Problem: Suppose $A(x)$ and $B(x)$ are polynomials of degree $n - 1$:

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

\[
B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}
\]

Compute $C(x) = A(x)B(x)$. We’ll assume $n$ is a power of 2.

How long does naive algorithm take? $O(n^2)$
Representation of Polynomials

Definition
The coefficient representation (CR) of a polynomial the vector of coefficients. E.g., \((1, 3, -2, 1)\) is the coefficient representation of

\[ f(x) = 1 + 3x - 2x^2 + x^3 \]

Definition
The point-value representation (PVR) of a polynomial: for \(n\) distinct points \(x_0, \ldots, x_{n-1}\) the PVR of \(f\) is

\[ \{(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_{n-1}, f(x_{n-1}))\} \]

E.g., \(f(x) \equiv \{(0, 1), (1, 3), (2, 7), (3, 19)\}\).

Lemma
Specifying the value of a function at \(n\) distinct points uniquely specifies a degree \(n - 1\) polynomial that goes through those points.
Polynomial Arithmetic in Point-Value Representation

- First attempt: Let \( x_0, \ldots, x_{n-1} \) be distinct and suppose

\[
A(x) \equiv \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}
\]

\[
B(x) \equiv \{(x_0, z_0), (x_1, z_1), \ldots, (x_{n-1}, z_{n-1})\}
\]

- Then surely,

\[
C(x) \equiv \{(x_0, y_0z_0), (x_1, y_1z_1), \ldots, (x_{n-1}, y_{n-1}z_{n-1})\}
\]

- **Issue:** While \( C(x_i) = y_i z_i \), \( C \) is a degree \( 2n - 2 \) polynomial and we need \( 2n - 1 \) distinct points to specify it.

- **Fix:** Assume \( A \) and \( B \) are specified on \( 2n - 1 \) distinct points.

- Can compute PVR of \( C \) is \( \Theta(n) \) time. But what about coefficient representation?
Framework for Fast Polynomial Multiplication

- Input: Coefficient representation of $A(x)$ and $B(x)$
- Step 1: Transform into PVR by evaluating at $2n - 1$ points
- Step 2: Multiply polynomials to get $C(x)$ in PVR
- Step 3: Transform PVR of $C(x)$ back into CR.

Naive implementation of step 1 takes $O(n^2)$ time. We'll do steps 1 and 3 in $O(n \log n)$ time.

**Important:** We can choose any distinct points for the PVR. Let's use the complex roots of unity...
Complex Roots of Unity

Definition
The $n$-th roots of unity are the complex solutions to the equation $x^n = 1$, i.e.,
\[ e^{2\pi i k/n} = \cos \frac{2\pi k}{n} + i \sin \frac{2\pi k}{n} \quad k = 0, \ldots, n - 1. \]

Let $\omega_n = e^{2\pi i / n}$.

Lemma (Halving Lemma)
If $n$ is even, then the squares of the $n$-th roots of unity are two copies of the $n/2$-th roots of unity:
\[ \{ (\omega_n^0)^2, \ldots, (\omega_n^{n-1})^2 \} = \{ \omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1} \} \cup \{ \omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1} \} \]
Divide and Conquer for Polynomial Evaluation

- Write degree $n - 1$ polynomial to be evaluated in terms of two degree $n/2 - 1$ polynomials:

\[
A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}
= (a_0 + a_2 x^2 + \ldots + a_{n-2} x^{n-2}) + x(a_1 + a_3 x^2 + \ldots + a_{n-1} x^{n-2})
= A_{even}(x^2) + xA_{odd}(x^2)
\]

- To evaluate $A$ at $n$-th roots of unity, we evaluate $A_{even}$ and $A_{odd}$ at

\[
x \in \{\omega_0^{n/2}, \omega_1^{n/2}, \ldots, \omega_0^{n/2-1}\}
\]

- If $T(n)$ is time to evaluate degree $n - 1$ poly at $n$-th roots of unity,

\[
T(1) = \Theta(1) \quad \text{and} \quad T(n) = 2T(n/2) + \Theta(n)
\]

- Use Master Theorem to conclude that $T(n) = \Theta(n \log n)$. 
Input: Coefficient representation of $A(x)$ and $B(x)$

- Step 1: Transform into PVR by evaluating at $2n - 1$ points
- Step 2: Multiply polynomials to get $C(x)$ in PVR
- Step 3: Transform PVR of $C(x)$ back into CR.

We now know:

1. Step 1 can be done in $O(n \log n)$ time.
2. Step 2 can be done in $O(n)$ time.

It turns out that Step 3 is almost identical to Step 1!
Polynomial Evaluation and Interpolation

Step 1 Revisited: Transform \((a_0, a_1, \ldots, a_{n-1})\) to

\[
\{(\omega_0^n, y_0), (\omega_1^n, y_1), \ldots, (\omega_{n-1}^n, y_{n-1})\}
\]

where \(y_i = A(\omega_i^n)\). In other words, we need to evaluate:

\[
V_n \cdot \begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix} = \begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
\]

where

\[
V_n = \begin{pmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n & \omega_n^2 & \omega_n^3 & \ldots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \ldots & \omega_n^{2(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \ldots & \omega_n^{(n-1)(n-1)}
\end{pmatrix}
\]
Polynomial Evaluation and Interpolation

Step 3 as inverse of Step 1: Need to transform

\[ \{(\omega_n^0, y_0), (\omega_n^1, y_1), \ldots, (\omega_n^{n-1}, y_{n-1})\} \]

into \((a_0, a_1, \ldots, a_{n-1})\) where \(y_i = A(\omega_n^i)\). In other words, we need

\[
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix} = V_n^{-1} \cdot
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
\]

The inverse of \(V_n\) is just \(V_n\) with \(\omega_n\) replaced by \(\omega_n^{-1}\)

\[
V_n^{-1} = \frac{1}{n} \begin{pmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega_n^{-1} & \omega_n^{-2} & \omega_n^{-3} & \ldots & \omega_n^{-(n-1)} \\
1 & \omega_n^{-2} & \omega_n^{-4} & \omega_n^{-6} & \ldots & \omega_n^{-2(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \omega_n^{-(n-1)-3} & \ldots & \omega_n^{-(n-1)(n-1)}
\end{pmatrix}
\]
Solving Step 3 Outline

- Need to compute:
  \[ a_k = \frac{\hat{A}(\omega_n^{-k})}{n} \text{ for } k = 0, \ldots, n - 1 \]
  where \( \hat{A}(x) = y_0 + y_1x + \ldots + y_{n-1}x^{n-1} \)

- Rewrite \( \hat{A}(x) = \hat{A}_{even}(x^2) + x\hat{A}_{odd}(x^2) \)

- To evaluate \( \hat{A} \) on \( \{\omega_0^n, \omega_{n-1}^n, \ldots, \omega_{(n-1)}^n\} \)
  it suffices to evaluate \( \hat{A}_{even} \) and \( \hat{A}_{odd} \) on \( \{\omega_0^{n/2}, \omega_{n/2}^{-1}, \ldots, \omega_{n/2}^{-(n/2-1)}\} \)
  because Halving Lemma also applies to \( \omega_{n}^{-1} \).

- Step 3 can also be done in \( O(n \log n) \) steps.