CS 4102: Algorithms

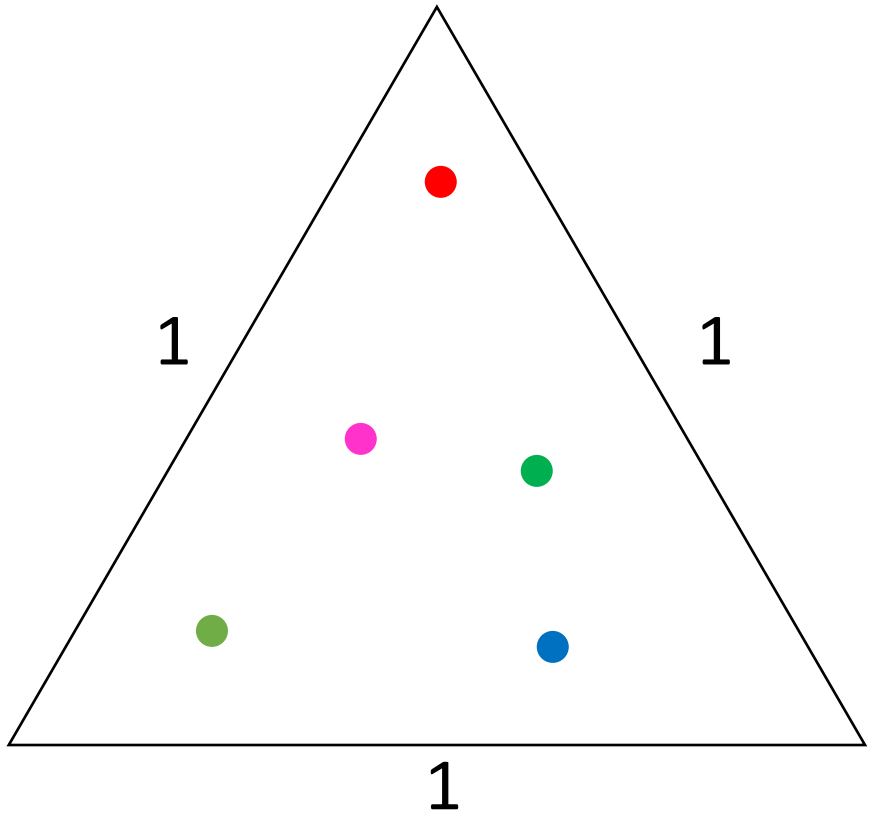
Lecture 6: Closest Pair of Points

David Wu

Fall 2019

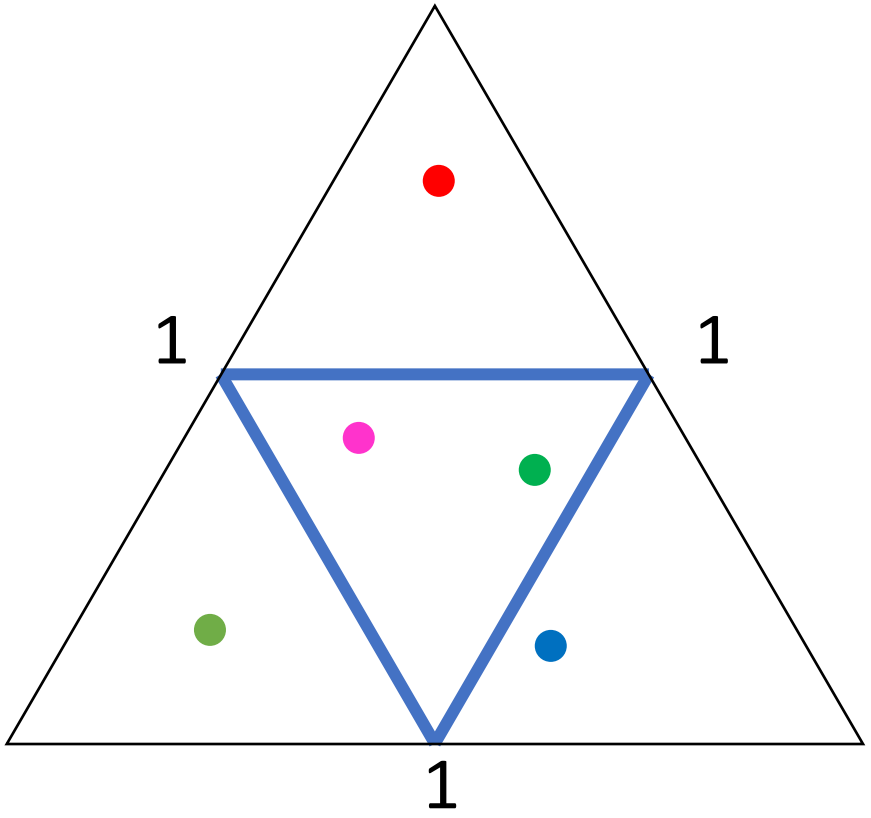
Warm Up

Given 5 points on the unit equilateral triangle, show there's always a pair of distance $\leq \frac{1}{2}$ apart



Warm Up

Given 5 points on the unit equilateral triangle, show there's always a pair of distance $\leq \frac{1}{2}$ apart



If points p_1, p_2 in same quadrant, then $d(p_1, p_2) \leq \frac{1}{2}$

Given five points, two must share the same quadrant

Pigeonhole Principle!

Today's Keywords

Closest pair of points (HW2)

Matrix multiplication (if we have time)

CLRS Readings: Chapter 4

Homework

HW1 due ~~**Thursday, September 12**~~ **Saturday, September 14, 11pm**

- Start early!
- Written (use LaTeX!) – Submit both **zip** and **pdf**!
- Asymptotic notation
- Recurrences
- Divide and conquer

HW2 released today, due **Thursday, September 19, 11pm**

- Programming assignment (Python or Java)
- Divide and conquer (Closest pair of points)

Recurrence Solving Techniques

Tree

Guess/Check

"Cookbook"

Substitution

Master Theorem

$$T(n) = aT(n/b) + f(n)$$

$$\delta = \log_b a$$

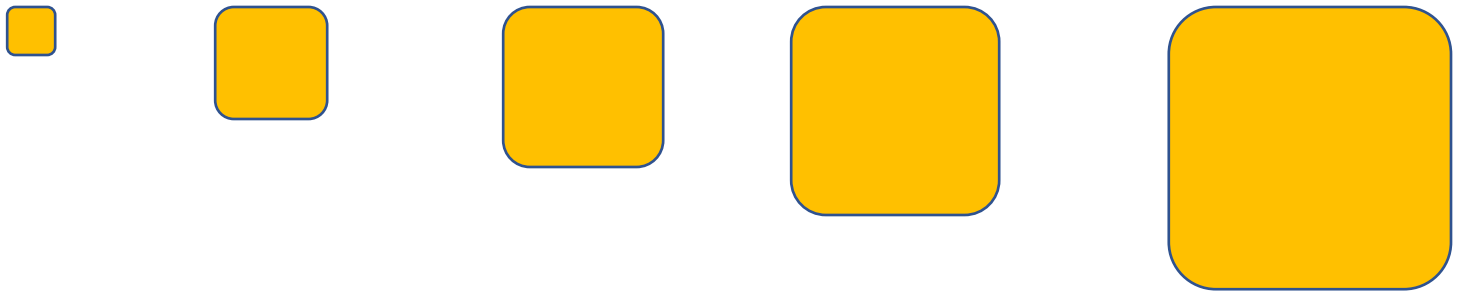
	Requirement on f	Implication
Case 1	$f(n) \in O\left(n^{\delta-\varepsilon}\right)$ for some constant $\varepsilon > 0$	$T(n) \in \Theta\left(n^{\delta}\right)$
Case 2	$f(n) \in \Theta\left(n^{\delta}\right)$	$T(n) \in \Theta\left(n^{\delta} \log n\right)$
Case 3	$f(n) \in \Omega\left(n^{\delta+\varepsilon}\right)$ for some constant $\varepsilon > 0$ **AND** $af\left(\frac{n}{b}\right) \leq cf(n)$ for constant $c < 1$ and sufficiently large n	$T(n) \in \Theta\left(f(n)\right)$

Three Cases

$$T(n) = f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + a^3 f\left(\frac{n}{b^3}\right) + \cdots + a^k f\left(\frac{n}{b^k}\right)$$

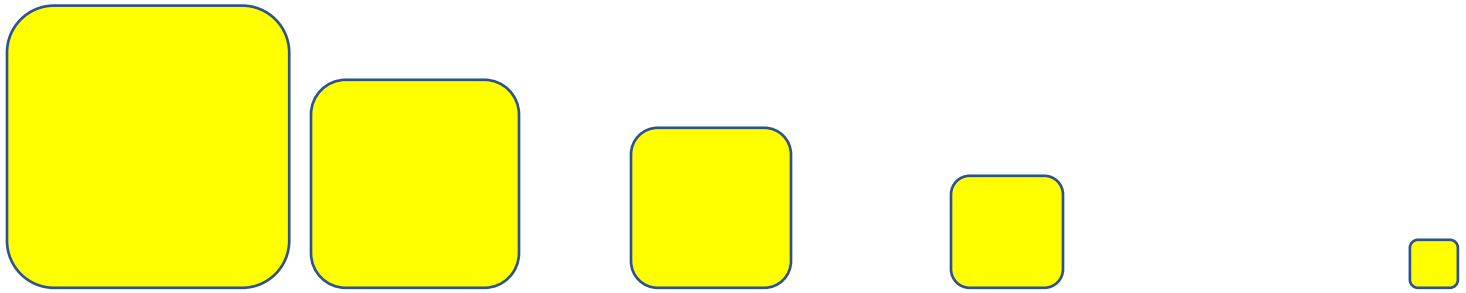
$$k = \log_b n$$

Case 1:
Most work happens at the leaves



Case 2:
Work happens consistently throughout

Case 3:
Most work happens at top of tree



Recurrence Solving Techniques

Tree

Guess/Check

“Cookbook”

Substitution

Substitution Method

Idea: Take a "difficult" recurrence and re-express it such that one of our other methods applies

Example:

$$T(n) = 2T(\sqrt{n}) + \log_2 n$$

Consider the following substitution: let $n = 2^m$ (i.e., $m = \log_2 n$)

$$T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$$ Rewrite recurrence in terms of m

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$ Consider substitution $S(m) = T(2^m)$

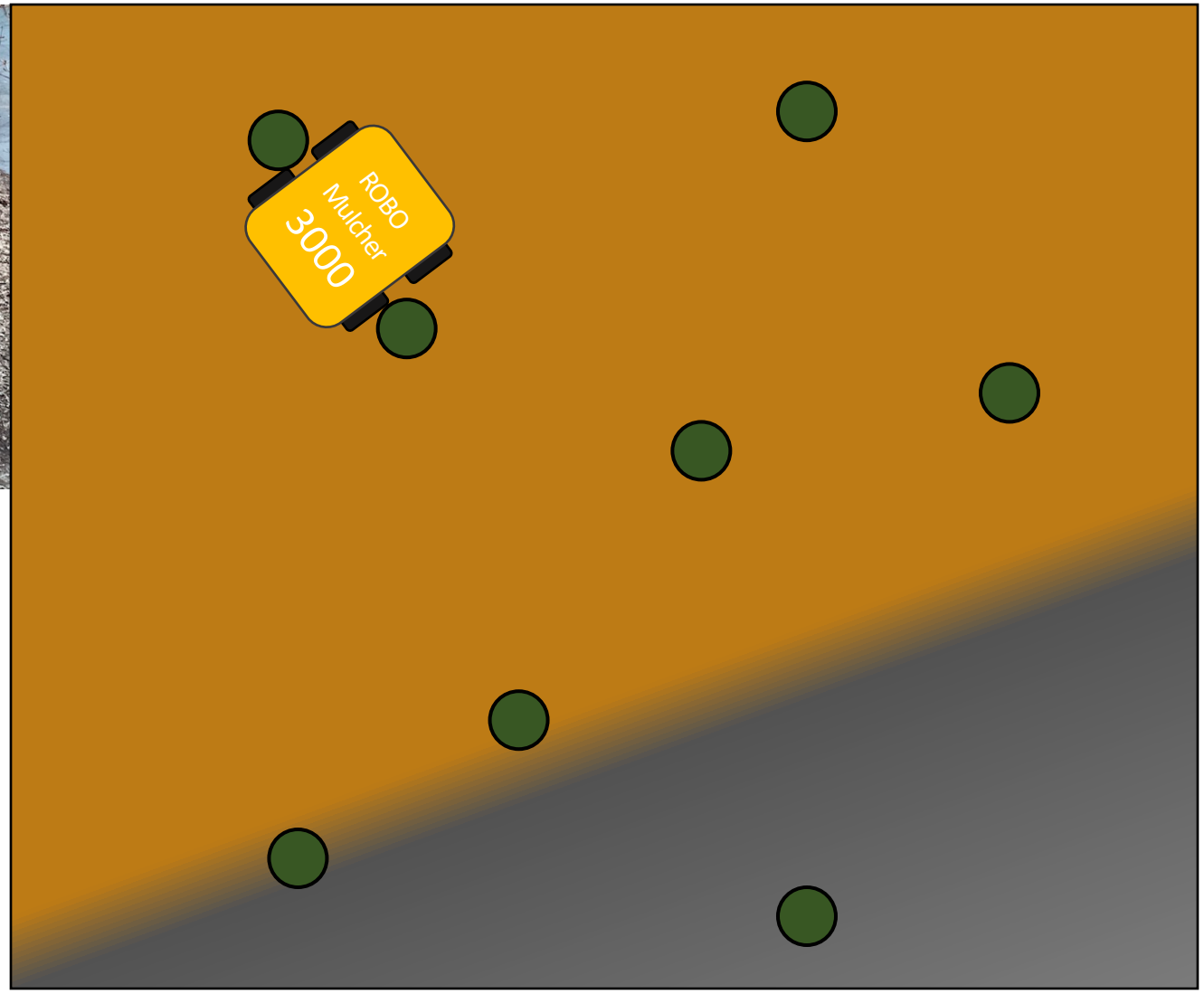
$$\Rightarrow S(m) = \Theta(m \log m)$$ Case 2 of Master Theorem

$$\Rightarrow T(n) = \Theta(\log n \log\log n)$$ Substitute back for T and n

Robbie’s Yard

There Has to be an Easier Way!

Constraints: Trees and Plants



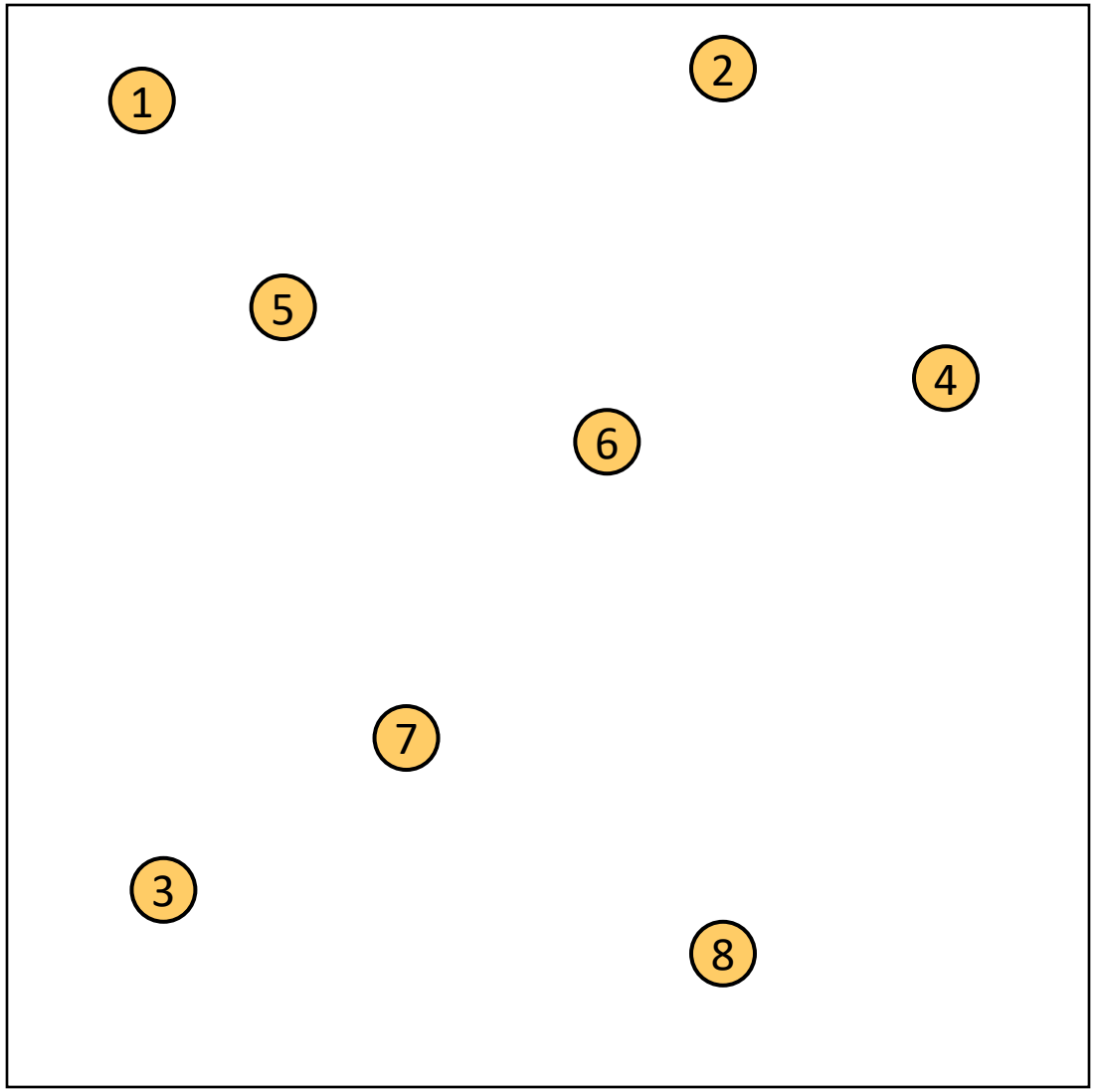
How wide can the robot be?

Objective: find closest pair of trees

Closest Pair of Points

Given: A list of points

Return: Pair of points with smallest distance apart



Closest Pair of Points: Naïve

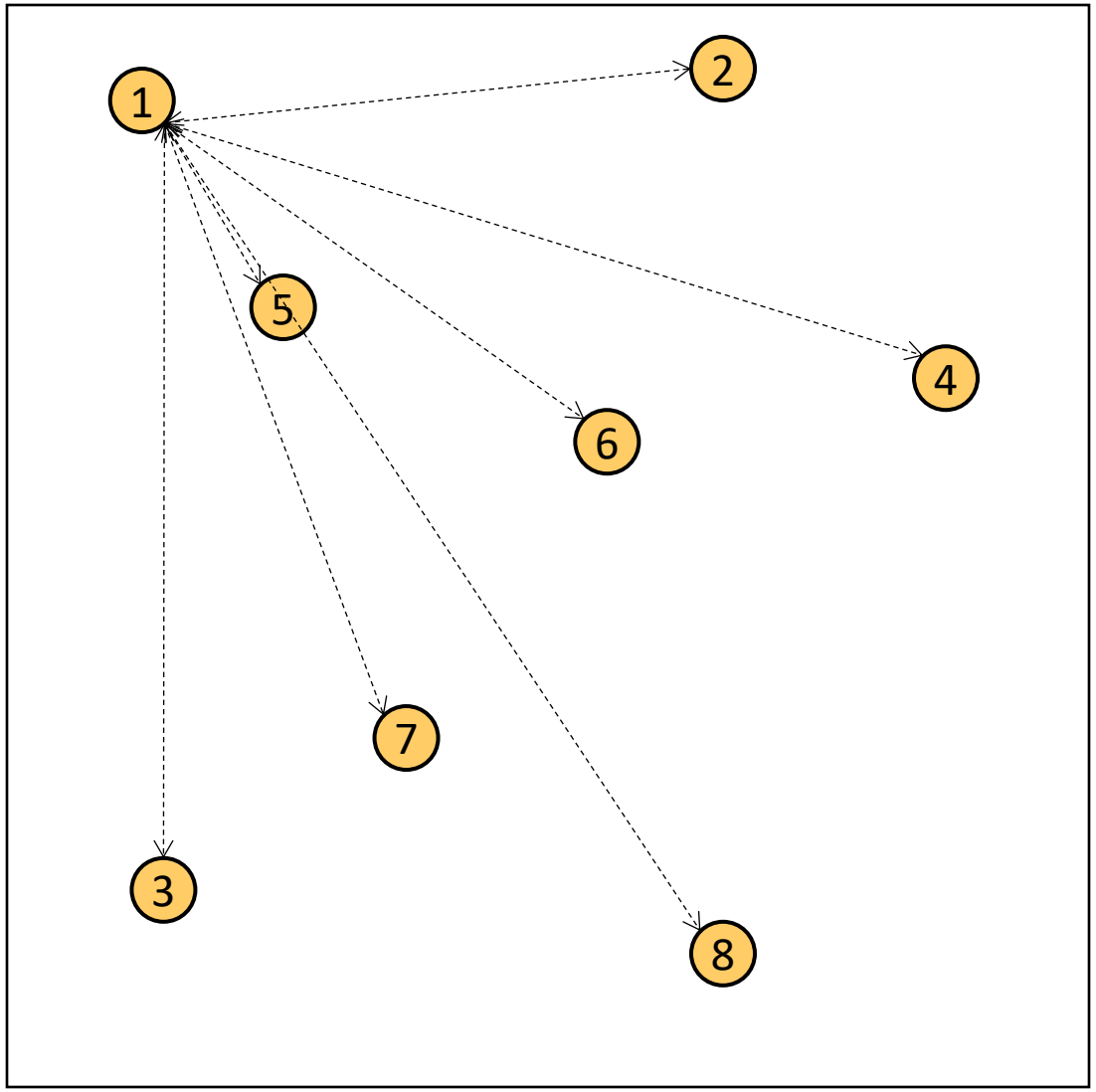
Given: A list of points

Return: Pair of points with smallest distance apart

Algorithm: Test every pair of points, return the closest

Running Time: $O(n^2)$

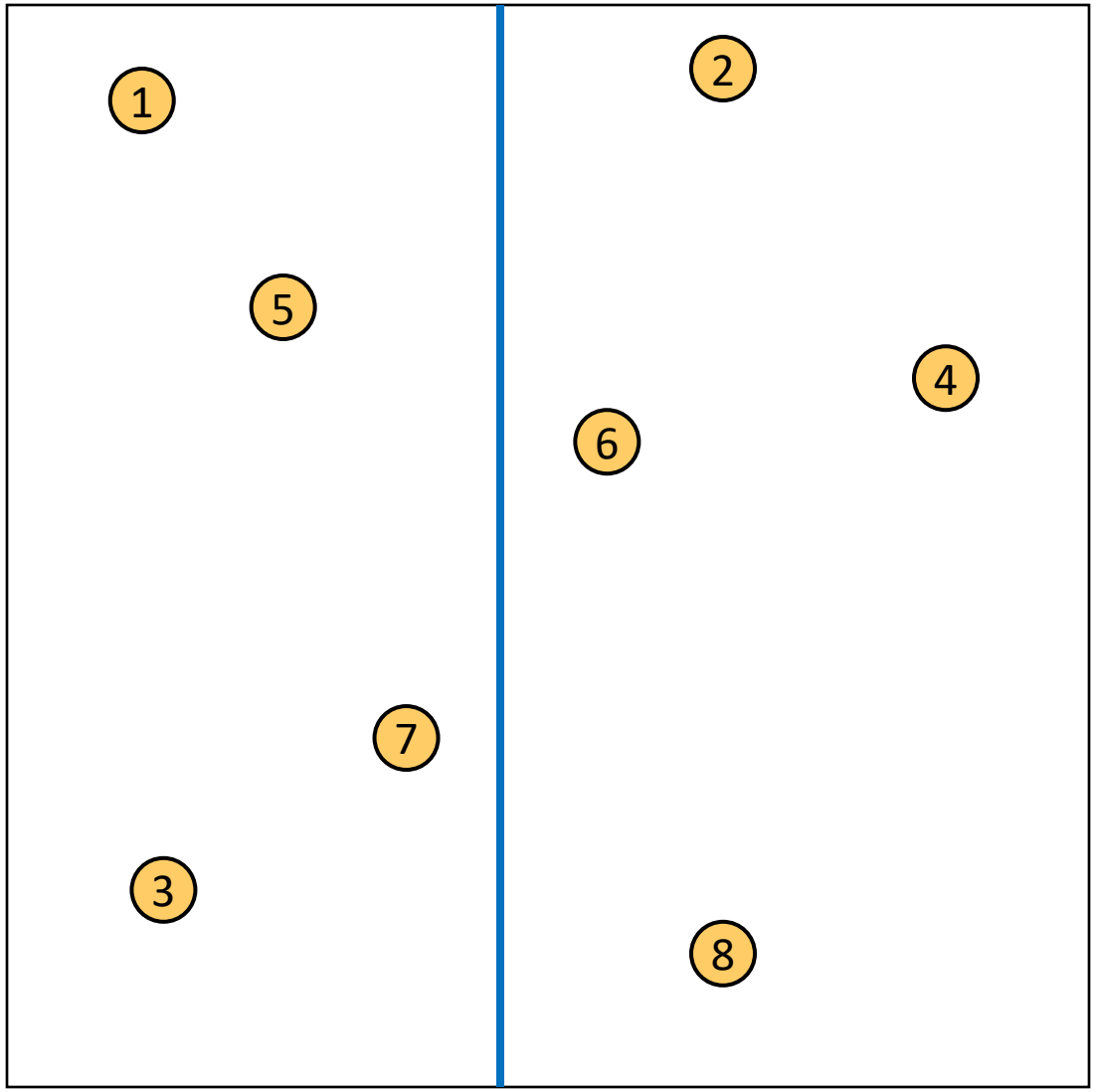
Goal: $O(n \log n)$



Closest Pair of Points: Divide and Conquer

Divide: How?

At median x coordinate



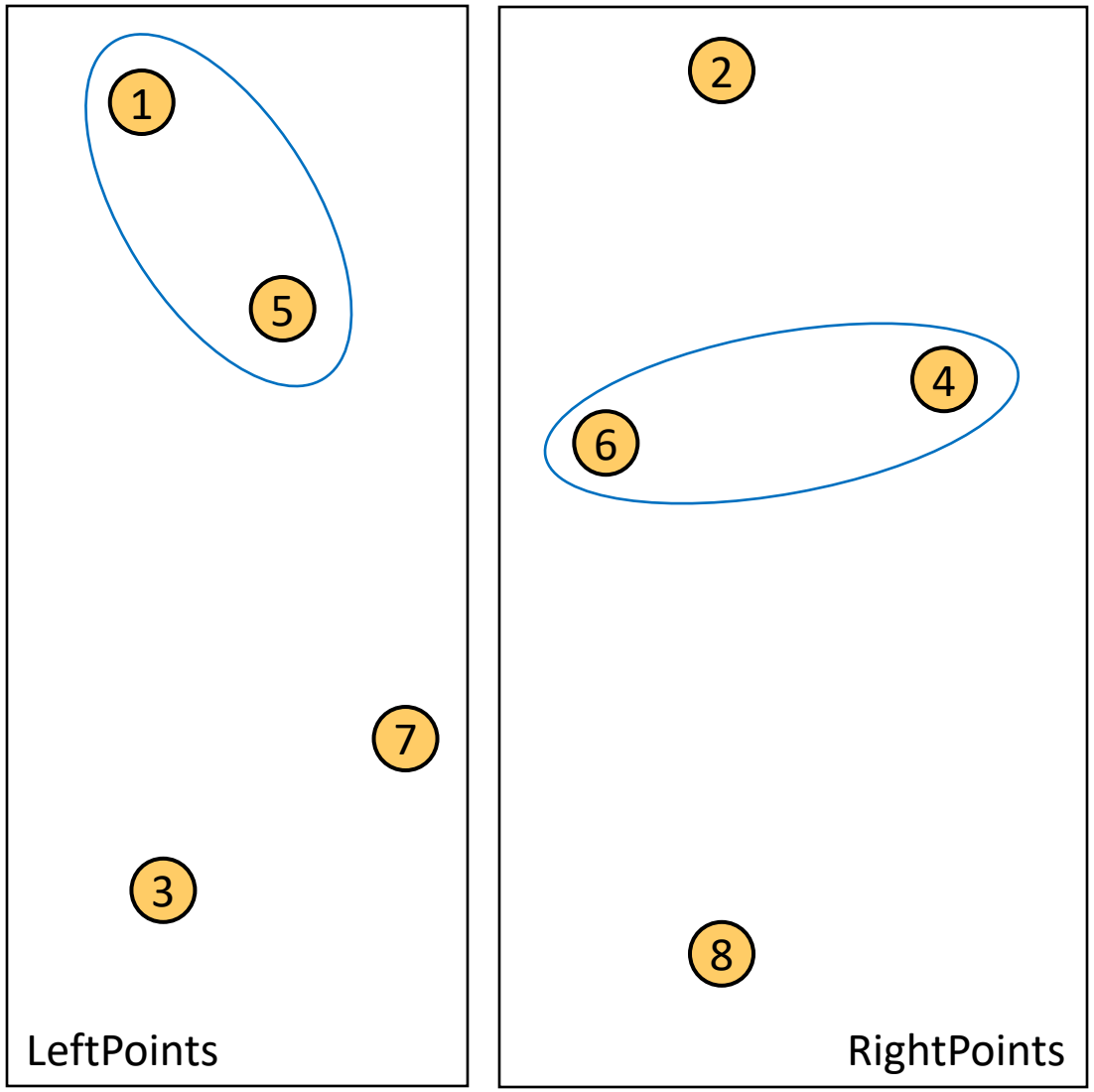
Closest Pair of Points: Divide and Conquer

Divide:

At median x coordinate

Conquer:

Recursively find closest pairs from LeftPoints and RightPoints



Closest Pair of Points: Divide and Conquer

Divide:

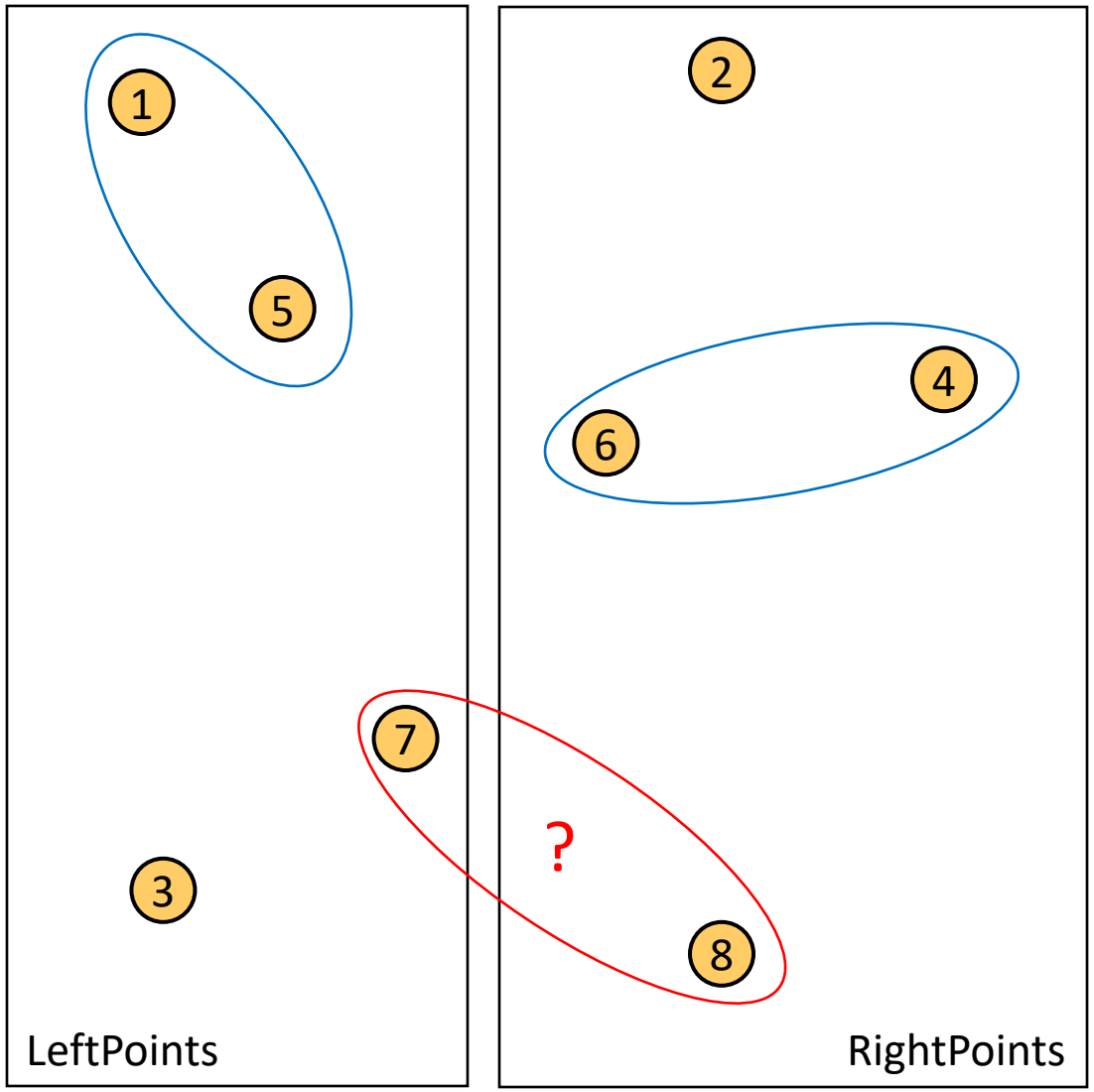
At median x coordinate

Conquer:

Recursively find closest pairs from LeftPoints and RightPoints

Combine:

Return smaller of left and right pairs Problem?



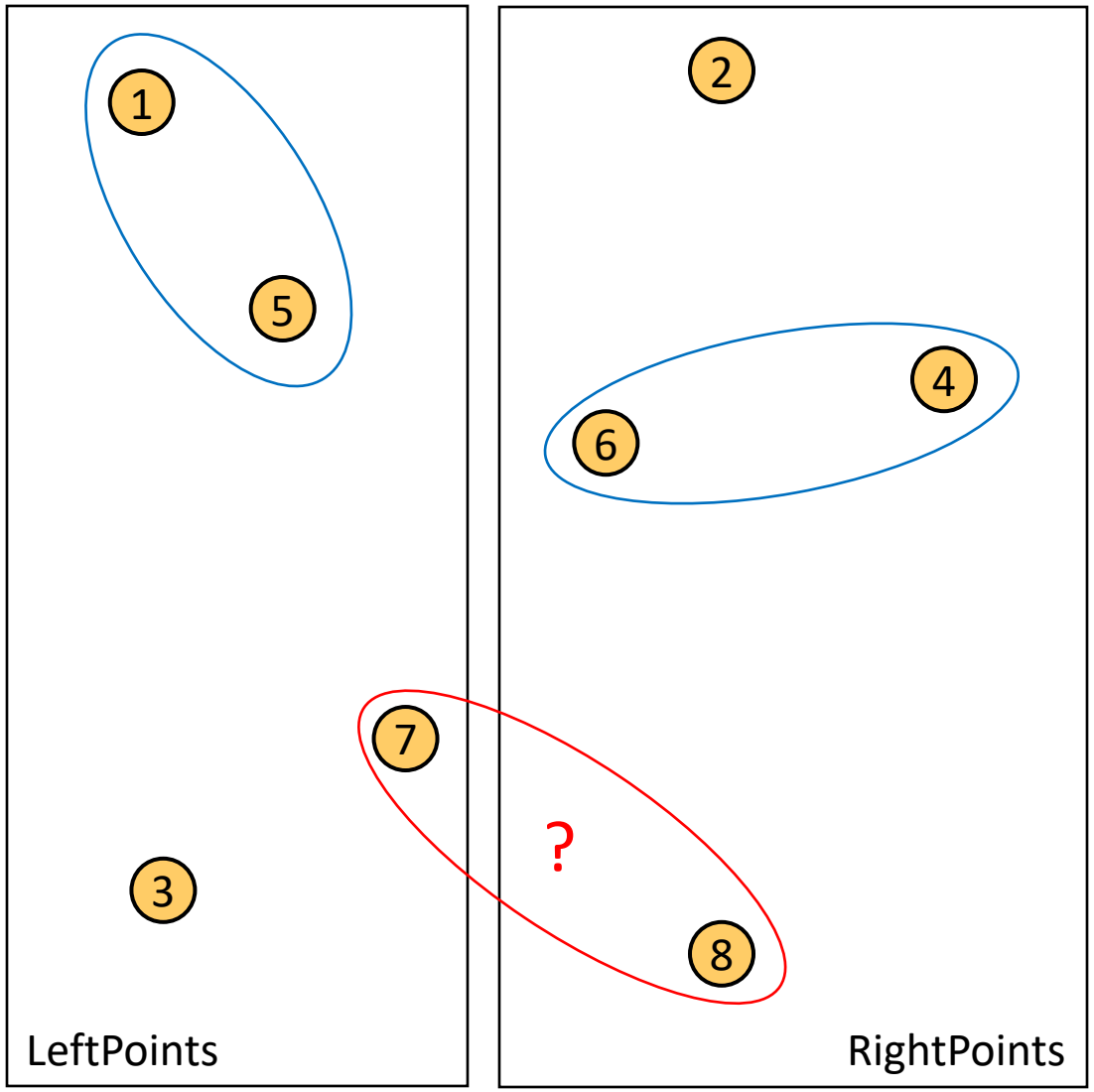
Closest Pair of Points: Divide and Conquer

Combine:

Case 1: Closest pair is completely in LeftPoints or RightPoints

Case 2: Closest pair spans our "cut"

Need to test points across the cut

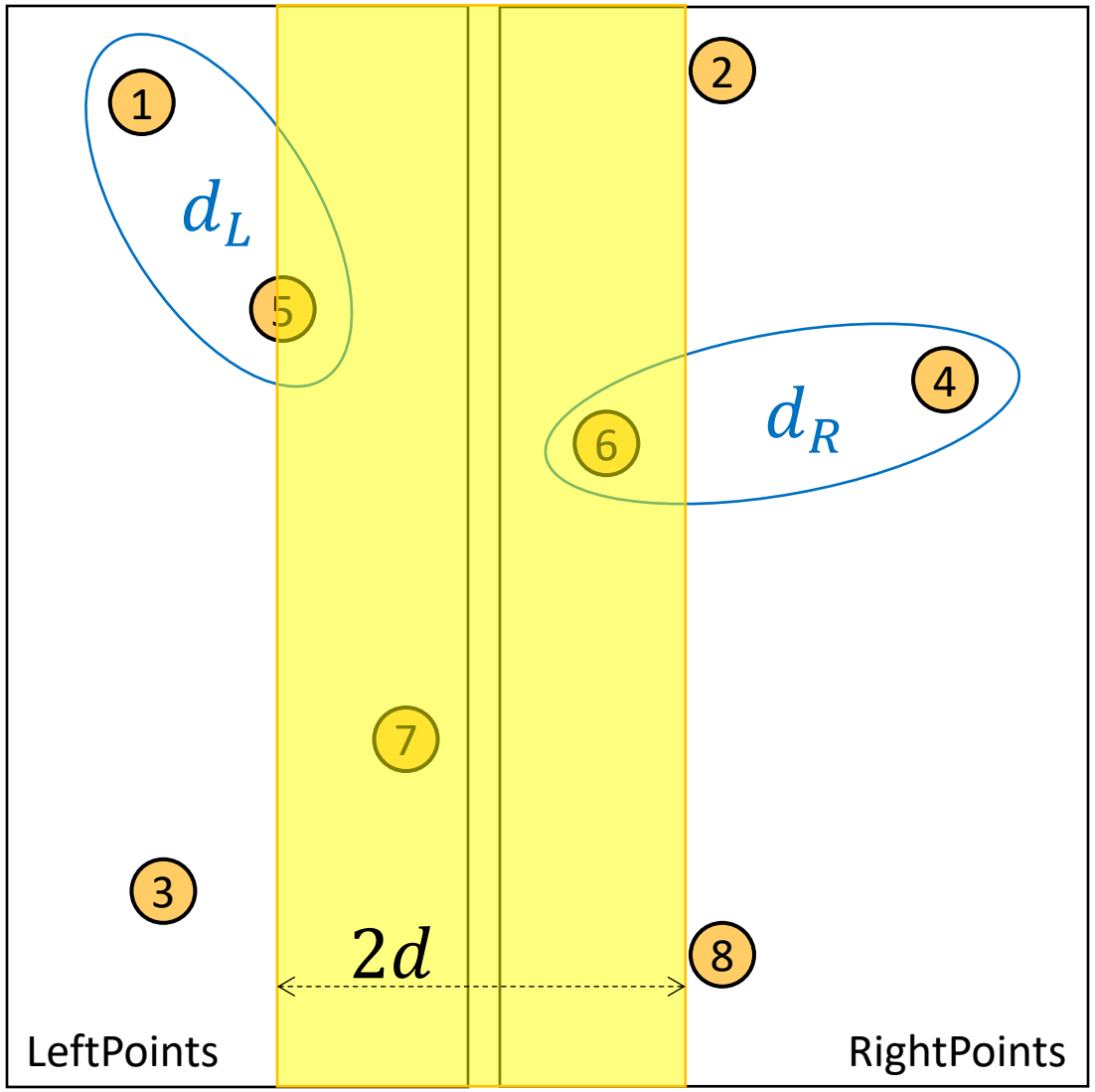


Spanning the Cut

Case 2: Closest pair spans our "cut"

Need to test points across the cut

- Compare all pairs of points within $d = \min\{d_L, d_R\}$ of the cut
- How many are there?



Spanning the Cut

Case 2: Closest pair spans our "cut"

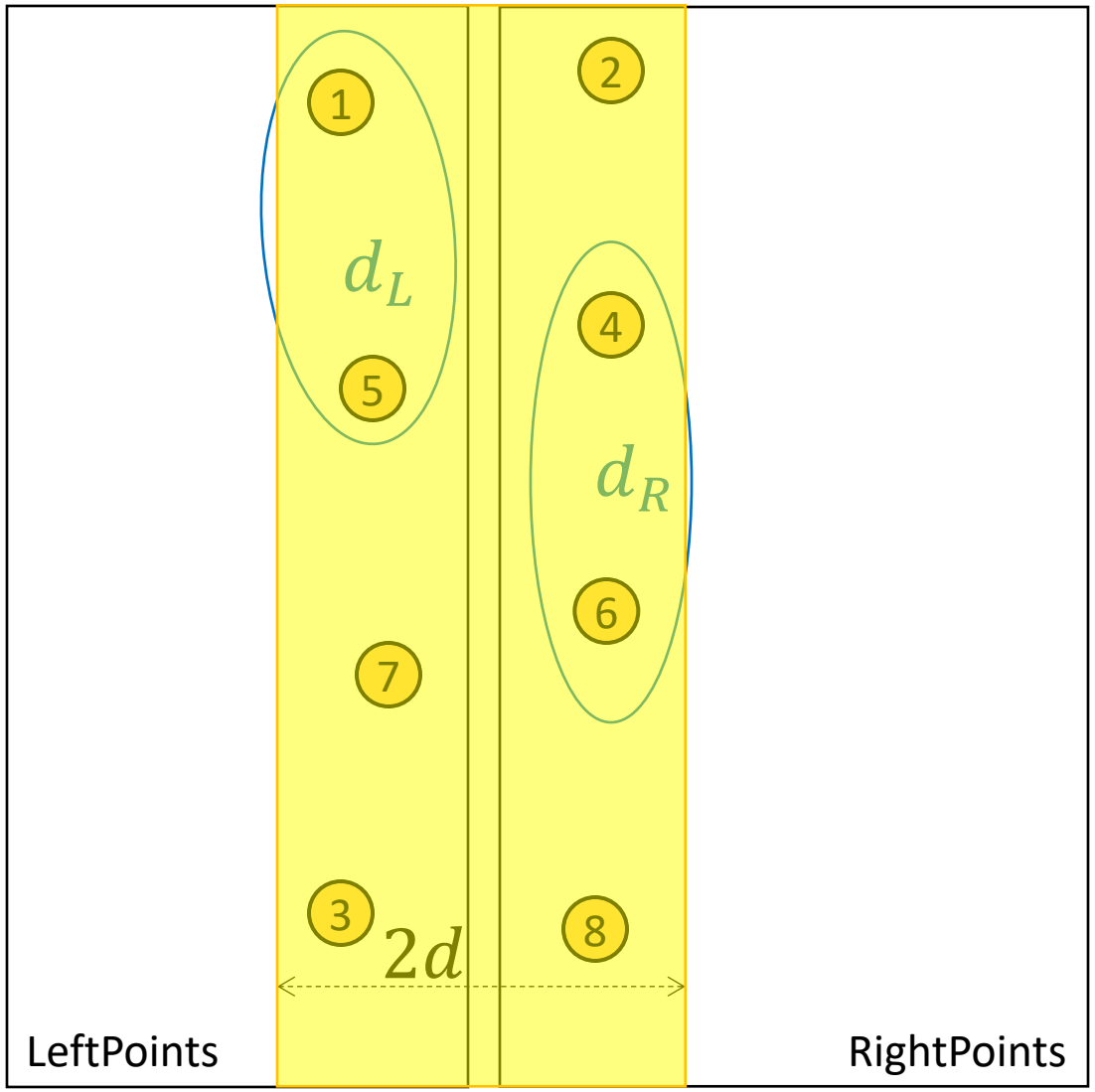
Need to test points across the cut

Compare all pairs of points within $d = \min\{d_L, d_R\}$ of the cut

How many are there?

In the worst case, **all** of the points!

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



Spanning the Cut

Case 2: Closest pair spans our "cut"

Need to test points across the cut

Compare all p

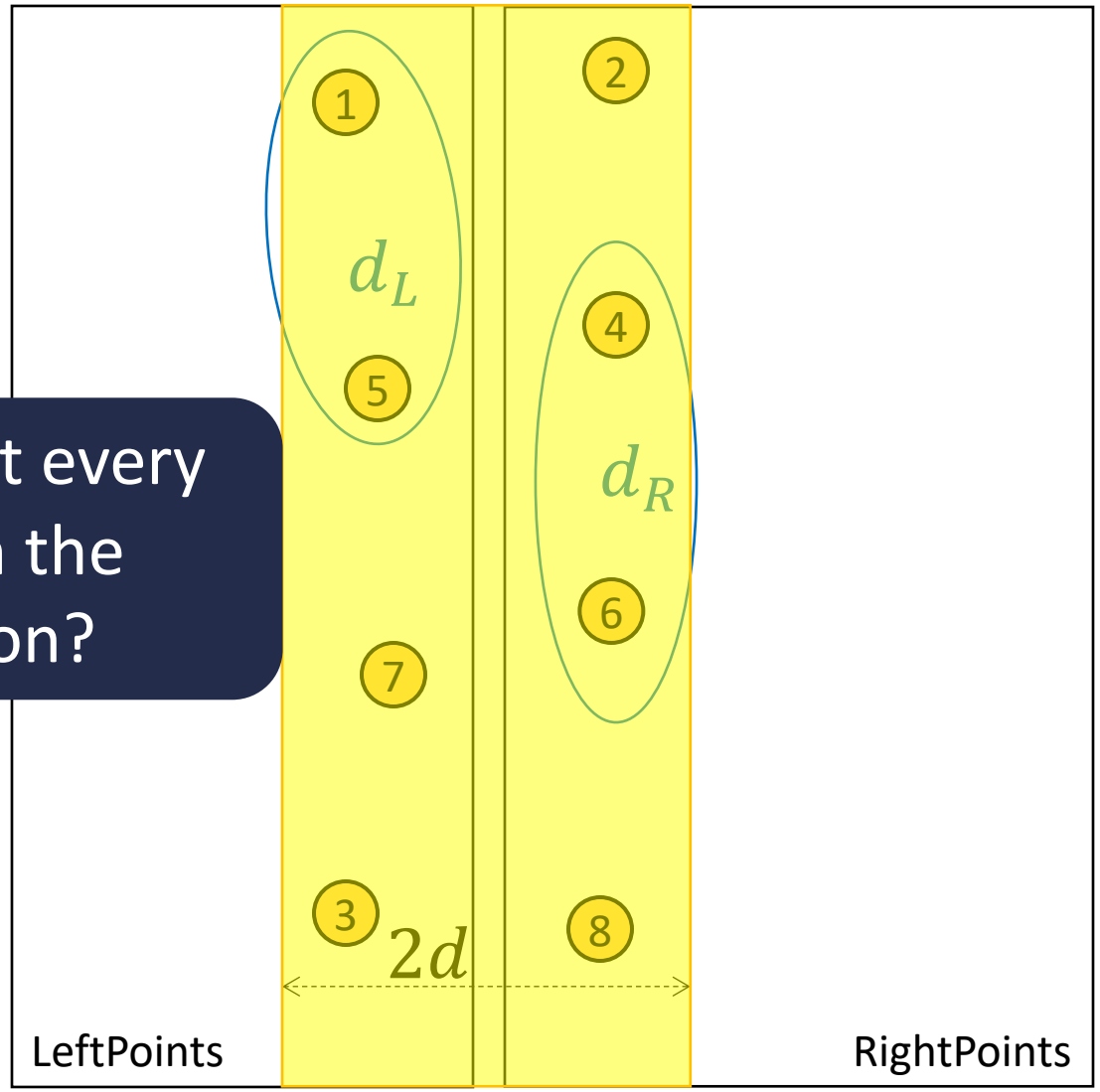
$d = \min\{d_L,$

How many ar

In the worst case, **all** of the points!

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

Do we need to test every pair of points in the boundary region?



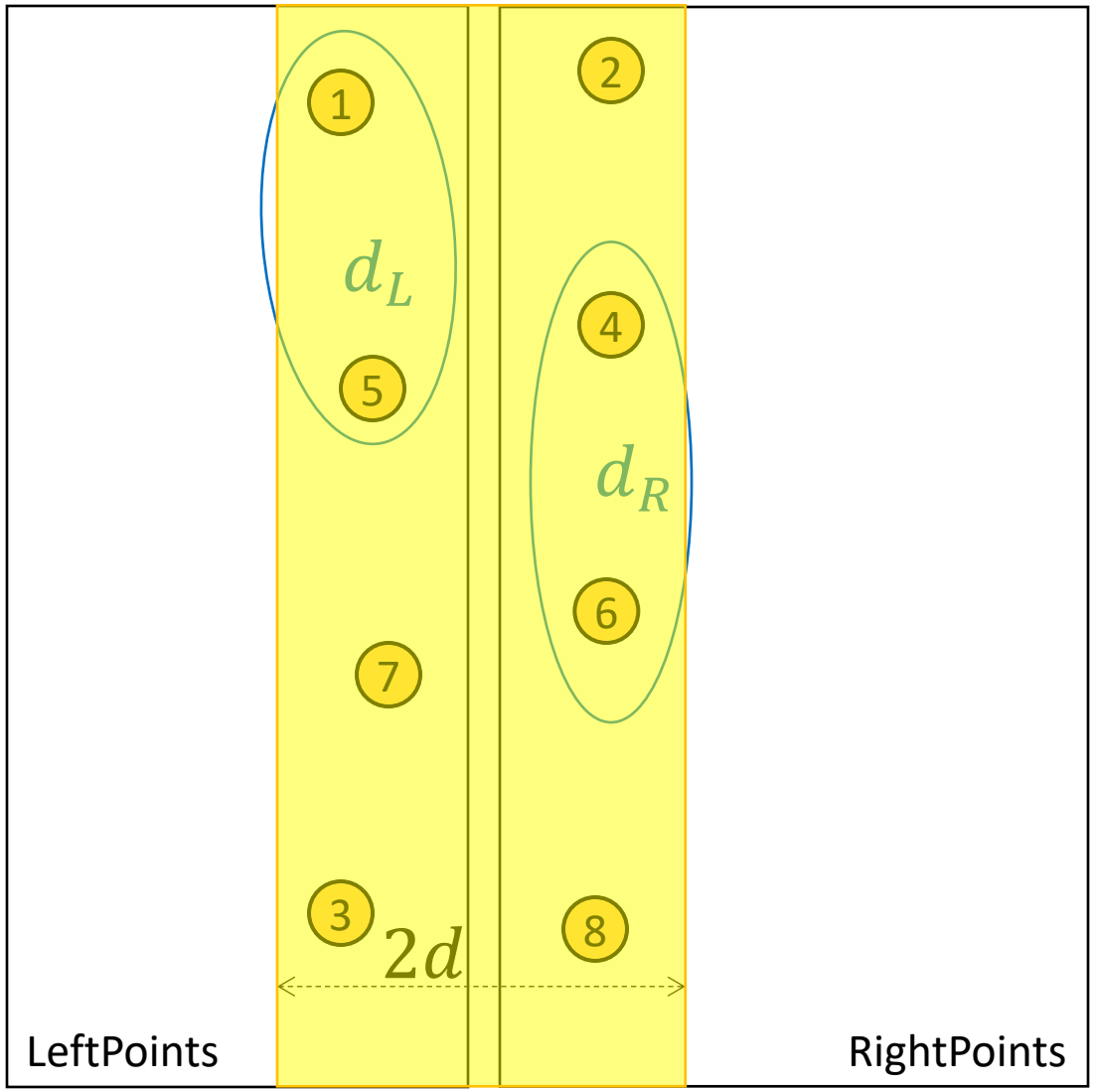
Spanning the Cut

Case 2: Closest pair spans our "cut"

Need to test points across the cut

Observation: We don't need to test all pairs!

Only need to test points within distance d of each another



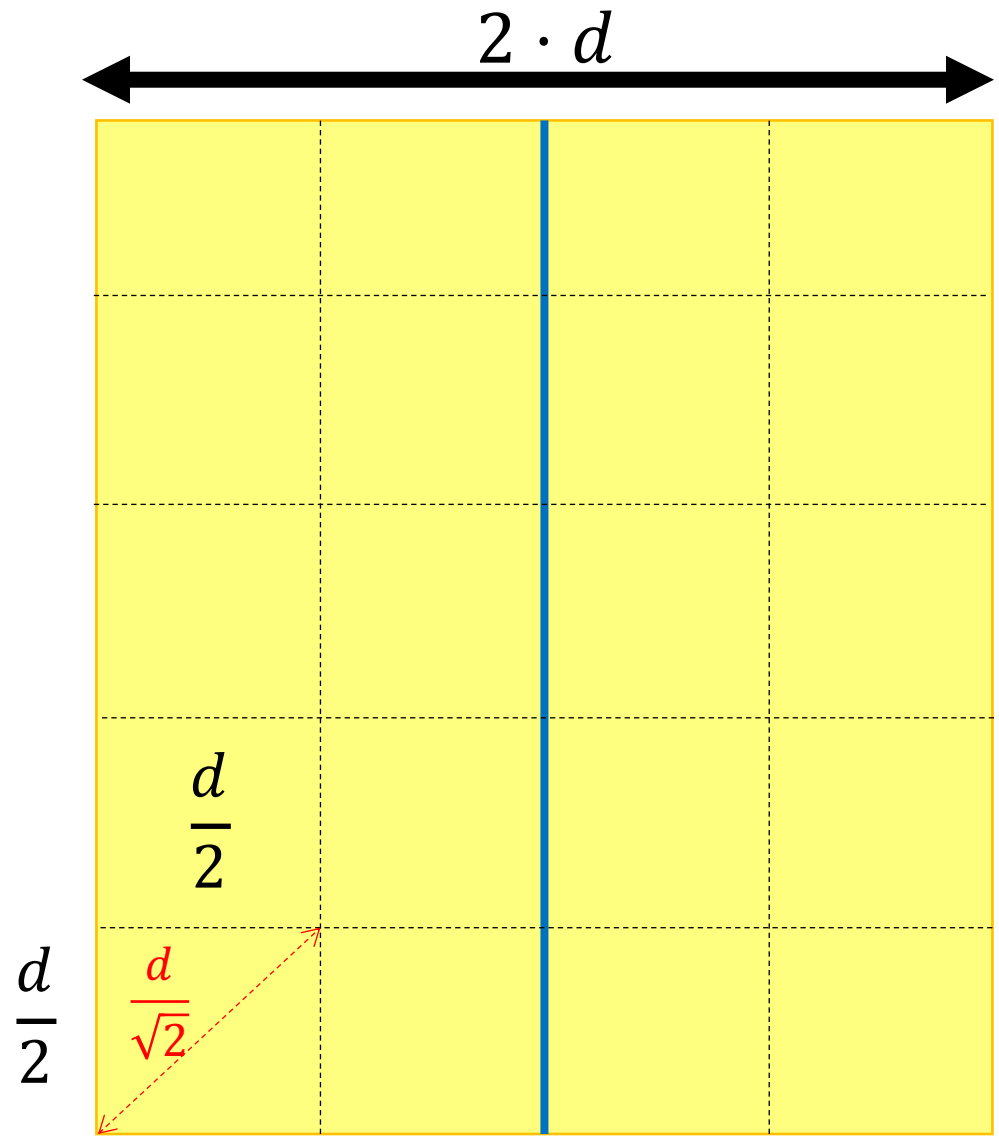
Reducing Search Space

Case 2: Closest pair spans our "cut"

Need to test points across the cut

Divide the boundary into squares with dimension $d/2$

How many points can be in a square? **at most 1**



Reducing Search Space

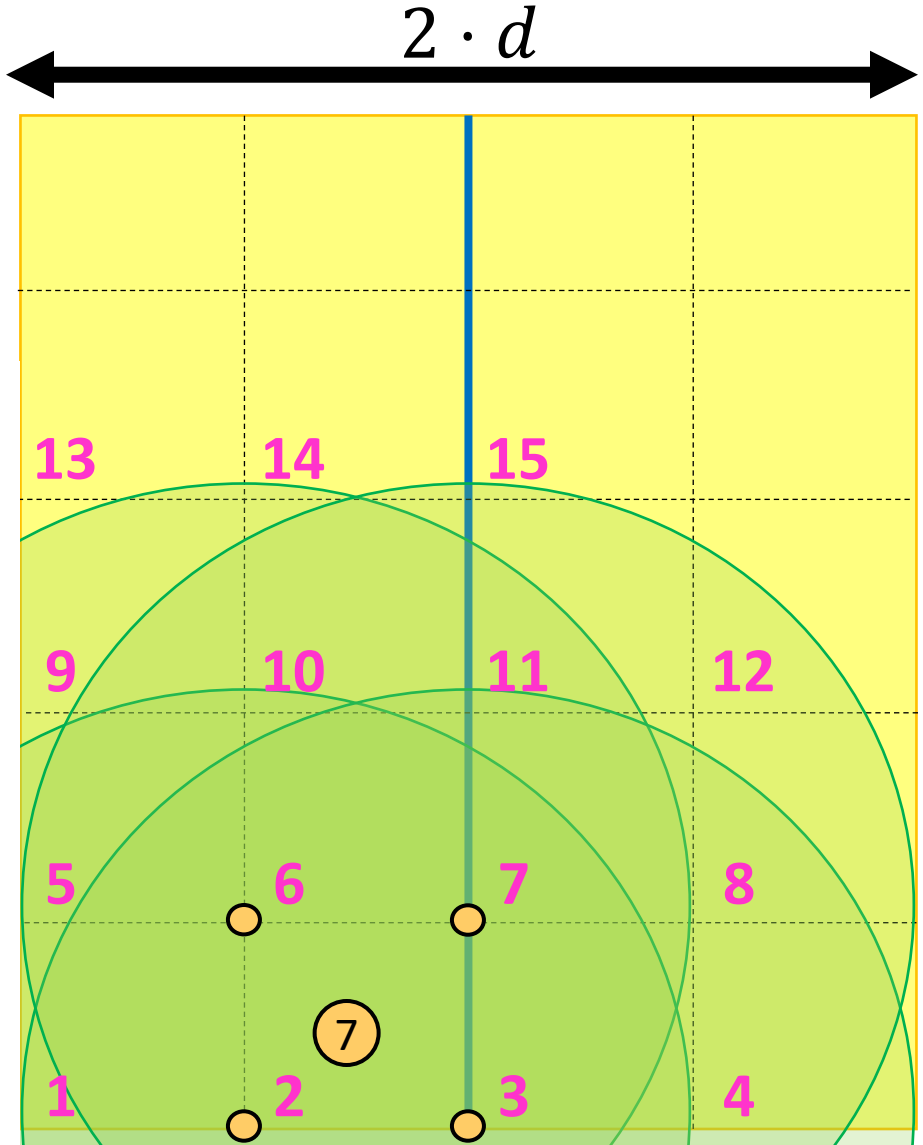
Case 2: Closest pair spans our "cut"

Need to test points across the cut

Divide the boundary into squares with dimension $d/2$

How many squares can contain a point $< d$ away and with smaller y-coordinate?

at most **15**



Reducing Search Space

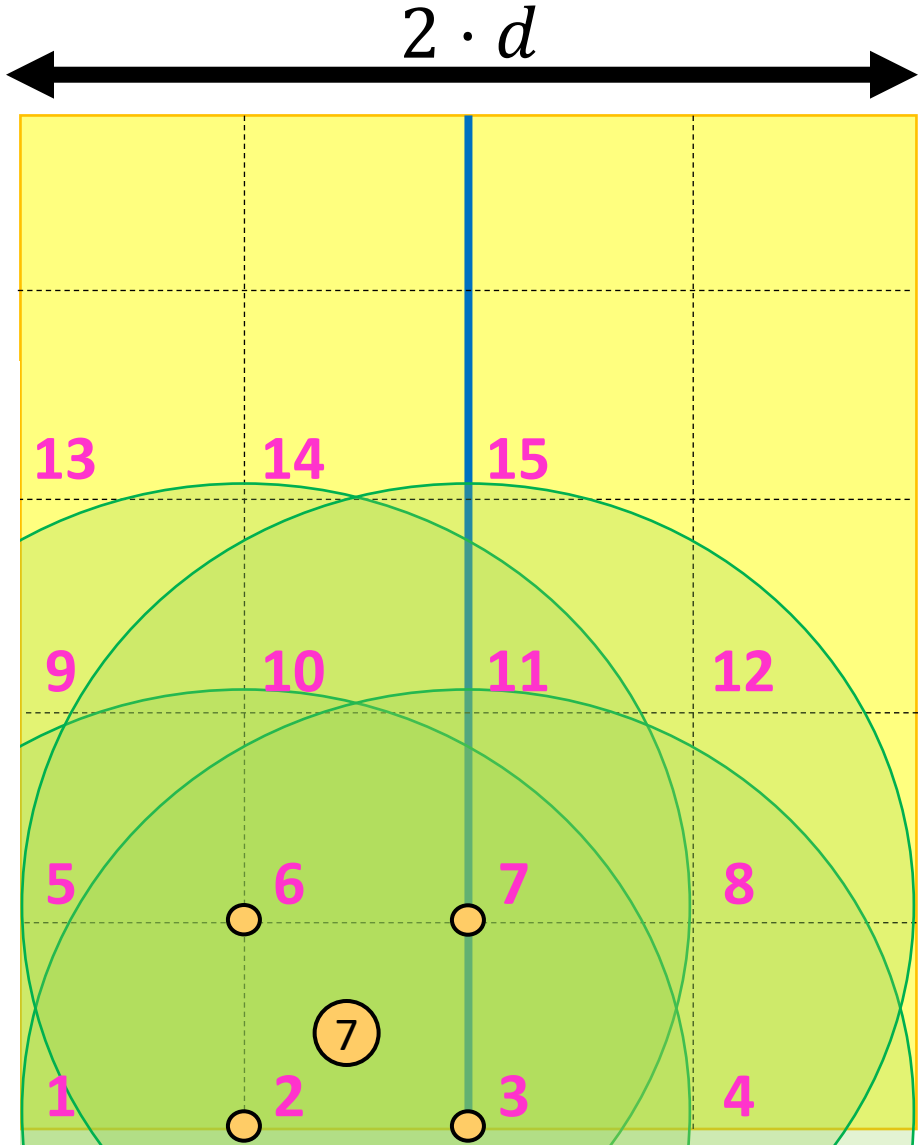
Case 2: Closest pair spans our "cut"

Need to test points across the cut

Why only consider points with smaller y-coordinate?

How many squares can contain a point $< d$ away and with **smaller y-coordinate**?

at most **15**



Closest Pair of Points: Divide and Conquer

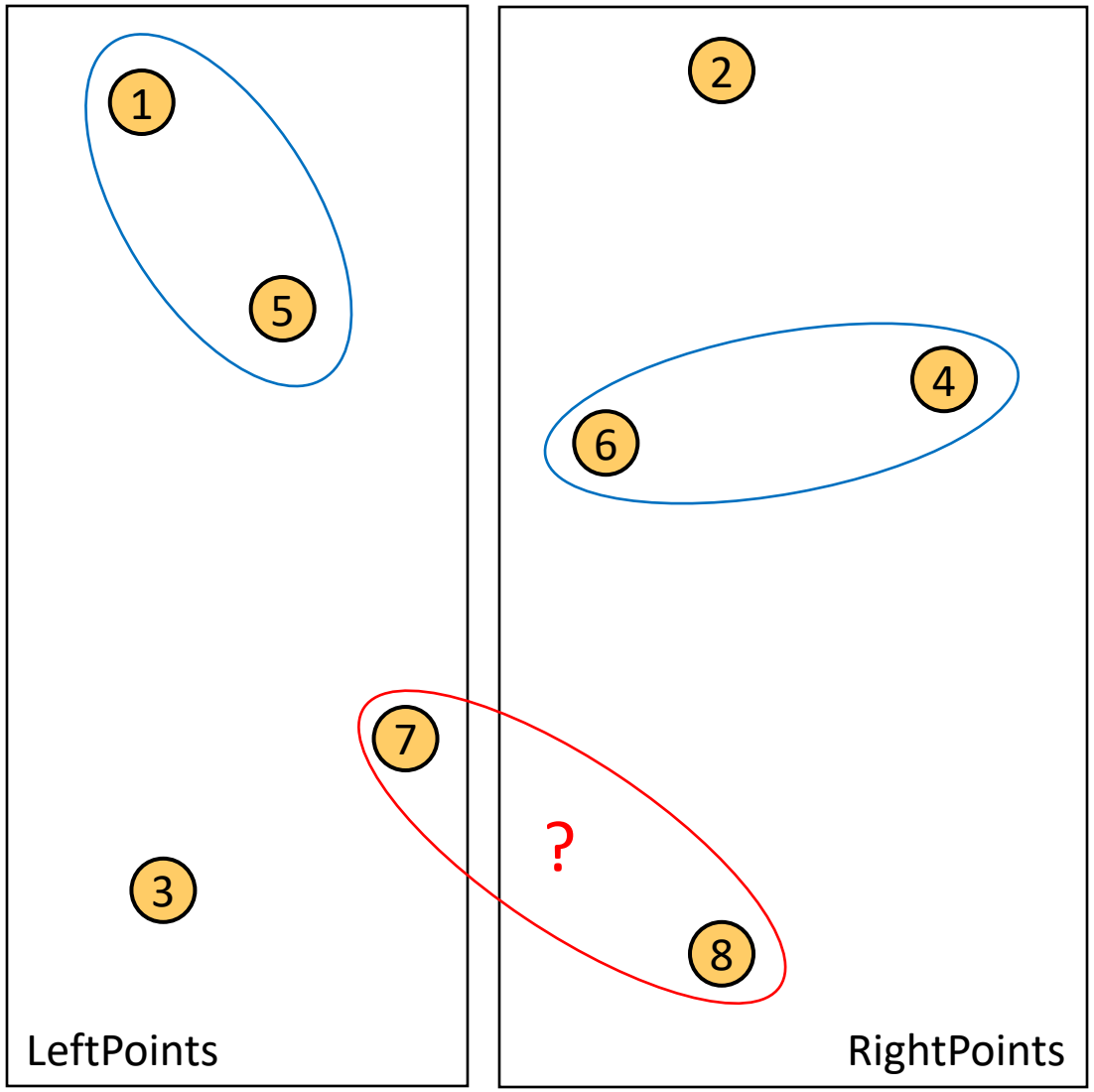
Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points based on x-coordinate

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

Combine:

- Construct list of points in the boundary
- Sort boundary points by y-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary points



Closest Pair of Points: Divide and Conquer

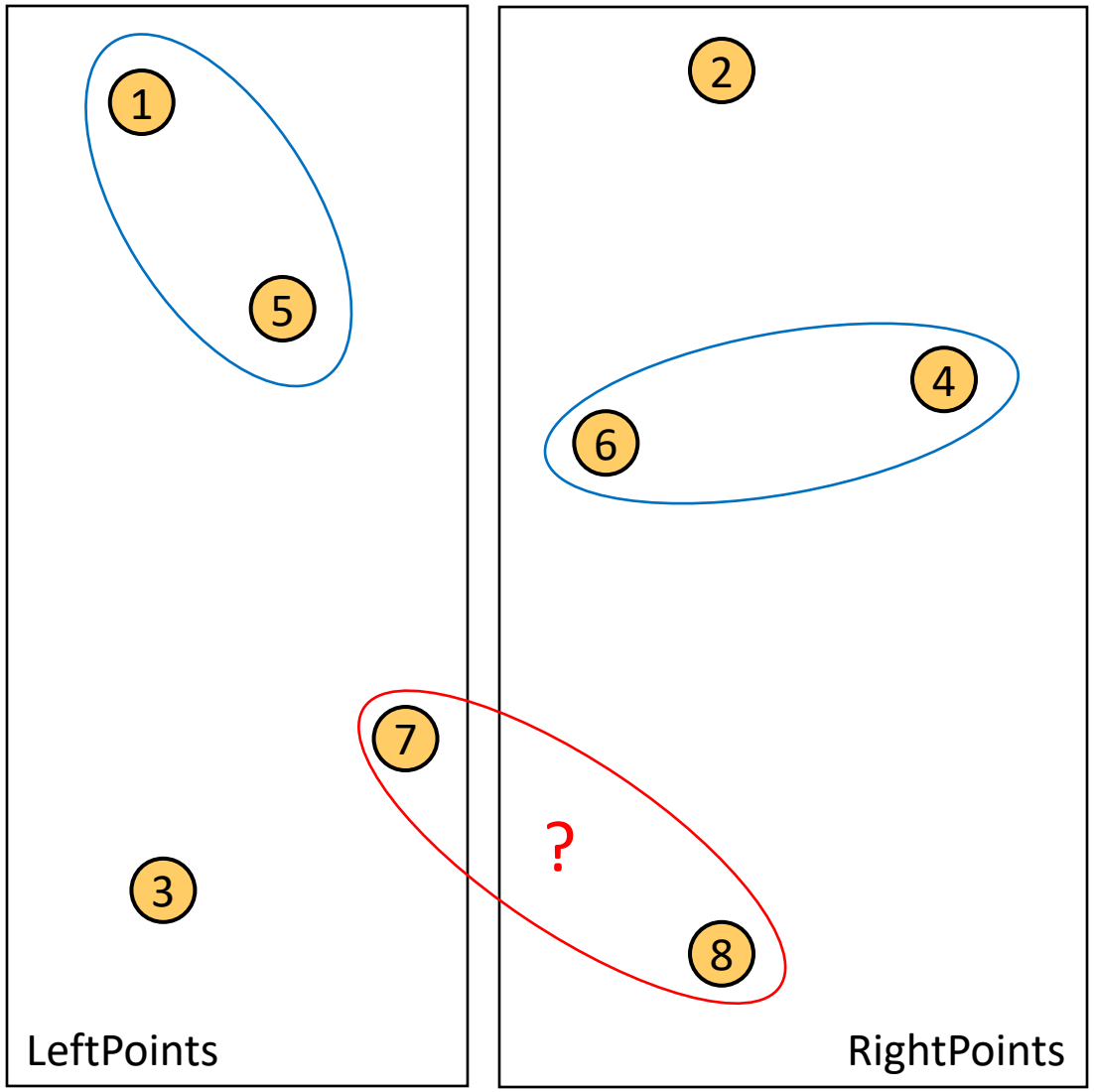
Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points

Looks like another $O(n \log n)$ algorithm – combine step is still too expensive

Combine:

- Construct list of points in t[illegible] boundary
- Sort boundary points by y-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary 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

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

Combine:

- Construct list of points in the boundary
- Sort boundary points by y-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary 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 boundary points by y-coordinate now becomes a **merge**

Listing Points in the Boundary

LeftPoints:

Closest Pair: $(1,5)$, $d_{1,5}$

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

RightPoints:

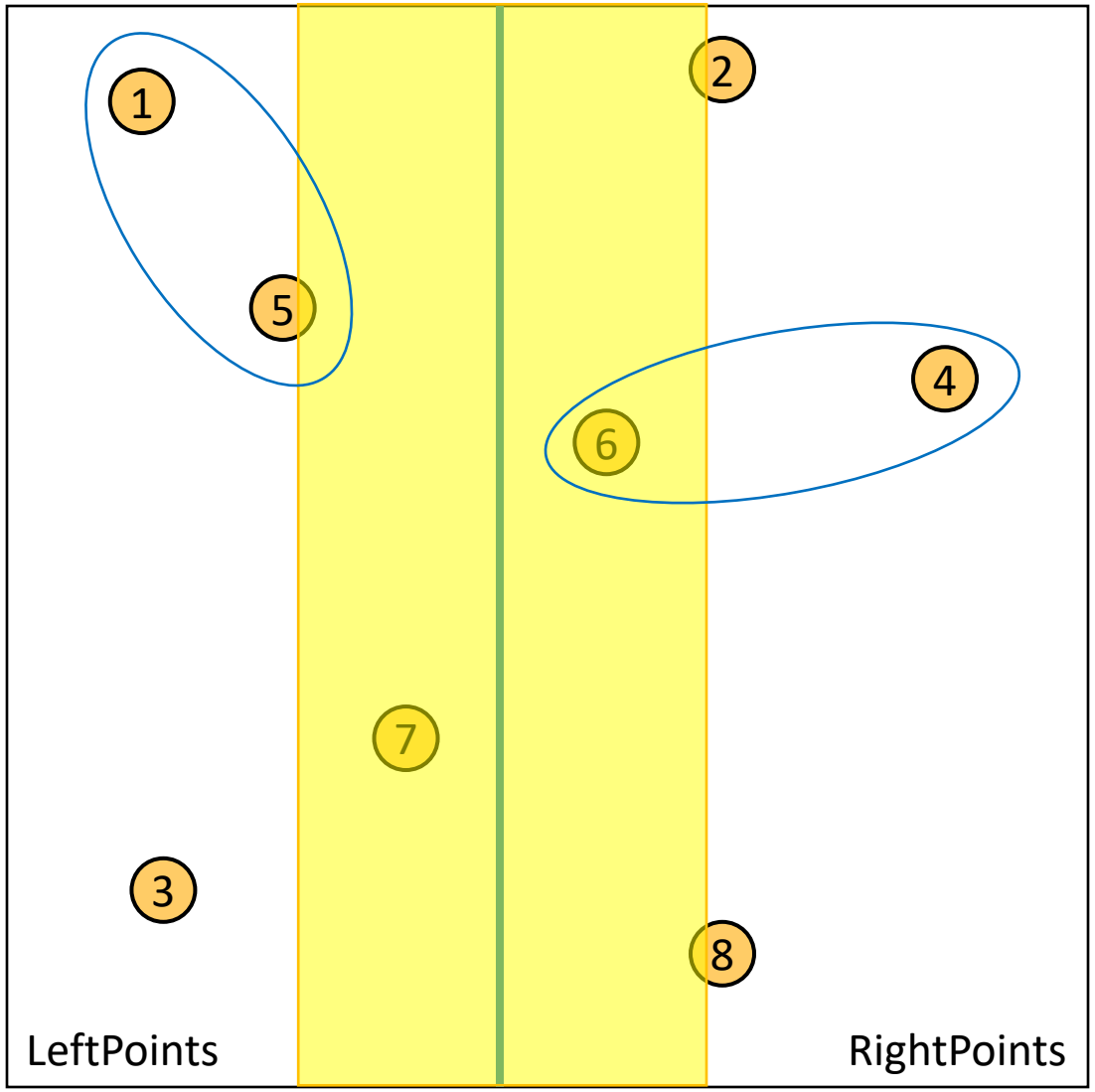
Closest Pair: $(4,6)$, $d_{4,6}$

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

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

Boundary 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

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

Combine:

- Construct list of points in the boundary
- Sort boundary points by y-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary points

Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points based on x-coordinate

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 boundary (sorted by y-coordinate)
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary 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:

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

$T(n)$

- $\Theta(n \log n)$ — **Initialization:** Sort points by x-coordinate
- $\Theta(1)$ — **Divide:** Partition points into two lists of points based on x-coordinate
- $2T(n/2)$ — **Conquer:** Recursively compute the closest pair of points in each list
- $\Theta(n)$ — **Combine:**
  - Merge sorted list of points by y-coordinate and construct list of points in the boundary (sorted by y-coordinate)
  - Compare each point in boundary to 15 points above it and save the closest pair
  - Output closest pair among left, right, and boundary points
- $\Theta(n)$
- $\Theta(1)$

Matrix Multiplication

$$\overset{n}{\begin{bmatrix}1 & 2 & 3\end{bmatrix}} \times \begin{bmatrix}2 & 4 & 6\\ 8 & 10 & 12\\ 14 & 16 & 18\end{bmatrix}$$

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

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

Run time? $O(n^3)$

Matrix Multiplication Divide and Conquer

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

Divide:

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

Matrix Multiplication Divide and Conquer

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

Matrix Multiplication Divide and Conquer

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

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

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

Case 1!

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

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

Can we do better?

Matrix Multiplication D&C

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

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

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

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:

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

7 Multiplications

18 Additions

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\frac{n^2}{4}$$

Strassen's Algorithm

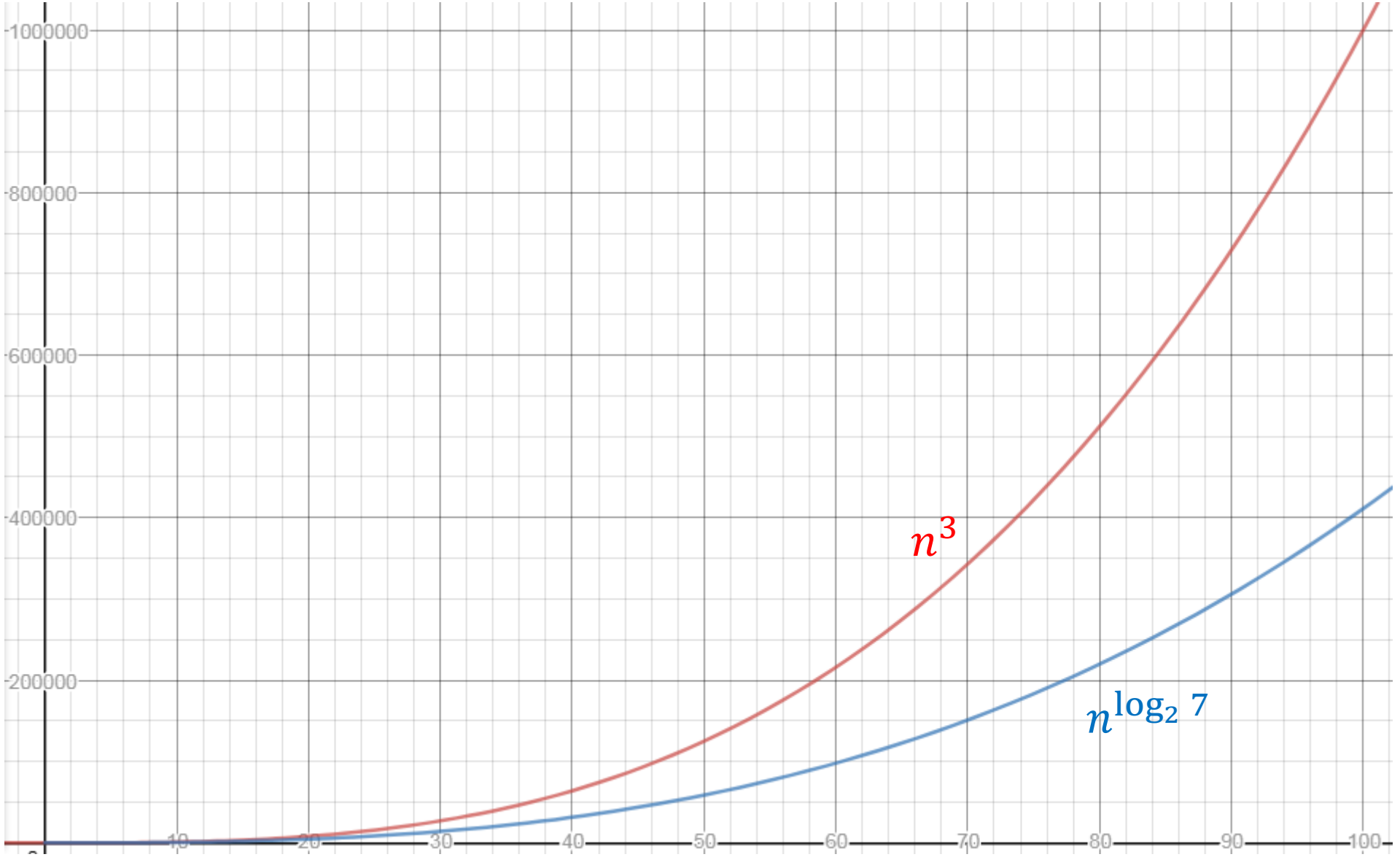
$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2$$

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

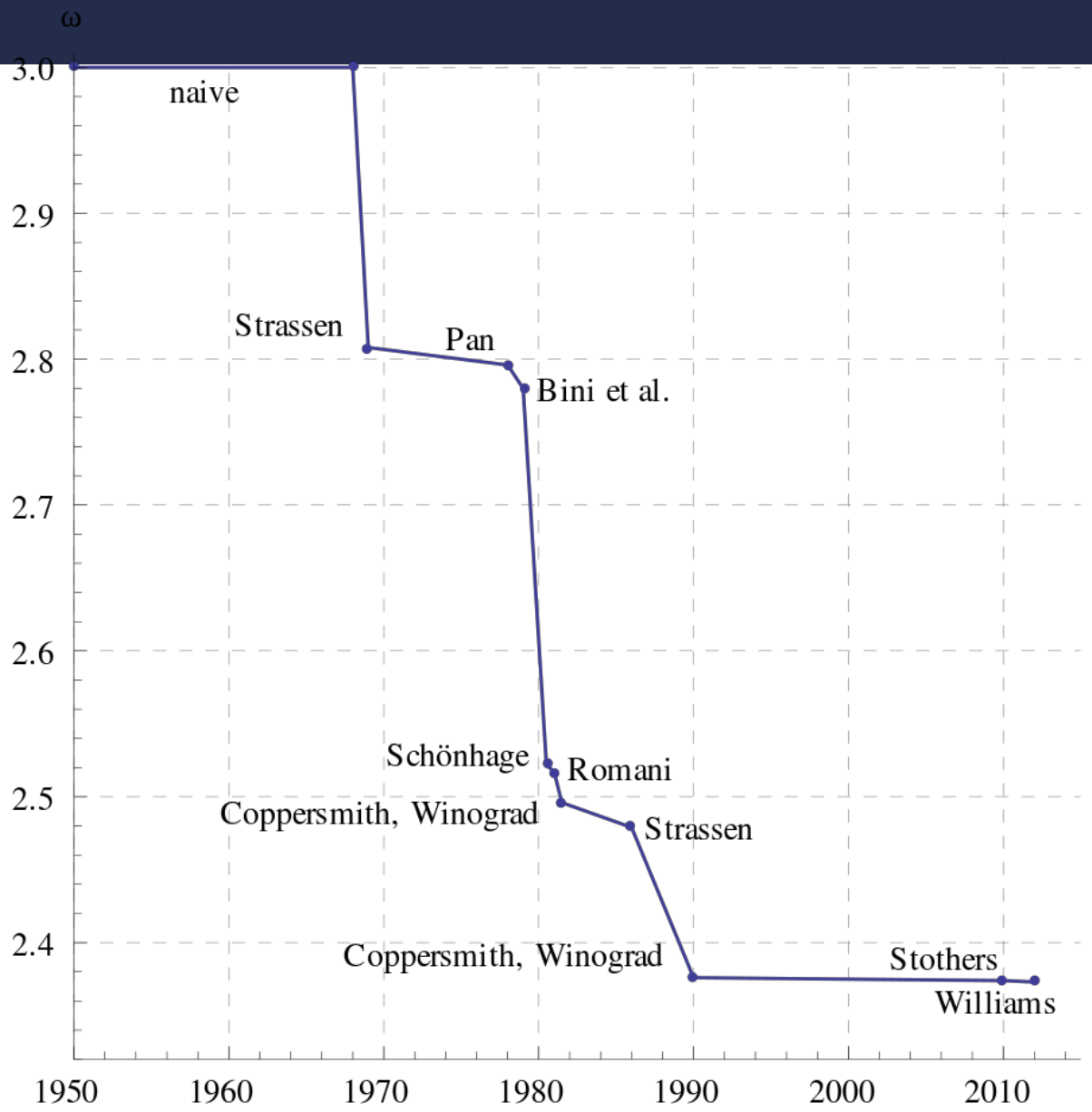
Case 1!

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

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


1000000
800000
600000
400000
200000
10
20
30
40
50
60
70
80
90
100
n^3
$n^{\log_2 7}$

Is This the Fastest?



Best possible is still unknown

Trivial lower bound: $\Omega(n^2)$