### Polynomials Evaluation and Matrices

Evaluation: Compute $A(n)$ from $a$:

$$
\begin{bmatrix}
A(x_0) \\
A(x_1) \\
\vdots \\
A(x_{n-1})
\end{bmatrix} = 
\begin{bmatrix}
1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\
\vdots & \vdots & \vdots & \ldots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1}
\end{bmatrix} \begin{bmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{n-1}
\end{bmatrix}
$$

Interpolation (going back to coefficient matrix):

How?

Compute inverse of matrix above.

Multiply $O(n^2)$!

This sounds expensive!!

Also, computing inverse not even easy.

### Using roots of unity

FFT: $\omega$ is complex $n$th root of unity and matrix is...

$$
M_n(\omega) = 
\begin{bmatrix}
1 & 1 & 1 & \ldots & 1 \\
1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \ldots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)(n-1)}
\end{bmatrix}
$$

Compute inverse of $M_n(\omega)$?

### Geometry and FFT

Rows are orthogonal.

Multiply by $M_n(\omega)$: project point onto each row (and scaled.)

Rigid Rotation (and scaling)!

Reverse Rotation is inverse operation.

Scaling: for rotation, axis should have length 1, FFT length $n$.

### Computing inverse.

FFT works with points with basic root of unity: $\omega$ or $\omega^{-1}$

1, $\omega$, $\omega^{-1}$, $\omega^2$, $\omega^{-2}$, $\ldots$ $\omega^{n-1}$

$\omega^{-1}$ is a primitive $n$th root of unity!

Evaluation: FFT$(a, \omega)$.

Interpolation: $\frac{1}{n}$ FFT$(a, \omega^{-1})$.

$\Rightarrow O(n \log n)$ time for multiplying degree $n$ polynomials.

### Algebrically.

Inversion formula: $(M_n(\omega))^{-1} = \frac{1}{n} M_n(\omega^{-1})$.

$$
C = M_n(\omega) \times M_n(\omega^{-1})
$$

Recall: $\omega = e^{2\pi i/n}$.

$$
c_i = \sum_{k=0}^{n-1} \omega^{ik} - \sum_{k=0}^{n-1} \omega^{ik} = \sum_{k=0}^{n-1} \omega^{ik} - 0 = \sum_{k=0}^{n-1} \omega^{ik} = \frac{1 - \omega^{in}}{1 - \omega^i}
$$

Case $i = j$: $r = \omega^0 = 1$ and $c_i = n$.

Case $i \neq j$:

$$
c_i - 1 + r + r^2 + \ldots + r^{n-1} = \frac{1 - r^n}{1 - r}
$$

$$
r^n = (\omega^{i-j})^n = (\omega^n)^{i-j} = 1^{i-j} \Rightarrow c_i = 0.
$$

For $C$ – diagonals are $n$ and the off-diagonals are $0$.

Divide by $n$ get identity!

Inversion formula: $M_n(\omega^{-1}) = \frac{1}{n} M_n(\omega^{-1})$.

### Multiplying polynomials?

Evaluation: $O(n \log n)$ if choose 1, $\omega$, $\omega^2$, $\ldots$, $\omega^{n-1}$.

Evaluation with FFT$(\omega)$ $O(n \log n)$

Coefficient representation:

$\mathbf{a} = \{a_0, a_1, \ldots, a_{n-1}\}$

Value representation:

$A(x_0), A(x_1), \ldots, A(x_{n-1})$

Interpolation with FFT$(\omega^{-1})$ $O(n \log n)$

Interpolation: From points $A(x_0), \ldots, A(x_{n-1})$ to “function”.

### Recall: Multiplying polynomials, coefficient/value representation

Evaluation: $O(n \log n)$ if choose 1, $\omega$, $\omega^2$, $\ldots$, $\omega^{n-1}$.
FFT: “M(ω)a”

Idea:
“M(ω)a” computed from...
“M(ω^2)a_0” and “M(ω^2)a_0”

FFT(a,ω):
if ω = 1 return a
(a_0, a_1, ..., a_{n/2} - 1) = FFT((a_0, a_2, ..., a_{n/2} - 2), ω^2)
(a'_0, a'_1, ..., a'_{n/2} - 1) = FFT((a_1, a_3, ..., a_{n/1} - 1), ω^2)
for j = 0 to n/2 - 1:
j_r = s_j + ω^j s'_j
j_{r+1/2} = s_j - ω^j s'_j
return (s_0, s_1, ..., s_{n-1})

Runtime: T(n) = T(n/2) + O(n)

Order on Left

FIFO: a closer look.

Group even/odd cols
ω^{n/2} = -1; ω^n = 1

Unfolding FFT.

Butterfly switches!

Expanding FFT...

Edges from lower half of FFT have multipliers!

FFT Network.

Row r node connected to row r ± 2^i node in level i + 1

Unique Paths.

Route from input i = 101 to output j = 000?
Flip first bit. Red (cross) edge.
Keep second bit. Blue (straight) edge.
Flip third bit. Red (cross edge).
Definitive FFT algorithm and code.

\[
\begin{align*}
A(x) &= \sum_{i=0}^{d} a_i x^i = A_L(x) + x^{d/2} A_H(x),
A_L(x) &= \sum_{i=0}^{d/2} a_i x^{i},
A_H(x) &= \sum_{i=d/2+1}^{d} a_i x^i,
B(x) &= \sum_{i=0}^{d} b_i x^i = B_L(x) + x^{d/2} B_H(x),
B_L(x) &= \sum_{i=0}^{d/2} b_i x^i,
B_H(x) &= \sum_{i=d/2+1}^{d} b_i x^i.
\end{align*}
\]

The product \(A(x)B(x)\) is

\[
A(x)B(x) = A_L(x)B_L(x) + x^{d/2}(A_L(x)B_H(x) + A_H(x)B_L(x)) + x^d A_H(x)B_H(x).
\]

Compute...

\[
A_L(x)B_L(x), \quad A_H(x)B_H(x), \quad (A_L(x) + A_H(x))(B_L(x) + B_H(x))
\]

and recurse

Time is \(O(d \log_2 3)\)

FFT does better. (But this is useful to see)