{-2\pi i \frac{n}{N}})^k$ for n running from 0 to $N-1$.

Replace a_k with $\frac{1}{N}F(k)$ and take the conjugate of the DFT matrix then we have Cooley and Tukey's radix-2 FFT.

Recall our choice to sample the polynomial on $\{e^{2\pi i \frac{k}{N}}\}_{k=0}^{N-1}$, this gives us the following transformation:

$$\begin{pmatrix} 1 & (e^{2\pi i \frac{0}{N}})^1 & (e^{2\pi i \frac{0}{N}})^2 & \cdots & (e^{2\pi i \frac{0}{N}})^{N-1} \\ 1 & (e^{2\pi i \frac{1}{N}})^1 & (e^{2\pi i \frac{1}{N}})^1 & \cdots & (e^{2\pi i \frac{1}{N}})^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (e^{2\pi i \frac{N-1}{N}})^1 & (e^{2\pi i \frac{N-1}{N}})^2 & \cdots & (e^{2\pi i \frac{N-1}{N}})^{N-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}.$$

Now recall again the discrete Fourier coefficients
$A_n = \dfrac{1}{N} \displaystyle\sum_{k=0}^{N-1} F(k)(e^{-2\pi i \frac{n}{N}})^k$ for n running from 0 to $N-1$.
Replace a_k with $\frac{1}{N}F(k)$ and take the conjugate of the DFT matrix then we have Cooley and Tukey's radix-2 FFT.

OVERVIEW

After multiplication/convolution, we need to convert samples back into coefficients.

Luckily, this is easy to do!

Let V be the DFT matrix as described earlier, then

$$V^{-1} = \frac{1}{N}\bar{V},$$

which can be proven by showing that

$$V\bar{V} = NI.$$

Then $\vec{a} = V^{-1}\vec{y}$.

After multiplication/convolution, we need to convert samples back into coefficients.
Luckily, this is easy to do!
Let V be the DFT matrix as described earlier, then

$$V^{-1} = \frac{1}{N}\bar{V},$$

which can be proven by showing that

$$V\bar{V} = NI.$$

Then $\vec{a} = V^{-1}\vec{y}$.

After multiplication/convolution, we need to convert samples back into coefficients.
Luckily, this is easy to do!
Let V be the DFT matrix as described earlier, then

$$V^{-1} = \frac{1}{N}\bar{V},$$

which can be proven by showing that

$$V\bar{V} = NI.$$

Then $\vec{a} = V^{-1}\vec{y}$.

After multiplication/convolution, we need to convert samples back into coefficients.
Luckily, this is easy to do!
Let V be the DFT matrix as described earlier, then

$$V^{-1} = \frac{1}{N}\bar{V},$$

which can be proven by showing that

$$V\bar{V} = NI.$$

Then $\vec{a} = V^{-1}\vec{y}$.

References

Demaine, Erik D. "3. Divide & Conquer: FFT." YouTube. MIT OpenCourseWare, 04 Mar. 2016. Web. 07 Mar. 2017.

Stein, Elias M., and Rami Shakarchi. "Chapter 7 Finite Fourier Analysis." Fourier Analysis: An Introduction. Princeton: Princeton UP, 2003. N. pag. Print.

References

- Demaine, Erik D. "3. Divide & Conquer: FFT." YouTube. MIT OpenCourseWare, 04 Mar. 2016. Web. 07 Mar. 2017.
- Stein, Elias M., and Rami Shakarchi. "Chapter 7 Finite Fourier Analysis." Fourier Analysis: An Introduction. Princeton: Princeton UP, 2003. N. pag. Print.