

Divide & Conquer II

(high-level comment)

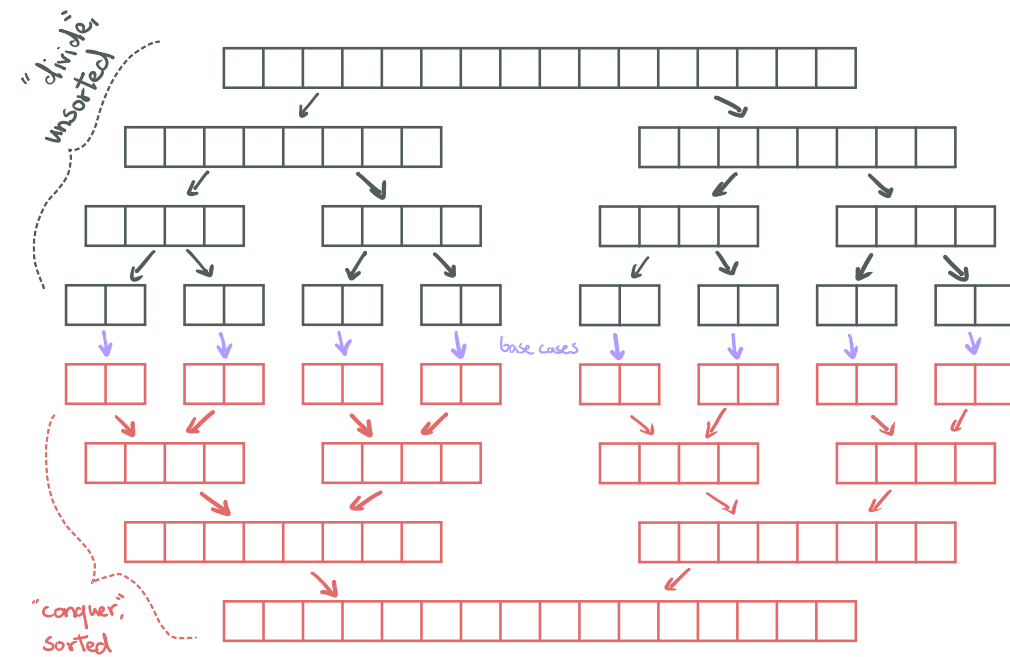
"only two algorithms"

① identify subproblems

② change the input

Divide + Conquer

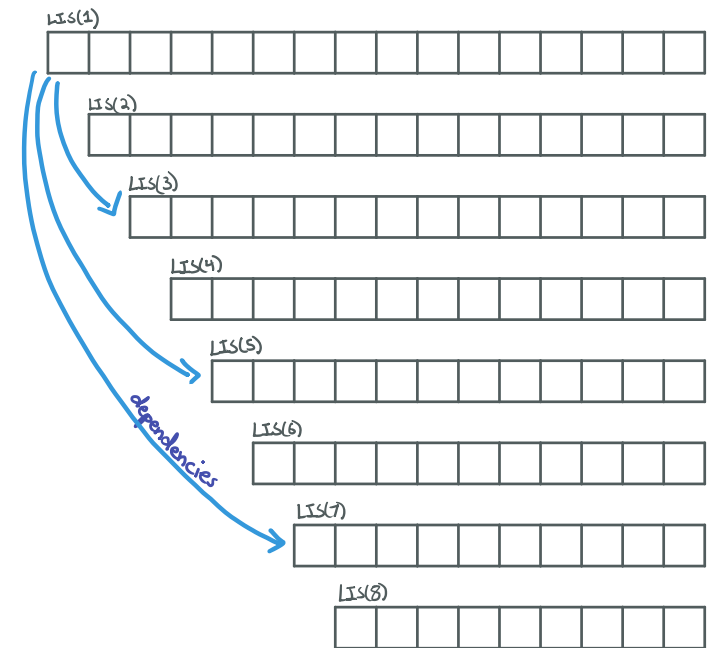
$\text{merge-sort}(A[1..n])$ = an array consisting of the elements of A sorted in increasing order.



- fewer, mostly disjoint subproblems
- correctness proof relatively easy
- very economical
- nearly linear time

Dynamic Programming

$\text{LIS}(i)$ = the length of the longest increasing subsequence of $A[i..n]$ where the subsequence starts with $A[i]$.



- many highly overlapping subproblems
- correctness more subtle
- simulates brute force/exhaustive search
- polynomial time

Divide + Conquer

① identify subproblems

② change the input

I. Integer multiplication

II. Selection

III. Minimum Euclidean distance

IV. Fourier Transforms

} last time

I. Integer multiplication

Input: $x, y \in \{0, 1\}^n$

Output: product $(xy) \in \{0, 1\}^{2n}$

$$\begin{array}{r} 1234 \\ \times 5678 \\ \hline 9872 \\ 8638 \\ + \\ \hline \end{array}$$

$O(n^2)$ time

Better?

Divide bits in half

$$x = x_1 2^{n/2} + x_2 \quad x_1, x_2, y_1, y_2 \in \{0, 1\}^{n/2}$$

$$y = y_1 2^{n/2} + y_2$$

$$x \cdot y = x_1 y_1 2^n + (x_1 y_2 + y_1 x_2) 2^{n/2} + x_2 y_2$$

which is 4 multiplies of $\frac{n}{2}$ bits

multiply($x[1..n], y[1..n]$)

1. If $n = 0$ then return 0.
2. If $n = 1$ then return $x[1] * y[1]$.
3. If n is not even then pad x and y with a leading 0, making n even.
// $O(n)$.
4. Let $x_1 = x[1..n/2]$, $x_2 = x[n/2+1..n]$, $y_1 = y[1..n/2]$, and $y_2 = y[n/2+1..n]$ (as $n/2$ -bit integers).
// $O(n)$.
5. Let $x_3[1..n/2+1] = x_1 + x_2$ and $y_3[1..n/2+1] = (y_1 + y_2)$ (as $(n/2+1)$ -bit integers).
// $O(n)$.
6. Let $Z_1 = \text{multiply}(x_1, y_1)$, $Z_2 = \text{multiply}(x_2, y_2)$, and $Z_3 = \text{multiply}(x_3, y_3)$.
// $3T((n+2)/2)$.

/* Note that multiplying by 2^n is just a bit-shift and takes $O(n)$ time. */

7. Return $2^n * Z_1 + 2^{n/2} * (Z_3 - Z_1 - Z_2) + Z_2$

Karatsuba's Trick

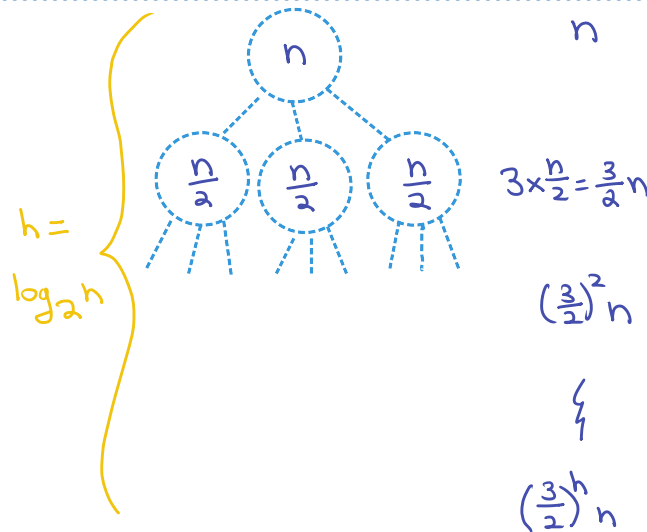
$$x \cdot y = x_1 y_1 2^n + (x_1 y_2 + y_1 x_2) 2^{n/2} + x_2 y_2$$

already computed

$$(x_1 + x_2)(y_1 + y_2) = \underbrace{x_1 y_1}_{\text{one multiply}} + \underbrace{x_1 y_2 + x_2 y_1}_{\text{middle term}} + x_2 y_2$$

$$x_1 y_2 + x_2 y_1 = x_1 y_1 + x_2 y_2 - (x_1 + x_2)(y_1 + y_2)$$

2 multiplies for the price of 1 (more)!



Running time

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

$$\sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i n = O\left(\left(\frac{3}{2}\right)^{\log_2 n} \cdot n\right) = O(n^{\log_2(3/2) + 1})$$

$$= O(n^{\log_2(3)})$$

II. Selection

Selection

Input Numbers $A[1..n]$ (not sorted)
index k

Goal: k th smallest element of $A[1..n]$

Easy: sort A , find k th
 $\Rightarrow O(n \log n)$

Better?

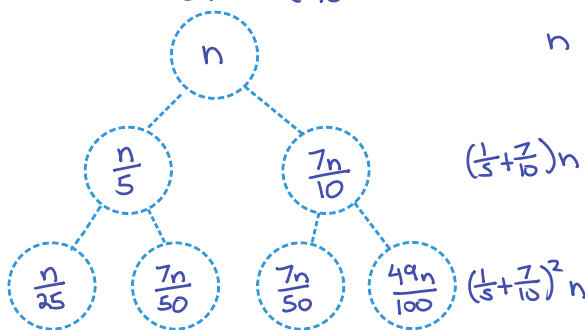
"divide along pivot"

Thought experiment: suppose we have

$\text{median}(A[1..n]) = \text{select}(\lceil n/2 \rceil, A[1..n])$
= the $\lceil n/2 \rceil$ th smallest element out of $A[1..n]$

in $O(n)$. we use median as pivot.

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$



$$\left(\frac{1}{5} + \frac{7}{10}\right)^3 n$$

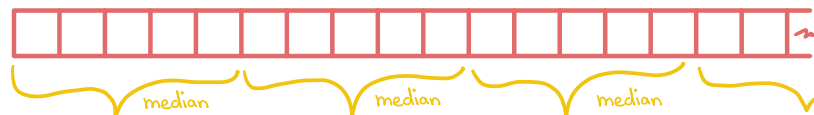
$$\sum_{i=0}^{\infty} \left(\frac{1}{5} + \frac{7}{10}\right)^i n = O(n)$$

{

"Median of Medians"

Blum, Floyd, Pratt,
Rivest, Tarjan

A



B



$n/5$

□ pivot p

$\text{select}(k, A[1..n])$

/* $\text{select}(k, A[1..n])$ = the k th smallest element in $A[1..n]$. We assume the elements in A are distinct for simplicity. (Otherwise, break ties consistently.) */

/* Here we implement $\text{select}(k, A[1..n])$ assuming a linear time subroutine for $\text{median}(A[1..n])$. */

1. If $n \leq 10$ then find the k th element by brute force.
2. Let $p = \text{median-of-medians}(A)$.
3. Compare p to each element in $A[1..n]$ to identify its rank ℓ .
4. If $k = \ell$ then return p .
5. If $k > \ell$

A. Let $B[1.. \ell - 1]$ be a list/array of the $\ell - 1$ elements smaller than p .

B. Return $\text{select}(k, B)$

6. If $k < \lceil n/2 \rceil$

A. Let $B[1.. n - \lceil n/2 \rceil]$ be a list/array of the $n - \lceil n/2 \rceil$ elements larger than p .

B. Return $\text{select}(k - \lceil n/2 \rceil, B)$

Claim

M-O-M pivot p has $\frac{3n}{10} \leq \text{rank}(p) \leq \frac{7n}{10}$

B sorted $\rightarrow$,



each set of 5 sorted

B



$n/5$



/// = def. smaller

/// = def. bigger

$$\frac{n}{5} \times \frac{1}{2} \times 3 = \frac{3n}{10}$$

Divide + Conquer

① identify subproblems

② change the input

I. Integer multiplication

II. Selection

} last
time

III. Minimum Euclidean distance

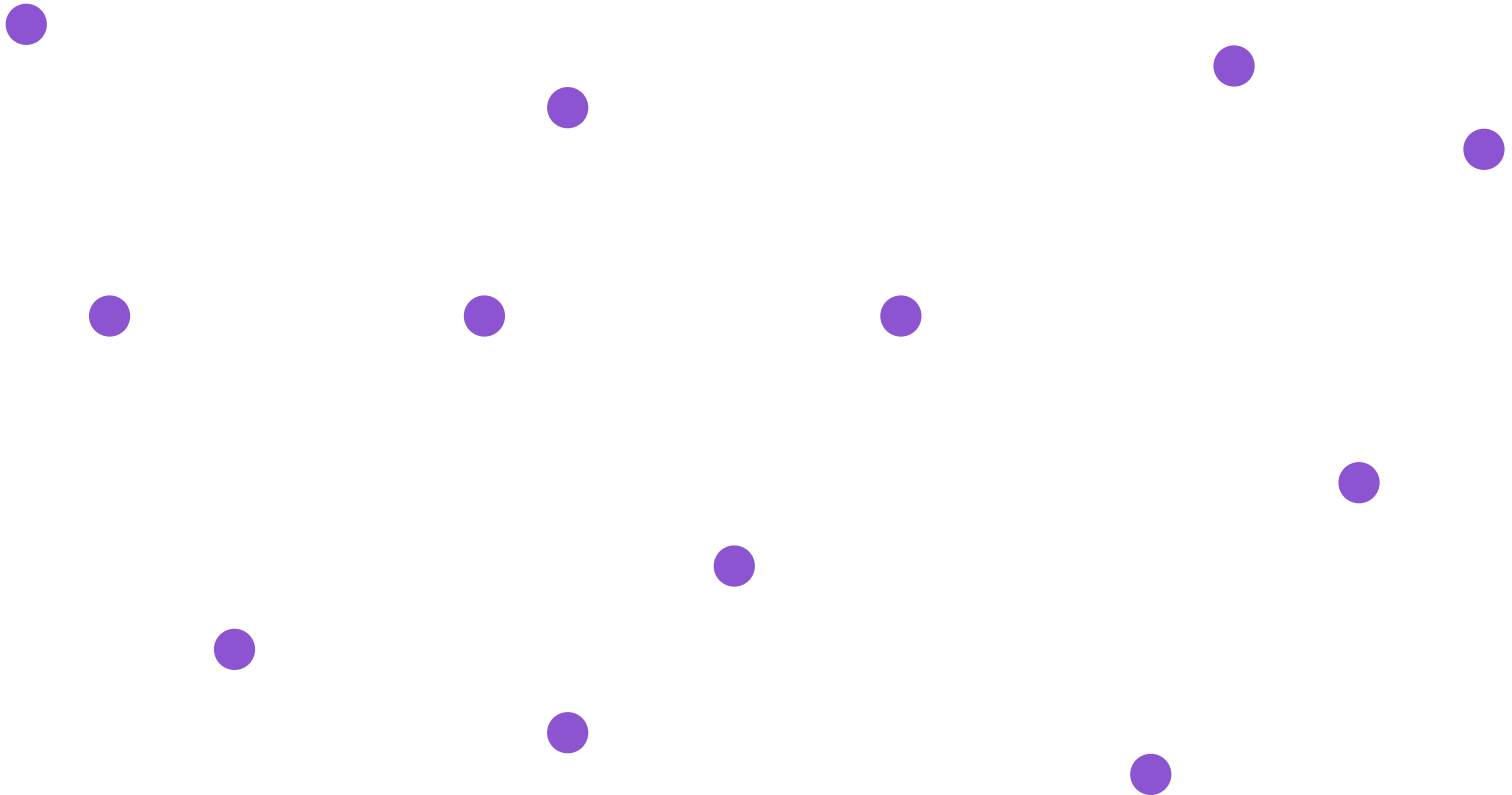
IV. Fourier Transforms

III. Minimum Euclidean distance

Input: n points in $\mathbb{R}^2$

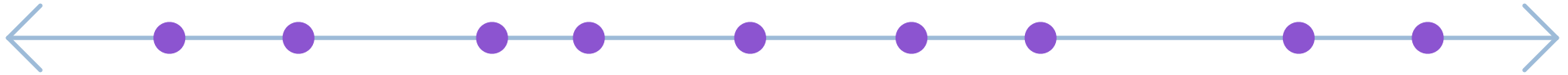
Goal: min distance over all pairs

$$\|x-y\| = \sqrt{(x_1-y_1)^2 + (x_2-y_2)^2}$$

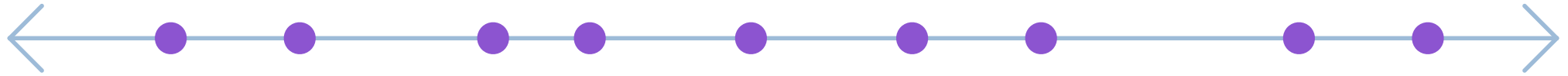


$O(n^2)$ obvious. Better?

Warmup: 1 dimension (all y -coordinates = 0)

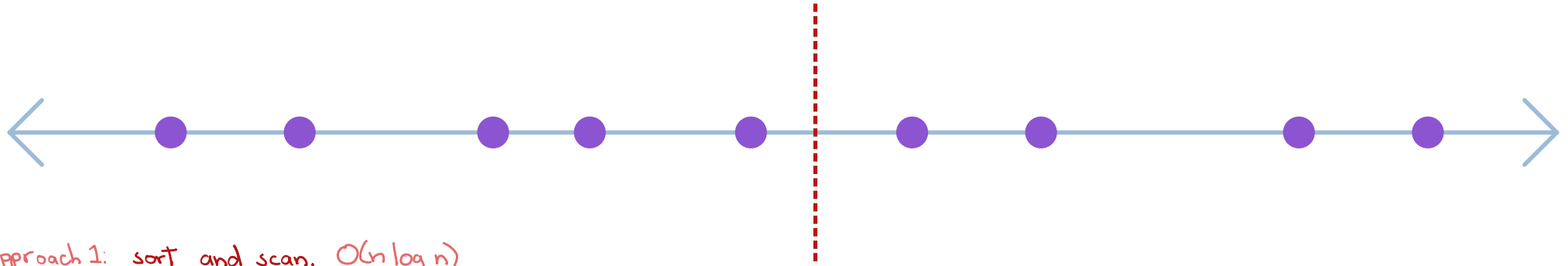


Warmup: 1 dimension (all y-coordinates = 0)



Approach 1: sort and scan. $O(n \log n)$
(hard to generalize to $\mathbb{R}^2$)

Warmup: 1 dimension (all y-coordinates = 0)



Approach 1: sort and scan. $O(n \log n)$
(hard to generalize to $\mathbb{R}^2$)

Approach 2.

Split at median, recurse

Compare w/ min pair "across" split

$O(n \log n)$ (like merge sort)

seems to generalize better

generalizing to $\mathbb{R}^2$

Approach 2.

Split at median, recurse

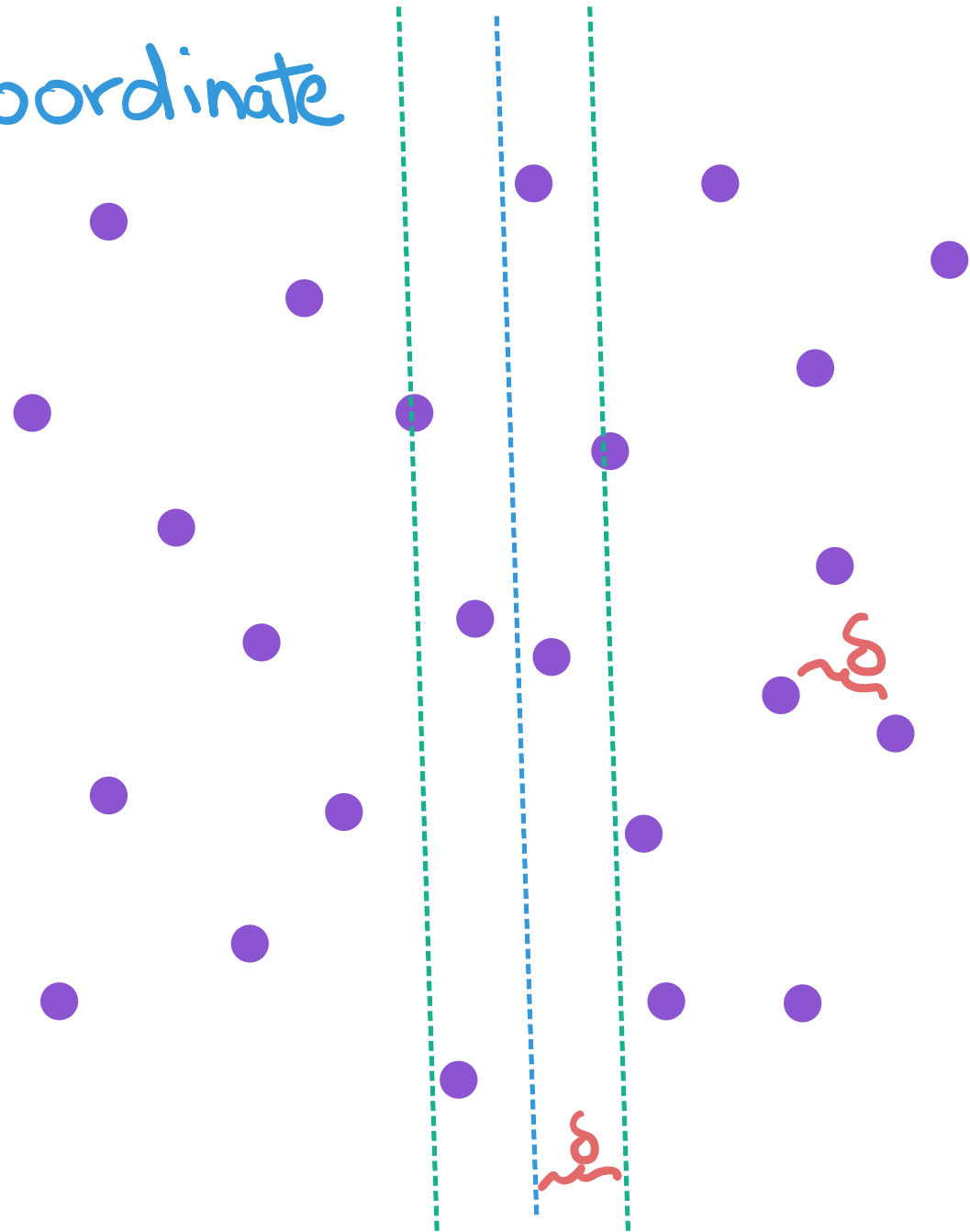
Compare w/ min pair "across" split

1. split at median x-coordinate

2. $\delta = \min$ dist. of
each half

3. look at "vertical band"
of width 2δ around
"median line"

4. get min distance
inside band



generalizing to $\mathbb{R}^2$

1. split at median x-coordinate
2. $\delta = \min$ dist. of each half
3. look at "vertical band" of width 2δ around "median line"

4. get min distance inside band

Approach 2.

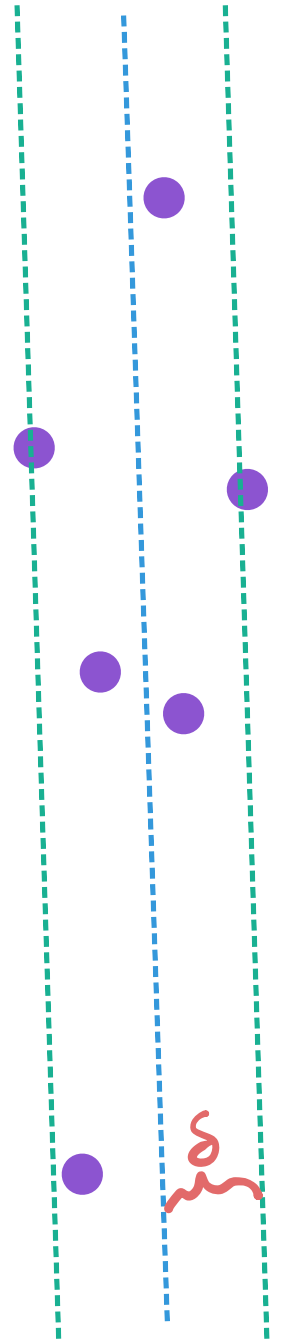
Split at median, recurse

Compare w/ min pair "across" split

Observations

- very narrow
- points on left side have
min distance $\geq \delta$
- points on right side have
min distance $\geq \delta$

intuitively: not dense,
fewer comparisons needed



generalizing to $\mathbb{R}^2$

1. split at median x -coordinate
2. $\delta = \min$ dist. of each half
3. look at "vertical band" of width 2δ around "median line"
4. get min distance inside band

Observations

- very narrow
 - points on left side have min distance $\geq \delta$
 - points on right side have min distance $\geq \delta$
- intuitively: not dense, fewer comparisons needed

Approach 2.

Split at median, recurse

Compare w/ min pair "across" split

min-distance(P)

/ We assume P is sorted by y -coordinate. (Otherwise sort P once in a preprocessing step.) */*

1. If $n \leq 1$ then return $+\infty$.

2. If $n = 2$ then return the distance between the two points in P .

3. Let a be the median of the x -coordinates.

// $O(n)$

/ Divide P along the vertical line $x = z$*

**/*

4. Let $P_1 = \{(x, y) \in P : x < a\}$ and $P_2 = \{(x, y) \in P : x \geq a\}$

// $O(n)$

/ Find the minimum distances within each half P_1 and P_2 recursively.*

**/*

5. Let $\delta_1 = \text{min-distance}(P_1)$, $\delta_2 = \text{min-distance}(P_2)$, and $\delta = \min\{\delta_1, \delta_2\}$.

// $2T(n/2)$

/ It remains to find the minimum distance across P_1 and P_2 .*

**/*

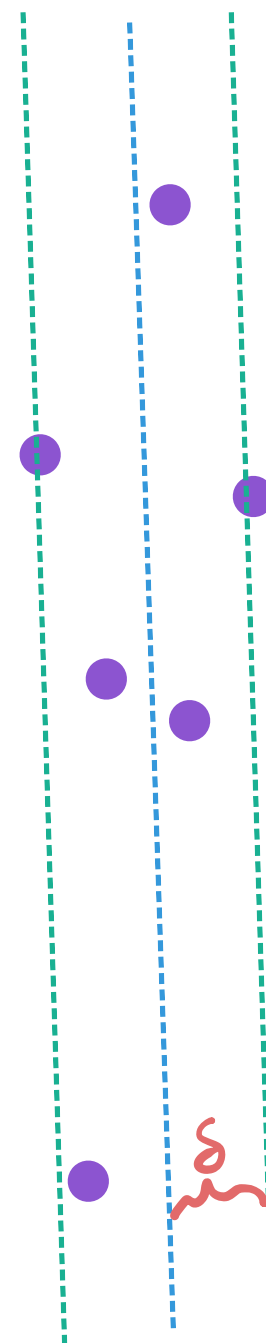
6. Let $P_3 = \{(x, y) \in P : |x - z| < \delta\}$ and let $p_1, \dots, p_k$ list P_3 in decreasing order of y -coordinate.

// $O(n)$

7. Let δ_3 be the minimum of $\|p_i - p_j\|$ over all i, j with $i < j < i + 10$.

// $O(n)$

8. Return $\min\{\delta, \delta_3\}$.



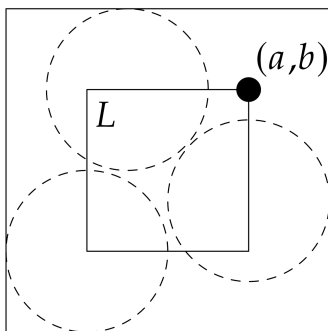
(a = median x-coordinate)

Lemma 11.3. *Let $b \in \mathbb{R}$. There are at most 10 points in P with coordinate in the rectangle $[a - \delta, a + \delta] \times [b - \delta, b]$,*

Observations

- very narrow
- points on left side have min distance $\geq \delta$
- points on right side have min distance $\geq \delta$
- intuitively: not dense, fewer comparisons needed

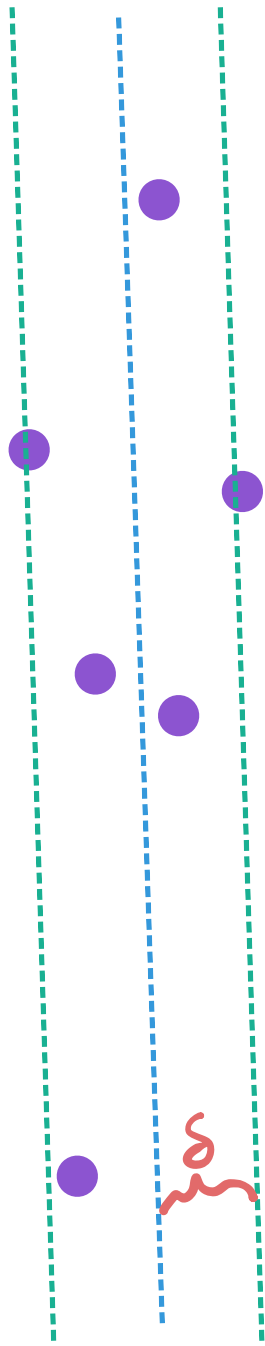
$$L = [a - \delta, a) \times [b - \delta, b] \text{ and } R = [a, a + \delta] \times [b - \delta, b]$$



```

min-distance(P)
/* We assume P is sorted by y-coordinate. (Otherwise sort P once in a preprocessing step.) */
1. If n ≤ 1 then return +∞.
2. If n = 2 then return the distance between the two points in P.
3. Let a be the median of the x-coordinates. // O(n)
/* Divide P along the vertical line x = a */
4. Let P1 = {(x, y) ∈ P : x < a} and P2 = {(x, y) ∈ P : x ≥ a} // O(n)
/* Find the minimum distances within each half P1 and P2 recursively. */
5. Let δ1 = min-distance(P1), δ2 = min-distance(P2), and δ = min{δ1, δ2}. // 2T(n/2)
/* It remains to find the minimum distance across P1 and P2. */
6. Let P3 = {(x, y) ∈ P : |x - a| < δ} and let p1, ..., pk list P3 in decreasing order of y-coordinate. // O(n)
7. Let δ3 be the minimum of ||pi - pj|| over all i, j with i < j < i + 10. // O(n)
8. Return min{δ, δ3}.

```



Lemma 11.3. Let $b \in \mathbb{R}$. There are at most 10 points in P with coordinate in the rectangle $[a - \delta, a + \delta] \times [b - \delta, b]$,

Observations

- very narrow
- points on left side have min distance $\geq \delta$
- points on right side have min distance $\geq \delta$
- intuitively: not dense, fewer comparisons needed

min-distance(P)

/ We assume P is sorted by y -coordinate. (Otherwise sort P once in a preprocessing step.) */*

1. If $n \leq 1$ then return $+\infty$.
2. If $n = 2$ then return the distance between the two points in P .
3. Let a be the median of the x -coordinates.

// $O(n)$

/ Divide P along the vertical line $x = z$*

**/*

4. Let $P_1 = \{(x,y) \in P : x < a\}$ and $P_2 = \{(x,y) \in P : x \geq a\}$

// $O(n)$

/ Find the minimum distances within each half P_1 and P_2 recursively.*

**/*

5. Let $\delta_1 = \text{min-distance}(P_1)$, $\delta_2 = \text{min-distance}(P_2)$, and $\delta = \min\{\delta_1, \delta_2\}$.

// $2T(n/2)$

/ It remains to find the minimum distance across P_1 and P_2 .*

**/*

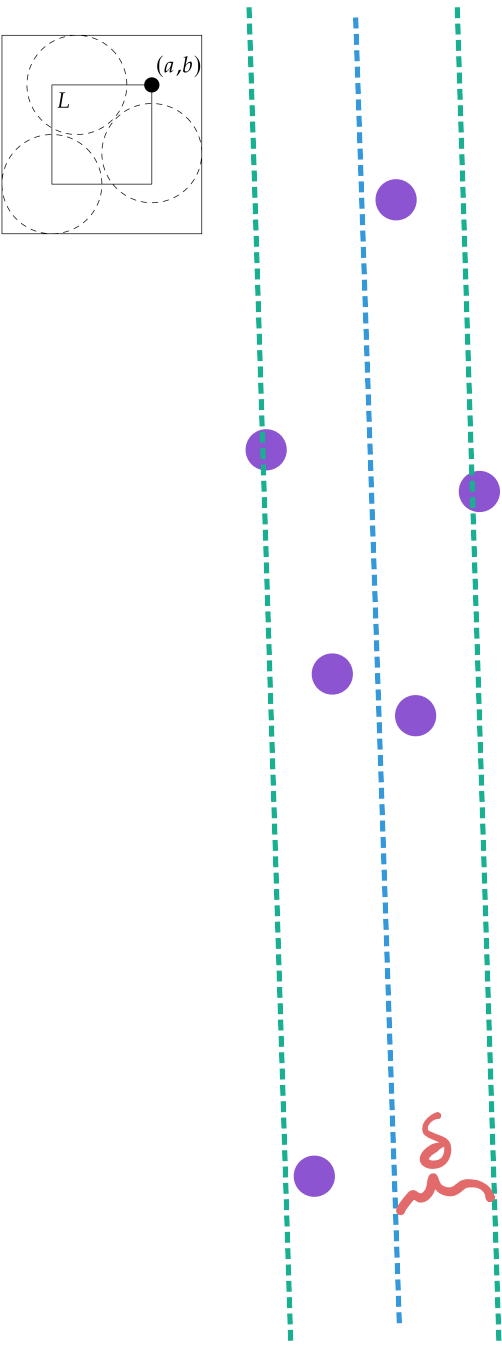
6. Let $P_3 = \{(x,y) \in P : |x - z| < \delta\}$ and let $p_1, \dots, p_k$ list P_3 in decreasing order of y -coordinate.

// $O(n)$

7. Let δ_3 be the minimum of $\|p_i - p_j\|$ over all i, j with $i < j < i + 10$.

// $O(n)$

8. Return $\min\{\delta, \delta_3\}$.



min-distance(P)

/ We assume P is sorted by y -coordinate. (Otherwise sort P once in a preprocessing step.) */*

1. If $n \leq 1$ then return $+\infty$.

2. If $n = 2$ then return the distance between the two points in P .

3. Let a be the median of the x -coordinates. $\parallel O(n)$

/ Divide P along the vertical line $x = z$ */*

4. Let $P_1 = \{(x, y) \in P : x < a\}$ and $P_2 = \{(x, y) \in P : x \geq a\}$ $\parallel O(n)$

/ Find the minimum distances within each half P_1 and P_2 recursively. */*

5. Let $\delta_1 = \text{min-distance}(P_1)$, $\delta_2 = \text{min-distance}(P_2)$, and $\delta = \min\{\delta_1, \delta_2\}$.

$\parallel 2T(n/2)$

/ It remains to find the minimum distance across P_1 and P_2 . */*

6. Let $P_3 = \{(x, y) \in P : |x - z| < \delta\}$ and let $p_1, \dots, p_k$ list P_3 in decreasing order of y -coordinate. $\parallel O(n)$

7. Let δ_3 be the minimum of $\|p_i - p_j\|$ over all i, j with $i < j < i + 10$. $\parallel O(n)$

8. Return $\min\{\delta, \delta_3\}$.

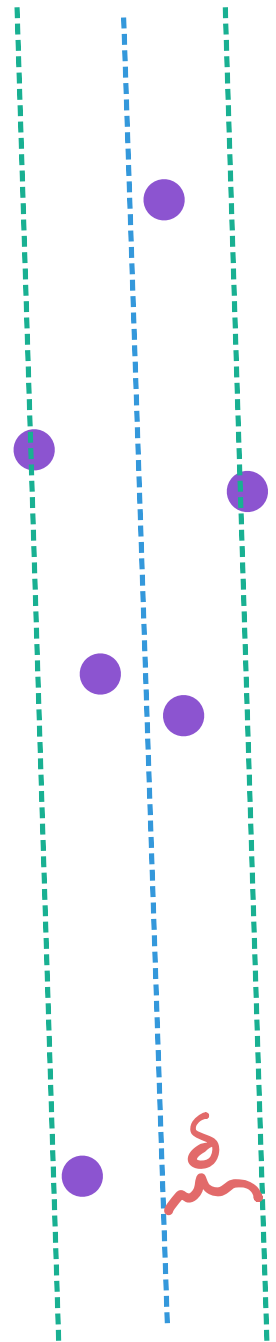
Observations

- very narrow

- points on left side have min distance $\geq \delta$

- points on right side have min distance $\geq \delta$

intuitively: not dense, fewer comparisons needed



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

Divide + Conquer

① identify subproblems

② change the input

I. Integer multiplication

II. Selection

III. Minimum Euclidean distance

IV. Fourier Transforms

} last time

Complex numbers $\mathbb{C}$

"imaginary i "

$$i^2 = -1$$

$$a + bi$$

"real part" "imaginary part"

• $a + bi$

$$(a+bi)(c+di) = ac + (bct+ad)i - bd$$

polynomials over $\mathbb{C}$:

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

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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



"Euler's identity"

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

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

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bct+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

n th roots of unity

any $w \in \mathbb{C}$ s.t. $w^n = 1$

e.g. $w = 1$, $w = -1$ for n even,
 $w = i$ for $n = 4$

$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$ is n th root of unity

$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$
give all n th roots of unity

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) \\ = ac + (bct+ad)i - bd$$

polynomials over $\mathbb{C}$:

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

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bctad)i - bd$$

polynomials over $\mathbb{C}$:

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

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(nth) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

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

Output:

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

"conjugate"

$$F_n a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

Application: mult. polynomials

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) \\ = ac + (bctad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^{n-1}$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

Application: mult. polynomials

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) \\ = ac + (bctad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^{n-1}$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

Lemma 11.7. For any n th root of unity ω except for $\omega = 1$, we have

$$1 + \omega + \omega^2 + \cdots + \omega^{n-1} = 0.$$

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^{n-1}$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

IV. Fourier Transforms

Lemma 11.7. For any n th root of unity ω except for $\omega = 1$, we have

$$1 + \omega + \omega^2 + \cdots + \omega^{n-1} = 0.$$

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = n a$.

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^{n-1}$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

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

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

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

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

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

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

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

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

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

$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

IV. Fourier Transforms

Complex numbers $\mathbb{C}$

$$(a+bi)(c+di) = ac + (bc+ad)i - bd$$

polynomials over $\mathbb{C}$:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

power series over $\mathbb{C}$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Euler's formula

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

"Euler's identity"

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

n th roots of unity

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

$$\omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}, \omega_n$$

(n th) discrete Fourier transform

Input: $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$

$$\text{let } p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$$

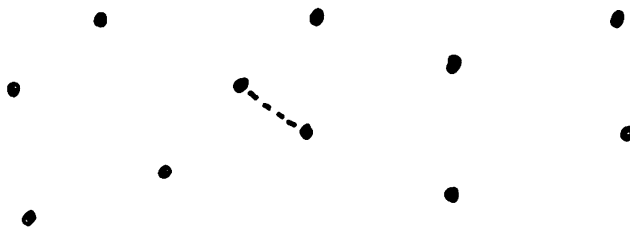
$$F_n a = (p(1), p(\omega_n), \dots, p(\omega_n^{n-1}))$$

$$F_n^* a = (p(1), p(\omega_n^{-1}), \dots, p(\omega_n^{1-n}))$$

time and we will formalize the argument later.

The randomized algorithm works better in practice.

11.3 Minimum distance in $\mathbb{R}^2$



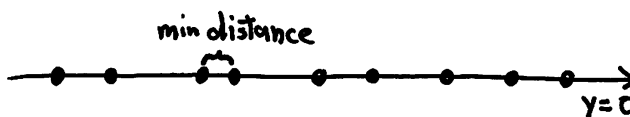
We consider the problem of computing the minimum pairwise distance among a set of points in the plane. The input consists of n points P in $\mathbb{R}^2$. The goal is to find the minimum Euclidean distance between any two points $p, q \in P$. Here we recall that for two points $p = (p_1, p_2)$ and $q = (q_1, q_2)$, the Euclidean distance $\|p - q\|$ is given by

$$\|p - q\| = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}.$$

We assume for simplicity that all the coordinates of all the points are distinct.⁴ Of course there is an $O(n^2)$ -time algorithm where we simply calculate the distance of every pair. The question is whether one could do better. In anticipation of a recursive algorithm, we first declare a recursive spec.

min-distance(P) = the minimum Euclidean distance between any pair of points in P (or $+\infty$ if there are less than 2 points in P).

The goal is to implement min-distance(P) exactly as described above.



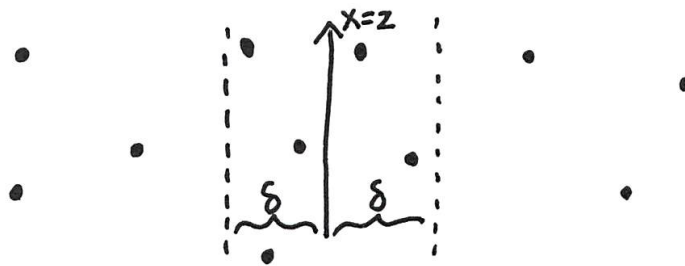
To build some intuition, it is helpful to consider the 1-dimensional case; or equivalently, assume that all points have the same y -coordinate, (say) $y = 0$. (We continue to assume the x -coordinates are distinct.) The most obvious approach is to sort the points by x -coordinate, and then scan and check all the consecutive pairs in sorted order. This takes $O(n \log n)$ time. This is very good but it does

⁴The fancy way to state this assumption is to say that the points are in “general position”. In practice, we can often assume general position by first perturbing the points by a sufficiently small amount. We also point out that the algorithms here are simple enough to adapt to duplicate coordinates. We assume general position to keep the algorithm clean and focus on the high-level ideas.

not seem to generalize to general y -coordinates. In $\mathbb{R}^2$, the minimum distance is not necessarily attained by a consecutive pair of points when listed in increasing order of x .

A second approach to the one-dimensional, $y = 0$ setting is to first find the median x -coordinate z . (We can compute z in $O(n)$ time via select from section 11.2). Let P_1 be the subset of points with x -coordinate $< z$ and P_2 be the subset of points with x -coordinate $\geq z$. We recursively compute the minimum distance in P_1 and P_2 . Let δ be the minimum of the two values returned. δ is the minimum distance among pairs of points within P_1 and within P_2 ; it remains to check the minimum distance of pairs *across* the two halves. To this end, we calculate the distance between the right-most-point on the left half P_1 , and the left-most-point on the right half P_2 . We return the minimum of this distance and δ .

The second approach almost generalizes to $\mathbb{R}^2$, except for the last step where we get the minimum distance across the two halves. The minimum distance across P_1 and P_2 is not necessarily between the two points closest to the dividing line $x = z$.



We do know that that any point in the minimum cross pair has x -coordinate within δ of z . (Otherwise the x -coordinate alone makes the distance of the cross-pair $> \delta$). Let

$$P_3 = \{(x, y) \in P : |x - z| \leq \delta\}$$

gather up all the points in P whose x -coordinates are within δ of z . Now consider the following geometric facts.

1. P_3 lies in a narrow, vertical band of width 2δ .
2. Any two points in $P_1 \cap P_3$ have distance at least δ , and any two points in $P_2 \cap P_3$ have distance at least δ . This generally forces P_3 to be spread apart.

Combining the two facts leads to the following observation (formalized in lemma 11.3):

For any point $p \in P_3$, there are at most a constant number of other points with y -coordinate within δ of p .

```

min-distance( $P$ )
/* We assume  $P$  is sorted by  $y$ -coordinate. (Otherwise sort  $P$  once in a preprocessing step.) */
1. If  $n \leq 1$  then return  $+\infty$ .
2. If  $n = 2$  then return the distance between the two points in  $P$ .
3. Let  $a$  be the median of the  $x$ -coordinates. //  $O(n)$ 
/* Divide  $P$  along the vertical line  $x = z$  */
4. Let  $P_1 = \{(x, y) \in P : x < a\}$  and  $P_2 = \{(x, y) \in P : x \geq a\}$  //  $O(n)$ 
/* Find the minimum distances within each half  $P_1$  and  $P_2$  recursively. */
5. Let  $\delta_1 = \text{min-distance}(P_1)$ ,  $\delta_2 = \text{min-distance}(P_2)$ , and  $\delta = \min\{\delta_1, \delta_2\}$ . //  $2T(n/2)$ 
/* It remains to find the minimum distance across  $P_1$  and  $P_2$ . */
6. Let  $P_3 = \{(x, y) \in P : |x - z| < \delta\}$  and let  $p_1, \dots, p_k$  list  $P_3$  in decreasing order of  $y$ -coordinate. //  $O(n)$ 
7. Let  $\delta_3$  be the minimum of  $\|p_i - p_j\|$  over all  $i, j$  with  $i < j < i + 10$ . //  $O(n)$ 
8. Return  $\min\{\delta, \delta_3\}$ .

```

Figure 11.8: Computing the minimum distance in $\mathbb{R}^2$ by divide-and-conquer along the horizontal axis.

We take advantage of this by sorting P_3 by y -coordinate, and only compute the distance between pairs of points that are within 10 ranks of one another in the sorted list. (By lemma 11.3, 10 is a sufficiently large constant.)

Pseudocode of the divide-and-conquer algorithm described above is given in fig. 11.8. As a relatively minor point, instead of sorting by y -coordinate in each recursive call, we can sort all of P once in a preprocessing step. We assume that P is already sorted by y -coordinate in the implementation in fig. 11.8.

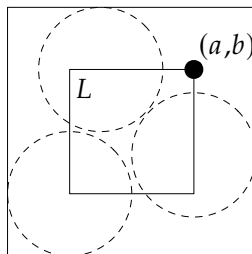
Lemma 11.3. *Let $b \in \mathbb{R}$. There are at most 10 points in P with coordinate in the rectangle $[a - \delta, a + \delta] \times [b - \delta, b]$,*

Proof. The following is an example of a “packing argument”, which is common in computational geometry.

We split the rectangle into two squares

$$L = [a - \delta, a) \times [b - \delta, b] \text{ and } R = [a, a + \delta] \times [b - \delta, b].$$

For each half we show that there are at most 5 points.



Consider the left half L . Any pair of points in L , being both in P_1 , has distance at least δ . For each point $p \in P \cap L$, draw a ball B_p with radius $\delta/2$ centered at p . Each ball B_p has area $\pi\delta^2/4$, and lies in the “padded” rectangle $[a - 1.5\delta, a + .5\delta] \times [b - 1.5\delta, b + .5\delta]$ of area $2\delta \times 2\delta = 4\delta^2$. These balls are also interior disjoint because the minimum distance between their centers is δ . The number of these balls that can fit in the padded rectangle, by comparing areas, is at most

$$\frac{4\delta^2}{\pi\delta^2/4} = \frac{16}{\pi} \approx 5.09.$$

Since the maximum number is also integer, we conclude there can be at most 5 balls, hence 5 points in $P \cap L$.

By the same argument (now applied to P_2 , instead of P_1) there are at most 5 points in $R \cap P$. ■

The overall proof of correctness is as follows. We will prove that fig. 11.8 correctly implements $\text{min-distance}(P)$ as specified above by induction on $|P|$. The base cases are when $|P| \leq 2$; if $|P| < 1$ then the distance is automatically $+\infty$; if $|P| = 2$, there is only one pairwise distance to consider and that is the value returned. Consider the general case where $|P| > 2$. We assume by induction that the implementation is correct for all strictly smaller point sets.

After dividing the input size into P_1 and P_2 , we know that the minimum distance is either within P_1 , within P_2 , or across P_1 and P_2 . By induction we assume that $\text{min-distance}(P_1)$ and $\text{min-distance}(P_2)$ return the (true) minimum distance within P_1 and P_2 , respectively. For pairs going across P_1 and P_2 , it suffices to compute the minimum distance in P_3 since otherwise the difference in x -coordinate is at least δ . Within P_3 , by lemma 11.3, it suffices to compare each point $p \in P_3$ with the 10 points with closest y -value, which is exactly what the algorithm does. So the algorithm also finds the minimum distance within P_3

correctly. The minimum distance from the three sets P_1 , P_2 , and P_3 gives the minimum distance of all of P , as claimed.

As for the running time, we have two recursive calls to point sets of size $n/2$, plus $O(n)$ additional work (as annotated in the code). This gives a recursion of the form

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

This is the same recurrence as merge-sort and is solved by $T(n) = O(n \log n)$.

11.4 Fourier transforms

Complex numbers. The discrete Fourier transform is an operation applied to polynomials that takes place in the complex plane $\mathbb{C}$. Here we will present the minimal algebraic facts regarding $\mathbb{C}$. Recall that a complex number $x \in \mathbb{C}$ has the form

$$x = a + \imath b,$$

where $a, b \in \mathbb{R}$ are real-valued numbers. a is called the “real part” of x and b is called the “complex part” of b . $\imath$ denotes “imaginary i ”, defined by $\imath^2 = -1$. We have

$$(a + \imath b)(c + \imath d) = ac + (bc + ad)\imath + bdi^2 \stackrel{(a)}{=} ac + (bc + ad)\imath - bd;$$

here (a) applies the identity $\imath^2 = -1$. With this, polynomials

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

are well defined for $x \in \mathbb{C}$. So are functions that can be expressed as convergent power series, such as the exponential function:

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Euler’s formula gives a geometric interpretation of the exponential function:

$$e^{\theta \imath} = \cos \theta + \imath \sin \theta.$$

From this we have Euler’s identity,

$$e^{\pi \imath} = -1,$$

as well as the identity

$$e^{2\pi \imath} = 1.$$

Roots of unity. Now, let $n \in \mathbb{N}$ be fixed. An **n th root of unity** is any (complex) number ω satisfying

$$\omega^n = 1.$$

For example, 1 is an n th root of unity for any n , and -1 is an n th root of unity for any even n . i is a 4th root of unity. More generally,

$$\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$$

is an n th root of unity, as well as ω_n^j for any $j \in \mathbb{Z}$. In fact, the n distinct powers of ω_n ,

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

give all the n th roots of unity.

The discrete Fourier transform. The **(n th) discrete Fourier transform**, denoted F_n , takes as input a polynomial

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

represented by its coefficients

$$a_0, \dots, a_{n-1} \in \mathbb{C},$$

and produces the vector obtained by evaluating $p(x)$ at each of the n roots of unity⁵. Here, identifying p with its coefficients $a \in \mathbb{C}^n$, we write

$$F_n a \stackrel{\text{def}}{=} (p(1), p(\omega_n), p(\omega_n^2), \dots, p(\omega_n^{n-1})).$$

The **conjugate (n th) discrete Fourier transform**, denoted $F_n^* a$, instead lists the values for the powers of ω_n^{-1} :

$$F_n^* a \stackrel{\text{def}}{=} (p(1), p(\omega_n^{-1}), p(\omega_n^{-2}), \dots, p(\omega_n^{1-n}))$$

Since $\omega_n^{-1} = \omega_n^{n-1}$, F_n^* actually evaluates the same points as $F_n a$, but the order of values is partly reversed.

Later we will show that F_n and F_n^* are nearly inverse to one another, in the following sense. In particular, they can each be used to “undo” the other.

⁵We mention that there are other “continuous” variants for Fourier transforms that we do not discuss.

Fact 11.4. For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.

What is the fastest way to compute F_n ? The most direct way to compute the discrete Fourier transform is to just evaluate each $p(\omega_n^i)$ separately, in $O(n^2)$ time overall. The **fast Fourier transform** algorithm applies divide and conquer to give the following improved running time.

Theorem 11.5. F_n and F_n^* can be computed in $O(n \log n)$ time.

11.4.1 Application: multiplying polynomials.

Below we will have a more technical discussion that proves both fact 11.4 and theorem 11.5. But to motivate that discussion we first ask: why is F_n useful? We first mention that it is incredibly useful for signal processing and refer to the wikipedia article for more on this application. But as another basic application that requires no new definitions, consider the problem of multiplying two degree- $(n-1)$ polynomials $p(x)$ and $q(x)$, represented by coefficients $a, b \in \mathbb{C}^n$ in respectively. Let $r(x) = p(x)q(x)$ be their product, with coefficients $c \in \mathbb{C}^n$. We can compute the coefficients of $r(x)$ explicitly in $O(n^2)$ time, by take the appropriate sums of products of coefficients of p and q . For example, the coefficient c_k corresponding to x^k in r is given by the *convolution*

$$c_k = \sum_{i=0}^k a_i b_{k-i}.$$

This is not a complicated formula but computationally it does represent a loop of size k , where k can be as large as $2n$. Using the convolution above to compute $r = pq$, we take $O(n)$ time per coefficient c_k , and $O(n^2)$ time total.

Instead of focusing on the coefficients of r , let us consider its Fourier transform $F_{2n}r$. The values $r(x) = p(x)q(x)$ at the $(2n)$ th roots of unity ω_{2n}^i are given by

$$r(\omega_{2n}^i) = p(\omega_{2n}^i)q(\omega_{2n}^i)$$

for each i . In contrast to convolutions, each value takes $O(1)$ time given the values on the RHS. Thus, given $F_{2n}p$ and $F_{2n}q$, computing $F_{2n}r$ is very quick, taking only $O(n)$ time.

Now recall that F_{2n} computes all $2n$ roots of unity, and F_{2n}^* effectively reverses the process and recovers the polynomial (times a factor of n) from the values at the roots of unity. We can compute $r = pq$ much more quickly by (a) taking the Fourier transforms of p and q , (b) multiplying the values together to get the Fourier transform of r , and then using the conjugate Fourier transform

(and dividing by $2n$) to get the coefficients of r . By the fast Fourier transform (theorem 11.5), the whole procedure takes only $O(n \log n)$ time!

$$(p, q) \xrightarrow{O(n \log n)} (F_{2n}p, F_{2n}q) \xrightarrow{O(n)} F_{2n}r \xrightarrow{O(n \log n)} F_{2n}^* F_{2n}r = 2nr$$

Corollary 11.6. *Two polynomials of degree $n - 1$ can be multiplied in $O(n \log n)$ time.*

We also point out that the fastest integer multiplication algorithms are based on the Fourier transform. Indeed, multiplying integers is not so different from multiplying polynomials – think of the bits of an integer in binary notation as the n coefficients of a degree- n polynomial.

11.4.2 Inverting F_n and F_n^*

We first prove Fact 11.4, which described a near-inverse relationship between F_n and F_n^* . We require the following identity.

Lemma 11.7. *For any n th root of unity ω except for $\omega = 1$, we have*

$$1 + \omega + \omega^2 + \cdots + \omega^{n-1} = 0.$$

For $\omega = 1$, the LHS is n .

Proof. We have

$$(1 - \omega)(1 + \omega + \omega^2 + \cdots + \omega^{n-1}) = 1 - \omega^n = 0,$$

so either $\omega = 1$ or $1 + \omega + \cdots + \omega^{n-1} = 0$. ■

Now we prove Fact 11.4. We restate the claim below for the reader's convenience.

Fact 11.4. *For all $a \in \mathbb{C}^n$, $F_n^* F_n a = na$.*

Proof. Take any $a \in \mathbb{C}^n$; we want to show that $F_n^* F_n a = na$. Indeed, for each coordinate $i \in \{0, \dots, n-1\}$, we have

$$\begin{aligned} (F_n^* F_n a)_i &\stackrel{(a)}{=} \sum_{j=0}^{n-1} (F_n a)_j (\omega_n^i)^j \stackrel{(b)}{=} \sum_{j=0}^{n-1} \left(\sum_{k=0}^{n-1} a_k (\omega_n^j)^k \right) (\omega_n^i)^j \\ &\stackrel{(c)}{=} \sum_{k=0}^{n-1} a_k \sum_{j=0}^{n-1} (\omega_n^{k-i})^j \stackrel{(d)}{=} na_i. \end{aligned}$$

Here (a) and (b) expand out the definitions of F_n^* and F_n . (c) interchanges sums. (d) is by lemma 11.7: unless $k - i = 0$ (or some other multiple of n), we have $\sum_{j=0}^{n-1} (\omega_n^{k-i})^j = 0$. ■

One implication of **Fact 11.4** is as follows.

Corollary 11.8. *A polynomial $p(x)$ of degree $n - 1$ is uniquely identified by its value at the n roots of unity $p(1), p(\omega_n), \dots, p(\omega_n^{n-1})$.*

The above is a special case of the fundamental theorem of algebra.

11.4.3 The fast Fourier transform

We finally arrive at the algorithms for F_n and F_n^* . As noted above, F_n^* is a rearrangement of the coordinates produced by F_n , so it suffices to only present an algorithm for F_n .

Let $p(x)$ be a degree $(n - 1)$ -degree with coefficients $a_0, \dots, a_{n-1} \in \mathbb{C}$:

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

For ease of discussion let us assume that n is a power of 2. Our goal is to compute the vector of values

$$F_n a \stackrel{\text{def}}{=} (p(1), p(\omega_n), p(\omega_n^2), \dots, p(\omega_n^{n-1}))$$

To avoid ambiguity we first give a recursive spec for our algorithm. Given $n \in \mathbb{N}$ and a vector $a \in \mathbb{C}^n$ (indexed as $a_0, a_1, \dots, a_{n-1} \in \mathbb{C}$ by convention), we define

$\text{Fourier}(a_0, \dots, a_{n-1}) =$ the n th discrete Fourier transform

$$F_n a = (p(1), p(\omega_n), p(\omega_n^2), \dots, p(\omega_n^{n-1}))$$

where $p(x)$ is the polynomial $p(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$.

The divide and conquer argument exploits the following identity for ω_n (for all $n \in \mathbb{N}$):

$$\omega_{2n} = \omega_n^2. \tag{11.4}$$

Indeed, recalling that $\omega_n \stackrel{\text{def}}{=} e^{2\pi i/n}$, we have

$$\omega_{2n}^2 = e^{(2\pi i/2n)^2} = e^{2\pi i/n} = \omega_n.$$

Now, consider $(F_{2n} a)_i$ for fixed $n \in \mathbb{N}$ and $a \in \mathbb{C}^{2n}$. We have

$$\begin{aligned} (F_{2n} a)_i &\stackrel{(a)}{=} a_0 + a_1 \omega_{2n}^i + a_2 (\omega_{2n}^i)^2 + a_3 (\omega_{2n}^i)^3 + \dots + a_{2n-1} (\omega_{2n}^i)^{2n-1} \\ &\stackrel{(b)}{=} \left(a_0 + a_2 (\omega_{2n}^i)^2 + \dots + a_{2n-2} (\omega_{2n}^i)^{(2n-2)} \right) \\ &\quad + \left(a_1 \omega_{2n}^i + a_3 (\omega_{2n}^i)^3 + \dots + a_{2n-1} (\omega_{2n}^i)^{(2n-1)} \right). \end{aligned} \tag{11.5}$$

Here (a) is by definition of F_{2n} , and (b) groups the sum into one sum for the even-index coordinates, and another sum for the odd-index coordinates.

Consider the first group with the even-index coefficients. All the powers of ω_{2n} are even, so by applying the identity (11.4), we have

$$a_0 + a_2(\omega_{2n}^i)^2 + \cdots + a_{2n-2}(\omega_{2n}^i)^{(2n-2)} = a_0 + a_2\omega_n^i + \cdots + a_{2n-2}(\omega_n^i)^{n-1}.$$

Now the RHS can be interpreted as evaluating a degree $n-1$ polynomial with coefficients $a_0, a_2, \dots, a_{2n-2}$ at ω_n^i . More specifically, if we let $a_{\text{even}} = (a_0, a_2, \dots, a_{2n-2})$, then we have

$$a_0 + a_2\omega_n^i + \cdots + a_{2n-2}(\omega_n^i)^{n-1} = F_{a_{\text{even}}}$$

– the n th Fourier transform of the even-index coefficients!

We can do something similar for the odd coefficients. Let $a_{\text{odd}} = (a_1, a_3, \dots, a_{n-1})$ be the vector of odd coefficients, and consider the second sum in (11.5). We have

$$\begin{aligned} & a_1\omega_{2n}^i + a_3(\omega_{2n}^i)^3 + \cdots + a_{2n-1}(\omega_{2n}^i)^{(2n-1)} \\ &= \omega_{2n}^i \left(a_1 + a_3(\omega_n^i)^2 + a_5(\omega_n^i)^4 + \cdots + a_{2n-2}(\omega_n^i)^{n-1} \right) \\ &= \omega_{2n}^i \left(a_1 + a_3(\omega_n^i)^2 + a_5(\omega_n^i)^4 + \cdots + a_{2n-2}(\omega_n^i)^{n-1} \right) \\ &= \omega_{2n}^i (F_n a_{\text{odd}}). \end{aligned}$$

Plugging all this back into (11.5), we have the clean identity

$$(F_{2n}a)_i = (F_n a_{\text{even}})_i + \omega_{2n}^i (F_n a_{\text{odd}})_i$$

for $i = 0, \dots, n-1$. This equation tells us that computing $F_{2n}a$ is largely reduced to computing $F_n a_{\text{even}}$ and $F_n a_{\text{odd}}$. More specifically, given $F_n a_{\text{even}}$ and $F_n a_{\text{odd}}$, we need only $O(2n)$ time to compute $F_{2n}a$. Meanwhile $F_n a_{\text{even}}$ and $F_n a_{\text{odd}}$ can be computed recursively, where each subproblem has half the size / dimension.

In fig. 11.9 on page 177, we give pseudocode for the recursive algorithm $\text{Fourier}(a_0, \dots, a_{n-1})$, based on the divide-and-conquer approach outlined above. The code in fig. 11.9 is modified slightly for mathematics above to remove the assumption that n is even. (The argument for odd n is the exact same, except there is one more even-index coefficient than odd-index coefficient.)

We now analyze the running time of $\text{Fourier}(a_0, \dots, a_{n-1})$. As annotated in the pseudocode, $\text{Fourier}(a_0, \dots, a_{n-1})$ consists of 2 recursive calls with half the number of coefficients, plus $O(n)$ work. Thus the running time obeys the recurrence

$$\begin{aligned} T(1) &= O(1) \\ T(n) &= 2T(n/2) + O(n). \end{aligned}$$


```

Fourier( $a_0, \dots, a_{n-1}$ )
/* We assume for simplicity that  $n$  is even. */
1. If  $n = 0$  then return the empty vector.
2. If  $n = 1$  then return  $a_0$ .
3. Let  $a_{\text{even}} = (a_0, a_2, a_4, \dots)$  be the vector of  $\lceil n/2 \rceil$  even-index coefficients and let
    $a_{\text{odd}} = (a_1, a_3, a_5, \dots)$  be the vector of  $\lfloor n/2 \rfloor$  odd-index coefficients. //  $O(n)$ 
4. Let  $(y_0, \dots, y_{\lceil n/2 \rceil - 1}) = \text{Fourier}(a_{\text{even}})$  //  $T(\lceil n/2 \rceil)$ 
5. Let  $(z_0, \dots, z_{\lfloor n/2 \rfloor - 1}) = \text{Fourier}(a_{\text{odd}})$ . //  $T(\lfloor n/2 \rfloor)$ 
6. For  $i = 0, 1, \dots, n - 1$ , let  $x_i = y_i + \omega_{2n}^i z_i$  //  $O(n)$ 
7. Return  $(x_0, \dots, x_{n-1})$ .
    
```

Figure 11.9: The fast Fourier transform.

This is the same recurrence as in merge-sort (cf. section 1.2.4) and is solved by

$$T(n) = O(n \log n).$$

This completes the proof of theorem 11.5.

11.5 Matrix multiplication

Let A and B be two $n \times n$ matrices. We can compute the product AB in $O(n^3)$ based on the formula

$$(AB)_{ij} = \sum_k A_{ik} B_{kj}.$$

In 1969, Strassen [Str69] showed that one can compute AB faster by leveraging divide-and-conquer, which was very surprising. Write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{ and } B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$