COMP251: Divide-and-Conquer (3)

Jérôme Waldispühl
School of Computer Science
McGill University
Based on (Kleinberg & Tardos, 2005)

Matrix multiplication

**Matrix multiplication.** Given two n-by-n matrices \( A \) and \( B \), compute \( C = A \cdot B \).

*Grade school.* \( \Theta(n^3) \) arithmetic operations.

\[
\begin{bmatrix}
c_{11} & c_{12} & \cdots & c_{1n} \\
c_{21} & c_{22} & \cdots & c_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix}
= \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\cdot
\begin{bmatrix}
b_{11} & b_{12} & \cdots & b_{1n} \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
\]

Q. Is grade school matrix multiplication algorithm asymptotically optimal?

Dot product

**Dot product.** Given two length-\( n \) vectors \( a \) and \( b \), compute \( c = a \cdot b \).

*Grade school.* \( \Theta(n) \) arithmetic operations.

\[
\begin{bmatrix}
a_1 \\
a_2 \\
\vdots \\
a_n
\end{bmatrix}
\cdot
\begin{bmatrix}
b_1 \\
b_2 \\
\vdots \\
b_n
\end{bmatrix}
= a_1 b_1 + a_2 b_2 + \cdots + a_n b_n
\]

Remark. Grade-school dot product algorithm is asymptotically optimal.

Block matrix multiplication

**Block matrix multiplication.**

\[
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\cdot
\begin{bmatrix}
b_{11} & b_{12} & \cdots & b_{1n} \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
= \begin{bmatrix}
a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1n}b_{n1} \\
a_{21}b_{11} + a_{22}b_{21} + \cdots + a_{2n}b_{n1} \\
\vdots & \ddots & \ddots & \vdots \\
a_{n1}b_{11} + a_{n2}b_{21} + \cdots + a_{nn}b_{n1}
\end{bmatrix}
\]

Stassen’s trick

**Key idea.** Multiply 2-by-2 blocks with only 7 multiplications.

(plus 11 additions and 7 subtractions)

\[
\begin{align*}
C_{11} &= a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31} + a_{14} b_{41} \\
C_{12} &= a_{11} b_{12} + a_{12} b_{22} + a_{13} b_{32} + a_{14} b_{42} \\
C_{21} &= a_{21} b_{11} + a_{22} b_{21} + a_{23} b_{31} + a_{24} b_{41} \\
C_{22} &= a_{21} b_{12} + a_{22} b_{22} + a_{23} b_{32} + a_{24} b_{42}
\end{align*}
\]

**Proof.**

\[
\begin{align*}
C_{11} &= P_1 + P_2 + P_3 + P_4 \\
C_{12} &= P_5 + P_6 + P_7 + P_8 \\
C_{21} &= P_9 + P_{10} + P_{11} + P_{12} \\
C_{22} &= P_1 + P_2 + P_3 + P_4
\end{align*}
\]

15-03-17
Strassen’s algorithm

\[
\text{STRASSEN}(A, B)
\]

\[
\mathbf{P}_1 \leftarrow \text{STRASSEN}(A_{11} + A_{12}B_{22}, A_{21}) + B_{12} \mathbf{R}_1 + B_{11} \mathbf{R}_2
\]

\[
\mathbf{P}_2 \leftarrow \mathbf{R}_1 - \mathbf{R}_2
\]

\[
\mathbf{P}_3 \leftarrow A_{11}A_{22} + A_{21}B_{12}
\]

\[
\mathbf{P}_4 \leftarrow A_{11}B_{11} + B_{12}(A_{22} + B_{22})
\]

\[
\mathbf{P}_5 \leftarrow (A_{12} + B_{22})B_{21}
\]

\[
\mathbf{P}_6 \leftarrow (A_{11} + A_{12})B_{22}
\]

\[
\mathbf{P}_7 \leftarrow A_{11}B_{11} + A_{12}B_{21}
\]

\[
\mathbf{C}_1 = \mathbf{P}_5 \times B_{22} - \mathbf{P}_3 \times B_{12} - \mathbf{P}_2 \times B_{11} + \mathbf{P}_4 \times B_{12}
\]

\[
\mathbf{C}_2 = \mathbf{P}_3 \times B_{22} + \mathbf{P}_2 \times B_{12} - \mathbf{P}_6 \times B_{11}
\]

\[
\mathbf{C}_3 = \mathbf{P}_1 \times B_{22} + \mathbf{P}_7 \times B_{12} + \mathbf{P}_4 \times B_{11}
\]

\[
\mathbf{C}_4 = \mathbf{P}_1 \times B_{11} - \mathbf{P}_5 \times B_{11} + \mathbf{P}_6 \times B_{21}
\]

\[
\mathbf{C}_5 = \mathbf{P}_1 \times B_{21}
\]

\[
\mathbf{C}_6 = \mathbf{P}_3 \times B_{11} - \mathbf{P}_4 \times B_{21}
\]

RETURN \(\mathbf{C}\)

Analysis of Strassen’s algorithm

**Theorem.** Strassen’s algorithm requires \(\Theta(n^{\log_2 7})\) arithmetic operations to multiply two \(n\times n\) matrices.

**Proof.** Apply case 1 of the master theorem to the recurrence:

\[
T(n) = 7 T\left(\frac{n}{2}\right) + O(n^{\log_2 7})
\]

Q. What if \(n\) is not a power of 2?

A. Could pad matrices with zeros.

Linear algebra reductions

**Matrix multiplication.** Given two \(n\times n\) matrices, compute their product.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Linear algebra</th>
<th>Order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiply</td>
<td>(A \times B)</td>
<td>(\Theta(n^3))</td>
</tr>
<tr>
<td>Matrix inverse</td>
<td>(A^{-1})</td>
<td>(\Theta(n^3))</td>
</tr>
<tr>
<td>Determinant</td>
<td>(</td>
<td>A</td>
</tr>
<tr>
<td>System of linear equations</td>
<td>(Ax=b)</td>
<td>(\Theta(n^3))</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>(A = LU)</td>
<td>(\Theta(n^3))</td>
</tr>
<tr>
<td>Least squares</td>
<td>(\min(</td>
<td></td>
</tr>
</tbody>
</table>

There are also \(n\times n\) arithmetic operations to multiply two \(n\times n\) matrices.

History of asymptotic complexity of matrix multiplication

<table>
<thead>
<tr>
<th>Year</th>
<th>Algorithm</th>
<th>Order of growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1968</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1970</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1971</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1972</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1982</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>1990</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>2010</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>2011</td>
<td>Strassen</td>
<td>(O(n^3))</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>(O(n^{2.373}))</td>
</tr>
<tr>
<td>2013</td>
<td>Williams</td>
<td>(O(n^{2.373}))</td>
</tr>
<tr>
<td>2015</td>
<td>Williams</td>
<td>(O(n^{2.373}))</td>
</tr>
<tr>
<td>2017</td>
<td>Williams</td>
<td>(O(n^{2.373}))</td>
</tr>
</tbody>
</table>

Note: Theoretical complexity to multiply two \(n\times n\) matrices.
Fourier analysis

Fourier theorem. [Fourier, Dirichlet, Riemann]. Any (suffciently smooth) periodic function can be expressed as the sum of a series of sinusoids.

\[ f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left( a_n \cos \frac{2\pi nt}{T} + b_n \sin \frac{2\pi nt}{T} \right) \]

\[ A = \frac{1}{T} \int_0^T f(t) \, dt \]

Euler’s identity

Euler’s identity. \( e^{ix} = \cos x + i\sin x \).

Sinusoids. Sum of sines and cosines = sum of complex exponentials.

Time domain vs. frequency domain

Signal. [Touch tone button 1]

Time domain.

Frequency domain.

Fast Fourier transform

FFT. Fast way to convert between time domain and frequency domain.

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

"If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it." — Numerical Recipes

Fast Fourier transform: applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ... digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson’s equation.
- Integer and polynomial multiplication.
- Shor’s quantum factoring algorithm.

"The FFT is one of the truly great computational developments of the 20th century. It has changed the face of science and engineering so much that it is not an exaggeration to say that without it we would be very different without the FFT."

— Charles van Loan
Polynomials: coefficient representation

**Polynomial** [coefficient representation]

\[ A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n \]
\[ B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n \]

**Add.** (or) arithmetic operations.

\[ A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1 x) + (a_2 + b_2 x^2) + \ldots + (a_n + b_n x^n) \]

**Multiply (convolve).** (or) but need \(2n+1\) points.

\[ A(x) \times B(x) = (a_0 b_0) + (a_0 b_1 x) + (a_0 b_2 x^2) + \ldots + (a_0 b_n x^n) + \ldots + (a_n b_0 x^n) \]

**Evaluate.** (or) using Lagrange's formula.

\[ A(x_0) = \sum_{i=0}^{n} \left( \frac{(x_0 - x_1)(x_0 - x_2)\ldots(x_0 - x_{i-1})(x_0 - x_{i+1})\ldots(x_0 - x_n)}{(x_i - x_1)(x_i - x_2)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} \right) a_i \]

**Polynomials: point-value representation**

**Fundamental theorem of algebra.** A degree \(n\) polynomial with complex coefficients has exactly \(n\) complex roots.

**Corollary.** A degree \(n-1\) polynomial \(A(x)\) is uniquely specified by its evaluation at \(n\) distinct values of \(x\).

![Fundamental theorem of algebra](https://example.com/fundamental_theorem_of_algebra.png)

**Converting between two representations**

**Tradeoff.** Fast evaluation or fast multiplication. We want both!

<table>
<thead>
<tr>
<th>Representation</th>
<th>Multiply</th>
<th>Evaluate</th>
</tr>
</thead>
<tbody>
<tr>
<td>coefficient</td>
<td>(O(n))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>point-value</td>
<td>(O(1))</td>
<td>(O(n))</td>
</tr>
</tbody>
</table>

**Goal.** Efficient conversion between two representations \(\rightarrow\) all ops fast.
Converting between two representations: brute force

**Coefficient → point-value.** Given a polynomial \( a_0 + a_1 x + \ldots + a_n x^n \), evaluate it at \( n + 1 \) distinct points \( x_0, \ldots, x_n \).

<table>
<thead>
<tr>
<th>( j )</th>
<th>( x_j )</th>
<th>( y_j )</th>
<th>( x_j^2 )</th>
<th>( x_j^3 )</th>
<th>( \ldots )</th>
<th>( x_j^n )</th>
<th>( a_0 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( \ldots )</th>
<th>( a_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_0 )</td>
<td>( y_0 )</td>
<td>( x_0^2 )</td>
<td>( x_0^3 )</td>
<td>( \ldots )</td>
<td>( x_0^n )</td>
<td>( a_0 )</td>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( \ldots )</td>
<td>( a_n )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 )</td>
<td>( y_1 )</td>
<td>( x_1^2 )</td>
<td>( x_1^3 )</td>
<td>( \ldots )</td>
<td>( x_1^n )</td>
<td>( a_0 )</td>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( \ldots )</td>
<td>( a_n )</td>
</tr>
<tr>
<td>2</td>
<td>( x_2 )</td>
<td>( y_2 )</td>
<td>( x_2^2 )</td>
<td>( x_2^3 )</td>
<td>( \ldots )</td>
<td>( x_2^n )</td>
<td>( a_0 )</td>
<td>( a_1 )</td>
<td>( a_2 )</td>
<td>( a_3 )</td>
<td>( \ldots )</td>
<td>( a_n )</td>
</tr>
</tbody>
</table>

**Running time.** \( O(n^2) \) for matrix-vector multiply (or \( x \) Horner’s).

Converting between two representations: brute force

**Point-value → coefficient.** Given \( n \) distinct points \( x_0, \ldots, x_n \) and values \( y_0, \ldots, y_n \), find unique polynomial \( a_0 + a_1 x + \ldots + a_n x^n \), that has given values at given points.

<table>
<thead>
<tr>
<th>( x_k )</th>
<th>( y_k )</th>
<th>( x_k^2 )</th>
<th>( x_k^3 )</th>
<th>( \ldots )</th>
<th>( x_k^n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( y_0 )</td>
<td>( x_0^2 )</td>
<td>( x_0^3 )</td>
<td>( \ldots )</td>
<td>( x_0^n )</td>
</tr>
<tr>
<td>1</td>
<td>( y_1 )</td>
<td>( x_1^2 )</td>
<td>( x_1^3 )</td>
<td>( \ldots )</td>
<td>( x_1^n )</td>
</tr>
<tr>
<td>2</td>
<td>( y_2 )</td>
<td>( x_2^2 )</td>
<td>( x_2^3 )</td>
<td>( \ldots )</td>
<td>( x_2^n )</td>
</tr>
</tbody>
</table>

**Running time.** \( O(n^2) \) for Gaussian elimination.

Divide-and-conquer

**Decimation in frequency.** Break up polynomial into low and high powers.

- \( A(0) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \)
- \( A_{\text{low}}(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{high}}(0) = a_2 x^2 + a_4 x^4 + a_6 x^6 \)

**Decimation in time.** Break up polynomial into even and odd powers.

- \( A(0) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7 \)
- \( A_{\text{even}}(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{odd}}(0) = a_2 x^2 + a_4 x^4 + a_6 x^6 \)
- \( A_{\text{even}}(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)

Coefficient to point-value representation: intuition

**Coefficient → point-value.** Given a polynomial \( a_0 + a_1 x + \ldots + a_n x^n \), evaluate it at \( n + 1 \) distinct points \( x_0, \ldots, x_n \). We get to choose which one.

**Divide.** Break up polynomial into even and odd powers.

- \( A(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{even}}(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{odd}}(0) = a_2 x^2 + a_4 x^4 + a_6 x^6 \)
- \( A(0) = A_{\text{even}}(0) + A_{\text{odd}}(0) \)

**Intuition.** Choose two points to be \( a \).

- \( A(1) = A_{\text{even}}(1) + A_{\text{odd}}(1) \)
- \( A(2) = A_{\text{even}}(2) - A_{\text{odd}}(2) \)

**Coefficient to point-value representation.**

**Divide.** Break up polynomial into even and odd powers.

- \( A(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{even}}(0) = a_0 + a_1 x + a_3 x^3 + a_5 x^5 + a_7 x^7 \)
- \( A_{\text{odd}}(0) = a_2 x^2 + a_4 x^4 + a_6 x^6 \)
- \( A(0) = A_{\text{even}}(0) + A_{\text{odd}}(0) \)

**Intuition.** Choose four complex points to be \( a \).\( \omega \)

- \( A(3) = A_{\text{even}}(3) - A_{\text{odd}}(3) \)
- \( A(5) = A_{\text{even}}(5) - A_{\text{odd}}(5) \)

Can evaluate polynomial of degree \( \leq 2 \) at 2 points by evaluating two polynomials of degree \( \leq 1 \) at 3 points.

**Discrete Fourier transform**

**Coefficient → point-value.** Given a polynomial \( a_0 + a_1 x + \ldots + a_n x^n \), evaluate it at \( n + 1 \) distinct points \( x_0, \ldots, x_n \).

**Key idea.** Choose \( \omega = \omega \) where \( \omega \) is principal \( n \) root of unity.

\[
\left[ \begin{array}{ccc}
0 & 1 & 1 & \ldots & 1 \\
1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{n(n-1)-1} \\
\end{array} \right] \left[ \begin{array}{c}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_n \\
\end{array} \right] = \left[ \begin{array}{c}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_n \\
\end{array} \right]
\]

Point values \( y_k \)
Roots of unity

**Def.** An $n$th root of unity is a complex number $z$ such that $z^n = 1$.

**Fact.** The $n$th roots of unity are $\omega^n, \omega^{2n}, \ldots, \omega^{(n-1)n}$ where $\omega = e^{2\pi i/n}$.

**Fast Fourier transform**

**Goal.** Evaluate a degree $n-1$ polynomial $a(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ at its $n$th roots of unity: $\omega^n, \omega^{2n}, \ldots, \omega^{(n-1)n}$.

**Divide.** Break up polynomial into even and odd powers.
- $A_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \ldots + a_{n-2}x^{n-2}$
- $A_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \ldots + a_{n-1}x^{n-1}$
- $A(0) = A_{\text{even}}(0) = A_{\text{odd}}(0)$

**Combine:**
- $A(\omega^k) = A_{\text{even}}(\omega^k) - A_{\text{odd}}(\omega^k), \text{ for } k = 0, \ldots, n/2 - 1$
- $A(\omega^k) = A_{\text{even}}(\omega^k) + A_{\text{odd}}(\omega^k), \text{ for } k = n/2, \ldots, n - 1$

**FFT: implementation**

```
FFT(n, a0, a1, ..., an-1)
if n = 1 return a0
(a0, a1, ..., an-1) = FFT(n/2, a0, a1, ..., an-2)
(a2, a3, ..., an-1) = FFT(n/2, a2, a3, ..., an-1)
for k = 0 to n/2 - 1
    \omega = e^{2\pi i k / n}
    y0 = a0 + a2 \omega + a4 \omega^2 + \ldots + a_{n-2} \omega^{n/2 - 1}
    y1 = a1 + a3 \omega + a5 \omega^2 + \ldots + a_{n-1} \omega^{n/2 - 1}
    return (y0, y1, ..., y0, y1)
```

**FFT: summary**

**Theorem.** The FFT algorithm evaluates a degree $n-1$ polynomial at each of the $n$th roots of unity in $O(n \log n)$ steps and $O(n)$ extra space.

**PF.** $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$

**Inverse discrete Fourier transform**

**Point-value to coefficient.** Given $n$ distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ that has given values at given points.

**Inverse DFT**

$$
\begin{bmatrix}
    y_0 \\
    y_1 \\
    \vdots \\
    y_{n-1}
\end{bmatrix}
= \begin{bmatrix}
    1 & 1 & 1 & \ldots & 1 \\
    1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1}
\begin{bmatrix}
    a_0 \\
    a_1 \\
    \vdots \\
    a_{n-1}
\end{bmatrix}
$$
Theorem

Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix $F_n$ is given by following formula:

$$G_n = \frac{1}{n} [1 \ 1 \ 1 \ \cdots \ 1 \ 1 \ \cdots \ 1 \ \cdots \ 1 \ 1]$$

$$[1 \ \omega \ \omega^2 \ \cdots \ \omega^{n-1} \ 1 \ \omega \ \omega^2 \ \cdots \ \omega^{n-1} \ 1 \ \cdots \ 1 \ \cdots \ 1 \ \cdots \ 1]$$

$$[1 \ \omega \ \omega^2 \ \cdots \ \omega^{n-1} \ 1 \ \omega \ \omega^2 \ \cdots \ \omega^{n-1} \ 1 \ \cdots \ 1 \ \cdots \ 1 \ \cdots \ 1]$$

Consequence. To compute inverse FFT, apply same algorithm but use $\omega = e^{j \pi / n}$ as principal $n$ root of unity (and divide the result by $n$).

Inverse FFT: implementation

Note. Need to divide result by $n$.

```
INVERSE-FFT(a, n)
  if n = 1 return a[0]
  a0, a1, a2, a3, a4 = INVERSE-FFT(a[0:n/2], n/2)
  w = -1  if n is odd
  w = 1  otherwise
  a = [a0, a1, a2, a3, a4, a3, a2, a1]
  return w * a
```

Inverse FFT: proof of correctness

Claim. $F_n$ and $G_n$ are inverses.

Proof:

$$[F_n, G_n]_{kj} = \frac{1}{n} \sum_{m=0}^{n-1} \omega^{km} \omega^{-m+jk} = \begin{cases} 1 & \text{if } k = j \\
0 & \text{otherwise} \end{cases}$$

Summation lemma. Let $\omega$ be a principal $n$ root of unity. Then

$$\sum_{k=0}^{n-1} \omega^k = \begin{cases} n & \text{if } k \equiv 0 \mod n \\
0 & \text{otherwise} \end{cases}$$

* If $\omega$ is a multiple of $\omega^j = 1 \Rightarrow$ series sums to $n$.
* Each $\omega$ root of unity $\omega^j$ is a root of $\omega^{n-1} = (\omega-1) (1 + \omega + \omega^2 + \cdots + \omega^{n-2})$.
* If $\omega^j = 1$ we have: $1 + \omega + \omega^2 + \cdots + \omega^{n-2} = 0 \Rightarrow$ series sums to 0.

Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree $n-1$ polynomial given values at each of the $n$ roots of unity in $O(n \log n)$ steps.

```
\begin{align*}
\text{coefficient representation} & \rightarrow \text{point-value representation} \\
\text{Ons log n} & \rightarrow \text{Olog xy} \\
\end{align*}
```

Polyomial multiplication

Theorem. Can multiply two degree $n-1$ polynomials in $O(n \log n)$ steps.

Proof. Multiply by $\omega^n$ to make $n$ a power of 2.

```
\begin{align*}
\text{coefficient representation} & \rightarrow \text{point-value representation} \\
\text{2 FFTs} & \rightarrow \text{Olog n} \\
\text{point-value representation} & \rightarrow \text{inverse FFT} \\
\text{On log n} & \rightarrow \text{Olog xy} \\
\end{align*}
```

FFT in practice?
FFT in practice

Fastest Fourier transform in the West. [Friske and Johnson]
- Optimized C library.
- Features: DFT, DCT, real, complex, any size, any dimension.
- Won 1999 Wilkinson Prize for Numerical Software.
- Portable, competitive with vendor tuned code.

Implementation details.
- Core algorithm is nonrecursive version of Cooley-Tukey.
- Instead of executing predetermined algorithm, it evaluates your hardware and uses a special-purpose compiler to generate an optimized algorithm catered to "shape" of the problem.
- Runs in $O(n \log n)$ time, even when $n$ is prime.
- Multidimensional FFTs.

FFT in practice

Integer multiplication, redux

Integer multiplication. Given two $n$-bit integers $a = a_{n-1} \ldots a_0$ and $b = b_{n-1} \ldots b_0$, compute their product $a \cdot b$.

Convolution algorithm.
- Form two polynomials.
- Note: $a \in \mathbb{Z}[x]$, $b \in \mathbb{Z}[x]$.
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $(12) = a \cdot b$.
- Running time: $O(n \log n)$ complex arithmetic operations.

Theory. [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.
Theory. [Furer 2007] $O(n \log n 2^{\log^* n})$ bit operations.

Practice. [GNU Multiple Precision Arithmetic Library]
It uses brute force, Karatsuba, and FFT, depending on the size of $n$. 