COMP251: Divide-and-Conquer (3)

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Kleinberg & Tardos, 2005)
Dot product

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

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

\[
a \cdot b = \sum_{i=1}^{n} a_i b_i
\]

\[
\begin{align*}
a &= \begin{bmatrix} .70 & .20 & .10 \end{bmatrix} \\
b &= \begin{bmatrix} .30 & .40 & .30 \end{bmatrix} \\
a \cdot b &= (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32
\end{align*}
\]

**Remark.** Grade-school dot product algorithm is asymptotically optimal.
Matrix multiplication

Matrix multiplication. Given two $n$-by-$n$ matrices $A$ and $B$, compute $C = AB$.

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

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

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

$$\begin{bmatrix}
.59 & .32 & .41 \\
.31 & .36 & .25 \\
.45 & .31 & .42
\end{bmatrix} = \begin{bmatrix}
.70 & .20 & .10 \\
.30 & .60 & .10 \\
.50 & .10 & .40
\end{bmatrix} \times \begin{bmatrix}
.80 & .30 & .50 \\
.10 & .40 & .10 \\
.10 & .30 & .40
\end{bmatrix}$$

Q. Is grade-school matrix multiplication algorithm asymptotically optimal?
Block matrix multiplication

\[
\begin{bmatrix}
152 & 158 & 164 & 170 \\
504 & 526 & 548 & 570 \\
856 & 894 & 932 & 970 \\
1208 & 1262 & 1316 & 1370
\end{bmatrix}
= \begin{bmatrix}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
12 & 13 & 14 & 15
\end{bmatrix}
\times
\begin{bmatrix}
16 & 17 & 18 & 19 \\
20 & 21 & 22 & 23 \\
24 & 25 & 26 & 27 \\
28 & 29 & 30 & 31
\end{bmatrix}
\]

\[
C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix}
0 & 1 \\
4 & 5
\end{bmatrix}
\times
\begin{bmatrix}
16 & 17 \\
20 & 21
\end{bmatrix}
+ \begin{bmatrix}
2 & 3 \\
6 & 7
\end{bmatrix}
\times
\begin{bmatrix}
24 & 25 \\
28 & 29
\end{bmatrix}
= \begin{bmatrix}
152 & 158 \\
504 & 526
\end{bmatrix}
\]
Matrix multiplication: warmup

To multiply two $n$-by-$n$ matrices $A$ and $B$:

- Divide: partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply 8 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
= \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\times
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21})
\]
\[
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22})
\]
\[
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21})
\]
\[
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

Running time. Apply case 1 of Master Theorem.

\[
T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)
\]

$\Rightarrow T(n) = \Theta(n^3)$
Strassen's trick

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

(plus 11 additions and 7 subtractions)

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \times
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = P_5 + P_4 - P_2 + P_6 \\
C_{12} = P_1 + P_2 \\
C_{21} = P_3 + P_4 \\
C_{22} = P_1 + P_5 - P_3 - P_7
\]

**Pf.** \( C_{12} = P_1 + P_2 \)

\[
= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}
\]

\[
= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark
\]

\[
P_1 \leftarrow A_{11} \times (B_{12} - B_{22}) \\
P_2 \leftarrow (A_{11} + A_{12}) \times B_{22} \\
P_3 \leftarrow (A_{21} + A_{22}) \times B_{11} \\
P_4 \leftarrow A_{22} \times (B_{21} - B_{11}) \\
P_5 \leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
P_6 \leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
P_7 \leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12})
\]
Strassen's algorithm

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

\textbf{if } (n = 1) \textbf{ return } A \times B.

Partition \(A\) and \(B\) into 2-by-2 block matrices.

\[
P_1 \leftarrow \text{STRASSEN}(n / 2, A_{11}, (B_{12} - B_{22})).
\]

\[
P_2 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{12}), B_{22}).
\]

\[
P_3 \leftarrow \text{STRASSEN}(n / 2, (A_{21} + A_{22}), B_{11}).
\]

\[
P_4 \leftarrow \text{STRASSEN}(n / 2, A_{22}, (B_{21} - B_{11})).
\]

\[
P_5 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{22}) \times (B_{11} + B_{22})).
\]

\[
P_6 \leftarrow \text{STRASSEN}(n / 2, (A_{12} - A_{22}) \times (B_{21} + B_{22})).
\]

\[
P_7 \leftarrow \text{STRASSEN}(n / 2, (A_{11} - A_{21}) \times (B_{11} + B_{12})).
\]

\[
C_{11} = P_5 + P_4 - P_2 + P_6.
\]

\[
C_{12} = P_1 + P_2.
\]

\[
C_{21} = P_3 + P_4.
\]

\[
C_{22} = P_1 + P_5 - P_3 - P_7.
\]

\textbf{return } C.
Analysis of Strassen's algorithm

**Theorem.** Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two $n$-by-$n$ matrices.

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

$$T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

**Q.** What if $n$ is not a power of 2?

**A.** Could pad matrices with zeros.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$
Strassen's algorithm: practice

Implementation issues.
  • Sparsity.
  • Caching effects.
  • Numerical stability.
  • Odd matrix dimensions.
  • Crossover to classical algorithm when $n$ is "small".

Common misperception. "Strassen is only a theoretical curiosity."
  • Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2,048$.
  • Range of instances where it's useful is a subject of controversy.
**Linear algebra reductions**

**Matrix multiplication.** Given two $n$-by-$n$ 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 multiplication</td>
<td>$A \times B$</td>
<td>$\Theta(MM(n))$</td>
</tr>
<tr>
<td>matrix inversion</td>
<td>$A^{-1}$</td>
<td>$\Theta(MM(n))$</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(MM(n))$</td>
</tr>
<tr>
<td>LU decomposition</td>
<td>$A = LU$</td>
<td>$\Theta(MM(n))$</td>
</tr>
<tr>
<td>least squares</td>
<td>$\min</td>
<td></td>
</tr>
</tbody>
</table>

Numerical linear algebra problems with the same complexity as matrix multiplication.


**Fast matrix multiplication: theory**

**Q.** Multiply two 2-by-2 matrices with 7 scalar multiplications?

**A.** Yes! [Strassen 1969]

\[ \Theta(n^{\log_2 7}) = O(n^{2.807}) \]

**Q.** Multiply two 2-by-2 matrices with 6 scalar multiplications?

**A.** Impossible. [Hopcroft and Kerr 1971]

\[ \Theta(n^{\log_2 6}) = O(n^{2.59}) \]

**Q.** Multiply two 3-by-3 matrices with 21 scalar multiplications?

**A.** Unknown.

\[ \Theta(n^{\log_3 21}) = O(n^{2.77}) \]

**Begun, the decimal wars have.** [Pan, Bini et al, Schönhage, ...]

- Two 20-by-20 matrices with 4,460 scalar multiplications.
  \[ O(n^{2.805}) \]
- Two 48-by-48 matrices with 47,217 scalar multiplications.
  \[ O(n^{2.7801}) \]
- A year later.
- December 1979.
- January 1980.
### 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>?</td>
<td>brute force</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>1969</td>
<td>Strassen</td>
<td>$O(n^{2.808})$</td>
</tr>
<tr>
<td>1978</td>
<td>Pan</td>
<td>$O(n^{2.796})$</td>
</tr>
<tr>
<td>1979</td>
<td>Bini</td>
<td>$O(n^{2.780})$</td>
</tr>
<tr>
<td>1981</td>
<td>Schönhage</td>
<td>$O(n^{2.522})$</td>
</tr>
<tr>
<td>1982</td>
<td>Romani</td>
<td>$O(n^{2.517})$</td>
</tr>
<tr>
<td>1982</td>
<td>Coppersmith-Winograd</td>
<td>$O(n^{2.496})$</td>
</tr>
<tr>
<td>1986</td>
<td>Strassen</td>
<td>$O(n^{2.479})$</td>
</tr>
<tr>
<td>1989</td>
<td>Coppersmith-Winograd</td>
<td>$O(n^{2.376})$</td>
</tr>
<tr>
<td>2010</td>
<td>Strother</td>
<td>$O(n^{2.3737})$</td>
</tr>
<tr>
<td>2011</td>
<td>Williams</td>
<td>$O(n^{2.3727})$</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>$O(n^{2+\varepsilon})$</td>
</tr>
</tbody>
</table>

Number of floating-point operations to multiply two $n$-by-$n$ matrices
Fourier analysis

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

\[ y(t) = \frac{2}{\pi} \sum_{k=1}^{N} \frac{\sin kt}{k} \quad N = 100 \]
Euler's identity

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

**Sinusoids.** Sum of sine and cosines = sum of complex exponentials.
Time domain vs. frequency domain

Signal. [touch tone button 1] \[ y(t) = \frac{1}{2} \sin(2\pi \cdot 697 \ t) + \frac{1}{2} \sin(2\pi \cdot 1209 \ t) \]

Time domain.

Frequency domain.

Reference: Cleve Moler, Numerical Computing with MATLAB
Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]

Magnitude of discrete Fourier transform.

Reference: Cleve Moler, Numerical Computing with MATLAB
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 life as we know it would be very different without the FFT.”

— Charles van Loan
Fast Fourier transform: brief history

Gauss (1805, 1866). Analyzed periodic motion of asteroid Ceres.


An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^n factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^n was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an N x N matrix which can be factored into sparse matrices, where m is proportional to log N. This results in a procedure requiring a number of operations proportional to N log N rather than N^3.

paper published only after IBM lawyers decided not to set precedent of patenting numerical algorithms
(conventional wisdom: nobody could make money selling software!)

Importance not fully realized until advent of digital computers.
Polynomials: coefficient representation

**Polynomial.** [coefficient representation]

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

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

**Add.** \(O(n)\) arithmetic operations.

\[
A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1) x + \cdots + (a_{n-1} + b_{n-1}) x^{n-1}
\]

**Evaluate.** \(O(n)\) using Horner's method.

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

**Multiply (convolve).** \(O(n^2)\) using brute force.

\[
A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i, \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}
\]
DEMONSTRATIO NOVA
THEOREMATIS
OMNEM FUNCTIONEM ALGEBRAICAM
RATIONalem INTEGRAM
VNIVS VARIABILIS
IN FACTORES REALES PRIMI VEL SECUNDI GRADVS
RESOLVI POSSE

AVCTORE
CAROLO FRIDERICO GAUSS
HELMSTADII
APVD C. G. FLECKEISEN. 1799

1.
Quaelibet aequatio algebraica determinata reduci potest ad formam \( x^m + Ax^{m-1} + Bx^{m-2} + \text{etc.} + M = 0 \), ita vt \( m \) sit numerus integer positius. Si partem primam huius aequationis per \( x \) denotamus, aequationisque \( X = 0 \) per plures valores inaequales ipsius \( x \) satisficer supponimus, puta ponendo \( x = \alpha, \ x = \beta, \ x = \gamma \) etc. functio \( X \) per productum e factoribus \( x - \alpha, \ x - \beta, \ x - \gamma \) etc. divisibilis erit. Vice versa, si productum e pluribus factoribus simplicibus \( x - \alpha, \ x - \beta, \ x - \gamma \) etc. functionem \( X \) metitur: aequationi \( X = 0 \) satisfiet, aequando ipsam \( x \) cuiunque quantitatum \( \alpha, \beta, \gamma \) etc. Denique si \( X \) producto ex \( m \) factoribus talibus simplicibus aequalis est (siue omnes diversi sint, siue quidam ex ipsis identici): alii factores simplices praeter hos functionem \( X \) metiri non poterunt. Quamobrem aequatio \( m^{th} \) gradus plures quam \( m \) radices habere nequit; simul vero patet, aequationem \( m^{th} \) gradus pauciores radices habere posse, etsi \( X \) in \( m \) factores simplices resolubilis sit:

"New proof of the theorem that every algebraic rational integral function in one variable can be resolved into real factors of the first or the second degree."
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 $x$
Polynomials: point-value representation

**Polynomial.** [point-value representation]

\[ A(x): (x_0, y_0), \ldots, (x_{n-1}, y_{n-1}) \]
\[ B(x): (x_0, z_0), \ldots, (x_{n-1}, z_{n-1}) \]

**Add.** \( O(n) \) arithmetic operations.

\[ A(x) + B(x): (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1}) \]

**Multiply (convolve).** \( O(n) \), but need \( 2n - 1 \) points.

\[ A(x) \times B(x): (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1}) \]

**Evaluate.** \( O(n^2) \) using Lagrange's formula.

\[ A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k} (x - x_j)}{\prod_{j \neq k} (x_k - x_j)} \]
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>$O(n^2)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>point-value</td>
<td>$O(n)$</td>
<td>$O(n^2)$</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 \( a_0 + a_1 x + ... + a_{n-1} x^{n-1} \), evaluate it at \( n \) distinct points \( x_0, ..., x_{n-1} \).

\[
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{bmatrix} =
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{bmatrix}
\]

**Running time.** \( O(n^2) \) for matrix-vector multiply (or \( n \) Horner's).
Converting between two representations: brute force

**Point-value ⇒ 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.

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  \vdots \\
  y_{n-1}
\end{bmatrix} = 
\begin{bmatrix}
  1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\
  1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\
  1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1}
\end{bmatrix} 
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  \vdots \\
  a_{n-1}
\end{bmatrix}
\]

Vandermonde matrix is invertible iff \( x_i \) distinct

**Running time.** \( O(n^3) \) for Gaussian elimination.

or \( O(n^{2.3727}) \) via fast matrix multiplication
Divide-and-conquer

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

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

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

- \( A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7. \)
- \( A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3. \)
- \( A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3. \)
- \( A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2). \)
Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at $n$ distinct points $x_0, \ldots, x_{n-1}$.

Divide. Break up polynomial into even and odd powers.

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

Intuition. Choose two points to be ±1.

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

Can evaluate polynomial of degree $\leq n$ at 2 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 1 point.
Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $a_0 + a_1 x + ... + a_{n-1} x^{n-1}$, evaluate it at $n$ distinct points $x_0, ..., x_{n-1}$. 

Divide. Break up polynomial into even and odd powers.

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

Intuition. Choose four complex points to be $\pm 1$, $\pm i$.

- $A(1) = A_{even}(1) + 1 A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1 A_{odd}(1)$.
- $A(i) = A_{even}(-1) + i A_{odd}(-1)$.
- $A(-i) = A_{even}(-1) - i A_{odd}(-1)$.

Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 2 points.
### Discrete Fourier transform

**Coefficient ⇒ point-value.** Given a polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at $n$ distinct points $x_0, \ldots, x_{n-1}$.

**Key idea.** Choose $x_k = \omega^k$ where $\omega$ is principal $n^{th}$ root of unity.

\[
\begin{bmatrix}
  y_0 \\
  y_1 \\
  y_2 \\
  y_3 \\
  \vdots \\
  y_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
  1 & 1 & 1 & 1 & \cdots & 1 \\
  1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{n-1} \\
  1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(n-1)} \\
  1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(n-1)} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \cdots & \omega^{(n-1)(n-1)}
\end{bmatrix}
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  a_3 \\
  \vdots \\
  a_{n-1}
\end{bmatrix}
\]

DFT

Fourier matrix $F_n$
Roots of unity

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

**Fact.** The $n^{th}$ roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

**Pf.** $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

**Fact.** The $\frac{1}{2}n^{th}$ roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$. 

![Diagram of roots of unity](image-url)
Fast Fourier transform

Goal. Evaluate a degree \( n - 1 \) polynomial \( A(x) = a_0 + ... + a_{n-1} x^{n-1} \) at its \( n^{th} \) roots of unity: \( \omega^0, \omega^1, ..., \omega^{n-1} \).

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

Conquer. Evaluate \( A_{\text{even}}(x) \) and \( A_{\text{odd}}(x) \) at the \( \frac{1}{2}n^{th} \) roots of unity: \( \nu^0, \nu^1, ..., \nu^{n/2-1} \).

Combine.
- \( A(\omega^k) = A_{\text{even}}(\nu^k) + \omega^k A_{\text{odd}}(\nu^k), \ 0 \leq k < n/2 \)
- \( A(\omega^{k+\frac{1}{2}n}) = A_{\text{even}}(\nu^k) - \omega^k A_{\text{odd}}(\nu^k), \ 0 \leq k < n/2 \)

\[ \nu^k = (\omega^k)^2 \]
\[ \nu^k = (\omega^{k+\frac{1}{2}n})^2 \]
\[ \omega^{k+\frac{1}{2}n} = -\omega^k \]
FFT: implementation

\[
\text{FFT}(n, a_0, a_1, a_2, \ldots, a_{n-1})
\]

\[\text{IF} \ (n = 1) \ \text{RETURN} \ a_0.\]

\[\begin{align*}
(e_0, e_1, \ldots, e_{n/2-1}) & \gets \text{FFT}(n/2, a_0, a_2, a_4, \ldots, a_{n-2}). \\
(d_0, d_1, \ldots, d_{n/2-1}) & \gets \text{FFT}(n/2, a_1, a_3, a_4, \ldots, a_{n-1}).
\end{align*}\]

\[\text{FOR} \ k = 0 \ \text{TO} \ n/2 - 1.
\]

\[\begin{align*}
\omega^k & \leftarrow e^{2\pi ik/n}. \\
y_k & \leftarrow e_k + \omega^k d_k. \\
y_{k + n/2} & \leftarrow e_k - \omega^k d_k.
\end{align*}\]

\[\text{RETURN} \ (y_0, y_1, y_2, \ldots, y_{n-1}).\]
**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.** $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$

assumes $n$ is a power of 2

---

**FFT: summary**

$O(n \log n)$

$\begin{align*}
\text{coefficient representation} & \quad \text{point-value representation} \\
(a_0, a_1, \ldots, a_{n-1}) & \quad (x_0, y_0, \ldots, x_{n-1}, y_{n-1})
\end{align*}$
FFT: recursion tree

```
a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7
```

```
a_0, a_2, a_4, a_6
```

```
a_1, a_3, a_5, a_7
```

```
a_0, a_4
```

```
a_2, a_6
```

```
a_1, a_5
```

```
a_3, a_7
```

"bit-reversed" order

```
a_0  a_4
000  100
```

```
a_2  a_6
010  110
```

```
a_1  a_5
001  101
```

```
a_3  a_7
011  111
```

inverse perfect shuffle
Inverse discrete Fourier transform

**Point-value ⇒ coefficient.** Given $n$ distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

\[
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
\vdots \\
a_{n-1}
\end{bmatrix}
= 
\begin{bmatrix}
1 & 1 & 1 & 1 & \ldots & 1 \\
1 & \omega^1 & \omega^2 & \omega^3 & \ldots & \omega^{n-1} \\
1 & \omega^2 & \omega^4 & \omega^6 & \ldots & \omega^{2(n-1)} \\
1 & \omega^3 & \omega^6 & \omega^9 & \ldots & \omega^{3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{3(n-1)} & \ldots & \omega^{(n-1)(n-1)}
\end{bmatrix}
^{-1}
\begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
y_3 \\
\vdots \\
y_{n-1}
\end{bmatrix}
\]

**Inverse DFT**  **Fourier matrix inverse $(F_n)^{-1}$**
Inverse discrete Fourier transform

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

$$G_n = \frac{1}{n} \begin{bmatrix}
1 & 1 & 1 & 1 & \cdots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \omega^{-3} & \cdots & \omega^{-(n-1)} \\
1 & \omega^{-2} & \omega^{-4} & \omega^{-6} & \cdots & \omega^{-2(n-1)} \\
1 & \omega^{-3} & \omega^{-6} & \omega^{-9} & \cdots & \omega^{-3(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \omega^{-3(n-1)} & \cdots & \omega^{-(n-1)(n-1)}
\end{bmatrix}$$

$F_n / \sqrt{n}$ is a unitary matrix

**Consequence.** To compute inverse FFT, apply same algorithm but use $\omega^{-1} = e^{-2\pi i/n}$ as principal $n^{th}$ root of unity (and divide the result by $n$).
Inverse FFT: proof of correctness

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

Pf.

$$\left(F_n G_n\right)_{kk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{kj} \omega^{-jk'} = \frac{1}{n} \sum_{j=0}^{n-1} \omega^{(k-k')j} = \begin{cases} 1 & \text{if } k = k' \\ 0 & \text{otherwise} \end{cases}$$

summation lemma (below)

Summation lemma. Let $\omega$ be a principal $n^{th}$ root of unity. Then

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

Pf.

- If $k$ is a multiple of $n$ then $\omega^k = 1 \Rightarrow$ series sums to $n$.
- Each $n^{th}$ root of unity $\omega^k$ is a root of $x^n - 1 = (x - 1) (1 + x + x^2 + ... + x^{n-1})$.
- if $\omega^k \neq 1$ we have: $1 + \omega^k + \omega^{k(2)} + ... + \omega^{k(n-1)} = 0 \Rightarrow$ series sums to 0. ■
Inverse FFT: implementation

**Note.** Need to divide result by $n$.

\[
\text{\textbf{\textsc{Inverse-Fft}}}(n, a_0, a_1, a_2, \ldots, a_{n-1})
\]

\[
\text{IF } (n = 1) \text{ RETURN } a_0.
\]

\[
(e_0, e_1, \ldots, e_{n/2-1}) \leftarrow \text{\textbf{\textsc{Inverse-Fft}}}(n/2, a_0, a_2, a_4, \ldots, a_{n-2}).
\]

\[
(d_0, d_1, \ldots, d_{n/2-1}) \leftarrow \text{\textbf{\textsc{Inverse-Fft}}}(n/2, a_1, a_3, a_4, \ldots, a_{n-1}).
\]

\[
\text{FOR } k = 0 \text{ TO } n/2 - 1.
\]

\[
\omega_k \leftarrow e^{-2\pi i k/n}.
\]

\[
y_k \leftarrow (e_k + \omega_k d_k).
\]

\[
y_{k+n/2} \leftarrow (e_k - \omega_k d_k).
\]

RETURN \((y_0, y_1, y_2, \ldots, y_{n-1})\).
Inverse FFT: summary

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

assumes $n$ is a power of 2
**Polynomial multiplication**

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

**Pf.**

pad with 0s to make \( n \) a power of 2

---

**coefficient representation**

\[
\begin{align*}
a_0, a_1, \ldots, a_{n-1} \\
b_0, b_1, \ldots, b_{n-1}
\end{align*}
\]

2 FFTs

\( O(n \log n) \)

---

**coefficient representation**

\[
\begin{align*}
c_0, c_1, \ldots, c_{2n-2}
\end{align*}
\]

inverse FFT

\( O(n \log n) \)

---

**point-value multiplication**

\( O(n) \)

---

**representation**

\[
\begin{align*}
A(\omega^0), \ldots, A(\omega^{2n-1}) \\
B(\omega^0), \ldots, B(\omega^{2n-1})
\end{align*}
\]

---

**representation**

\[
\begin{align*}
C(\omega^0), \ldots, C(\omega^{2n-1})
\end{align*}
\]
FFT in practice?
FFT in practice

Fastest Fourier transform in the West. [Frigo 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.

http://www.fftw.org
Integer multiplication, redux

**Integer multiplication.** Given two $n$-bit integers $a = a_{n-1} \ldots a_1 a_0$ and $b = b_{n-1} \ldots b_1 b_0$, compute their product $a \cdot b$.

**Convolution algorithm.**
- Form two polynomials. $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$
- Note: $a = A(2), b = B(2)$. $B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$
- Compute $C(x) = A(x) \cdot B(x)$.
- Evaluate $C(2) = a \cdot b$.
- Running time: $O(n \log n)$ complex arithmetic operations.

**Theory.** [Schönhage-Strassen 1971] $O(n \log n \log \log n)$ bit operations.

**Theory.** [Fürer 2007] $n \log n \ 2^{O(\log^* n)}$ bit operations.

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