Fast Fourier Transform

Nevin Kapur

Computer Science, Caltech

Polynomials

- A **polynomial** in variable x:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j .$$

$a_0, a_1, \ldots, a_{n-1}$ are **coefficients** of the polynomial, drawn from a field. (Think $\mathbb{C}$.)

Polynomials

- A **polynomial** in variable x:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j .$$

  $a_0, a_1, \ldots, a_{n-1}$ are **coefficients** of the polynomial, drawn from a field. (Think $\mathbb{C}$.)

- The **degree** of a polynomial is its highest nonzero coefficient.

< > - +

Polynomials

- A **polynomial** in variable x:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j .$$

  $a_0, a_1, \ldots, a_{n-1}$ are **coefficients** of the polynomial, drawn from a field. (Think $\mathbb{C}$.)

- The **degree** of a polynomial is its highest nonzero coefficient.
- Any integer strictly greater than the degree is a **degree-bound** of the polynomial. A degree-bound of n implies a degree between 0 and $n-1$.

Polynomials

- A **polynomial** in variable x:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j.$$

  $a_0, a_1, \ldots, a_{n-1}$ are **coefficients** of the polynomial, drawn from a field. (Think $\mathbb{C}$.)

- The **degree** of a polynomial is its highest nonzero coefficient.
- Any integer strictly greater than the degree is a **degree-bound** of the polynomial. A degree-bound of n implies a degree between 0 and $n-1$.
- **polynomial addition**: $C(x) = A(x) + B(x)$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \; B(x) = \sum_{j=0}^{n-1} b_j x^j, \qquad C(x) = \sum_{j=0}^{n-1} c_j x^j,$$

  where $c_j = a_j + b_j$.

- **polynomial multiplication**: $C(x) = A(x)B(x)$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \; B(x) = \sum_{j=0}^{n-1} b_j x^j, \qquad C(x) = \sum_{j=0}^{2n-2} c_j x^j,$$

where

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}. \qquad \textbf{convolution}$$

< > - +

- **polynomial multiplication**: $C(x) = A(x)B(x)$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \; B(x) = \sum_{j=0}^{n-1} b_j x^j, \qquad C(x) = \sum_{j=0}^{2n-2} c_j x^j,$$

where

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}.$$

convolution

- The naïve algorithm takes time proportional to

$$1 + 2 + \cdots + (2n-1) = \Theta(n^2).$$

< > - +

- **polynomial multiplication**: $C(x) = A(x)B(x)$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j, \; B(x) = \sum_{j=0}^{n-1} b_j x^j, \qquad C(x) = \sum_{j=0}^{2n-2} c_j x^j,$$

  where

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}. \qquad \textbf{convolution}$$

- The naïve algorithm takes time proportional to

$$1 + 2 + \cdots + (2n-1) = \Theta(n^2).$$

- We'll see an algorithm that works in $O(n \log n)$ time.

Representation of polynomials

- Represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$ as a vector of coefficients $\mathbf{a} = (a_0, a_1, \ldots, a_n)$.

Representation of polynomials

- Represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$ as a vector of coefficients $\mathbf{a} = (a_0, a_1, \ldots, a_n)$.
- **Evaluating** the polynomial at a given point x_0 takes time $\Theta(n)$ using Horner's rule. (Additional and multiplication take constant time.)

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0 a_{n-1}))\ldots).$$

< > - +

Representation of polynomials

- Represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$ as a vector of coefficients $\mathbf{a} = (a_0, a_1, \ldots, a_n)$.
- **Evaluating** the polynomial at a given point x_0 takes time $\Theta(n)$ using Horner's rule. (Additional and multiplication take constant time.)

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0 a_{n-1}))\ldots).$$

- **Adding** two polynomials takes $\Theta(n)$ time:

$$\mathbf{c} = \mathbf{a} + \mathbf{b},$$

$c_j = a_j + b_j$, $j = 0, 1, \ldots, n-1$.

Representation of polynomials

- Represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$ as a vector of coefficients $\mathbf{a} = (a_0, a_1, \ldots, a_n)$.
- **Evaluating** the polynomial at a given point x_0 takes time $\Theta(n)$ using Horner's rule. (Additional and multiplication take constant time.)

$$A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0 a_{n-1}))\ldots).$$

- **Adding** two polynomials takes $\Theta(n)$ time:

$$\mathbf{c} = \mathbf{a} + \mathbf{b},$$

$c_j = a_j + b_j$, $j = 0, 1, \ldots, n-1$.

- **Multiplying** two polynomials naïvely takes $\Theta(n^2)$. Use the convolution of $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{c} = \mathbf{a} \otimes \mathbf{b}.$$

Alternative representation: point-value

- Use n **point-value pairs** to represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$:

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

such that $(x_k)_{k=0}^{n-1}$ are all distinct and $y_k = A(x_k)$.

< > - +

Alternative representation: point-value

- Use n **point-value pairs** to represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$:

  $$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

  such that $(x_k)_{k=0}^{n-1}$ are all distinct and $y_k = A(x_k)$.
- coefficient representation $\longrightarrow$ point-value representation: pick n distinct points and use Horner's rule to evaluate their values. This takes $O(n^2)$ time, but can be speeded up to $O(n \log n)$ by choosing the n points carefully.

Alternative representation: point-value

- Use n **point-value pairs** to represent $A(x) = \sum_{j=0}^{n-1} a_j x^j$:

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

  such that $(x_k)_{k=0}^{n-1}$ are all distinct and $y_k = A(x_k)$.

- coefficient representation $\longrightarrow$ point-value representation: pick n distinct points and use Horner's rule to evaluate their values. This takes $O(n^2)$ time, but can be speeded up to $O(n \log n)$ by choosing the n points carefully.

- point-value representation $\longrightarrow$ coefficient representation: **(Interpolation)**
  Theorem. A set of n point-value pairs uniquely determines a polynomial of degree-bound n.
  Proof. $y_k = A(x_k)$ is equivalent to a matrix equation which has a unique inverse. $\checkmark$
  Time: $O(n^3)$ using LU-decomposition; $O(n^2)$ using Lagrange's formula.

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.
- **Multiplying** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = AB$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 y'_0, \ldots, y_{n-1} y'_{n-1}$.

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.
- **Multiplying** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = AB$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 y'_0, \ldots, y_{n-1} y'_{n-1}$.
  - But, if A and B have degree-bound n, then C has degree-bound $2n$.

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.
- **Multiplying** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = AB$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 y'_0, \ldots, y_{n-1} y'_{n-1}$.
  - But, if A and B have degree-bound n, then C has degree-bound $2n$.
  - n points are not enough to uniquely determine C.

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.
- **Multiplying** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = AB$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 y'_0, \ldots, y_{n-1} y'_{n-1}$.
  - But, if A and B have degree-bound n, then C has degree-bound $2n$.
  - n points are not enough to uniquely determine C.
  - Hence, we start with an **extended point-value representations** for A and B:
  $$\{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}$$
  and
  $$\{(x_0, y'_0) \ldots, (x_{2n-1}, y'_{2n-1})\}.$$

Operations with point-value representations

- **Adding** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = A + B$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 + y'_0, \ldots, y_{n-1} + y'_{n-1}$.
- **Multiplying** two polynomials: if A and B are evaluated at the same points $x_0, \ldots, x_{n-1}$ with values $y_0, \ldots, y_{n-1}$ and $y'_0, \ldots, y'_{n-1}$, then $C = AB$ evaluated at $x_0, \ldots, x_{n-1}$ is $y_0 y'_0, \ldots, y_{n-1} y'_{n-1}$.
  - But, if A and B have degree-bound n, then C has degree-bound $2n$.
  - n points are not enough to uniquely determine C.
  - Hence, we start with an **extended point-value representations** for A and B:

    $$\{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}$$

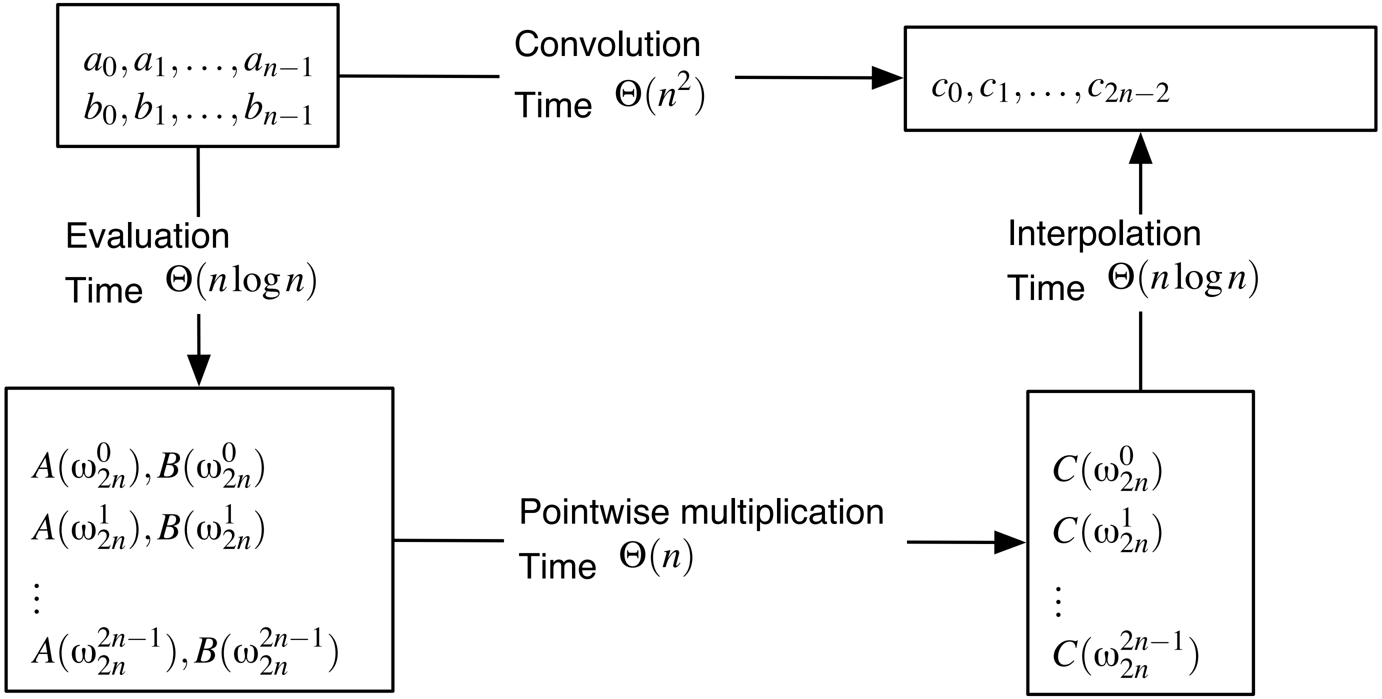
    and

    $$\{(x_0, y'_0) \ldots, (x_{2n-1}, y'_{2n-1})\}.$$

- Both addition and multiplication can be accomplished in $O(n)$ time.

< > - +

Fast multiplication



- $\omega_{2n}^0, \omega_{2n}^1, \ldots, \omega_{2n}^{2n-1}$ are the $2n-$th roots of unity.

< > - +

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.
- n nth roots of unity:

$$\omega_n^0, w_n^1, \ldots, w_n^{n-1}, \qquad \omega_n := e^{2\pi i/n}.$$

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.
- n nth roots of unity:

$$\omega_n^0, w_n^1, \ldots, w_n^{n-1}, \qquad \omega_n := e^{2\pi i/n}.$$

- Facts:

< > - +

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.
- n nth roots of unity:

$$\omega_n^0, w_n^1, \ldots, w_n^{n-1}, \qquad \omega_n := e^{2\pi i/n}.$$

- Facts:
  - $\omega_{dn}^{dk} = \omega_n^k$ **(Cancellation)**

$$\omega_{dn}^{dk} = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = w_n^k. \quad \checkmark$$

< > - +

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.
- n nth roots of unity:

$$\omega_n^0, w_n^1, \ldots, w_n^{n-1}, \qquad \omega_n := e^{2\pi i/n}.$$

- Facts:
  - $\omega_{dn}^{dk} = \omega_n^k$ **(Cancellation)**

$$\omega_{dn}^{dk} = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = w_n^k. \quad \checkmark$$

  - For $n > 0$ an even integer, $\omega_n^{n/2} = \omega_2 = -1$.

Roots of unity

- A **complex nth root of unity** is a complex number ω such that $\omega^n = 1$.
- n nth roots of unity:

$$\omega_n^0, w_n^1, \ldots, w_n^{n-1}, \qquad \omega_n := e^{2\pi i/n}.$$

- Facts:
  - $\omega_{dn}^{dk} = \omega_n^k$ **(Cancellation)**

$$\omega_{dn}^{dk} = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = w_n^k. \quad \checkmark$$

  - For $n > 0$ an even integer, $\omega_n^{n/2} = \omega_2 = -1$.
  - For $n > 0$ even, the squares of the nth roots of unity are the $(n/2)$nd roots of the unity. **(Halving)**

$$(\omega_n^k)^2 = \omega_{n/2}^k \quad \text{and} \quad (\omega_n^{k+n/2})^2 = (\omega_n^k)^2,$$

so that each $(n/2)$nd root is obtained exactly twice. $\checkmark$

Roots of unity, cont.

- For $n > 0$ and $k \neq 0$ not divisible by n,

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Summing the geometric series,

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0.$$

Note that the denominator is zero only if $w_n^k = 1$ only when k is divisible by n. $\checkmark$

< > - +

Discrete Fourier Transform

- Given a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$, its **Discrete Fourier Transform (DFT)** is defined as the vector $\mathbf{y} = \text{DFT}_n(a) = (y_0, \ldots, y_{n-1})$, where

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

< > - +

Discrete Fourier Transform

- Given a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$, its **Discrete Fourier Transform (DFT)** is defined as the vector $\mathbf{y} = \text{DFT}_n(a) = (y_0, \ldots, y_{n-1})$, where

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

- Recall that the evaluation step of our fast multiplication algorithm is just DFTs of the input polynomials.

< > - +

Discrete Fourier Transform

- Given a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$, its **Discrete Fourier Transform (DFT)** is defined as the vector $\mathbf{y} = \text{DFT}_n(a) = (y_0, \ldots, y_{n-1})$, where

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

- Recall that the evaluation step of our fast multiplication algorithm is just DFTs of the input polynomials.
- **Fast Fourier Transform (FFT)**: a method to compute the DFT in time $O(n \log n)$. [Recall that the naïve evaluation will take $\Theta(n^2)$ time.]

Discrete Fourier Transform

- Given a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$, its **Discrete Fourier Transform (DFT)** is defined as the vector $\mathbf{y} = \mathrm{DFT}_n(a) = (y_0, \ldots, y_{n-1})$, where

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

- Recall that the evaluation step of our fast multiplication algorithm is just DFTs of the input polynomials.
- **Fast Fourier Transform (FFT)**: a method to compute the DFT in time $O(n \log n)$. [Recall that the naïve evaluation will take $\Theta(n^2)$ time.]
  - **Idea:** Consider two "smaller" polynomials

$$A^{[0]}(x) = a_0 + a_2 x + \ldots a_{n-2} x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3 x + \ldots a_{n-1} x^{n/2-1}$$

< > - +

Discrete Fourier Transform

- Given a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$, its **Discrete Fourier Transform (DFT)** is defined as the vector $\mathbf{y} = \mathrm{DFT}_n(a) = (y_0, \ldots, y_{n-1})$, where

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}.$$

- Recall that the evaluation step of our fast multiplication algorithm is just DFTs of the input polynomials.
- **Fast Fourier Transform (FFT)**: a method to compute the DFT in time $O(n \log n)$. [Recall that the naïve evaluation will take $\Theta(n^2)$ time.]
  - **Idea:** Consider two "smaller" polynomials

$$A^{[0]}(x) = a_0 + a_2 x + \ldots a_{n-2} x^{n/2-1}$$

$$A^{[1]}(x) = a_1 + a_3 x + \ldots a_{n-1} x^{n/2-1}$$

  - Notice that $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$.

Divide-and-conquer

To evaluate A at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$:

- Evaluate the degree-bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points

  $$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2.$$

  [Notice that these are the squares of the nth roots of unity. If n is assumed to be even, they are just the $(n/2)$nd roots of unity, each appearing exactly twice.]

Divide-and-conquer

To evaluate A at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$:

- Evaluate the degree-bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points
$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2.$$
[Notice that these are the squares of the nth roots of unity. If n is assumed to be even, they are just the $(n/2)$nd roots of unity, each appearing exactly twice.]
- Combine them according to
$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2).$$

Divide-and-conquer

To evaluate A at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$:

- Evaluate the degree-bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points

  $$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2.$$

  [Notice that these are the squares of the nth roots of unity. If n is assumed to be even, they are just the $(n/2)$nd roots of unity, each appearing exactly twice.]
- Combine them according to

  $$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2).$$
- The subproblems are themselves DFT$_{n/2}$ computations. (Evaluate two $n/2$ degree-bound polynomials at the $n/2$nd roots of unity.)

Divide-and-conquer

To evaluate A at $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$:

- Evaluate the degree-bound $n/2$ polynomials $A^{[0]}(x)$ and $A^{[1]}(x)$ at the points
$$(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2.$$
[Notice that these are the squares of the nth roots of unity. If n is assumed to be even, they are just the $(n/2)$nd roots of unity, each appearing exactly twice.]
- Combine them according to
$$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2).$$
- The subproblems are themselves DFT$_{n/2}$ computations. (Evaluate two $n/2$ degree-bound polynomials at the $n/2$nd roots of unity.)
- A natural recursive algorithm follows . . . .

FFT

FFT($\mathbf{a}$)

$n \leftarrow$ length[**a**]
if $n = 1$ **then return a**
$\omega_n \leftarrow e^{2\pi i/n}$
$\omega \leftarrow 1$
$\mathbf{a}^{[0]} \leftarrow (a_0, a_2, \ldots, a_{n-2})$
$\mathbf{a}^{[1]} \leftarrow (a_1, a_3, \ldots, a_{n-1})$
$\mathbf{y}^{[0]} \leftarrow$ FFT$(\mathbf{a}^{[0]})$
$\mathbf{y}^{[1]} \leftarrow$ FFT$(\mathbf{a}^{[1]})$
for $k \leftarrow 0$ **to** $n/2 - 1$
    $y_k \leftarrow y_k^{[0]} + \omega y_k^{[1]}$
    $y_{k+(n/2)} \leftarrow y_k^{[0]} - \omega y_k^{[1]}$ $\triangleright\ \omega_n^{k+(n/2)} = -\omega_n^k$
    $\omega \leftarrow \omega\omega_n$
return y

Time complexity: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$.

Interpolation

- We can compute the DFTs of the polynomials we wish to multiply in $\Theta(n \log n)$ time.

Interpolation

- We can compute the DFTs of the polynomials we wish to multiply in $\Theta(n \log n)$ time.
- These can be multiplied pointwise in linear time, which gives us point-value pair representation of their product.

Interpolation

- We can compute the DFTs of the polynomials we wish to multiply in $\Theta(n \log n)$ time.
- These can be multiplied pointwise in linear time, which gives us point-value pair representation of their product.
- Next, we need a way to go from the point-value pair representation to the coefficient representation, i.e, $a_0, a_1, \ldots, a_{n-1}$, s.t.,

$$
\begin{aligned}
A(\omega_n^0) &= \sum_{j=0}^{n-1} a_j (\omega_n^0)^j &&= y_0 \\
A(\omega_n^1) &= \sum_{j=0}^{n-1} a_j (\omega_n^1)^j &&= y_1 \\
&\vdots \\
A(\omega_n^{n-1}) &= \sum_{j=0}^{n-1} a_j (\omega_n^{n-1})^j &&= y_{n-1}
\end{aligned}
$$

Inverting

- In matrix notation

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix}}_{\mathbf{V_n}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

Inverting

- In matrix notation

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix}}_{\mathbf{V_n}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

- In other words,

$$\mathbf{a} = \mathbf{V_n}^{-1}\mathbf{y}.$$

< > - +

Inverting

- In matrix notation

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{bmatrix}}_{\mathbf{V_n}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}.$$

- In other words,

$$\mathbf{a} = \mathbf{V_n}^{-1}\mathbf{y}.$$

- It turns out the inverse of $\mathbf{V_n}$ has a nice form . . . .

Inverse DFT

Theorem. With $\mathbf{V_n} = (\omega_n^{kj})_{j,k=0,\ldots,n-1}$, $\mathbf{V_n}^{-1} = (\omega_n^{-kj}/n)_{j,k=0,\ldots,n-1}$.

Proof. Compute the $ij$the entry of $\mathbf{V_n}^{-1}\mathbf{V_n}$:

$$[\mathbf{V_n}^{-1}\mathbf{V_n}]_{ij} = \sum_{k=0}^{n-1} (\omega_n^{-ik}/n)(\omega_n^{kj}) = \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{k(j-i)}.$$

If $i = j$, then this equals 1; otherwise it equals 0 by the summation lemma. $\checkmark$

< > - +

Inverse DFT

Theorem. With $\mathbf{V_n} = (\omega_n^{kj})_{j,k=0,\ldots,n-1}$, $\mathbf{V_n}^{-1} = (\omega_n^{-kj}/n)_{j,k=0,\ldots,n-1}$.

Proof. Compute the $ij$the entry of $\mathbf{V_n}^{-1}\mathbf{V_n}$:

$$[\mathbf{V_n}^{-1}\mathbf{V_n}]_{ij} = \sum_{k=0}^{n-1} (\omega_n^{-ik}/n)(\omega_n^{kj}) = \frac{1}{n}\sum_{k=0}^{n-1} \omega_n^{k(j-i)}.$$

If $i = j$, then this equals 1; otherwise it equals 0 by the summation lemma. ✓

- Thus the coefficients are given by

$$a_j = \frac{1}{n}\sum_{k=0}^{n-1} y_k \omega_n^{-kj}.$$

- But this is just like computing the FFT, with ω_n replaced by ω_n^{-1}, so that the coefficients can be computed in $\Theta(n \log n)$ time.