

# CMPSCI 611: Advanced Algorithms

## Lecture 2: More Divide and Conquer

Andrew McGregor

Last Compiled: September 17, 2009

# Outline

Recap from Last Lecture

Closest Pair of Points in a Plane

Fast Fourier Transform and Polynomial Multiplication

# Divide and Conquer Methodology

- ▶ Goal: Solve problem  $P$  on an instance  $I$  of “size”  $n$ .

# Divide and Conquer Methodology

- ▶ Goal: Solve problem  $P$  on an instance  $I$  of “size”  $n$ .
- ▶ Divide & Conquer Method:
  - ▶ Transform  $I$  into smaller instances  $I_1, \dots, I_a$  each of “size”  $n/b$
  - ▶ Solve problem  $P$  on each of  $I_1, \dots, I_a$  by recursion
  - ▶ Combine the solutions to get a solution of  $I$

# Divide and Conquer Methodology

- ▶ Goal: Solve problem  $P$  on an instance  $I$  of “size”  $n$ .
- ▶ Divide & Conquer Method:
  - ▶ Transform  $I$  into smaller instances  $I_1, \dots, I_a$  each of “size”  $n/b$
  - ▶ Solve problem  $P$  on each of  $I_1, \dots, I_a$  by recursion
  - ▶ Combine the solutions to get a solution of  $I$
- ▶ **Example 1 (Merge Sort):** To sort list of  $n$  numbers, divide into 2 lists of size  $n/2$ , sort each list, and merge.

# Divide and Conquer Methodology

- ▶ Goal: Solve problem  $P$  on an instance  $I$  of “size”  $n$ .
- ▶ Divide & Conquer Method:
  - ▶ Transform  $I$  into smaller instances  $I_1, \dots, I_a$  each of “size”  $n/b$
  - ▶ Solve problem  $P$  on each of  $I_1, \dots, I_a$  by recursion
  - ▶ Combine the solutions to get a solution of  $I$
- ▶ **Example 1 (Merge Sort)**: To sort list of  $n$  numbers, divide into 2 lists of size  $n/2$ , sort each list, and merge.
- ▶ **Example 2 (Strassen's Algorithm)**: To multiply two  $n \times n$  matrices, transform instance into 7 instances of multiplying  $n/2 \times n/2$  matrices together, combine the solutions together.

## Analyzing Divide and Conquer Algorithms

Let  $T(n)$  be running time of algorithm on instance of size  $n$ . Then

$$T(1) = \Theta(1), T(n) = aT(n/b) + \Theta(n^\alpha)$$

where  $\Theta(n^\alpha)$  is time to make new instances and combine solutions.

## Analyzing Divide and Conquer Algorithms

Let  $T(n)$  be running time of algorithm on instance of size  $n$ . Then

$$T(1) = \Theta(1), T(n) = aT(n/b) + \Theta(n^\alpha)$$

where  $\Theta(n^\alpha)$  is time to make new instances and combine solutions.

### Theorem (Master Theorem)

If  $a, b, \alpha$  are constants, for  $\beta = \log_b a$ ,

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$

## Analyzing Divide and Conquer Algorithms

Let  $T(n)$  be running time of algorithm on instance of size  $n$ . Then

$$T(1) = \Theta(1), T(n) = aT(n/b) + \Theta(n^\alpha)$$

where  $\Theta(n^\alpha)$  is time to make new instances and combine solutions.

### Theorem (Master Theorem)

If  $a, b, \alpha$  are constants, for  $\beta = \log_b a$ ,

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$

- ▶ **Example 1 (Merge Sort):**  $a = b = 2$  and  $\alpha = 1$ . Therefore  $\beta = \log_2 2 = 1$  and running time is  $O(n \log n)$ .

## Analyzing Divide and Conquer Algorithms

Let  $T(n)$  be running time of algorithm on instance of size  $n$ . Then

$$T(1) = \Theta(1), T(n) = aT(n/b) + \Theta(n^\alpha)$$

where  $\Theta(n^\alpha)$  is time to make new instances and combine solutions.

### Theorem (Master Theorem)

If  $a, b, \alpha$  are constants, for  $\beta = \log_b a$ ,

$$T(n) = \begin{cases} \Theta(n^\alpha) & \text{if } \alpha > \beta \\ \Theta(n^\beta) & \text{if } \alpha < \beta \\ \Theta(n^\alpha \log n) & \text{if } \alpha = \beta \end{cases}$$

- ▶ **Example 1 (Merge Sort):**  $a = b = 2$  and  $\alpha = 1$ . Therefore  $\beta = \log_2 2 = 1$  and running time is  $O(n \log n)$ .
- ▶ **Example 2 (Matrix Multiplication):**  $a = 7, b = 2$  and  $\alpha = 2$ . Therefore  $\beta = \log_2 7 \approx 2.81$  and running time is  $\Theta(n^{2.81})$ .

# Outline

Recap from Last Lecture

Closest Pair of Points in a Plane

Fast Fourier Transform and Polynomial Multiplication

## Finding Minimum Distance between Points on a Plane

**Problem:** Given  $n$  distinct points  $p_1, \dots, p_n \in \mathbb{R}^2$ , find

minimum distance between any two points =  $\min_{i,j:i \neq j} d(p_i, p_j)$

where  $d$  is the “Euclidean” distance.

## Finding Minimum Distance between Points on a Plane

**Problem:** Given  $n$  distinct points  $p_1, \dots, p_n \in \mathbb{R}^2$ , find

minimum distance between any two points =  $\min_{i,j:i \neq j} d(p_i, p_j)$

where  $d$  is the “Euclidean” distance. I.e., if  $p = (x_1, y_1)$  and  $q = (x_2, y_2)$  then

$$d(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

## Finding Minimum Distance between Points on a Plane

**Problem:** Given  $n$  distinct points  $p_1, \dots, p_n \in \mathbb{R}^2$ , find

minimum distance between any two points =  $\min_{i,j:i \neq j} d(p_i, p_j)$

where  $d$  is the “Euclidean” distance. I.e., if  $p = (x_1, y_1)$  and  $q = (x_2, y_2)$  then

$$d(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

How long does naive algorithm take?

## Finding Minimum Distance between Points on a Plane

**Problem:** Given  $n$  distinct points  $p_1, \dots, p_n \in \mathbb{R}^2$ , find

minimum distance between any two points =  $\min_{i,j:i \neq j} d(p_i, p_j)$

where  $d$  is the “Euclidean” distance. I.e., if  $p = (x_1, y_1)$  and  $q = (x_2, y_2)$  then

$$d(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

How long does naive algorithm take?  $O(n^2)$

## Finding Minimum Distance between Points on a Plane

**Problem:** Given  $n$  distinct points  $p_1, \dots, p_n \in \mathbb{R}^2$ , find

minimum distance between any two points =  $\min_{i,j:i \neq j} d(p_i, p_j)$

where  $d$  is the “Euclidean” distance. I.e., if  $p = (x_1, y_1)$  and  $q = (x_2, y_2)$  then

$$d(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

How long does naive algorithm take?  $O(n^2)$

We'll do it in  $O(n \log n)$  steps.

# Minimum Distance Algorithm

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .
2. Recursively find minimum distance within  $P_L$  and  $P_R$ :

$$\delta_L = \min_{p,q \in P_L: p \neq q} d(p, q)$$

$$\delta_R = \min_{p,q \in P_R: p \neq q} d(p, q)$$

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .
2. Recursively find minimum distance within  $P_L$  and  $P_R$ :

$$\delta_L = \min_{p,q \in P_L: p \neq q} d(p, q)$$

$$\delta_R = \min_{p,q \in P_R: p \neq q} d(p, q)$$

3. Compute  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$  and return

$$\min(\delta_L, \delta_R, \delta_M)$$

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .
2. Recursively find minimum distance within  $P_L$  and  $P_R$ :

$$\delta_L = \min_{p,q \in P_L: p \neq q} d(p, q)$$

$$\delta_R = \min_{p,q \in P_R: p \neq q} d(p, q)$$

3. Compute  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$  and return

$$\min(\delta_L, \delta_R, \delta_M)$$

**Note:** If Step 3 takes  $\Theta(n^2)$  time, we get

$$T(n) = 2T(n/2) + \Theta(n^2), T(1) = \Theta(1)$$

and hence...

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .
2. Recursively find minimum distance within  $P_L$  and  $P_R$ :

$$\delta_L = \min_{p,q \in P_L: p \neq q} d(p, q)$$

$$\delta_R = \min_{p,q \in P_R: p \neq q} d(p, q)$$

3. Compute  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$  and return

$$\min(\delta_L, \delta_R, \delta_M)$$

**Note:** If Step 3 takes  $\Theta(n^2)$  time, we get

$$T(n) = 2T(n/2) + \Theta(n^2), T(1) = \Theta(1)$$

and hence...  $T(n) = \Theta(n^2)$ . If we can do Step 3 in  $\Theta(n)$  time, we get...

## Minimum Distance Algorithm

1. Divide points  $P$  with a vertical line that separates points into sets  $P_L$  and  $P_R$  where  $|P_L| = |P_R| = n/2$ .
2. Recursively find minimum distance within  $P_L$  and  $P_R$ :

$$\delta_L = \min_{p,q \in P_L: p \neq q} d(p, q)$$

$$\delta_R = \min_{p,q \in P_R: p \neq q} d(p, q)$$

3. Compute  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$  and return

$$\min(\delta_L, \delta_R, \delta_M)$$

**Note:** If Step 3 takes  $\Theta(n^2)$  time, we get

$$T(n) = 2T(n/2) + \Theta(n^2), T(1) = \Theta(1)$$

and hence...  $T(n) = \Theta(n^2)$ . If we can do Step 3 in  $\Theta(n)$  time, we get...  $T(n) = \Theta(n \log n)$ .

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$
- ▶ Don't need to check all pairs  $p \in P_L, q \in P_R$  once we know  $\delta$ :

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$
- ▶ Don't need to check all pairs  $p \in P_L, q \in P_R$  once we know  $\delta$ :
  - ▶ Let  $P_M$  be points with  $x$ -coords between  $m - \delta$  and  $m + \delta$

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$
- ▶ Don't need to check all pairs  $p \in P_L, q \in P_R$  once we know  $\delta$ :
  - ▶ Let  $P_M$  be points with  $x$ -coords between  $m - \delta$  and  $m + \delta$
  - ▶ For each  $p \in P_L \cap P_M$ , suffices to compare with  $q \in P_R \cap P_M$  where  $y$ -coord of  $q$  is at most  $\delta$  different from  $y$ -coord of  $p$ . There are at most  $\Theta(1)$  such points!

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$
- ▶ Don't need to check all pairs  $p \in P_L, q \in P_R$  once we know  $\delta$ :
  - ▶ Let  $P_M$  be points with  $x$ -coords between  $m - \delta$  and  $m + \delta$
  - ▶ For each  $p \in P_L \cap P_M$ , suffices to compare with  $q \in P_R \cap P_M$  where  $y$ -coord of  $q$  is at most  $\delta$  different from  $y$ -coord of  $p$ . There are at most  $\Theta(1)$  such points!
- ▶ Hence, number of comparisons to find  $\min(\delta, \delta_M)$  is  $O(n)$

## Making Step 3 Efficient

- ▶ Need to find  $\min(\delta_L, \delta_R, \delta_M)$  where  $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- ▶ Suppose that the dividing line is  $x = m$  and  $\delta = \min(\delta_L, \delta_R)$
- ▶ Don't need to check all pairs  $p \in P_L, q \in P_R$  once we know  $\delta$ :
  - ▶ Let  $P_M$  be points with  $x$ -coords between  $m - \delta$  and  $m + \delta$
  - ▶ For each  $p \in P_L \cap P_M$ , suffices to compare with  $q \in P_R \cap P_M$  where  $y$ -coord of  $q$  is at most  $\delta$  different from  $y$ -coord of  $p$ . There are at most  $\Theta(1)$  such points!
- ▶ Hence, number of comparisons to find  $\min(\delta, \delta_M)$  is  $O(n)$
- ▶ Need to also identify the relevant comparisons  $O(n)$  time
  - ▶ **Pre-process:** Make two copies of points sorted by each coord
  - ▶ Ensure both lists are passed to each recursion sorted
  - ▶ Given sorted lists, it's easy to find the relevant points

# Outline

Recap from Last Lecture

Closest Pair of Points in a Plane

Fast Fourier Transform and Polynomial Multiplication

# Polynomial Multiplication

**Problem:** Suppose  $A(x)$  and  $B(x)$  are polynomials of degree  $n - 1$ :

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

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

Compute  $C(x) = A(x)B(x)$ .

# Polynomial Multiplication

**Problem:** Suppose  $A(x)$  and  $B(x)$  are polynomials of degree  $n - 1$ :

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

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

Compute  $C(x) = A(x)B(x)$ .

How long does naive algorithm take?

# Polynomial Multiplication

**Problem:** Suppose  $A(x)$  and  $B(x)$  are polynomials of degree  $n - 1$ :

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

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

Compute  $C(x) = A(x)B(x)$ .

How long does naive algorithm take?  $O(n^2)$

# Representation of Polynomials

## Definition

The *coefficient representation* (CR) of a polynomial the vector of coefficients. E.g.,  $(1, 3, -2, 1)$  is the coefficient representation of

$$f(x) = 1 + 3x - 2x^2 + x^3$$

# Representation of Polynomials

## Definition

The *coefficient representation* (CR) of a polynomial the vector of coefficients. E.g.,  $(1, 3, -2, 1)$  is the coefficient representation of

$$f(x) = 1 + 3x - 2x^2 + x^3$$

## Definition

The *point-value representation* (PVR) of a polynomial: for  $n$  distinct points  $x_0, \dots, x_{n-1}$  the PVR of  $f$  is

$$\{(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_{n-1}, f(x_{n-1}))\}$$

E.g.,  $f(x) \equiv \{(0, 1), (1, 3), (2, 7), (3, 19)\}$ .

# Representation of Polynomials

## Definition

The *coefficient representation* (CR) of a polynomial the vector of coefficients. E.g.,  $(1, 3, -2, 1)$  is the coefficient representation of

$$f(x) = 1 + 3x - 2x^2 + x^3$$

## Definition

The *point-value representation* (PVR) of a polynomial: for  $n$  distinct points  $x_0, \dots, x_{n-1}$  the PVR of  $f$  is

$$\{(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_{n-1}, f(x_{n-1}))\}$$

E.g.,  $f(x) \equiv \{(0, 1), (1, 3), (2, 7), (3, 19)\}$ .

## Lemma

*Specifying the value of a function at  $n$  distinct points uniquely specifies a degree  $n - 1$  polynomial that goes through those points.*

# Polynomial Arithmetic in Point-Value Representation

- ▶ First attempt: Let  $x_0, \dots, x_{n-1}$  be distinct and suppose

$$A(x) \equiv \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

$$B(x) \equiv \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$$

# Polynomial Arithmetic in Point-Value Representation

- ▶ First attempt: Let  $x_0, \dots, x_{n-1}$  be distinct and suppose

$$A(x) \equiv \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

$$B(x) \equiv \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$$

- ▶ Then surely,

$$C(x) \equiv \{(x_0, y_0 z_0), (x_1, y_1 z_1), \dots, (x_{n-1}, y_{n-1} z_{n-1})\}$$

# Polynomial Arithmetic in Point-Value Representation

- ▶ First attempt: Let  $x_0, \dots, x_{n-1}$  be distinct and suppose

$$A(x) \equiv \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

$$B(x) \equiv \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$$

- ▶ Then surely,

$$C(x) \equiv \{(x_0, y_0 z_0), (x_1, y_1 z_1), \dots, (x_{n-1}, y_{n-1} z_{n-1})\}$$

- ▶ **Issue:** While  $C(x_i) = y_i z_i$ ,  $C$  is a degree  $2n - 2$  polynomial and we need  $2n - 1$  distinct points to specify it.

# Polynomial Arithmetic in Point-Value Representation

- ▶ First attempt: Let  $x_0, \dots, x_{n-1}$  be distinct and suppose

$$A(x) \equiv \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

$$B(x) \equiv \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$$

- ▶ Then surely,

$$C(x) \equiv \{(x_0, y_0 z_0), (x_1, y_1 z_1), \dots, (x_{n-1}, y_{n-1} z_{n-1})\}$$

- ▶ **Issue:** While  $C(x_i) = y_i z_i$ ,  $C$  is a degree  $2n - 2$  polynomial and we need  $2n - 1$  distinct points to specify it.
- ▶ **Fix:** Assume  $A$  and  $B$  are specified on  $2n - 1$  distinct points. Can compute PVR of  $C$  is  $\Theta(n)$  time.

## Polynomial Arithmetic in Point-Value Representation

- ▶ First attempt: Let  $x_0, \dots, x_{n-1}$  be distinct and suppose

$$A(x) \equiv \{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

$$B(x) \equiv \{(x_0, z_0), (x_1, z_1), \dots, (x_{n-1}, z_{n-1})\}$$

- ▶ Then surely,

$$C(x) \equiv \{(x_0, y_0 z_0), (x_1, y_1 z_1), \dots, (x_{n-1}, y_{n-1} z_{n-1})\}$$

- ▶ **Issue:** While  $C(x_i) = y_i z_i$ ,  $C$  is a degree  $2n - 2$  polynomial and we need  $2n - 1$  distinct points to specify it.
- ▶ **Fix:** Assume  $A$  and  $B$  are specified on  $2n - 1$  distinct points. Can compute PVR of  $C$  is  $\Theta(n)$  time.
- ▶ But what about coefficient representation?

# Framework for Fast Polynomial Multiplication

- ▶ Input: Coefficient representation of  $A(x)$  and  $B(x)$

# Framework for Fast Polynomial Multiplication

- ▶ Input: Coefficient representation of  $A(x)$  and  $B(x)$
- ▶ Step 1: Transform into PVR by evaluating at  $2n - 1$  points

# Framework for Fast Polynomial Multiplication

- ▶ Input: Coefficient representation of  $A(x)$  and  $B(x)$
- ▶ Step 1: Transform into PVR by evaluating at  $2n - 1$  points
- ▶ Step 2: Multiply polynomials to get  $C(x)$  in PVR

# Framework for Fast Polynomial Multiplication

- ▶ Input: Coefficient representation of  $A(x)$  and  $B(x)$
- ▶ Step 1: Transform into PVR by evaluating at  $2n - 1$  points
- ▶ Step 2: Multiply polynomials to get  $C(x)$  in PVR
- ▶ Step 3: Transform PVR of  $C(x)$  back into CR.

# Framework for Fast Polynomial Multiplication

- ▶ Input: Coefficient representation of  $A(x)$  and  $B(x)$
- ▶ Step 1: Transform into PVR by evaluating at  $2n - 1$  points
- ▶ Step 2: Multiply polynomials to get  $C(x)$  in PVR
- ▶ Step 3: Transform PVR of  $C(x)$  back into CR.

**Important:** We can choose any distinct points for the PVR. Let's use the complex roots of unity...

# Complex Roots of Unity

## Definition

The  $n$ -th roots of unity are the  $n$  complex solutions to the equation  $x^n = 1$ , i.e.,

$$e^{2\pi ik/n} = \cos \frac{2\pi k}{n} + i \sin \frac{2\pi k}{n} \quad k = 0, \dots, n-1.$$

Let  $\omega_n = e^{2\pi i/n}$ .

# Complex Roots of Unity

## Definition

The  $n$ -th roots of unity are the  $n$  complex solutions to the equation  $x^n = 1$ , i.e.,

$$e^{2\pi ik/n} = \cos \frac{2\pi k}{n} + i \sin \frac{2\pi k}{n} \quad k = 0, \dots, n-1.$$

Let  $\omega_n = e^{2\pi i/n}$ .

## Lemma

*If  $n$  is even, then the squares of the  $n$ -th roots of unity are two copies of the  $n/2$ -th roots of unity:*

$$\{(\omega_n^0)^2, (\omega_n^1)^2, \dots, (\omega_n^{n-1})^2\}$$

*equals two copies of  $\{\omega_{n/2}^0, \omega_{n/2}^1, \dots, \omega_{n/2}^{n/2-1}\}$ .*