CS 3100
Data Structures and Algorithms 2

Lecture 10: D&C: Closest Pair of Points
(Horton’s version of slides)

Co-instructors: Robbie Hott and Tom Horton

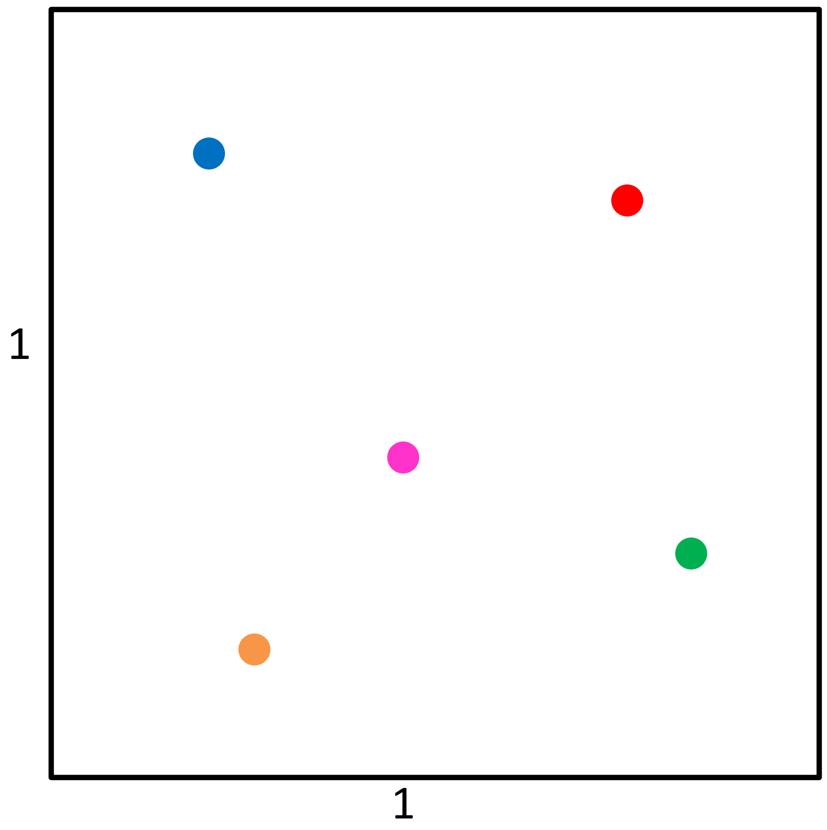
Fall 2023

Readings in CLRS 4th edition:

- Four pages from CLRS 3rd edition on CPP (on our schedule webpage)

Warm Up

Given any 5 points on the unit square, show there's always a pair distance $\leq \frac{\sqrt{2}}{2}$ apart

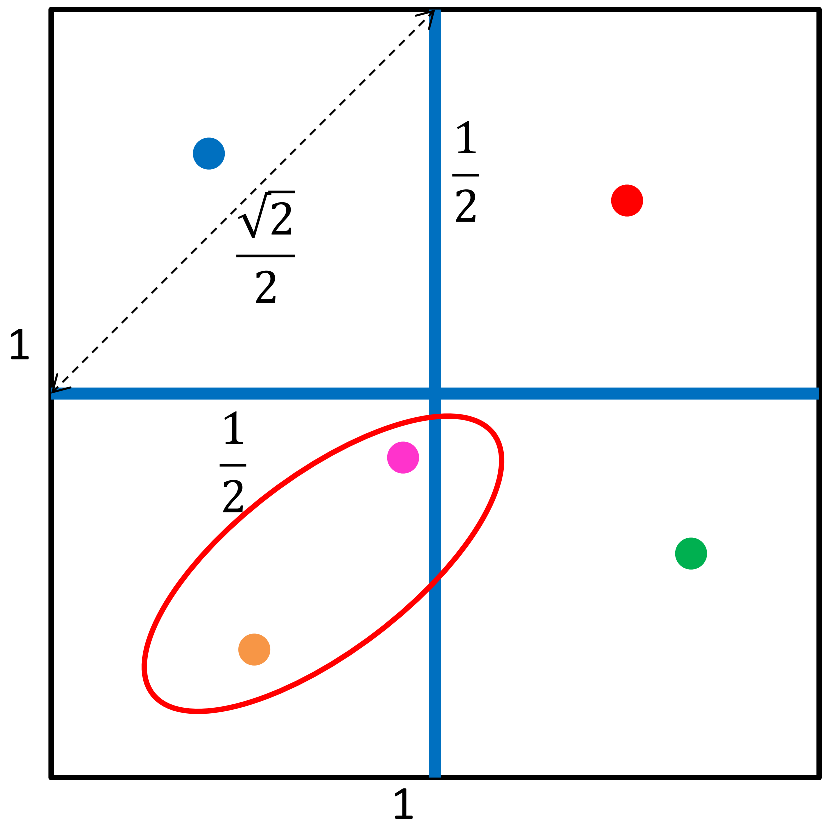


Warm Up Solution

If points p_1, p_2 in same quadrant, then $\delta(p_1, p_2) \leq \frac{\sqrt{2}}{2}$

Given 5 points, two must share the same quadrant

Pigeonhole Principle!



- At a local grocery store, early in the Covid-19 pandemic
- The pigeonhole principle enforcing social distancing!

Announcements

- This slide set:
  - Some Master Theorem examples
  - Closest-pair of points -- which is PA2!

- Upcoming dates
  - PS2 due September 29 (Friday) at 11:59pm
  - PA2 due October 8 (Sunday) at 11:59pm
  - Quizzes 1 and 2 Thursday October 5 in class
- Course email (comes to both professors and head TAs):

cs3100@cshelpdesk.atlassian.net

Review In-Class Activity

Master Theorem Example 1

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

- Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
- Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
- Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

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

Case 2

$$\Theta\left(n^{\log_2 2} \log n\right) = \Theta(n \log n)$$

Master Theorem Example 2

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

$$T(n) = 4T\left(\frac{n}{2}\right) + 5n$$

Case 1

$$\Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$$

Master Theorem Example 3

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

$$T(n) = 3T\left(\frac{n}{2}\right) + 8n$$

Case 1

$$\Theta\left(n^{\log_2 3}\right) \approx \Theta(n^{1.585})$$

Master Theorem Example 4

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

$$T(n) = 2T\left(\frac{n}{2}\right) + 15n^3$$

Case 3

Master Theorem Example 4

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

$$T(n) = 2T\left(\frac{n}{2}\right) + 15n^3$$

Case 3

$$\Theta(n^3)$$

Important: For Case 3, need to additionally check that $2f(n/2) \leq cf(n)$ for constant $c < 1$ and sufficiently large n

$$2f(n/2) = 30(n/2)^3 = \frac{30}{8}n^3 \leq \frac{1}{4}(15n^3)$$

Master Theorem Example 5

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

$$T(n) = 16T\left(\frac{n}{4}\right) + 15n^{1.5}$$

Master Theorem Example 6

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af\left(\frac{n}{b}\right) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

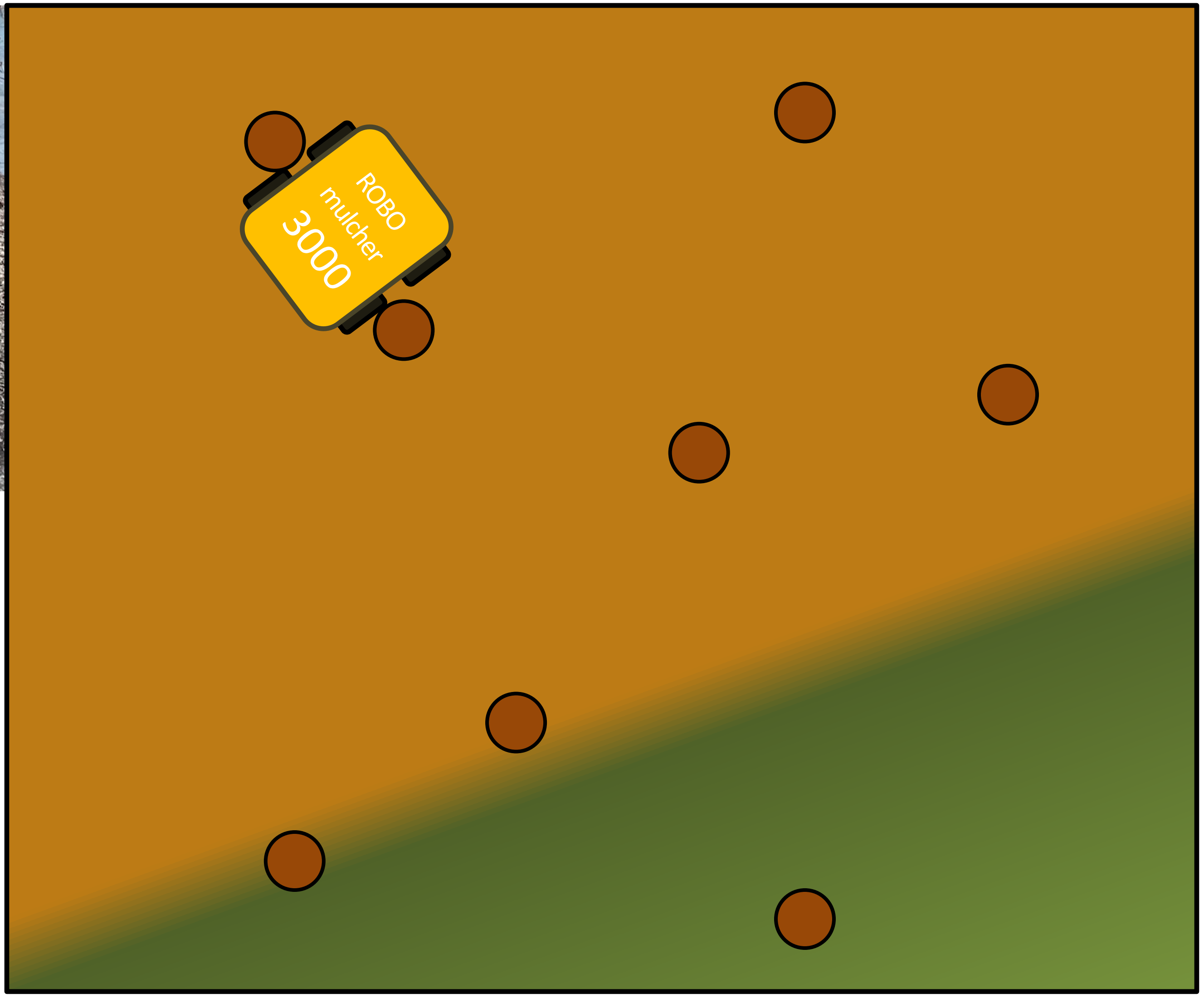
$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Robbie's Yard

Robbie’s Yard

There has to be an easier way!

Constraints: Trees and Plants



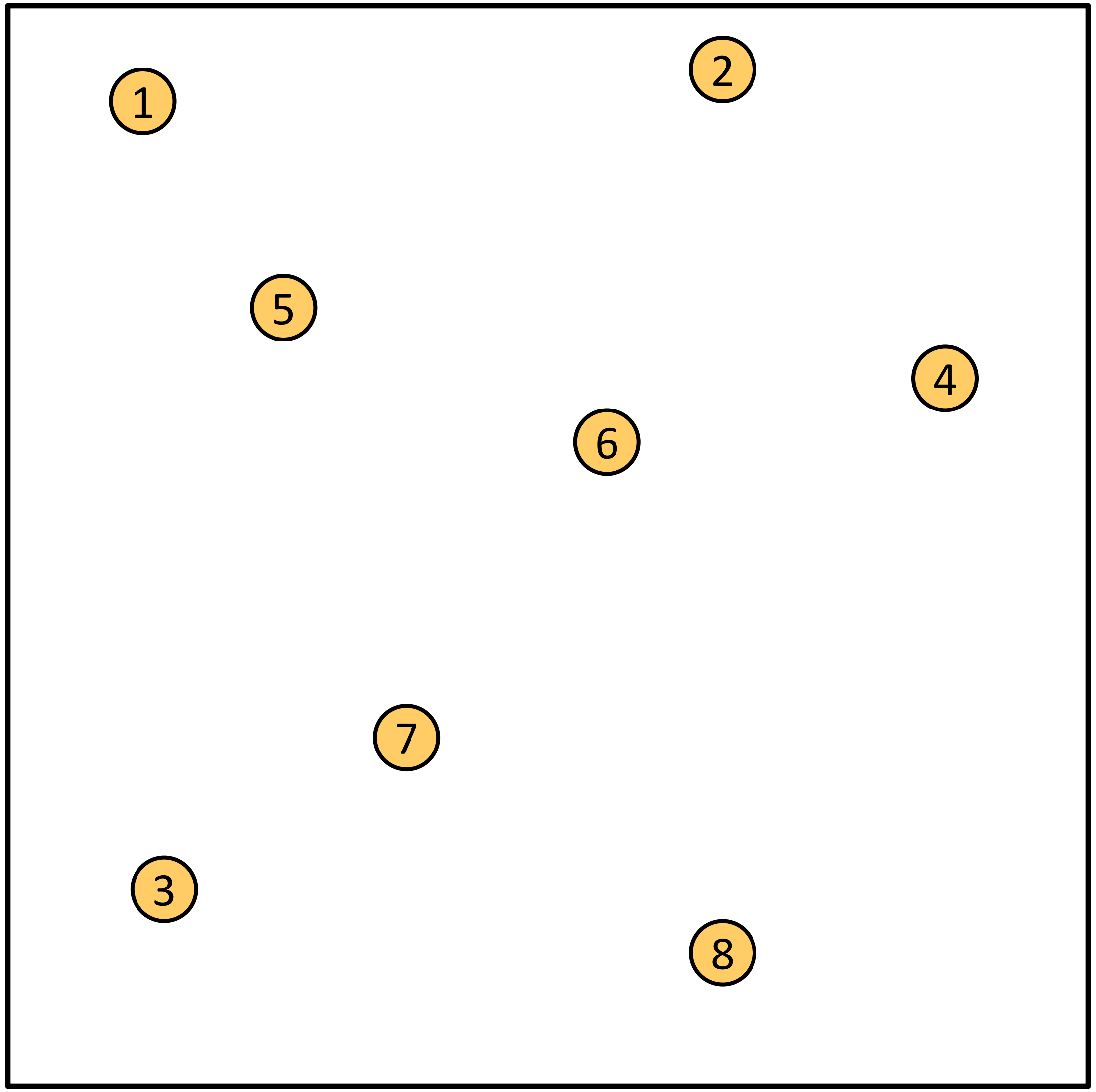
Need to find:

Closest Pair of Trees - how wide can the robot be?

Closest Pair of Points

Given:
A list of points

Return:
Pair of points with smallest distance apart



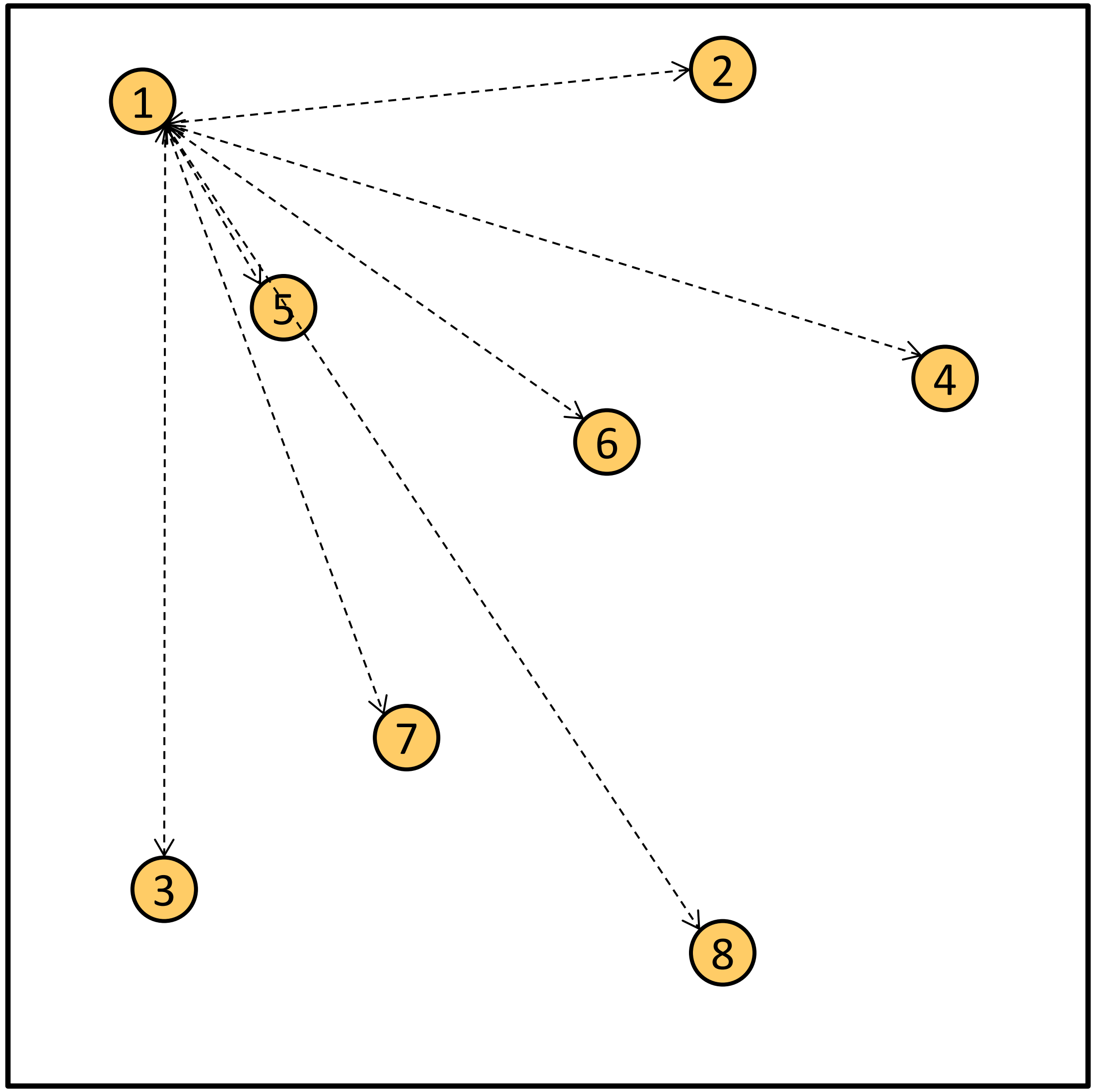
Closest Pair of Points: Naïve

Given:
A list of points

Return:
Pair of points with smallest distance apart

Algorithm: $O(n^2)$
Test every pair of points, return the closest.

We can do better! $\Theta(n \log n)$



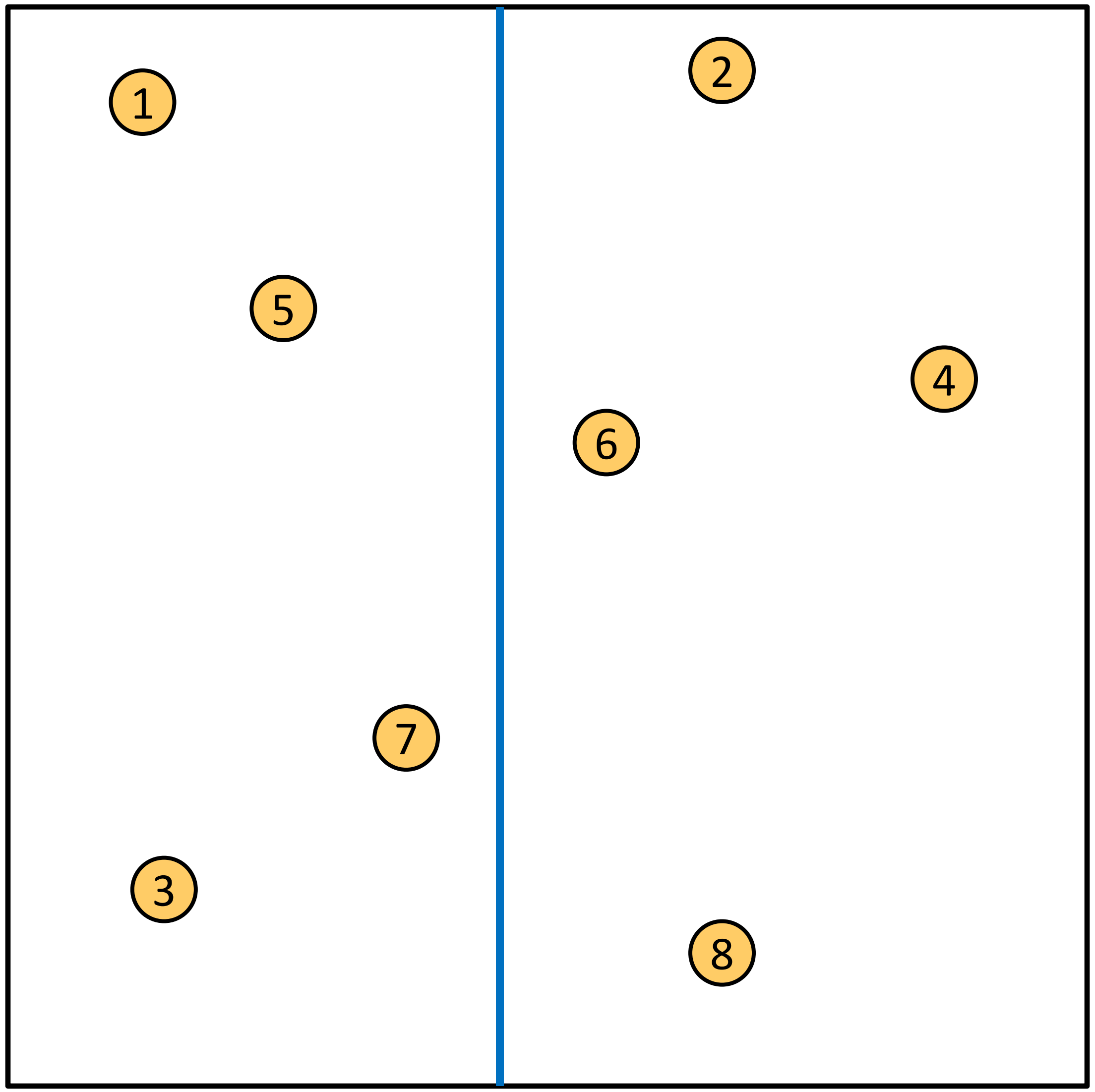
Closest Pair of Points: D&C

Divide: How?

At median x coordinate

Conquer:

Combine:



Closest Pair of Points: D&C

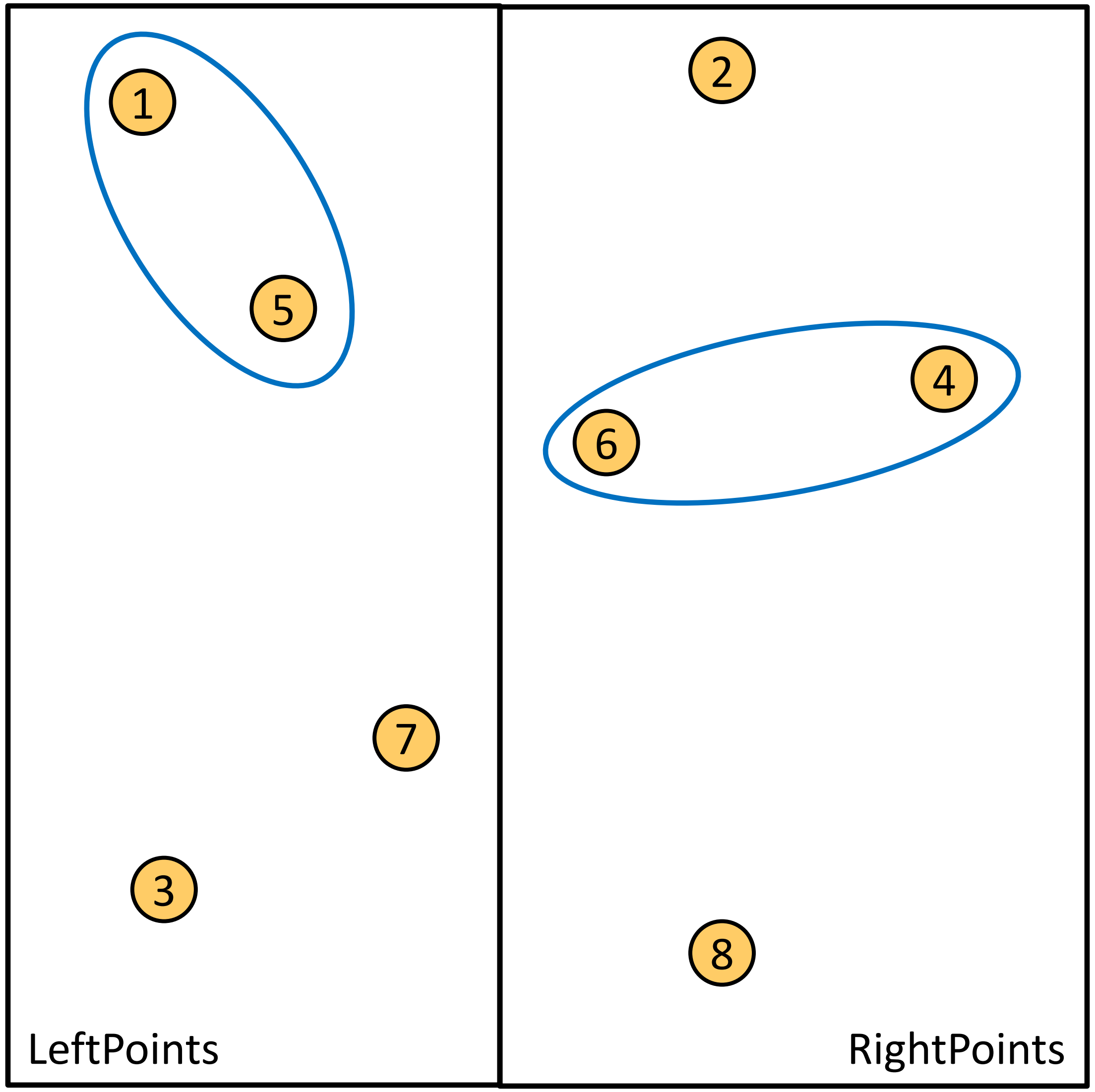
Divide:

At median x coordinate

Conquer:

Recursively find closest pairs from Left and Right

Combine:



Closest Pair of Points: D&C

Divide:

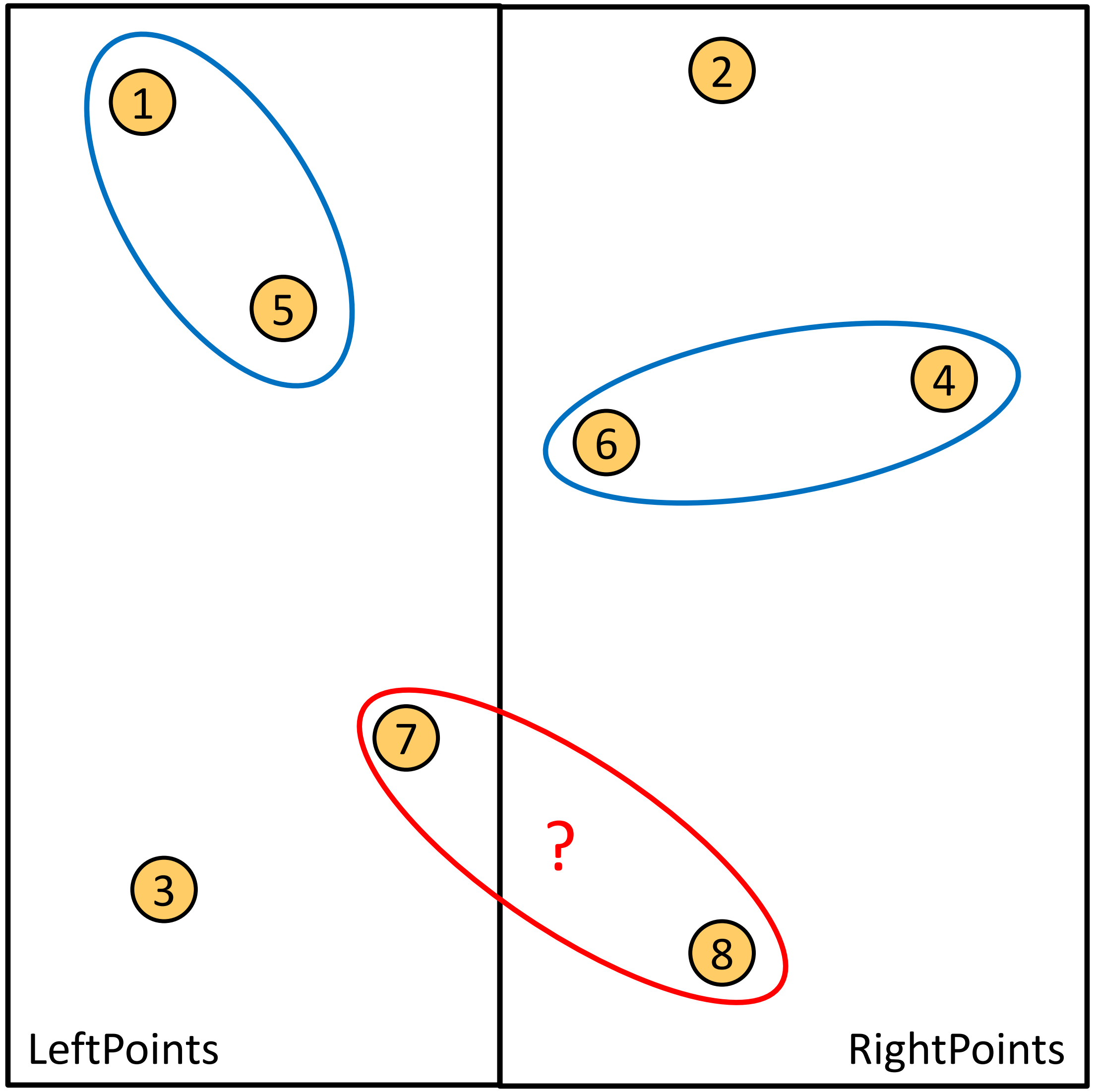
At median x coordinate

Conquer:

Recursively find closest pairs from Left and Right

Combine:

Return min of Left and Right pairs Problem?



Closest Pair of Points: D&C

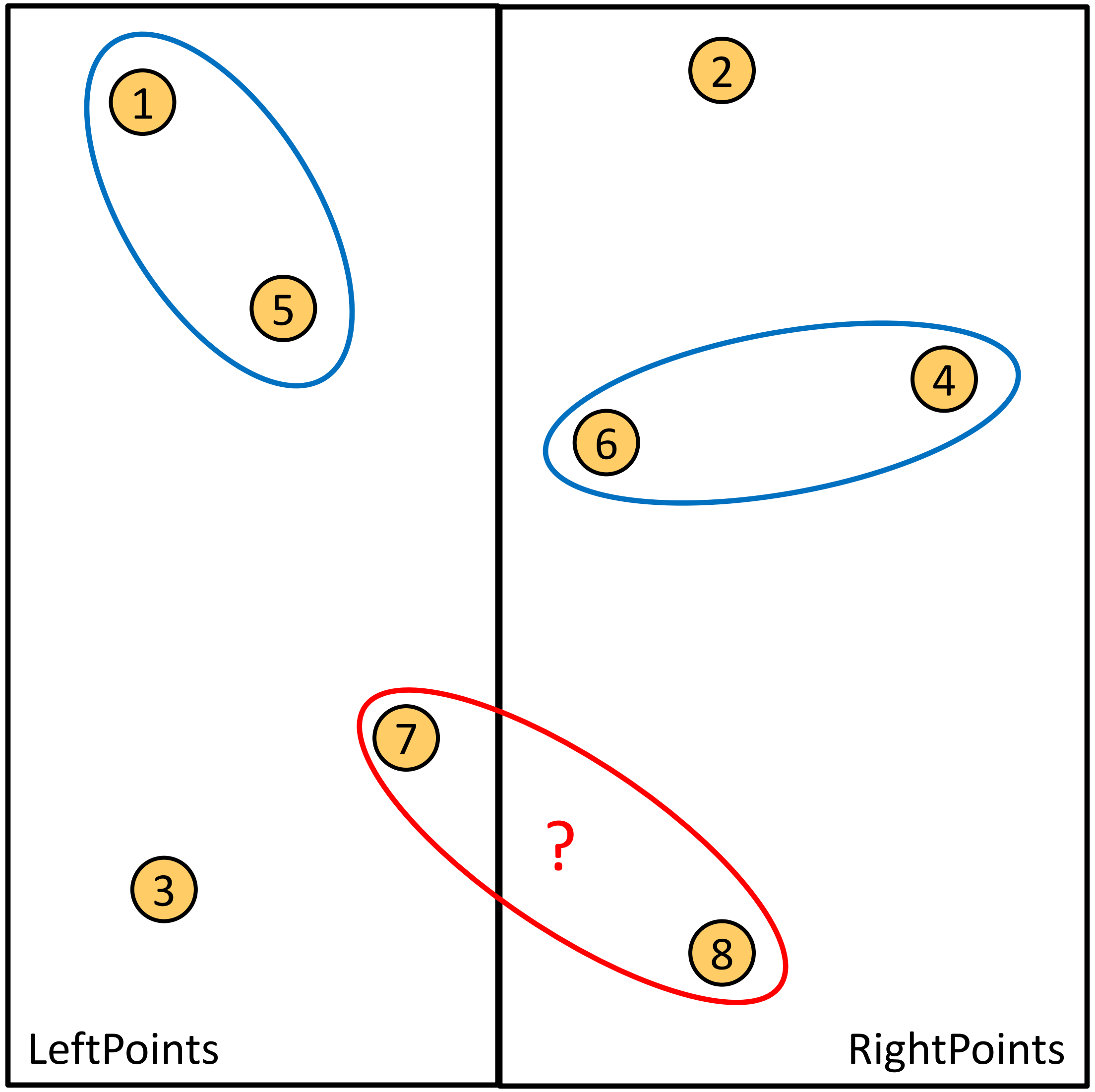
Combine:

2 Cases:

1. Closest Pair is completely in Left or Right

2. Closest Pair Spans our "Cut"

Need to test points across the cut



Spanning the Cut

Combine:

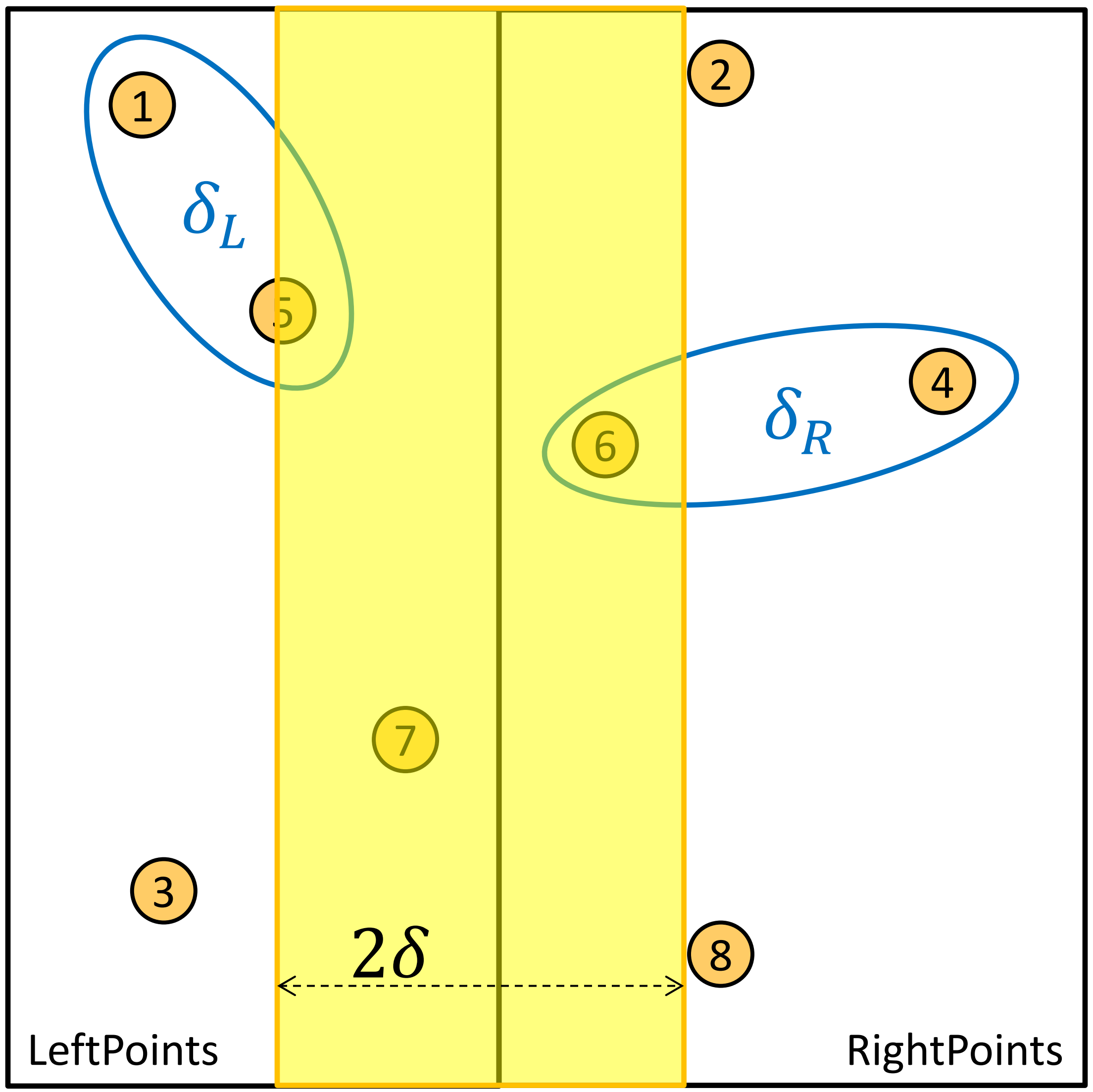
2. Closest Pair Spanned our "Cut"

Need to test points across the cut

Compare all points within $\delta = \min\{\delta_L, \delta_R\}$ of the cut.

(In the "runway")

How many are there?



Spanning the Cut

Combine:

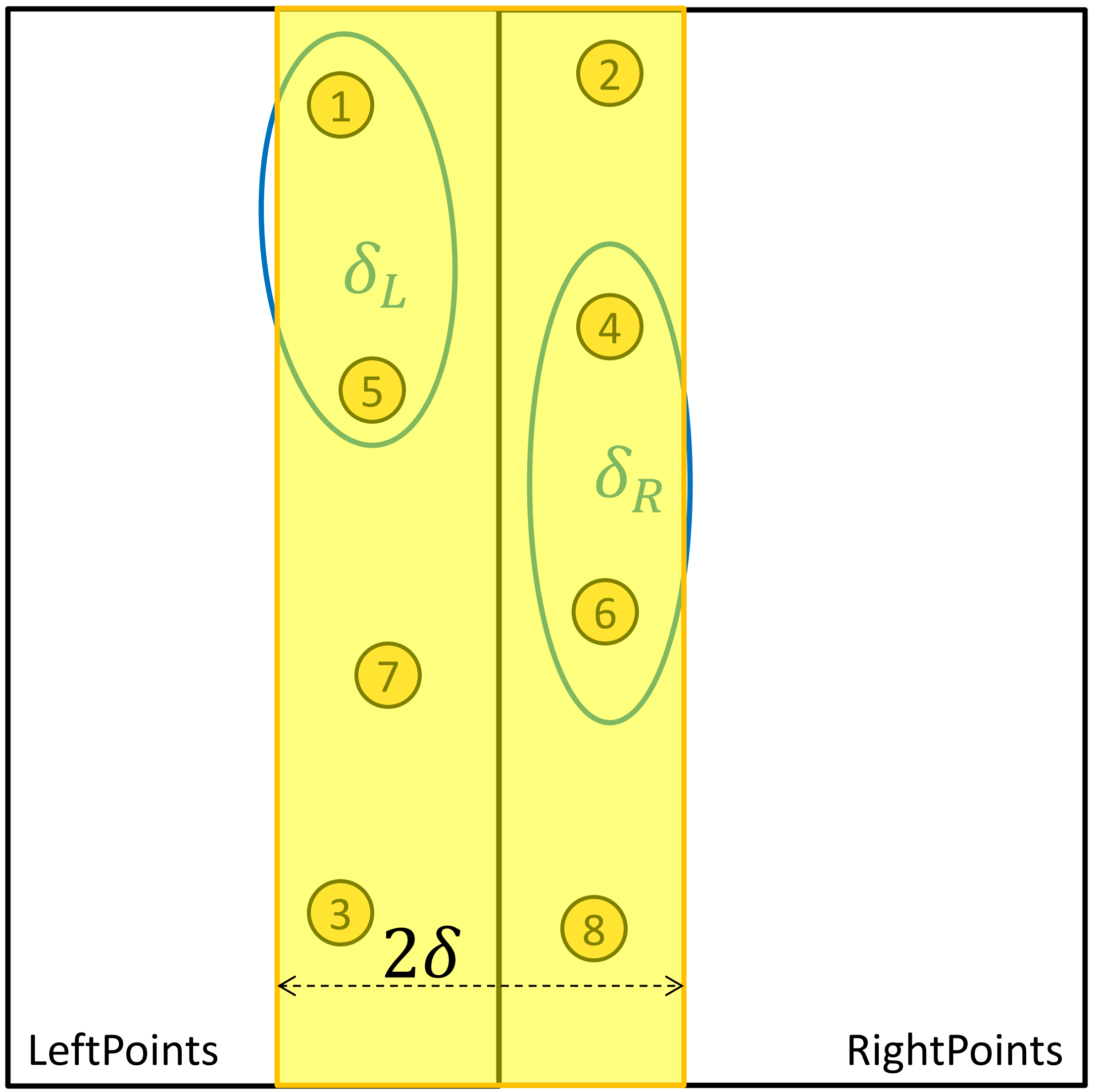
2. Closest Pair Spanned our “Cut”

Need to test points across the cut

Slow approach Compare all points within $\delta = \min\{\delta_L, \delta_R\}$ of the cut.

How many are there?

$$T(n) = 2T\left(\frac{n}{2}\right) + \left(\frac{n}{2}\right)^2$$
$$= \Theta(n^2)$$



Spanning the Cut

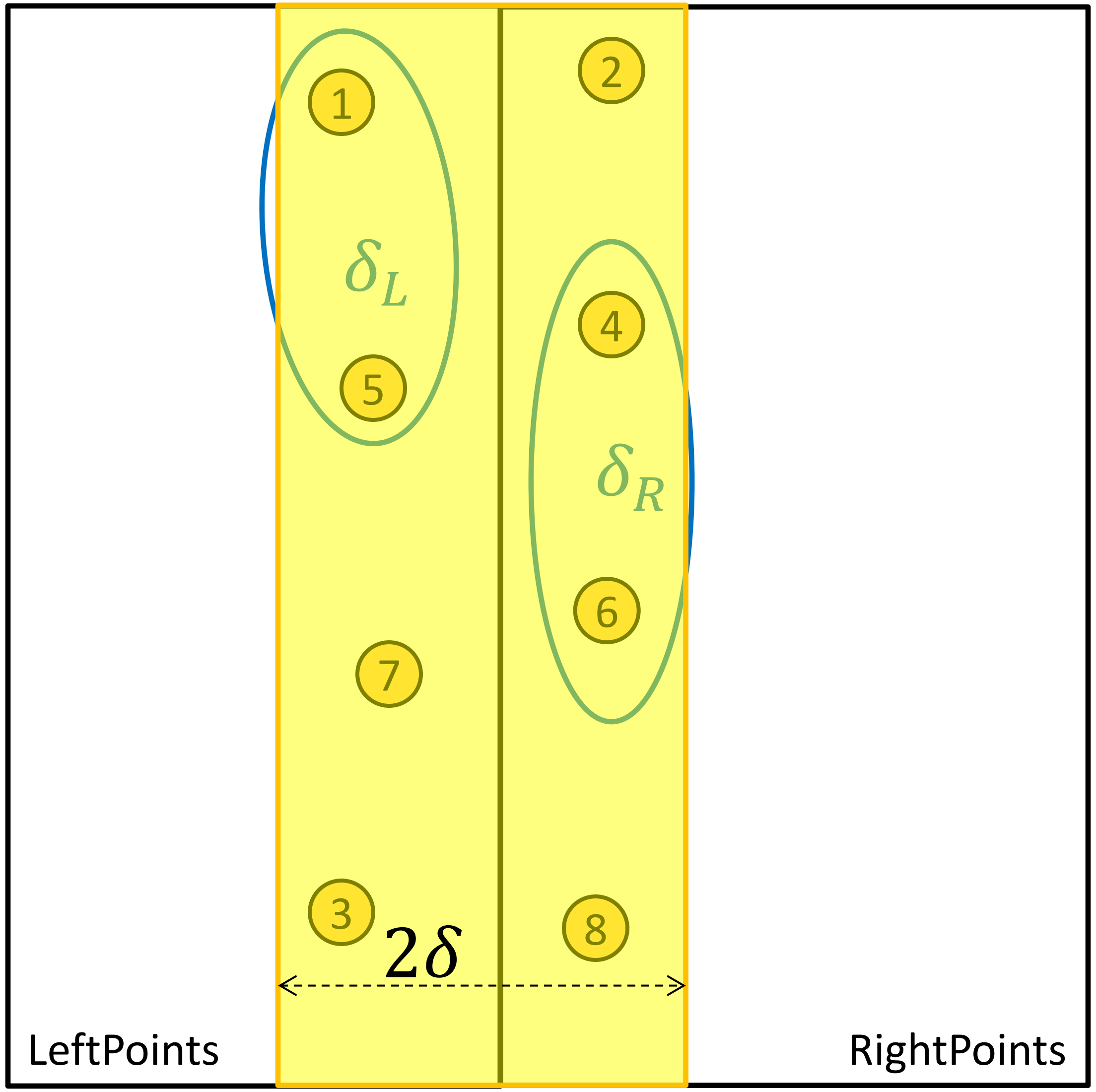
Combine:

2. Closest Pair Spanned our "Cut"

Need to test points across the cut

We don't need to test all pairs!

Don't need to test points that are $> \delta$ from one another



Our Strategy for Combine Step

- Before we go into details, let’s explain our strategy
  - Our goal: find the pair crossing the cut that has distance $< \delta$ **<u>and</u>** whose distance is the minimum of such pairs
- We want to avoid the following $\Theta(n^2)$ approach:
  - For each point in the runway, compare **to all others in the runway** to see if they cross the cut and are closer than δ
- We’re going to find an approach that’s $\Theta(n)$:
  - For each point in the runway, compare to $\boldsymbol{k}$ **near-by points in the runway** to see if they cross the cut and are closer than δ
  - Doesn’t matter what k is. As long as it’s a constant!
  - Here are 2 ways to find a valid k, both based on geometry

#1: Showing k=15 is Valid

Reducing Search Space

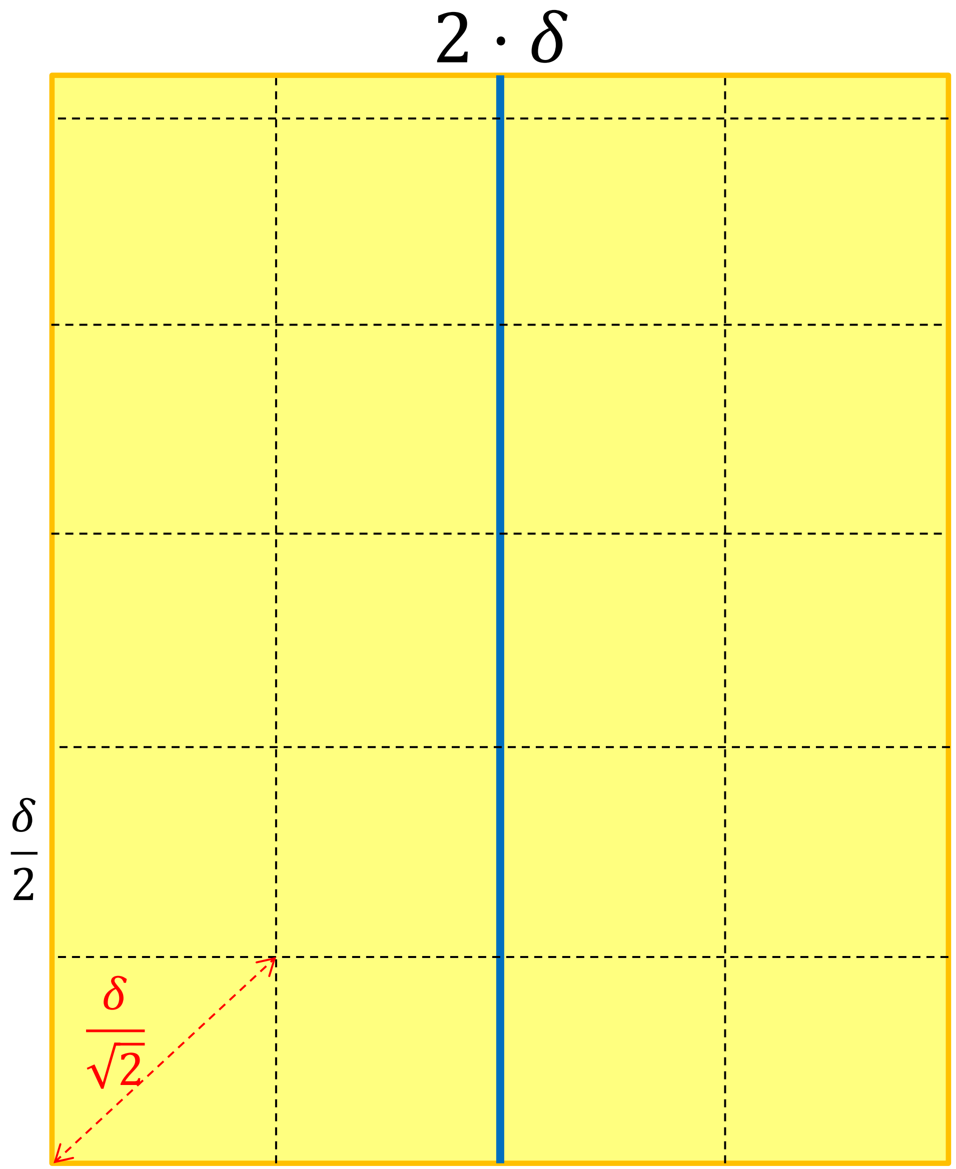
Combine:

2. Closest Pair Spanned our "Cut"

Need to test points across the cut

Divide the "runway" into square cubbies of size $\frac{\delta}{2}$

Each cubby will have at most 1 point!



Reducing Search Space

Combine:

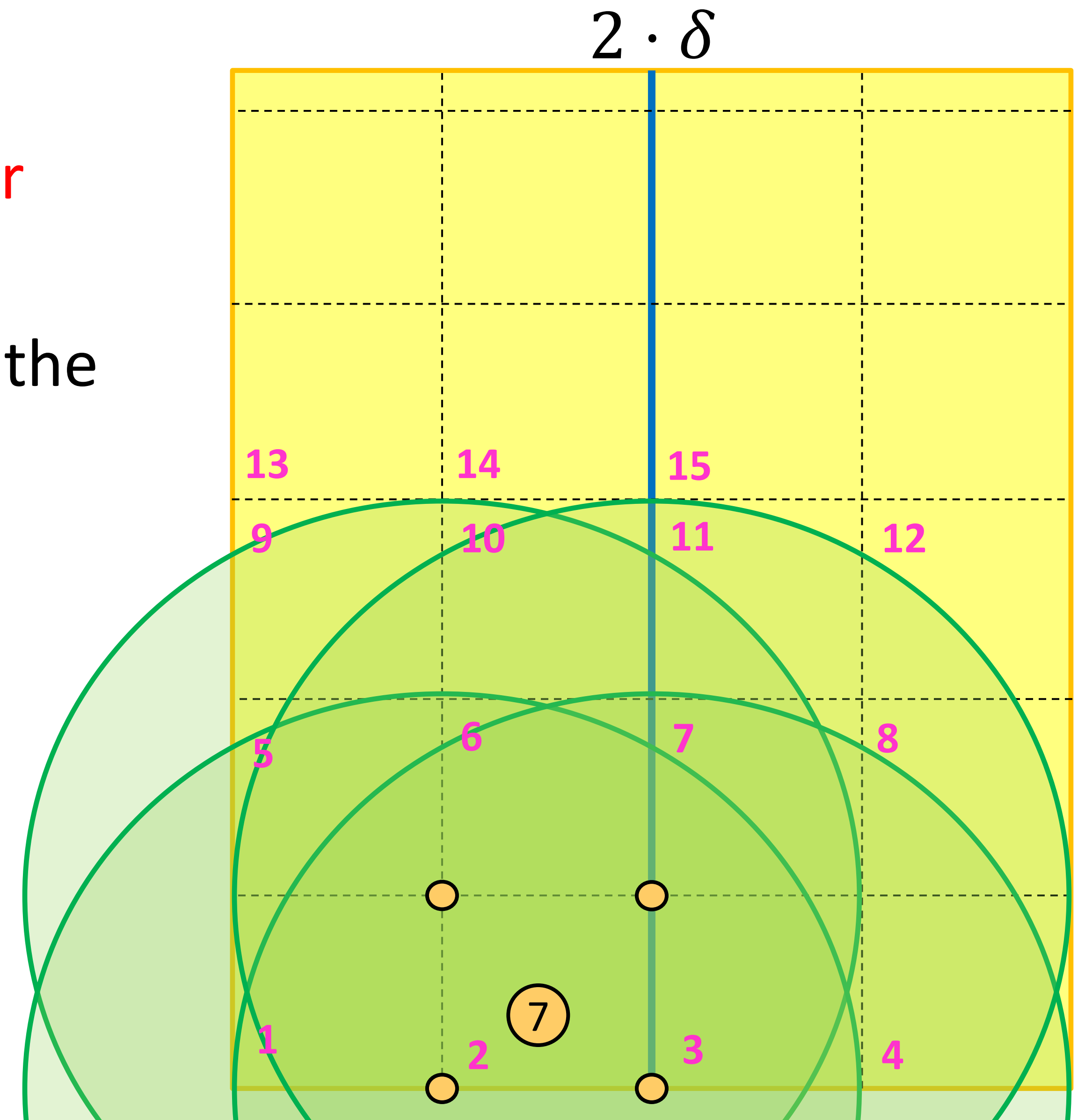
2. Closest Pair Spanned our "Cut"

Need to test points across the cut

Divide the "runway" into square cubbies of size $\frac{\delta}{2}$

How many cubbies could contain a point $< \delta$ away?

Each point compared to ≤ 15 other points



#2: Showing k=7 is Valid

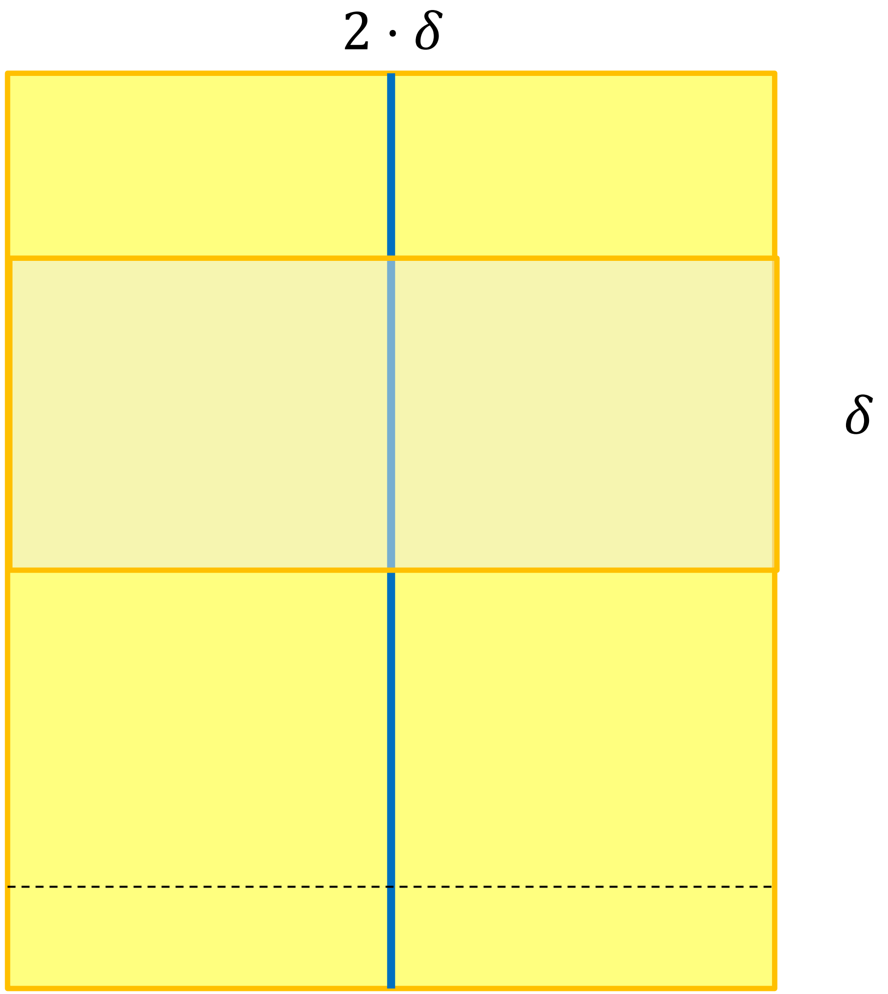
Reducing Search Space

Combine:

Need to test points across the cut

Claim #1: if two points are the closest pair that cross the cut, then you can surround them in a box that's $2 \cdot \delta$ wide by δ tall.

Let's draw some examples.

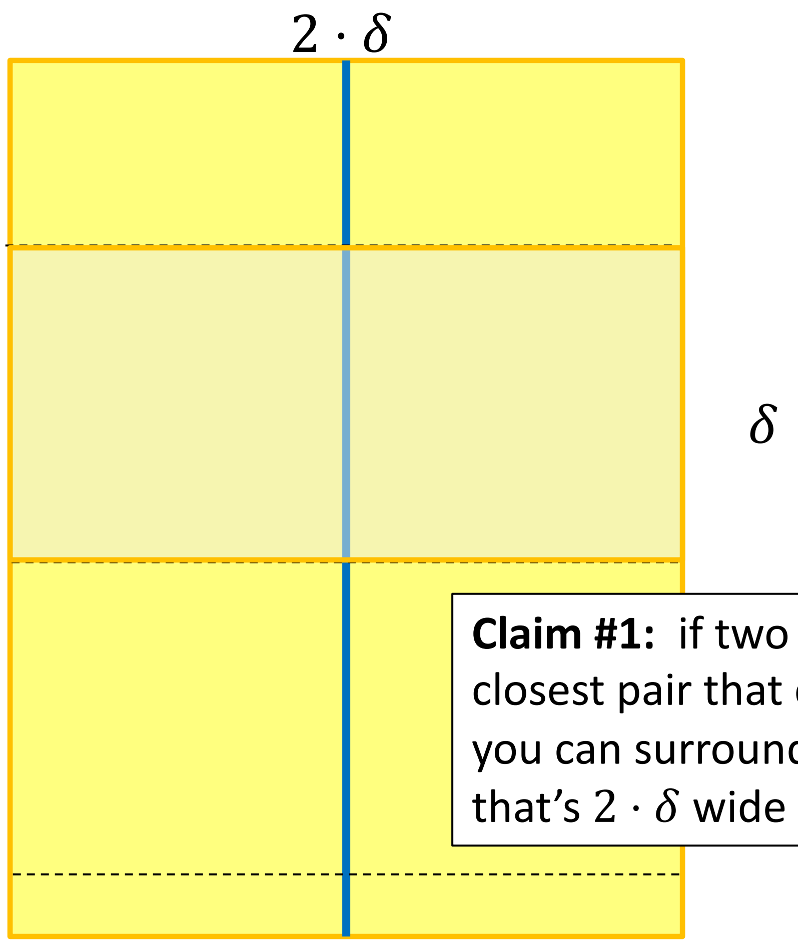


Reducing Search Space

Assume you're checking in increasing y-order, and you've reached the first point of the closest pair.
Do you have to look at **all points above it** to be <u>guaranteed</u> to find the other point and the minimum distance?

No!

- Imagine you drew a box with its bottom at point's y-coordinate.
- See Claim #1.
- Claim #2: only 8 points can be in the box.



Claim #1: if two points are the closest pair that cross the cut, then you can surround them in a box that's $2 \cdot \delta$ wide by δ tall.

Spanning the Cut

Combine:

2. Closest Pair Spanned our "Cut"

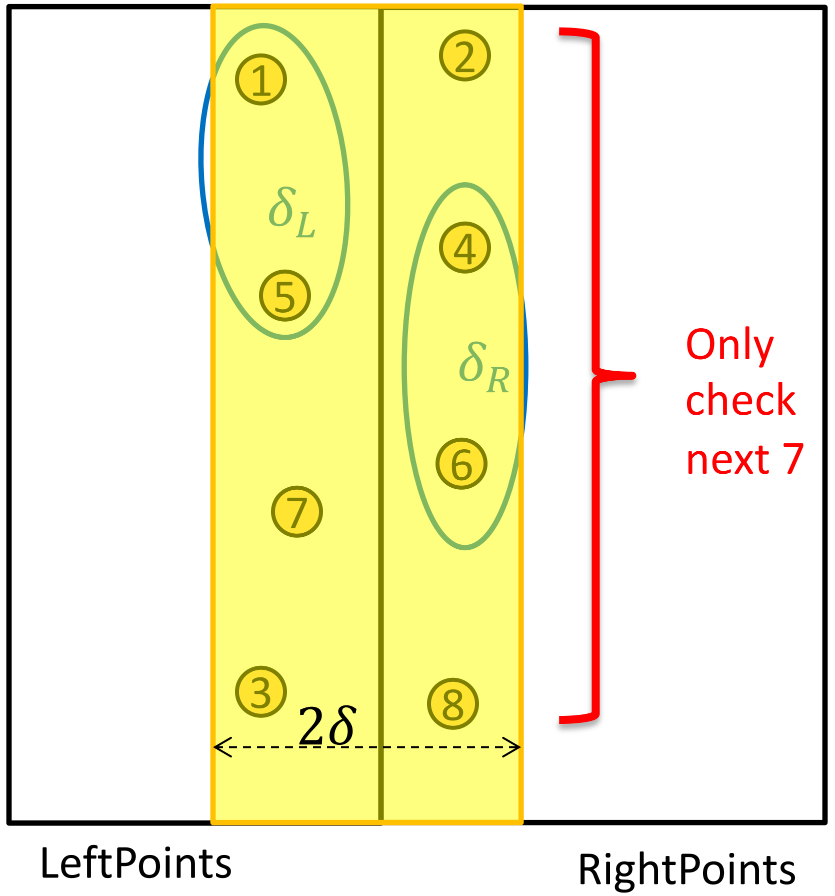
Consider points in runway in increasing y-order.

For a given point *p*, we can *prove* the 8^{th} point and beyond is more than δ from *p*.

(pp. 1041-2 in CLRS 3^{rd} edition PDF)

So for each point in runway, check next 7 points in y-order.

$\Theta(n)$



Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate

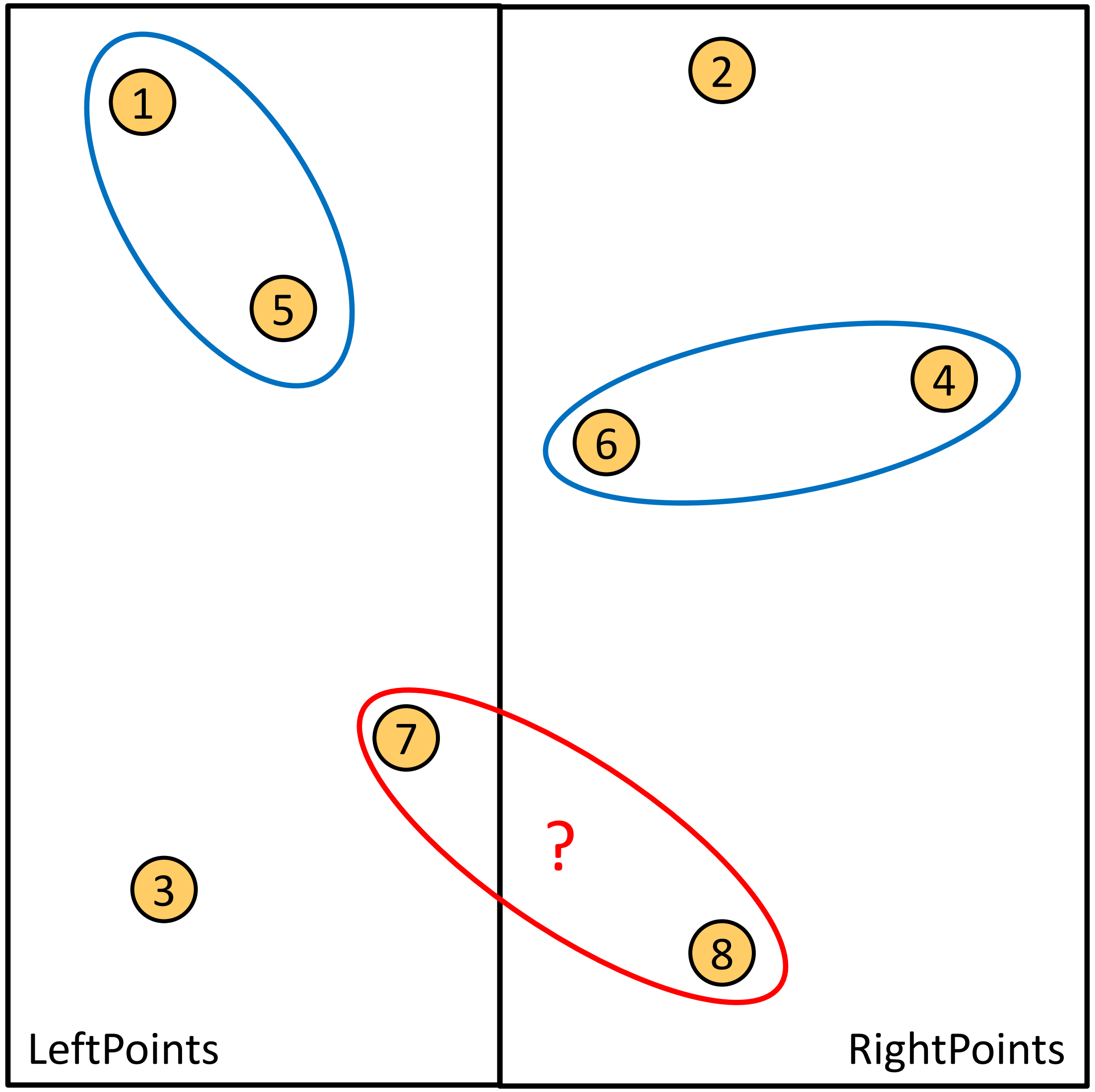
Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

Conquer: Recursively compute the closest pair of points in each list

Base case?

Combine:

- Construct list of points in the runway (x-coordinate within distance δ of median)
- Sort runway points by y-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points



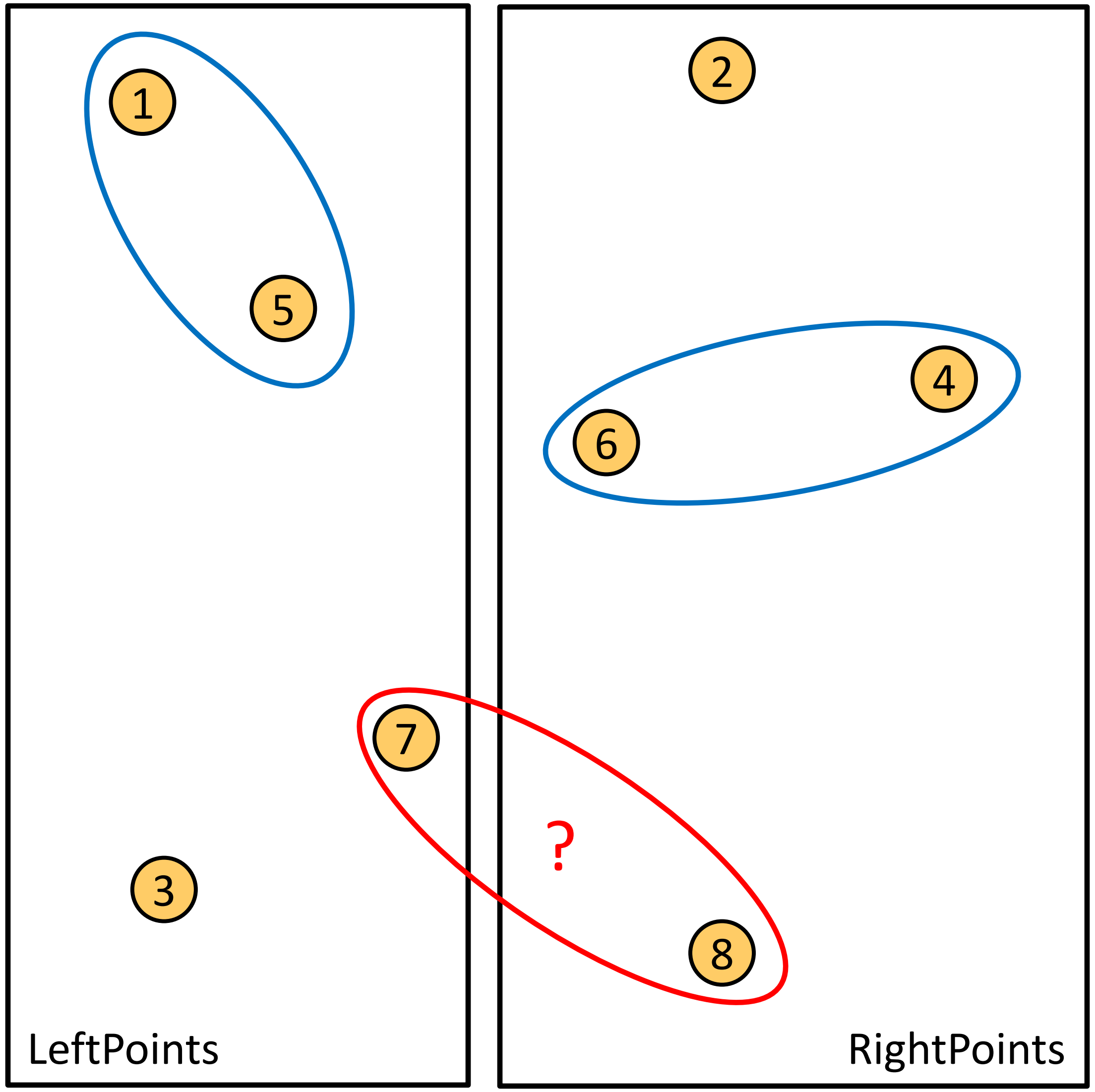
Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

But sorting is an $O(n \log n)$ algorithm – combine step is still too expensive! We need $O(n)$

- Construct list of points in … way (x-coordinate within distan… δ of median)
- Sort runway points by y-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points



Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

Conquer: Recursively compute the closest pair of points in each list

Base case?

Combine:

- Construct list of points in the runway (x-coordinate within distance δ of median)
- Sort runway points by y-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

Possible Solution #1 to this? Maintain additional information in the recursion

- Minimum distance among pairs of points in the list
- List of points sorted according to y-coordinate

Instead of sorting runway points by y-coordinate, use this index by y coordinate?

Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

Conquer: Recursively compute the closest pair of points in each list

Base case?

Combine:

- Construct list of points in the runway (x-coordinate within distance δ of median)
- Sort runway points by y-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

Possible Solution #2 to this?

- Merge sorted list of points by y-coordinate and construct list of points in the runway (sorted by y-coordinate)
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

Closest Pair of Points: Divide and Conquer

What is the running time?

$\Theta(n \log n)$

$T(n)$

$\Theta(n \log n)$ **Initialization:** Sort points by x-coordinate

$\Theta(1)$ **Divide:** Partition points into two lists of points based on x-coordinate (split at the median x)

$2T(n/2)$ **Conquer:** Recursively compute the closest pair of points in each list

Combine:

- $\Theta(n)$ Somehow access runway points in increasing y-coordinate order
- $\Theta(n)$ Compare each point in runway to 7 or 15 points above it and save the closest pair
- $\Theta(1)$ Output closest pair among left, right, and runway points

$$T(n) = 2T(n/2) + \Theta(n)$$

Case 2 of Master's Theorem

$$T(n) = \Theta(n \log n)$$

CPP and PA2

- You've got the algorithm strategy!
- There's trickiness in the details to avoid $\omega(n)$ in processing the runway
- Advice: write the $\theta(n^2)$ solution to check you D&C solution for correctness