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: $O\left(n^0 \log n\right) = O(\log n)$

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

Matrix Multiplication

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

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

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

- $\boldsymbol{n^3}$ multiplications
- $\boldsymbol{n^3 - n^2}$ additions

Multiplying Matrices

```
for i ← 1 to n
    for j ← 1 to n
        C[i, j] ← 0
        for k ← 1 to n
            C[i, j] ← C[i, j] + A[i, k]·B[k, j]
        endfor
    endfor
endfor
```

Can we improve this with divide and conquer?

Multiplying Matrices

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

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

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

Multiplying Matrices

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

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

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

Multiplying Matrices

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \bullet \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \bullet \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]$$

$$= \left[\begin{array}{cc|cc} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} & \circ & a_{11}b_{14}+a_{12}b_{24}+a_{13}b_{34}+a_{14}b_{44} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} & \circ & a_{21}b_{14}+a_{22}b_{24}+a_{23}b_{34}+a_{24}b_{44} \\ \hline a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} & \circ & a_{31}b_{14}+a_{32}b_{24}+a_{33}b_{34}+a_{34}b_{44} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} & \circ & a_{41}b_{14}+a_{42}b_{24}+a_{43}b_{34}+a_{44}b_{44} \end{array}\right]$$

$$= \left[\begin{array}{c|c} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ \hline A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \end{array}\right]$$

$\frac{n}{2} \times \frac{n}{2}$ 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: $T(n) = 8\,T(n/2) + O(n^2)$

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 $O\left(n^{\log_b a}\right) = O(n^3)$ *No savings!*

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 $\mathbf{2} \times \mathbf{2}$ matrix multiplication!
and the number of multiplications is what really matters

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

Recurrence: $\boldsymbol{T(n) = 7\, T(n/2) + O(n^2)}$

Apply Master Theorem:

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

Strassen's Divide and Conquer (1968)

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

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

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

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

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

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

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

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

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

$$
\begin{array}{r}
695273 \\
\times\ 123412 \\
\hline
1390546 \\
695273 \\
2781092 \\
2085819 \\
1390546 \\
695273 \\
\hline
85805031476
\end{array}
\qquad
\begin{array}{r}
110110 \\
\times\ 101110 \\
\hline
0 \\
110110 \\
110110 \\
110110 \\
0 \\
110110 \\
\hline
100110110100
\end{array}
$$

Decimal　　　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

x_1	x_0		y_1	y_0

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

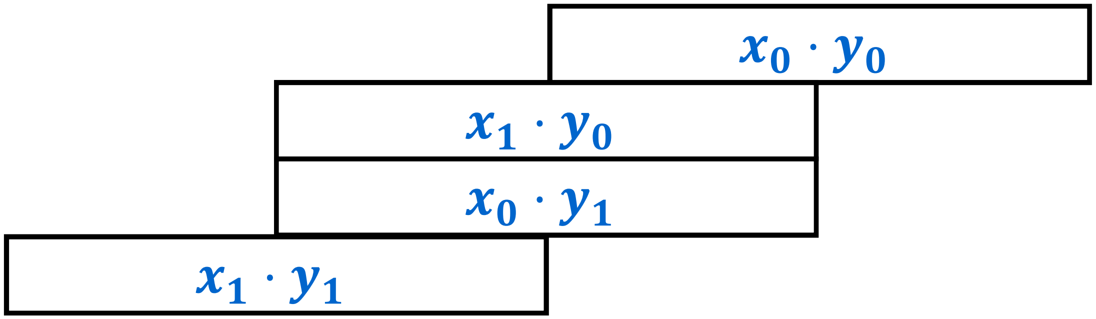
Then $x \cdot y = (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0)$

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

Divide and conquer:

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

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



Integer Multiplication: Divide and Conquer

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

x_1	x_0

y_1	y_0

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

Master Theorem:

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

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

No savings!

Karatsuba's Divide and Conquer Algorithm (1963)

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

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

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

If we compute $\boldsymbol{(x_1 + x_0) \cdot (y_1 + y_0) = x_1 \cdot y_1 + (x_1 \cdot y_0 + x_0 \cdot y_1) + x_0 \cdot y_0}$ 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 $\boldsymbol{x \cdot y = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_0 + x_0 \cdot y_1) \cdot 2^{n/2} + x_0 \cdot y_0}$

Karatsuba:

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

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

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

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 $O(\boldsymbol{n} \log \boldsymbol{n})$ 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.
- $O(\boldsymbol{n} \log \boldsymbol{n})$ 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 $\boldsymbol{p}(x)$: integer combination of powers of x

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

Polynomial Multiplication:

Given: $\boldsymbol{p}(x) = \boldsymbol{a}_{n-1}\, x^{n-1} + \boldsymbol{a}_{n-1}\, x^{n-2} + \cdots + \boldsymbol{a}_2\, x^2 + \boldsymbol{a}_1\, x + \boldsymbol{a}_0$

and $\boldsymbol{q}(x) = \boldsymbol{b}_{n-1}\, x^{n-1} + \boldsymbol{b}_{n-1}\, x^{n-2} + \cdots + \boldsymbol{b}_2\, x^2 + \boldsymbol{b}_1\, x + \boldsymbol{b}_0$

Compute: (Vector of coefficients of) polynomial $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{p}(\boldsymbol{x})\, \boldsymbol{q}(\boldsymbol{x})$

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

Basic algorithm: Compute all $\boldsymbol{n}^{\mathbf{2}}$ products $\boldsymbol{a}_i \boldsymbol{b}_j$ and collect terms.

Essential Idea for FFT: Polynomial Interpolation

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

- $\boldsymbol{r}(x) = \boldsymbol{c_{n-1}} x^{n-1} + \cdots + \boldsymbol{c_2} x^2 + \boldsymbol{c_1} x + \boldsymbol{c_0}$

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

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

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

$$\begin{aligned}
\boldsymbol{c_{n-1} y_0^{n-1}} &+ \ldots + \boldsymbol{c_2 y_0^2} &+ \boldsymbol{c_1 y_0} &+ \boldsymbol{c_0} &= \boldsymbol{r(y_0)} \\
\boldsymbol{c_{n-1} y_1^{n-1}} &+ \ldots + \boldsymbol{c_2 y_1^2} &+ \boldsymbol{c_1 y_1} &+ \boldsymbol{c_0} &= \boldsymbol{r(y_1)} \\
\ldots \\
\boldsymbol{c_{n-1} y_{n-1}^{n-1}} &+ \ldots + \boldsymbol{c_2 y_{n-1}^2} &+ \boldsymbol{c_1 y_{n-1}} &+ \boldsymbol{c_0} &= \boldsymbol{r(y_{n-1})}
\end{aligned}$$

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

$$\left.\begin{array}{l} r(y_0) = p(y_0) \cdot q(y_0), \dots, r(y_{2n-1}) = p(y_{2n-1}) \cdot q(y_{2n-1}) \end{array}\right\} O(n)$$

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 $O(\boldsymbol{n})$ time.

Using $\boldsymbol{n}$ unrelated points would be $O(\boldsymbol{n}^2)$ 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 $\boldsymbol{\omega}$, also include $-\boldsymbol{\omega}$
- For every evaluation point $\boldsymbol{\omega}$, use $\boldsymbol{\omega}^2$ in the recursive evaluation.
- Half-size problems involve *odd* and *even* degree sub-polynomials

Key FFT ideas

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

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

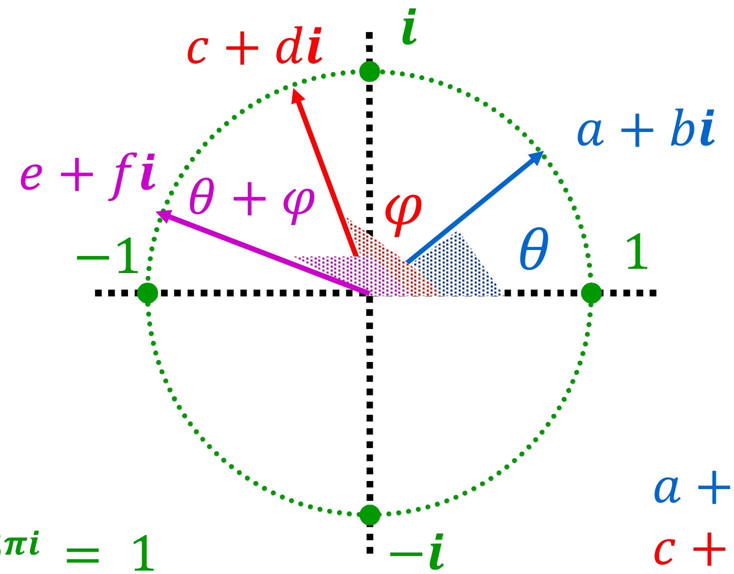
where $p_{even}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2}\, x^{n/2-1}$

and $p_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1}\, x^{n/2-1}$

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

Complex Numbers Review

$$i^2 = -1$$



$e^{2\pi i} = 1$

$e^{\pi i} = -1$

To multiply complex numbers

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

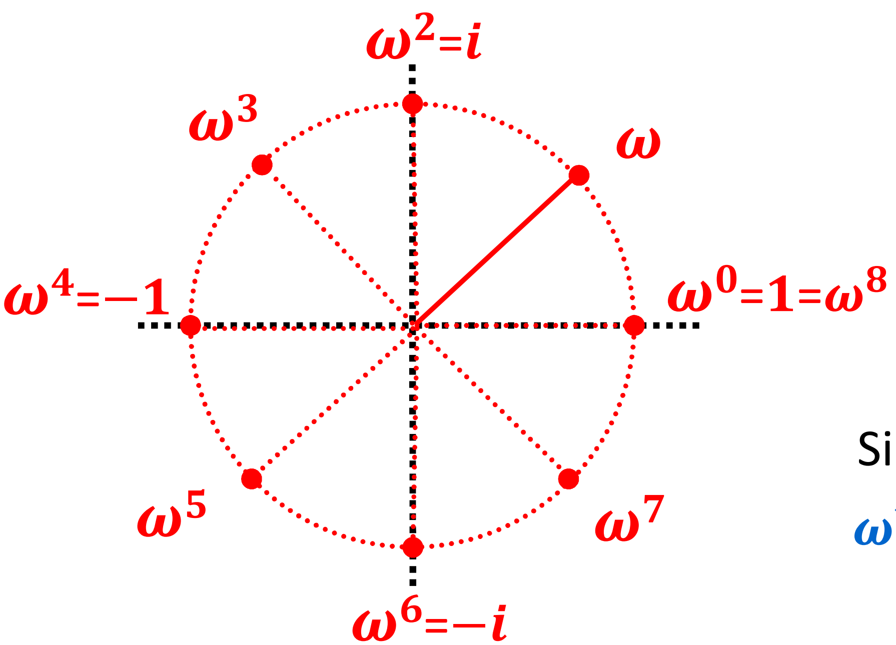
$$e + f\boldsymbol{i} = (a + b\boldsymbol{i})(c + d\boldsymbol{i})$$

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

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

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

Use powers of ω "primitive" n^{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.

ω^2 is a "primitive" $n/2^{\text{th}}$ root of 1.

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

$\omega^{n/2}, \omega^{n/2+1}, \ldots, \omega^{n-1} = -1, -\omega, \ldots, -\omega^{n/2-1}$

FFT Evaluation: Recursion for n a power of 2

Goal:

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

Recursive Algorithm

- Split coefficients of p into polynomials p_{even} and p_{odd} — $O(n)$
- Recursively evaluate p_{even} at $1, \omega^2, \omega^4, \ldots, \omega^{n-2}$ — $T(n/2)$
- Recursively evaluate p_{odd} at $1, \omega^2, \omega^4, \ldots, \omega^{n-2}$ — $T(n/2)$

Powers of ω^2

- Combine to compute p at $1, \omega^1, \omega^2, \ldots, \omega^{n/2-1}$ using $p(\omega^k) = p_{even}(\omega^{2k}) + \omega^k p_{odd}(\omega^{2k})$. — $O(n)$
- Combine to compute p at $\omega^{n/2}, \omega^{n/2+1}, \ldots, \omega^{n-1}$ (equivalently, $-1, -\omega^1, -\omega^2, \ldots, -\omega^{n/2-1}$) using $p(-\omega^k) = p_{even}(\omega^{2k}) - \omega^k p_{odd}(\omega^{2k})$ — $O(n)$

$T(n) = 2\,T(n/2) + O(n)$
so $T(n)$ is $O(n \log n)$

Fast Fourier Transform: Multiplying Polynomials

FFT$(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{n/2})${

// Assume that $\boldsymbol{p}$ and $\boldsymbol{q}$ have degree $\boldsymbol{n/2-1}$

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

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

$\Big\}\ O(\boldsymbol{n}\log \boldsymbol{n})$

Multiply values to compute

$$\boldsymbol{r(1) = p(1) \cdot q(1), \ldots, r\left(\omega^{n-1}\right) = p\left(\omega^{n-1}\right) \cdot q(\omega^{n-1})}$$

$\Big\}\ O(\boldsymbol{n})$

Interpolate: Solve systems of equations for $\boldsymbol{r}(x) = \boldsymbol{p}(x)\boldsymbol{q}(x)$

given $\boldsymbol{r(1), \ldots, r\left(\omega^{n-1}\right)}$

}

Polynomial Interpolation

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

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

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

…

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

…

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

Can solve this in a very slick way...

Interpolation Algorithm

Define a new polynomial

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

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

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 $\boldsymbol{p}(z) = \boldsymbol{a_1} \cdot z + \boldsymbol{a_0}$ $\quad$ $\boldsymbol{q}(z) = \boldsymbol{b_1} \cdot z + \boldsymbol{b_0}$ compute

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

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

Have $\mathbf{3}$ coefficients of $\boldsymbol{r}$ to compute.

Idea: Evaluate each of $\boldsymbol{p}$ and $\boldsymbol{q}$ at $\mathbf{3}$ points, $\mathbf{0}$, $\mathbf{1}$, $-\mathbf{1}$, and multiply results

- $\boldsymbol{r}(\mathbf{0}) = \boldsymbol{p}(\mathbf{0}) \cdot \boldsymbol{q}(\mathbf{0}) = \boldsymbol{a_0 b_0}$
- $\boldsymbol{r}(\mathbf{1}) = \boldsymbol{p}(\mathbf{1}) \cdot \boldsymbol{q}(\mathbf{1}) = (\boldsymbol{a_0} + \boldsymbol{a_1})\,(\boldsymbol{b_0} + \boldsymbol{b_1})$
- $\boldsymbol{r}(-\mathbf{1}) = \boldsymbol{p}(-\mathbf{1}) \cdot \boldsymbol{q}(-\mathbf{1}) = (\boldsymbol{a_0} - \boldsymbol{a_1})\,(\boldsymbol{b_0} - \boldsymbol{b_1})$

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