Algorithms and Data Structures
Fast Fourier Transform
Complex numbers

Any polynomial $p(x)$ of degree $d$ ought to have $d$ roots. (I.e., $p(x) = 0$ should have $d$ solutions.)

But the equation

$$x^2 + 1 = 0$$

has no solutions at all if we restrict our attention to real numbers.

Introduce a special symbol $i$ to stand for a solution to $(\ast)$. Then $i^2 = -1$ and $(\ast)$ has the required two solutions, $i$ and $-i$.

Adding $i$ allows all polynomial equations to be solved! Indeed a polynomial of degree $d$ has $d$ roots (taking account of multiplicities). This is the *Fundamental Theorem of Algebra*. 

ADS: lects 5 & 6 – slide 2 –
Roots of Unity

In particular, 

\[ x^n = 1 \]

has \( n \) solutions in the complex numbers. They may be written 

\[ 1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1} \]

where \( \omega_n \) is the principal \( n \)th root of unity:

\[ \omega_n = \cos\left(\frac{2\pi}{n}\right) + i \sin\left(\frac{2\pi}{n}\right), \quad (\dagger). \]

**Convention:** from now on \( \omega_n \) denotes the principal \( n \)th root of unity given by (\( \dagger \)).

**Note:** \( e^{iu} = \cos u + i \sin u \) so \( \omega_n = e^{2\pi i/n} \).
8th Roots of Unity

\[ w_8 = e^{i\frac{2\pi}{8}} = \cos \left(\frac{2\pi}{8}\right) + i\sin \left(\frac{2\pi}{8}\right) = \frac{1+i}{\sqrt{2}} \]

“Wheel” representation of 8th roots-of-unity (complex plane).
Same wheel structure for any \( n \) (then \( \omega_n \) found at angle \( 2\pi/n \)).
The Discrete Fourier Transform (DFT)

Instance A sequence of $n$ complex numbers

$$a_0, a_1, a_2, \ldots, a_{n-1},$$

$n$ IS A POWER-OF-2.

Output The sequence of $n$ complex numbers

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

obtained by evaluating the polynomial

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

at the $n$th roots of unity.
The Discrete Fourier Transform (DFT)

Instance  A sequence of $n$ complex numbers

$$a_0, a_1, a_2, \ldots, a_{n-1},$$

$n$ IS A POWER-OF-2.

Output  The sequence of $n$ complex numbers

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

obtained by evaluating the polynomial

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

at the $n$th roots of unity.

The DFT is a fingerprint of size $n$ of a polynomial.

CLASS QUESTION: It’s not the only fingerprint (why?)

ADS: lects 5 & 6 – slide 5 –
Motivation for algorithms for DFT/Inverse DFT

**Direct.** Signal processing: mapping between time and frequency domains.

**Indirect.** Subroutine in numerous applications, e.g., multiplying polynomials or large integers, cyclic string matching, etc.

It is important, therefore to find the fastest method. There is an obvious $\Theta(n^2)$ algorithm. Can we do better?

YES! Really cool algorithm (Fast Fourier Transform (FFT)) runs in $O(n \lg n)$ time. Published by Cooley & Tukey in 1965 - basics known by Gauss in 1805!

Used in *every* Digital Signal Processing application. Probably the most Important algorithm of today. We will show how to apply FFT to do polynomial multiplication in $O(n \lg n)$ (not most common application, but cute).
We are interested in evaluating:

\[ A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}, \]

\( n \) a \text{POWER-OF-2}. Put

\[ A_{\text{even}}(y) = a_0 + a_2 y + \cdots + a_{n-2} y^{n/2-1}, \]
\[ A_{\text{odd}}(y) = a_1 + a_3 y + \cdots + a_{n-1} y^{n/2-1}, \]

so that

\[ A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2). \quad \text{(\#)} \]

To evaluate \( A(x) \) at the \( n \)th roots of unity, we need to evaluate \( A_{\text{even}}(y) \) and \( A_{\text{odd}}(y) \) at the points \( 1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)} \).

\textit{We’ll show now that these are DFTs.} (wrt \( n/2 \))
Key Facts

Assuming $n$ is even:

- $\omega_n^2 = (e^{\frac{2\pi i}{n}})^2 = e^{\frac{2\pi i}{n/2}} = \omega_{n/2}$, and
- $\omega_n^{n/2} = (e^{\frac{2\pi i}{n}})^{n/2} = e^{\pi i} = -1$.

Thus we have the following relationships between $\omega_n$ and $\omega_{n/2}$:

<table>
<thead>
<tr>
<th>1</th>
<th>$\omega_n^2$</th>
<th>...</th>
<th>$\omega_n^{n-2}$</th>
<th>$\omega_n^n$</th>
<th>$\omega_n^{n+2}$</th>
<th>...</th>
<th>$\omega_n^{2(n-1)}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\omega_{n/2}$</td>
<td>...</td>
<td>$\omega_{n/2}^{n/2-1}$</td>
<td>1</td>
<td>$\omega_{n/2}$</td>
<td>...</td>
<td>$\omega_{n/2}^{n/2-1}$</td>
</tr>
</tbody>
</table>
Key Facts

Assuming $n$ is even:

\[ \omega_n^2 = (e^{\frac{2\pi i}{n}})^2 = e^{\frac{2\pi i}{n^2}} = \omega_{n/2}, \text{ and} \]

\[ \omega_{n/2}^n = (e^{\frac{2\pi i}{n}})^{n/2} = e^{\pi i} = -1. \]

Thus we have the following relationships between $\omega_n$ and $\omega_{n/2}$:

\[
\begin{array}{cccccccc}
1 & \omega_n^2 & \ldots & \omega_{n}^{n-2} & \omega_n^n & \omega_n^{n+2} & \ldots & \omega_n^{2(n-1)} \\
\| & \| & \ldots & \| & \| & \| & \ldots & \|
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & \omega_{n/2} & \ldots & \omega_{n/2}^{n/2-1} & 1 & \omega_{n/2} & \ldots & \omega_{n/2}^{n/2-1} \\
\| & \| & \| & \| & \| & \| & \| & \|
\end{array}
\]

So evaluating $A_{\text{odd}}(x), A_{\text{even}}(x)$ at $\omega^2$ for all $n$th-roots-of-unity (in order to implement (#)), is TWO “sweeps” of evaluating $A_{\text{odd}}(x), A_{\text{even}}(x)$ at the $n/2$th-roots.
“Divide”: a warning

In performing the “Divide” part of Divide-and-Conquer to DFT, it was important that the “Divide” was based on odd/even.

Suppose we had instead partitioned $A(x)$ into small/larger terms:

$$A_{\text{small}}(y) = a_0 + a_1 y + \cdots + a_{n/2-1} y^{n/2-1},$$
$$A_{\text{big}}(y) = a_{n/2} + a_{n/2+1} y + \cdots + a_{n-1} y^{n/2-1}$$

Then we would have

$$A(x) = A_{\text{small}}(x) + x^{n/2} A_{\text{big}}(x).$$

However, to evaluate $A(x)$ at the $n$th roots of unity, we would need to evaluate $A_{\text{small}}(y)$ and $A_{\text{big}}(y)$ at all of the $n$th roots of unity.

ADS: lects 5 & 6 – slide 9 –
“Divide”: a warning

In performing the “Divide” part of Divide-and-Conquer to DFT, it was important that the “Divide” was based on odd/even.

Suppose we had instead partitioned $A(x)$ into small/larger terms:

\[
A_{\text{small}}(y) = a_0 + a_1 y + \cdots + a_{n/2 - 1} y^{n/2-1},
\]
\[
A_{\text{big}}(y) = a_{n/2} + a_{n/2+1} y + \cdots + a_{n-1} y^{n/2-1}
\]

Then we would have

\[
A(x) = A_{\text{small}}(x) + x^{n/2} A_{\text{big}}(x).
\]

However, to evaluate $A(x)$ at the $n$th roots of unity, we would need to evaluate $A_{\text{small}}(y)$ and $A_{\text{big}}(y)$ at all of the $n$th roots of unity.

So for recursive calls: we would reduce the degree of the polynomial (to $n/2 - 1$), but would NOT reduce the “number of roots”. We would lose the relationship between degree of poly. and number of roots, which is crucial.

ADS: lects 5 & 6 – slide 9 –
Key Facts (cont’d)

\[ A(1) = A_{\text{even}}(1) + 1 \cdot A_{\text{odd}}(1) \]

\[ A(\omega_n) = A_{\text{even}}(\omega_n^2) + \omega_n A_{\text{odd}}(\omega_n^2) \]
\[ = A_{\text{even}}(\omega_{n/2}) + \omega_n A_{\text{odd}}(\omega_{n/2}) \]

\[ A(\omega_n^2) = A_{\text{even}}(\omega_{n/2}^2) + \omega_n^2 A_{\text{odd}}(\omega_{n/2}^2) \]

\[ \vdots \]

\[ A(\omega_n^{n/2-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) + \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1}) \]

The x co-efficient on \( xA_{\text{odd}}(x^2) \) of (♯) stays positive until \( x = \omega_n^{n/2} \).
Key Facts (cont’d)

\[ A(\omega_n^{n/2}) = A_{\text{even}}(1) - 1 \cdot A_{\text{odd}}(1) \]

\[ A(\omega_n^{n/2+1}) = A_{\text{even}}(\omega_{n/2}) - \omega_n A_{\text{odd}}(\omega_{n/2}) \]

\[ \vdots \]

\[ A(\omega_n^{n-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) - \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1}) \]

From \( \omega_n^{n/2} \) on, the \( x \) co-efficient of \( x A_{\text{odd}}(x^2) \) of (#) is negative.

We will use this negative relationship (with the \( j < n/2 \) case) on lines 8., 9. of our pseudocode.
The Fast Fourier Transform (FFT)

\[ A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}, \]

assume \( n \) is a power of 2. Compute

\[ A(1), A(\omega_n), A(\omega_n^2), \ldots, A(\omega_n^{n-1}), \quad (*) \]

as follows:

1. If \( n = 1 \) then \( A(x) \) is a constant so task is trivial. Otherwise split \( A \) into \( A_{\text{even}} \) and \( A_{\text{odd}} \).

2. By making two recursive calls compute the values of \( A_{\text{even}}(y) \) and \( A_{\text{odd}}(y) \) at the \( (n/2) \) points \( 1, \omega_{n/2}, \omega_{n/2}^2, \ldots, \omega_{n/2}^{n/2-1} \).

3. Compute the values \((*)\) by using the equation

\[ A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2). \]
Algorithm $\text{FFT}_n(\langle a_0, \ldots, a_{n-1} \rangle)$

1. if $n = 1$ then return $\langle a_0 \rangle$
2. else
3. \[ \omega_n \leftarrow e^{2\pi i / n} \]
4. \[ \omega \leftarrow 1 \]
5. $\langle y_0^{\text{even}}, \ldots, y_{n/2-1}^{\text{even}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_0, a_2, \ldots, a_{n-2} \rangle)$
6. $\langle y_0^{\text{odd}}, \ldots, y_{n/2-1}^{\text{odd}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_1, a_3, \ldots, a_{n-1} \rangle)$
7. for $k \leftarrow 0$ to $n/2 - 1$ do
8. \[ y_k \leftarrow y_k^{\text{even}} + \omega y_k^{\text{odd}} \]
9. \[ y_{k+n/2} \leftarrow y_k^{\text{even}} - \omega y_k^{\text{odd}} \]
10. \[ \omega \leftarrow \omega \omega_n \]
11. return $\langle y_0, \ldots, y_{n-1} \rangle$

Algorithm assumes $n$ is a power of 2. Why? (CLASS discussion).
Algorithm \( \text{FFT}_n(\langle a_0, \ldots, a_{n-1} \rangle) \)

1. if \( n = 1 \) then return \( \langle a_0 \rangle \)
2. else
3. \( \omega_n \leftarrow e^{2\pi i / n} \)
4. \( \omega \leftarrow 1 \)
5. \( \langle y_0^{\text{even}}, \ldots, y_{n/2-1}^{\text{even}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_0, a_2, \ldots, a_{n-2} \rangle) \)
6. \( \langle y_0^{\text{odd}}, \ldots, y_{n/2-1}^{\text{odd}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_1, a_3, \ldots, a_{n-1} \rangle) \)
7. for \( k \leftarrow 0 \) to \( n/2 - 1 \) do
8. \( y_k \leftarrow y_k^{\text{even}} + \omega y_k^{\text{odd}} \)
9. \( y_{k+n/2} \leftarrow y_k^{\text{even}} - \omega y_k^{\text{odd}} \)
10. \( \omega \leftarrow \omega \omega_n \)
11. return \( \langle y_0, \ldots, y_{n-1} \rangle \)

Algorithm assumes \( n \) is a power of 2. Why? (CLASS discussion).
Analysis

$T(n)$ worst-case running time of FFT.

Lines 1–4: $\Theta(1)$

Lines 5–6: $\Theta(1) + 2T(n/2)$

Loop, 7–10: $\Theta(n)$

Line 11: $\Theta(1)$

Yields the following recurrence:

$$T(n) = 2T(n/2) + \Theta(n).$$

Solution:

$$T(n) = \Theta(n \cdot \log(n)).$$
The Discrete Fourier Transform

Recall

- The DFT maps a tuple \(\langle a_0, \ldots, a_{n-1}\rangle\) to the tuple \(\langle y_0, \ldots, y_{n-1}\rangle\) defined by
  \[
y_j = \sum_{k=0}^{n-1} a_k \omega_n^{jk},
  \]
  where \(\omega_n = e^{2\pi i/n}\) is the principal \(n\)th root of unity.

- Thus for every \(n\) (power of 2) we may view \(\text{DFT}_n\) as mapping \(\mathbb{C}^n \rightarrow \mathbb{C}^n\), where \(\mathbb{C}\) denote the complex numbers.

- \(\text{FFT}\) (the Fast Fourier Transform) is an algorithm computing \(\text{DFT}_n\) in time \(\Theta(n \lg(n))\).
The inverse DFT

\[ \text{DFT}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n \]

\[ \langle a_0, \ldots, a_{n-1} \rangle \mapsto \langle y_0, \ldots, y_{n-1} \rangle \]

Can we go back from \( \langle y_0, \ldots, y_{n-1} \rangle \) to \( \langle a_0, \ldots, a_{n-1} \rangle \)?

More precisely:

1. Is \( \text{DFT}_n \) invertible, that is, is it one-to-one and onto?
2. If the answer to (1) is ‘yes’, can we compute \( \text{DFT}^{-1}_n \) efficiently?
The inverse DFT

\[ \text{DFT}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n \]
\[ \langle a_0, \ldots, a_{n-1} \rangle \mapsto \langle y_0, \ldots, y_{n-1} \rangle \]

Question

Can we go back from \( \langle y_0, \ldots, y_{n-1} \rangle \) to \( \langle a_0, \ldots, a_{n-1} \rangle \)?
The inverse DFT

\[
\text{DFT}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n \\
\langle a_0, \ldots, a_{n-1} \rangle \mapsto \langle y_0, \ldots, y_{n-1} \rangle
\]

Question

Can we go back from \( \langle y_0, \ldots, y_{n-1} \rangle \) to \( \langle a_0, \ldots, a_{n-1} \rangle \)?

More precisely:

1. Is \( \text{DFT}_n \) invertible, that is, is it one-to-one and onto?
2. If the answer to (1) is ‘yes’, can we compute \( \text{DFT}_n^{-1} \) efficiently?
An alternative view on the DFT

DFT$_n$ is the linear mapping described by the matrix

$$V_n = \begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)}
\end{pmatrix}.$$

That is, we have

$$V_n \begin{pmatrix}
a_0 \\
\vdots \\
a_{n-1}
\end{pmatrix} = \begin{pmatrix}
y_0 \\
\vdots \\
y_{n-1}
\end{pmatrix}.$$
An alternative view on the DFT

DFT\(_n\) is the linear mapping described by the matrix

\[
V_n = \begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)}
\end{pmatrix}.
\]

That is, we have

\[
V_n \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}
\]

We will NOT actually perform the naïve matrix mult. (we will do much better: \(O(n \lg n)\))
Inverse of DFT

Claim: $V_n$ is a van-der-Monde matrix and thus invertible.

Proof: Define the following “Inverse” matrix:

$$V_n^{-1} = \frac{1}{n} \begin{pmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n^{-1} & \omega_n^{-2} & \ldots & \omega_n^{-(n-1)} \\
1 & \omega_n^{-2} & \omega_n^{-4} & \ldots & \omega_n^{-2(n-1)} \\
1 & \omega_n^{-3} & \omega_n^{-6} & \ldots & \omega_n^{-3(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{- (n-1)} & \omega_n^{-2(n-1)} & \ldots & \omega_n^{-(n-1)(n-1)}
\end{pmatrix}.$$
Inverse of DFT (proof)

**Verification:** We must check that $V_n V_n^{-1} = I_n$:
Want $\ell\ell$-th entry $= 1 \forall \ell$, and $\ell j$-th entry $= 0 \forall \ell, j$ with $\ell \neq j$.

Expanding …

\[
(V_n V_n^{-1})_{\ell j} = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{\ell k} \omega_n^{-kj}
\]
Inverse of DFT (proof)

**Verification:** We must check that $V_n V_n^{-1} = I_n$:
Want $\ell \ell$-th entry $= 1 \ \forall \ell$, and $\ell j$-th entry $= 0 \ \forall \ell, j$ with $\ell \neq j$.

Expanding ... 

$$(V_n V_n^{-1})_{\ell j} = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{\ell k} \omega_n^{-kj}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{(\ell - j)k},$$
Inverse of DFT (proof)

**Verification:** We must check that $V_n V_n^{-1} = I_n$:

Want $\ell\ell$-th entry $= 1 \ \forall \ell$, and $\ell j$-th entry $= 0 \ \forall \ell, j$ with $\ell \neq j$.

Expanding ...

$$
(V_n V_n^{-1})_{\ell j} = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{\ell k} \omega_n^{-kj}
$$

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

$$
= \begin{cases} 
1 & \text{if } \ell = j \text{ (because } \omega_n^{\ell-j} = 1) \\
0 & \text{otherwise}
\end{cases}
$$

**ADS:lects 5 & 6 – slide 19 –**
Inverse of DFT (proof)

Verification: We must check that $V_nV_n^{-1} = I_n$:
Want $\ell \ell$-th entry $= 1 \ \forall \ell$, and $\ell j$-th entry $= 0 \ \forall \ell, j$ with $\ell \neq j$.
Expanding ...

$$(V_nV_n^{-1})_{\ell j} = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{\ell k} \omega_n^{-kj}$$

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

$$= \begin{cases} 1 & \text{if } \ell = j \ (\text{because } \omega_n^{\ell-j} = 1) \\ 0 & \text{otherwise} \end{cases}$$

$(V_nV_n^{-1})_{\ell j} = 0$ case uses the fact that for all $r \neq 0 \ (r = (\ell - j))$

we have $\sum_{k=0}^{n-1} \omega_n^{rk} = 0$. 

ADS: lects 5 & 6 – slide 19 –
Inverse of DFT

We have shown $\text{DFT}_n$ is invertible with

\[
\text{DFT}_n^{-1} : \begin{pmatrix}
y_0 \\
\vdots \\
y_{n-1}
\end{pmatrix} \mapsto V_n^{-1} \begin{pmatrix}
y_0 \\
\vdots \\
y_{n-1}
\end{pmatrix} = \begin{pmatrix}
a_0 \\
\vdots \\
a_{n-1}
\end{pmatrix}.
\]

Problem

If we were to apply $V_n^{-1}$ directly in order to recover $\langle a_0, \ldots, a_{n-1} \rangle$, the evaluation of $V_n^{-1} \langle y_0, \ldots, y_{n-1} \rangle$ would take $\Theta(n^2)$ time!!!

Solution

Take another look back at the $V_n^{-1}$ matrix, and see that it is more-or-less a “flipped-over” DFT.

ADS: lects 5 & 6 – slide 20 –
Inverse of DFT

We have shown $\text{DFT}_n$ is invertible with

$$\text{DFT}_n^{-1} : \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} \mapsto V_n^{-1} \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$  

Problem

If we are were to apply $V_n^{-1}\langle y_0, \ldots, y_{n-1} \rangle$ directly in order to recover $\langle a_0, \ldots, a_{n-1} \rangle$, the evaluation of $V_n^{-1}\langle y_0, \ldots, y_{n-1} \rangle$ would take $\Theta(n^2)$ time!!!
Inverse of DFT

We have shown $DFT_n$ is invertible with

$$DFT_n^{-1} : \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} \mapsto V_n^{-1} \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$ 

Problem

If we are were to apply $V_n^{-1}\langle y_0, \ldots, y_{n-1} \rangle$ directly in order to recover $\langle a_0, \ldots, a_{n-1} \rangle$, the evaluation of $V_n^{-1}\langle y_0, \ldots, y_{n-1} \rangle$ would take $\Theta(n^2)$ time!!!

Solution

Take another look back at the $V_n^{-1}$ matrix, and see that it is more-or-less a “flipped-over” DFT.
Inverse DFT (efficient) Algorithm

\( \omega_n^{-1} \) is an \( n \)th root of unity (though not the principal one). Note that

\[
(\omega_n^{-1})^j = 1/\omega_j = \omega_n^n/\omega_j = \omega_n^{n-j},
\]

for every \( 0 \leq j < n \).
Inverse DFT (efficient) Algorithm

\( \omega_n^{-1} \) is an \( n \)th root of unity (though not the principal one). Note that

\[
(\omega_n^{-1})^j = 1/\omega_n^j = \omega_n^n/\omega_n^j = \omega_n^{n-j},
\]

for every \( 0 \leq j < n \).

Inverse FFT

- Compute \( \text{DFT}_n \langle y_0, \ldots, y_{n-1} \rangle \) (deliberately using \( \text{DFT}_n \), not inverse), to obtain the result \( \langle d_0, \ldots, d_{n-1} \rangle \).
- Flip the sequence \( d_1, d_2, \ldots, d_{n-1} \) in this result (keeping \( d_0 \) fixed), then divide every term by \( n \).

\[
a_i = \begin{cases} 
\frac{d_0}{n} & \text{if } i = 0 \\
\frac{d_{n-i}}{n} & \text{if } 1 \leq i \leq n - 1
\end{cases}
\]

Worst-case running time is \( \Theta(n \log(n)) \).
Inverse DFT (efficient) Algorithm

$\omega_n^{-1}$ is an $n$th root of unity (though not the principal one). Note that

$$(\omega_n^{-1})^j = 1/\omega_n = \omega_n^n/\omega_n^j = \omega_n^{n-j},$$

for every $0 \leq j < n$.

Inverse FFT

- Compute $\text{DFT}_n\langle y_0, \ldots, y_{n-1} \rangle$ (*deliberately* using $\text{DFT}_n$, not inverse), to obtain the result $\langle d_0, \ldots, d_{n-1} \rangle$.
- Flip the sequence $d_1, d_2, \ldots, d_{n-1}$ in this result (keeping $d_0$ fixed), then divide every term by $n$.

$$a_i = \begin{cases} 
\frac{d_0}{n} & \text{if } i = 0 \\
\frac{d_{n-i}}{n} & \text{if } 1 \leq i \leq n-1 
\end{cases}$$

Worst-case running time is $\Theta(n \log(n))$.
Our Application! Multiplication of Polynomials

Input: \[ p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} \]
\[ q(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{m-1} x^{m-1}. \]

Required output:

\[ p(x)q(x) = (a_0 b_0) \]
\[ + (a_0 b_1 + a_1 b_0) x \]
\[ + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2 \]
\[ \vdots \]
\[ + (a_{n-2} b_{m-1} + a_{n-1} b_{m-2}) x^{n+m-3} \]
\[ + (a_{n-1} b_{m-1}) x^{n+m-2} \]

Naive method uses \( \Theta(nm) \) arithmetic operations

CAN WE DO BETTER?
Interpolation

Theorem

Let $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{C}$ pairwise distinct and $y_0, \ldots, y_{n-1} \in \mathbb{C}$. Then there exists exactly one polynomial $p(X)$ of degree at most $n - 1$ such that for $0 \leq k \leq n - 1$

$$p(\alpha_k) = y_k.$$
Interpolation

Theorem

Let \( \alpha_0, \ldots, \alpha_{n-1} \in \mathbb{C} \) pairwise distinct and \( y_0, \ldots, y_{n-1} \in \mathbb{C} \). Then there exists exactly one polynomial \( p(X) \) of degree at most \( n - 1 \) such that for \( 0 \leq k \leq n - 1 \)

\[
p(\alpha_k) = y_k.
\]

▶ The sequence

\[
\langle (\alpha_0, y_0), \ldots, (\alpha_{n-1}, y_{n-1}) \rangle
\]

is called a point-value representation of the polynomial \( p \).

▶ The process of computing a polynomial from a point-value representation is called **interpolation**.
Observation

Suppose we have two polynomials $p(X)$ (of degree $n - 1$) and $q(X)$ (of degree $m - 1$). Assume $\max\{m, n\} = n$. If $\langle (\alpha_0, y_0), \ldots, (\alpha_{n+m-2}, y_{n+m-2}) \rangle$ and $\langle (\alpha_0, z_0), \ldots, (\alpha_{n+m-2}, z_{n+m-2}) \rangle$ are point-value representations $p(X)$ and $q(X)$ respectively (evaluated at exactly the same points), then

$$\langle (\alpha_0, y_0z_0), \ldots, (\alpha_{n+m-2}, y_{n+m-2}z_{n+m-2}) \rangle$$

is a point-value representation of $p(X)q(X)$ (with enough points to allow us to recover $pq(X)$ by interpolation).
Multiplication of polynomials (cont’d)

![Diagram showing the process of multiplying polynomials]

- **Standard representation of two polynomials** → **Multiplication** → **Standard representation of product**
- **Point-value representation** → **Evaluation** → **Multiplication** → **Pointwise multiplication** → **Interpolation** → **Point-value representation of product**

We take the solid-arrow route, using 3 steps, to achieve performance $\Theta(n \log(n))$. 

*ADS: lects 5 & 6 – slide 25 –*
we take the solid-arrow route, using 3 steps, to achieve performance $\Theta(n \lg(n))$. 

$ADS: \text{lects 5 \& 6} – \text{slide 25} –$
Key idea

Let \( n' \) be the smallest power of 2 such that \( n' \geq n + m - 1 \). Use the \( n' \)-th roots of unity as the evaluation points:
\[
\alpha_0 = 1, \quad \alpha_1 = \omega_{n'}, \quad \alpha_2 = \omega_{n'}^2, \ldots, \quad \alpha_{n'-1} = \omega_{n'}^{n'-1}.
\]

Then

- evaluation \( \equiv \) DFT, and
- interpolation \( \equiv \) inverse DFT
Key idea

Let $n'$ be the smallest power of 2 such that $n' \geq n + m - 1$. Use the $n'$-th roots of unity as the evaluation points:

$\alpha_0 = 1, \alpha_1 = \omega_{n'}, \alpha_2 = \omega_{n'}^2, \ldots, \alpha_{n'-1} = \omega_{n'}^{n'-1}$.

Then

- evaluation $\equiv$ DFT, and
- interpolation $\equiv$ inverse DFT

Overall running time is

\[
\begin{align*}
\Theta(n' \log n') &= \Theta(n \log n) \quad \text{(FFT)} \\
+ \Theta(n') &= \Theta(n) \quad \text{(pointwise multiplication)} \\
+ \Theta(n' \log n') &= \Theta(n \log n) \quad \text{(inverse FFT)} \\
\hline
\end{align*}
\]

$\Theta(n \log n)$
Reading Assignment

[CLRS] (2nd and 3rd ed) Section 30.2 and 30.3.

Problems

1. Exercise 30.2-2 of [CLRS].
2. Let $f(x) = 3 \cos(2x)$. For $0 \leq k \leq 3$, let $a_k = f(2\pi k/4)$. Compute the DFT of $\langle a_0, \ldots, a_3 \rangle$.
   Do the same for $f(x) = 5 \sin(x)$.
3. Exercise 30.2-3 of [CLRS].
4. Exercise 30.2-7 of [CLRS].