CS 3100
Data Structures and Algorithms 2
Lecture 9: D&C: Closest Pair of Points

Co-instructors: Robbie Hott and Ray Pettit

Spring 2024

Readings in CLRS 4th edition:

- Section 4.5

Announcements

- PS4 coming soon
- Office hours
  - Prof Hott Office Hours: Mondays 11a-12p, Fridays 10-11a and 2-3p
  - Prof Pettit Office Hours: Mondays and Wednesdays 2:30-4:00p
  - TA office hours posted on our website
- Quizzes 1-2 coming February 29, 2024
  - Both quizzes taken the same day
  - If you have SDAC, please schedule for 1 exam (*not a quiz*)

Divide and Conquer

[CLRS Chapter 4]

Divide:

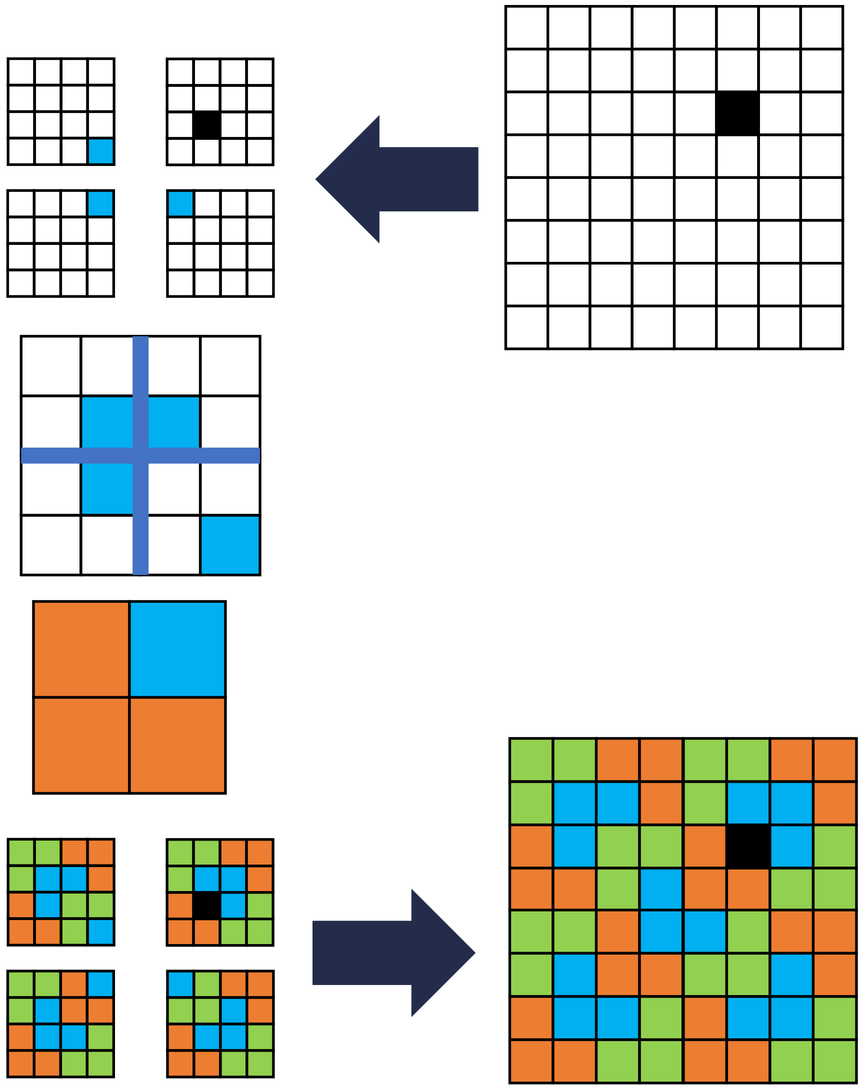
- Break the problem into multiple subproblems, each smaller instances of the original

Conquer:

- If the suproblems are “large”:
  - Solve each subproblem recursively
- If the subproblems are “small”:
  - Solve them directly (base case)

Combine:

- Merge solutions to subproblems to obtain solution for original problem



Observation

Divide: $D(n)$ time

Conquer: Recurse on smaller problems of size $s_1, \ldots, s_k$

Combine: $C(n)$ time

Recurrence:

- $T(n) = D(n) + \sum_{i \in [k]} T(s_i) + C(n)$

Many divide and conquer algorithms have recurrences are of form:

- $T(n) = a \cdot T(n/b) + f(n)$

a and b are constants

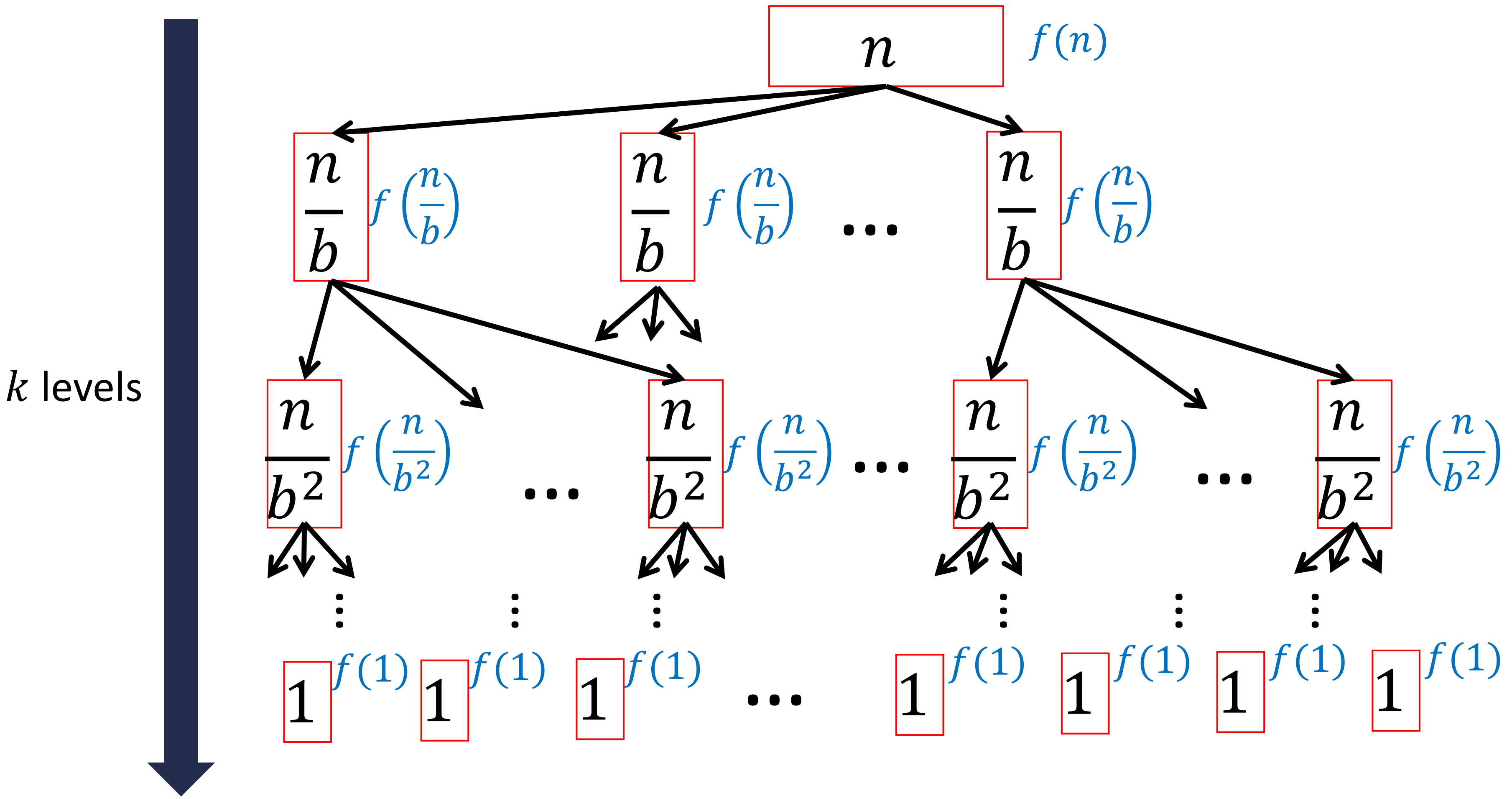
Mergesort: $T(n) = 2T(n/2) + n$

Divide and Conquer Multiplication: $T(n) = 4T(n/2) + 5n$

Karatsuba Multiplication: $T(n) = 3T(n/2) + 8n$

General Recurrence

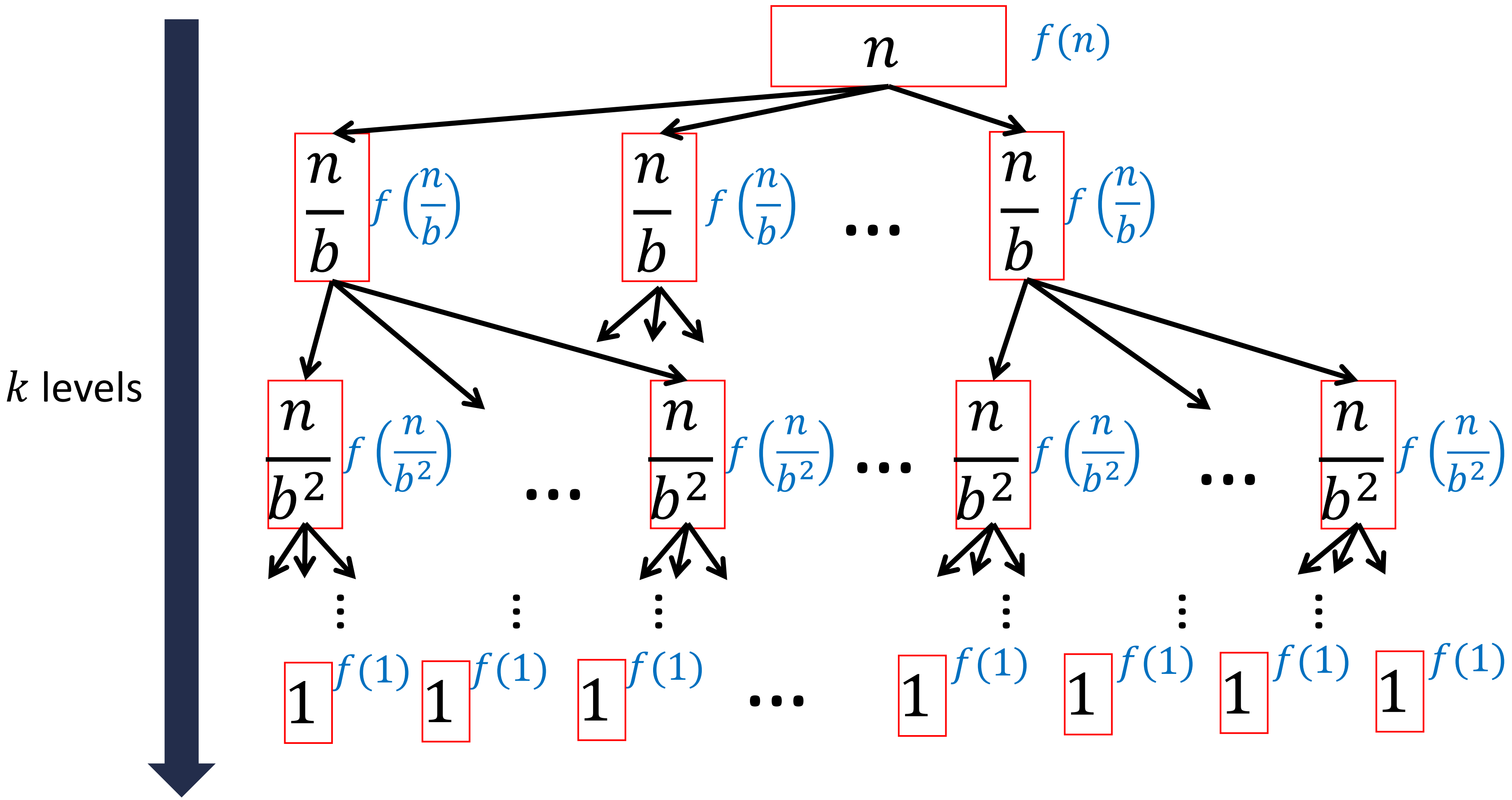
$$T(n) = aT(n/b) + f(n)$$



Number of subproblems	Cost of subproblem
1	$f(n)$
a	$f(n/b)$
a^2	$f(n/b^2)$
a^k	$f(n/b^k)$

General Recurrence

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



Number of subproblems	Cost of subproblem
1	$f(n)$
a	$f(n/b)$
a^2	$f(n/b^2)$
$a^{\log_b n} = n^{\log_b a}$	$f(n/b^k)$

Three Cases

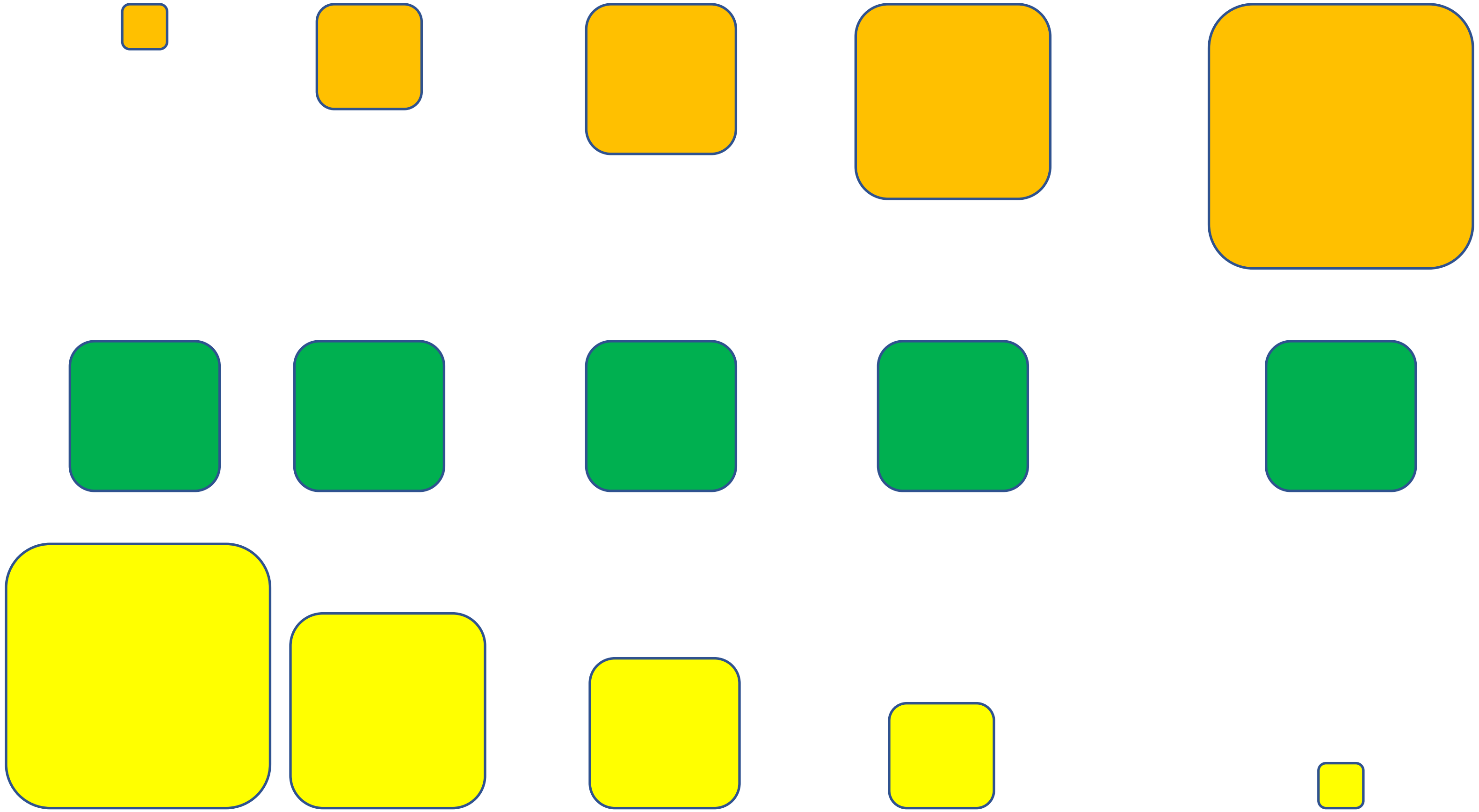
$$T(n) = f(n) + af\left(\frac{n}{b}\right) + a^2f\left(\frac{n}{b^2}\right) + a^3f\left(\frac{n}{b^3}\right) + \cdots + a^kf\left(\frac{n}{b^k}\right)$$

$$k = \log_b n$$

Case 1:
Most work happens at the leaves

Case 2:
Work happens consistently throughout

Case 3:
Most work happens at top of tree



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 $\varepsilon > 0$	$T(n) \in \Theta\left(n^{\delta}\right)$

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	$f(n) \in \Theta\left(n^{\delta}\right)$	$T(n) \in \Theta\left(n^{\delta} \log n\right)$

Master Theorem

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

$$\delta = \log_b a$$

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

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

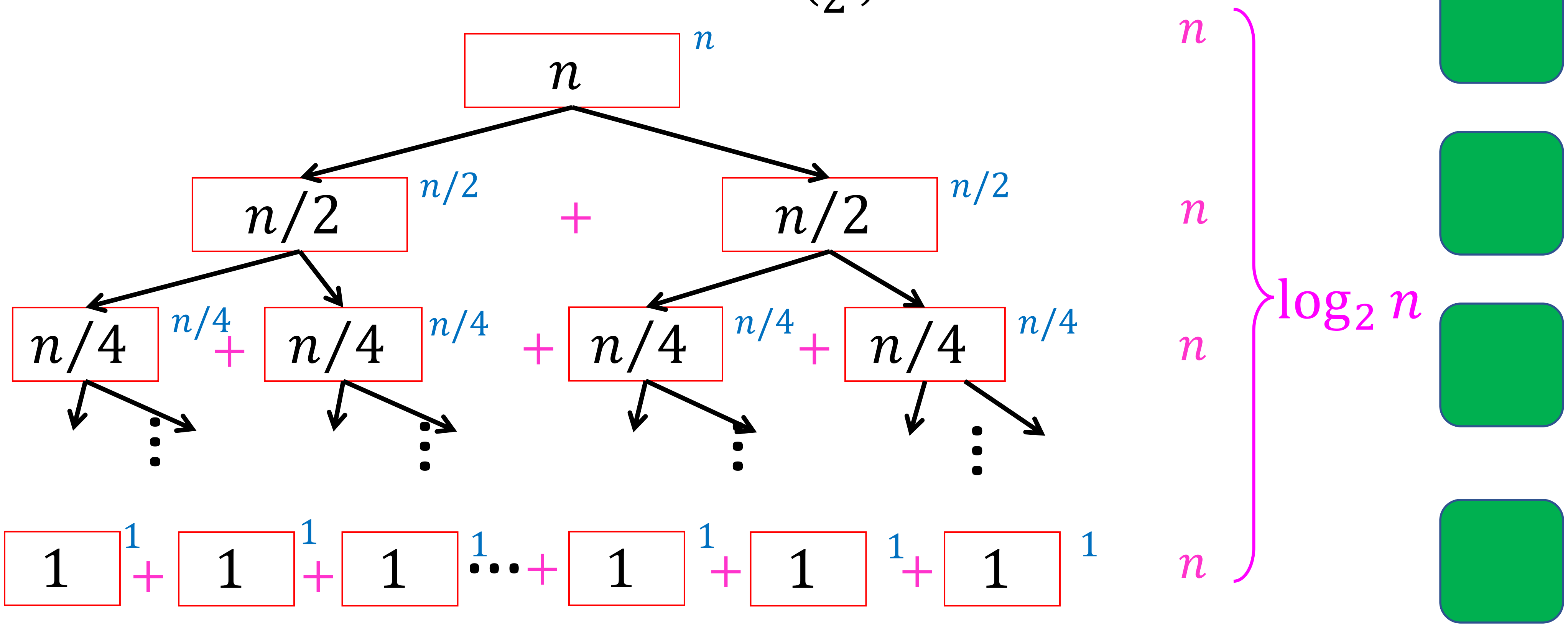
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Case 2

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

Tree method

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



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

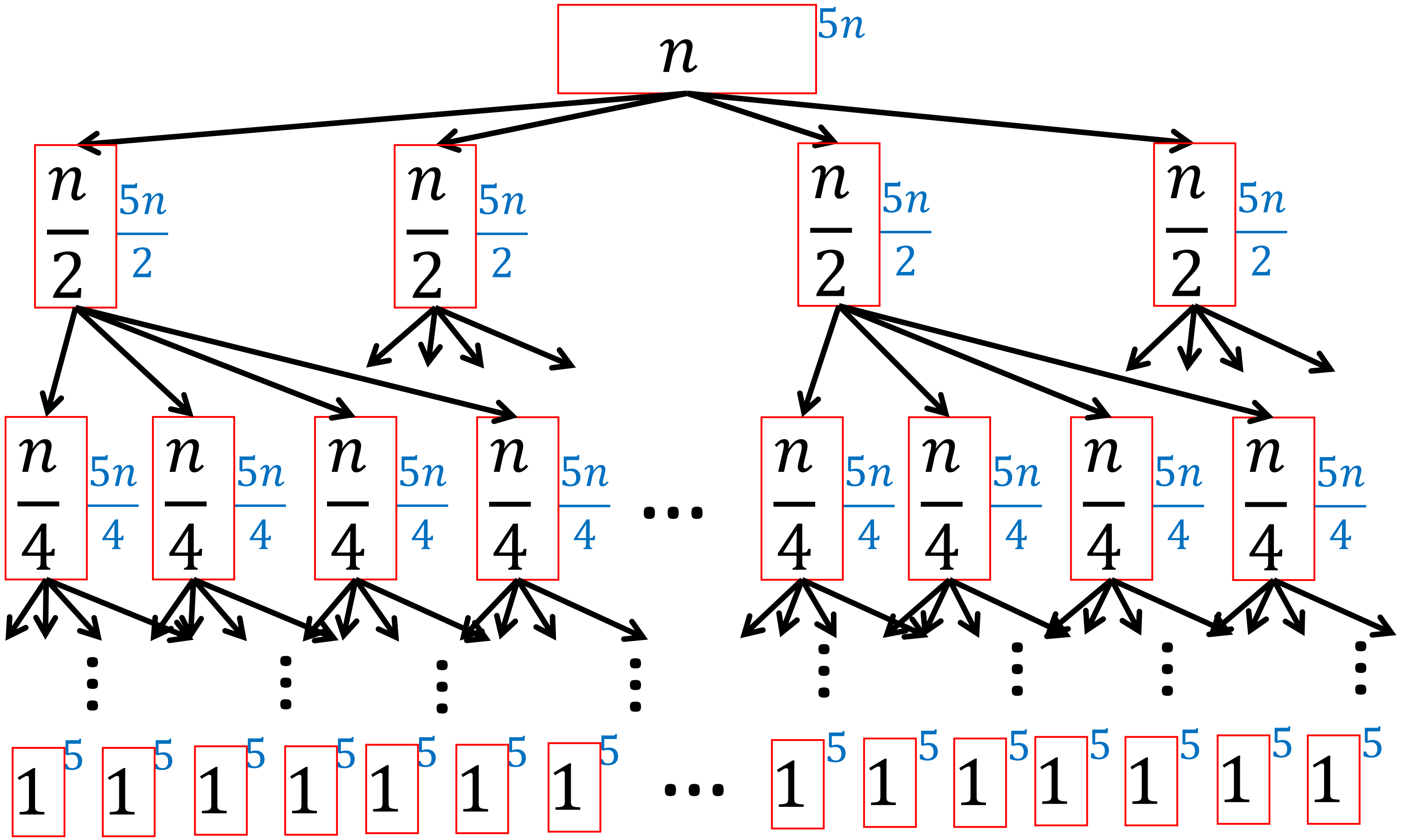
$$T(n) = 4T\left(\frac{n}{2}\right) + 5n$$

Case 1

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

Tree method

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



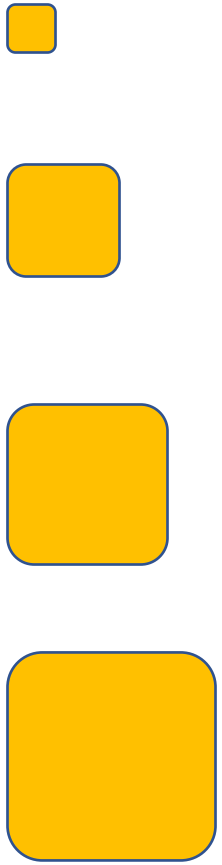
$5n$

$\frac{4}{2} \cdot 5n$

$\frac{16}{4} \cdot 5n$

⋮

$2^{\log_2 n} \cdot 5n$



Tree method

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

Cost is increasing with the recursion depth (due to large number of subproblems)

Most of the work happening in the leaves

$$5n$$

$$\frac{4}{2} \cdot 5n$$

$$\frac{16}{4} \cdot 5n$$

$$\vdots$$

$$2^{\log_2 n} \cdot 5n$$

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

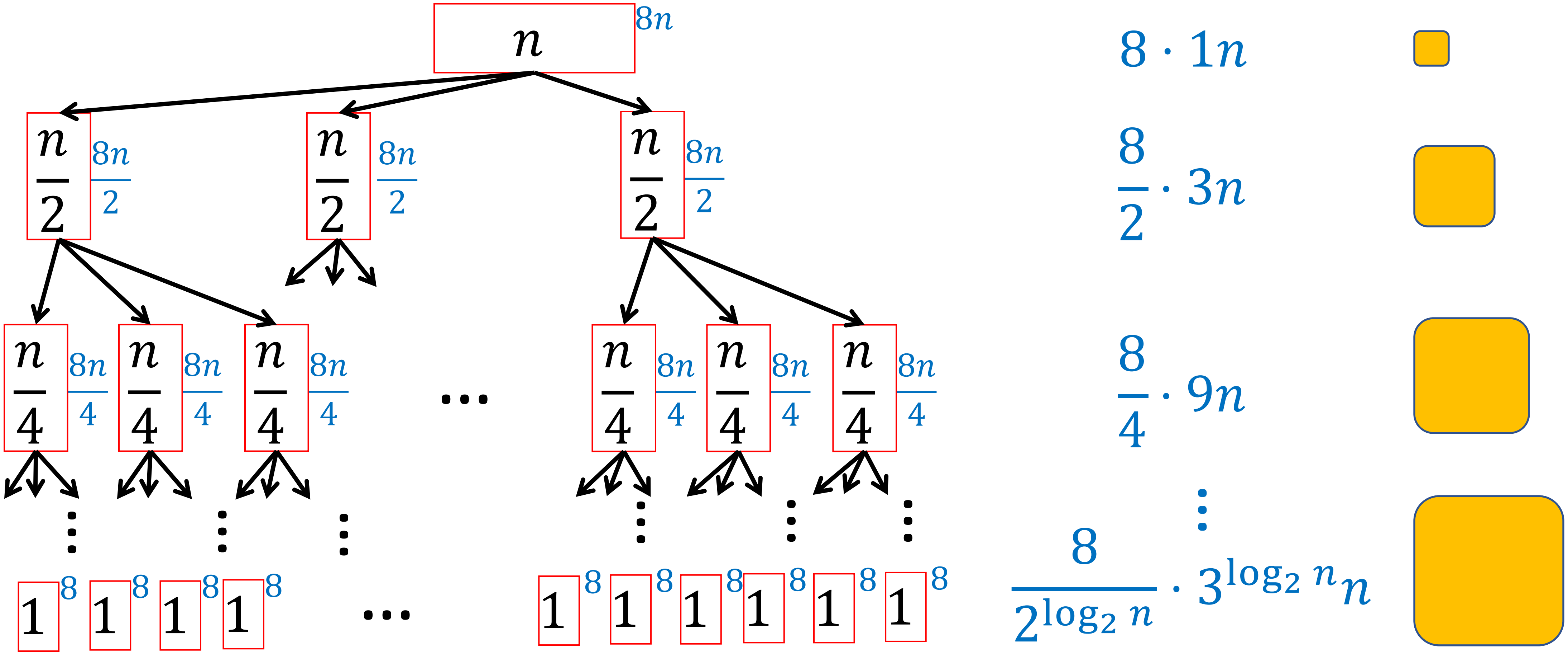
$$T(n) = 3T\left(\frac{n}{2}\right) + 8n$$

Case 1

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

Karatsuba

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



$$8 \cdot 1n$$

$$\frac{8}{2} \cdot 3n$$

$$\frac{8}{4} \cdot 9n$$

$$\vdots$$

$$\frac{8}{2^{\log_2 n}} \cdot 3^{\log_2 n} n$$

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

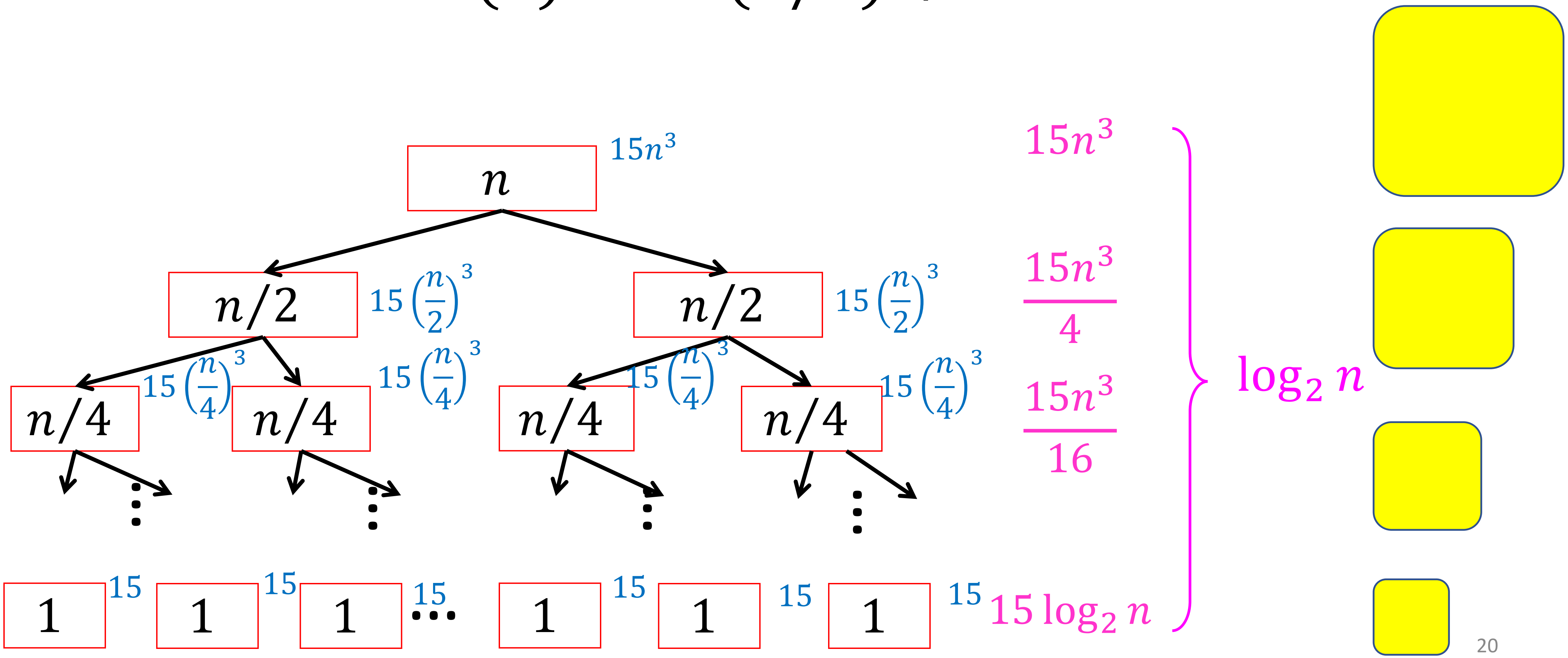
$$\Theta(n^3)$$

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 4 (Visually)

$$T(n) = 2T(n/2) + 15n^3$$

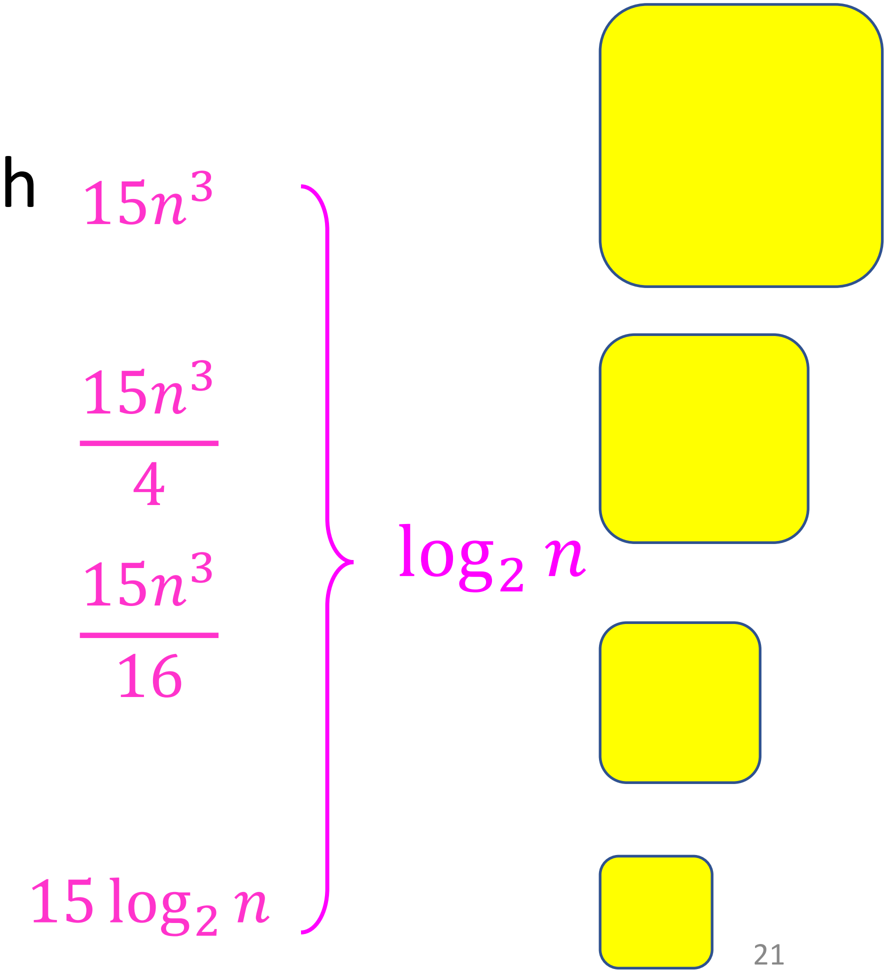


Master Theorem Example 4 (Visually)

$$T(n) = 2T(n/2) + 15n^3$$

Cost is <u>decreasing</u> with the recursion depth (due to high *non-recursive* cost)

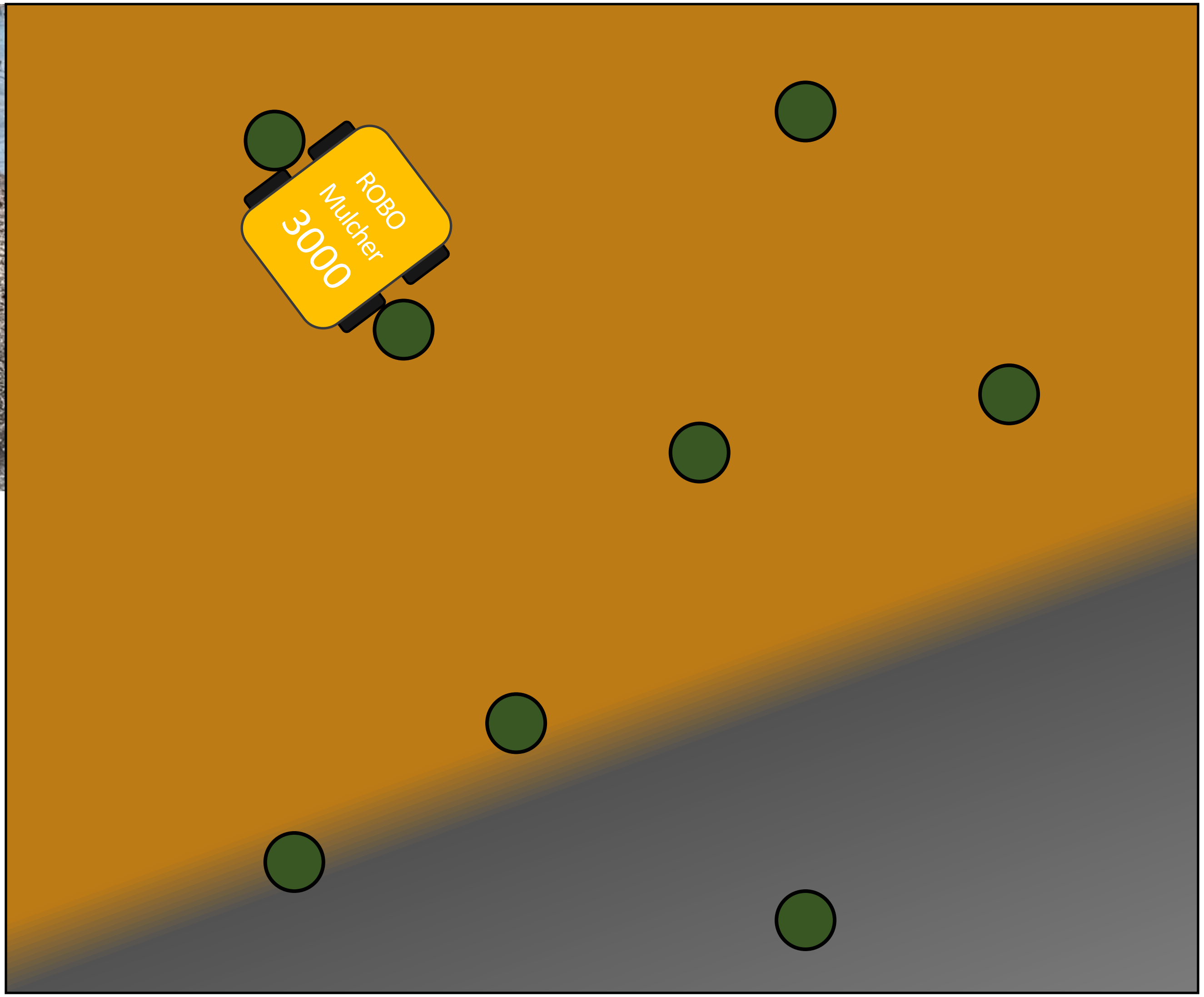
Most of the work happening at the top



Robbie’s Yard

There Has to be an Easier Way!

Constraints: Trees and Plants



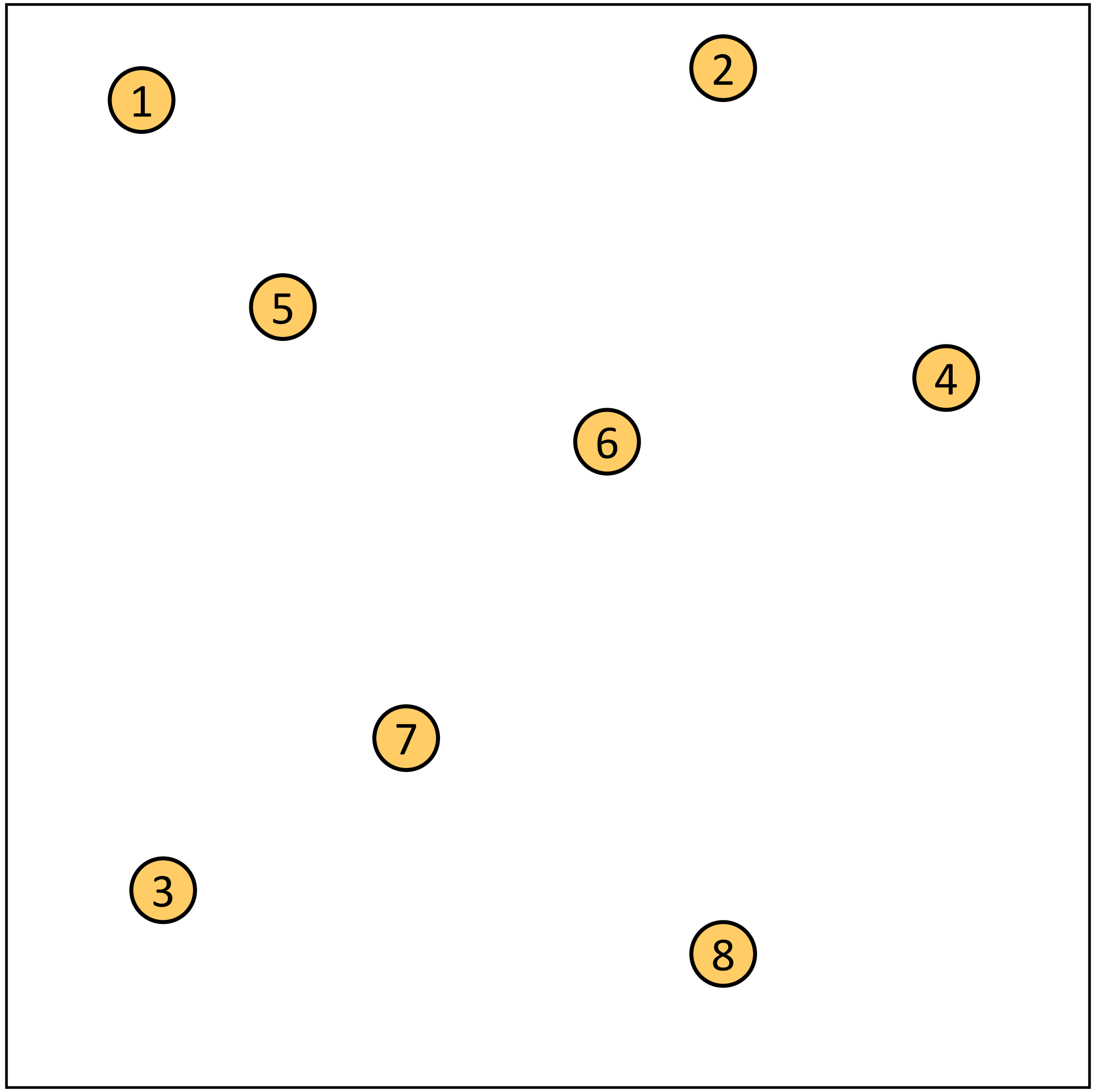
How wide can the robot be?

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

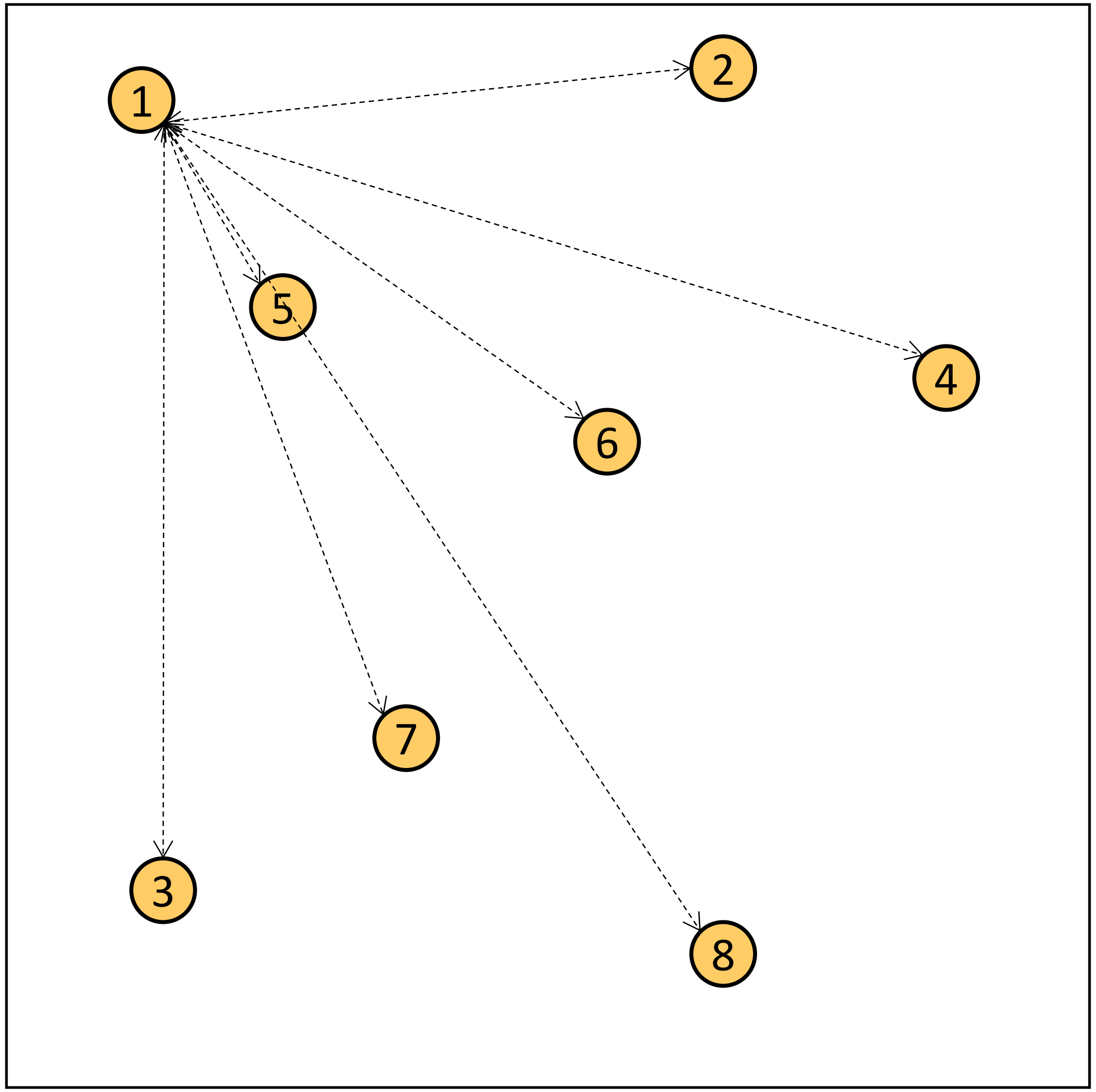
Given: A list of points

Return: Pair of points with smallest distance apart

Algorithm: Test every pair of points, return the closest

Running Time: $O(n^2)$

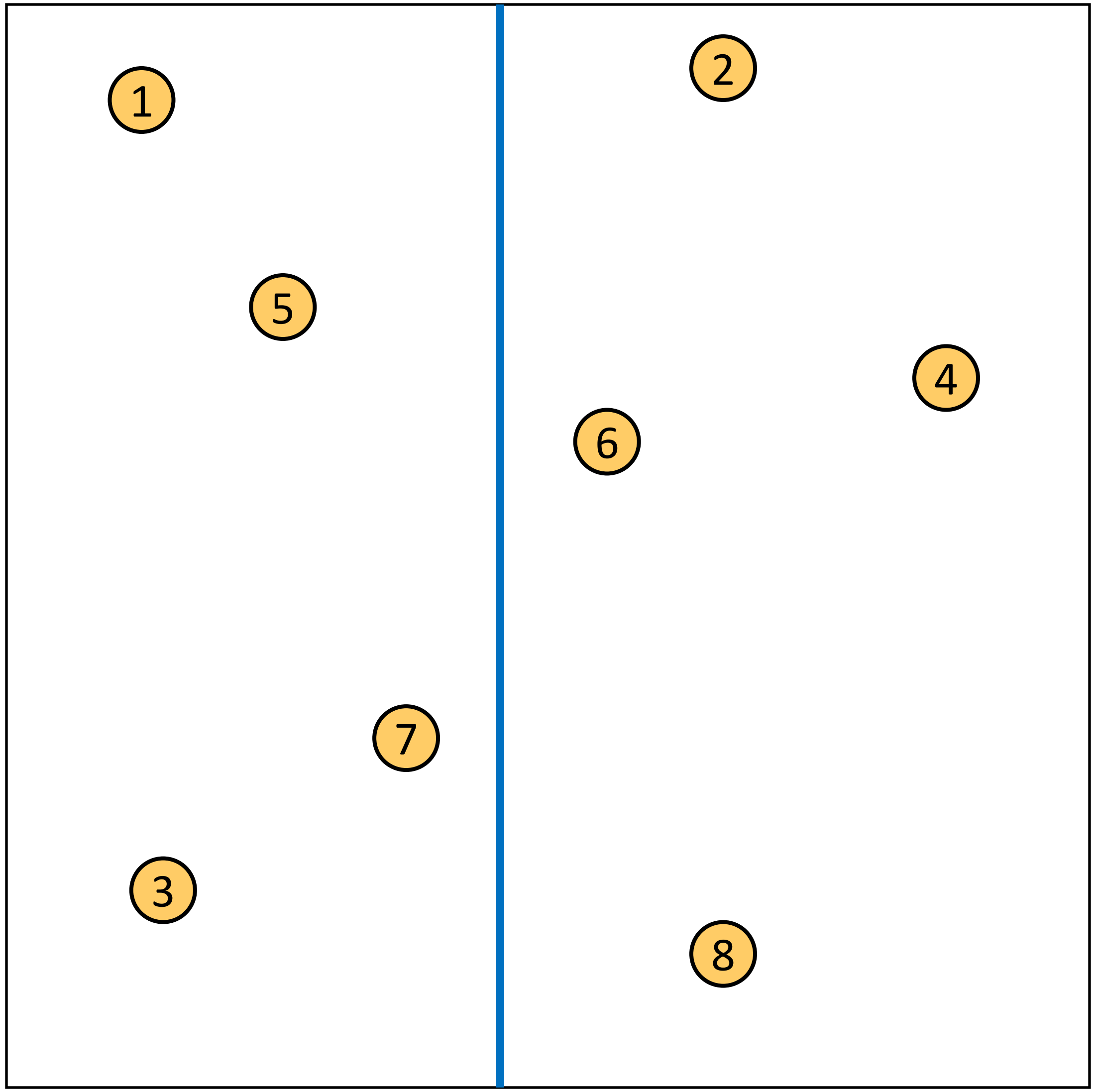
Goal: $O(n \log n)$



Closest Pair of Points: Divide and Conquer

Divide: How?

At median x coordinate



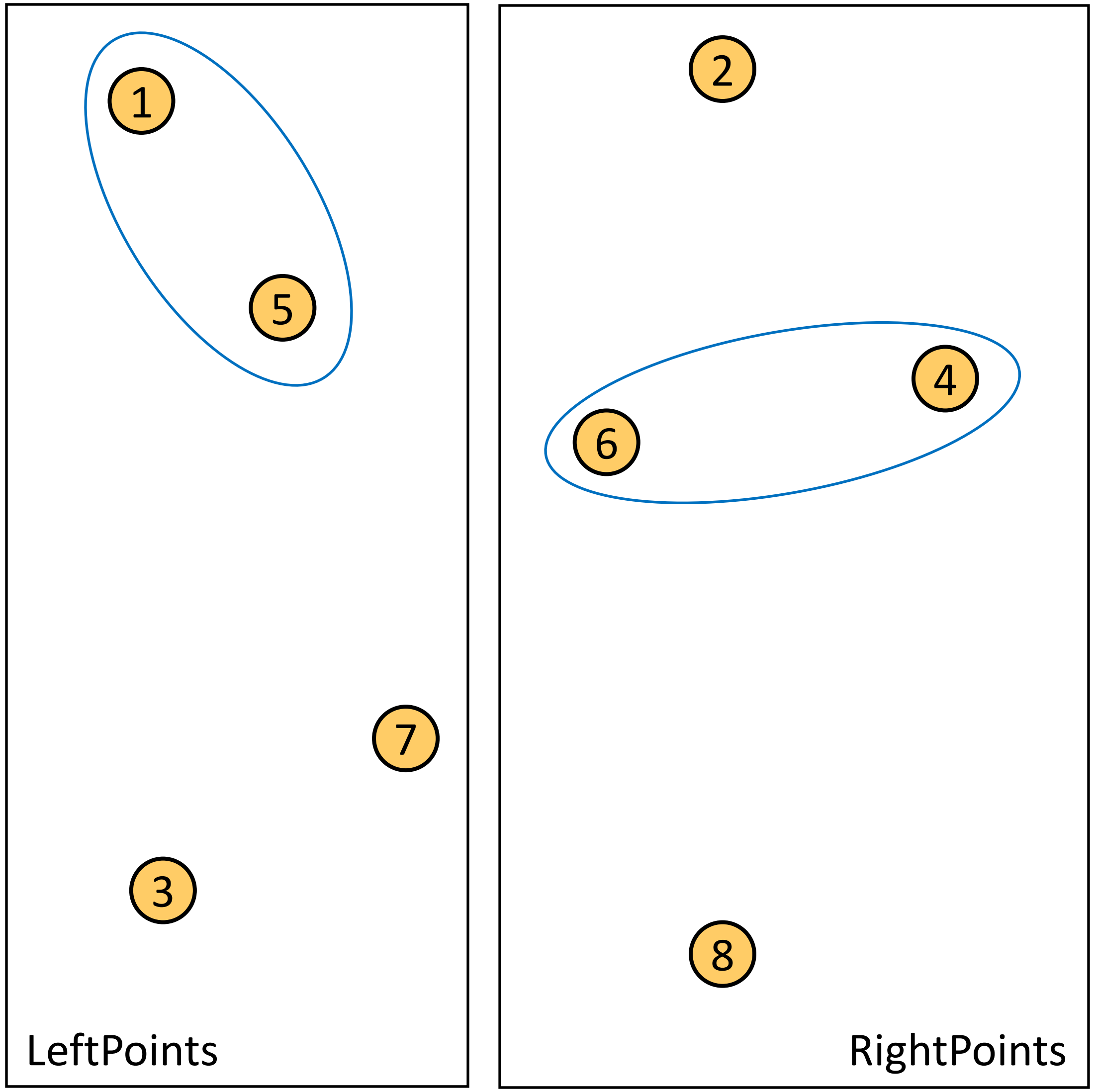
Closest Pair of Points: Divide and Conquer

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

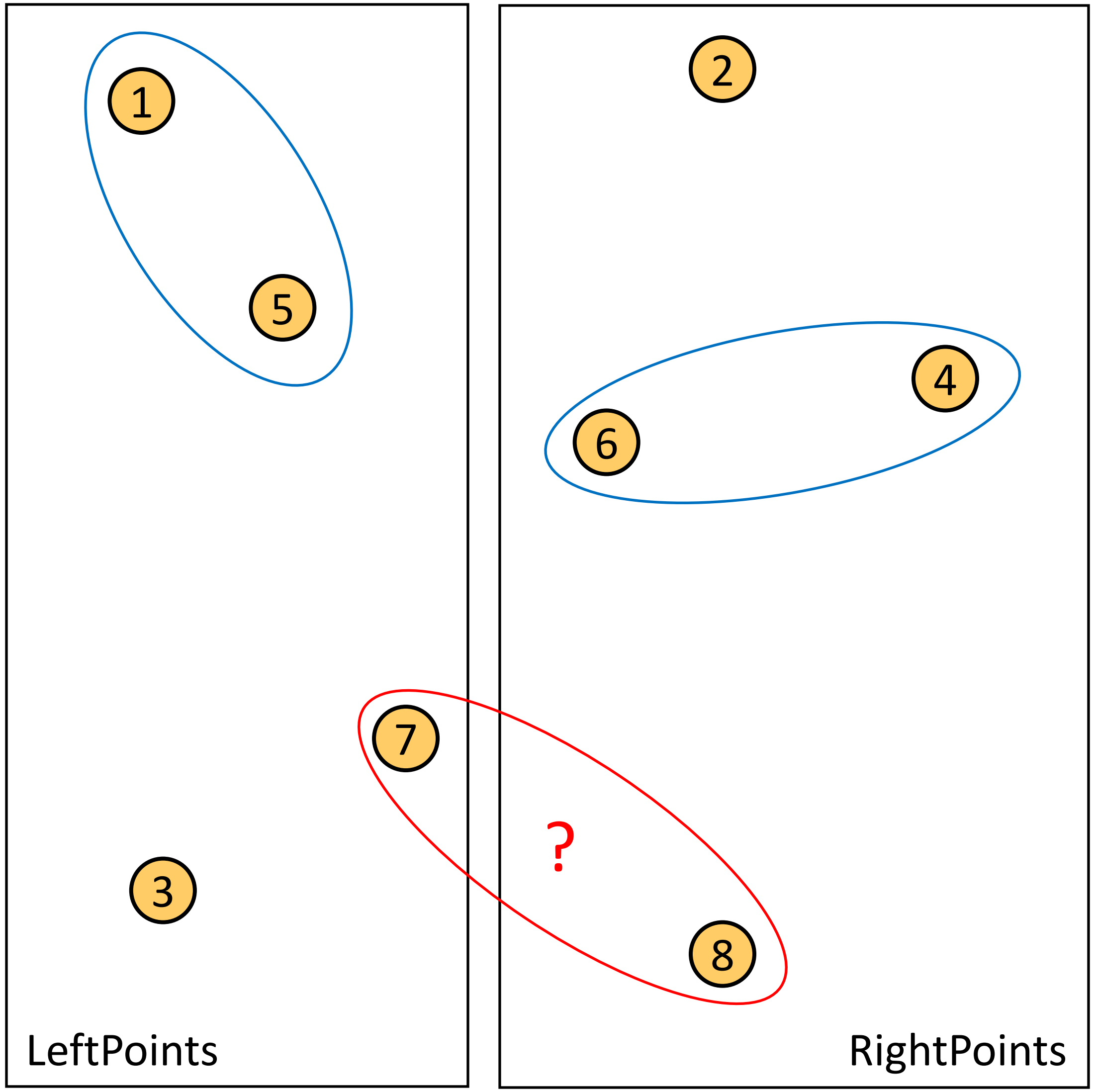
Conquer:

Recursively find closest pairs from LeftPoints and RightPoints

Combine:

Return smaller of left and right pairs

Problem?



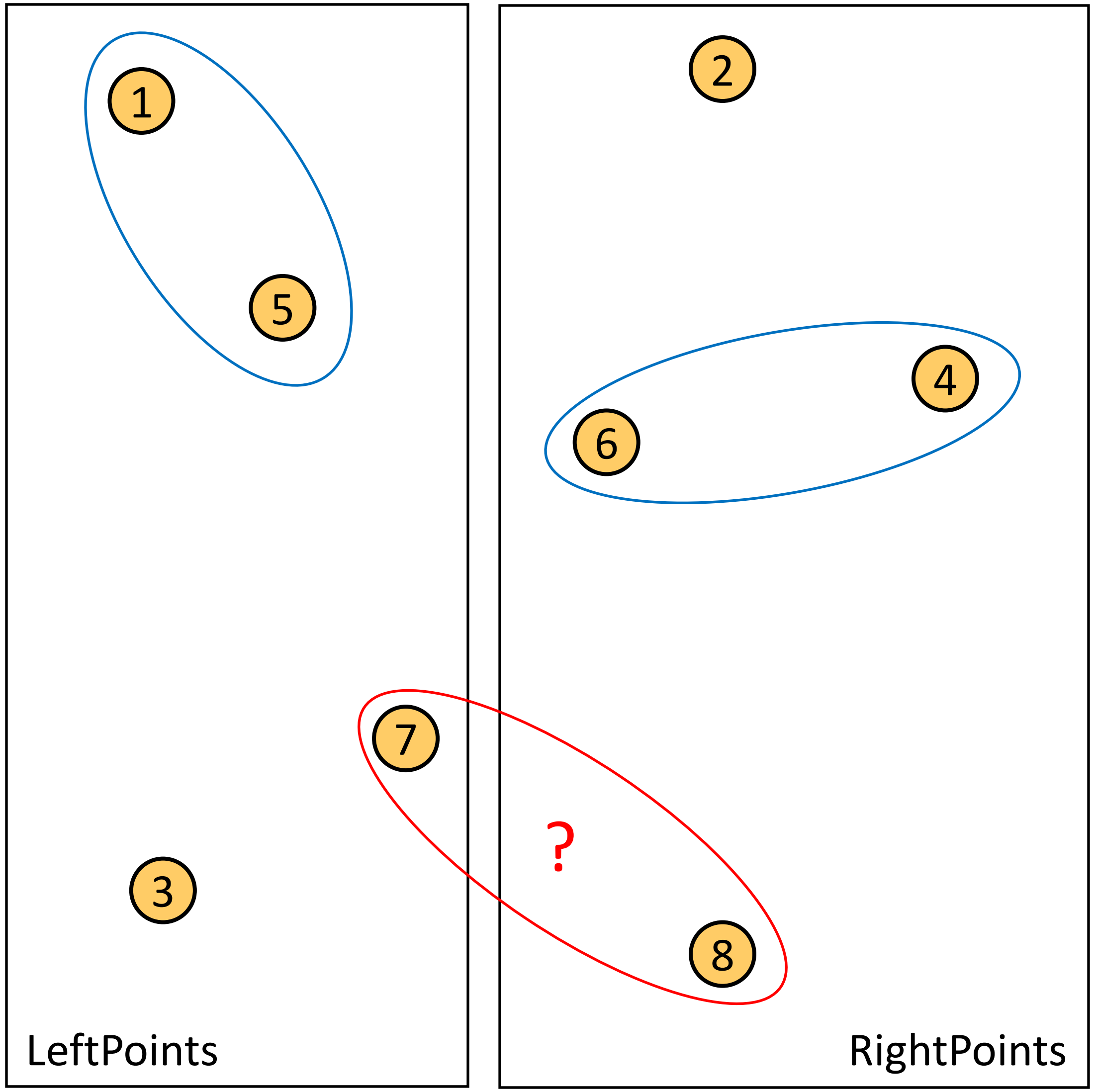
Closest Pair of Points: Divide and Conquer

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



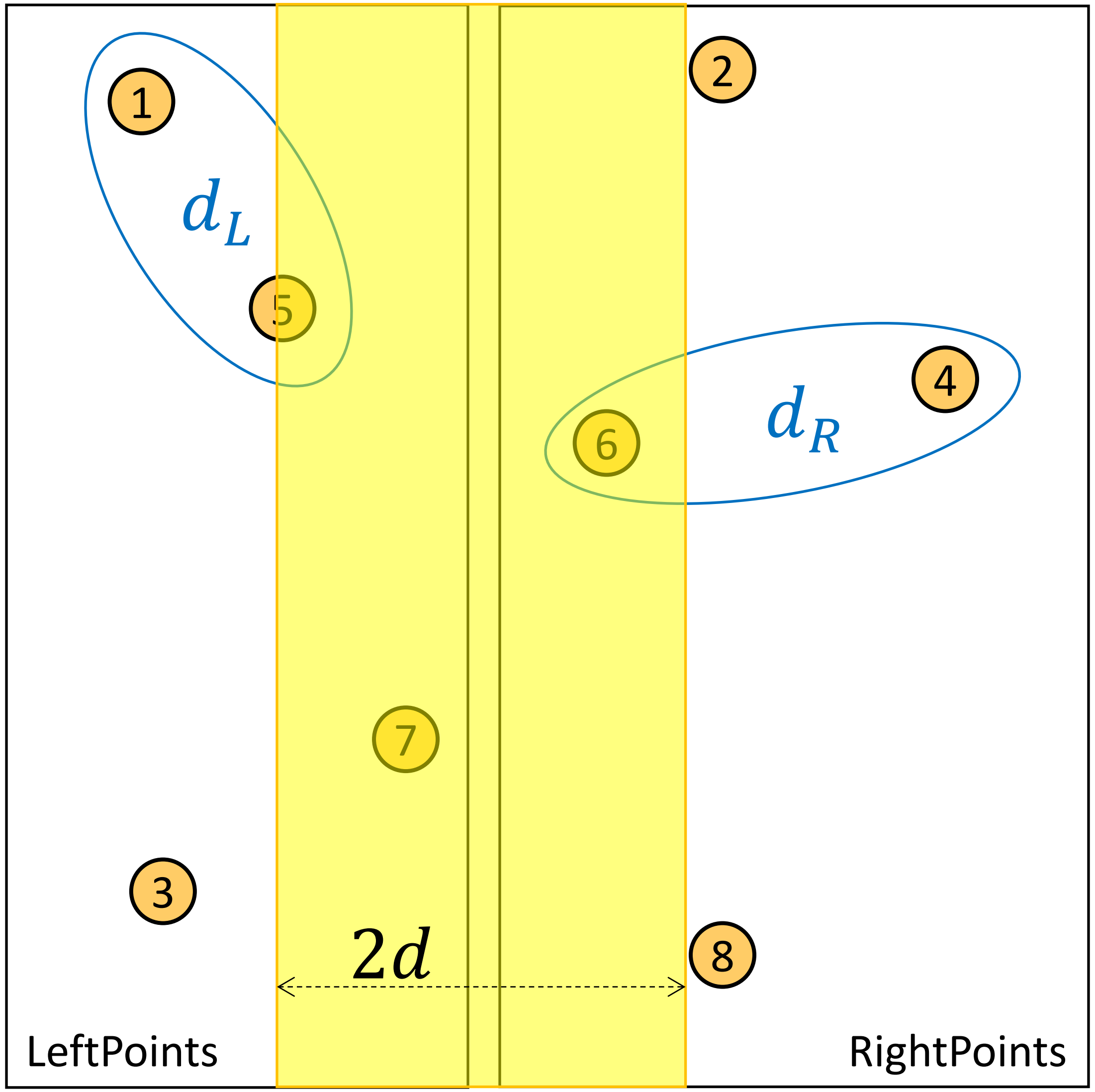
Spanning the Cut

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?



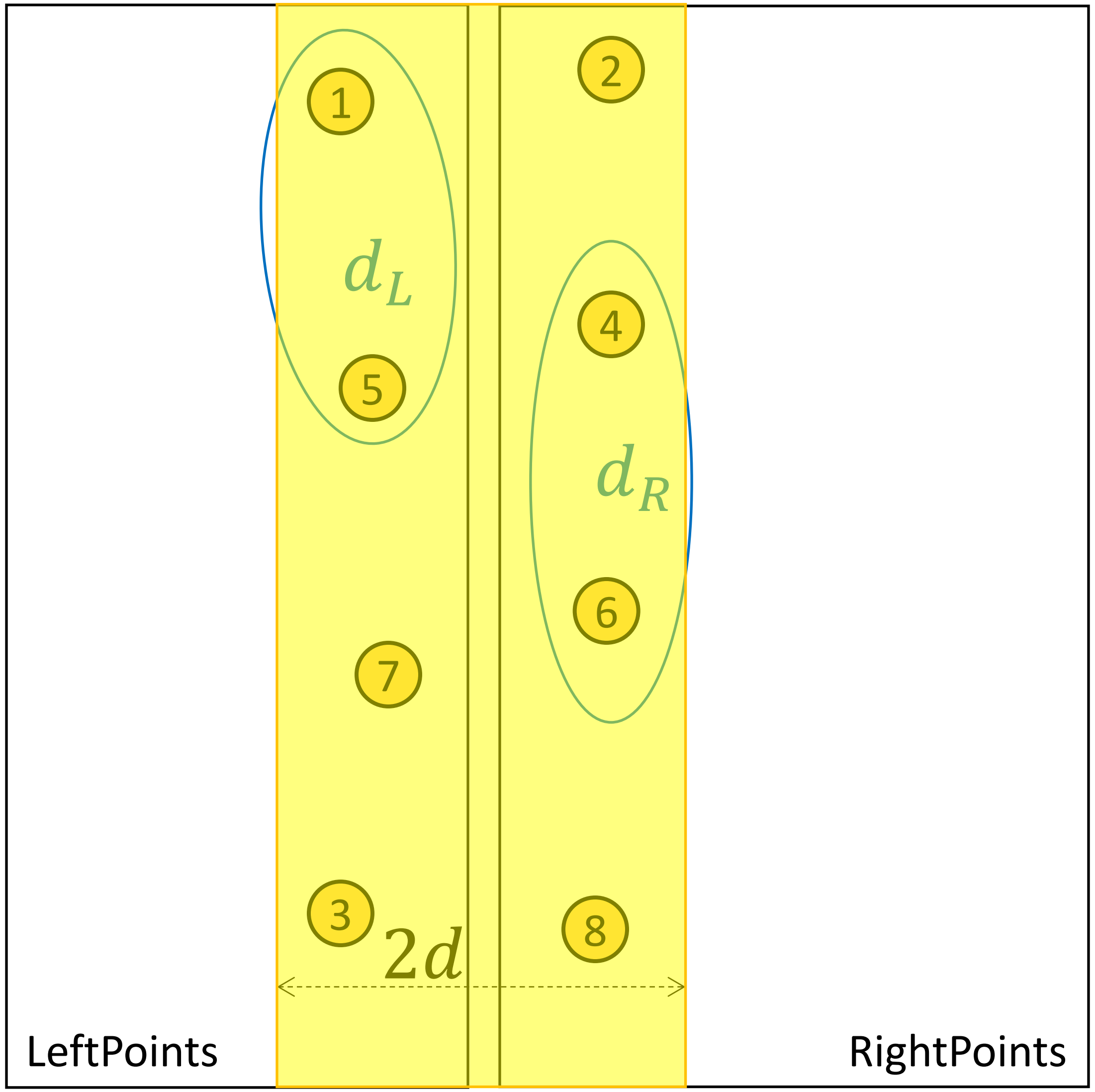
Spanning the Cut

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

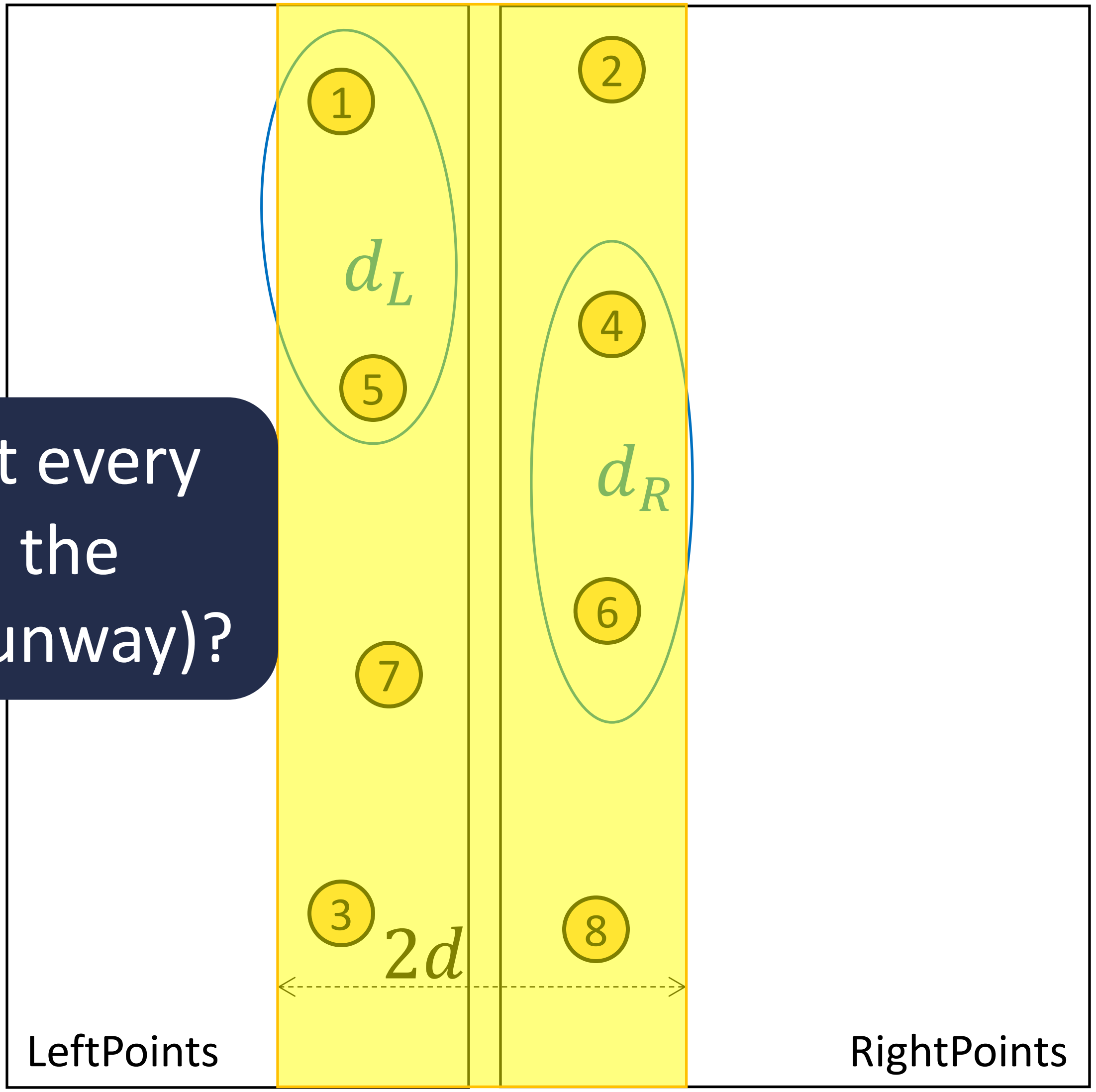
Compare all p
$d = \min\{d_L,$

How many ar

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

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

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



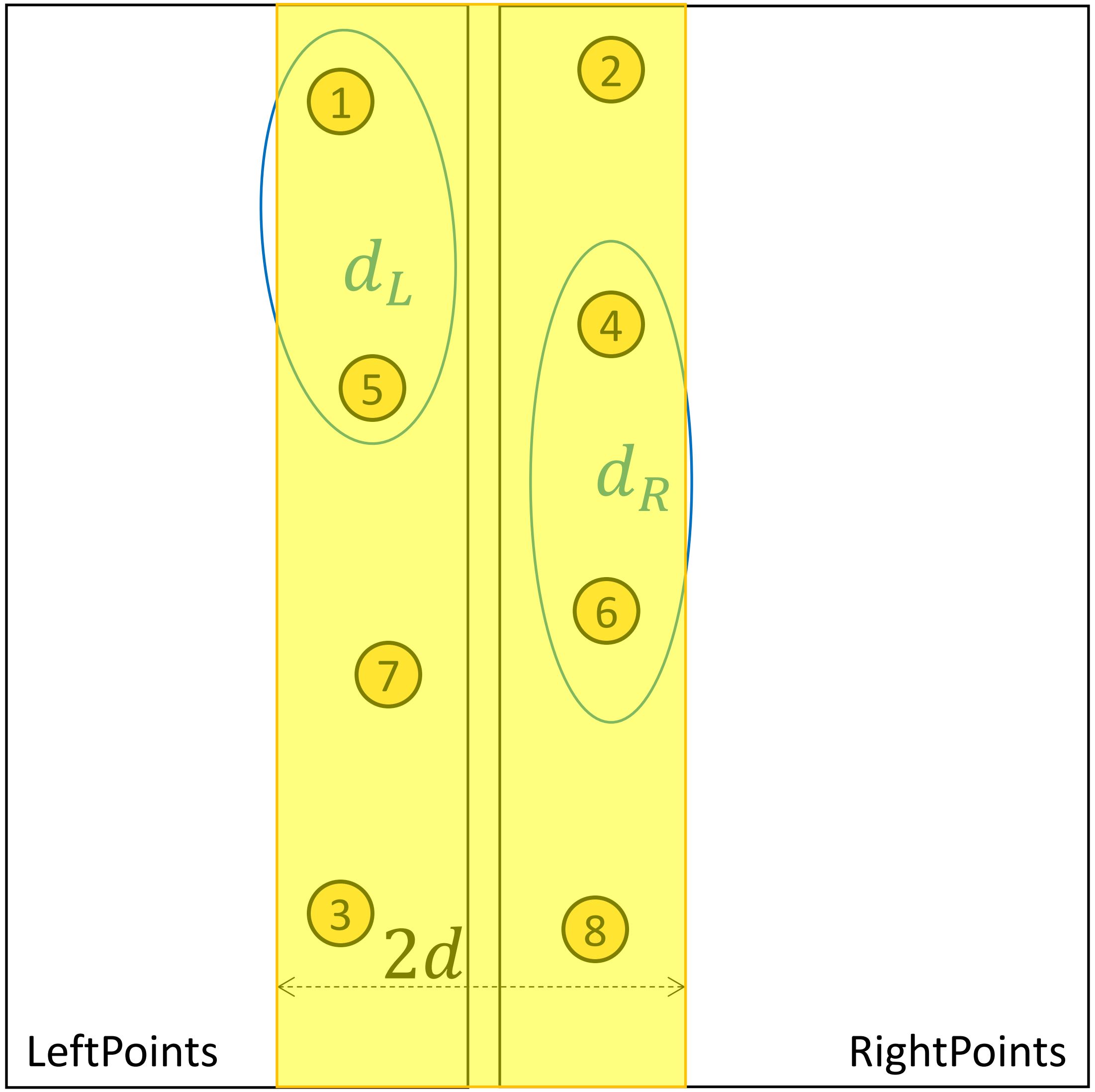
Spanning the Cut

Case 2: Closest pair spans our "cut"

Need to test points across the cut

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

Only need to test points within distance d of each another



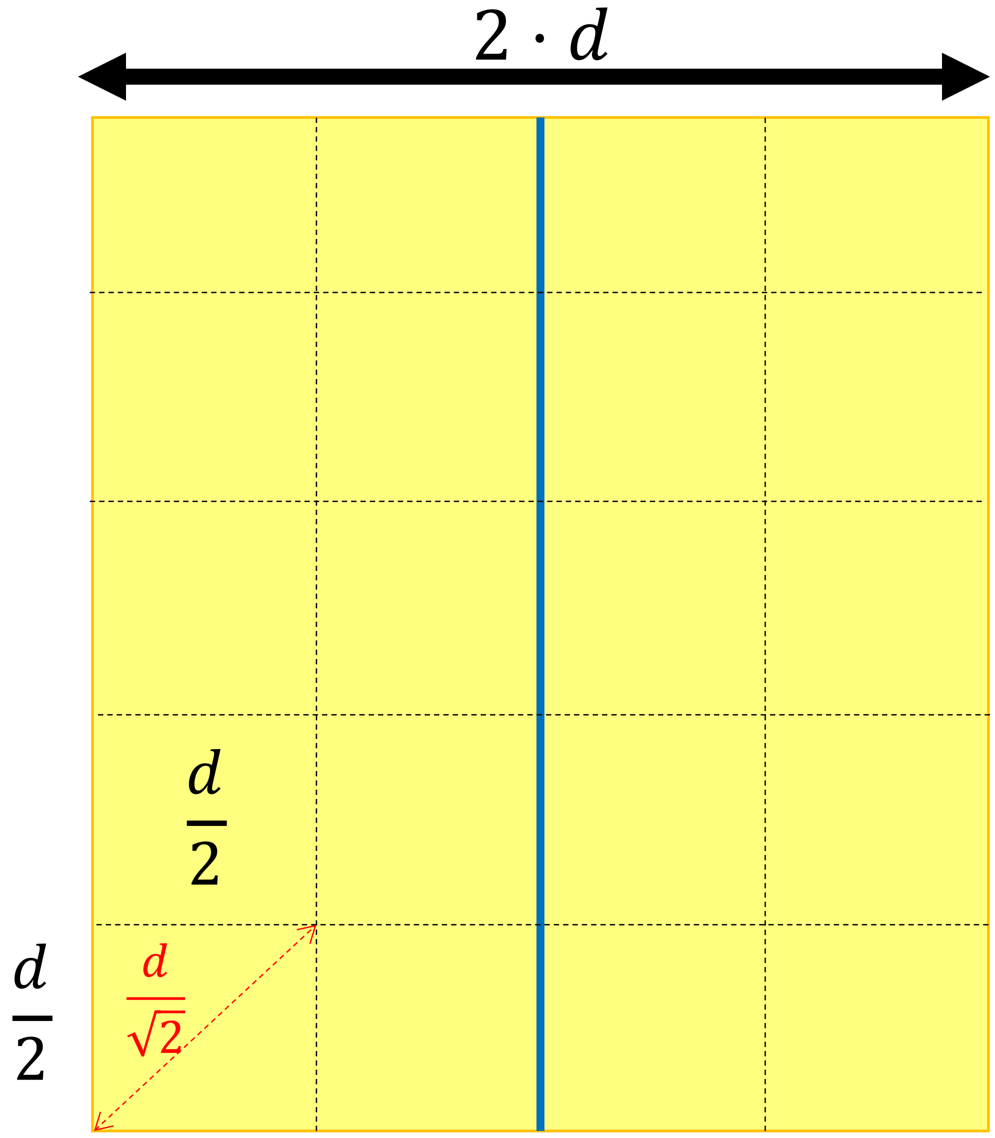
Reducing Search Space

Case 2: Closest pair spans our "cut"

Need to test points across the cut

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

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



Reducing Search Space

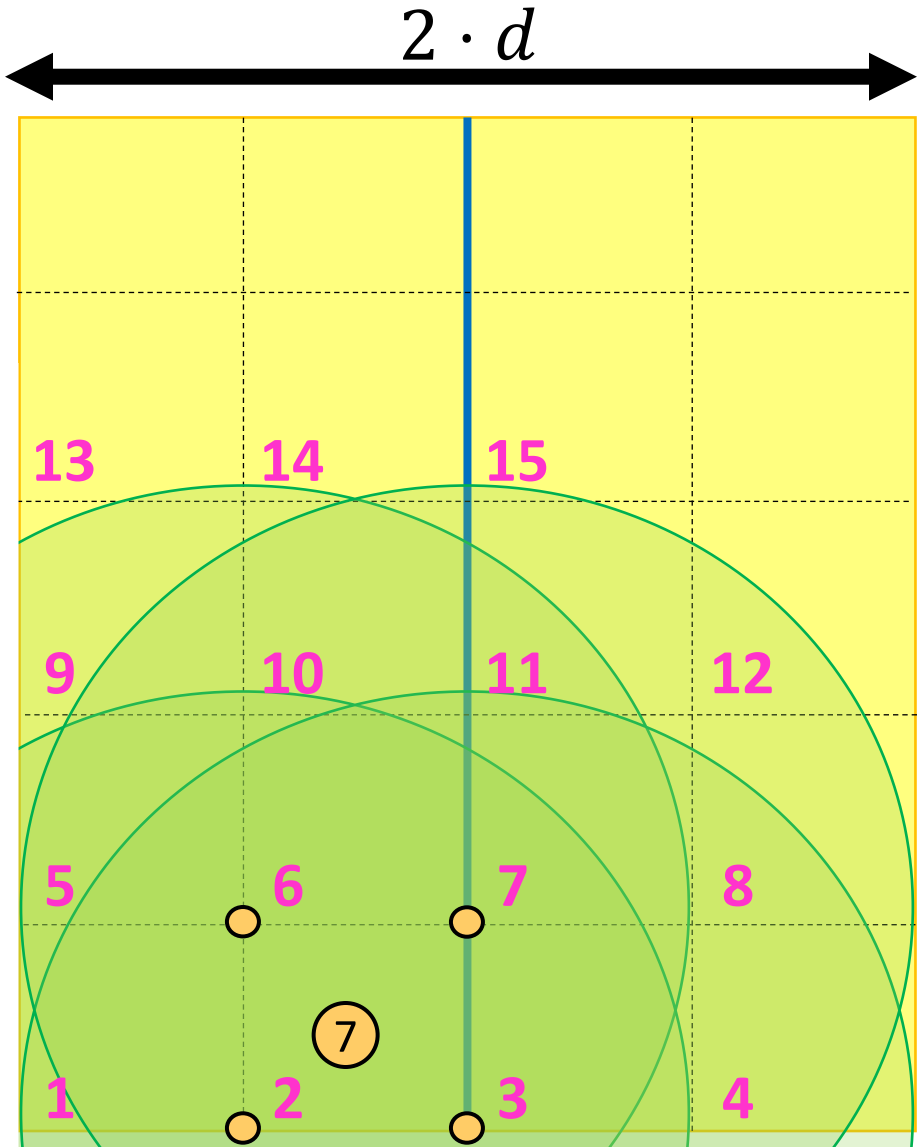
Case 2: Closest pair spans our "cut"

Need to test points across the cut

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

How many squares can contain a point $< d$ away?

at most **15**



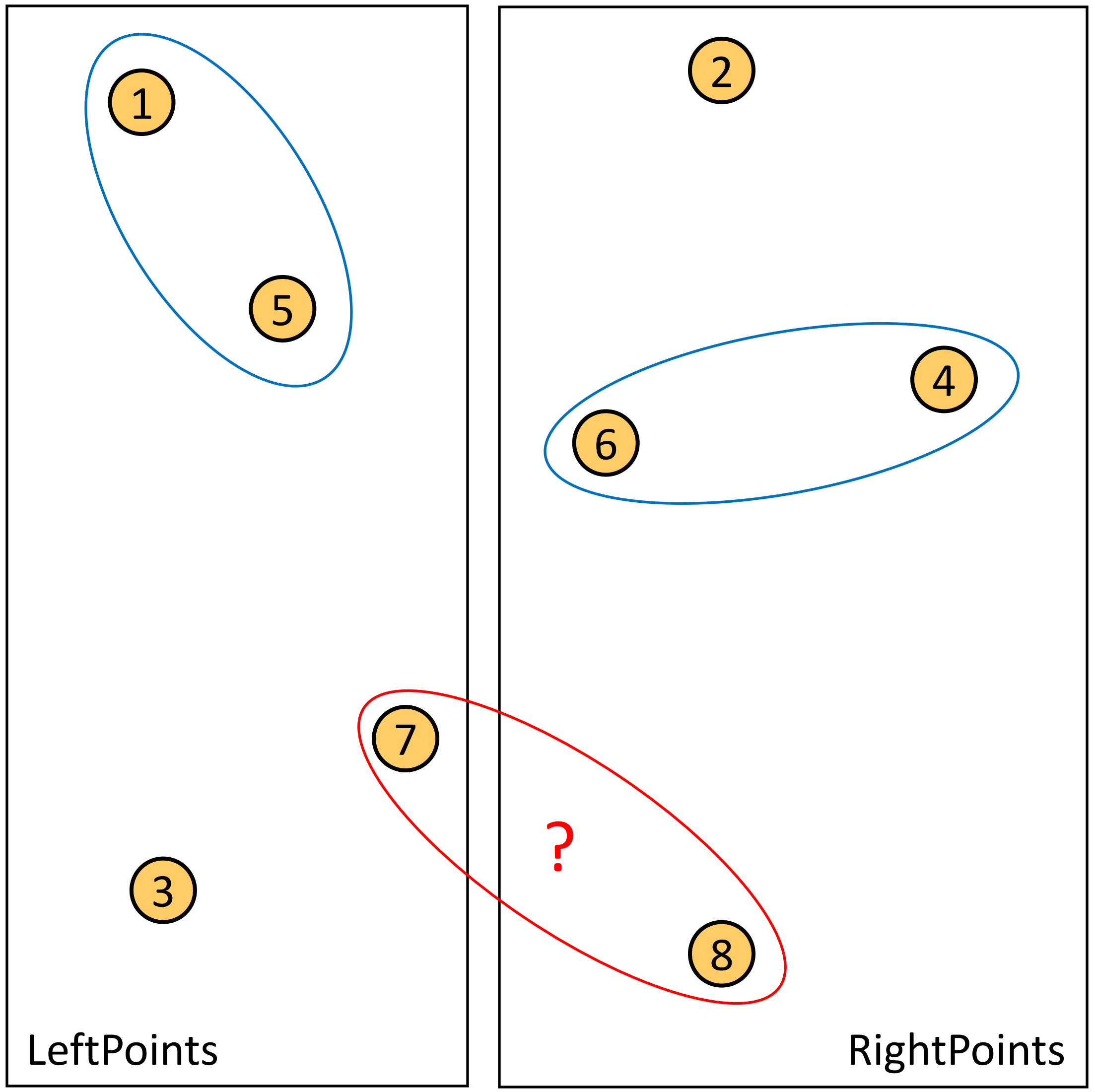
Closest Pair of Points: Divide and Conquer

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 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

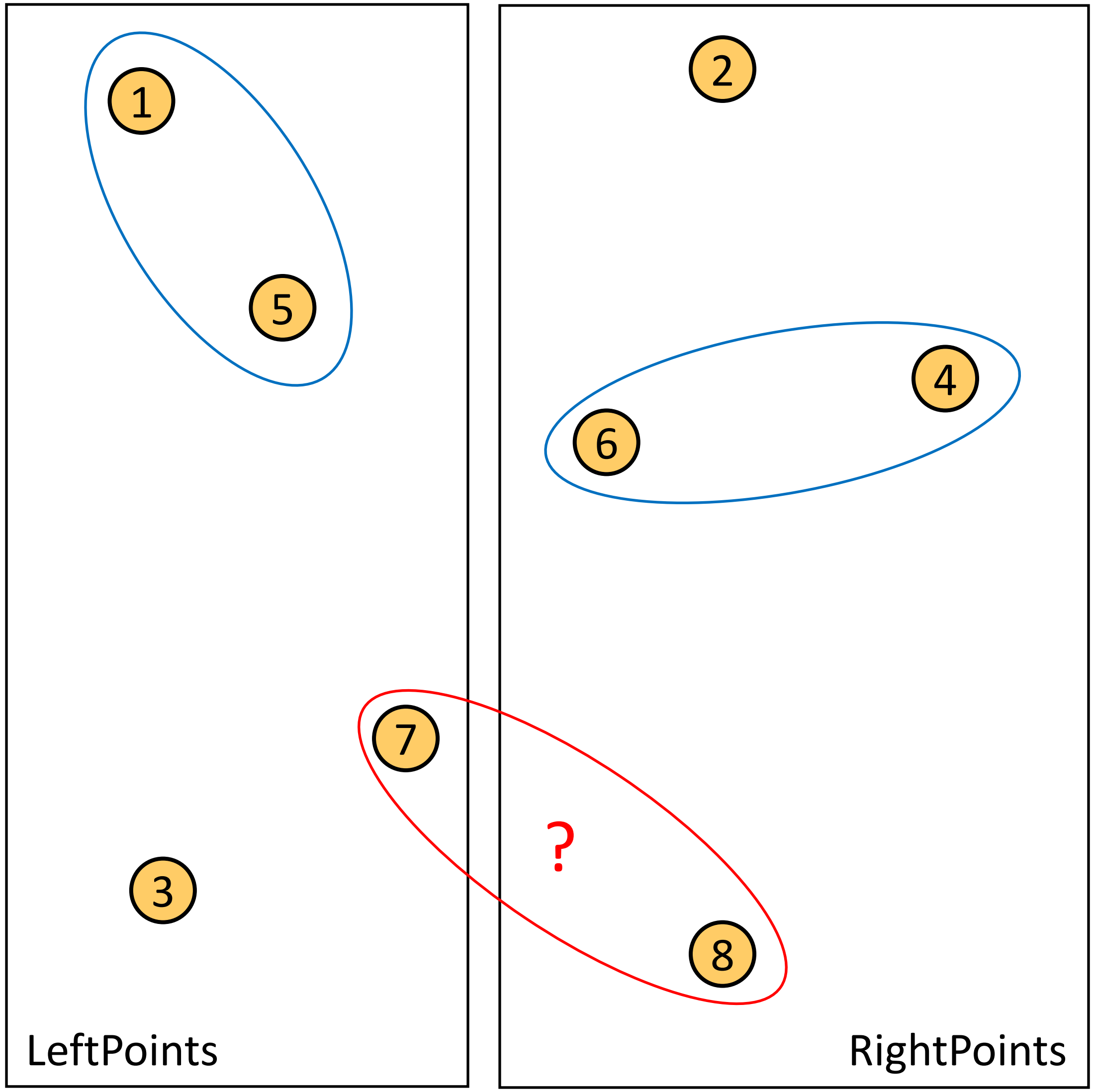
Initialization: Sort points by x-coordinate

Divide: Partition points into two lists of points

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

Combine:

- Construct list of points in t boundary
- 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

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

Combine:

- Construct list of points in the boundary
- 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 Boundary

LeftPoints:

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

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

RightPoints:

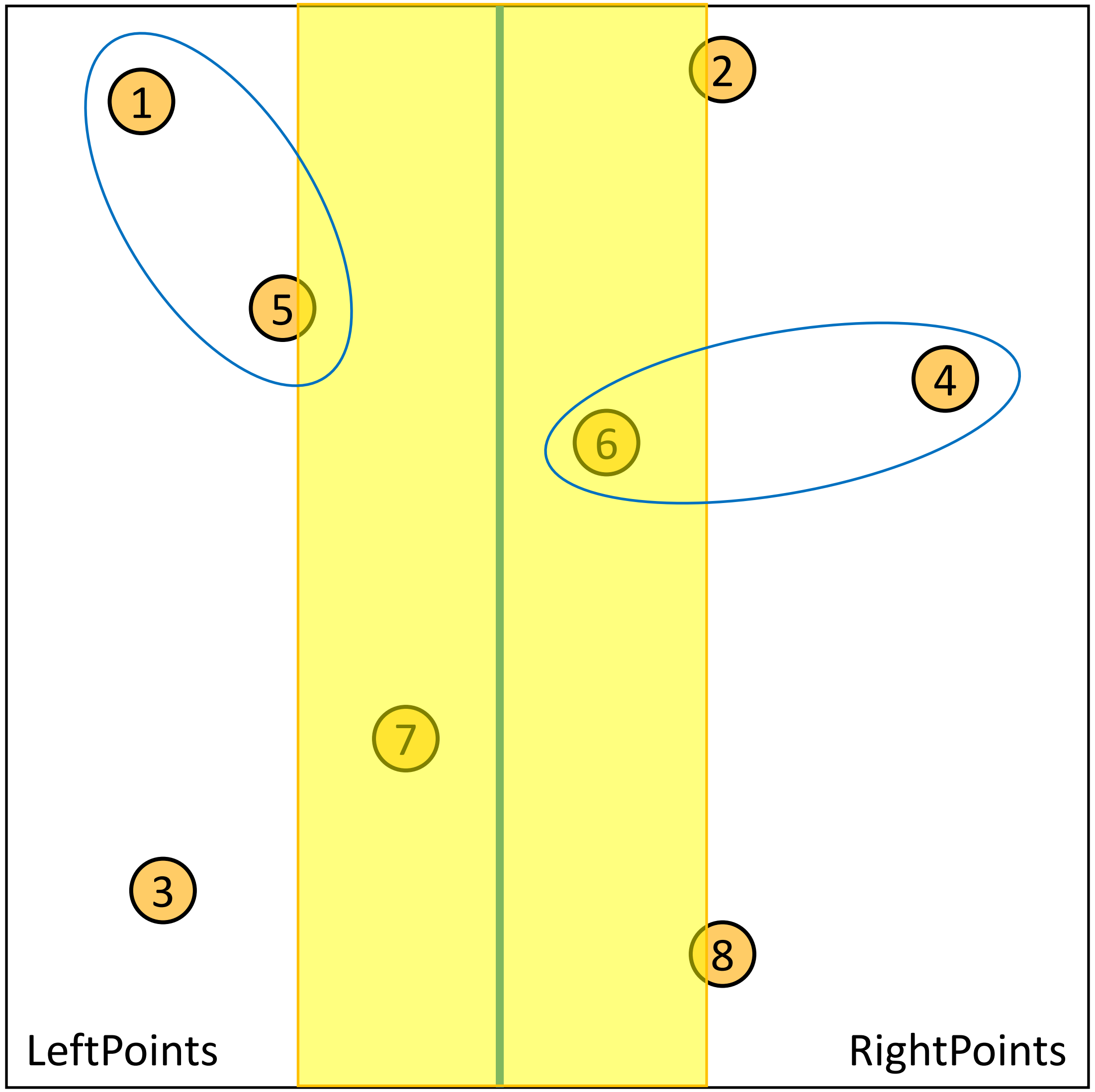
Closest Pair: $(4,6)$, $d_{4,6}$

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

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

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

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

Combine:
- Construct list of points in the boundary
- 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

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?

$\Theta(n \log n)$

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

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

$2T(n/2)$ **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