

---

# **CS483 Analysis of Algorithms**

## **Lecture 03 – Divide-n-Conquer \***

Jyh-Ming Lien

February 06, 2008

---

\*this lecture note is based on *Algorithms* by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani and *Introduction to the Design and Analysis of Algorithms* by Anany Levitin.

# Today, we will learn...

▷ Today, we will learn...

Introduction

Sort & Select

Multiplication

Conclusion

- In this lecture we will two main topics:
  - Sort and selection
    - ▷ Mergesort and quicksort
    - ▷ Binary search
    - ▷ Closest-pair and convex-hull algorithms
  - Multiplication
    - ▷ Multiplication of large integers
    - ▷ Matrix multiplication
    - ▷ Polynomial multiplication
- We will approach these problems using the divide-and-conquer technique

Today, we will learn...

▷ Introduction

Divide and Conquer

Divide and Conquer

Divide and Conquer

Examples

Master Theorem

Sort & Select

Multiplication

Conclusion

# Introduction

# Divide and Conquer

Today, we will learn...

Introduction

▷ Divide and Conquer

Divide and Conquer

Divide and Conquer

Examples

Master Theorem

Sort & Select

Multiplication

Conclusion

- Divide and conquer was a successful military strategy long before it became an algorithm design strategy
  - **Coalition uses divide-conquer plan in Fallujah**  
*By Rowan Scarborough and Bill Gertz, THE WASHINGTON TIMES*  
Coalition troops are employing a divide-and-conquer strategy in Fallujah, Iraq, capitalizing on months of pinpointed intelligence to seal off terrorist-held neighborhoods and then attack enemy pockets.
- Example: Your CS 483 instructor give you a 50-question assignment today and ask you to turn it in the tomorrow. What should you do?

# Divide and Conquer

Today, we will learn...

Introduction

Divide and Conquer

▷ Divide and Conquer

Divide and Conquer

Examples

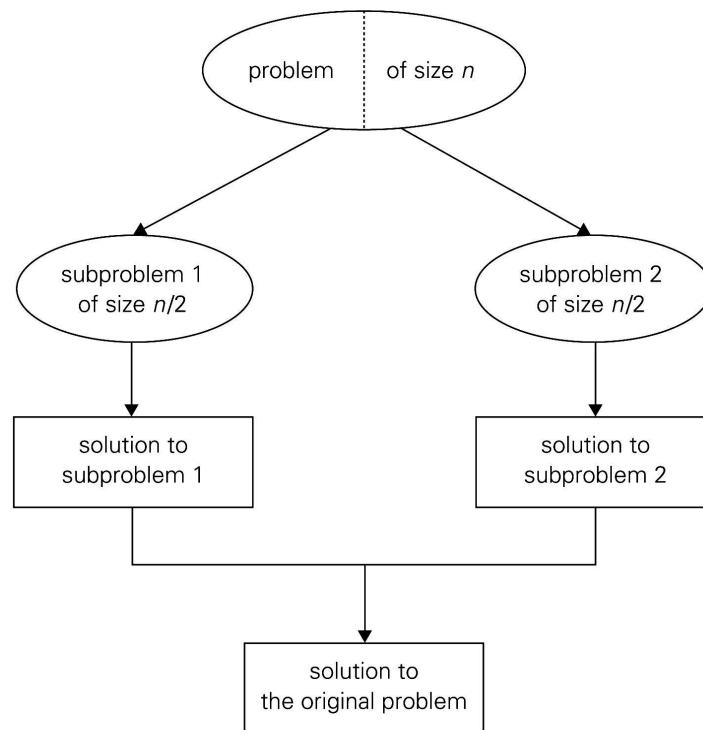
Master Theorem

Sort & Select

Multiplication

Conclusion

- The most-well known algorithm design strategy:
  1. Divide instance of problem into two or more smaller instances
  2. Solve smaller instances recursively
  3. Obtain solution to original (larger) instance by combining these solutions



# Divide and Conquer Examples

Today, we will learn...

Introduction

Divide and Conquer

Divide and Conquer

    Divide and Conquer

▷ Examples

Master Theorem

Sort & Select

Multiplication

Conclusion

- Example: Given a list  $A = \{2, 3, 6, 4, 12, 1, 7\}$ , compute  $\sum_{i=1}^7 A_i$

# Master Theorem

Today, we will learn...

Introduction

Divide and Conquer

Divide and Conquer

Divide and Conquer

Examples

▷ Master Theorem

Sort & Select

Multiplication

Conclusion

- If we have a problem of size  $n$  and our algorithm divides the problems into  $b$  instances, with  $a$  of them needing to be solved. Then we can set up our running time  $T(n)$  as:  $T(n) = aT(n/b) + f(n)$ , where  $f(n)$  is the time spent on dividing and merging.
- **Master Theorem:** If  $f(n) \in \Theta(n^d)$ , with  $d \geq 0$ , then

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

- Examples:
  1.  $T(n) = 4T(n/2) + n \Rightarrow T(n) =$
  2.  $T(n) = 4T(n/2) + n^2 \Rightarrow T(n) =$
  3.  $T(n) = 4T(n/2) + n^3 \Rightarrow T(n) =$

Today, we will learn...

Introduction

▷ Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

# Sort & Select

# Sorting: Mergesort

Today, we will learn...

Introduction

Sort & Select

▷ Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Given an array of  $n$  numbers, sort the element from small to large.

**Algorithm 0.1:** MERGESORT( $A[1 \cdots n]$ )

**if**  $n = 1$

**then return**

$B \leftarrow A[1 \cdots \lfloor \frac{n}{2} \rfloor]$

$C \leftarrow A[\lceil \frac{n}{2} \rceil \cdots n]$

*MergeSort(B)*

*MergeSort(C)*

*Merge(B, C, A)*

# Sorting: Mergesort

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

▷ Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Merge two sorted arrays,  $B$  and  $C$  and put the result in  $A$

**Algorithm 0.2:** MERGE( $B[1 \cdots p], C[1 \cdots q], A[1 \cdots p + q]$ )

```
i ← 0; j ← 0; k ← 0
for k ∈ {1, 2, ..., p + q}
    do { if B[i] < C[j]
        then A[k] = B[i]; i ← i + 1
        else A[k] = C[j]; j ← j + 1
```

# Sorting: Mergesort Example

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

    Sorting: Mergesort

▷ Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Example: 24, 11, 91, 10, 22, 32, 22, 3, 7, 99

- Is Mergesort stable?

# Analysis of Merge Sort

Today, we will learn...

$C_{worst}(n)$

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

▷ Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

# Sorting: Quicksort

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

▷ Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Given an array of  $n$  numbers, sort the element from small to large.

**Algorithm 0.3:**  $\text{QUICKSORT}(A[1 \cdots n])$

**if**  $n = 1$

**then return**

Create two arrays  $B, C$

**for**  $i \in \{2, 3, \dots, n\}$

**do** {  
        **if**  $A[i] < A[1]$   
            **then**  $B \leftarrow A[i]$   
            **else**  $C \leftarrow A[i]$

$\text{Quicksort}(B)$

$\text{Quicksort}(C)$

$A \leftarrow (B, A[1], C)$

- $A[1]$  in the above algorithm is called **pivot**



# Sorting: Quicksort Example

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

    Sorting: Quicksort

▷ Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Example: 24, 11, 91, 10, 22, 32, 22, 3, 7, 22

- Is Quicksort stable?

# Analysis of Quicksort

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

▷ Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

□  $C_{worst}(n)$

□  $C_{best}(n)$

□  $C_{avg}(n)$

# Why is Quicksort quicker?

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

  Why is Quicksort

▷ quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Because quicksort allows very fast “in-place partition”

## Algorithm 0.4: PARTITION( $A[a \cdots b]$ )

$p \leftarrow A[a]$

$i \leftarrow a + 1; j \leftarrow b$

**repeat**

{   **while**  $A[i] < p$   
    **do**  $i \leftarrow i + 1$   
    **while**  $A[j] > p$   
    **do**  $j \leftarrow j - 1$   
    **if**  $i < j$   
      **then** swap( $A[i], A[j]$ )

**until**  $i \geq j$

swap( $A[a], A[j]$ )

# The Selection Problem

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

▷ The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Similar to partition in quicksort!
- Find the  $k$ -th smallest element in an array  $A$  with  $n$  unique elements

## Algorithm 0.5: $\text{SELECT}(A[1 \cdots n], k)$

```
 $p \leftarrow A[1]$ 
 $A_1 \leftarrow$  all elements smaller than  $p$ 
 $A_2 \leftarrow$  all elements larger than  $p$ 
if  $|A_1| = k - 1$ 
    then return ( $p$ )
else if  $|A_1| < k - 1$ 
    then return ( $\text{SELECT}(A_2, k - |A_1|)$ )
else return ( $\text{SELECT}(A_1, k)$ )
```

- The algorithm above will work well for  $A$  with unique elements. How do you change to make it work for more general cases? create another array  $A_3$  to store elements equals to  $p$  and change the comparison accordingly.
- Time complexity: On average  $O(n \log n)$  and in worst case  $O(n^2)$

# Binary Search

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

▷ Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Imagine that you are placed in an unknown building and you are given a room number, you need to find your CS 483 instructor. What will you do?
- **Binary Search:**
  - Very efficient algorithm for searching in **sorted array**  
**Example:** find 70 in {3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 98}
  - Efficient search in even in high dimensional unknown space  
**Example:**

# Binary Search

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

▷ Binary Search

Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Given a sorted array  $A$  of  $n$  numbers, find a key  $K$  in  $A$

**Algorithm 0.6:**  $\text{BINARYSEARCH}(A[1 \cdots n], K)$

$a \leftarrow 1; b \leftarrow n$

**while**  $a < b$

**do**  $\begin{cases} m \leftarrow \lfloor \frac{a+b}{2} \rfloor \\ \text{if } K = A[m] \text{return } (m) \\ \text{else if } K < A[m] r \leftarrow m - 1 \\ \text{else } l \leftarrow m + 1 \end{cases}$

**return**  $(-1)$

- Binary search is in fact a bad (degenerate) example of divide-and-conquer

# Analysis of Binary Search

---

- $C_{worst}(n)$
- $C_{best}(n)$
- $C_{avg}(n)$

# Closest Pair

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

▷ Closest Pair

Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Find the closest distance between points in a given point set

**Algorithm 0.7:**  $\text{CP}(P[1 \dots n])$

**comment:**  $P$  is a set  $n$  points

**if**  $n = 2$

**then** { Sort  $P[1], P[2]$  vertically  
        **return** ( $|P[1] - P[2]|$ )

**else** { Compute a center vertical line  $c$

$P_1 \leftarrow$  point on the left of  $c$   
         $P_2 \leftarrow$  point on the right of  $c$   
         $d \leftarrow \min(\text{CP}(P_1), \text{CP}(P_2))$   
        **return** (Combine( $c, P, P_1, P_2, d$ ))

# Closest Pair

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

▷ Closest Pair

Convex Hull

Quickhull

Multiplication

Conclusion

- Find the closest distance between points in a given point set

**Algorithm 0.8:** COMBINE( $c, P, P_1, P_2, d$ )

$P = \text{Merge}(P_1, P_2)$

$P'_1 \leftarrow \text{points } \in P_1 \text{ to } c \text{ within } d$

$P'_2 \leftarrow \text{points } \in P_2 \text{ to } c \text{ within } d$

$d_{min} = d$

**for each**  $p \in P'_1$

$Q \leftarrow \text{points of } P'_2 \text{ within } d \text{ from } p$

**do** { **for each**  $q \in Q$

**do** { **if**  $|p - q| < d_{min}$

**then**  $d_{min} = |p - q|$

**return** ( $d_{min}$ )

- What is the time complexity?

$$T(n) = 2T(n/2) + O(6n) = O(n \log n)$$

# Convex Hull

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

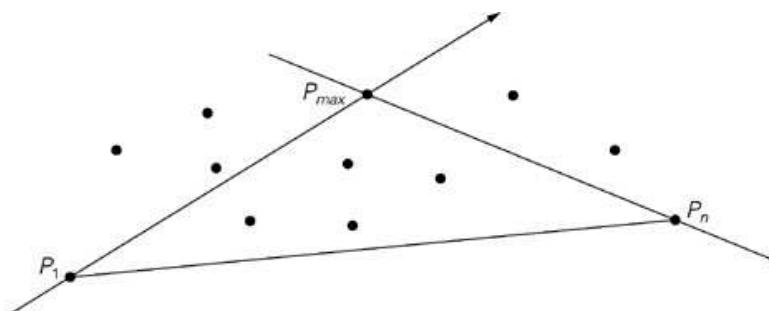
▷ Convex Hull

Quickhull

Multiplication

Conclusion

- Here we consider a divide-and-conquer algorithm called **quickhull**
- Quickhull is similar to quicksort  
why?
- Observations (given a point set  $P$  in 2-d):
  - The leftmost and rightmost points in  $P$  must be part of the convex hull
  - The furthest point away from any line must be part of the convex hull
  - Points in the triangle formed by any three points in  $P$  will **not** be part of the convex hull



# Quickhull

Today, we will learn...

Introduction

Sort & Select

Sorting: Mergesort

Sorting: Mergesort

Sorting: Mergesort

Example

Analysis of Merge Sort

Sorting: Quicksort

Sorting: Quicksort

Example

Analysis of Quicksort

Why is Quicksort quicker?

The Selection Problem

Binary Search

Binary Search

Closest Pair

Closest Pair

Convex Hull

▷ Quickhull

Multiplication

Conclusion

## □ Qhull

**Algorithm 0.9:**  $\text{QHULL}(P[1 \dots n])$

**comment:**  $P$  is a set  $n$  points

$p_1 \leftarrow$  leftmost point;  $p_2 \leftarrow$  rightmost point

$\text{QHULL}(P, p_1, p_2)$

$\text{QHULL}(P, p_2, p_1)$

**Algorithm 0.10:**  $\text{QHULL}(P[1 \dots n], p_1, p_2)$

**if**  $n = 0$

**then return**  $(\overline{p_1 p_2})$

$p_m \leftarrow$  farthest point to the left of  $\overline{p_1 p_2}$

Remove points  $\in \{P \cap \Delta(p_1 p_2 p_m)\}$  from  $P$

Animation: [http://www.cs.princeton.edu/~ah/alg\\_anim/version1/QuickHull.html](http://www.cs.princeton.edu/~ah/alg_anim/version1/QuickHull.html)

$\text{QHULL}(P, p_m, p_2)$

# Analysis of Quickhull

---

- Worst case: when the points are all on the convex hull boundary.

$$C_{worst}(n) = C_{worst}(n - 1) + n = \Theta(n^2)$$

- Best case: when the convex hull is a triangle!

$$C_{best}(n) = \Theta(n)$$

- Avg case: ???  $C_{avg}(n)$

Today, we will learn...

[Introduction](#)

[Sort & Select](#)

 [Multiplication](#)

Intenger multiplication

Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

[Conclusion](#)

# Multiplication

# Intenger multiplication

Today, we will learn...

[Introduction](#)

[Sort & Select](#)

[Multiplication](#)

▷ Intenger multiplication

Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

[Conclusion](#)

- What is the time complexity of multiplying two integers using the algorithms we learned in elementary schools?

Example: how do you compute this:  $12345 \times 67890$ ?

- Is there a better way of multiplying two integers than this elementary-school method?

Carl Friedrich Gauss (1777-1855) discovered that

$$AB = (a10^{\frac{n}{2}} + b)(c10^{\frac{n}{2}} + d) = K_2 10^n + K_1 10^{\frac{n}{2}} + K_0, \text{ where } K_2 = ac, \\ K_0 = bd, K_1 = (a + b)(c + d) - (K_0 + K_2).$$

Example: how do you compute this:  $12345 \times 67890$ ?



Carl Friedrich Gauss

# Intenger multiplication

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intenger multiplication

▷ Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

## □ Divid-and-conquer interger multiplication

**Algorithm 0.11:**  $M(A[1 \dots n], B[1 \dots n])$

```
if  $n = 1$ 
    then return ( $A[1]B[1]$ )
else
     $a \leftarrow A[1 \dots \frac{n}{2}], b \leftarrow A[\frac{n}{2} + 1 \dots n]$ 
     $c \leftarrow B[1 \dots \frac{n}{2}], d \leftarrow B[\frac{n}{2} + 1 \dots n]$ 
     $K_2 \leftarrow M(a, c)$ 
     $K_0 \leftarrow M(b, d)$ 
     $K_1 \leftarrow M(a + b, c + d) - (K_0 + K_2)$ 
    return ( $K_2 10^n + K_1 10^{\frac{n}{2}} + K_0$ )
```

## □ What is the time complexity? First we formulate the time complexity as: $T(n) = 3T(\frac{n}{2}) + O(n)$ Using Master Theorem, we have $a = 3$ , $b = 2$ and $d = 1$ . So, $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.6})$

# Matrix multiplication

Today, we will learn...

[Introduction](#)

[Sort & Select](#)

[Multiplication](#)

Integer multiplication

Integer multiplication

▷ Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

[Conclusion](#)

## Strassen's Matrix Multiplication:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & A_{12} \\ B_{21} & B_{22} \end{bmatrix}$$
$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

- $m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $m_2 = (A_{21} + A_{22})B_{11}$
- $m_3 = A_{11}(B_{12} - B_{22})$
- $m_4 = A_{22}(B_{21} - B_{11})$
- $m_5 = (A_{11} + A_{12})B_{22}$
- $m_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
- $m_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

# Matrix multiplication

Today, we will learn...

[Introduction](#)

[Sort & Select](#)

[Multiplication](#)

Integer multiplication

Integer multiplication

Matrix multiplication

▷ Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

[Conclusion](#)

- What is the time complexity?

$T(n) = 7T\left(\frac{n}{2}\right) + O(n)$  Using Master Theorem, we have  $a = 7$ ,  $b = 4$  and  $d = 1$ . So,  $T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$

- Do you still remember what the time complexity of the brute-force algorithm is?

# Polynomial multiplication

Today, we will learn...

Introduction

Sort & Select

Multiplication

Integer multiplication

Integer multiplication

Matrix multiplication

Matrix multiplication

Polynomial

▷ multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

- two degree- $n$  polynomials:

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$B(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_1 x + b_0$$

- Multiplication of two degree- $n$  polynomial

$$C(x) = A(x)B(x) = c_{2n} x^{2n} + c_{2n-1} x^{2n-1} + \cdots + c_1 x + c_0$$

- The coefficient  $c_k$  is:

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_{k-1} b_1 + a_k b_0$$

- A brute force method for computing  $C(x)$  will have time complexity=

- Can we do better?

# Representing polynomial

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intger multiplication

Intger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

    Representing  
    ▷ polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

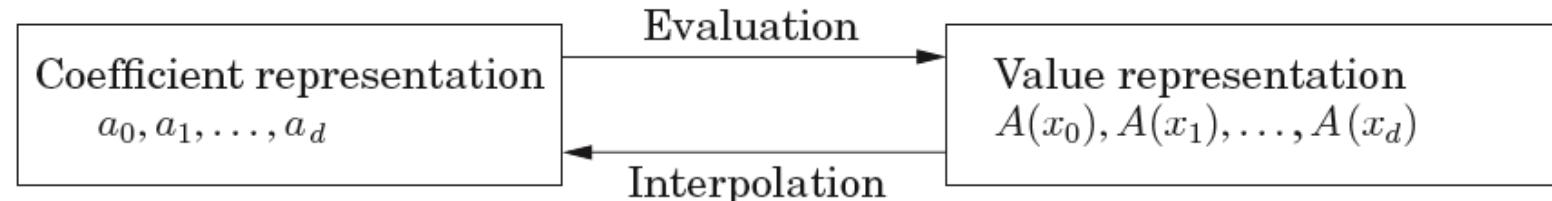
A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

- **Fact:** A degree- $n$  polynomial is uniquely defined by any  $n + 1$  distinct points
- A degree- $n$  polynomial  $A(x)$  can be represented by:
  - $a_0, a_1, \dots, a_{n-1}, a_n$ , or
  - $A(x_0), A(x_1), A(x_2), \dots, A(x_{n-1}), A(x_n)$ , where  $x_i$  can be any distinct values.
- We can convert between these two representations:



- The value representation allows us to develop faster algorithm!
  - We only need  $2n + 1$  points for  $C(x)$
  - It's easy and efficient to generate these  $2n + 1$  points from  $A(x)$  and  $B(x)$

# Polynomial multiplication

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intger multiplication

Intger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

    Polynomial

    ▷ multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

## □ General idea:

1. Convert  $A$  and  $B$  to value representation (Evaluation)
2. Perform multiplication to obtain  $C$  in value representation
3. Convert  $C$  back to coefficient representation (Interpolation)

### Coefficient representation

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

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

**Multiplication  $O(n^2)$**

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

↓ Evaluation  $O(n \log n)$

$$A(x_0), A(x_1), \dots, A(x_{2n})$$

$$B(x_0), B(x_1), \dots, B(x_{2n})$$

↑ Interpolation  $O(n \log n)$

**Multiplication  $O(n)$**

$$C(x_i) = A(x_i)B(x_i)$$

### Value representation

# Polynomial evaluation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intger multiplication

Intger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

▷ Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

- $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$
- Polynomial evaluation: Given  $x$ , compute  $f(x)$
- Brute force algorithm

## Algorithm 0.12: F( $x$ )

```
fx ← 0
for  $i \in \{0 \dots n\}$ 
  do  $\left\{ \begin{array}{l} y \leftarrow 1 \\
    \b{for} j \in \{0 \dots i - 1\} \\
      \b{do} y \leftarrow y \cdot x \\
    fx \leftarrow fx + a_i \cdot y \end{array} \right.$ 
```

- Time complexity of this brute force algorithm?  $\sum_{i=0}^n i = \Theta(n^2)$
- Can we do better?

# Horner's Rule

Today, we will learn...

Introduction

Sort & Select

Multiplication

Integer multiplication

Integer multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

▷ Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

## □ Horner's rule

$$\begin{aligned}f(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \\&= (a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_1) x + a_0 \\&= (\cdots (a_n x + a_{n-1}) x + \cdots) x + a_0\end{aligned}$$

## □ Polynomial evaluation using Horner's rule

**Algorithm 0.13:**  $F(x)$

```
fx ← an
for i ∈ {n – 1 … 0}
  do fx ← x · fx + ai
```

## □ Time complexity: $\Theta(n)$

## □ Example: $f(x) = 2x^4 - x^3 + 3x^2 + x - 5$ at $x = 4$

# A $n \log n$ time polynomial evaluation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Integer multiplication

Integer multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

▷ polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

- Basic idea: How we select  $x_i$  affects the run time.
- Example: If we pick  $\pm x_0, \pm x_1, \dots, \pm x_{n/2-1}$ , then  $A(x_i)$  and  $A(-x_i)$  have many overlap
  - $x^5 + 2x^4 + 3x^3 + 4x^2 + 5x + 6 = (2x^4 + 4x^2 + 6) + x(x^4 + 3x^2 + 5)$
  - $A(x) = A_e(x^2) + xA_o(x^2)$
  - When evaluate  $x_i$ ,  $A(x_i) = A_e(x_i^2) + x_i A_o(x_i^2)$
  - When evaluate  $-x_i$ ,  $A(-x_i) = A_e(x_i^2) - x_i A_o(x_i^2)$
- What we need is  $x_i$  such that

# $n$ -th roots of unity

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intger multiplication

Intger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

▷  $n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

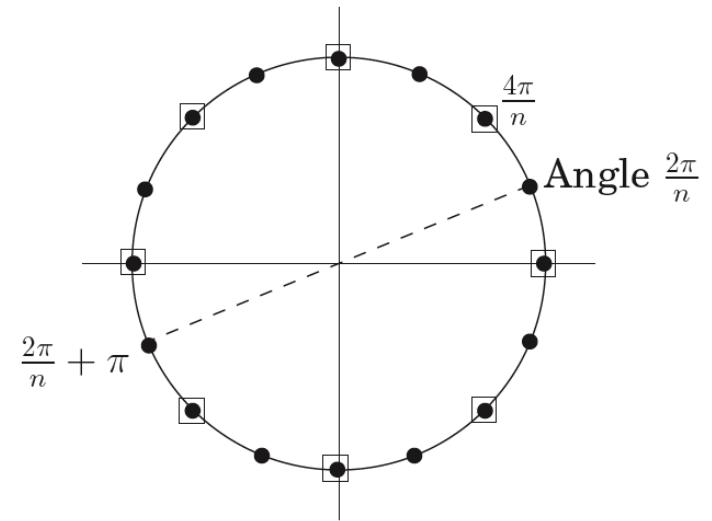
Conclusion

- **Idea:** Use  $n$ -th roots of unity:  $z^n = 1$  as our  $x_i$

- **Background:**

- Complex number  $z = r(\cos(\theta) + i \sin(\theta))$ 
  - ▷ Usually denoted as  $re^{i\theta}$  or  $(r, \theta)$
  - ▷  $(r_1, \theta_1) \times (r_2, \theta_2) = (r_1 r_2, \theta_1 + \theta_2)$
- Let  $\omega_n = \cos\left(\frac{2\pi}{n}\right) + i \sin\left(\frac{2\pi}{n}\right) = e^{2\pi i/n}$  be a complex  $n$ -th root of unity
- Other roots include:  $\omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n^n$
- Properties:

- ▷  $\omega_n^j = -\omega_n^{j+n/2}$
- ▷ Therefore,  $(\omega_n^j)^2 = (-\omega_n^{j+n/2})^2$
- ▷ Moreover,  $(\omega_n^j)^2 = \omega_{n/2}^j$
- ▷  $\sum_{i=1}^n \omega_n^i = \frac{1-\omega_n^n}{1-\omega_n} = 0$



# $n$ -th roots of unity

## □ Examples $n = 8$ :

Today, we will learn...

Introduction

Sort & Select

Multiplication

Integer multiplication

Integer multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

▷  $n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

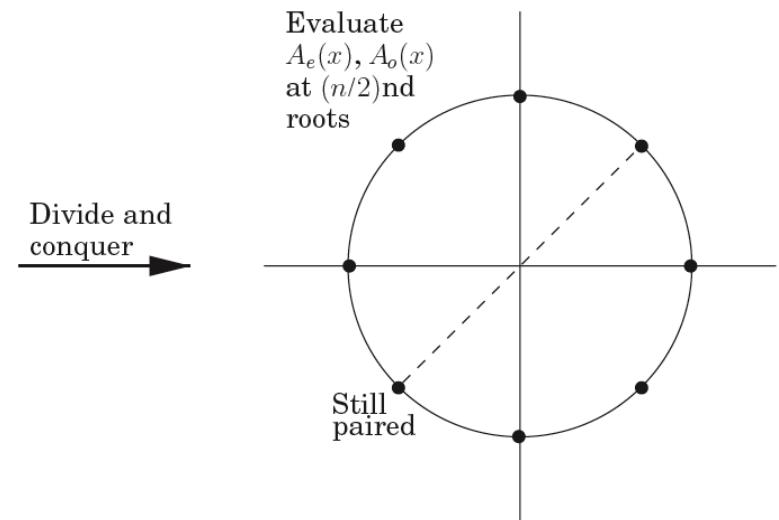
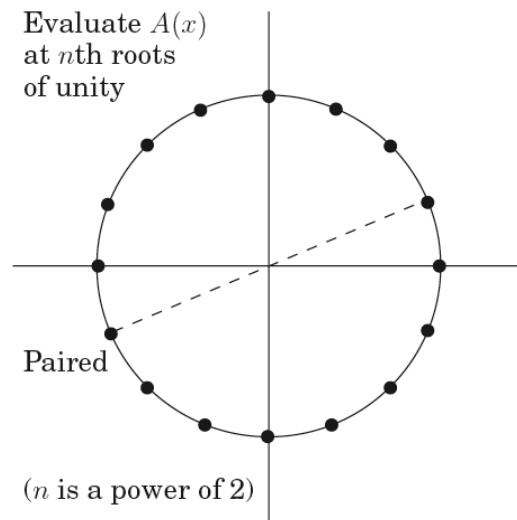
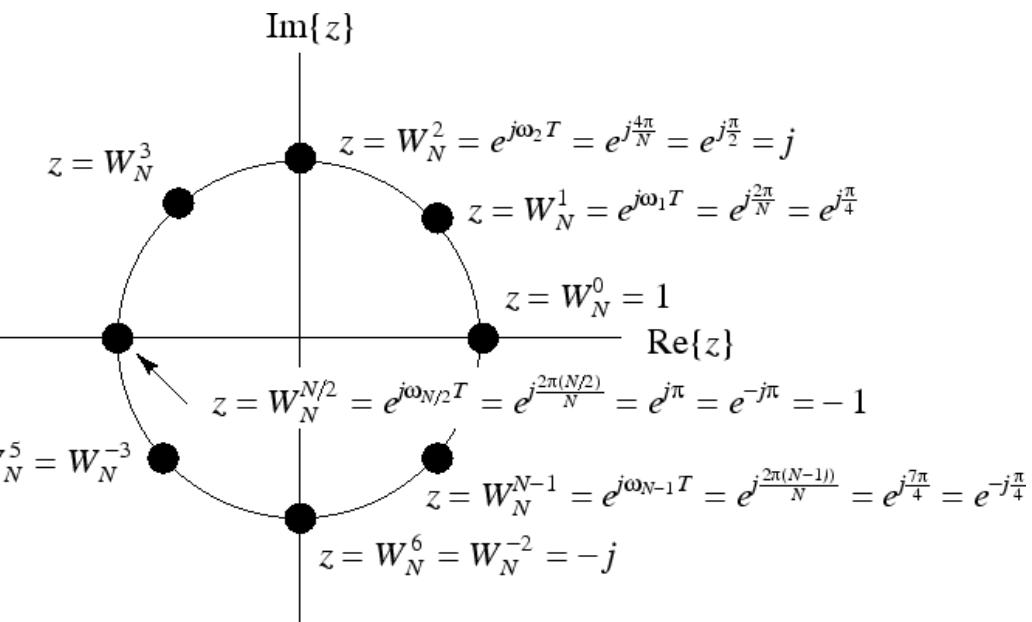
polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion



# A $n \log n$ time polynomial evaluation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intger multiplication

Intger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

▷ polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

## □ FFT

**Algorithm 0.14:**  $\text{FFT}((a_0, a_1, a_2, \dots, a_{n-1}), \omega)$

**if**  $\omega = 1$

**then return** ( $A(1)$ )

$(s_0, s_1, \dots, s_{n/2-1}) = \text{FFT}((a_0, a_2, \dots, a_{n-2}), \omega^2)$

$(s'_0, s'_1, \dots, s'_{n/2-1}) = \text{FFT}((a_1, a_3, \dots, a_{n-1}), \omega^2)$

**for**  $j = 0$  to  $n/2 - 1$

**do**  $\begin{cases} r_j \leftarrow s_j + \omega^j s'_j \\ r_{j+n/2} \leftarrow s_j - \omega^j s'_j \end{cases}$

**return** ( $r_0, \dots, r_{n-1}$ )

## □ Time complexity?

# A $n \log n$ time polynomial evaluation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Integer multiplication

Integer multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

▷ polynomial evaluation

A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

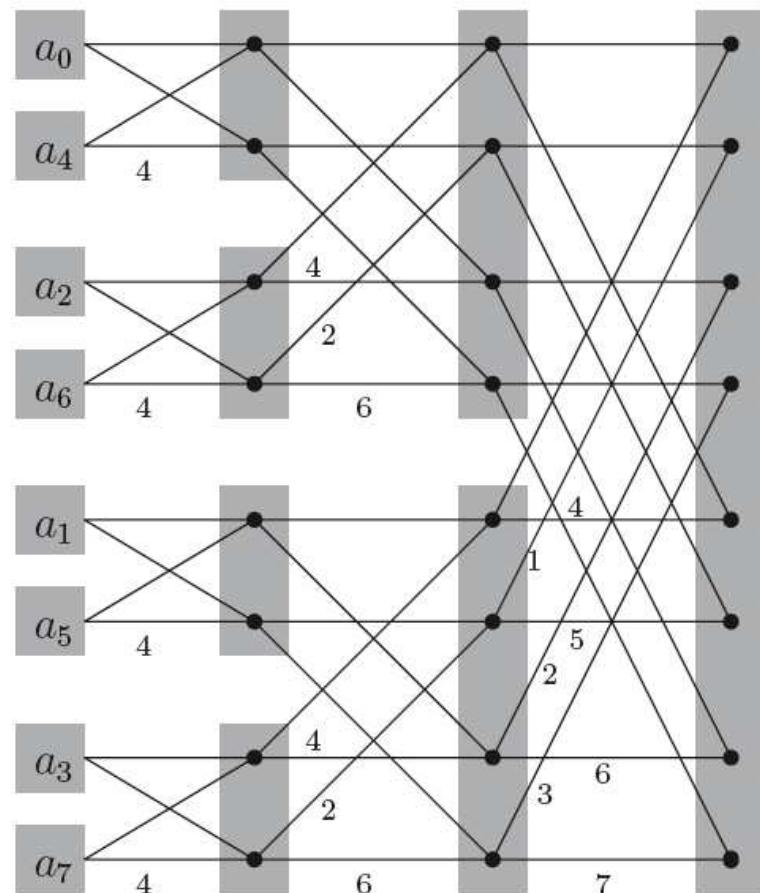
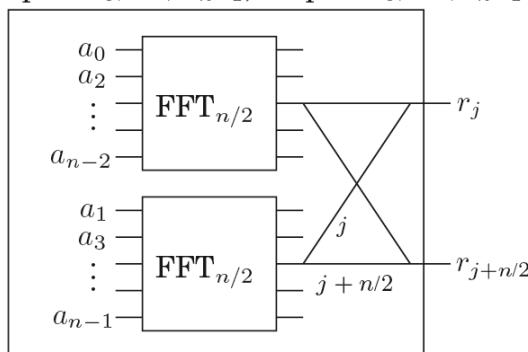
polynomial interpolation

A closer look

Conclusion

## □ Hardware implementation

FFT <sub>$n$</sub>  (input:  $a_0, \dots, a_{n-1}$ , output:  $r_0, \dots, r_{n-1}$ )



# A $n \log n$ time polynomial interpolation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intenger multiplication

Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial

▷ interpolation

A  $n \log n$  time

polynomial interpolation

A closer look

Conclusion

- Convert the values  $C(x_i)$  back to coefficients:  $\{c_i\} = \text{FFT}(C(x_i), \omega^{-1})$
- Here is why

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

- $M_n(\omega) =$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega^j & \omega^{2j} & \cdots & \omega^{(n-1)j} \\ & & \vdots & & \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix} \leftarrow \begin{array}{l} \text{row for } \omega^0 = 1 \\ \omega \\ \omega^2 \\ \vdots \\ \omega^j \\ \vdots \\ \omega^{n-1} \end{array}$$

- Entry  $(j, k)$  of  $M_n$  is  $\omega^{jk}$

# A $n \log n$ time polynomial interpolation

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intenger multiplication

Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial interpolation

    A  $n \log n$  time

    polynomial

▷ interpolation

A closer look

Conclusion

- $M_n(\omega)$  is invertible, i.e., column  $j$  and column  $k$  are orthogonal

– *proof:*

- Inversion formula  $M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$

– *proof:*

# A closer look

Today, we will learn...

Introduction

Sort & Select

Multiplication

Intenger multiplication

Intenger multiplication

Matrix multiplication

Matrix multiplication

Polynomial multiplication

Representing polynomial

Polynomial multiplication

Polynomial evaluation

Horner's Rule

A  $n \log n$  time

polynomial evaluation

$n$ -th roots of unity

$n$ -th roots of unity

A  $n \log n$  time

polynomial evaluation

A  $n \log n$  time

polynomial evaluation

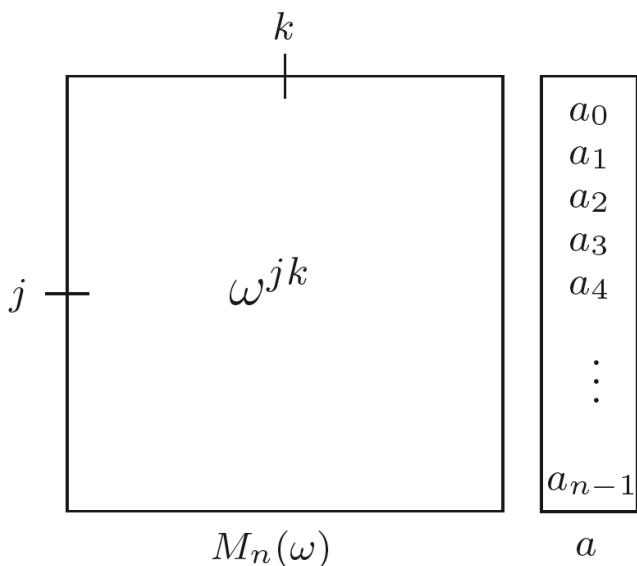
A  $n \log n$  time

polynomial interpolation

A  $n \log n$  time

polynomial interpolation

▷ A closer look



Conclusion

Today, we will learn...

Introduction

Sort & Select

Multiplication

▷ Conclusion

Summary

# Conclusion

# Summary

Today, we will learn...

Introduction

Sort & Select

Multiplication

Conclusion

▷ Summary

## □ Summary

- Sort and select
    - ▷ Mergesort and quicksort<sup>1</sup>
    - ▷ Binary search
    - ▷ Closest-pair and convex-hull algorithms
  - Multiplication
    - ▷ Multiplication of large integers - from Gauss
    - ▷ Matrix multiplication
    - ▷ Polynomial multiplication - FFT<sup>1</sup> (Also from Gauss)
- ## □ Divide-n-conquer strategy
- Advantages of
    - ▷ Make problems easier
    - ▷ Easy parallelization
  - Disadvantages of Divide-n-conquer strategy
    - ▷ Recursion can be slow
    - ▷ Subproblems may overlap