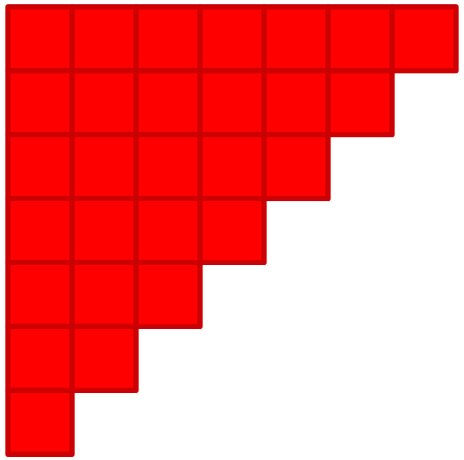
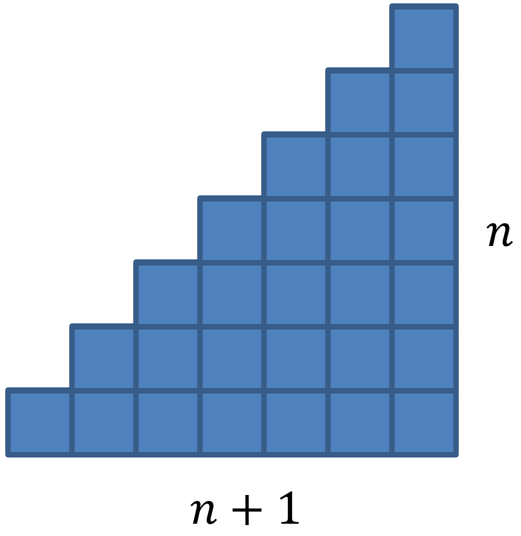
Warm up

Simplify:

$$1 + 2 + 3 + \cdots + (n-1) + n =$$

$$1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2}$$





Today's Keywords

- Divide and Conquer
- Closest Pair of Points
- Matrix Multiplication
- Strassen's Algorithm

CLRS Readings

- Chapter 4
- Chapter 33

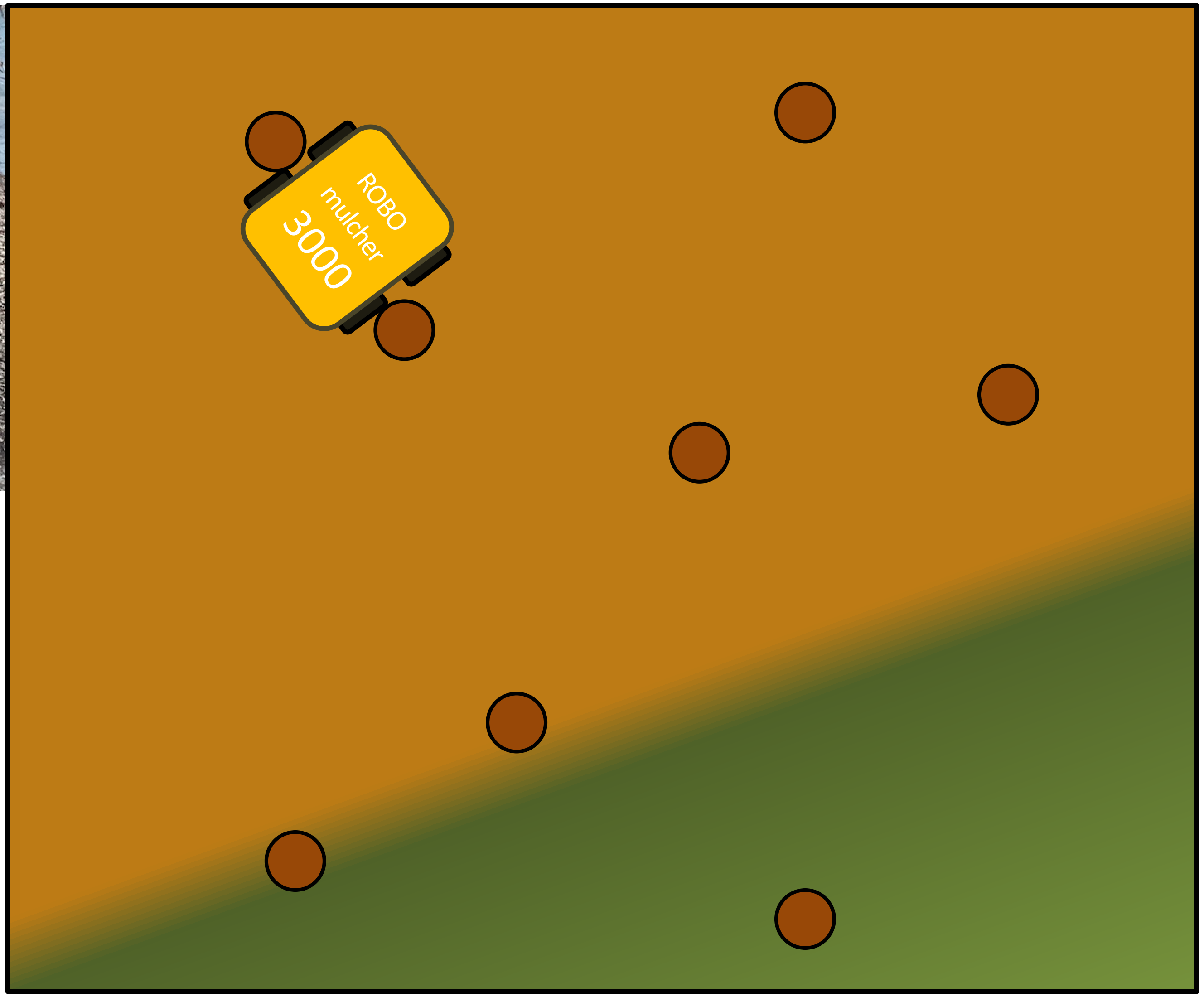
Homeworks

- HW2 due Thursday 2/6 at 11pm
  - Written (use Latex!) – Submit BOTH pdf and zip!
  - Asymptotic notation
  - Recurrences
  - Master Theorem
  - Divide and Conquer
- HW3 coming Thursday
  - Programming! (Java or Python 2/3)

Robbie's Yard

There has to be an easier way!

Constraints: Trees and Plants

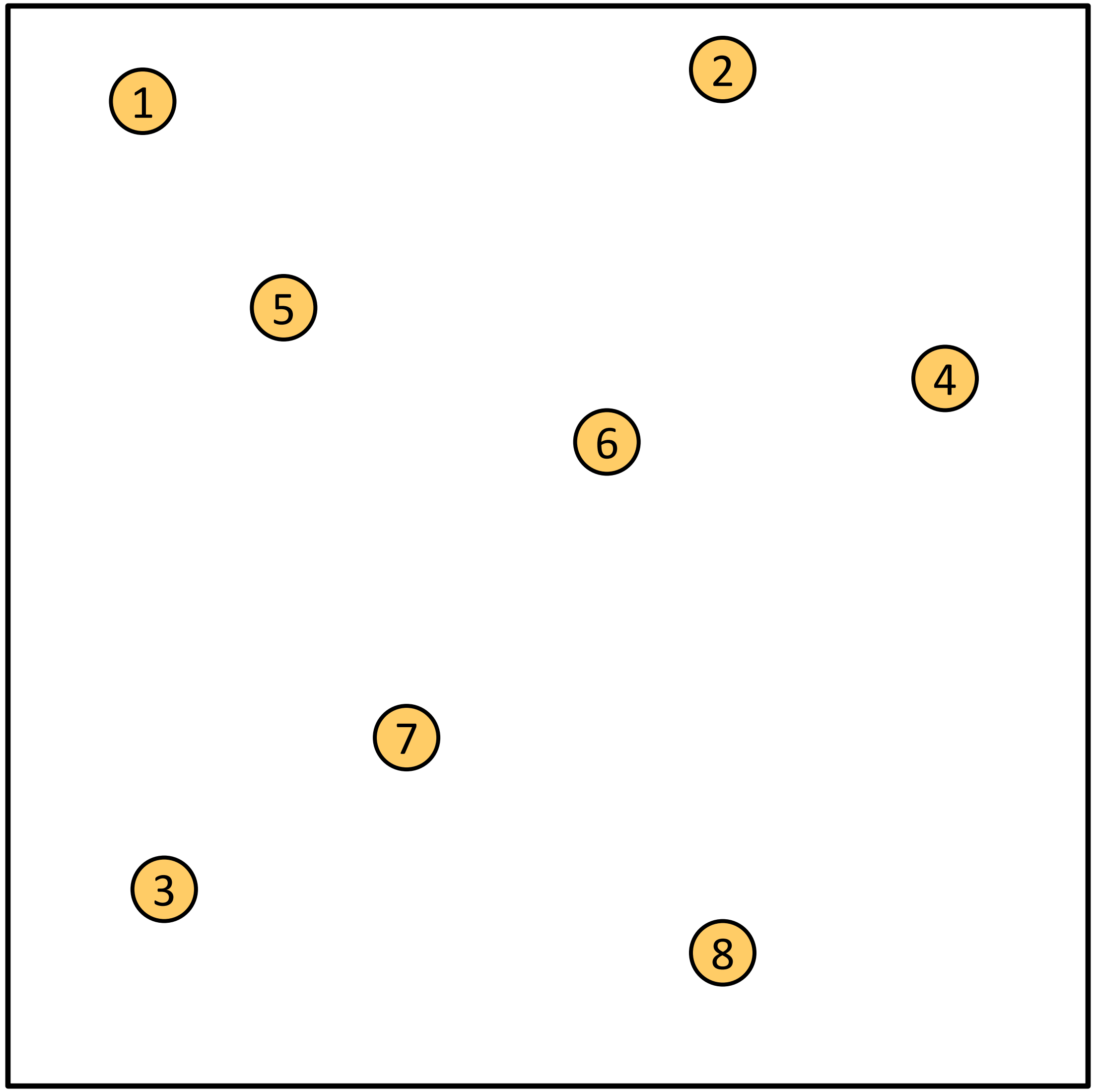


Need to find:
Closest Pair of Trees - how wide can the robot be?

Closest Pair of Points

Given:
A list of points

Return:
Pair of points with smallest distance apart



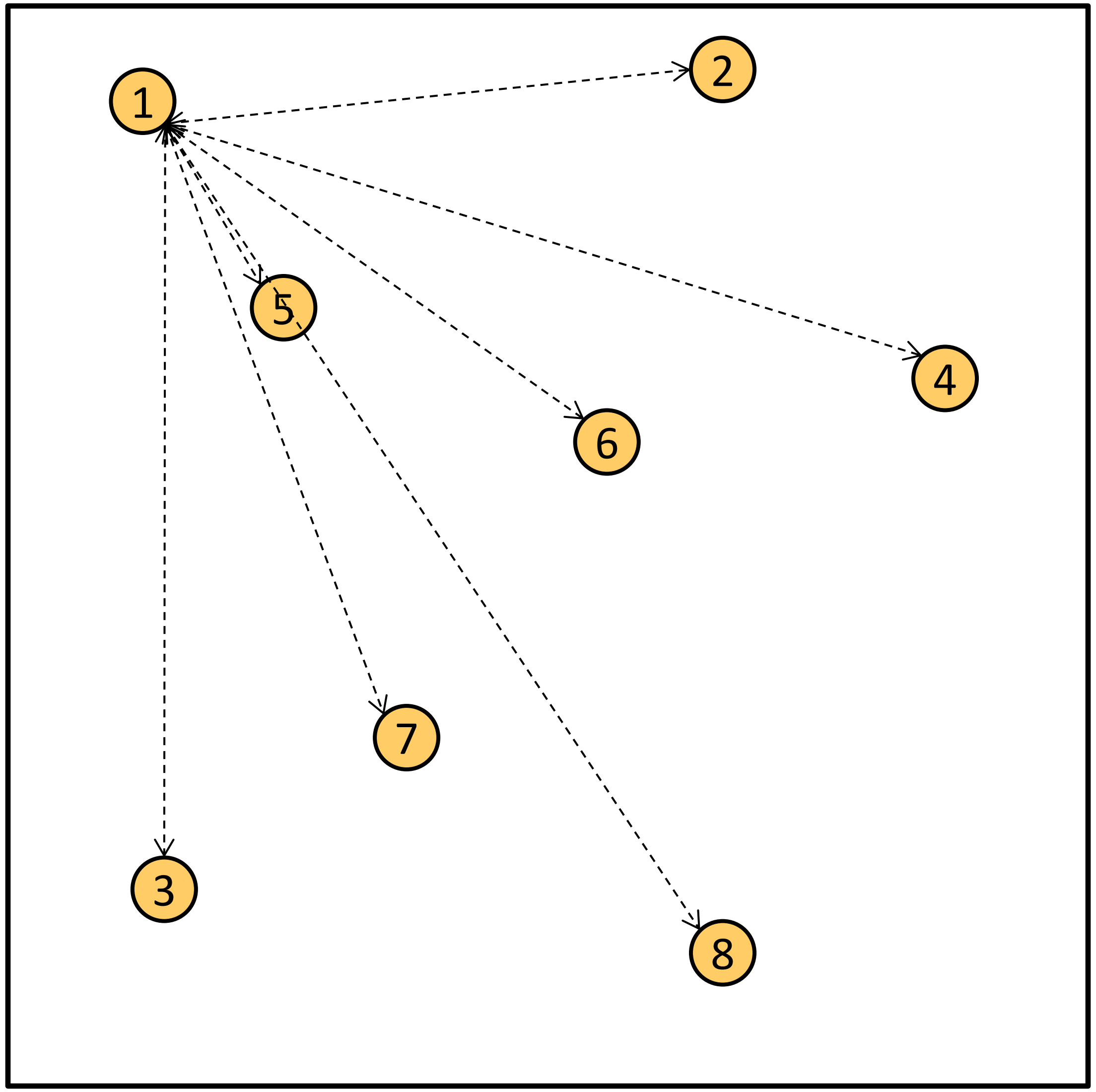
Closest Pair of Points: Naïve

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! $\Theta(n \log n)$



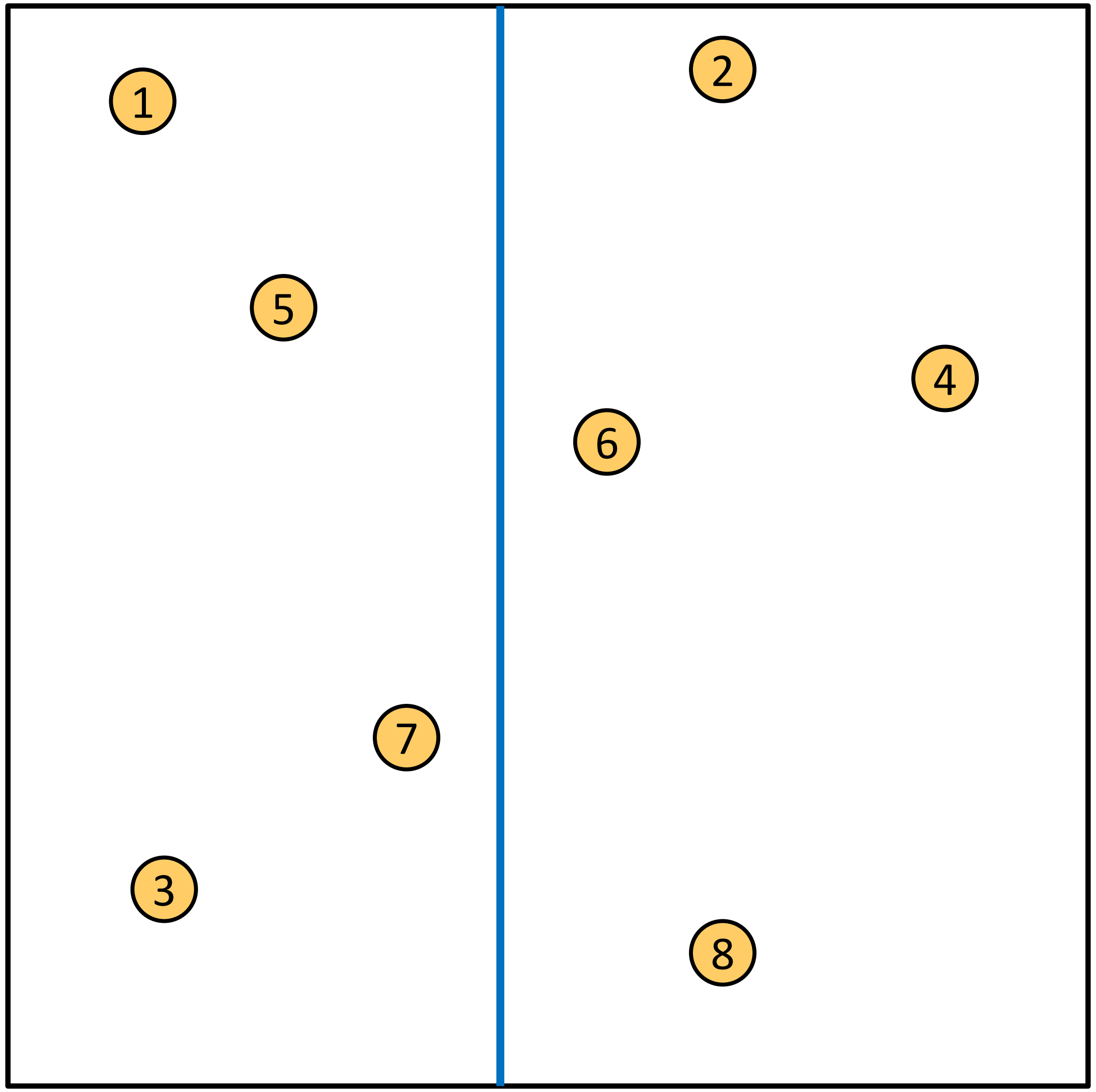
Closest Pair of Points: D&C

Divide: How?

At median x coordinate

Conquer:

Combine:



Closest Pair of Points: D&C

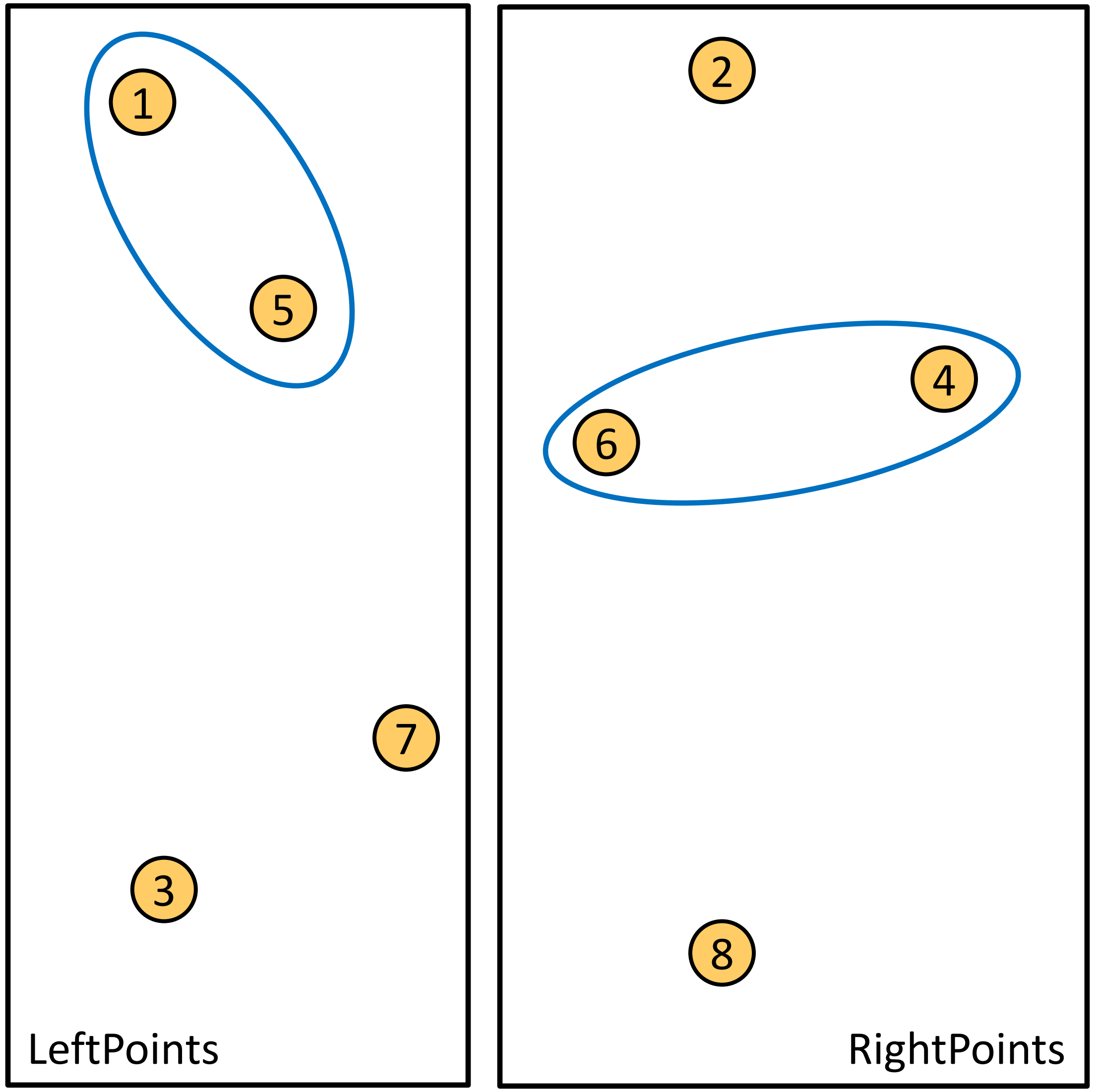
Divide:

At median x coordinate

Conquer:

Recursively find closest pairs from Left and Right

Combine:

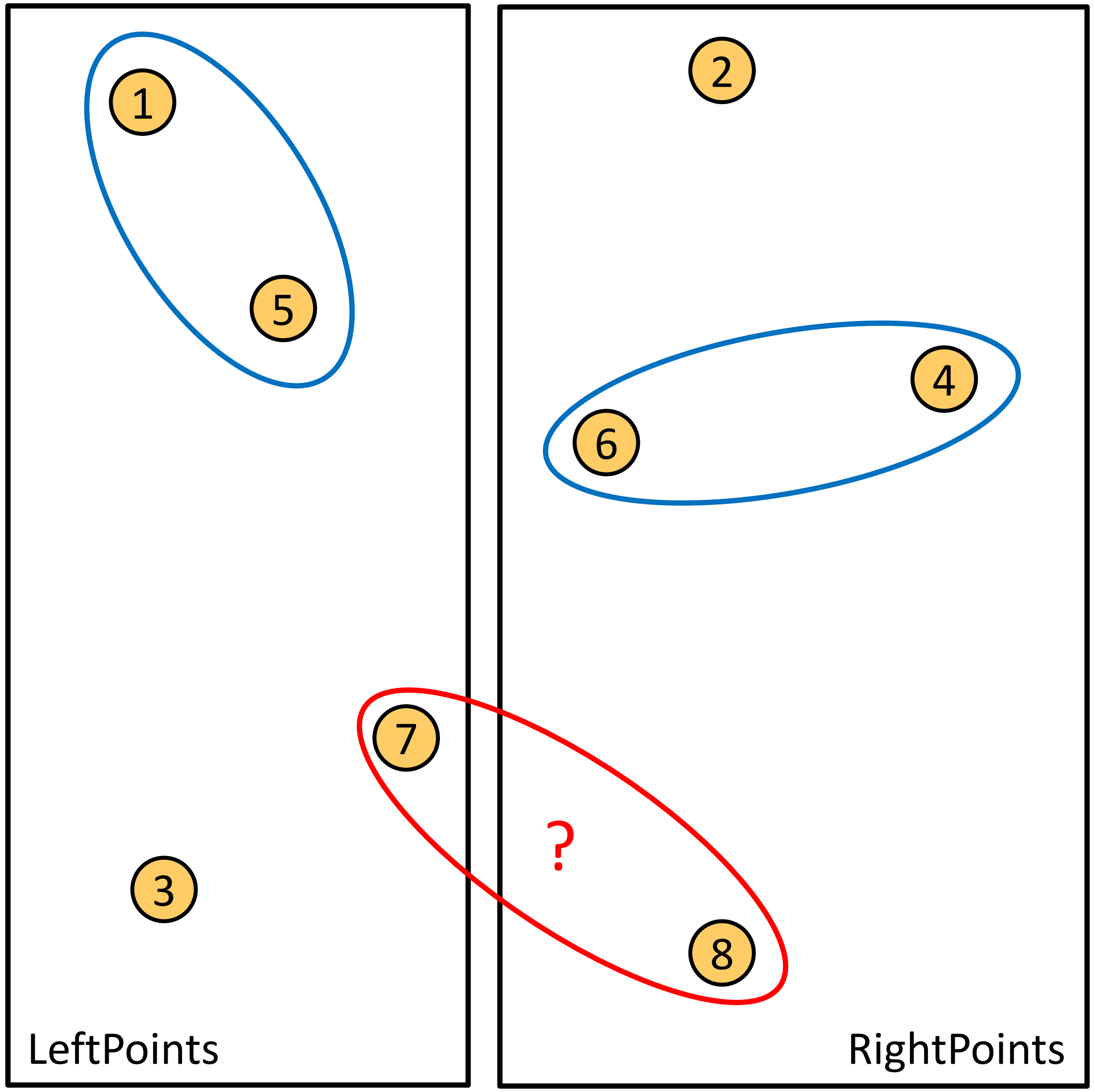


Closest Pair of Points: D&C

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

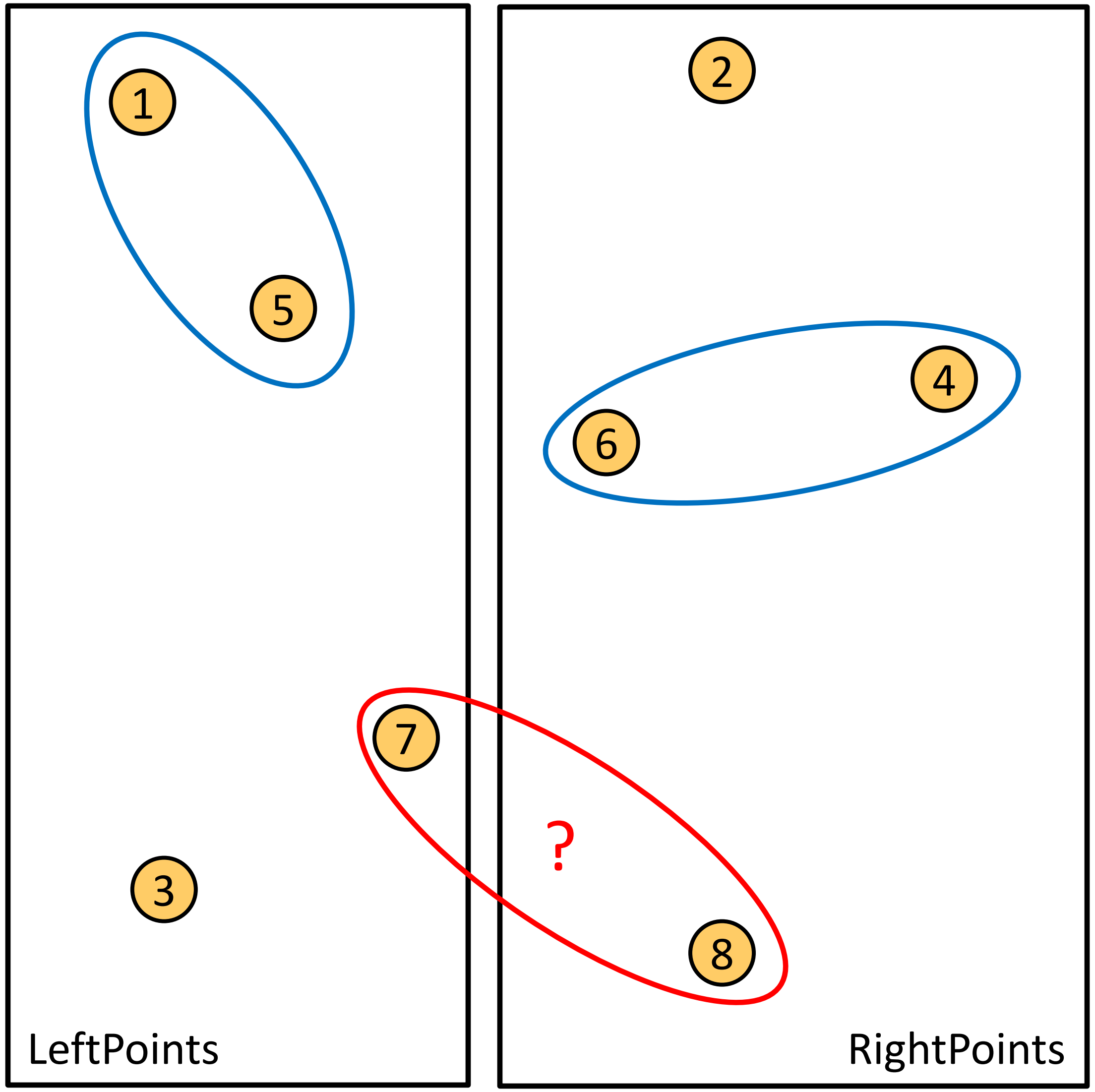
Combine:

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

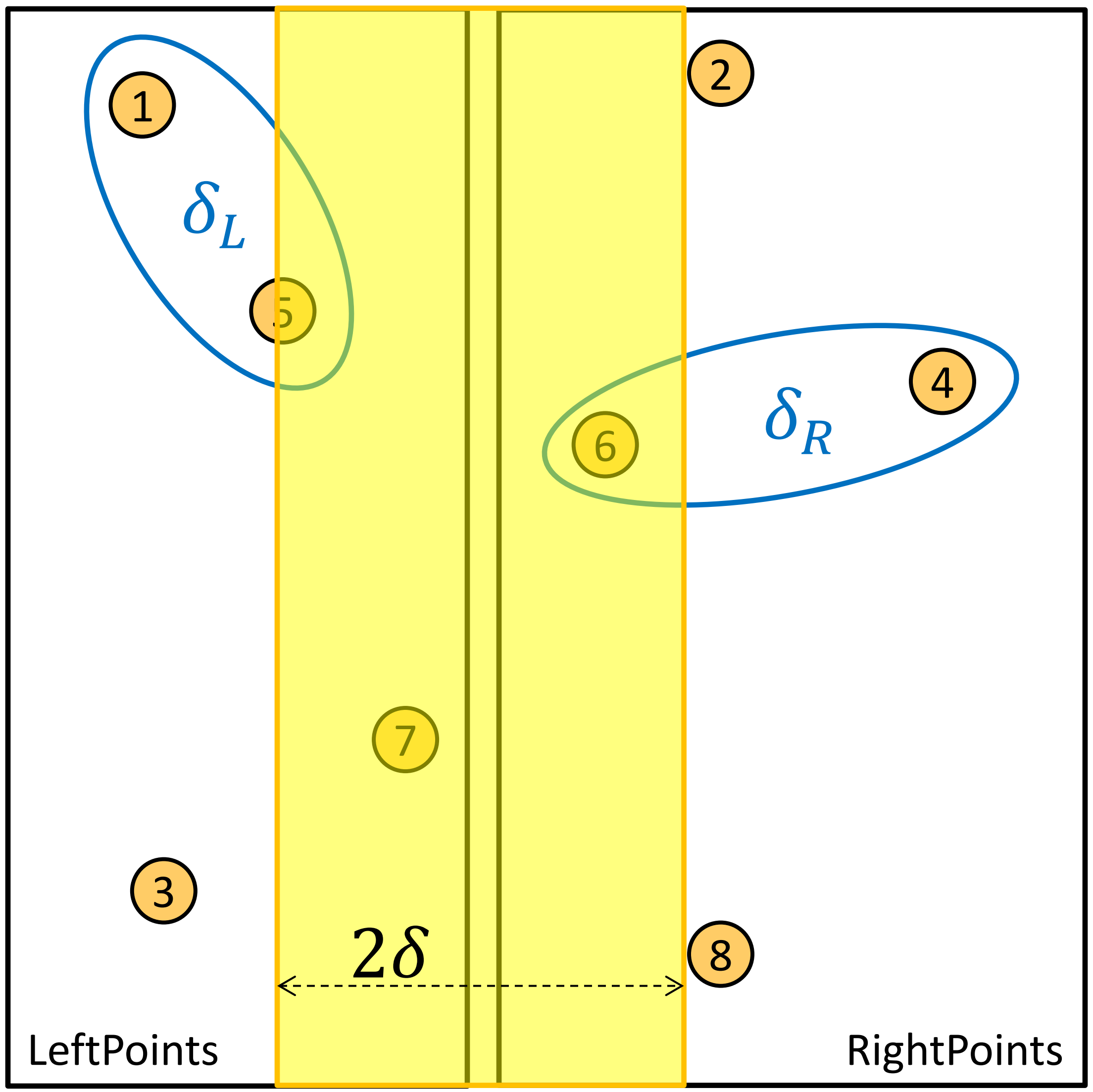
Combine:

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.

How many are there?



Spanning the Cut

Combine:

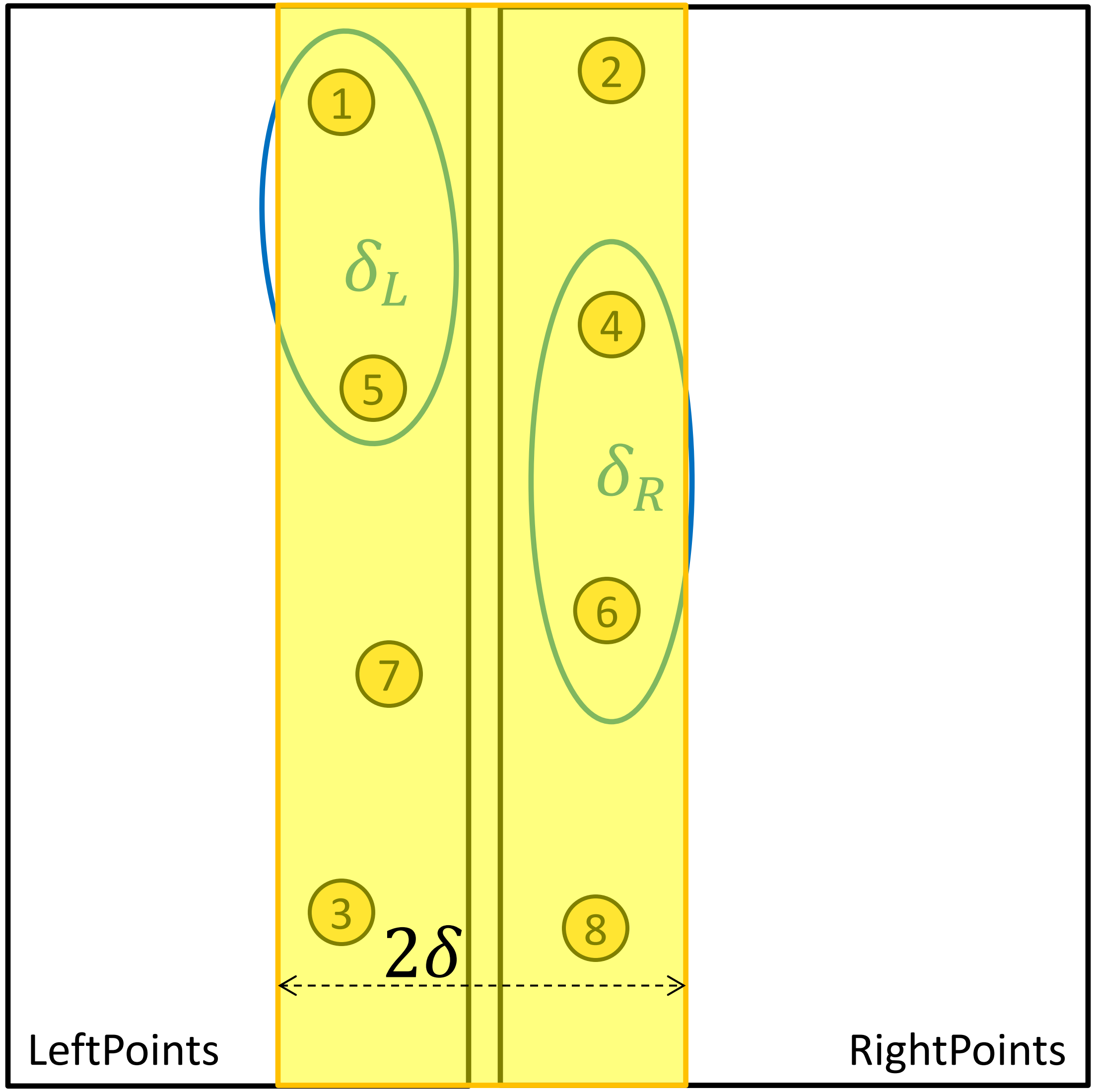
2. Closest Pair Spanned our "Cut"

Need to test points across the cut

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

How many are there?

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



Spanning the Cut

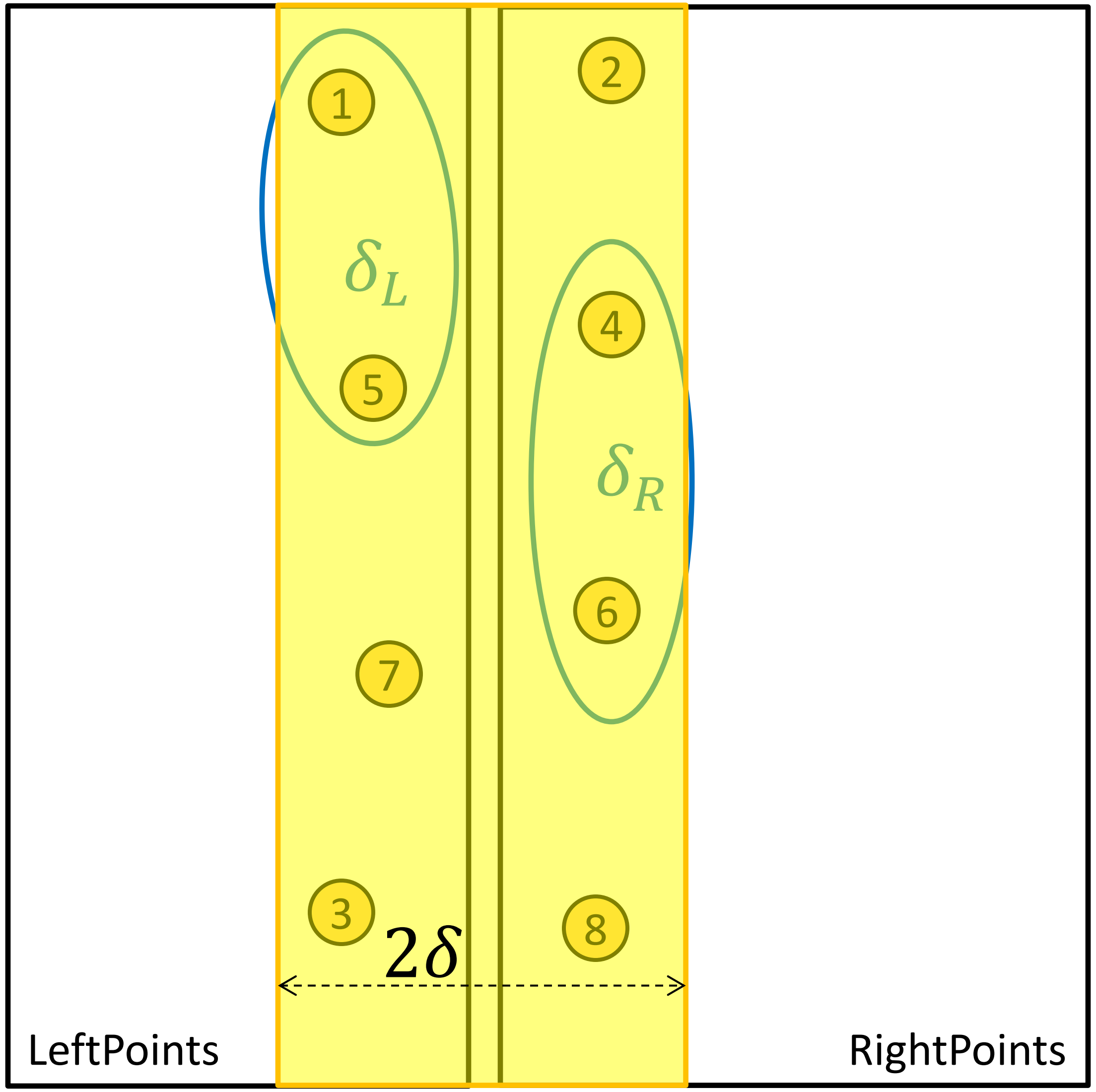
Combine:

2. Closest Pair Spanned our “Cut”

Need to test points across the cut

We don’t need to test all pairs!

Only need to test points within δ of one another



Reducing Search Space

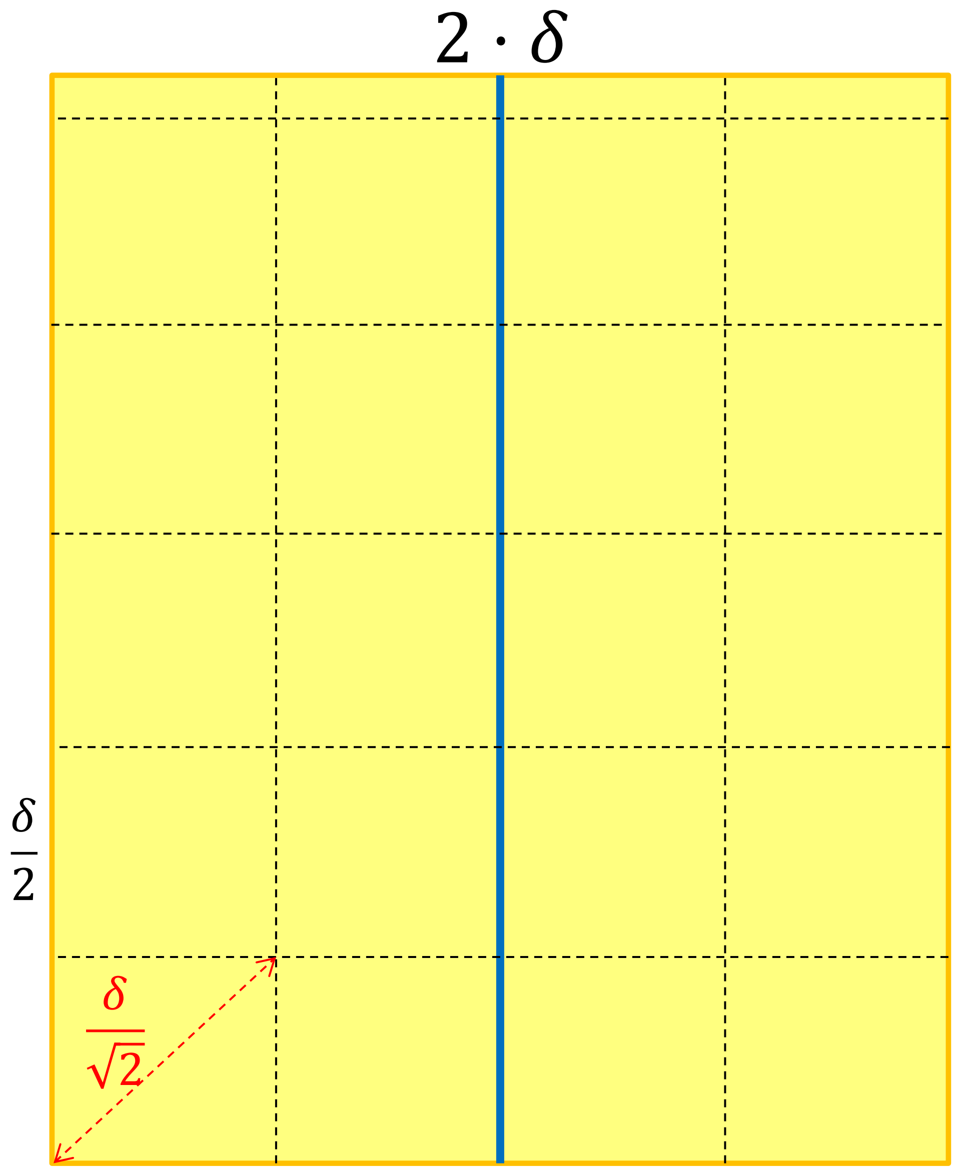
Combine:

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:

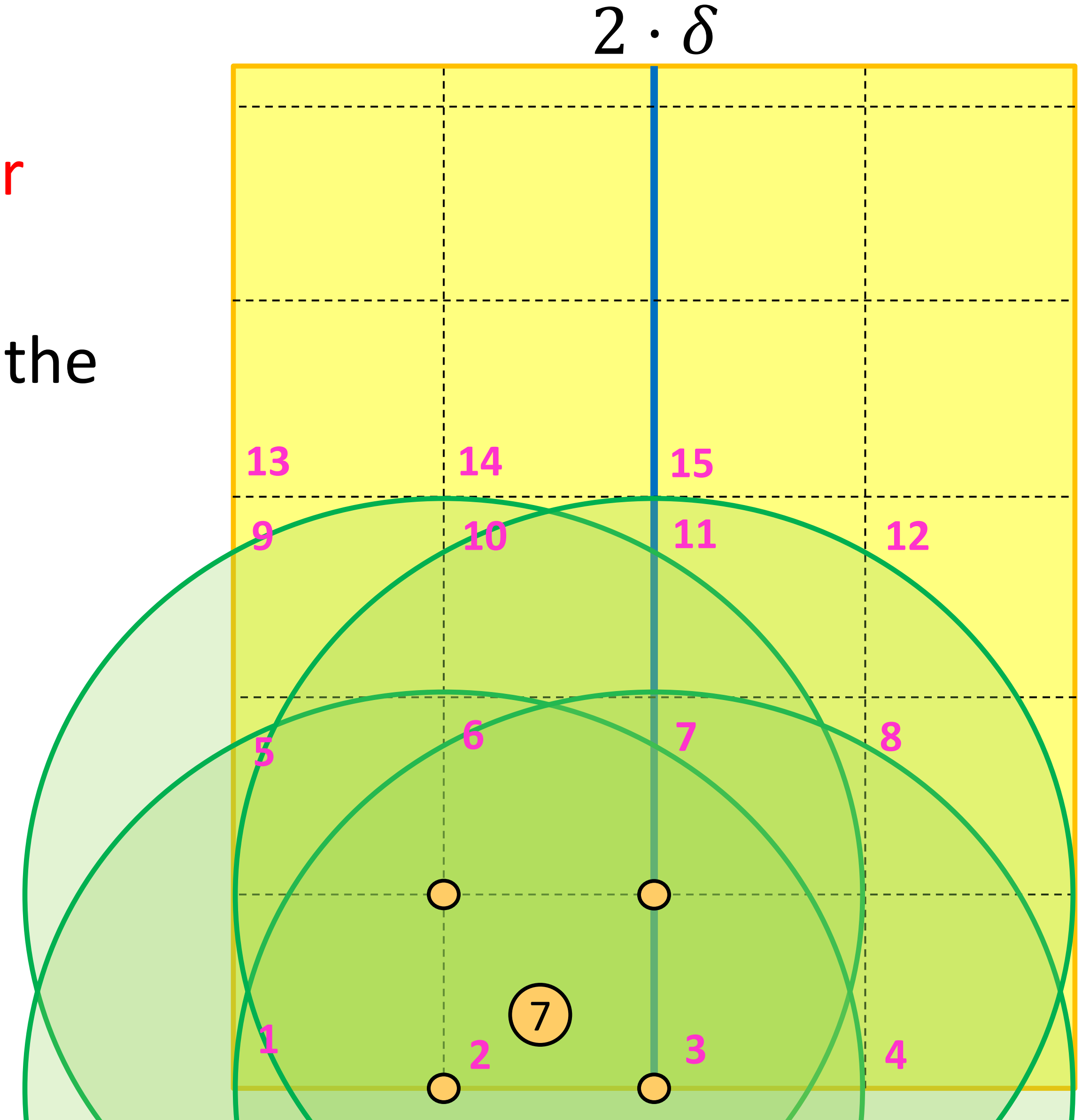
2. Closest Pair Spanned our “Cut”

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



Closest Pair of Points: Divide and Conquer

Initialization: Sort points by x-coordinate

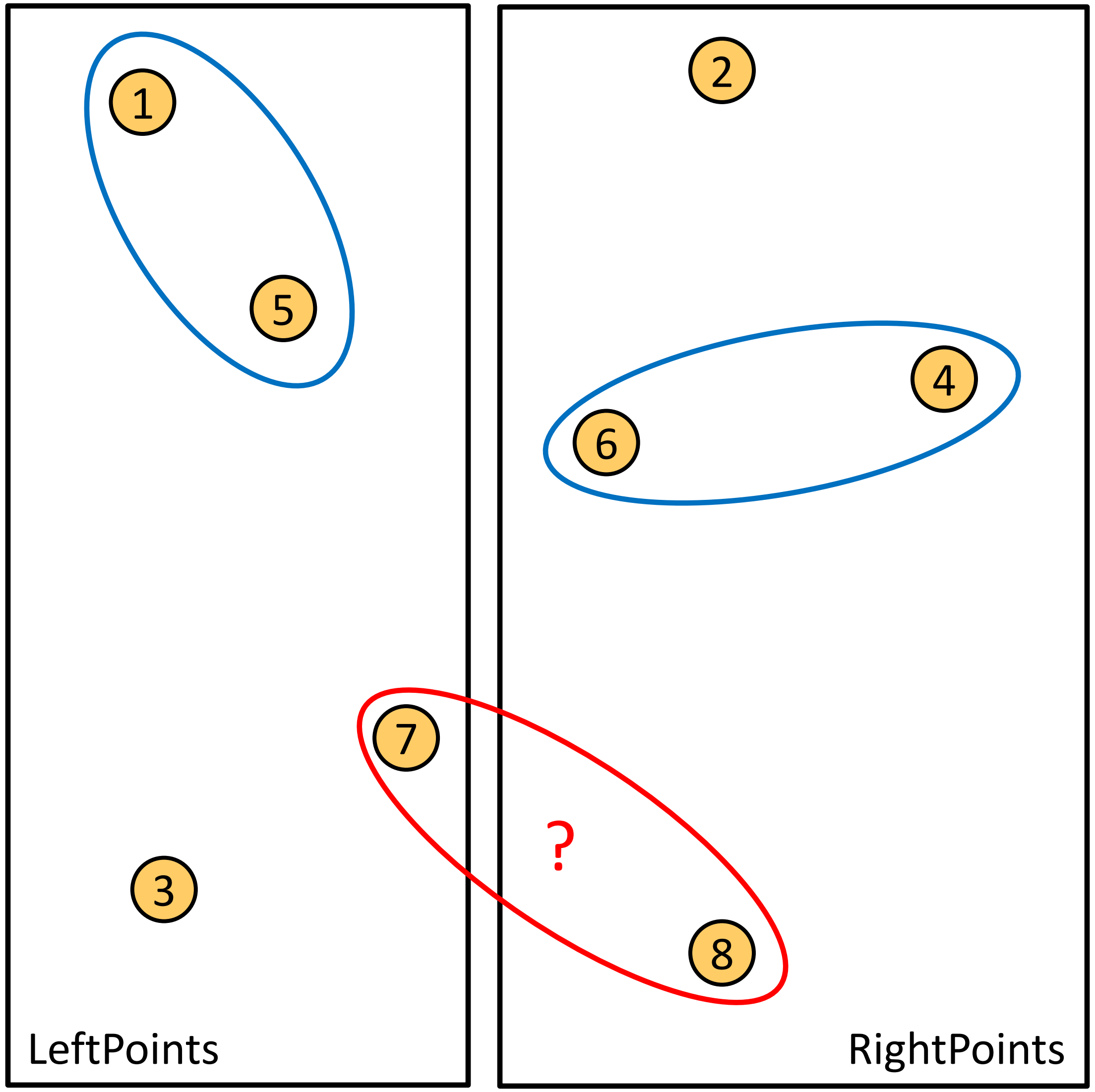
Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

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 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points



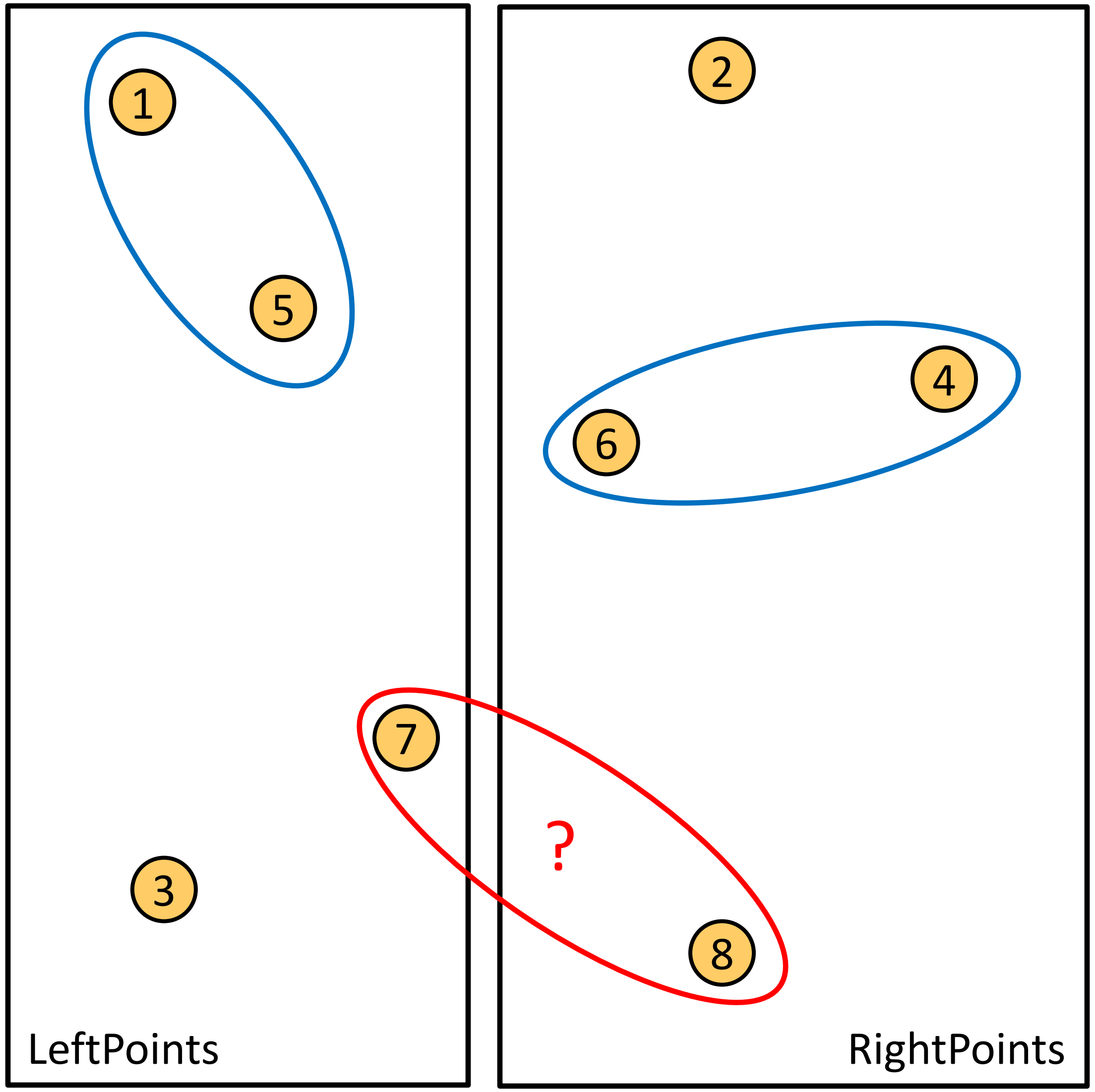
Closest Pair of Points: Divide and Conquer

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 i[n run]way (x-coordinate within distan[ce] δ of median)
- Sort runway points by y-coordinate
- Compare each point in runway to 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 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

Solution: Maintain additional information in the recursion

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

Sorting runway points by y-coordinate now becomes a **merge**

Listing Points in the Runway

Output on Left:

Closest Pair: $(1, 5)$, $\delta_{1,5}$

Sorted Points: $[3,7,5,1]$

Output on Right:

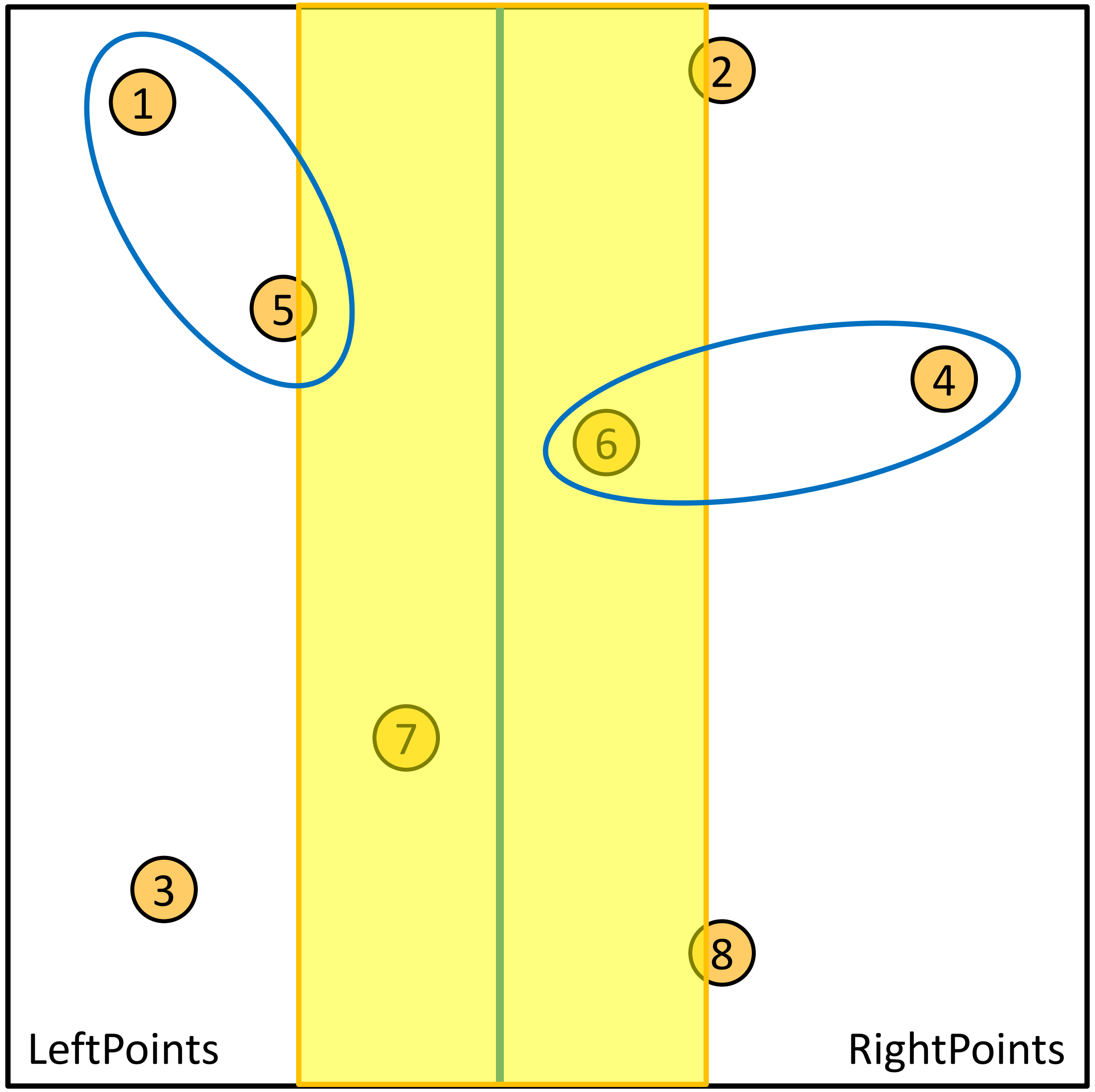
Closest Pair: $(4,6)$, $\delta_{4,6}$

Sorted Points: $[8,6,4,2]$

Merged Points: $[8,3,7,6,4,5,1,2]$

Runway Points: $[8,7,6,5,2]$

Both of these lists can be computed by a *single* pass over the lists



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 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

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

Combine:

- 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 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?

$\Theta(n \log n)$

$T(n)$

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

Case 2 of Master's Theorem

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

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

$\Theta(1)$ **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:

- $\Theta(n)$ Merge sorted list of points by y-coordinate and construct list of points in the runway (sorted by y-coordinate)
- $\Theta(n)$ Compare each point in runway to 15 points above it and save the closest pair
- $\Theta(1)$ Output closest pair among left, right, and runway points

Matrix Multiplication

$$\begin{array}{cc} & n \\ n & \begin{bmatrix} \boxed{1 \quad 2 \quad 3} \\ 4 \quad 5 \quad 6 \\ 7 \quad 8 \quad 9 \end{bmatrix} \end{array} \times \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

$$= \begin{bmatrix} 2+16+42 & 4+20+48 & 6+24+54 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$= \begin{bmatrix} 60 & 72 & 84 \\ 132 & 162 & 192 \\ 204 & 252 & 300 \end{bmatrix}$$

Run time? $O(n^3)$

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

Divide:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 \\ b_9 & b_{10} & b_{11} & b_{12} \\ b_{13} & b_{14} & b_{15} & b_{16} \end{bmatrix}$$

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \qquad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Combine:

$$AB = \begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}$$

Run time? $T(n) = 8T\left(\frac{n}{2}\right) + \boxed{4\left(\frac{n}{2}\right)^2}$ Cost of additions

Matrix Multiplication D&C

$$T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2$$

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

$$a = 8, b = 2, f(n) = n^2$$

Case 1!

$$n^{\log_b a} = n^{\log_2 8} = n^3$$

$$T(n) = \Theta(n^3)$$

We can do better…

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \qquad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$$AB = \boxed{\begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}}$$

Idea: Use a Karatsuba-like technique on this

Strassen's Algorithm

Multiply $n \times n$ matrices (A and B)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \qquad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Calculate:

$$Q_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$
$$Q_2 = (A_{2,1} + A_{2,2})B_{1,1}$$
$$Q_3 = A_{1,1}(B_{1,2} - B_{2,2})$$
$$Q_4 = A_{2,2}(B_{2,1} - B_{1,1})$$
$$Q_5 = (A_{1,1} + A_{1,2})B_{2,2}$$
$$Q_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$Q_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

Find AB:

$$\begin{bmatrix} Q_1 + Q_4 - Q_5 + Q_7 & Q_3 + Q_5 \\ Q_2 + Q_4 & Q_1 - Q_2 + Q_3 + Q_6 \end{bmatrix}$$

$$\begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}$$

Number Mults.: 7 Number Adds.: 18

$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2$$

Strassen's Algorithm

$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2$$

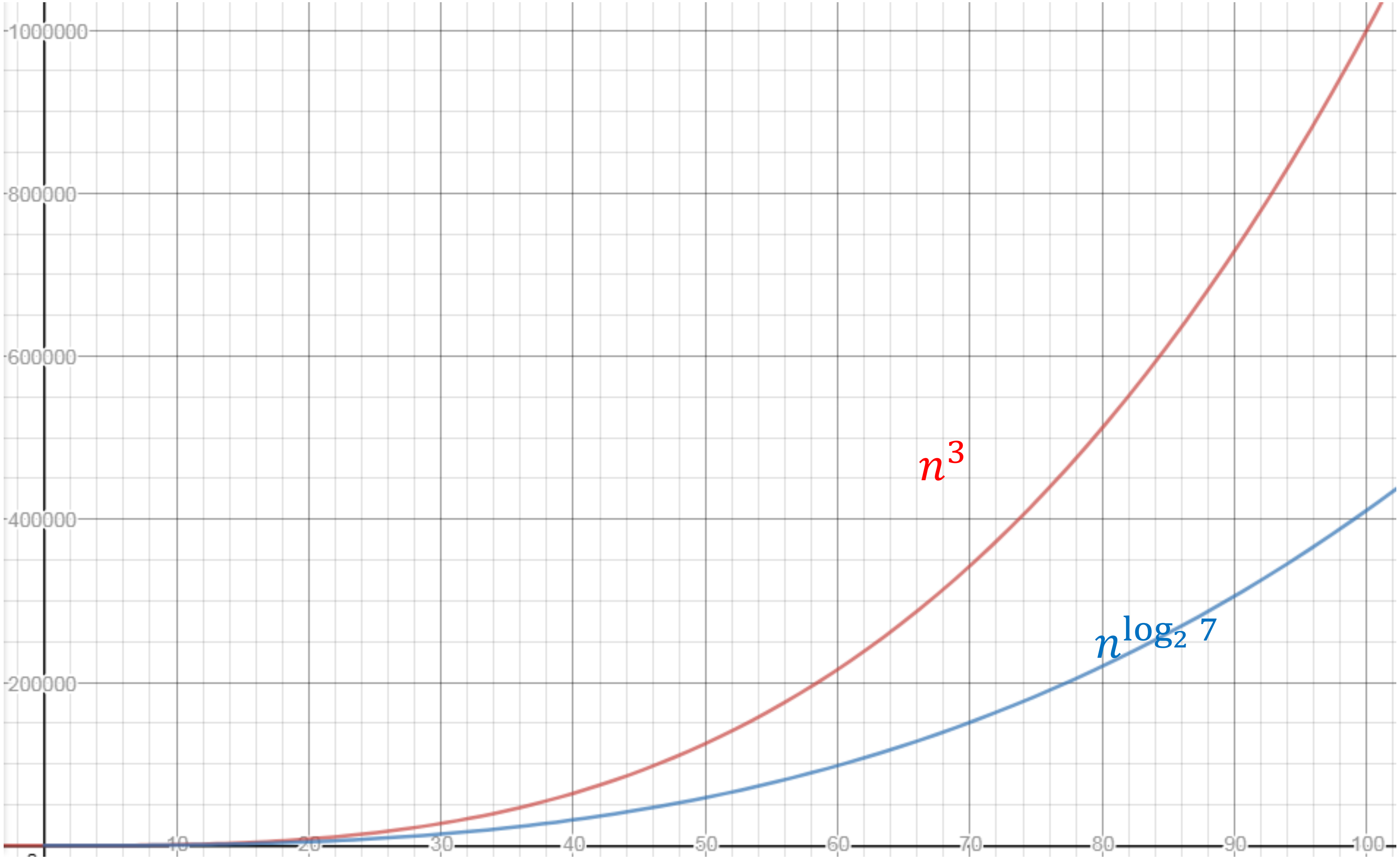
$$a = 7, b = 2, f(n) = \frac{9}{2}n^2$$

Case 1!

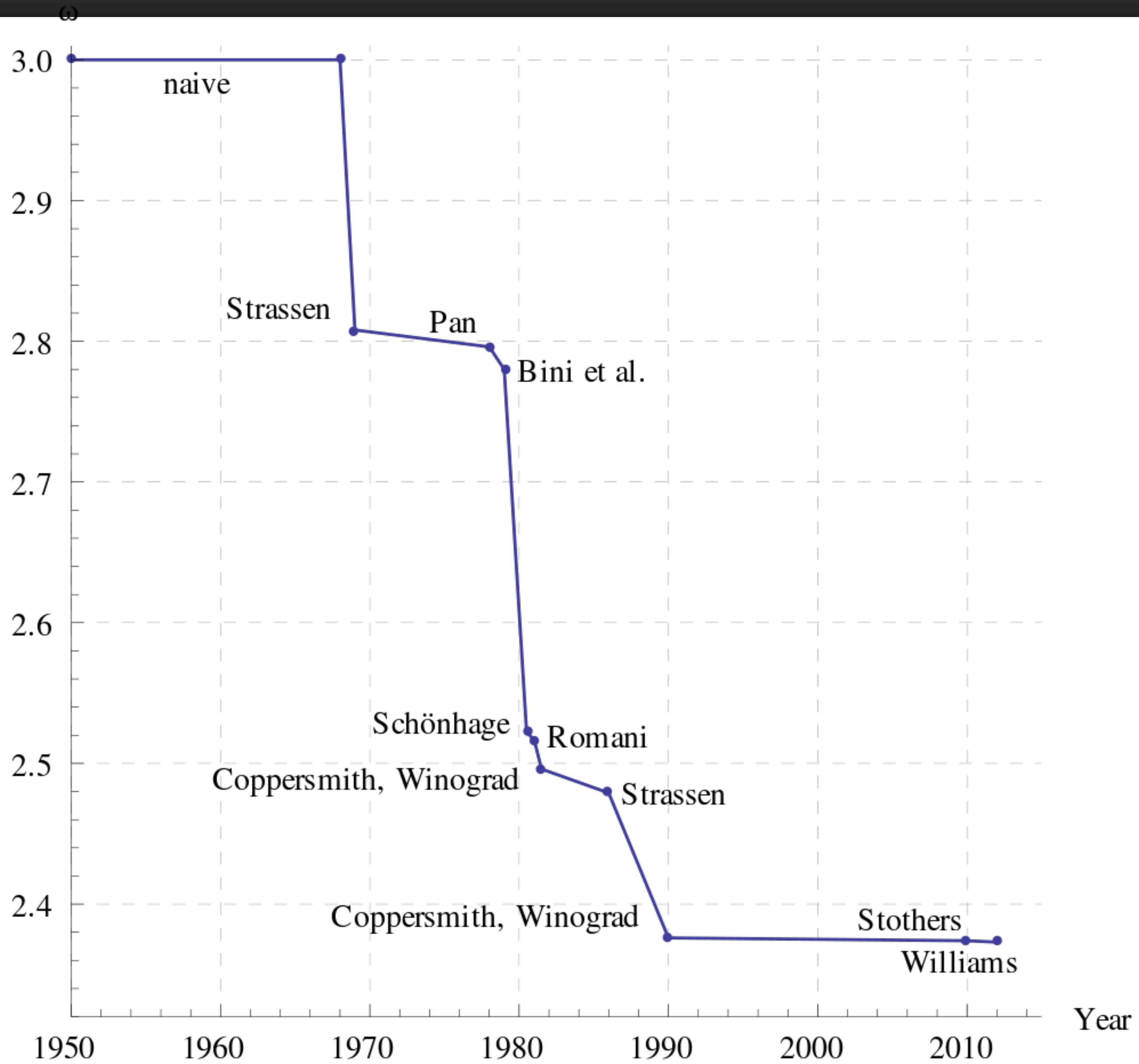
$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.807}$$

$$T(n) = \Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2.807})$$

Strassen's Algorithm



Is this the fastest?



Best possible is unknown

May not even exist!