

CS101 Algorithms and Data Structures

Divide and Conquer
Textbook Ch 1.4



Outline

- Master Theorem
- Integer Multiplication
- Matrix Multiplication
- Convolution and FFT

Master Theorem



- Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = a T\left(\frac{n}{b}\right) + f(n)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$.

- Terms.
 - $a \geq 1$ is the number of subproblems.
 - $b \geq 2$ is the factor by which the subproblem size decreases.
 - $f(n) \geq 0$ is the work to divide and combine subproblems.
- Recursion tree. [assuming n is a power of b]
 - a = branching factor.
 - a^k = number of subproblems at level k .
 - $1 + \log_b n$ levels.
 - n / b^k = size of subproblem at level k .

Master Theorem

- If $T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d)$ for constants $a > 0, b > 1, d \geq 0$, then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$



Outline

- Master Theorem
- Integer Multiplication
- Matrix Multiplication
- Convolution and FFT

Integer addition and subtraction

- Addition. Given two n -bit integers x and y , compute $x + y$.
- Subtraction. Given two n -bit integers x and y , compute $x - y$.
- Grade-school algorithm. $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
<hr/>								
1	0	1	0	1	0	0	1	0

Remark. Grade-school addition and subtraction algorithms are optimal.

Integer multiplication

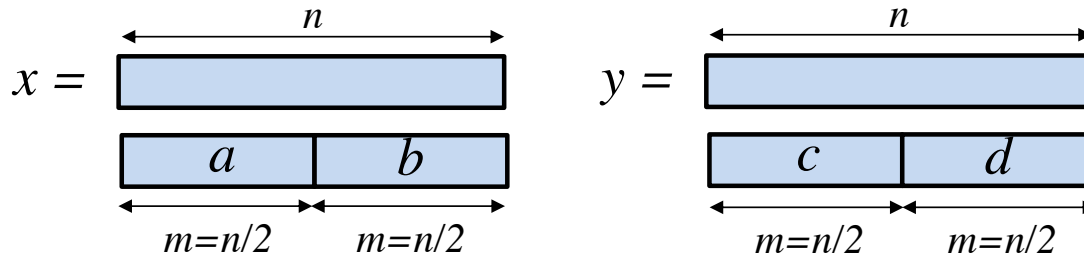
- Multiplication. Given two n -bit integers x and y , compute $x \times y$.
- Grade-school algorithm (long multiplication). $\Theta(n^2)$ bit operations.

[illegible]

六、吸生

Divide-and-conquer multiplication

- To multiply two n -bit integers x and y :
 - Divide x and y into low- and high-order bits.
 - Multiply **four** $\frac{1}{2}n$ -bit integers, recursively.
 - Add and shift to obtain result.



Then

$$x y = (2^m a + b) (2^m c + d) = 2^{2m} \boxed{ac} + 2^m (\boxed{bc} + \boxed{ad}) + \boxed{bd}$$

1
2
3
4

Ex. $x = \underbrace{1\ 0\ 0\ 0}_a \underbrace{1\ 1\ 0\ 1}_b$ $y = \underbrace{1\ 1\ 1\ 0}_c \underbrace{0\ 0\ 0\ 1}_d$

MULTIPLY(x, y, n)

IF ($n = 1$)

RETURN $x y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{MULTIPLY}(a, c, m)$.

$f \leftarrow \text{MULTIPLY}(b, d, m)$.

$g \leftarrow \text{MULTIPLY}(b, c, m)$.

$h \leftarrow \text{MULTIPLY}(a, d, m)$.

RETURN $2^{2m} e + 2^m (g + h) + f$.

← $\Theta(n)$

← bit opt

← $4 T(\lceil n / 2 \rceil)$

← $\Theta(n)$

Question 1

- How many bit operations to multiply two n -bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ & \text{if } n > 1 \end{cases}$$




A. $T(n) = \Theta(n^{1/2})$.

B. $T(n) = \Theta(n \log n)$.

C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.

D. $T(n) = \Theta(n^2)$.

Recursion Tree for Integer Multiplication

Level		Problem #	Problem size	Work
0		1	n	n
1		4	(n/2)	4(n/2)
2		4 ²	(n/4)	4 ² (n/4)
...	4 4 4 4
k=log ₂ n	4 ^k =4 ^{log₂ n}	1	4 ^k (n / 2 ^k)= 4 ^{log₂ n} (n / 2 ^{log₂ n})

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$

$$\text{Total work} = \sum_1^k 4^k(n / 2^k) = \sum_1^k n2^k = n^2$$

Karatsuba trick

- To multiply two n -bit integers x and y :
 - Divide x and y into low- and high-order bits.
 - To compute middle term $bc + ad$, use identity:

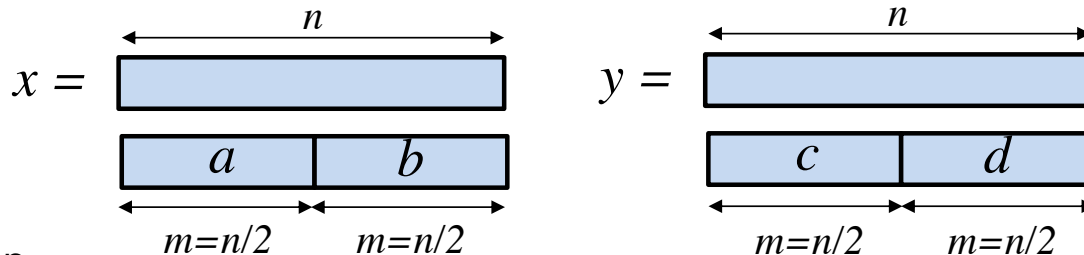
$$bc + ad = ac + bd - (a - b)(c - d)$$

- Multiply only **three** $\frac{1}{2}n$ -bit integers, recursively.



Divide-and-conquer multiplication

- To multiply two n -bit integers x and y :
 - Divide x and y into low- and high-order bits.
 - Multiply **four** $\frac{1}{2}n$ -bit integers, recursively.
 - Add and shift to obtain result.



Then

$$\begin{aligned}
 xy &= (2^m a + b)(2^m c + d) = 2^{2m} \underbrace{ac}_{\text{1}} + 2^m (\underbrace{bc}_{\text{2}} + \underbrace{ad}_{\text{3}}) + \underbrace{bd}_{\text{4}} \\
 &= 2^{2m} \underbrace{ac}_{\text{1}} + 2^m (\underbrace{ac}_{\text{1}} + \underbrace{bd}_{\text{2}} - \underbrace{(a-b)(c-d)}_{\text{3}}) + \underbrace{bd}_{\text{2}}
 \end{aligned}$$

KARATSUBA-MULTIPLY(x, y, n)

IF ($n = 1$)

 RETURN $x \cdot y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow \text{KARATSUBA-MULTIPLY}(a, c, m)$.

$f \leftarrow \text{KARATSUBA-MULTIPLY}(b, d, m)$.

$g \leftarrow \text{KARATSUBA-MULTIPLY}(\lfloor a - b \rfloor, \lfloor c - d \rfloor, m)$.

 Flip sign of g if needed.

 RETURN $2^{2m} e + 2^m (e + f - g) + f$.

← $\Theta(n)$

← $3 T(\lceil n / 2 \rceil)$

← $\Theta(n)$

Question 2

- How many bit operations to multiply two n -bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$




A. $T(n) = \Theta(n^{1/2})$.

B. $T(n) = \Theta(n \log n)$.

☒ C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.

D. $T(n) = \Theta(n^2)$.

Recursion Tree for Integer Multiplication

Level		Problem #	Problem size	Work
0		1	n	n
1		3	(n/2)	3(n/2)
2		3 ²	(n/4)	3 ² (n/4)
...	3 3 ... 3 3
k=log ₂ n	3 ^k =3 ^{log₂n}	1	3 ^k (n / 2 ^k)= 3 ^{log₂n} (n / 2 ^{log₂n})

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$


$$\text{Total work} = \sum_0^{k-1} 4^k(n / 2^k) = \sum_0^{k-1} n(3/2)^k = n^{\log_2 3} = n^{1.585}$$

Outline

- Master Theorem
- Integer Multiplication
- **Matrix Multiplication**
- Convolution and FFT

Dot product

- Dot product. Given two length- n vectors a and b , compute $c = a \cdot b$.
- Grade-school. $\Theta(n)$ arithmetic operations.

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


$$a = [.70 \quad .20 \quad .10]$$

$$b = [.30 \quad .40 \quad .30]$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Matrix multiplication

- Matrix multiplication.

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

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

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

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

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

Block matrix multiplication

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

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

Block matrix multiplication

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

$$\begin{array}{c} B_{22} \\ \swarrow \end{array}
 C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

$$C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22})$$

Block 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.

why sub,

n -by- n matrices

$$C = A \times B$$

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

$\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}}$$

8 matrix multiplications
(of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions
(of $\frac{1}{2}n$ -by- $\frac{1}{2}n$ matrices)

Strassen's trick

- Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

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

scalars

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$C_{12} = P_1 + P_2$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_1 + P_5 - P_3 - P_7$$

$$P_1 \leftarrow A_{11} (B_{12} - B_{22})$$

$$P_2 \leftarrow (A_{11} + A_{12}) B_{22}$$

$$P_3 \leftarrow (A_{21} + A_{22}) B_{11}$$

$$P_4 \leftarrow A_{22} (B_{21} - B_{11})$$

$$P_5 \leftarrow (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$P_6 \leftarrow (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$P_7 \leftarrow (A_{11} - A_{21}) (B_{11} + B_{12})$$

7 scalar multiplications

assume n is a power of 2

STRASSEN(n, A, B)

IF ($n = 1$) RETURN $A B$.

Partition A and B into $\frac{1}{2}n$ -by- $\frac{1}{2}n$ blocks.

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

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

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

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

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

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

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

$7 T(n / 2) + \Theta(n^2)$

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

$\Theta(n^2)$

RETURN C .

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

Analysis of Strassen's algorithm

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

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

- When n is not a power of 2, pad matrices with zeros to be n' -by- n' , where $n \leq n' < 2n$ and n' is a power of 2.

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

Strassen's algorithm: practice

- **Implementation issues.**
 - Sparsity.
 - Caching.
 - n not a power of 2.
 - Numerical stability.
 - Non-square matrices.
 - Storage for intermediate submatrices.
 - Crossover to classical algorithm when n is “small.”
 - Parallelism for multi-core and many-core architectures.
- Common misperception. “*Strassen's algorithm is only a theoretical curiosity.*”
 - Apple reports 8x speedup when $n \approx 2,048$.
 - Range of instances where it's useful is a subject of controversy.

Question 3

matrix

- Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n -by- n matrices?

A. $\Theta(n^{\log_3 21})$

B. $\Theta(n^{\log_2 21})$

C. $\Theta(n^{\log_9 21})$

D. $\Theta(n^2)$

Question 4

- Is it possible to multiply two 3-by-3 matrices using only 21 scalar multiplications?

A. Yes.

B. No.

C. Unknown.

✓
matrix

Fast matrix multiplication: theory

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

- A. Yes! [Strassen 1969]

$$\Theta(n^{\log_2 7}) = O(n^{2.81})$$

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

- A. Impossible. [Hopcroft–Kerr, Winograd 1971]

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

- Begun, the decimal wars have. [Pan 1978, Bini et al., Schönhage, ...]

- Two 70-by-70 matrices with 143,640 scalar multiplications. $O(n^{2.7962})$

- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$

- A year later. $O(n^{2.7799})$

- December 1979. $O(n^{2.521813})$

- January 1980. $O(n^{2.521801})$

History of arithmetic complexity of matrix multiplication

year	algorithm	arithmetic operations
1858	"grade school"	$O(n^3)$
1969	Strassen	$O(n^{2.808})$
1978	Pan	$O(n^{2.796})$
1979	Bini	$O(n^{2.780})$
1981	Schönhage	$O(n^{2.522})$
1982	Romani	$O(n^{2.517})$
1982	Coppersmith–Winograd	$O(n^{2.496})$
1986	Strassen	$O(n^{2.479})$
1989	Coppersmith–Winograd	$O(n^{2.3755})$
2010	Strother	$O(n^{2.3737})$
2011	Williams	$O(n^{2.372873})$
2014	Le Gall	$O(n^{2.372864})$
	???	$O(n^{2+\varepsilon})$

Galactic
algorithms

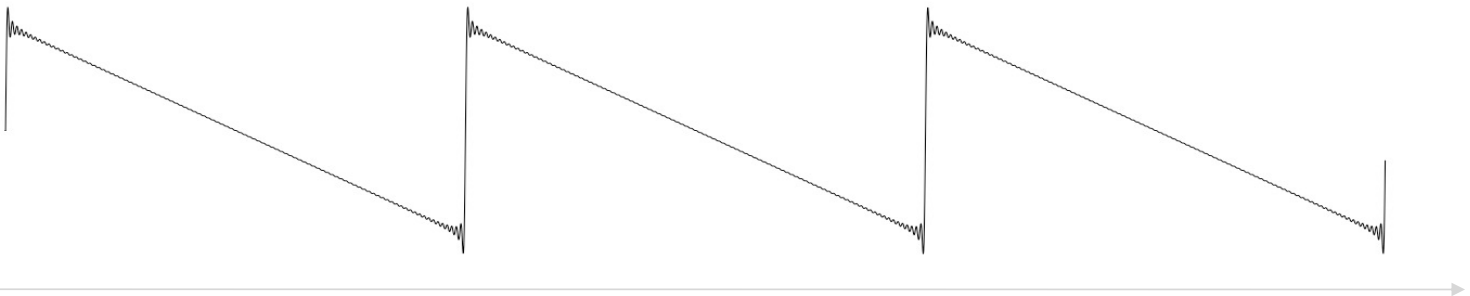
Number of arithmetic operations to multiply two n -by- n matrices

Outline

- Master Theorem
- Integer Multiplication
- Matrix Multiplication
- Convolution and FFT

Outline

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



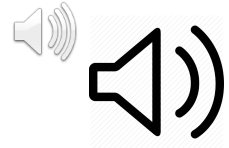
$$y(t) = \frac{2}{\pi} \sum_{k=1}^n \frac{\sin kt}{k}$$

$$n = 100$$

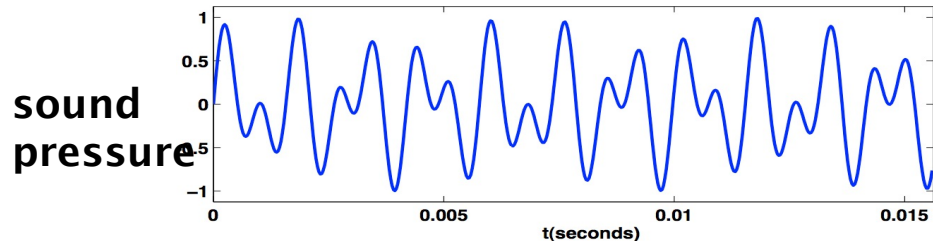
Time domain vs. frequency domain

- Signal. [touch tone button 1]

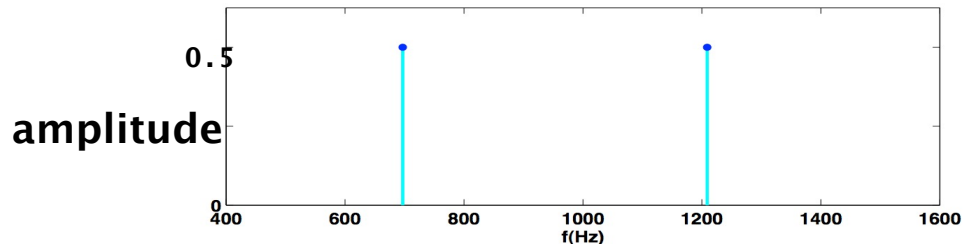
$$y(t) = \frac{1}{2} \sin(2\pi \cdot 697 t) + \frac{1}{2} \sin(2\pi \cdot 1209 t)$$



- Time domain.



- Frequency domain.



Fast Fourier Transform (FFT)

- **FFT.** Fast way to convert between time domain and frequency domain.
- **Alternate viewpoint.** Fast way to multiply and evaluate **polynomials**.

we take this approach

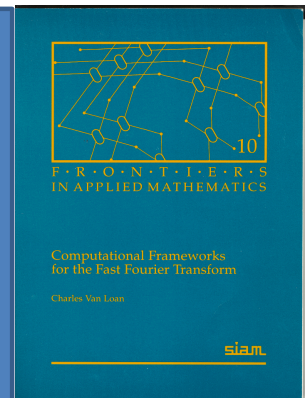


Fast Fourier transform: applications

- 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.
- Runge–König (1924). Laid theoretical groundwork.
- Danielson–Lanczos (1942). Efficient algorithm, x-ray crystallography.
- Cooley–Tukey (1965). Detect nuclear tests in Soviet Union and track submarines. Rediscovered and popularized FFT.

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^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m 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 $N \times N$ matrix which can be factored into m 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^2 .



Importance not fully realized until emergence of digital computers.

Polynomials: coefficient representation

- **Univariate polynomial.** [coefficient representation]

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

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_{n-1}x^{n-1}$$

- **Addition.** $O(n)$ arithmetic operations.

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

- **Evaluation.** $O(n)$ using Horner's method.

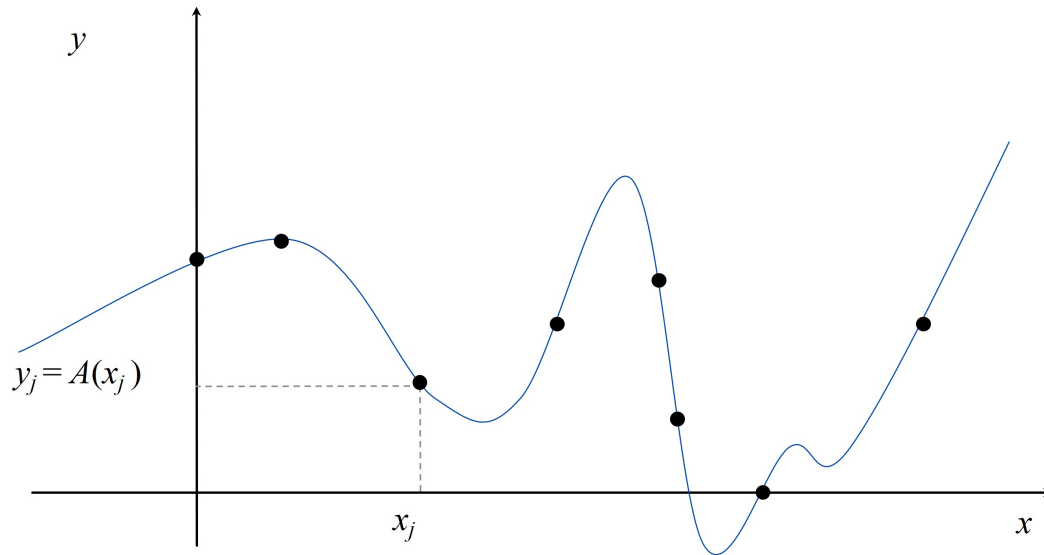
$$A(x) = a_0 + (x(a_1 + x(a_2 + \dots + x(a_{n-2} + x(a_{n-1}))) \dots))$$

- **Multiplication (linear convolution).** $O(n^2)$ using brute force.

$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i \quad \text{where} \quad c_i = \sum_{j=0}^i a_j b_{i-j}$$

Polynomials: point-value representation

- **Fundamental theorem of algebra.** A degree n univariate polynomial with complex coefficients has exactly n complex roots.
- **Corollary.** A degree $n - 1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x .



Polynomials: point-value representation

- **Univariate polynomial.** [point-value representation]

$$A(x): (x_0, y_0), \dots, (x_{n-1}, y_{n-1})$$

$$B(x): (x_0, z_0), \dots, (x_{n-1}, z_{n-1})$$

- **Addition.** $O(n)$ arithmetic operations.

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

- **Multiplication.** $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

$$A(x) B(x): (x_0, y_0 \cdot z_0), \dots, (x_{2n-1}, y_{2n-1} \cdot z_{2n-1})$$

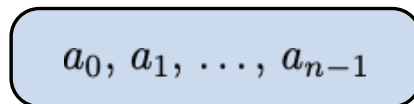
Converting between two representations

- **Tradeoff.** Either fast evaluation or fast multiplication. We want both!

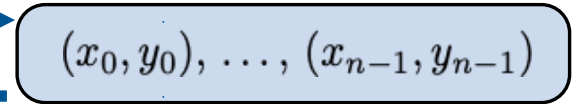
representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

- **Goal.** Efficient conversion between two representations \Rightarrow all ops fast.

coefficient representation



point-value representation

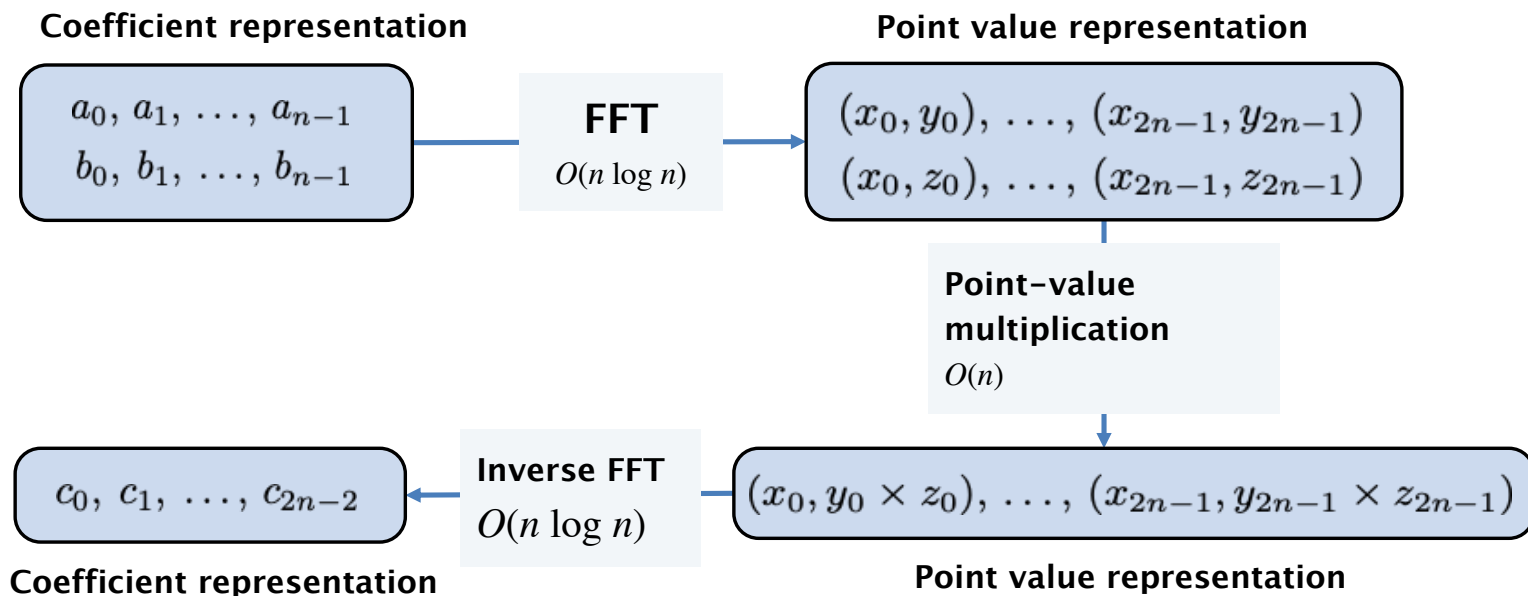


Converting between two representations

- Application. Polynomial multiplication (coefficient representation).

$$\text{DFT: } X(k) = \sum_{n=0}^{N-1} \boxed{x(n)} e^{-i\frac{2\pi k}{N}n} = \sum_{n=0}^{N-1} x(n) \boxed{W_N^{kn}} \quad W_N^k = e^{-i\frac{2\pi k}{N}}$$

a_0, a_1, \dots, a_{n-1}
 $1, x^1, x^2, \dots, x^{n-1}$



Converting between two representations: brute force

- **Coefficient \Rightarrow point-value.** Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, 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. $O(n^2)$ via matrix–vector multiply (or n Horner's).

分位不是目标

方法之后 设计并美

Converting between two representations: brute force

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

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

Vandermonde matrix is invertible if x_i distinct

- **Running time.** $O(n^3)$ via Gaussian elimination.

or $O(n^{2.38})$ via fast matrix multiplication

Question 5



- Which divide-and-conquer approach to use to evaluate polynomials?

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7.$$

A. Divide polynomial into low- and high-degree terms.

$$A_{low}(x) = a_0 + a_1x + a_2x^2 + a_3x^3.$$

$$A_{high}(x) = a_4 + a_5x + a_6x^2 + a_7x^3.$$

B. Divide polynomial into even- and odd-degree terms.

$$A_{even}(x) = a_0 + a_2x + a_4x^2 + a_6x^3.$$

$$A_{odd}(x) = a_1 + a_3x + a_5x^2 + a_7x^3.$$

C. Either A or B.

D. Neither A nor B.

Divide-and-conquer

- Decimation in time. Divide into even- and odd- degree terms.

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

- $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$

- $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$

- $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$

- Decimation in frequency. Divide into low- and high-degree terms.


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

- $A_{\text{low}}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$

- $A_{\text{high}}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3.$

- $A(x) = A_{\text{low}}(x) + x^4 A_{\text{high}}(x).$

Coefficient to point-value representation: intuition

- **Coefficient \Rightarrow point-value.** Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} . 

We get to choose which ones!

- **Divide.** Break up polynomial into even- and odd-degree terms.

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

$$- A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$$

$$- A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$$


$$- A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

$$- A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2).$$


- **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 $n-1$ at 2 points by evaluating two polynomials of degree $\frac{1}{2}n - 1$ at only 1 point. 

Coefficient to point-value representation: intuition

- **Coefficient \Rightarrow point-value.** Given a polynomial $A(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$, evaluate it at n distinct points x_0, \dots, x_{n-1} .


We get to choose which ones!
- **Divide.** Break up polynomial into even- and odd-degree terms.
 - $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7.$
 - $A_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3.$
 - $A_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3.$
 - $A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$
 - $A(-x) = A_{\text{even}}(x^2) - x A_{\text{odd}}(x^2).$
- **Intuition.** Choose two points to be ± 1 .

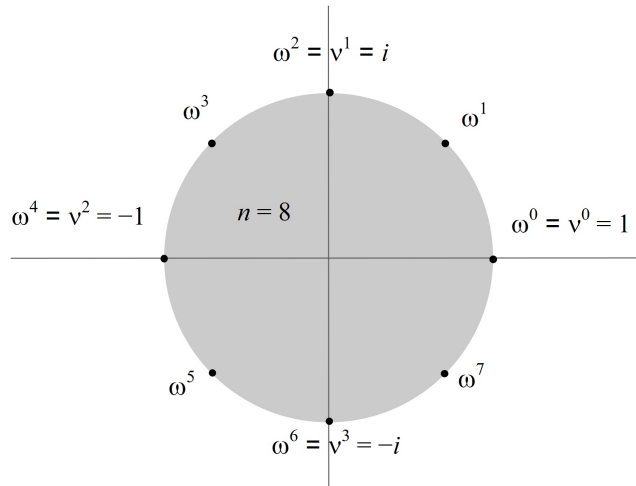
<ul style="list-style-type: none"> – $A(1) = A_{\text{even}}(1) + 1 A_{\text{odd}}(1).$ – $A(-1) = A_{\text{even}}(1) - 1 A_{\text{odd}}(1).$ – $A(i) = A_{\text{even}}(-1) + i A_{\text{odd}}(-1).$ – $A(-i) = A_{\text{even}}(-1) - i A_{\text{odd}}(-1).$ 		 <p>Can evaluate polynomial of degree $n-1$ at 4 points by evaluating two polynomials of degree $\frac{1}{2}n - 1$ at only 2 points.</p>
--	--	--

Euler's identity

- Euler's identity. $e^{ix} = \cos x + i \sin x$.
- Sinusoids. Sum of sine and cosines = sum of complex exponentials.

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: $\omega^0, \omega^1, \dots, \omega^{n-1}$ 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 $\frac{1}{2}n^{\text{th}}$ roots of unity are: $v^0, v^1, \dots, v^{n/2-1}$ where $v = \omega^2 = e^{4\pi i / n}$.



$$\omega^k = -\omega^{k + \frac{1}{2}n}$$

Fast Fourier transform

- **Goal.** Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \dots, \omega^{n-1}$.
- **Divide.** Break up polynomial into even- and odd-degree terms.
 - $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \dots + a_{n-2} x^{n/2-1}$.
 - $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \dots + a_{n-1} x^{n/2-1}$.
 - $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
 - $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$.
- **Conquer.** Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $1/2 n^{th}$ roots of unity: $v^0, v^1, \dots, v^{n/2-1}$.
- **Combine.**

$\omega^k = \omega^{k + 1/2 n}$

 - $y_k = A(\omega^k) = A_{even}(v^k) + \omega^k A_{odd}(v^k), \quad 0 \leq k < n/2$.
 - $y_{k + 1/2 n} = A(\omega^{k + 1/2 n}) = A_{even}(v^k) - \omega^k A_{odd}(v^k), \quad 0 \leq k < n/2$.

FFT: implementation

- **Goal.** Evaluate a degree $n - 1$ polynomial $A(x) = a_0 + \dots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \dots, \omega^{n-1}$.
 - $y_k = A(\omega^k) = A_{\text{even}}(v^k) + \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2.$
 - $y_{k + \frac{1}{2}n} = A(\omega^{k + \frac{1}{2}n}) = A_{\text{even}}(v^k) - \omega^k A_{\text{odd}}(v^k), \quad 0 \leq k < n/2.$

FFT($n, a_0, a_1, a_2, \dots, a_{n-1}$)

IF ($n = 1$) *RETURN* a_0 .

$(e_0, e_1, \dots, e_{n/2-1}) \leftarrow \text{FFT}(n/2, a_0, a_2, a_4, \dots, a_{n-2}).$

$(d_0, d_1, \dots, d_{n/2-1}) \leftarrow \text{FFT}(n/2, a_1, a_3, a_5, \dots, a_{n-1}).$

FOR $k = 0$ *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 $(y_0, y_1, y_2, \dots, y_{n-1}).$

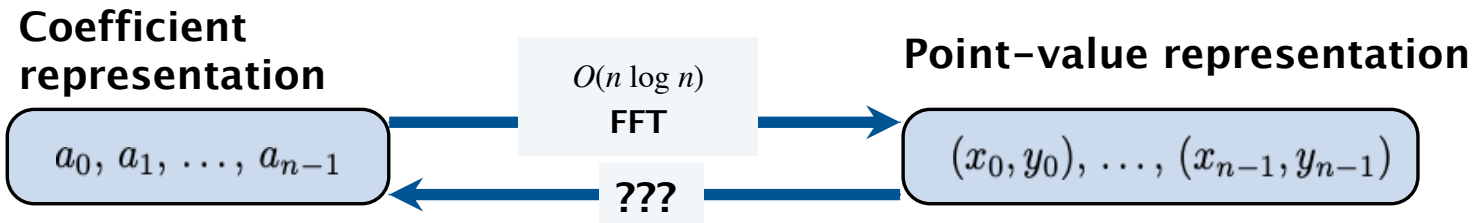
$2 T(n/2)$

$\Theta(n)$

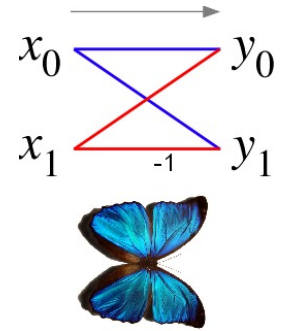
FFT: summary

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

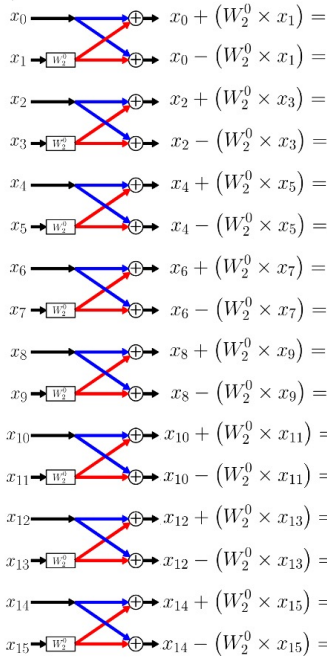
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



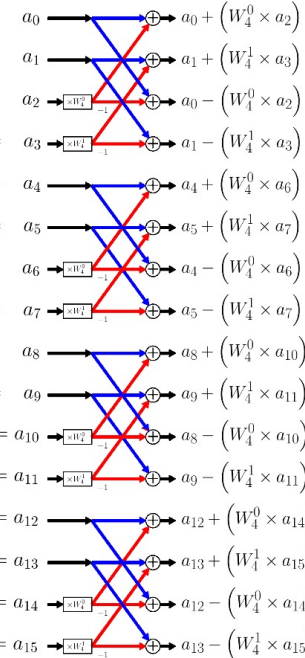
FFT Butterfly



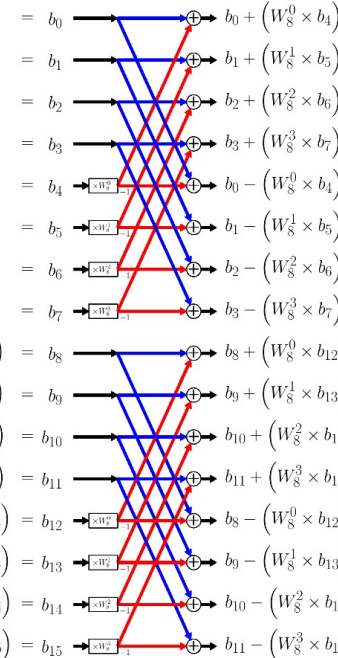
Stage 0



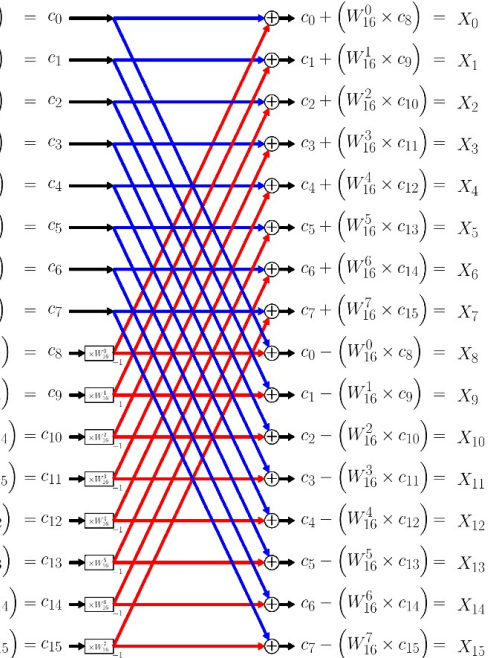
Stage 1



Stage 2



Stage 3



Ref: http://en.wikipedia.org/wiki/Butterfly_diagram