

**CSE 421**

# **Introduction to Algorithms**

## **Lecture 9: Divide and Conquer Matrix & Integer Multiplication**

# Algorithm Design Techniques

## Divide & Conquer

- Divide instance into subparts.
- Solve the parts recursively.
- Conquer by combining the answers

## Last Time: Solving Divide and Conquer Recurrences

Master Theorem: Suppose that  $T(n) = a \cdot T(n/b) + O(n^k)$  for  $n > b$ .

- If  $a < b^k$  then  $T(n)$  is  $O(n^k)$
- If  $a = b^k$  then  $T(n)$  is  $O(n^k \log n)$
- If  $a > b^k$  then  $T(n)$  is  $O(n^{\log_b a})$

Binary search:  $a = 1, b = 2, k = 0$  so  $a = b^k$ : Solution:  $O(n^0 \log n) = O(\log n)$

Mergesort:  $a = 2, b = 2, k = 1$  so  $a = b^k$ : Solution:  $O(n^1 \log n) = O(n \log n)$

# Matrix Multiplication

$$\begin{matrix} n \\ \left[ \begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right] \cdot \left[ \begin{array}{cccc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array} \right] \end{matrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

Multiplying  $n \times n$  matrices: Entry  $c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}$

- $n^3$  multiplications
- $n^3 - n^2$  additions

# Multiplying Matrices

```
for  $i \leftarrow 1$  to  $n$ 
  for  $j \leftarrow 1$  to  $n$ 
     $C[i, j] \leftarrow 0$ 
    for  $k \leftarrow 1$  to  $n$ 
       $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$ 
    endfor
  endfor
endfor
```

Can we improve this with divide and conquer?

# Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

$\frac{n}{2} \times \frac{n}{2}$  matrix multiplications inside the  $n \times n$  computation

# Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

$\frac{n}{2} \times \frac{n}{2}$  matrix multiplications inside the  $n \times n$  computation

# Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

A<sub>11</sub> A<sub>12</sub> B<sub>11</sub> B<sub>12</sub>  
A<sub>21</sub> A<sub>22</sub> B<sub>21</sub> B<sub>22</sub>

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} & a_{11}b_{14} + a_{12}b_{24} + a_{13}b_{34} + a_{14}b_{44} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} & a_{21}b_{14} + a_{22}b_{24} + a_{23}b_{34} + a_{24}b_{44} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} + a_{34}b_{41} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} + a_{34}b_{42} & a_{31}b_{14} + a_{32}b_{24} + a_{33}b_{34} + a_{34}b_{44} \\ a_{41}b_{11} + a_{42}b_{21} + a_{43}b_{31} + a_{44}b_{41} & a_{41}b_{12} + a_{42}b_{22} + a_{43}b_{32} + a_{44}b_{42} & a_{41}b_{14} + a_{42}b_{24} + a_{43}b_{34} + a_{44}b_{44} \end{bmatrix}$$

A<sub>11</sub>B<sub>11</sub> + A<sub>12</sub>B<sub>21</sub> A<sub>11</sub>B<sub>12</sub> + A<sub>12</sub>B<sub>22</sub>  
A<sub>21</sub>B<sub>11</sub> + A<sub>22</sub>B<sub>21</sub> A<sub>21</sub>B<sub>12</sub> + A<sub>22</sub>B<sub>22</sub>

$\frac{n}{2} \times \frac{n}{2}$  matrix multiplications inside the  $n \times n$  computation



# Multiplying Matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

$\frac{n}{2} \times \frac{n}{2}$  matrix multiplications inside the  $n \times n$  computation

# Multiplying Matrices: Divide and Conquer

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

$\frac{n}{2} \times \frac{n}{2}$  matrix operations inside the  $n \times n$  computation:

8 matrix multiplications:  $T(n/2)$  each

4 matrix additions:  $(n/2)^2$  each; total  $O(n^2)$

Recurrence:  $T(n) = 8T(n/2) + O(n^2)$

Apply Master Theorem:

$a = 8, b = 2, k = 2$ . Now  $b^k = 2^2 = 4$  so  $a > b^k$  and  $\log_b a = 3$ .

Solution:  $T(n)$  is  $O(n^{\log_b a}) = O(n^3)$  *No savings!*

$n^{\log_b a} = n^3$

# Strassen's Divide and Conquer (1968)

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{pmatrix}$$

Key observations: This picture looks just like  $2 \times 2$  matrix multiplication!  
and the number of multiplications is what really matters

**Strassen:** Can multiply  $2 \times 2$  matrices using only 7 multiplications!  
(and many more additions)

Recurrence:  $T(n) = 7T(n/2) + O(n^2)$

$n^{2.81}$

Apply Master Theorem:

$a = 7, b = 2, k = 2$  so solution  $T(n)$  is  $O(n^{\log_2 7}) = O(n^{2.8074})!$

# Strassen's Divide and Conquer (1968)

$$P_1 \leftarrow \underline{A_{12}}(B_{11} + \underline{B_{21}}); \quad P_2 \leftarrow A_{21}(B_{12} + B_{22})$$

$$P_3 \leftarrow (\underline{A_{11}} - A_{12})\underline{B_{11}}; \quad P_4 \leftarrow (A_{22} - A_{21})B_{22}$$

$$P_5 \leftarrow (A_{22} - A_{12})(B_{21} - B_{22})$$

$$P_6 \leftarrow (A_{11} - A_{21})(B_{12} - B_{11})$$

$$P_7 \leftarrow (A_{21} - A_{12})(B_{11} + B_{22})$$

$$\underline{C_{11}} \leftarrow \underline{P_1 + P_3}; \quad C_{12} \leftarrow P_2 + P_3 + P_6 - P_7$$

$$C_{21} \leftarrow P_1 + P_4 + P_5 + P_7; \quad C_{22} \leftarrow P_2 + P_4$$

# Fast Matrix Multiplication

Using Strassen's  $O(n^{2.8074})$  algorithm:

- Practical for exact calculations on large matrices
  - Not numerically stable with approximations
- Stop recursion when  $n < 32$  and use simple algorithm instead
  - This kind of stopping of recursion is typical for divide and conquer

$O(n^w)$

Decades of theoretical improvements since:

- Best current time  $O(n^{2.3728596})$
- None of these improvements is practical (require  $n$  in the millions and more)

**Open:** Is there an  $O(n^2)$  time matrix multiplication algorithm?

# Integer Multiplication

```
  695273
× 123412
-----
 1390546
 695273
2781092
2085819
1390546
 695273
-----
85805031476
```

Decimal

```
  110110
× 101110
-----
           0
  110110
 110110
110110
   0
 110110
-----
100110110100
```

Binary

Elementary school algorithm

$O(n^2)$  time for  $n$ -bit integers

# Integer Multiplication: Divide and Conquer

Break up each  $n$ -bit integer  $x$  and  $y$  into two  $n/2$ -bit integers



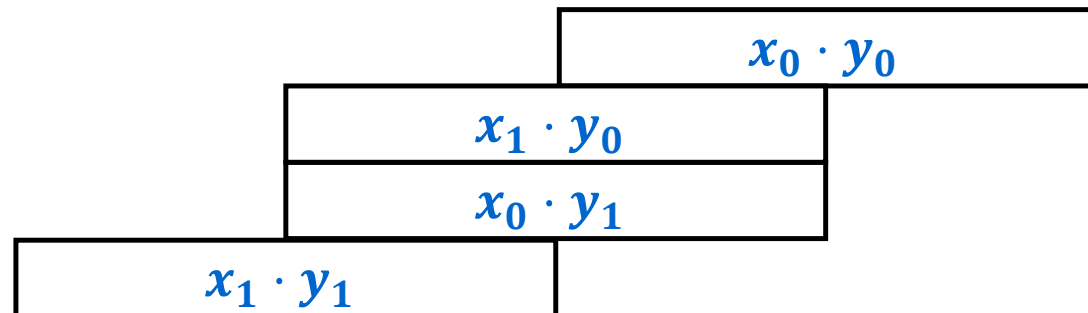
so  $x = x_1 \cdot 2^{n/2} + x_0$  and  $y = y_1 \cdot 2^{n/2} + y_0$ .

$$\begin{aligned} \text{Then } x \cdot y &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 2^{n/2} + x_0 \cdot y_0 \end{aligned}$$

Divide and conquer:

- Solve 4 size  $n/2$  subproblems
- Shift answers, add results  $O(n)$

Recurrence:  $T(n) = 4T(n/2) + O(n)$



# Integer Multiplication: Divide and Conquer

Break up each  $n$ -bit integer  $x$  and  $y$  into two  $n/2$ -bit integers



so  $x = x_1 \cdot 2^{n/2} + x_0$  and  $y = y_1 \cdot 2^{n/2} + y_0$ .

Then  $x \cdot y = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0)$

$$= x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 2^{n/2} + x_0 \cdot y_0$$

Divide and conquer:

- Solve **4** size  $n/2$  subproblems
- Shift answers, add results  $O(n)$

Recurrence:  $T(n) = 4T(n/2) + O(n)$

Master Theorem:

- $a = 4, b = 2, k = 1$
- $a > b^k$

So  $T(n)$  is  $O(n^{\log_b a}) = O(n^2)$

*No savings!*



# Karatsuba's Divide and Conquer Algorithm (1963)

We want to compute  $x \cdot y = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 2^{n/2} + x_0 \cdot y_0$

For divide and conquer, we already have to compute  $x_1 \cdot y_1$  and  $x_0 \cdot y_0$

We just need that middle term  $(x_1 \cdot y_0 + x_0 \cdot y_1)$  which looks like two multiplications.

If we compute  $(x_1 + x_0) \cdot (y_1 + y_0) = x_1 \cdot y_1 + (x_1 \cdot y_0 + x_0 \cdot y_1) + x_0 \cdot y_0$  then we can cancel off the first and last parts to get the middle term we need and we only use one multiplication.

# Karatsuba's Divide and Conquer Algorithm (1963)

We want to compute  $x \cdot y = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 2^{n/2} + x_0 \cdot y_0$

## Karatsuba:

Use only **3** “half-size” multiplications by computing middle term more efficiently

- Multiply to get  $t_2 = x_1 \cdot y_1$ .  $T(n/2)$
- Multiply to get  $t_0 = x_0 \cdot y_0$ .  $T(n/2)$
- Add to get  $x_1 + x_0$  and  $y_1 + y_0$ .  $O(n)$  :  $n/2 + 1$  bit answers
- Multiply to get  $s = (x_1 + x_0) \cdot (y_1 + y_0)$   $T(n/2 + 1)$   
 $= x_1 \cdot y_1 + (x_1 \cdot y_0 + x_0 \cdot y_1) + x_0 \cdot y_0$
- Compute  $t_1 = s - t_2 - t_0$  which equals  $x_1 \cdot y_0 + x_0 \cdot y_1$   $O(n)$
- Shift  $t_1$  and  $t_2$ , add results to  $t_0$   $O(n)$

Recurrence:  $T(n) = 3 T(n/2 + 1) + O(n)$  Solution:  $T(n)$  is  $O(n^{\log_2 3}) = O(n^{1.585})$

# Fast Multiplication and the Fast Fourier Transform (FFT)

Fast integer multiplication is used for multi-precision arithmetic

- Relevant input-size measure: # of 64-bit words of precision

Karatsuba's algorithm is not the fastest for integer multiplication

- Fastest is  $O(n \log n)$  time based on the **Fast Fourier Transform (FFT)**
  - [Schoenhage-Strassen 1971, Fürer 2007, Harvey-Hoeven 2019]
  - Many messy details. We'll focus on FFT itself!

**Fast Fourier Transform (FFT)** [Cooley-Tukey 1967]

- Efficient conversion back-and-forth between a signal and its frequencies.
- $O(n \log n)$  time algorithm for multiplying **polynomials**.
- Practical variant is standard for computing the Discrete Cosine Transform (DCT)
  - Workhorse of modern signal processing.

# Polynomial Multiplication

Variable  $x$

Polynomial  $p(x)$ : integer combination of powers of  $x$

- e.g., quadratic polynomial  $p(x) = 3x^2 + 2x + 1$
- Represent by a vector of integer coefficients  $[3, 2, 1]$

**Polynomial Multiplication:**

**Given:**  $p(x) = a_{n-1} x^{n-1} + a_{n-1} x^{n-2} + \dots + a_2 x^2 + a_1 x + a_0$

and  $q(x) = b_{n-1} x^{n-1} + b_{n-1} x^{n-2} + \dots + b_2 x^2 + b_1 x + b_0$

**Compute:** (Vector of coefficients of) polynomial  $r(x) = p(x) q(x)$

e.g.,  $(3x + 1)(2x + 3) = 6x^2 + 9x + 2x + 3 = 6x^2 + 11x + 3$

Basic algorithm: Compute all  $n^2$  products  $a_i b_j$  and collect terms.

# Essential Idea for FFT: Polynomial Interpolation

Suppose  $r$  is an unknown degree  $n - 1$  polynomial with coefficients  $c_{n-1}, \dots, c_0$

- $r(x) = c_{n-1}x^{n-1} + \dots + c_2x^2 + c_1x + c_0$

Suppose you have values of  $r$  at  $n$  distinct points:  $y_0, \dots, y_{n-1}$

- $r(y_0), \dots, r(y_{n-1})$

This gives a system of  $n$  linear equations in  $c_{n-1}, \dots, c_0$

$$c_{n-1}y_0^{n-1} + \dots + c_2y_0^2 + c_1y_0 + c_0 = r(y_0)$$

$$c_{n-1}y_1^{n-1} + \dots + c_2y_1^2 + c_1y_1 + c_0 = r(y_1)$$

...

$$c_{n-1}y_{n-1}^{n-1} + \dots + c_2y_{n-1}^2 + c_1y_{n-1} + c_0 = r(y_{n-1})$$

*green = known  
red = unknown*

**Fact:** If the points are distinct, this system has a unique solution.

# Fast Fourier Transform: Multiplying Polynomials

**FFT**( $p, q, n$ ){

// Assume that  $p$  and  $q$  have degree  $n - 1$

// Depends on good sequence of  $2n$  points  $y_0, y_1, \dots, y_{2n-1}$

Compute evaluations  $p(y_0), \dots, p(y_{2n-1})$

Compute evaluations  $q(y_0), \dots, q(y_{2n-1})$

Multiply values to compute

$r(y_0) = p(y_0) \cdot q(y_0), \dots, r(y_{2n-1}) = p(y_{2n-1}) \cdot q(y_{2n-1})$  }  $O(n)$

Interpolate: Solve systems of equations for  $r(x) = p(x)q(x)$   
given  $r(y_0), \dots, r(y_{2n-1})$  and  $y_0, y_1, \dots, y_{2n-1}$

}

Any set of distinct points suffice. FFT chooses them to make evaluation/interpolation easy.

# FFT: Choosing evaluation points

Computing a single evaluation takes  $O(n)$  time.

~~Complexity of  $n^2$~~

Using  $n$  unrelated points would be  $O(n^2)$  total time

- *No savings!*

Instead use divide and conquer:

- Choose related points and do it recursively on half-size problems
- In the recursion should only have half as many points

Key FFT ideas:

- For every evaluation point  $\omega$ , also include  $-\omega$
- For every evaluation point  $\omega$ , use  $\omega^2$  in the recursive evaluation.
- Half-size problems involve *odd* and *even* degree sub-polynomials

# Key FFT ideas

$$\begin{aligned} p(\omega) &= a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 + a_4\omega^4 + \dots + a_{n-2}\omega^{n-2} + a_{n-1}\omega^{n-1} \\ &= a_0 + a_2\omega^2 + a_4\omega^4 + \dots + a_{n-2}\omega^{n-2} \\ &\quad + a_1\omega + a_3\omega^3 + a_5\omega^5 + \dots + a_{n-1}\omega^{n-1} \\ &= p_{\text{even}}(\omega^2) + \omega p_{\text{odd}}(\omega^2) \end{aligned}$$

$$\begin{aligned} p(-\omega) &= a_0 - a_1\omega + a_2\omega^2 - a_3\omega^3 + a_4\omega^4 - \dots + a_{n-2}\omega^{n-2} - a_{n-1}\omega^{n-1} \\ &= a_0 + a_2\omega^2 + a_4\omega^4 + \dots + a_{n-2}\omega^{n-2} \\ &\quad - (a_1\omega + a_3\omega^3 + a_5\omega^5 + \dots + a_{n-1}\omega^{n-1}) \\ &= p_{\text{even}}(\omega^2) - \omega p_{\text{odd}}(\omega^2) \end{aligned}$$

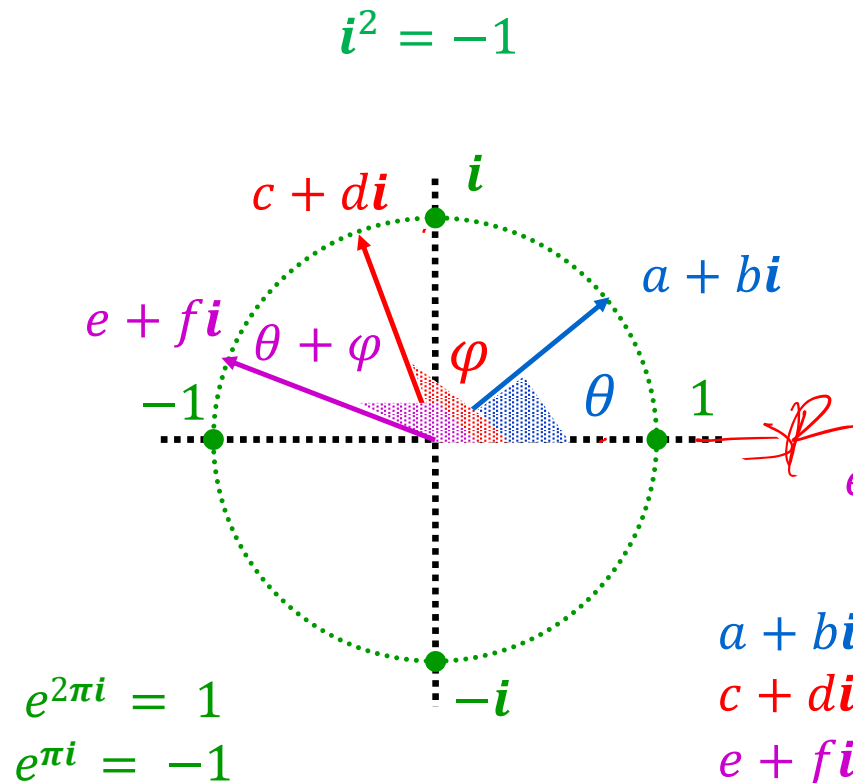
where  $p_{\text{even}}(x) = a_0 + a_2x + a_4x^2 + \dots + a_{n-2}x^{n/2-1}$

and  $p_{\text{odd}}(x) = a_1 + a_3x + a_5x^2 + \dots + a_{n-1}x^{n/2-1}$

To continue recursion, need some of the squares to be the negation of others! **Complex numbers**



# Complex Numbers Review



To multiply complex numbers

- add angles
- multiply lengths  
(only need length 1 for FFT)

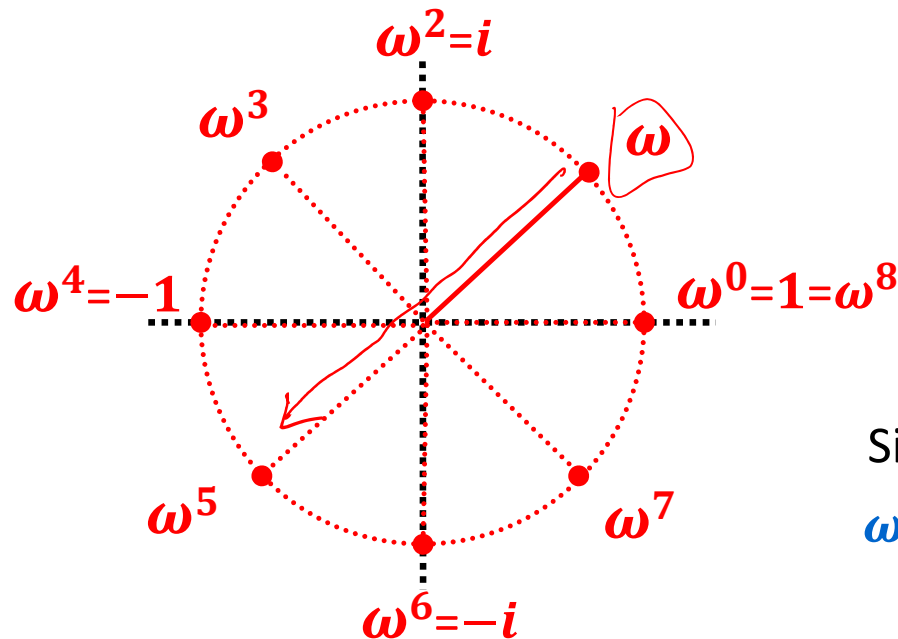
$$e + fi = (a + bi)(c + di)$$

$$a + bi = \cos \theta + i \sin \theta = e^{\theta i}$$

$$c + di = \cos \varphi + i \sin \varphi = e^{\varphi i}$$

$$e + fi = \cos(\theta + \varphi) + i \sin(\theta + \varphi) = e^{(\theta + \varphi)i}$$

# Use powers of $\omega$ “primitive” $n^{\text{th}}$ root of 1: $\omega^n = 1$



$\omega = e^{\frac{2\pi i}{n}} = \cos\left(\frac{2\pi}{n}\right) + i \sin\left(\frac{2\pi}{n}\right)$   
so can explicitly compute with its powers.

$\omega^2$  is a “primitive”  $n/2^{\text{th}}$  root of 1.

Since  $\omega^{n/2} = -1$  we have  
 $\omega^{n/2}, \omega^{n/2+1}, \dots, \omega^{n-1} = -1, -\omega, \dots, -\omega^{n/2-1}$

# FFT Evaluation: Recursion for $n$ a power of 2

## Goal:

- Evaluate  $p$  at  $1, \omega, \omega^2, \omega^3, \dots, \omega^{n-1}$

## Now

- $p_{\text{even}}$  and  $p_{\text{odd}}$  have degree  $n/2 - 1$  where

$$p(\omega^k) = p_{\text{even}}(\omega^{2k}) + \omega^k p_{\text{odd}}(\omega^{2k})$$

$$p(-\omega^k) = p_{\text{even}}(\omega^{2k}) - \omega^k p_{\text{odd}}(\omega^{2k})$$

## Recursive Algorithm

- Evaluate  $p_{\text{even}}$  at  $1, \omega^2, \omega^4, \dots, \omega^{n-2}$
- Evaluate  $p_{\text{odd}}$  at  $1, \omega^2, \omega^4, \dots, \omega^{n-2}$
- Combine to compute  $p$  at  $1, \omega^1, \omega^2, \dots, \omega^{n/2-1}$
- Combine to compute  $p$  at  $-1, -\omega^1, -\omega^2, \dots, -\omega^{n/2-1}$   
(i.e. at  $\omega^{n/2}, \omega^{n/2+1}, \dots, \omega^{n-1}$ )

Powers of  $\omega^2$

$$T(n) = 2T(n/2) + O(n)$$

so

$$T(n) \text{ is } O(n \log n)$$

# Fast Fourier Transform: Multiplying Polynomials

**FFT**( $p, q, n/2$ ) {

// Assume that  $p$  and  $q$  have degree  $n/2 - 1$

Compute evaluations  $p(1), \dots, p(\omega^{n-1})$

Compute evaluations  $q(1), \dots, q(\omega^{n-1})$

Multiply values to compute

$$r(1) = p(1) \cdot q(1), \dots, r(\omega^{n-1}) = p(\omega^{n-1}) \cdot q(\omega^{n-1})$$

Interpolate: Solve systems of equations for  $r(x) = p(x)q(x)$   
given  $r(1), \dots, r(\omega^{n-1})$

}  $O(n \log n)$

}  $O(n)$

}

# Polynomial Interpolation

System of  $n$  linear equations in  $c_{n-1}, \dots, c_0$ :

$$c_{n-1} \mathbf{1} + \dots + c_2 \mathbf{1} + c_1 \mathbf{1} + c_0 = r(\mathbf{1})$$

$$c_{n-1} \omega^{n-1} + \dots + c_2 \omega^2 + c_1 \omega + c_0 = r(\omega)$$

...

$$c_{n-1} \omega^{(n-1)k} + \dots + c_2 \omega^{2k} + c_1 \omega^k + c_0 = r(\omega^k)$$

...

$$c_{n-1} \dots + \dots + c_2 \dots + c_1 \dots + c_0 = r(\omega^{n-1})$$

Can solve this in a very slick way...

# Interpolation Algorithm

Define a new polynomial

- $s(x) = r(1) + r(\omega) \cdot x + r(\omega^2) \cdot x^2 + \dots + r(\omega^{n-1}) \cdot x^{n-1}$
- Run FFT evaluation for  $s(1), \dots, s(\omega^{n-1})$

$O(n \log n)$

**Claim:** Setting  $c_j = s(\omega^{n-j})/n$  for each  $j$  gives the correct answer.

**Proof:** Then  $s(\omega^{n-j}) = \sum_{i=0}^{n-1} r(\omega^i) \cdot (\omega^{n-j})^i = \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} c_k (\omega^i)^k \cdot (\omega^{n-j})^i$

$$= \sum_{k=0}^{n-1} c_k \sum_{i=0}^{n-1} (\omega^k)^i \cdot (\omega^{-j})^i$$
$$= \sum_{k=0}^{n-1} c_k \sum_{i=0}^{n-1} (\omega^{k-j})^i$$

Now  $\omega^{k-j}$  is a solution to equation  $y^n - 1 = (y - 1)(y^{n-1} + \dots + y + 1) = 0$

If  $k \neq j$  then  $\omega^{k-j} \neq 1$  so  $\sum_{i=0}^{n-1} (\omega^{k-j})^i = 0$ ; if  $k = j$  then  $\sum_{i=0}^{n-1} (\omega^{k-j})^i = n$  ■

# Polynomial Multiplication: Degree 1 gives alt Karatsuba

Given  $p(z) = a_1 \cdot z + a_0$        $q(z) = b_1 \cdot z + b_0$  compute

$$r(z) = a_1 b_1 \cdot z^2 + (a_1 b_0 + a_0 b_1) \cdot z + a_0 b_0$$

Just as Strassen's Algorithm was based on multiplying  $2 \times 2$  matrices with few products, this is based on multiplying degree 1 polynomials using few products.

Have 3 coefficients of  $r$  to compute.

**Idea:** Evaluate each of  $p$  and  $q$  at 3 points,  $0, 1, -1$ , and multiply results

- $r(0) = p(0) \cdot q(0) = a_0 b_0$
- $r(1) = p(1) \cdot q(1) = (a_0 + a_1) (b_0 + b_1)$
- $r(-1) = p(-1) \cdot q(-1) = (a_0 - a_1) (b_0 - b_1)$

Can express  $(a_1 b_0 + a_0 b_1)$  and  $a_1 b_1$  as linear combinations of  $r(0), r(1), r(-1)$