The Fast Fourier Transform
and Polynomial Multiplication
Version of September 9, 2016
Polynomial Multiplication

If \[ A(x) = \sum_{i=0}^{n-1} a_i x^i, \quad B(x) = \sum_{i=0}^{n-1} b_i x^i \]
then

\[ C(x) = A(x)B(x) \iff C(x) = \sum_{i=0}^{2n-1} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^{i} a_i b_{i-j} \]
Polynomial Multiplication

If \( A(x) = \sum_{i=0}^{n-1} a_i x^i \), \( B(x) = \sum_{i=0}^{n-1} b_i x^i \) then

\[ C(x) = A(x)B(x) \iff C(x) = \sum_{i=0}^{2n-1} c_i x^i \text{ where } c_i = \sum_{j=0}^{i} a_i b_{i-j} \]

- sequence \(< c_i >\) is convolution of \(< a_i >\) and \(< b_i >\)
- polynomial multiplication equivalent to calculating convolutions
- Straightforward multiplication alg is \( \Theta(n^2) \)
- Divide and conquer Karatsuba mult is \( O(n^{\log_2 3}) \)
Polynomial Multiplication

If \( A(x) = \sum_{i=0}^{n-1} a_i x^i \), \( B(x) = \sum_{i=0}^{n-1} b_i x^i \) then

\[
C(x) = A(x)B(x) \iff C(x) = \sum_{i=0}^{2n-1} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^{i} a_i b_{i-j}
\]

- sequence \(< c_i >\) is convolution of \(< a_i >\) and \(< b_i >\)
- polynomial multiplication equivalent to calculating convolutions
- Straightforward multiplication alg is \( \Theta(n^2) \)
- Divide and conquer Karatsuba mult is \( O(n^{\log_2 3}) \)

This Lecture
- \( O(n \log n) \) divide and conquer algorithm
- Uses Fast Fourier Transform (FFT)
- FFT calculates the Discrete Fourier Transform (DFT)
Polynomial Evaluation & Interpolation

Coefficient Representation

• \( A(x) = \sum_{i=0}^{n-1} a_i x^i \)

• Evaluation of \( A(x) \) for fixed \( x \): \( O(n) \) time
• Evaluation at \( n \) fixed values, \( x_0, x_1, \ldots, x_{n-1} \): \( O(n^2) \) time
Polynomial Evaluation & Interpolation

**Coefficient Representation**

- \( A(x) = \sum_{i=0}^{n-1} a_i x^i \)
- Evaluation of \( A(x) \) for fixed \( x \): \( O(n) \) time
- Evaluation at \( n \) fixed values, \( x_0, x_1, \ldots, x_{n-1} \): \( O(n^2) \) time

**Point Representation**

- Polynomial of degree \( n \) uniquely represented by \( n+1 \) values
  - e.g., 2 points determine a line; 3 points, a parabola
- Reconstructing coefficients of \( A(x) \) from \( n+1 \) values
  \( A(x_1), A(x_2), \ldots, A(x_n) \)
  requires \( O(n^2) \) time using Lagrangian Interpolation
Review of Lagrangian Interpolation

To construct degree-$n$ polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set

$$I_i(x) = \prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j}$$
Review of Lagrangian Interpolation

To construct degree-\( n \) polynomial \( A(x) \) from values \( A(x_0), A(x_1), \ldots A(x_n) \)

Set

\[
I_i(x) = \prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j} \quad \Rightarrow \quad I_i(x_j) = \begin{cases} 
0 & \text{if } j \neq i \\
1 & \text{if } j = i 
\end{cases}
\]
Review of Lagrangian Interpolation

To construct degree-$n$ polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set
\[
I_i(x) = \prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j}
\]
⇒
\[
I_i(x_j) = \begin{cases} 
0 & \text{if } j \neq i \\
1 & \text{if } j = i 
\end{cases}
\]

Set
\[
P(x) = \sum_i A(x_i) I_i(x).
\]
⇒
\[
P(x_i) = A(x_i) I_i(x_i) = A(x_i).
\]
Review of Lagrangian Interpolation

To construct degree-$n$ polynomial $A(x)$ from values $A(x_0), A(x_1), \ldots A(x_n)$

Set $I_i(x) = \prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j}$ \implies I_i(x_j) = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i \end{cases}$

Set $P(x) = \sum_i A(x_i)I_i(x)$. \implies P(x_i) = A(x_i)I_i(x_i) = A(x_i)$.

Since $P(x)$ has degree $n$ (why?), $P(x)$ is the unique polynomial satisfying $P(x_i) = A(x_i)$ for all $i$, so $A(x) = P(x)$.
Review of Lagrangian Interpolation

To construct degree-\(n\) polynomial \(A(x)\) from values \(A(x_0), A(x_1), \ldots A(x_n)\)

Set
\[
I_i(x) = \prod_{0 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j}
\]

\[
\Rightarrow \quad I_i(x_j) = \begin{cases} 
0 & \text{if } j \neq i \\
1 & \text{if } j = i 
\end{cases}
\]

Set \(P(x) = \sum_i A(x_i) I_i(x)\).

\[
\Rightarrow \quad P(x_i) = A(x_i) I_i(x_i) = A(x_i).
\]

Since \(P(x)\) has degree \(n\) (why?), \(P(x)\) is the unique polynomial satisfying \(P(x_i) = A(x_i)\) for all \(i\), so \(A(x) = P(x)\).

Note: \(P(x)\) can be constructed in \(O(n^2)\) time.
A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\
B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]
A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\
B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]

Polynomial Multiplication/Convolution

\[
O(n^2)
\]

\[
C(x) : c_0, \ldots, c_{2n-1}
\]

\[
C(x) = A(x)B(x)
\]
A New Approach To Polynomial Multiplication

\( x_0, x_1, \ldots, x_{2n-1} \) are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\ B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]

\[\text{Polynomial Multiplication/Convolution} \quad O(n^2)\]

\[
C(x) : c_0, \ldots, c_{2n-1}
\]

\[C(x) = A(x)B(x)\]

Evaluate \( A(x_i), B(x_i) \) for all \( x_i \)

\[O(n^2)\]

Pointwise Multiplication: \( C(x_i) = A(x_i)B(x_i) \)

\[O(n)\]

Interpolate coefficients of \( C(x) \)

\[O(n^2)\]

\[
A(x_i), B(x_i) \quad 0 \leq i \leq 2n - 1
\]

\[
C(x_i) \quad 0 \leq i \leq 2n - 1
\]
A New Approach To Polynomial Multiplication

$x_0, x_1, \ldots, x_{2n-1}$ are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\
B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]

Polynomial Multiplication/Convolution

\[
O(n^2)
\]

\[
C(x) : c_0, \ldots, c_{2n-1}
\]

\[
C(x) = A(x)B(x)
\]

- An alternative convolution method is to
  (1) evaluate all values $A(x_i), B(x_i)$
  (2) pointwise multiply to find $C(x_i) = A(x_i)B(x_i)$
  (3) interpolate to find $C(x)$

- Since Evaluation and Interpolation require $O(n^2)$, new method still requires $O(n^2)$. 
A New Approach To Polynomial Multiplication

\(x_0, x_1, \ldots, x_{2n-1}\) are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\
B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]

Polynomial Multiplication/Convolution

\[O(n^2)\]

\[
C(x) : c_0, \ldots, c_{2n-1}
\]

Interpolate coefficients of \(C(x)\)

\[O(n^2)\]

\[
C(x) = A(x)B(x)
\]

Evaluate \(A(x_i), B(x_i)\)

\[O(n^2)\]

\[
0 \leq i \leq 2n - 1
\]

\[
A(x_i), B(x_i)
\]

Pointwise Multiplication: \(C(x_i) = A(x_i)B(x_i)\)

\[O(n)\]

\[
C(x_i)
\]

\[
0 \leq i \leq 2n - 1
\]

• Assumption was that the \(x_i\) were arbitrary values
• If \(x_i\) are specified to be the (complex) roots of unity, FFT can evaluate and interpolate the values in \(O(n \log n)\) time.
A New Approach To Polynomial Multiplication

\( x_0, x_1, \ldots, x_{2n-1} \) are given values

\[
\begin{align*}
A(x) & : a_0, \ldots, a_{n-1} \\
B(x) & : b_0, \ldots, b_{n-1}
\end{align*}
\]

Polynomial Multiplication/Convolution

\[
C(x) = A(x)B(x)
\]

\( O(n^2) \)

Evaluate \( A(x_i), B(x_i) \)

\( 0 \leq i \leq 2n - 1 \)

Pointwise Multiplication: \( C(x_i) = A(x_i)B(x_i) \)

\( O(n) \)

Interpolate coefficients of \( C(x) \)

\( O(n \log n) \)

- Assumption was that the \( x_i \) were arbitrary values
- If \( x_i \) are specified to be the (complex) roots of unity, FFT can evaluate and interpolate the values in \( O(n \log n) \) time.
- Resulting in an \( O(n \log n) \) convolution algorithm!
Review of Complex Numbers & Roots of Unity

• \( i = \sqrt{-1} \)

• \( e^{ix} = \cos x + i \sin x \)

• \( w_n = e^{2\pi i/n} \) is the \( n \)'th principle root of unity; \( w_n^n = 1 \)

• The \( n \) roots of unity are \( w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1} \)

  \( w_n^i \) denotes \( (w_n)^i \)

• These \( n \) roots of unity are the \( n \) points, \( e^{2j\pi i/n}, j = 0, 1, \ldots, n - 1 \), equally spaced around the circle
Review of Complex Numbers & Roots of Unity

- $i = \sqrt{-1}$
- $e^{ix} = \cos x + i \sin x$
- $w_n = e^{2\pi i/n}$ is the $n$'th principle root of unity; $w_n^n = 1$
- The $n$ roots of unity are $w_n^0, w_n^1, w_n^2, \ldots, w_n^{n-1}$
  $w^i_n$ denotes $(w_n)^i$
- These $n$ roots of unity are the $n$ points, $e^{2j\pi i/n}$, $j = 0, 1, \ldots, n - 1$, equally spaced around the circle

If $n$ is even

- If $i < n/2$ then
  \[
  (w_n^i)^2 = w_n^{2i} = w_n^{i/2}
  \]
  and
  \[
  (w_n^{i+n/2})^2 = w_n w_n^{2i} = w_n^{i/2}
  \]
- $w_n^{n/2} = -1$
FFT Definition

FFT Definition

• Input: Coefficients $a_0, a_1, \ldots, a_{n-1}$

• Output: Values $A(w_0^0), A(w_1^1), \ldots A(w_{n-1}^n)$. Output is called the Discrete Fourier Transform ($DFT_n(A)$) of sequence $< a_i >$

• Will denote algorithm by $FFT_n(A)$

Assumption

• $n$ is a power of 2;

• if not, add zero coefficients, increasing size to power of 2
FFT Definition

- **Input**: Coefficients $a_0, a_1, \ldots, a_{n-1}$
- **Output**: Values $A(w^n_0), A(w^n_1), \ldots A(w^n_{n-1})$.
  Output is called the *Discrete Fourier Transform* ($DFT_n(A)$) of sequence $< a_i >$
- Will denote algorithm by $FFT_n(A)$

**Assumption**
- $n$ is a power of 2;
- if not, add zero coefficients, increasing size to power of 2

**Termination Condition**
- If $n = 1$ then sequence has only one value $a_0$ and $FFT_n(A)$ just returns the value $a_0$. 
Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$
Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts:

$$A(x) = A_0(x^2) + x A_1(x^2)$$

$$A_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots$$

$$A_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots$$
FFT Main Body

Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts:

$$A(x) = A_0(x^2) + xA_1(x^2)$$

$$A_0(x) = a_0 + a_2x + a_4x^2 + \cdots$$

$$A_1(x) = a_1 + a_3x + a_5x^2 + \cdots$$

If $i < \frac{n}{2}$ then

$$A(w_n^i) = A_0 \left( (w_n^i)^2 \right) + w_n^i A_1 \left( (w_n^i)^2 \right)$$

$$= A_0 \left( w_n^{i/2} \right) + w_n^i A_1 \left( w_n^{i/2} \right)$$
Want to evaluate $A(w_n^0), A(w_n^1), \ldots A(w_n^{n-1})$

Split $A(x)$ into even and odd parts:

$$A(x) = A_0(x^2) + xA_1(x^2)$$

$$A_0(x) = a_0 + a_2x + a_4x^2 + \cdots$$

$$A_1(x) = a_1 + a_3x + a_5x^2 + \cdots$$

If $i < \frac{n}{2}$ then

$$A(w_n^i) = A_0 \left( (w_n^i)^2 \right) + w_n^i A_1 \left( (w_n^i)^2 \right)$$

$$= A_0 \left( w_n^{i/2} \right) + w_n^i A_1 \left( w_n^{i/2} \right)$$

$$A(w_n^{i+n/2}) = A_0 \left( \left( w_n^{i+n/2} \right)^2 \right) - w_n^i A_1 \left( \left( w_n^{i+n/2} \right)^2 \right)$$

$$= A_0 \left( w_n^{i/2} \right) - w_n^i A_1 \left( w_n^{i/2} \right)$$
Just saw that
when \( n \) even

\[
A(w_n^i) = A_0 \left( w_n^{i/2} \right) + w_n^i A_1 \left( w_n^{i/2} \right)
\]

\[
A(w_n^{i+n/2}) = A_0 \left( w_n^{i/2} \right) - w_n^i A_1 \left( w_n^{i/2} \right)
\]

(always true since \( n \) a power of 2)
FFT Algorithm

Just saw that when $n$ even

$A(w_n^i) = A_0 \left( w_{n/2}^i \right) + w_n^i A_1 \left( w_{n/2}^i \right)$

(always true since $n$ a power of 2)

$A(w_n^{i+n/2}) = A_0 \left( w_{n/2}^i \right) - w_n^i A_1 \left( w_{n/2}^i \right)$

Note that $A_0(x), A_1(x)$ have degree $\frac{n}{2} - 1$ so can recursively call FFT to evaluate them on the $n/2$ roots of unity.
FFT Algorithm

Just saw that
when \( n \) even
\[
A(w_n^i) = A_0 \left( \frac{w_n^i}{2} \right) + w_n^i A_1 \left( \frac{w_n^i}{2} \right)
\]
(\textit{always true since } \( n \) \textit{ a power of 2})
\[
A(w_n^{i+n/2}) = A_0 \left( \frac{w_n^i}{2} \right) - w_n^i A_1 \left( \frac{w_n^i}{2} \right)
\]

Note that \( A_0(x), A_1(x) \) have degree \( \frac{n}{2} - 1 \) so can recursively call \( FFT \) to evaluate them on the \( n/2 \) roots of unity.

\[
\text{FFT}_n(A)
\]
If \( n = 1 \)

return \( a_0 \)

otherwise

Evaluate \( FFT_{n/2}(A_0), FFT_{n/2}(A_1) \)

Calculate \( A(w_n^i) \) for all \( i \) using those precalculated values
### FFT Algorithm

Just saw that

when \( n \) even

\[
A(w_n^i) = A_0\left(\frac{w_n^i}{2}\right) + w_n^i A_1\left(\frac{w_n^i}{2}\right)
\]

(always true since \( n \) a power of 2)

\[
A(w_n^i+n/2) = A_0\left(\frac{w_n^i}{2}\right) - w_n^i A_1\left(\frac{w_n^i}{2}\right)
\]

Note that \( A_0(x), A_1(x) \) have degree \( \frac{n}{2} - 1 \) so can recursively call \( FFT \) to evaluate them on the \( n/2 \) roots of unity.

\[
\frac{FFT_n(A)}{T(n)}
\]

- If \( n = 1 \)
  - return \( a_0 \)
- otherwise
  - Evaluate \( FFT_{n/2}(A_0), FFT_{n/2}(A_1) \) \( 2T(n/2) \)
  - Calculate \( A(w_n^i) \) for all \( i \) using those precalculated values \( O(n) \)

Running Time: \( T(n) = 2T(n/2) + O(n) \)
FFT Algorithm

Just saw that
when \( n \) even

\[
A(\omega_n^i) = A_0 \left( \frac{\omega_{n/2}^i}{\omega_n^i} \right) + \omega_n^i A_1 \left( \frac{\omega_{n/2}^i}{\omega_n^i} \right)
\]

\[
A(\omega_n^{i+n/2}) = A_0 \left( \frac{\omega_n^i}{\omega_n^{i+n/2}} \right) - \omega_n^i A_1 \left( \frac{\omega_n^i}{\omega_n^{i+n/2}} \right)
\]

(always true since \( n \) a power of 2)

Note that \( A_0(x), A_1(x) \) have degree \( \frac{n}{2} - 1 \) so can recursively call FFT to evaluate them on the \( n/2 \) roots of unity.

---

\[
\text{FFT}_n(A) \quad T(n)
\]

If \( n = 1 \)
return \( a_0 \)
otherwise

Evaluate \( \text{FFT}_{n/2}(A_0), \text{FFT}_{n/2}(A_1) \)
\( 2T(n/2) \)

Calculate \( A(\omega_n^i) \) for all \( i \) using those precalculated values \( O(n) \)

---

Running Time: \( T(n) = 2T(n/2) + O(n) \)

\( \Rightarrow \quad T(n) = O(n \log n) \)
FFT Based Interpolation

• Just saw that $FFT_n(A)$ evaluates $A(x)$ at $n$ roots of unity in $O(n \log n)$ time.

• Now need to see how to efficiently interpolate $A$’s coefficients given $DFT_n(A) = \{A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1})\}$.

• Amazingly, (proof soon)

  Lemma: If $d_j = A(w_n^j)$ set $D(x) = \sum_j d_j x^j$. Then, for all $i < n$,

  $$D(w_n^i) = \begin{cases} 
  na_0 & \text{if } i = 0 \\
  na_{n-i} & \text{if } i \neq 0
  \end{cases}$$

• This says that given $DFT_n(A)$ we can calculate coefficients of $A$ in $O(n \log n)$ time by another call to FFT and dividing answers by $n$. 
A New Approach To Polynomial Multiplication

\[ A(x) : a_0, \ldots, a_{n-1} \]
\[ B(x) : b_0, \ldots, b_{n-1} \]

Polynomial Multiplication/Convolution

\[ O(n^2) \]

\[ C(x) : c_0, \ldots, c_{2n-1} \]
\[ C(x) = A(x)B(x) \]

Use FFT to evaluate \( A(w_n^i), B(w_n^i) \) for all \( i < 2n - 1 \)

Pointwise Multiplication:
\[ C(w_n^i) = A(w_n^i)B(w_n^i) \]

\[ O(n) \]

Interpolate coefficients of \( C(x) \) using FFT

\[ 0 \leq i \leq 2n - 1 \]
A New Approach To Polynomial Multiplication

\[ A(x) : a_0, \ldots, a_{n-1} \]
\[ B(x) : b_0, \ldots, b_{n-1} \]

Polynomial Multiplication/Convolution

\[ C(x) = A(x)B(x) \]

\[ O(n^2) \]

Use FFT to evaluate \( A(w_n^i), B(w_n^i) \) for all \( 0 \leq i < 2n - 1 \)

\[ A(w_n^i), B(w_n^i) \]

\[ 0 \leq i \leq 2n - 1 \]

\[ O(n \log n) \]

\[ C(w_n^i) \]

\[ 0 \leq i \leq 2n - 1 \]

\[ O(n \log n) \]

Pointwise Multiplication: \( C(w_n^i) = A(w_n^i)B(w_n^i) \)

\[ O(n) \]

Following the blue arrows gives an \( O(n \log n) \) polynomial multiplication algorithm!
Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:
Lemma: If \( d_j = A(w_n^i) \) then, for all \( i < n \),

\[
D(w_n^i) = \begin{cases} 
na_0 & \text{if } i = 0 \\
na_{n-i} & \text{if } i \neq 0 
\end{cases}
\]

Proof:

First note

\[
\sum_{j=0}^{n-1} (w_n^t)^j = \begin{cases} 
n & \text{if } t = 0, n \\
0 & \text{if } t \neq 0, n 
\end{cases}
\]
Lemma: If $d_j = A(w^i_n)$ then, for all $i < n$,

$$D(w^i_n) = \begin{cases} 
na_0 & \text{if } i = 0 \\
na_{n-i} & \text{if } i \neq 0
\end{cases}$$

Proof:

First note 

$$\sum_{j=0}^{n-1} (w^t_n)^j = \begin{cases} 
n & \text{if } t = 0, n \\
0 & \text{if } t \neq 0, n
\end{cases}$$

If $t = 0, n$, $w^t_n = 1$ and obvious. If $t \neq 0, 1$, 

$$\sum_{j=0}^{n-1} (w^t_n)^j = \frac{w^{nt}-1}{w^t_n-1} = 0$$
**Lemma:** If \( d_j = A(w_n^i) \) then, for all \( i < n \),

\[
D(w_n^i) = \begin{cases} 
na_0 & \text{if } i = 0 \\
na_{n-i} & \text{if } i \neq 0
\end{cases}
\]

**Proof:**

First note

\[
\sum_{j=0}^{n-1} (w_n^t)^j = \begin{cases} 
n & \text{if } t = 0, n \\
0 & \text{if } t \neq 0, n
\end{cases}
\]

If \( t = 0, n \), \( w_n^t = 1 \) and obvious. If \( t \neq 0, 1 \),

\[
\sum_{j=0}^{n-1} (w_n^t)^j = \frac{w_n^{nt-1}}{w_n^t-1} = 0
\]

\[
D(w_n^i) = \sum_{j=0}^{n-1} d_j (w_n^i)^j
\]

\[
= \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} a_k \left( w_n^j \right)^k \right) (w_n^i)^j
\]

\[
= \sum_{k=0}^{n-1} a_k \left( \sum_{j=0}^{n-1} \left( w_n^{k+i} \right)^j \right)
\]
Lemma: If $d_j = A(w_n^i)$ then, for all $i < n$,

$$D(w_n^i) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$

Proof:

First note \[ \sum_{j=0}^{n-1} (w_n^t)^j = \begin{cases} n & \text{if } t = 0, n \\ 0 & \text{if } t \neq 0, n \end{cases} \]

If $t = 0, n$, $w_n^t = 1$ and obvious. If $t \neq 0, 1$, \[ \sum_{j=0}^{n-1} (w_n^t)^j = \frac{w_n^{nt} - 1}{w_n^t - 1} = 0 \]

$$D(w_n^i) = \sum_{j=0}^{n-1} d_j (w_n^i)^j$$

$$= \sum_{j=0}^{n-1} \left( \sum_{k=0}^{n-1} a_k (w_n^j)^k \right) (w_n^i)^j$$

$$= \sum_{k=0}^{n-1} a_k \left( \sum_{j=0}^{n-1} (w_n^{k+i})^j \right) = \begin{cases} na_0 & \text{if } i = 0 \\ na_{n-i} & \text{if } i \neq 0 \end{cases}$$
Odds and Ends

• Note that FFT presented here as divide-and-conquer algorithm

• Can also be written iteratively and also implemented in a dedicated circuit (butterfly pattern)

• Can design $O(n \log n)$ algorithms that work on powers of other numbers, e.g., $n = 3^k$ or $n = 5^k$.

• Some other transforms using other orthogonal function bases can be implemented quickly, similar to the FFT, e.g., Hadamard Transforms

• Gauss actually developed something similar to FFT in 1805. Rediscovered many times afterwards

• Was “forgotten” until Cooley and Tukey published a paper describing it in 1965. After that, became one of the most used algorithms in the world!
A Matrix View of DFTs

Let $V$ be the Vandermonde matrix on the left and use $A$ to denote the vector of the $a_i$.
The DFT can then be seen as calculating $VA = DFT(A)$.

A little bit of work (similar to lemma on slides) shows that the $(j, k)$-th entry of the inverse matrix of $V$ has value $w_n^{-kj}/n$.
Using the fact that $w_n^{-k} = w_n^{n-k}$ we see that
The \((j, k)\)-th entry of the inverse matrix of \(V\) has value \(w_n^{-kj}/n\) and \(w_n^{-k} = w_n^{-k}\) so

\[
V = \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & w_n & w_n^2 & \cdots & w_n^{n-1} \\
1 & w_n^2 & w_n^4 & \cdots & w_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w_n^{n-1} & w_n^{2(n-1)} & \cdots & w_n^{(n-1)(n-1)}
\end{pmatrix}
\quad \text{and} \quad
V^{-1} = \frac{1}{n} \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & w_n^{-1} & w_n^{-2} & \cdots & w_n^{-(n-1)(n-1)} \\
1 & w_n^{-2} & w_n^{-4} & \cdots & w_n^{-(2(n-2))} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w_n^{-n} & w_n^{-2n} & \cdots & w_n^{-(n-1)(n-1)}
\end{pmatrix}
\]

Since \(v\left(\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}\right) = \begin{pmatrix} A(w_n^0) \\ A(w_n^1) \\ A(w_n^2) \\ \vdots \\ A(w_n^{n-1}) \end{pmatrix}\), we have

\(v^{-1}\left(\begin{pmatrix} A(w_n^0) \\ A(w_n^1) \\ A(w_n^2) \\ \vdots \\ A(w_n^{n-1}) \end{pmatrix}\right) = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}\)

This allows implementing the inverse DFT by mutiplying by \(V^{-1}\). But, from the above, multiplying by \(V^{-1}\) is the same as multiplying by \(V\), flipping some of the results and then dividing by \(n\).

This can be done in \(O(n \log n)\) by running the FFT algorithm and then doing \(O(n)\) more work.
**Definition:** The *Vandermonde matrix* on *n* values \(x_1, x_2, \ldots, x_n\) is the matrix below.

If \(x_i \neq x_j\) for all \(i, j\), the Vandermonde matrix is invertable.