More Divide and Conquer: Quicksort and Closest Pair of Points

CS 4102: Algorithms

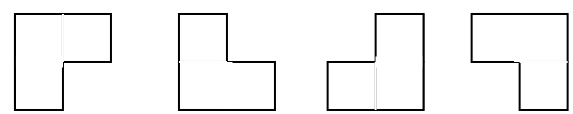
Fall 2021

Mark Floryan and Tom Horton

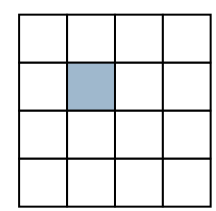
Trominoes

Next Example: Trominos

- Tiling problems
  - For us, a game: Trominos
  - In “real” life: serious tiling problems regarding component layout on VLSI chips
- Definitions
  - Tromino



  - A deficient board
    - n x n where n = 2^k
    - exactly one square missing



- Problem statement:
  - Given a deficient board, tile it with trominos
    - Exact covering, no overlap

Trominos: Playing the Game, Strategy

- Java app for Trominos:
  http://www3.amherst.edu/~nstarr/puzzle.html

- How can we approach this problem using Divide and Conquer?
- Small solutions: Can we solve them directly?
  - Yes: 2 x 2 board
- Next larger problem: 4 x 4 board
  - Hmm, need to divide it
  - Four 2 x 2 boards
  - Only one of these four has the missing square
    - Solve it directly!
  - What about the other three?

Trominos: Key to the Solution

- Place one tromino where three 2 x 2 boards connect
  - You now have three 2 x 2 deficient boards
  - Solve directly!
- General solution for deficient board of size n
  - Divide into four boards
  - Identify the smaller board that has the removed tile
  - Place one tromino that covers the corner of the other three
  - Now recursively process all four deficient boards
  - Don’t forget! First, check for n==2

```
Input Parameters: n, a power of 2 (the board size);
                  the location L of the missing square
Output Parameters: None
tile(n,L) {
   if (n == 2) {
      // the board is a right tromino T
      tile with T
      return
   }
   divide the board into four n/2 × n/2 subboards
   place one tromino as in Figure 5.1.4(b)
   // each of the 1 × 1 squares in this tromino
   // is considered as missing
   let m1,m2,m3,m4 be the locations of the missing squares
   tile(n/2,m1)
   tile(n/2,m2)
   tile(n/2,m3)
   tile(n/2,m4)
}
```

Trominos: Analysis

- What do we count? What's the basic operation?
  - Note we place a tromino and it stays put
  - No loops or conditionals other than placing a tile
  - Assume placing or drawing a tromino is constant
  - Assume that finding which subproblem has the missing tile is constant
- Conclusion: we can just count how many trominos are placed
- How many fit on a n x n board?
  - $(n^2 - 1) / 3$
- Do you think this optimal?

Trominos: Analysis

- Runtime?
- If 'n' is the size of one board dimension (nxn board)
  - 4 subproblems of size n/2 x n/2
  - O(1) to place one tromino "across the cuts" and "combine"
- $T(n) = 4T(n/2) + 1 = ??$
- Also, think intuitively. There are n^2 board spaces and each "round" you are placing one tromino (3 spaces)
  - So at least n^2 / 3 JUST to place the Trominos

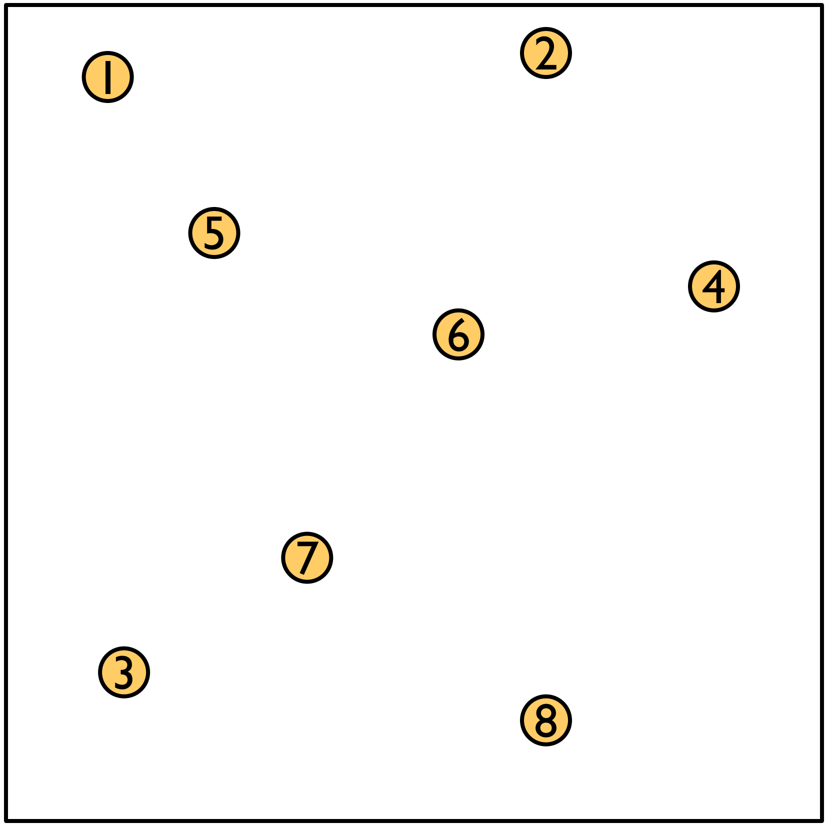
Closest Pair of Points

Readings: CLRS 33.4

Closest Pair of Points in 2D Space

Given:
A list of points

Return:
Distance of the pair of points that are closest together
(or possibly the pair too)



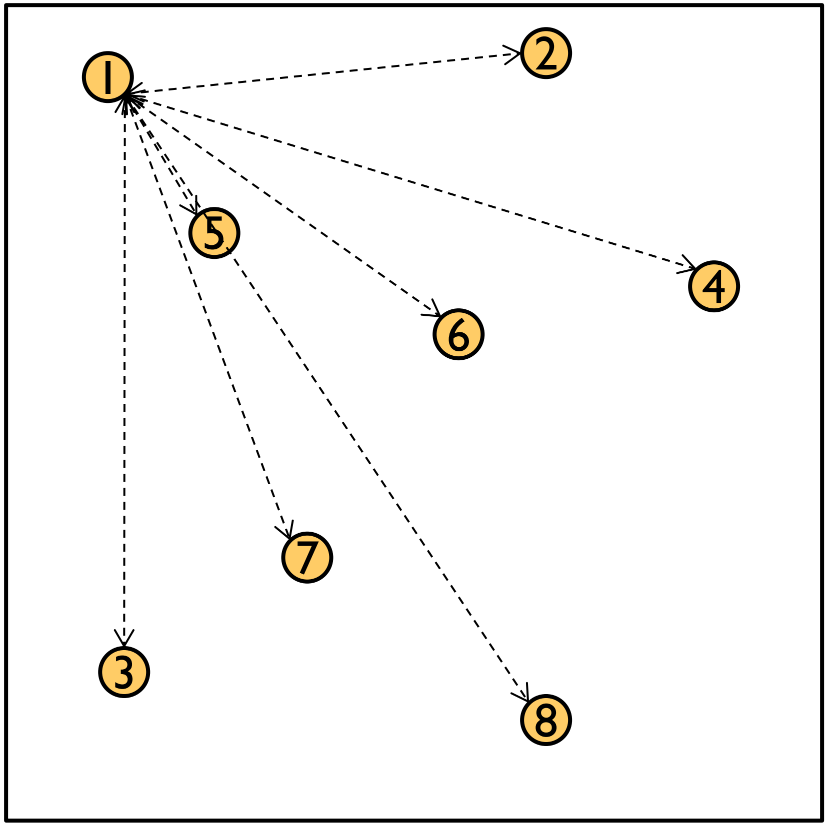
Closest Pair of Points: Naïve

Given:
A list of points

Return:
Distance of the closest pair of points

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

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



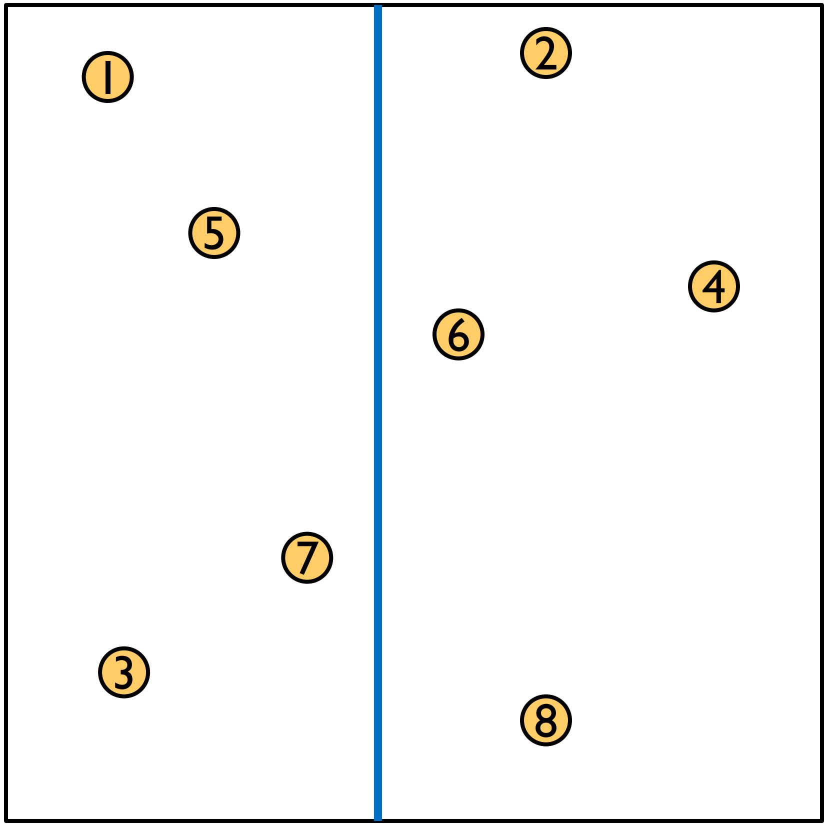
Closest Pair of Points: D&C

Divide: How?

At median x coordinate

Conquer:

Combine:



Closest Pair of Points: D&C

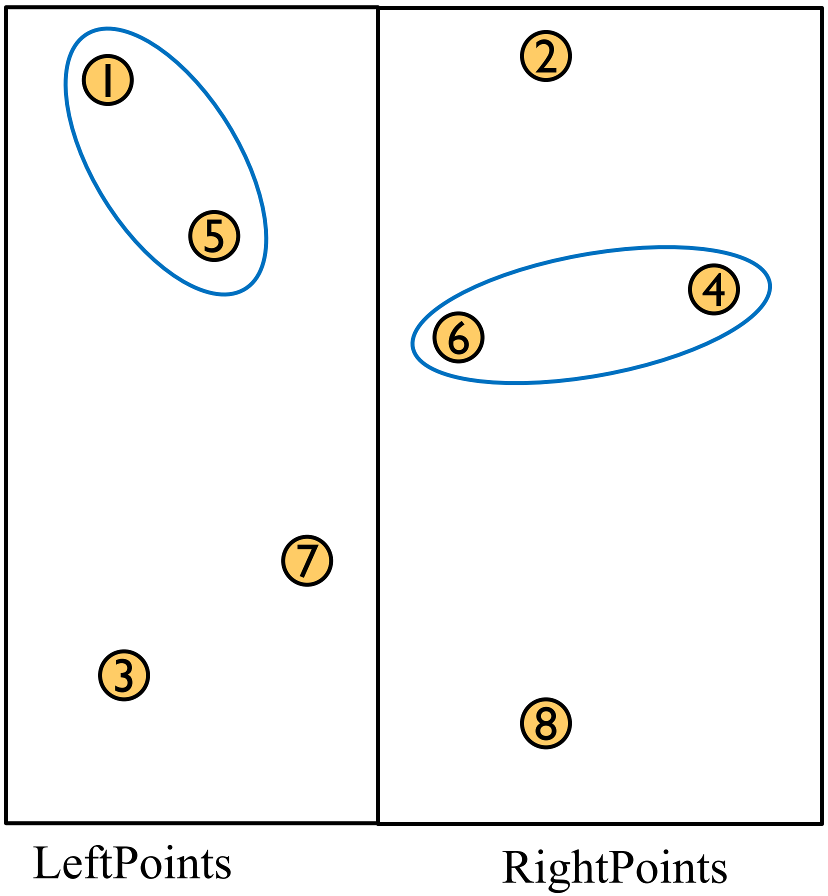
Divide:

At median x coordinate

Conquer:

Recursively find closest pairs from Left and Right

Combine:



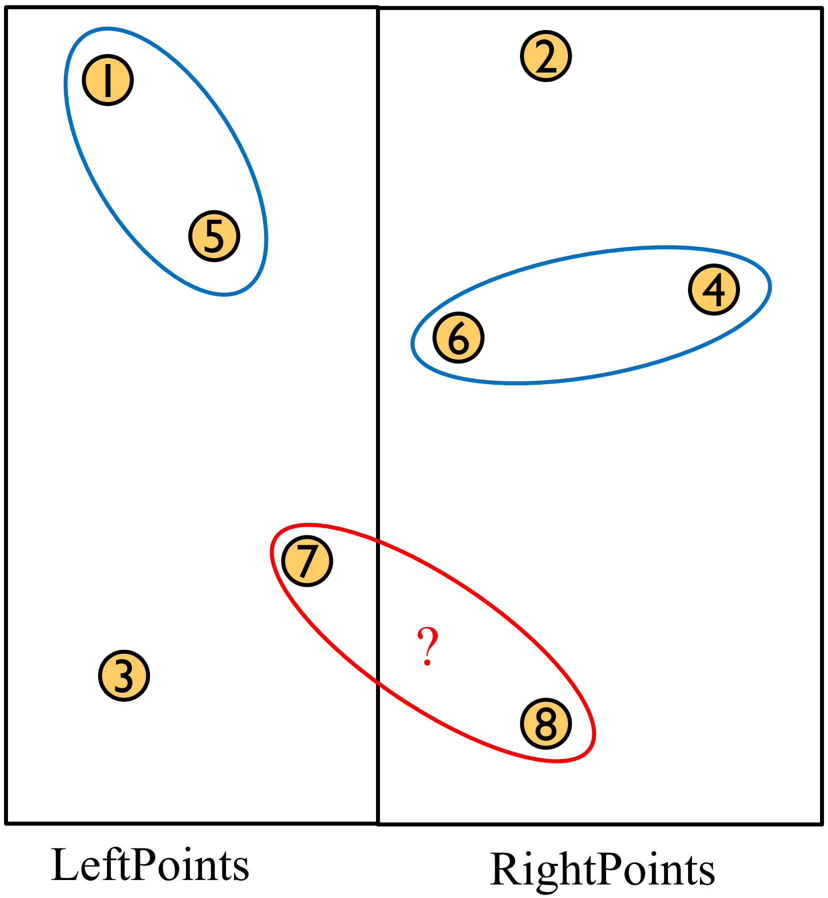
Closest Pair of Points: D&C

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

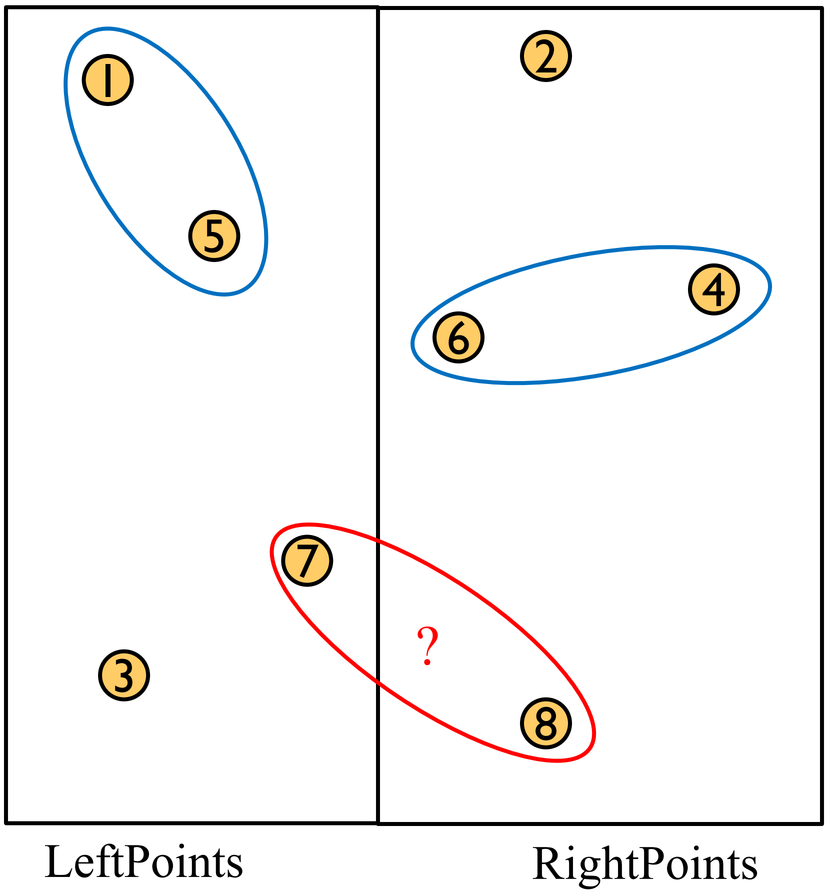
Combine:

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

Define "runway" or "strip" along the cut.

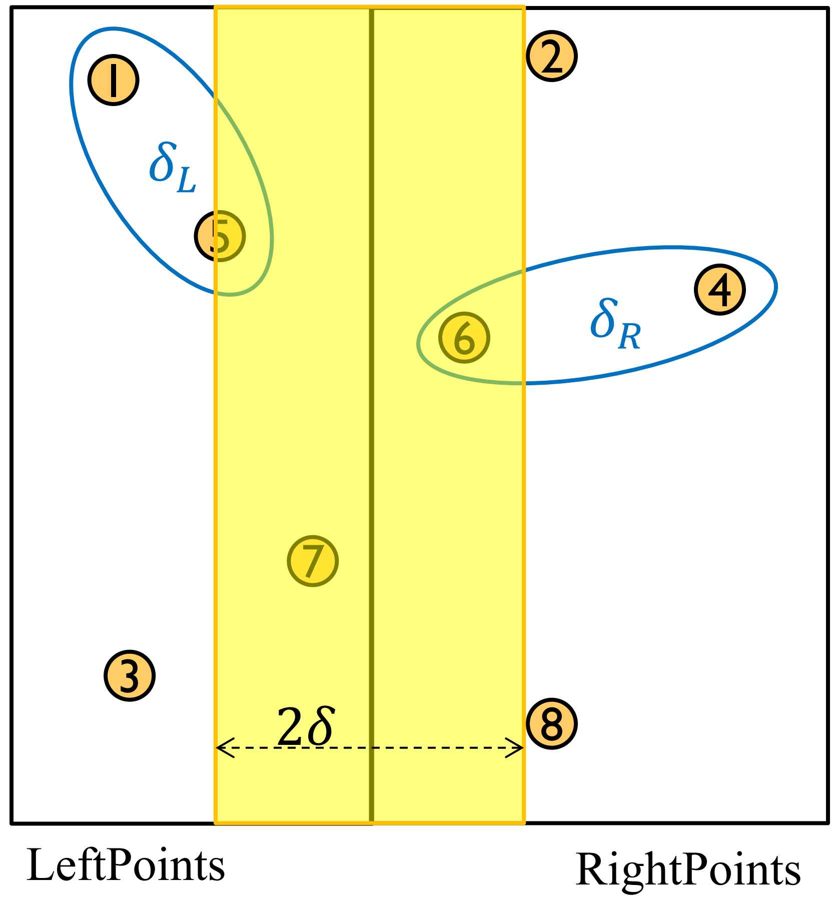
Combine:

2. Closest Pair Spanned our "Cut"

Need to test points across the cut.

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

How many are there?



Spanning the Cut

Combine:

2. Closest Pair Spanned our "Cut"

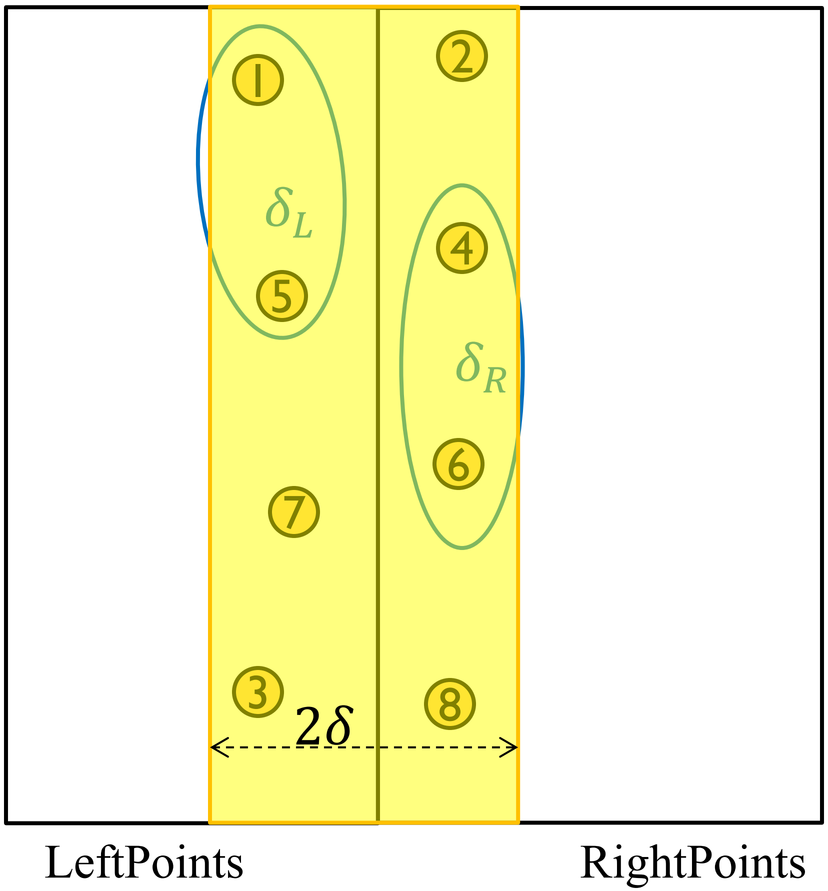
Need to test points across the cut

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

How many are there?

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

Define "runway" or "strip" along the cut.



Spanning the Cut

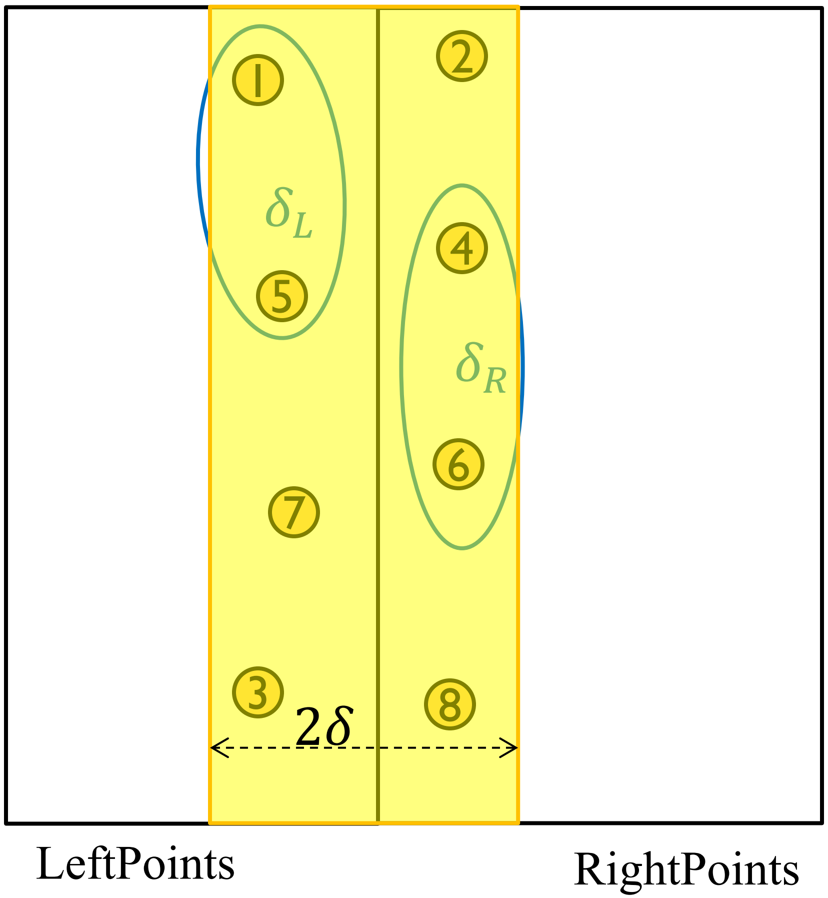
Combine:

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 any points that are > δ from one another



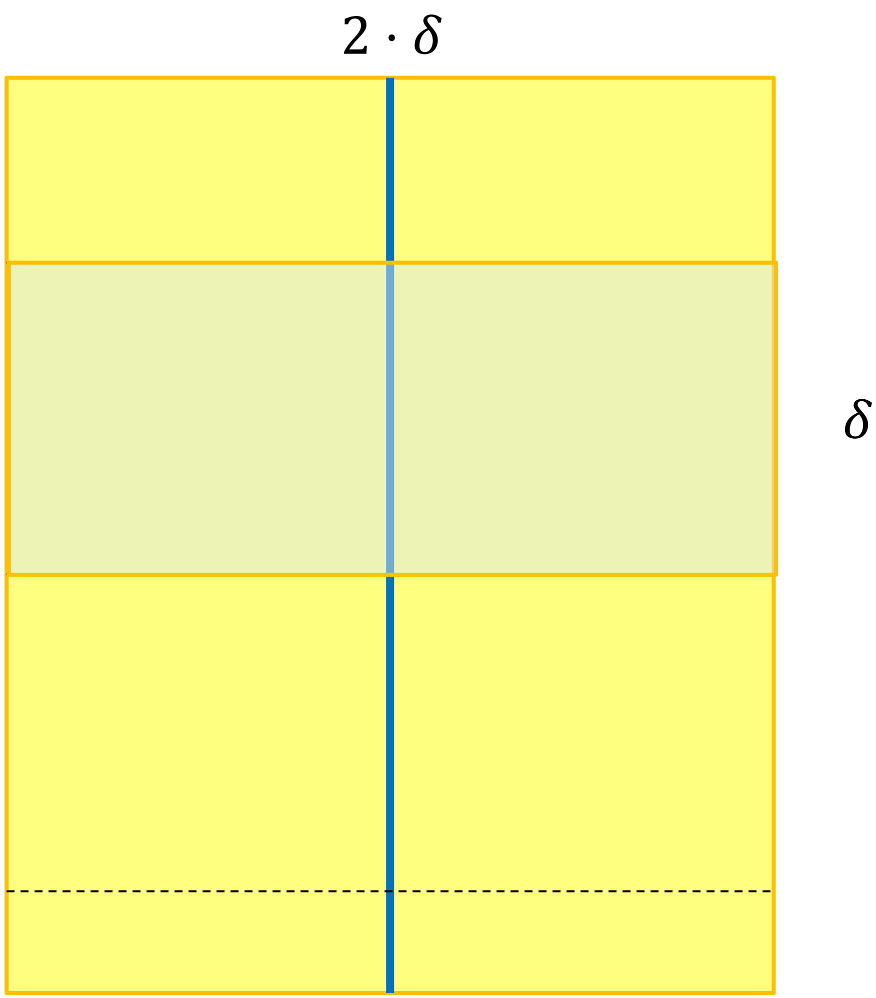
Reducing Search Space

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

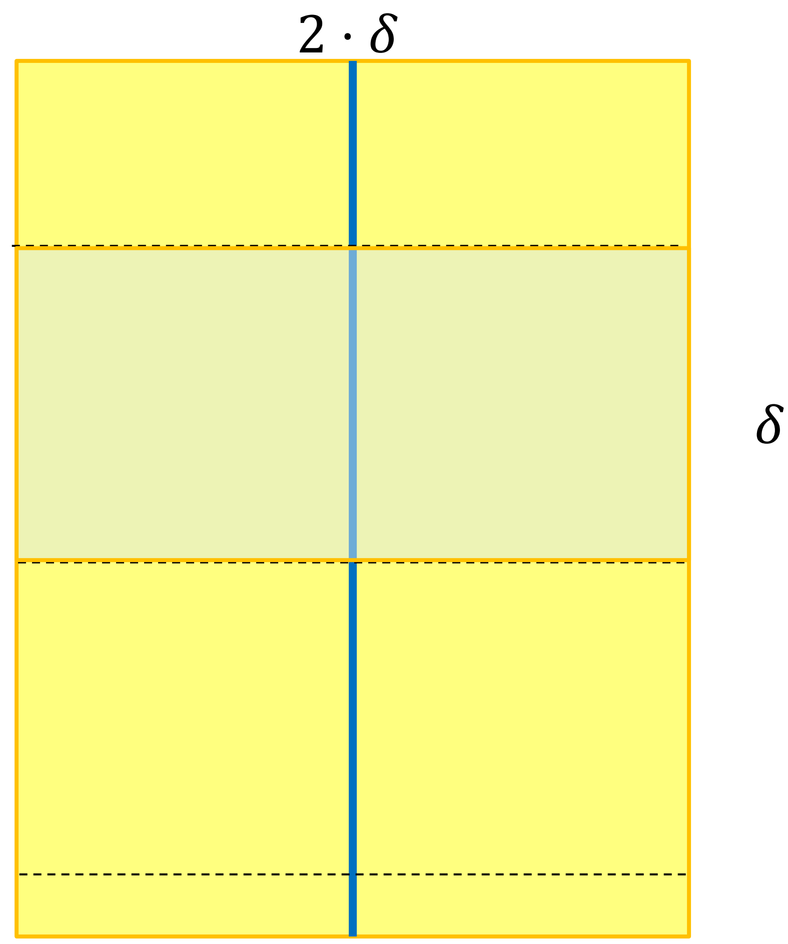
Claim #1: if two points are the closest pair that cross the cut, then you can surround them in a box that's $2 \cdot \delta$ wide by δ tall.

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 guaranteed 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.



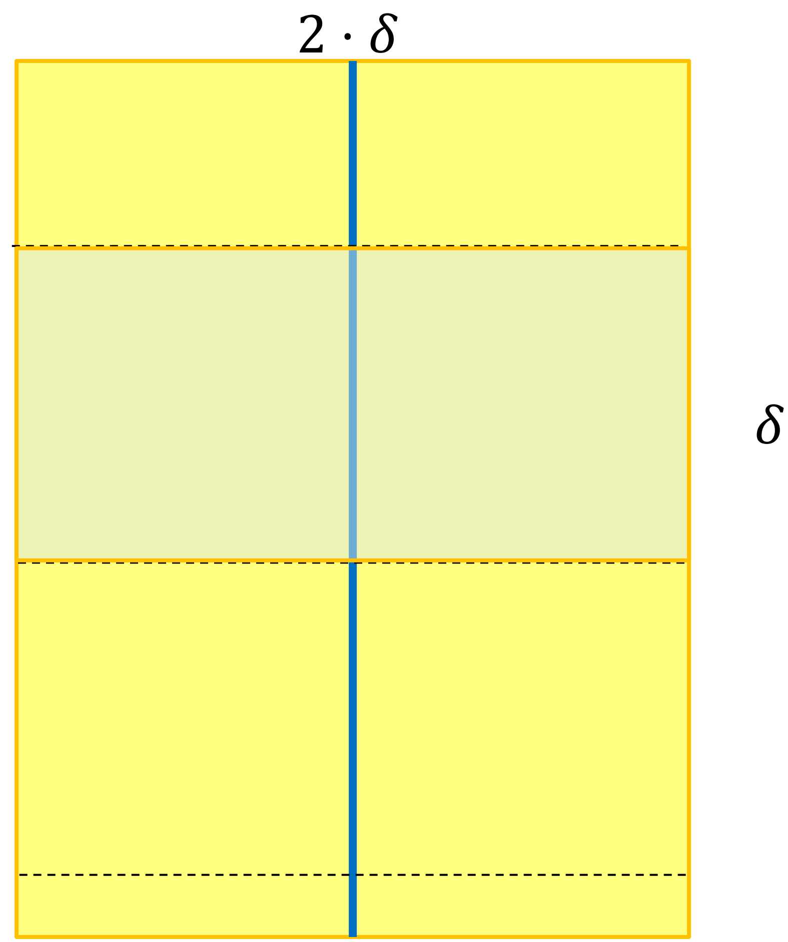
Reducing Search Space

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.

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.



Spanning the Cut

Combine:

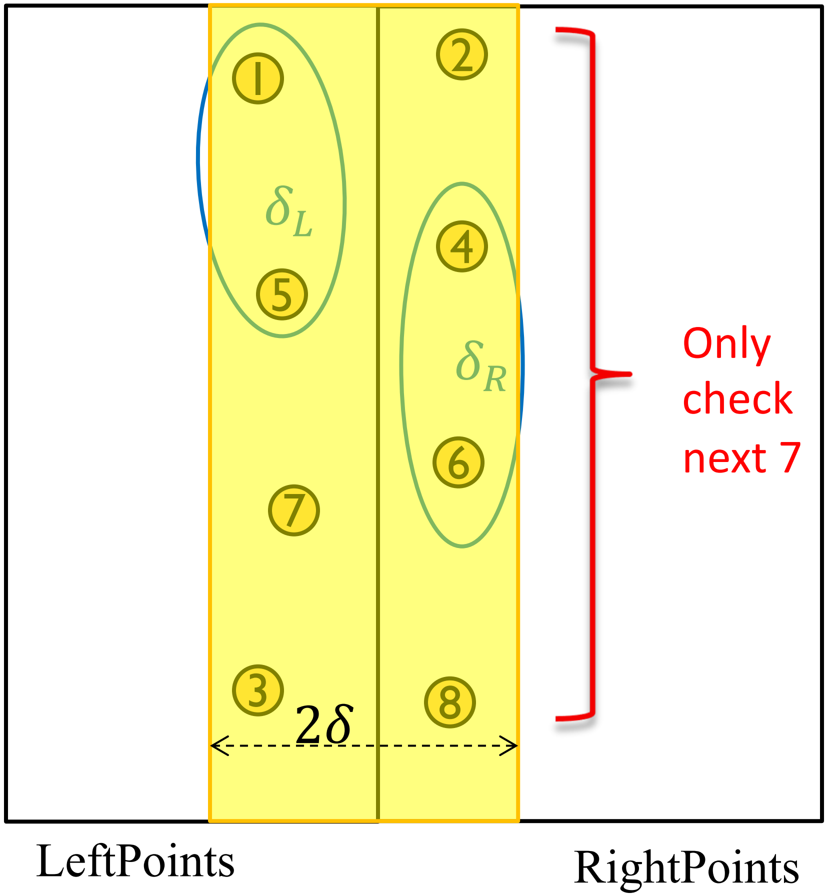
2. Closest Pair Spanned our "Cut"

Consider points in strip in increasing y-order.

For a given point p, we can *prove* the 8th point and beyond is more than δ from p.
(pp. 1041-2 in CLRS)

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

$\boldsymbol{\Theta(n)}$ ***Better*!**



Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate
(Later we'll also need to process points by y-coordinate, too.)

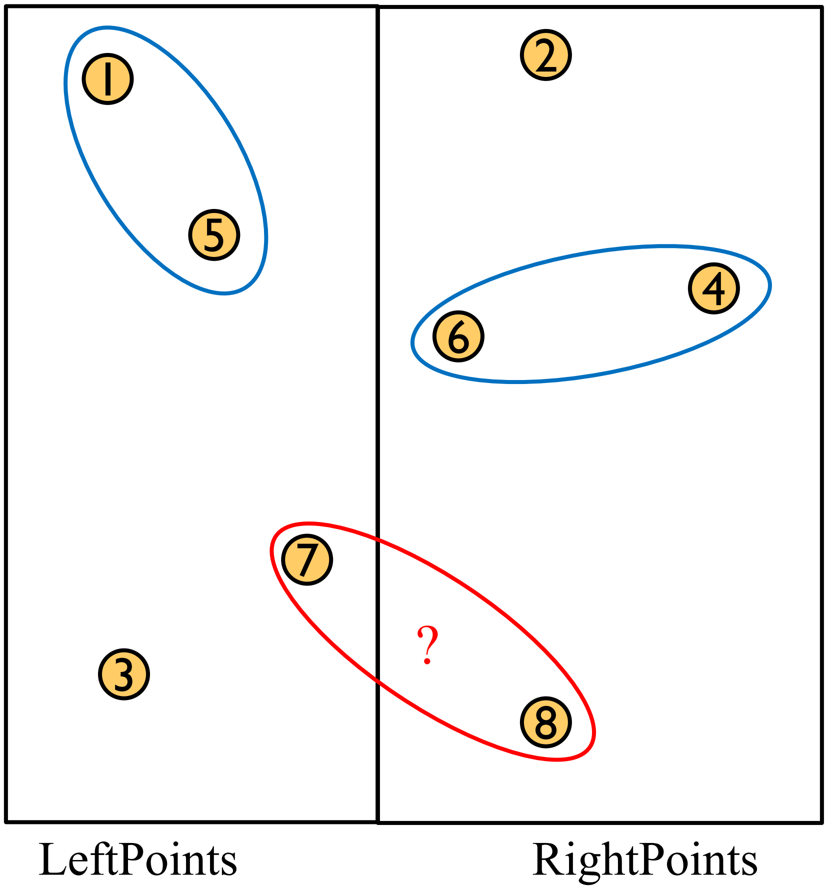
Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

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

Base case?

Combine:

- Consider only points in the runway (x-coordinate within distance δ of median)
- Process runway points by y-coordinate
- Compare each point in runway to 7 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?

$\Theta(n \log n)$

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

Case 2 of Master's Theorem

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

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

$T(n)$:

$\Theta(1)$ — **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

$\Theta(n)$ — **Combine:**
- Process runway points by y-coordinate and Compare each point in runway to 7 points above it and save the closest pair

$\Theta(1)$
- Output closest pair among left, right, and runway points

Summary for Closest Pair of Points

- Comparing all pairs is a brute-force fail
  - Except for small inputs
- Divide and conquer a big improvement
- Needed to find an efficient way for part of the combine step
  - Geometry came through for us here!
  - Only needed to look at constant number of points for each point in the strip
- Implementation subtleties
  - Don't want to sort the strip by y-coordinate in each recursive call
  - In initialization, create an "index" that lets you process all points in order by y-coordinate
  - (There are other ways to address this.)