

# Closest Pairs

COMP 3711H - HKUST  
Version of 27/11/2014  
M. J. Golin

# Outline

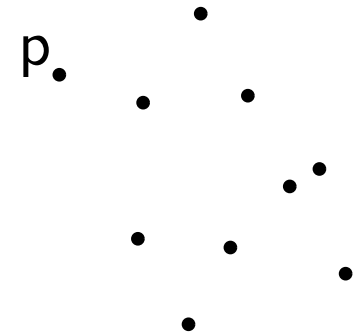
- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

# Introduction

- Most algorithms seen so far have assumed that items have 1-dimensional weights or keys, i.e., they can be ordered.
- There is often a need, e.g., graphics, data bases, to manipulate multi-dimensional data. The area of algorithms that treats geometric (multi-dimensional) data is called **computational geometry**.
- In this set of slides, we will see two algorithms for solving the 2-dimensional **closest-pair** problem.
- The first algorithm will take  $O(n \log n)$  worst case time to find the closest pair among  $n$  points. The second algorithm will be randomized and only require  $O(n)$  average time.

# Problem Definitions

- Problem input is  $P = \{p_1, p_2, \dots, p_n\}$ , a set of  $n$  2-dimensional points, where  $p = (p.x, p.y)$



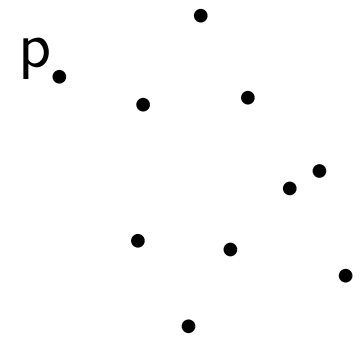
# Problem Definitions

- Problem input is  $P = \{p_1, p_2, \dots, p_n\}$ , a set of  $n$  2-dimensional points, where  $p = (p.x, p.y)$

- distance between  $p$  and  $p'$  is

$$d(p, p') = \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2},$$

- Set  $d(p, P) = \min_{p' \in P - \{p\}} d(p, p')$  to be distance of nearest point to  $p$ .



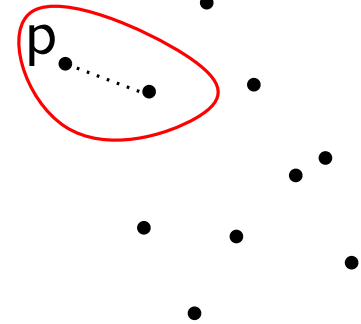
# Problem Definitions

- Problem input is  $P = \{p_1, p_2, \dots, p_n\}$ , a set of  $n$  2-dimensional points, where  $p = (p.x, p.y)$

- distance between  $p$  and  $p'$  is

$$d(p, p') = \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2},$$

- Set  $d(p, P) = \min_{p' \in P - \{p\}} d(p, p')$  to be distance of nearest point to  $p$ .



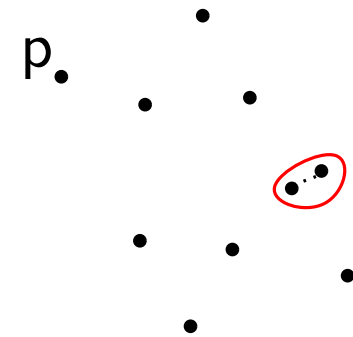
# Problem Definitions

- Problem input is  $P = \{p_1, p_2, \dots, p_n\}$ , a set of  $n$  2-dimensional points, where  $p = (p.x, p.y)$

- distance between  $p$  and  $p'$  is

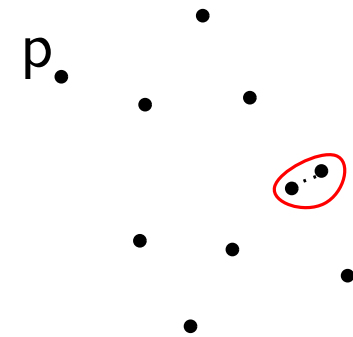
$$d(p, p') = \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2},$$

- Set  $d(p, P) = \min_{p' \in P - \{p\}} d(p, p')$  to be distance of nearest point to  $p$ .
- Set  $\delta(P) = \min_{p \in P} d(p, P)$  to be the *closest pair distance* in  $P$ .



# Problem Definitions

- Problem input is  $P = \{p_1, p_2, \dots, p_n\}$ , a set of  $n$  2-dimensional points, where  $p = (p.x, p.y)$



- distance between  $p$  and  $p'$  is
$$d(p, p') = \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2},$$
- Set  $d(p, P) = \min_{p' \in P - \{p\}} d(p, p')$  to be distance of nearest point to  $p$ .
- Set  $\delta(P) = \min_{p \in P} d(p, P)$  to be the *closest pair distance* in  $P$ .
- The **Closest-Pair Problem** is to find  $p, p' \in P$  such that  $d(p, p') = \delta(P)$



# Outline

- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

# The Gridding Lemma

## The Gridding Lemma:

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points and  $\delta = \delta(P)$  their closest-pair distance.

Grid the plane into  $\frac{\delta}{2} \times \frac{\delta}{2}$  boxes using horizontal and vertical lines. Then

(a) No box holds more than 1 point

(b) If  $d(p, p') = \delta$  then the boxes containing  $p$  and  $p'$

are within two horizontal and two vertical boxes of each other.

# The Gridding Lemma

## The Gridding Lemma:

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points and  $\delta = \delta(P)$  their closest-pair distance.

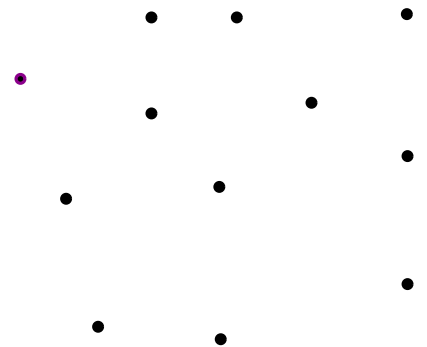
Grid the plane into  $\frac{\delta}{2} \times \frac{\delta}{2}$  boxes using horizontal and vertical lines. Then

(a) No box holds more than 1 point

(b) If  $d(p, p') = \delta$  then the boxes containing  $p$  and  $p'$

are within two horizontal and two vertical boxes of each other.

---



# The Gridding Lemma

## The Gridding Lemma:

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points and  $\delta = \delta(P)$  their closest-pair distance.

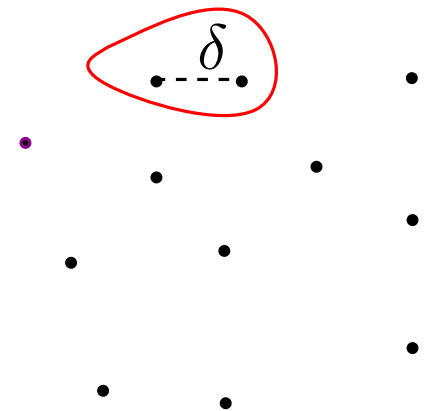
Grid the plane into  $\frac{\delta}{2} \times \frac{\delta}{2}$  boxes using horizontal and vertical lines. Then

(a) No box holds more than 1 point

(b) If  $d(p, p') = \delta$  then the boxes containing  $p$  and  $p'$

are within two horizontal and two vertical boxes of each other.

---



# The Gridding Lemma

## The Gridding Lemma:

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points and  $\delta = \delta(P)$  their closest-pair distance.

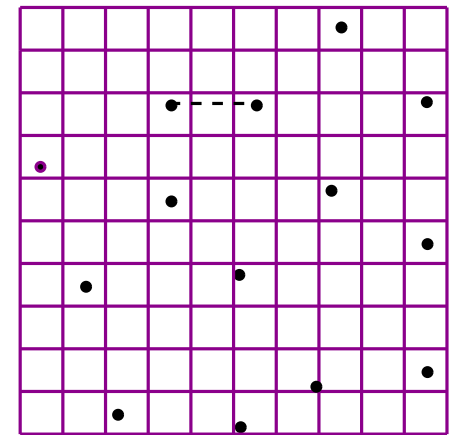
Grid the plane into  $\frac{\delta}{2} \times \frac{\delta}{2}$  boxes using horizontal and vertical lines. Then

(a) No box holds more than 1 point

(b) If  $d(p, p') = \delta$  then the boxes containing  $p$  and  $p'$

are within two horizontal and two vertical boxes of each other.

---



# The Gridding Lemma

## The Gridding Lemma:

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of points and

$\delta = \delta(P)$  their closest-pair distance.

Grid the plane into  $\frac{\delta}{2} \times \frac{\delta}{2}$  boxes using horizontal and vertical lines. Then

(a) No box holds more than 1 point

(b) If  $d(p, p') = \delta$  then the boxes containing  $p$  and  $p'$

are within two horizontal and two vertical boxes of each other.

---

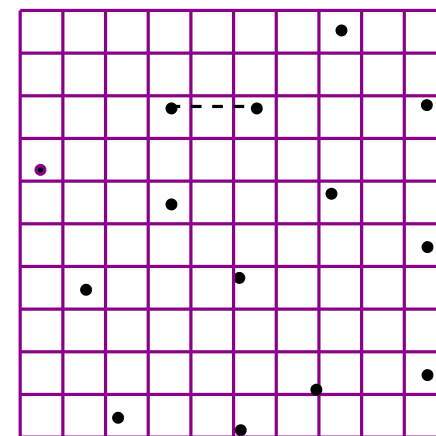
## Proof:

(a) if two points are in same box they can be at most  $\delta/\sqrt{2} < \delta$  away from each other, contradicting def of  $\delta$ .

(b) if two points  $p, p'$  are more than two boxes horizontally (or vertically) away from each other then

$|p.x - p'.x| > \delta$  ( $|p.y - p'.y| > \delta$ ), so  $d(p, p') > \delta$

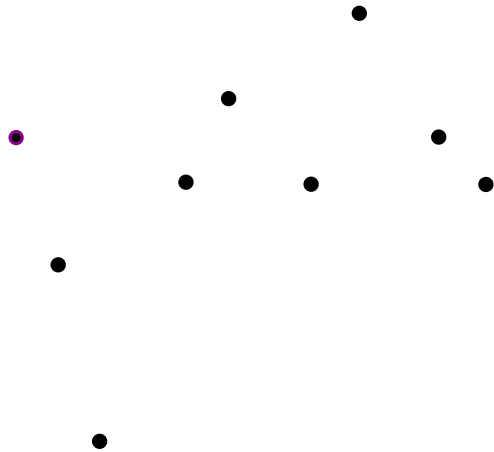
contradicting definition of  $\delta$ .



# Outline

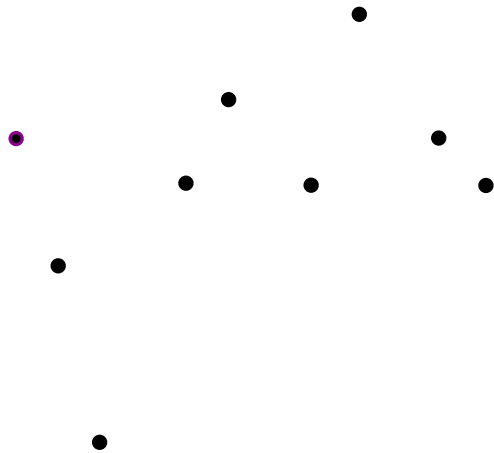
- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

# A Sweep Algorithm



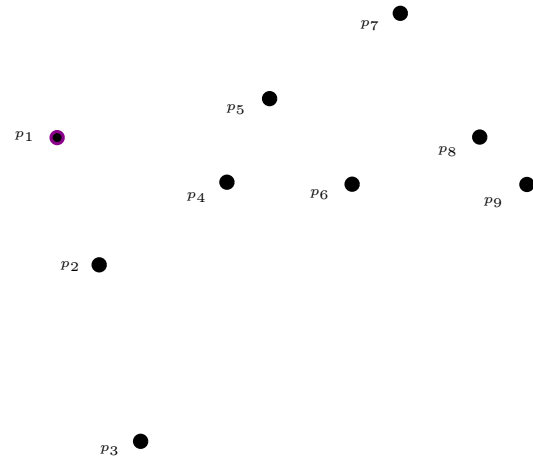


# A Sweep Algorithm



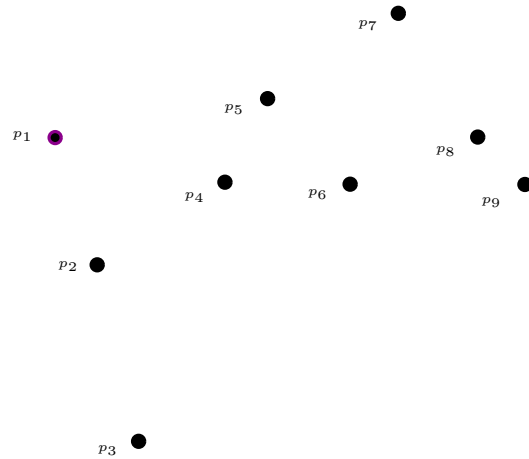
Sort the points by  $x$  coordinate.

# A Sweep Algorithm



Sort the points by  $x$  coordinate.

# A Sweep Algorithm

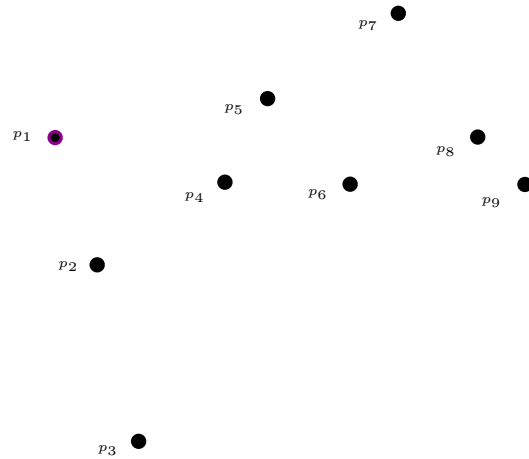


Sort the points by  $x$  coordinate.

Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

# A Sweep Algorithm



Sort the points by  $x$  coordinate.

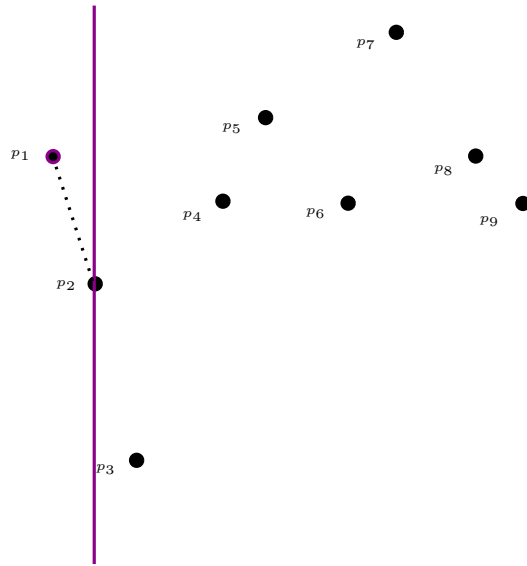
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



Sort the points by  $x$  coordinate.

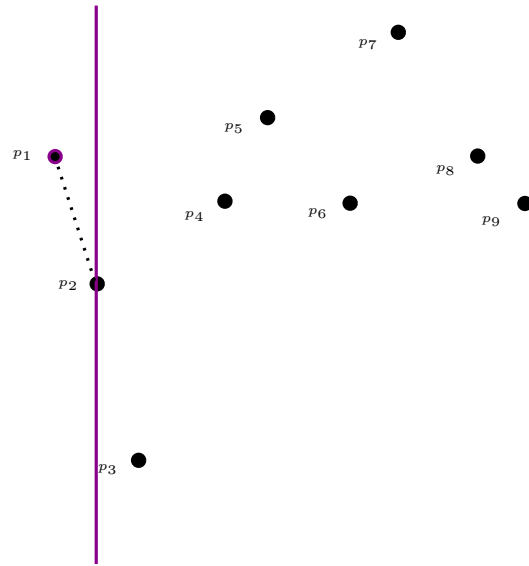
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

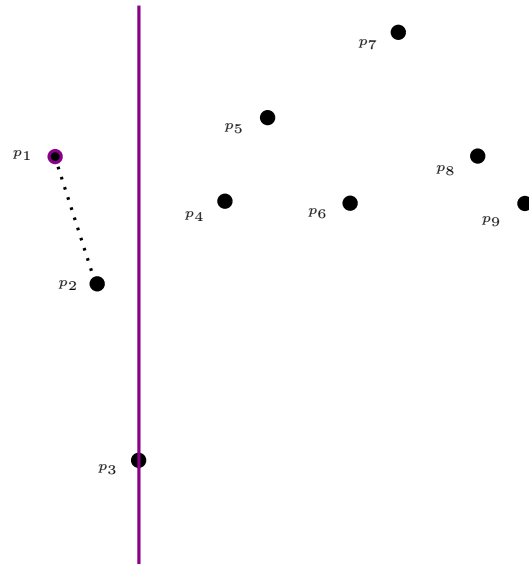
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

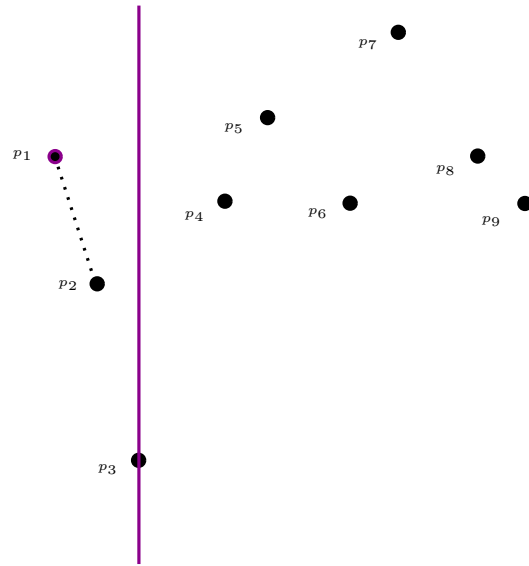
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

Set  $P_i = \{p_1, \dots, p_i\}$ .

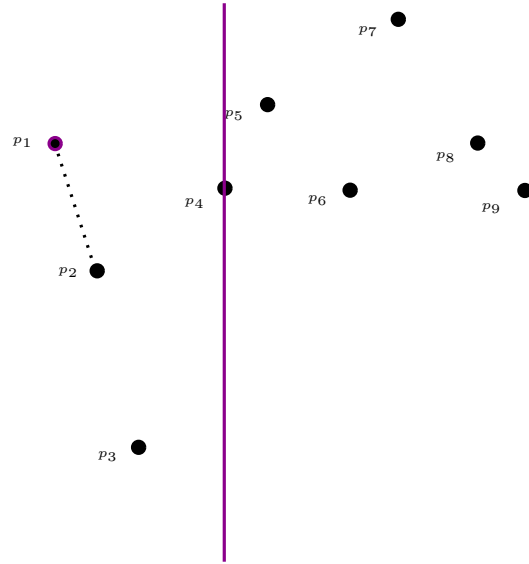
Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$



# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

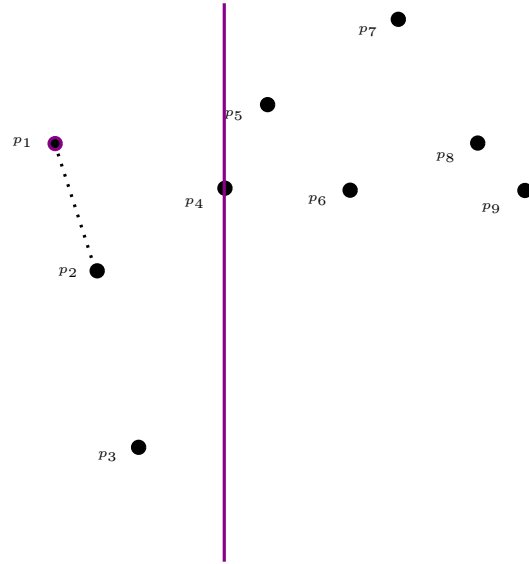
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

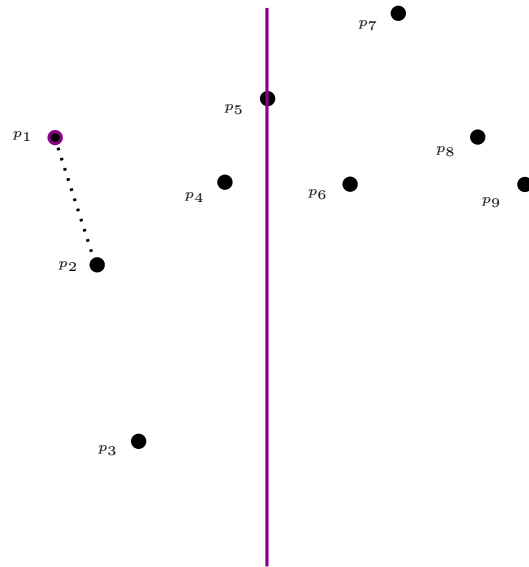
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

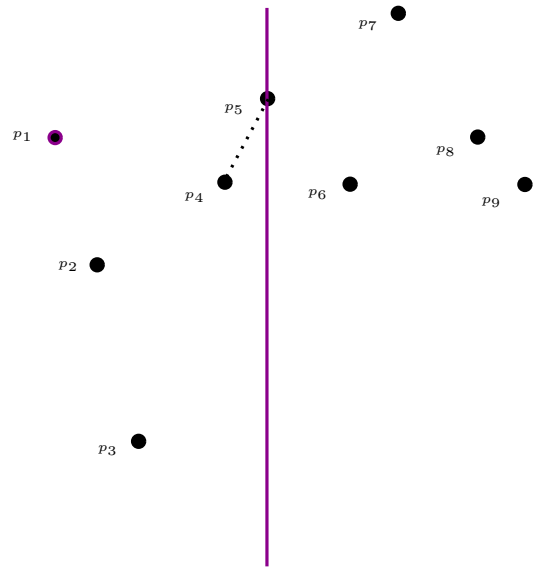
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

Sort the points by  $x$  coordinate.

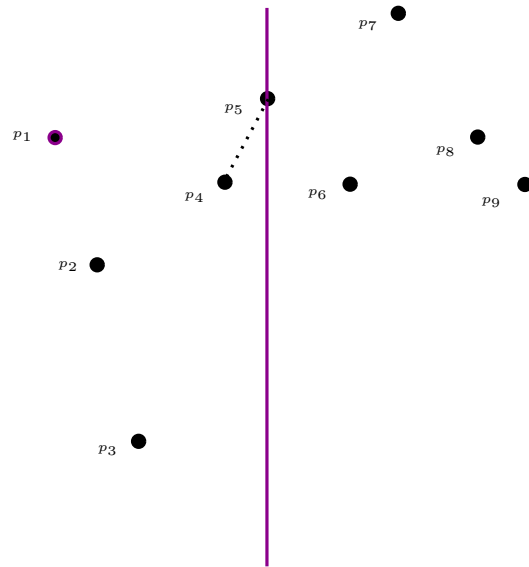
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

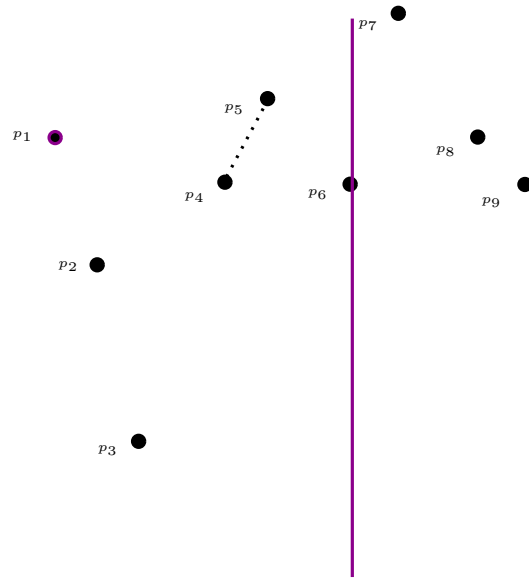
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

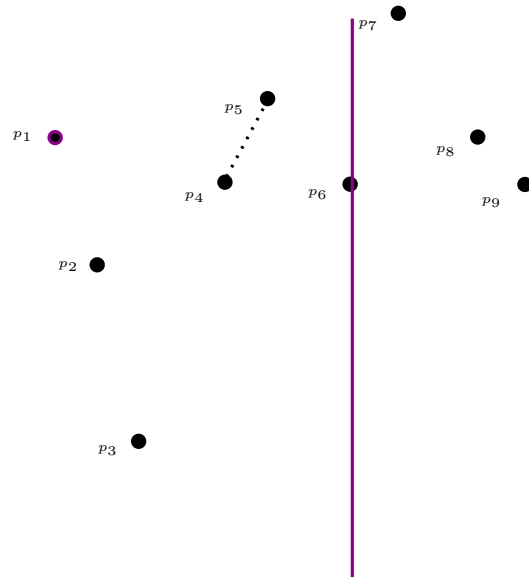
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

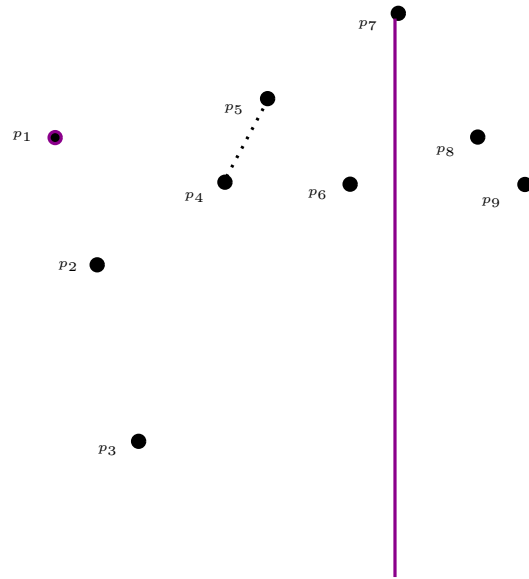
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

Set  $P_i = \{p_1, \dots, p_i\}$ .

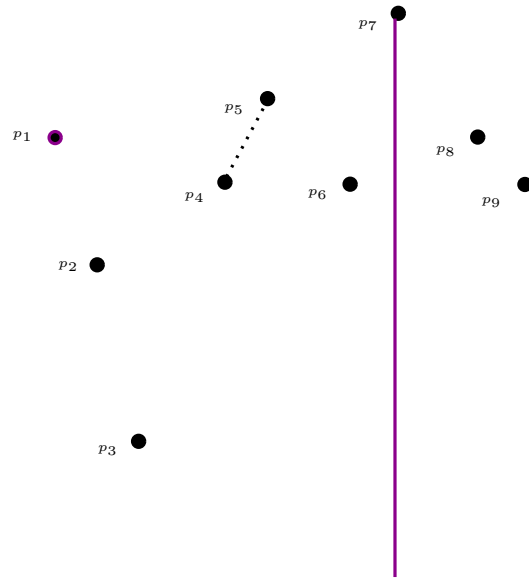
Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$



# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

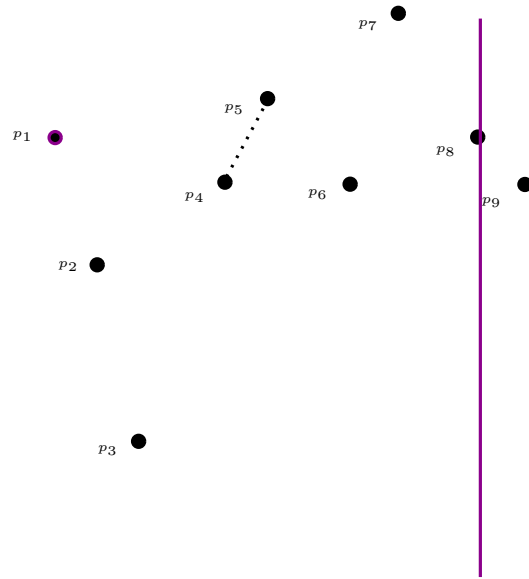
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

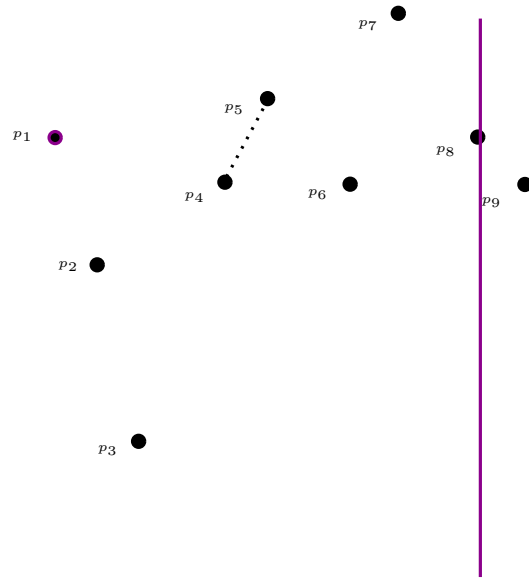
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

$$\delta(P_8) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

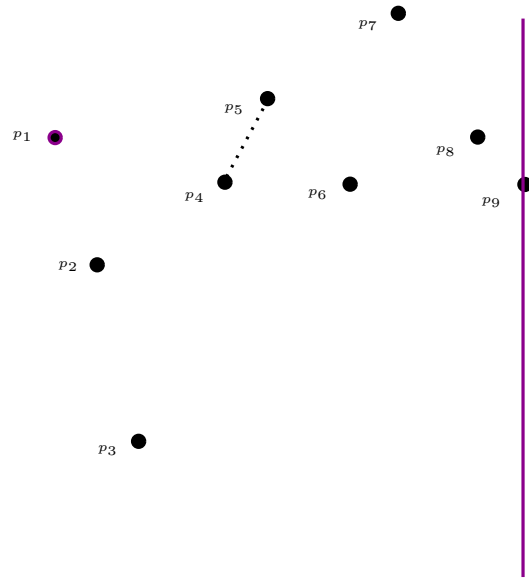
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

$$\delta(P_8) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

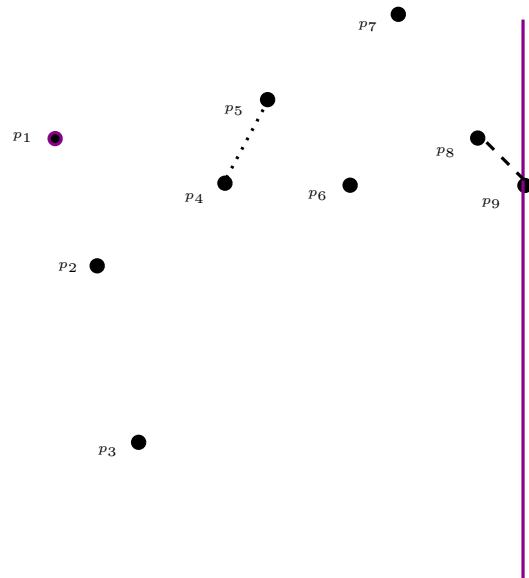
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

$$\delta(P_8) = d(p_5, p_4)$$

Sort the points by  $x$  coordinate.

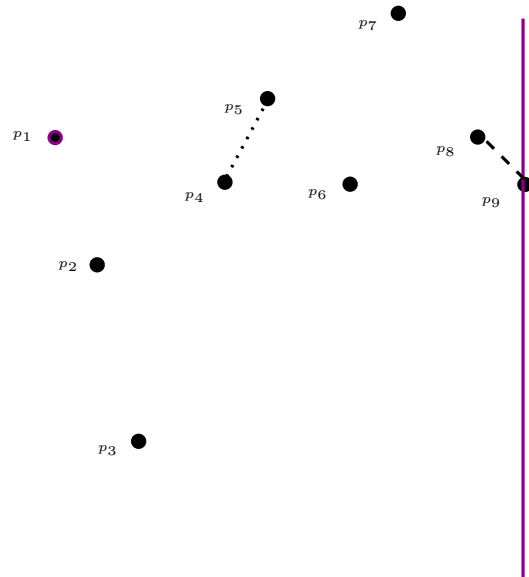
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

$$\delta(P_8) = d(p_5, p_4)$$

$$\delta(P_9) = d(p_9, p_8)$$

Sort the points by  $x$  coordinate.

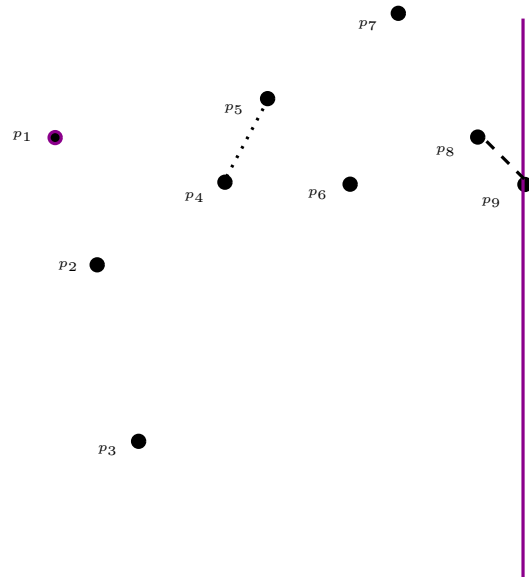
Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

# A Sweep Algorithm



Sort the points by  $x$  coordinate.

Set  $P_i = \{p_1, \dots, p_i\}$ .

Note  $\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$

Algorithm sweeps a vertical line through points, i.e.,  $p_2, p_3, \dots, p_n$ , at each step using equation to calculate  $\delta(P_i)$ .

At end,  $\delta = \delta(P_n)$

$$\delta(P_2) = d(p_2, p_1)$$

$$\delta(P_3) = d(p_2, p_1)$$

$$\delta(P_4) = d(p_2, p_1)$$

$$\delta(P_5) = d(p_5, p_4)$$

$$\delta(P_6) = d(p_5, p_4)$$

$$\delta(P_7) = d(p_5, p_4)$$

$$\delta(P_8) = d(p_5, p_4)$$

$$\delta(P_9) = d(p_9, p_8)$$

$$\delta = \delta(P_9) = d(p_9, p_8)$$

# Updating One Sweep Step

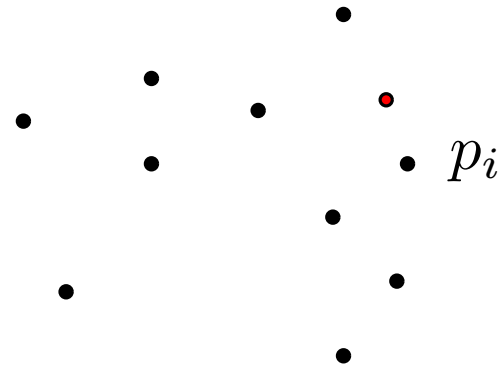
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$





# Updating One Sweep Step

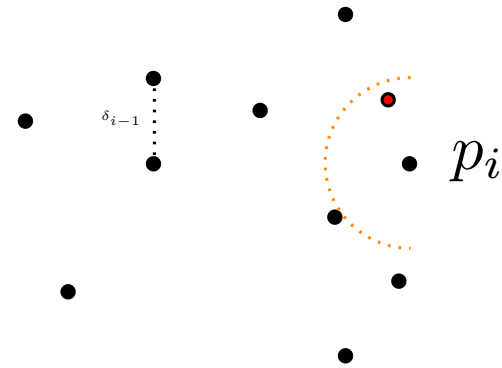
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



# Updating One Sweep Step

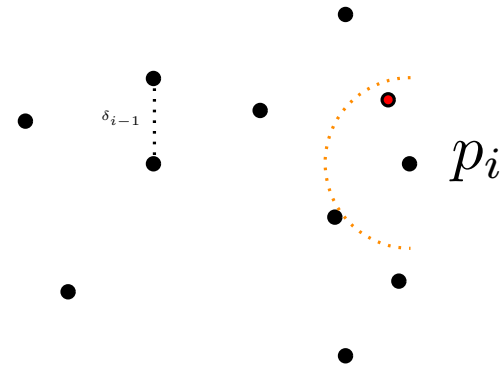
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



# Updating One Sweep Step

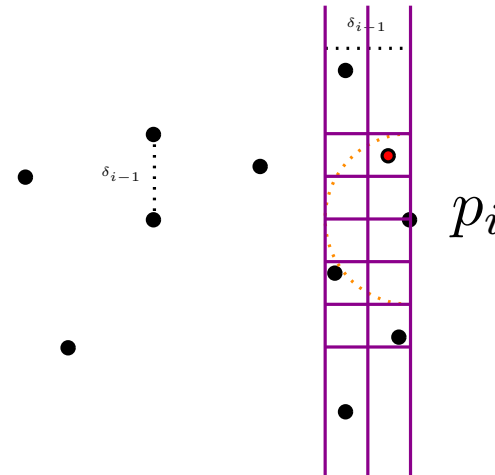
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



# Updating One Sweep Step

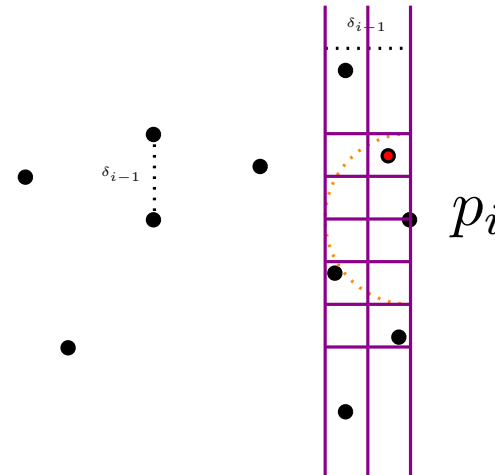
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



If we grid the plane with a  $\frac{\delta_{i-1}}{2} \times \frac{\delta_{i-1}}{2}$  grid with  $p_i$  as the origin, then  $p_j$  must be within 2 horizontal grid lines and 2 vertical grid lines of  $p_j$ .

# Updating One Sweep Step

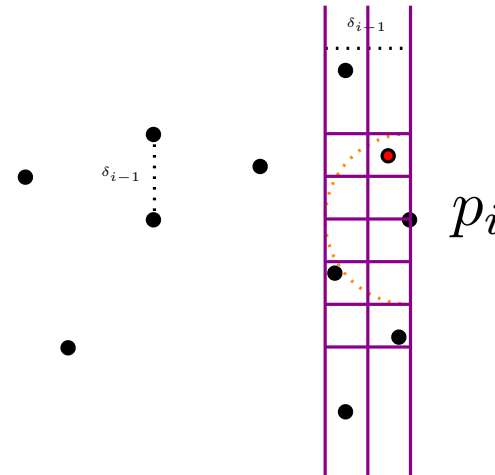
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



If we grid the plane with a  $\frac{\delta_{i-1}}{2} \times \frac{\delta_{i-1}}{2}$  grid with  $p_i$  as the origin, then  $p_j$  must be within 2 horizontal grid lines and 2 vertical grid lines of  $p_j$ .

# Updating One Sweep Step

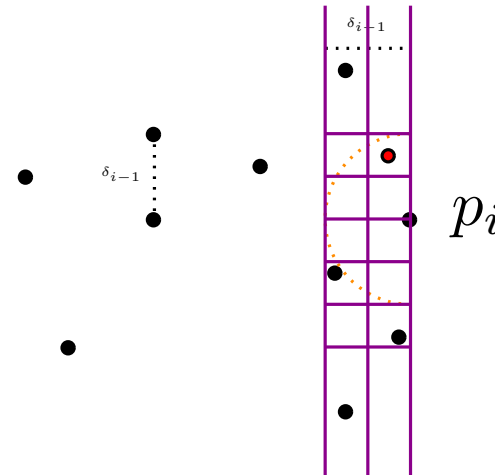
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



If we grid the plane with a  $\frac{\delta_{i-1}}{2} \times \frac{\delta_{i-1}}{2}$  grid with  $p_i$  as the origin, then  $p_j$  must be within 2 horizontal grid lines and 2 vertical grid lines of  $p_j$ .

Gridding Lemma tells us there is at most one point per box.

# Updating One Sweep Step

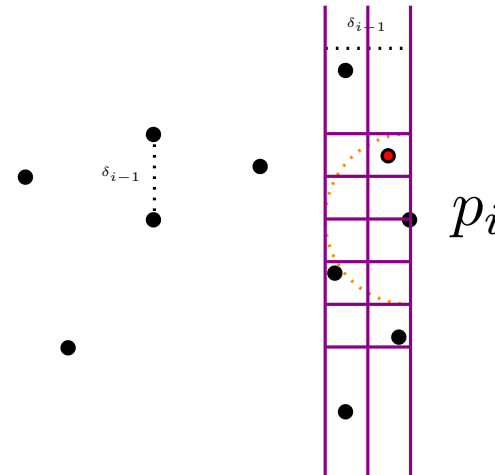
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



If we grid the plane with a  $\frac{\delta_{i-1}}{2} \times \frac{\delta_{i-1}}{2}$  grid with  $p_i$  as the origin, then  $p_j$  must be within 2 horizontal grid lines and 2 vertical grid lines of  $p_i$ .

Gridding Lemma tells us there is at most one point per box.

$$\text{Let } S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$$

# Updating One Sweep Step

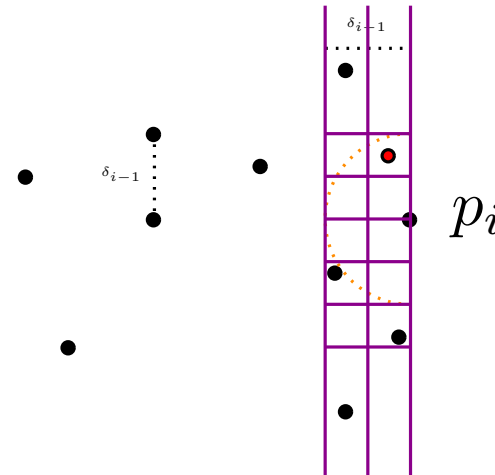
$$\delta(P_i) = \min(\delta(P_{i-1}), d(p_i, P_{i-1}))$$

Set  $\delta_i = \delta(P_i)$  and  $d_i = d(p_i, P_{i-1})$ .

If  $d_i < \delta_{i-1}$  then

$d_i = d(p_i, p_j)$  for some  $j < i$  with

$$|p_i.x - p_j.x| \leq d_i = \delta_i < \delta_{i-1}$$



If we grid the plane with a  $\frac{\delta_{i-1}}{2} \times \frac{\delta_{i-1}}{2}$  grid with  $p_i$  as the origin, then  $p_j$  must be within 2 horizontal grid lines and 2 vertical grid lines of  $p_j$ .

Gridding Lemma tells us there is at most one point per box.

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

This implies

if  $d_i < \delta_{i-1}$  then

$d_i = (p_i, p)$  where

(a)  $p \in S_i$  and

(b)  $p$  is one of the 4 points above  $p_i$  in  $S_i$

or

$p$  is one of the 4 points below  $p_i$  in  $S_i$ .



# Updating One Sweep Step (ii)

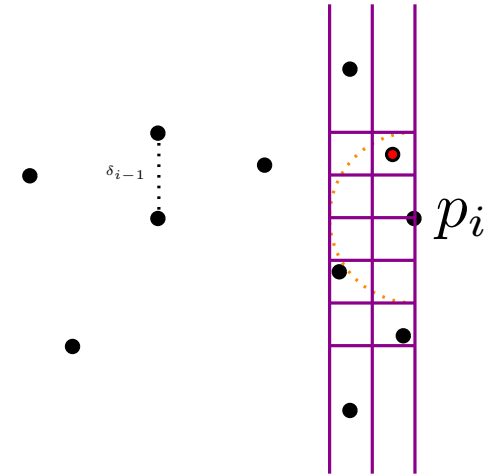
$$\delta_i = \min(\delta_{i-1}, d(p_i, P_{i-1}))$$

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

if  $d_i < \delta_{i-1}$  then  $d_i = (p_i, p)$  where

(a)  $p \in S_i$  and

(b)  $p$  is one of the 4 points above  $p_i$  in  $S_i$  or  
 $p$  is one of the 4 points below  $p_i$  in  $S_i$ .



# Updating One Sweep Step (ii)

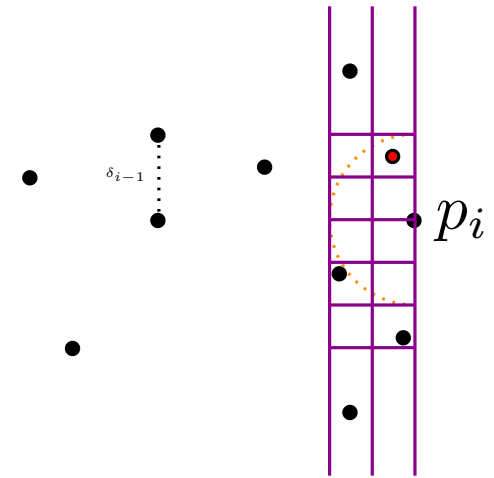
$$\delta_i = \min(\delta_{i-1}, d(p_i, P_{i-1}))$$

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

if  $d_i < \delta_{i-1}$  then  $d_i = (p_i, p)$  where

(a)  $p \in S_i$  and

(b)  $p$  is one of the 4 points above  $p_i$  in  $S_i$  or  
 $p$  is one of the 4 points below  $p_i$  in  $S_i$ .



Suppose we knew  $\delta_{i-1}$  and had the points in  $S_i$  kept in a balanced binary search tree ordered by increasing  $y$  coordinate.

# Updating One Sweep Step (ii)

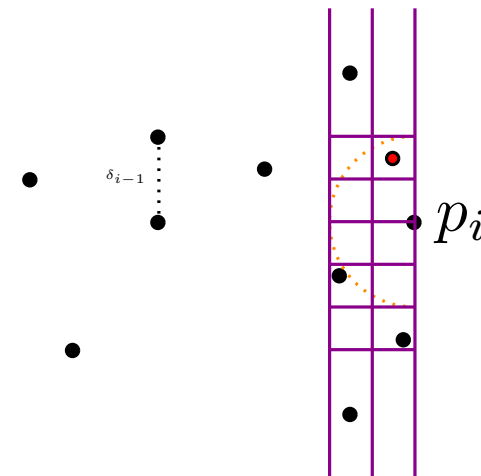
$$\delta_i = \min(\delta_{i-1}, d(p_i, P_{i-1}))$$

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

if  $d_i < \delta_{i-1}$  then  $d_i = (p_i, p)$  where

(a)  $p \in S_i$  and

(b)  $p$  is one of the 4 points above  $p_i$  in  $S_i$  or  
 $p$  is one of the 4 points below  $p_i$  in  $S_i$ .



Suppose we knew  $\delta_{i-1}$  and had the points in  $S_i$  kept in a balanced binary search tree ordered by increasing  $y$  coordinate.

We could then calculate  $\delta_i$  in  $O(\log n)$  time as follows

- In  $S_i$ , find the 4 points above  $p_i$  and four points below it  
This takes  $O(\log n)$  time
- Find the distances between these (at most) 8 points and  $p_i$ .  
This takes  $O(1)$  time. Call this minimum distance  $d'$   
(note that there might be no points in  $S_i$ . In this case  $d' = \infty$ )
- Set  $\delta_i = \min(\delta_{i-1}, d')$

# The Full Sweep Algorithm

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

We just developed the following algorithm to calculate  $\delta_2, \delta_3, \dots, \delta_n$

- A Set  $\delta_2 = d(p_1, p_2)$ ;  
Create a  $y$ -sorted balanced BST on  $S_2 = \{p_1\}$
- B For  $i = 3$  to  $n$ 
  1. Update the BST from storing  $S_{i-1}$  to storing  $S_i$
  2. In  $O(\log n)$  time use the BST storing  $S_i$  to calculate  $\delta_i$

# The Full Sweep Algorithm

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

We just developed the following algorithm to calculate  $\delta_2, \delta_3, \dots, \delta_n$

- A Set  $\delta_2 = d(p_1, p_2)$ ;  
Create a  $y$ -sorted balanced BST on  $S_2 = \{p_1\}$
- B For  $i = 3$  to  $n$ 
  1. Update the BST from storing  $S_{i-1}$  to storing  $S_i$
  2. In  $O(\log n)$  time use the BST storing  $S_i$  to calculate  $\delta_i$

Step (A) uses  $O(1)$  time. Since step (B)(2) uses  $O(\log n)$  time per step and it's called  $O(n)$  times, all the calls to it use only  $O(n \log n)$  time.

# The Full Sweep Algorithm

Let  $S_i = \{p \in P_{i-1} : p_i.x - p.x < \delta_{i-1}\}$

We just developed the following algorithm to calculate  $\delta_2, \delta_3, \dots, \delta_n$

- A Set  $\delta_2 = d(p_1, p_2)$ ;  
Create a  $y$ -sorted balanced BST on  $S_2 = \{p_1\}$
- B For  $i = 3$  to  $n$ 
  1. Update the BST from storing  $S_{i-1}$  to storing  $S_i$
  2. In  $O(\log n)$  time use the BST storing  $S_i$  to calculate  $\delta_i$

Step (A) uses  $O(1)$  time. Since step (B)(2) uses  $O(\log n)$  time per step and it's called  $O(n)$  times, all the calls to it use only  $O(n \log n)$  time.

If we can implement step (B)(1) in  $O(n \log n)$  total time the entire algorithm then is  $O(n \log n)$ .

# The Final Piece of Analysis

**The Problem:** Given  $p_1, p_2, \dots, p_n$  sorted by nondecreasing  $x$ -coordinate and a sequence  $\delta_2 \geq \delta_3 \geq \dots \geq \delta_{n-1}$ , iteratively construct balanced BSTs storing the sets  $S_2, S_3, \dots, S_n$  sorted by  $y$ -coordinate. Use only  $O(n \log n)$  time in total.

# The Final Piece of Analysis

**The Problem:** Given  $p_1, p_2, \dots, p_n$  sorted by nondecreasing  $x$ -coordinate and a sequence  $\delta_2 \geq \delta_3 \geq \dots \geq \delta_{n-1}$ , iteratively construct balanced BSTs storing the sets  $S_2, S_3, \dots, S_n$  sorted by  $y$ -coordinate. Use only  $O(n \log n)$  time in total.

---

For  $k < i$ , if  $p_k \notin S_i$  then  $p_{k-1} \notin S_i$  because

$$p_i.x - p_{k-1}.x \geq p_i.x - p_k.x \geq \delta_{i-1}$$

so  $S_i$  is a continuous sequence of points  $p_k, p_{k+1}, \dots, p_{i-1}$ .

Furthermore, if  $p_k \notin S_{i-1}$  then  $p_k \notin S_i$ , because

$$p_i.x - p_k.x \geq p_{i-1}.x - p_k.x > \delta_{i-2} \geq \delta_{i-1}$$



# The Final Piece of Analysis

**The Problem:** Given  $p_1, p_2, \dots, p_n$  sorted by nondecreasing  $x$ -coordinate and a sequence  $\delta_2 \geq \delta_3 \geq \dots \geq \delta_{n-1}$ , iteratively construct balanced BSTs storing the sets  $S_2, S_3, \dots, S_n$  sorted by  $y$ -coordinate. Use only  $O(n \log n)$  time in total.

---

For  $k < i$ , if  $p_k \notin S_i$  then  $p_{k-1} \notin S_i$  because

$$p_i.x - p_{k-1}.x \geq p_i.x - p_k.x \geq \delta_{i-1}$$

so  $S_i$  is a continuous sequence of points  $p_k, p_{k+1}, \dots, p_{i-1}$ .

Furthermore, if  $p_k \notin S_{i-1}$  then  $p_k \notin S_i$ , because

$$p_i.x - p_k.x \geq p_{i-1}.x - p_k.x > \delta_{i-2} \geq \delta_{i-1}$$

---

$\Rightarrow S_{i-1}$  can be updated to  $S_i$  by (a) adding  $p_{i-1}$  to the BST and then (b) walking rightwards from the leftmost point in  $S_{i-1}$ , removing points from the BST, until finding the new leftmost point in  $S_i$ .

# The Final Piece of Analysis

**The Problem:** Given  $p_1, p_2, \dots, p_n$  sorted by nondecreasing  $x$ -coordinate and a sequence  $\delta_2 \geq \delta_3 \geq \dots \geq \delta_{n-1}$ , iteratively construct balanced BSTs storing the sets  $S_2, S_3, \dots, S_n$  sorted by  $y$ -coordinate. Use only  $O(n \log n)$  time in total.

---

For  $k < i$ , if  $p_k \notin S_i$  then  $p_{k-1} \notin S_i$  because

$$p_i.x - p_{k-1}.x \geq p_i.x - p_k.x \geq \delta_{i-1}$$

so  $S_i$  is a continuous sequence of points  $p_k, p_{k+1}, \dots, p_{i-1}$ .

Furthermore, if  $p_k \notin S_{i-1}$  then  $p_k \notin S_i$ , because

$$p_i.x - p_k.x \geq p_{i-1}.x - p_k.x > \delta_{i-2} \geq \delta_{i-1}$$

---

$\Rightarrow S_{i-1}$  can be updated to  $S_i$  by (a) adding  $p_{i-1}$  to the BST and then (b) walking rightwards from the leftmost point in  $S_{i-1}$ , removing points from the BST, until finding the new leftmost point in  $S_i$ .

---

Every point is added once and removed at most once, so we perform  $O(n)$  inserts and  $O(n)$  deletes from the BST, requiring only  $O(n \log n)$  time

## Odds & Ends

- This approach of *sorting the points by  $x$  then updating solution by moving from left to right* is a common one in computational geometry. It is called the **sweep-line technique**.
- It is possible to prove that  $O(n \log n)$  is a lower bound for solving this problem in the **algebraic decision tree** model of computation (a generalization of the comparison tree model we used for sorting). So, our algorithm is **optimal** in this model.

## Odds & Ends

- This approach of *sorting the points by  $x$  then updating solution by moving from left to right* is a common one in computational geometry. It is called the **sweep-line technique**.
- It is possible to prove that  $O(n \log n)$  is a lower bound for solving this problem in the **algebraic decision tree** model of computation (a generalization of the comparison tree model we used for sorting). So, our algorithm is **optimal** in this model.
- With randomization and the use of the floor function ( $\lfloor x \rfloor$ ) this can be improved to an  $O(n)$  average case time randomized algorithm.

# Outline

- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

# A Randomized Algorithm

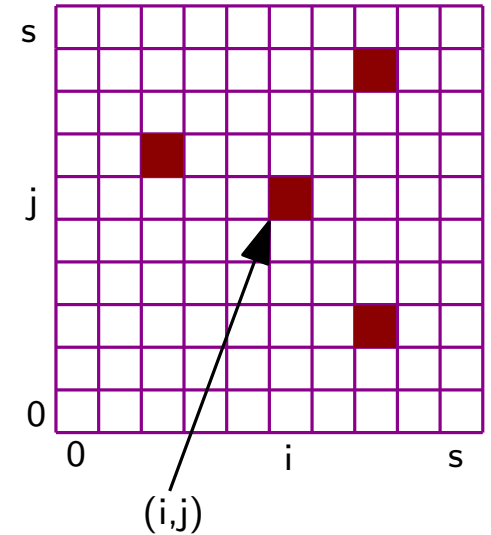
- In this section we will see a randomized algorithm for finding the closest pair in  $O(n)$  average case time.
- Input is  $n$  two-dimensional points.
- The randomization will come from two places
  - The first step of the algorithm will be to randomly order the input. That is, it will choose one of the  $n!$  possible orderings of the  $n$  points uniformly at random and label the points as  $p_1, p_2, \dots, p_n$  in that order
  - The only other place randomization is used will be in the use of (universal) hashing
- Before starting the main part of the algorithm we perform an  $O(n)$  scan finding  $x_{\min}$ ,  $x_{\max}$ ,  $y_{\min}$ ,  $y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points.

# Outline

- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

# Hashing a Sparse Array

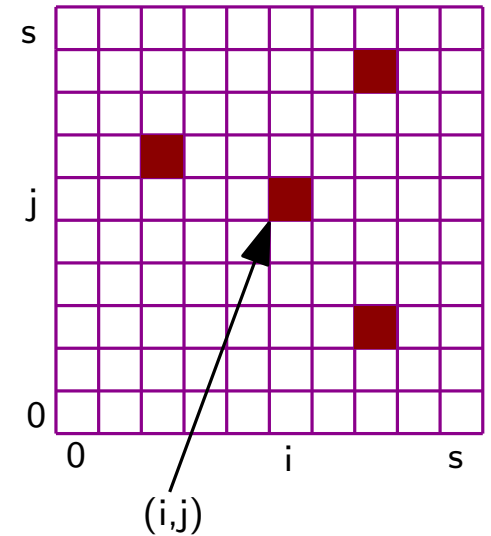
- Given an  $s \times s$  array in which at most  $n$  items exist (are non-zero).
- We would like to be able to perform the following two operations in  $O(1)$  time each.
  - Insert( $x$ ): Insert new item  $x$  into the array
  - Search( $x$ ): If  $x$  exists, return it (else return *nil*)
- In practice, array item is a pointer to a record.  
Sucessful search returns pointer to record's data.





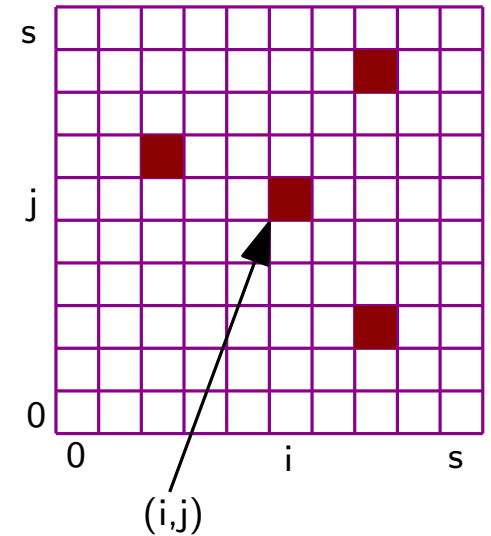
# Hashing a Sparse Array

- Given an  $s \times s$  array in which at most  $n$  items exist (are non-zero).
- We would like to be able to perform the following two operations in  $O(1)$  time each.
  - Insert( $x$ ): Insert new item  $x$  into the array
  - Search( $x$ ): If  $x$  exists, return it (else return *nil*)
- In practice, array item is a pointer to a record. Successful search returns pointer to record's data.
- This can be solved using hashing
- Array index is  $(i, j)$  with  $0 \leq i, j \leq s$ .  
Can map array indices uniquely to integer  $i + js < s^2 + 1$



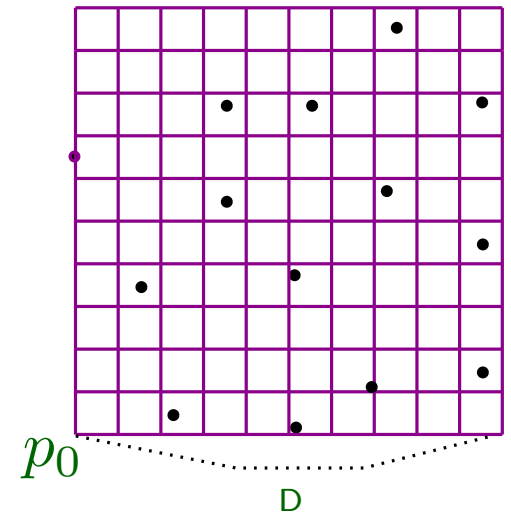
# Hashing a Sparse Array

- Given an  $s \times s$  array in which at most  $n$  items exist (are non-zero).
- We would like to be able to perform the following two operations in  $O(1)$  time each.
  - Insert( $x$ ): Insert new item  $x$  into the array
  - Search( $x$ ): If  $x$  exists, return it (else return *nil*)
- In practice, array item is a pointer to a record. Successful search returns pointer to record's data.
- This can be solved using hashing
- Array index is  $(i, j)$  with  $0 \leq i, j \leq s$ .  
Can map array indices uniquely to integer  $i + js < s^2 + 1$
- Applying **universal hashing** with  $U = s^2 + 1$  and  $m = n$  gives  $O(1)$  average case insert and search



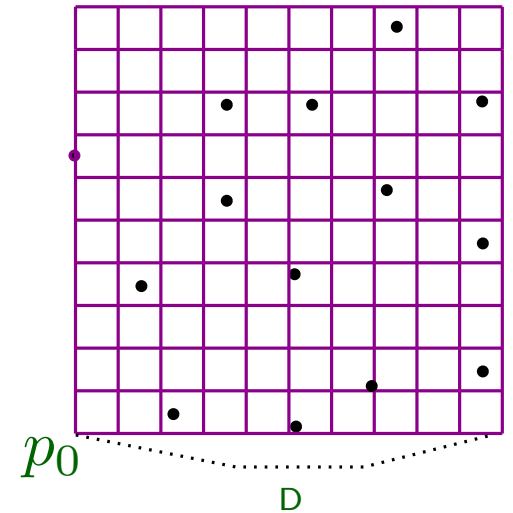
# Bucketing: Hashing a Grid

- Let  $p_0 = (x_{\min}, y_{\min})$ ;  
 $D = \max(x_{\max} - x_{\min}, y_{\max} - y_{\min})$ .
- The entire point set is contained in the square of side  $D$  with lower-left corner  $p_0$ .
- A *grid of size  $d$*  will denote a horizontal/vertical grid with grid separation  $d$  between adjacent parallel lines and  $p_0$  as the origin.



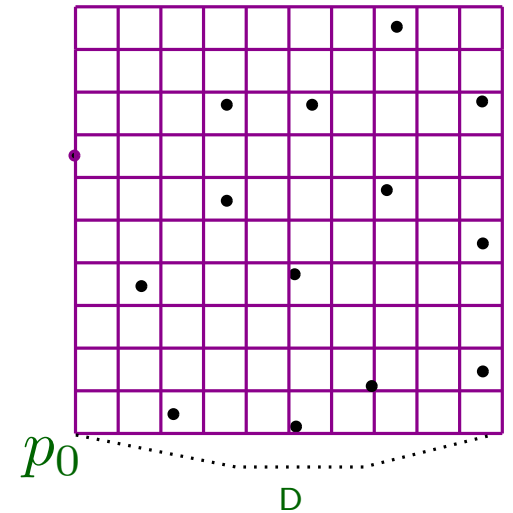
# Bucketing: Hashing a Grid

- Let  $p_0 = (x_{\min}, y_{\min})$ ;  
 $D = \max(x_{\max} - x_{\min}, y_{\max} - y_{\min})$ .
- The entire point set is contained in the square of side  $D$  with lower-left corner  $p_0$ .
- A *grid of size  $d$*  will denote a horizontal/vertical grid with grid separation  $d$  between adjacent parallel lines and  $p_0$  as the origin.
- The grid box labelled  $(X, Y)$  will contain all points  $(x, y)$  with  $X \leq x \leq X + d$  and  $Y \leq y \leq Y + d$   
(and appropriate tie breaking rule for points on the boundaries).



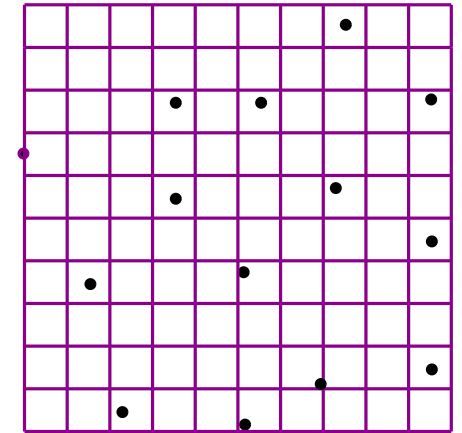
# Bucketing: Hashing a Grid

- Let  $p_0 = (x_{\min}, y_{\min})$ ;  
 $D = \max(x_{\max} - x_{\min}, y_{\max} - y_{\min})$ .
- The entire point set is contained in the square of side  $D$  with lower-left corner  $p_0$ .
- A *grid of size  $d$*  will denote a horizontal/vertical grid with grid separation  $d$  between adjacent parallel lines and  $p_0$  as the origin.
- The grid box labelled  $(X, Y)$  will contain all points  $(x, y)$  with  $X \leq x \leq X + d$  and  $Y \leq y \leq Y + d$  (and appropriate tie breaking rule for points on the boundaries).
- $(x, y)$  will be in the box  $(\lfloor \frac{x - x_{\min}}{d} \rfloor, \lfloor \frac{y - y_{\min}}{d} \rfloor)$
- Maximum box coordinate is  $\lceil D/d \rceil$



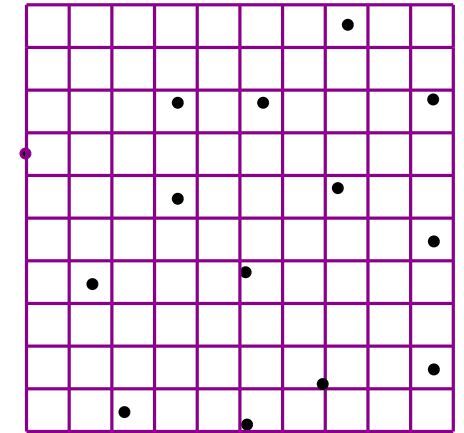
# Hashing the Nearest Neighbor Grid

- $(x, y)$  will be in the box  $(\lfloor \frac{x-x_{\min}}{d} \rfloor, \lfloor \frac{y-y_{\min}}{d} \rfloor)$
- Maximum box coordinate is  $\lceil D/d \rceil$



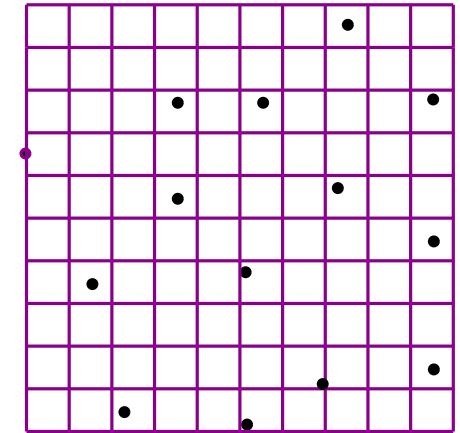
# Hashing the Nearest Neighbor Grid

- $(x, y)$  will be in the box  $(\lfloor \frac{x - x_{\min}}{d} \rfloor, \lfloor \frac{y - y_{\min}}{d} \rfloor)$
- Maximum box coordinate is  $\lceil D/d \rceil$
- Using the Array Hashing technique discussed before, we can hash the points by storing them in their corresponding array element (bucket). Non-empty buckets will return a pointer to list of points in that grid box.



# Hashing the Nearest Neighbor Grid

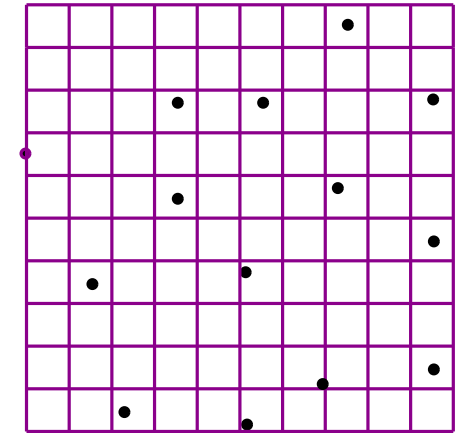
- $(x, y)$  will be in the box  $(\lfloor \frac{x-x_{\min}}{d} \rfloor, \lfloor \frac{y-y_{\min}}{d} \rfloor)$
- Maximum box coordinate is  $\lceil D/d \rceil$
- Using the Array Hashing technique discussed before, we can hash the points by storing them in their corresponding array element (bucket). Non-empty buckets will return a pointer to list of points in that grid box.
- Inserting point into grid takes average  $O(1)$  time. Searching for points in grid box takes average  $O(1)$  time plus # of points in box.





# Hashing the Nearest Neighbor Grid

- $(x, y)$  will be in the box  $(\lfloor \frac{x-x_{\min}}{d} \rfloor, \lfloor \frac{y-y_{\min}}{d} \rfloor)$
- Maximum box coordinate is  $\lceil D/d \rceil$
- Using the Array Hashing technique discussed before, we can hash the points by storing them in their corresponding array element (bucket). Non-empty buckets will return a pointer to list of points in that grid box.
- Inserting point into grid takes average  $O(1)$  time. Searching for points in grid box takes average  $O(1)$  time plus # of points in box.
- Let  $P_i = \{p_1, p_2, \dots, p_n\}$ ,  $\delta_i = \delta(P_i)$  and grid size is  $d = \delta_i/2$
- Gridding Lemma says that each grid box contains at most 1 point
- $\Rightarrow$  Finding points in a grid box takes average  $O(1)$  time

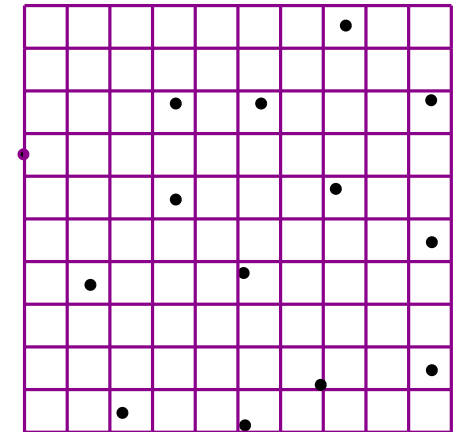


# Outline

- Introduction
- The Gridding Lemma
- An  $O(n \log n)$  sweep line algorithm
- An  $O(n)$  randomized algorithm
  - Hashing a point set
  - The algorithm

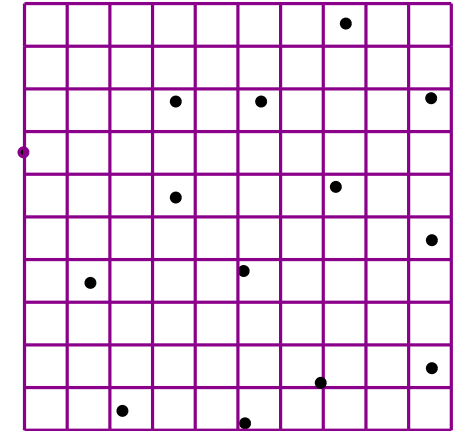
# One Algorithmic Step

- Suppose we know  $\delta_i$  and have hashed and stored the pointset  $P_i$  with grid size  $d = \delta_i/2$ .
- How can we calculate  $\delta_{i+1}$  and, after completion, have  $P_{i+1}$  hashed and stored with grid size  $\delta_{i+1}/2$ ?



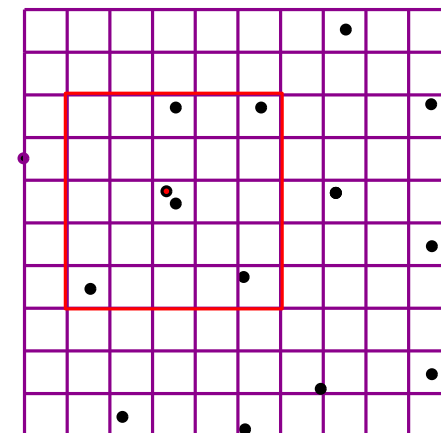
# One Algorithmic Step

- Suppose we know  $\delta_i$  and have hashed and stored the pointset  $P_i$  with grid size  $d = \delta_i/2$ .
- How can we calculate  $\delta_{i+1}$  and, after completion, have  $P_{i+1}$  hashed and stored with grid size  $\delta_{i+1}/2$ ?
- Note that  $\delta_{i+1} = \min(\delta_i, d(p_{i+1}, P_i))$



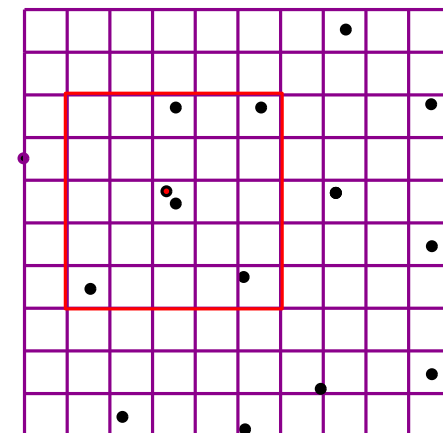
# One Algorithmic Step

- Suppose we know  $\delta_i$  and have hashed and stored the pointset  $P_i$  with grid size  $d = \delta_i/2$ .
- How can we calculate  $\delta_{i+1}$  and, after completion, have  $P_{i+1}$  hashed and stored with grid size  $\delta_{i+1}/2$ ?
- Note that  $\delta_{i+1} = \min(\delta_i, d(p_{i+1}, P_i))$
- $\delta_{i+1} < \delta_i$  iff  $d(p_{i+1}, P_i) = d(p_{i+1}, p) < \delta_i$  for  $p \in P_i$  with  $p$  in one of the 25 boxes within distance 2 of the box containing  $p_{i+1}$



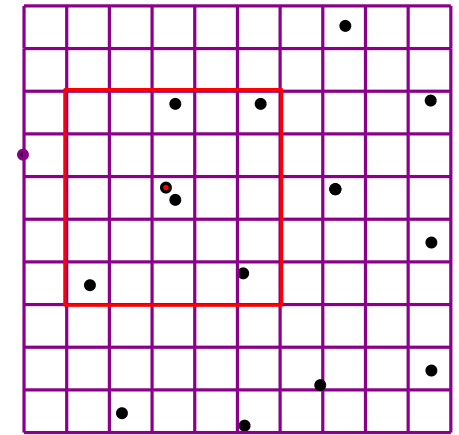
# One Algorithmic Step

- Suppose we know  $\delta_i$  and have hashed and stored the pointset  $P_i$  with grid size  $d = \delta_i/2$ .
- How can we calculate  $\delta_{i+1}$  and, after completion, have  $P_{i+1}$  hashed and stored with grid size  $\delta_{i+1}/2$ ?
- Note that  $\delta_{i+1} = \min(\delta_i, d(p_{i+1}, P_i))$
- $\delta_{i+1} < \delta_i$  iff  $d(p_{i+1}, P_i) = d(p_{i+1}, p) < \delta_i$  for  $p \in P_i$  with  $p$  in one of the 25 boxes within distance 2 of the box containing  $p_{i+1}$
- Using the hash search, these 25 boxes and their contents (at most one point each) can be found in  $O(1)$  time. In  $O(1)$  further time, find  $d'$ , the min distance between  $p_{i+1}$  and these  $\leq 25$  points.
- If  $d' \geq \delta_i$  then  $\delta_{i+1} = \delta_i$   
 $\Rightarrow$  insert  $p_{i+1}$  into the grid in  $O(1)$  average time
- If  $d' < \delta_i$  then  $\delta_{i+1} = d'$   
 $\Rightarrow$  Throw away the entire old grid and  
in  $O(i)$  average time, regrid  $P_{i+1}$  with the new  $d = d'/2$



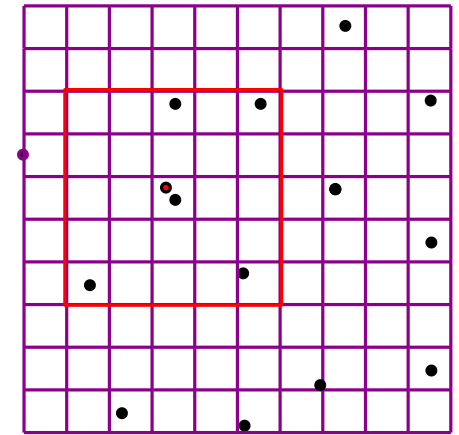
# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .
- For  $i = 3$  to  $n$  do
  - (A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$
  - (B) If  $d' > \delta_i$  then
    - (i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid
    - else ( $d' < \delta_i$ )
    - (ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .



# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .
- For  $i = 3$  to  $n$  do
  - (A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$
  - (B) If  $d' > \delta_i$  then
    - (i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old gridelse ( $d' < \delta_i$ )
    - (ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .

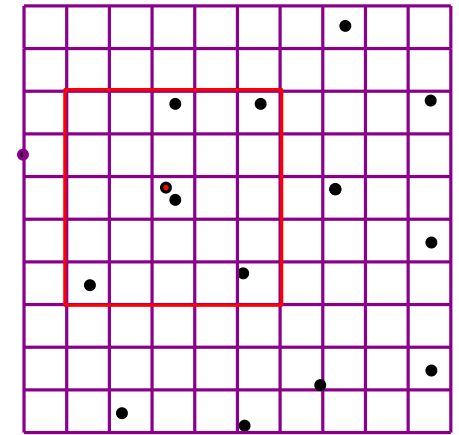


Average Run Time



# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .
- For  $i = 3$  to  $n$  do
  - (A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$
  - (B) If  $d' > \delta_i$  then
    - (i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid
    - else ( $d' < \delta_i$ )
    - (ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .



Average Run Time

1st Step:  $O(n)$

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points

- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .

- For  $i = 3$  to  $n$  do

(A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$

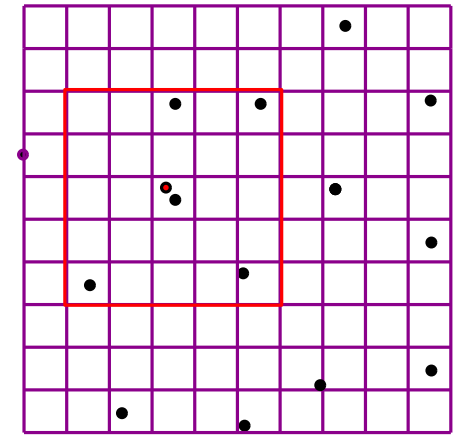
(B) If  $d' > \delta_i$  then

(i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid

else ( $d' < \delta_i$ )

(ii) Set  $\delta_{i+1} = d'$  and

Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .



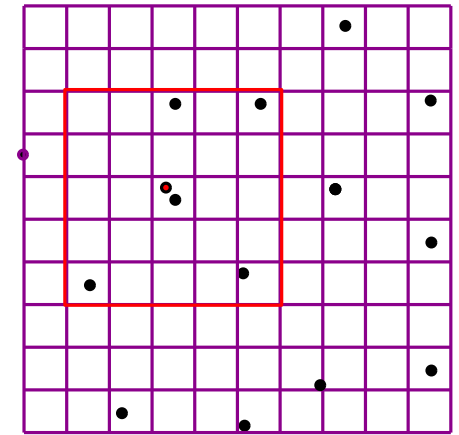
Average Run Time

1st Step:  $O(n)$

2nd Step:  $O(1)$

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .



- For  $i = 3$  to  $n$  do

(A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$

(B) If  $d' > \delta_i$  then

(i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid

else ( $d' < \delta_i$ )

(ii) Set  $\delta_{i+1} = d'$  and

Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .

## Average Run Time

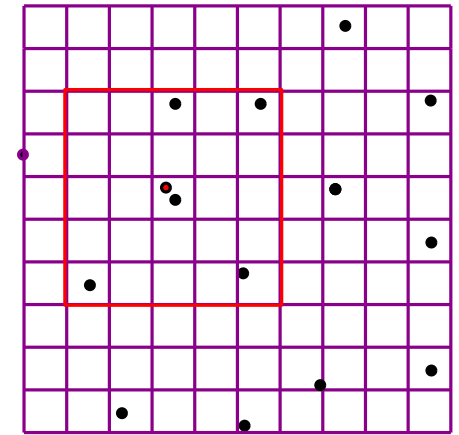
1st Step:  $O(n)$

2nd Step:  $O(1)$

All 3(A):  $O(n)$

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .



- For  $i = 3$  to  $n$  do

(A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$

(B) If  $d' > \delta_i$  then

(i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid

else ( $d' < \delta_i$ )

(ii) Set  $\delta_{i+1} = d'$  and

Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .

## Average Run Time

1st Step:  $O(n)$

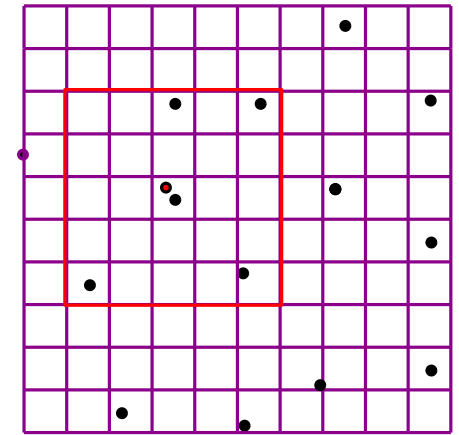
2nd Step:  $O(1)$

All 3(A):  $O(n)$

All 3(B)(i):  $O(n)$

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = d(p_1, p_2)$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .
- For  $i = 3$  to  $n$  do
  - (A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$
  - (B) If  $d' > \delta_i$  then
    - (i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid
    - else ( $d' < \delta_i$ )
    - (ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .



## Average Run Time

1st Step:  $O(n)$

2nd Step:  $O(1)$

All 3(A):  $O(n)$

All 3(B)(i):  $O(n)$

---

$= O(n) +$  time  
for all 3(B)(ii)

# Final Piece

- We just saw the algorithm is  $O(n)$  average running time with exception of 3(B)(ii)
- 3(B)(ii) can be rephased as, in step  $i$ 
  - if  $\delta_{i+1} > \delta_i$  do nothing
  - else do  $O(i + 1)$  average work

# Final Piece

- We just saw the algorithm is  $O(n)$  average running time with exception of 3(B)(ii)
- 3(B)(ii) can be rephased as, in step  $i$   
if  $\delta_{i+1} > \delta_i$  do nothing  
else do  $O(i+1)$  average work
- Note that trivially  $\delta_{i+1} = d(p_j, p_k)$  for some  $j, k \leq i+1$ .  
 $\delta_{i+1} < \delta_i$  only if  $j = i+1$  or  $k = i+1$ .
- Points are in random order, so prob of this happening is  $\leq 2/(i+1)$   
i.e.,  $\Pr(\delta_{i+1} < \delta_i) \leq 2/(i+1)$

# Final Piece

- We just saw the algorithm is  $O(n)$  average running time with exception of 3(B)(ii)
- 3(B)(ii) can be rephased as, in step  $i$   
if  $\delta_{i+1} > \delta_i$  do nothing  
else do  $O(i+1)$  average work
- Note that trivially  $\delta_{i+1} = d(p_j, p_k)$  for some  $j, k \leq i+1$ .  
 $\delta_{i+1} < \delta_i$  only if  $j = i+1$  or  $k = i+1$ .
- Points are in random order, so prob of this happening is  $\leq 2/(i+1)$   
i.e.,  $\Pr(\delta_{i+1} < \delta_i) \leq 2/(i+1)$
- Let  $X_i$  be amount of work done by 3(B)(ii) at step  $i+1$   
 $X_i = 0$  with Prob  $1 - \frac{2}{i+1}$ ;  $X_i = O(i+1)$  with Prob  $\frac{2}{i+1}$ .  
 $\Rightarrow$  expected value  $E(X_i) = \frac{2}{i+1} O(i+1) = O(1)$

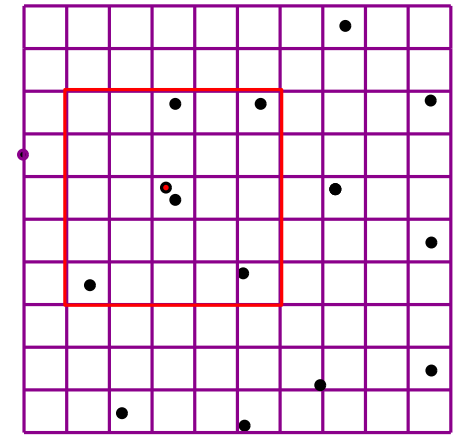


# Final Piece

- We just saw the algorithm is  $O(n)$  average running time with exception of 3(B)(ii)
- 3(B)(ii) can be rephased as, in step  $i$   
if  $\delta_{i+1} > \delta_i$  do nothing  
else do  $O(i+1)$  average work
- Note that trivially  $\delta_{i+1} = d(p_j, p_k)$  for some  $j, k \leq i+1$ .  
 $\delta_{i+1} < \delta_i$  only if  $j = i+1$  or  $k = i+1$ .
- Points are in random order, so prob of this happening is  $\leq 2/(i+1)$   
i.e.,  $\Pr(\delta_{i+1} < \delta_i) \leq 2/(i+1)$
- Let  $X_i$  be amount of work done by 3(B)(ii) at step  $i+1$   
 $X_i = 0$  with Prob  $1 - \frac{2}{i+1}$ ;  $X_i = O(i+1)$  with Prob  $\frac{2}{i+1}$ .  
 $\Rightarrow$  expected value  $E(X_i) = \frac{2}{i+1} O(i+1) = O(1)$
- Total work by 3(B)(ii) is  $Y = \sum_{i=2}^{n-1} X_i$ . Then total average work is  
 $E(Y) = E\left(\sum_{i=2}^{n-1} X_i\right) = \sum_{i=2}^{n-1} E(X_i) = O(n)$

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = p_1, p_2$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .



- For  $i = 3$  to  $n$  do

(A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$

(B) If  $d' > \delta_i$  then

(i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid

else ( $d' < \delta_i$ )

(ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .

## Average Run Time

1st Step:  $O(n)$

2nd Step:  $O(1)$

All 3(A):  $O(n)$

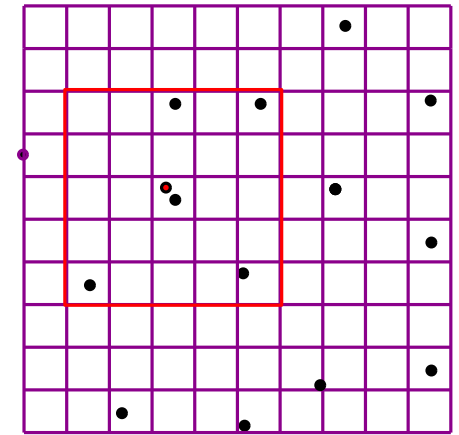
All 3(B)(i):  $O(n)$

---

$= O(n) +$  time  
for all 3(B)(ii)

# The Complete Algorithm

- Randomly label the points as  $p_1, p_2, \dots, p_n$   
Perform an  $O(n)$  scan finding  $x_{\min}, x_{\max}, y_{\min}, y_{\max}$ , the smallest and largest  $x$  and  $y$  coordinates among all of the points
- Set  $\delta_2 = p_1, p_2$ .  
Hash  $P_2$  into the grid with  $d = \delta_2/2$ .
- For  $i = 3$  to  $n$  do
  - (A) Find the min distance  $d'$  between  $p_{i+1}$  and the  $\leq 25$  points in the 25 grid boxes surrounding  $p_{i+1}$
  - (B) If  $d' > \delta_i$  then
    - (i) Set  $\delta_{i+1} = \delta_i$  and insert  $p_{i+1}$  into old grid
    - else ( $d' < \delta_i$ )
    - (ii) Set  $\delta_{i+1} = d'$  and  
Regrid all of  $P_{i+1}$  with grid size  $d = \delta_{i+1}/2$ .



## Average Run Time

1st Step:  $O(n)$

2nd Step:  $O(1)$

All 3(A):  $O(n)$

All 3(B)(i):  $O(n)$

All 3(B)(ii):  $O(n)$

$= O(n)$