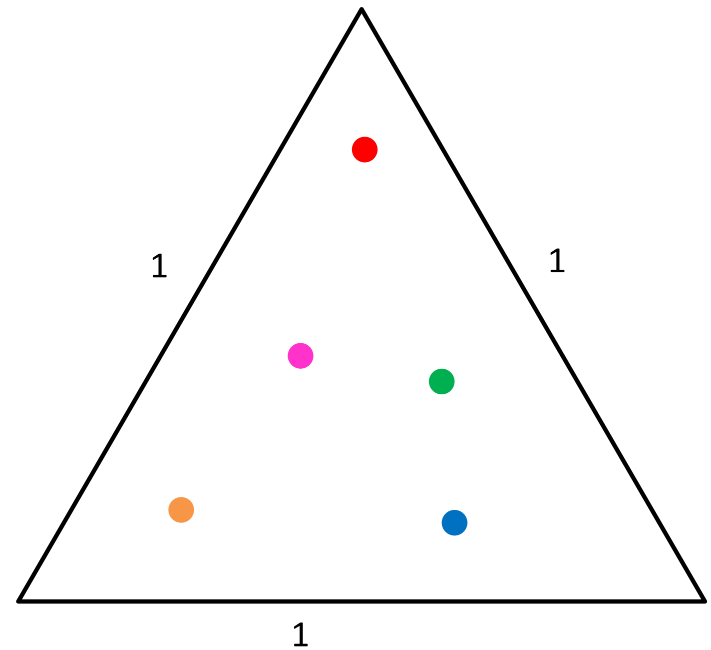
CS4102 Algorithms

Spring 2020

Warm up

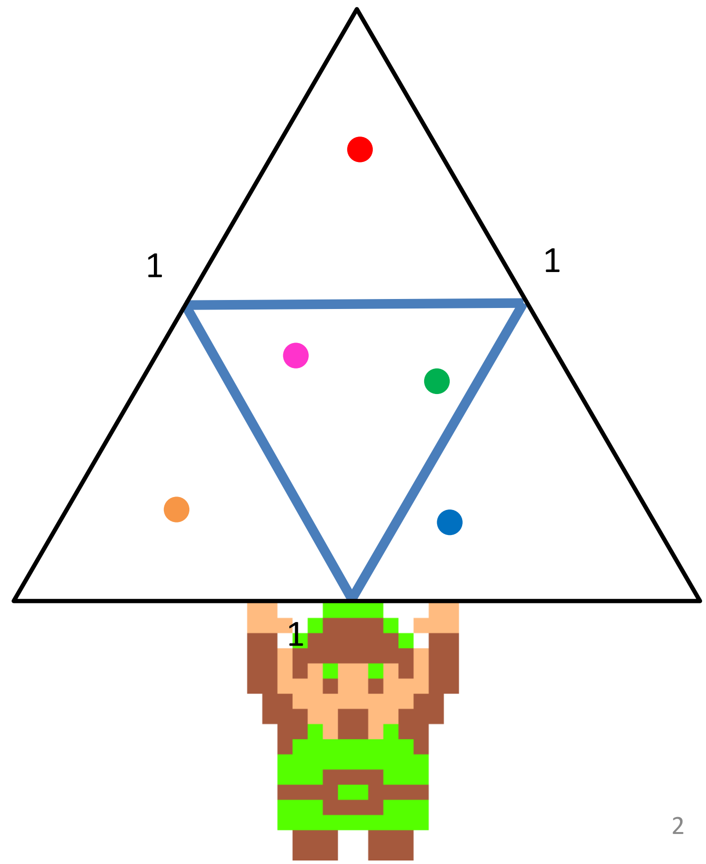
Given 5 points on the unit equilateral triangle, show there's always a pair of distance $\leq \frac{1}{2}$ apart



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

Given 5 points, two must share the same quadrant

Pigeonhole Principle!



Today's Keywords

- Divide and Conquer
- Closest Pair of Points

CLRS Readings

- Chapter 4

Homeworks

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

Recurrence Solving Techniques

Tree

Guess/Check

"Cookbook"

Substitution

Master Theorem

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

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

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

Case 3: if $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ 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 $T(n) \in \Theta(f(n))$

3 Cases

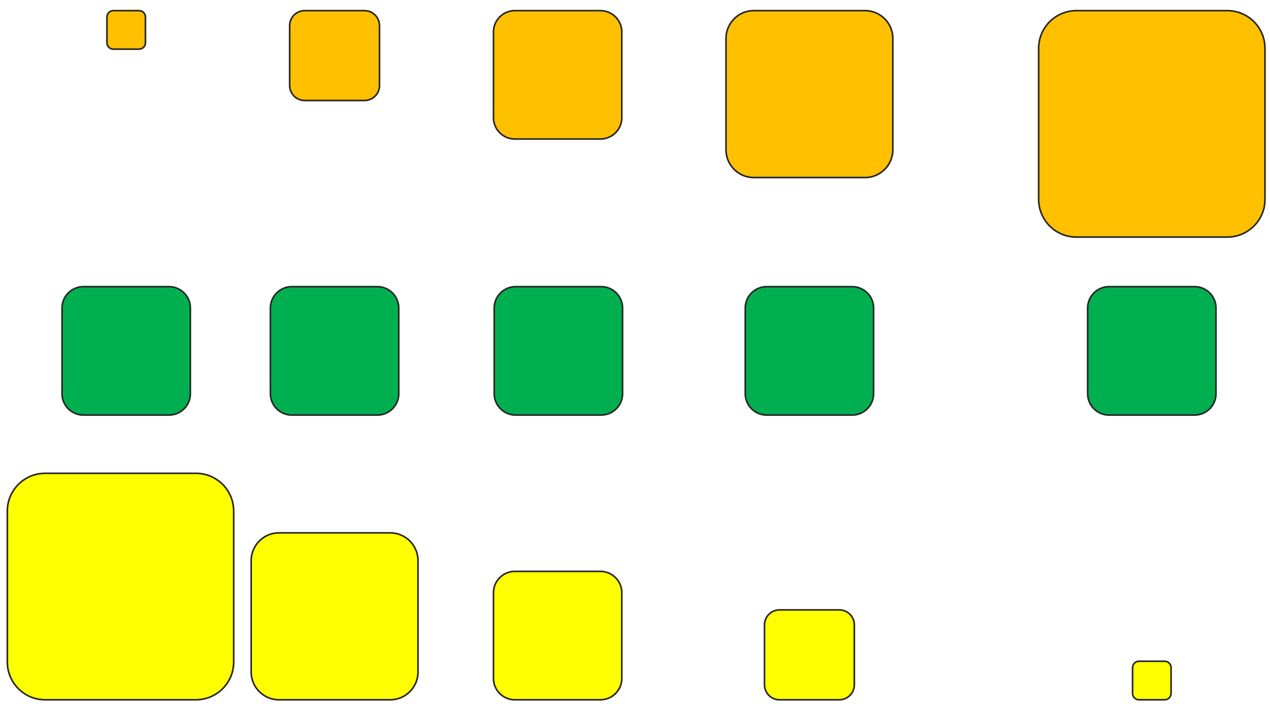
$L = \log_b n$

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

Case 1:
Most work happens at the leaves

Case 2:
Work happens consistently throughout

Case 3:
Most work happens at top of tree



Historical Aside: Master Theorem

Jon Bentley

Dorothea Haken

No Picture Found

James Saxe

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

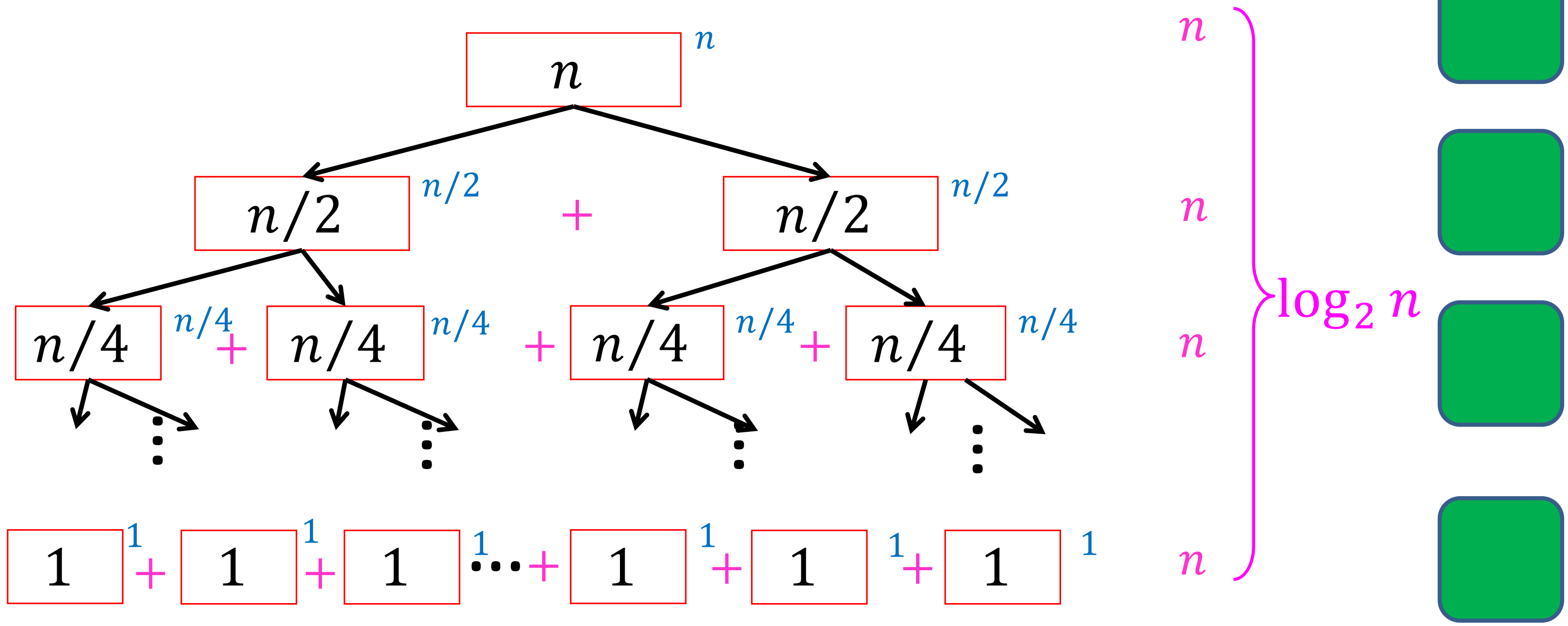
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

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

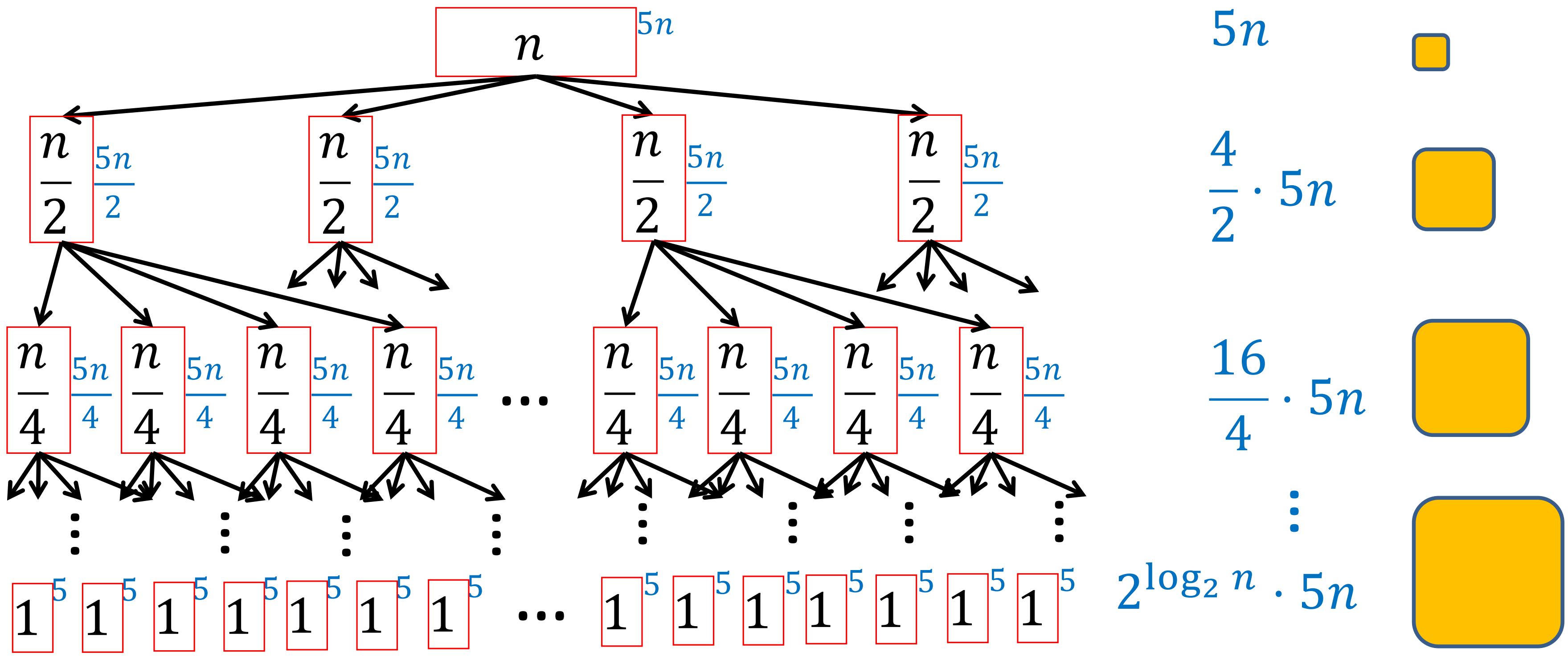
$$T(n) = 4T\left(\frac{n}{2}\right) + 5n$$

Case 1

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

Tree method

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



Tree method

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

Cost is <u>increasing</u> with the recursion depth
(due to large number of subproblems)

Most of the work happening in the leaves

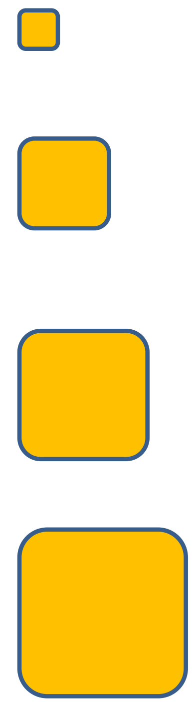
$5n$

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

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

$\vdots$

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



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

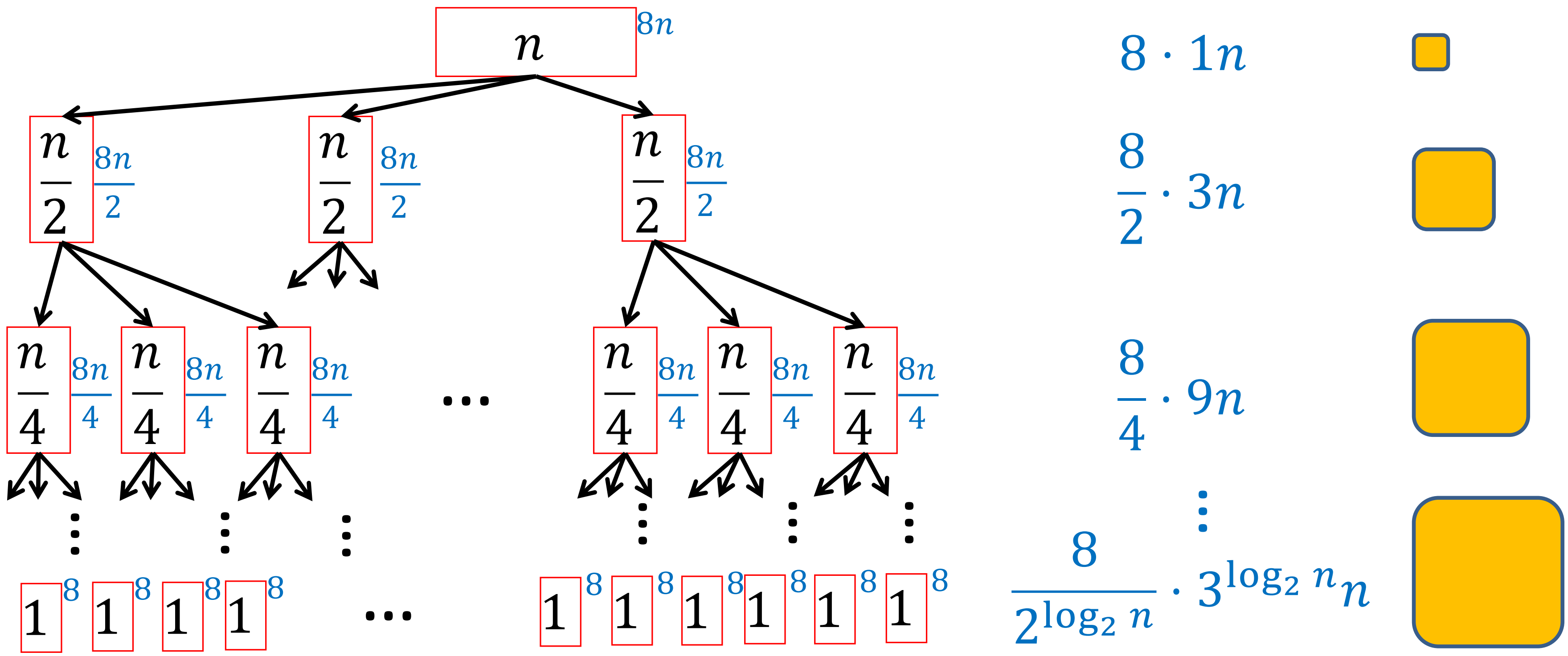
$$T(n) = 3T\left(\frac{n}{2}\right) + 8n$$

Case 1

$$\Theta\left(n^{\log_2 3}\right) \approx \Theta(n^{1.5})$$

Karatsuba

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



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

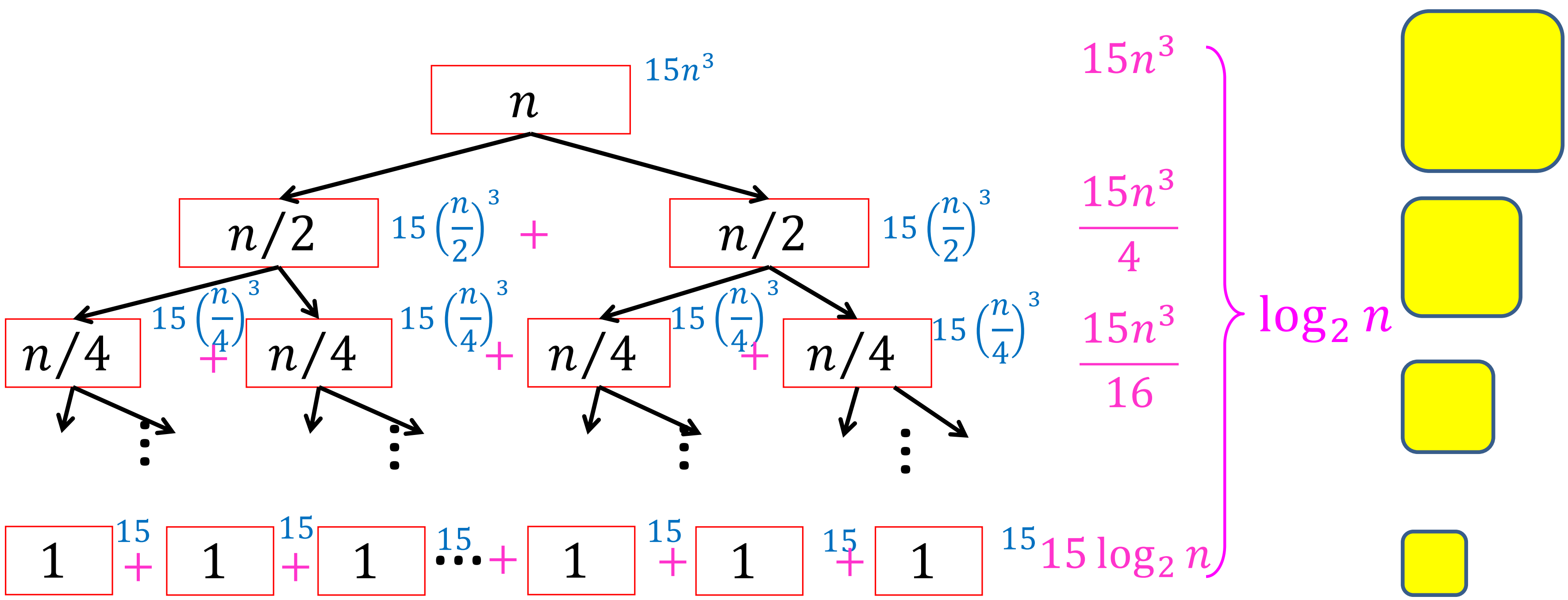
$$T(n) = 2T\left(\frac{n}{2}\right) + 15n^3$$

Case 3

$$\Theta(n^3)$$

Tree method

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



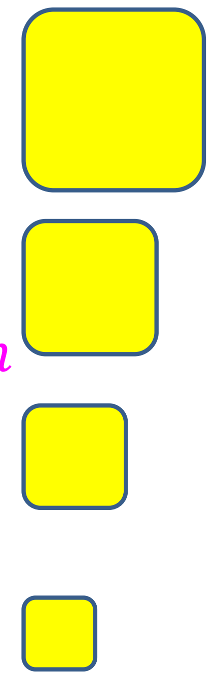
Tree method

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

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

Most of the work happening at the top

$$\left.\begin{array}{c}15n^3\\[1em] \dfrac{15n^3}{4}\\[1em] \dfrac{15n^3}{16}\\[3em] 15\log_2 n\end{array}\right\}\log_2 n$$



Recurrence Solving Techniques

Tree

Guess/Check

"Cookbook"

Substitution

Substitution Method

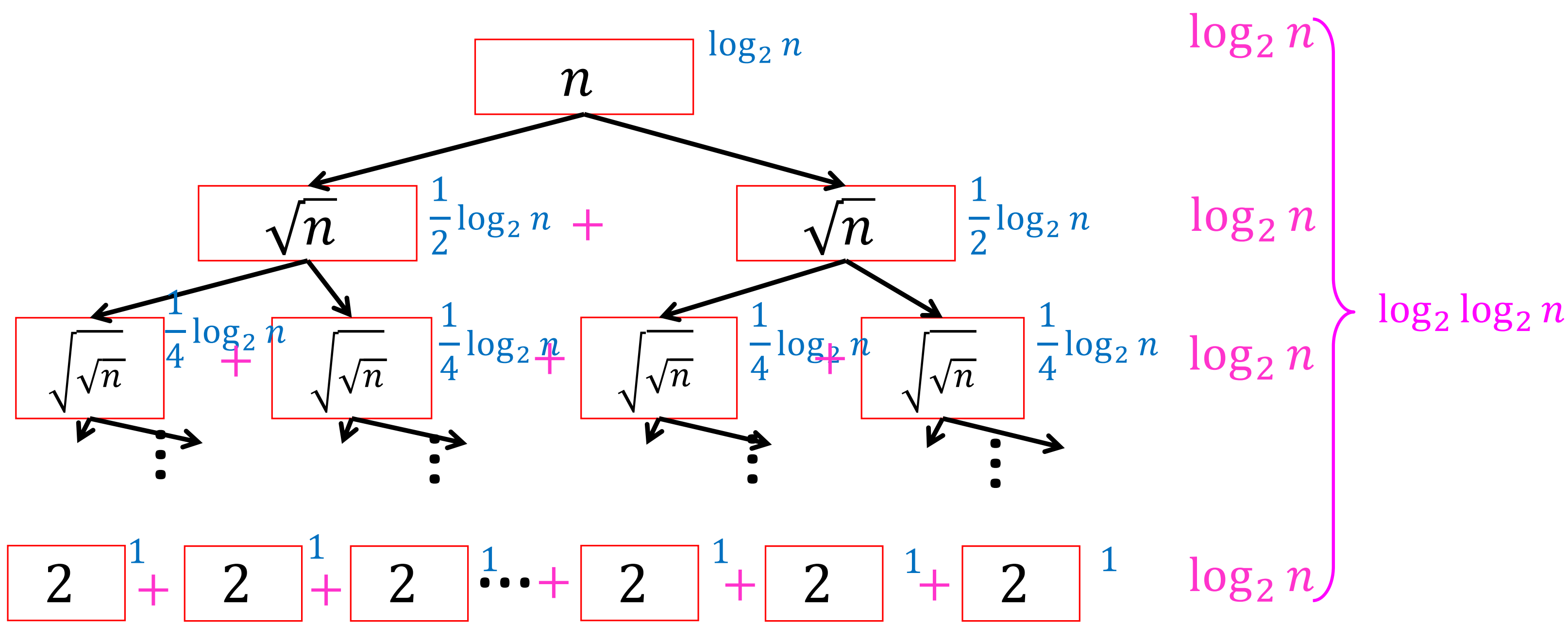
- Idea: take a "difficult" recurrence, re-express it such that one of our other methods applies.
- Example:

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

Tree method

$$\log_2 n^{1/2} = \frac{1}{2}\log_2 n$$

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



$$T(n) = O(\log_2 n \cdot \log_2 \log_2 n)$$

Substitution Method

$$T(n) = 2T(\sqrt{n}) + \log_2 n$$
$$= 2T\left(n^{1/2}\right) + \log_2 n$$

I don't like the ½ in the exponent

Let $n = 2^m$, i.e. $m = \log_2 n$

Now the variable is in the exponent on both sides!

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

Let $S(m) = 2S\left(\frac{m}{2}\right) + m$ Case 2!

Let $S(m) = \Theta(m \log m)$ Substitute Back

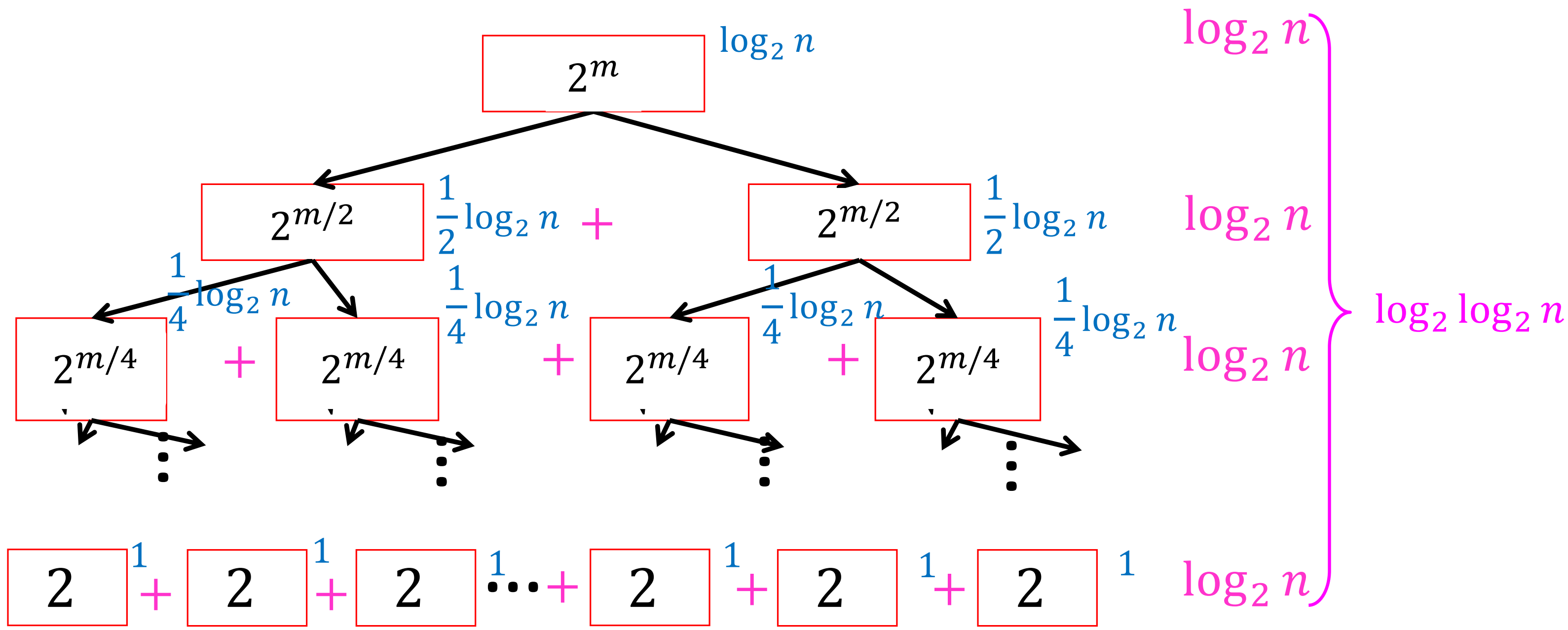
S will operate exactly as T, just redefined in terms of the exponent

$S(m) = T(2^m)$

Let $\mathrm{T}(n) = \Theta(\log n \log \log n)$

Tree method

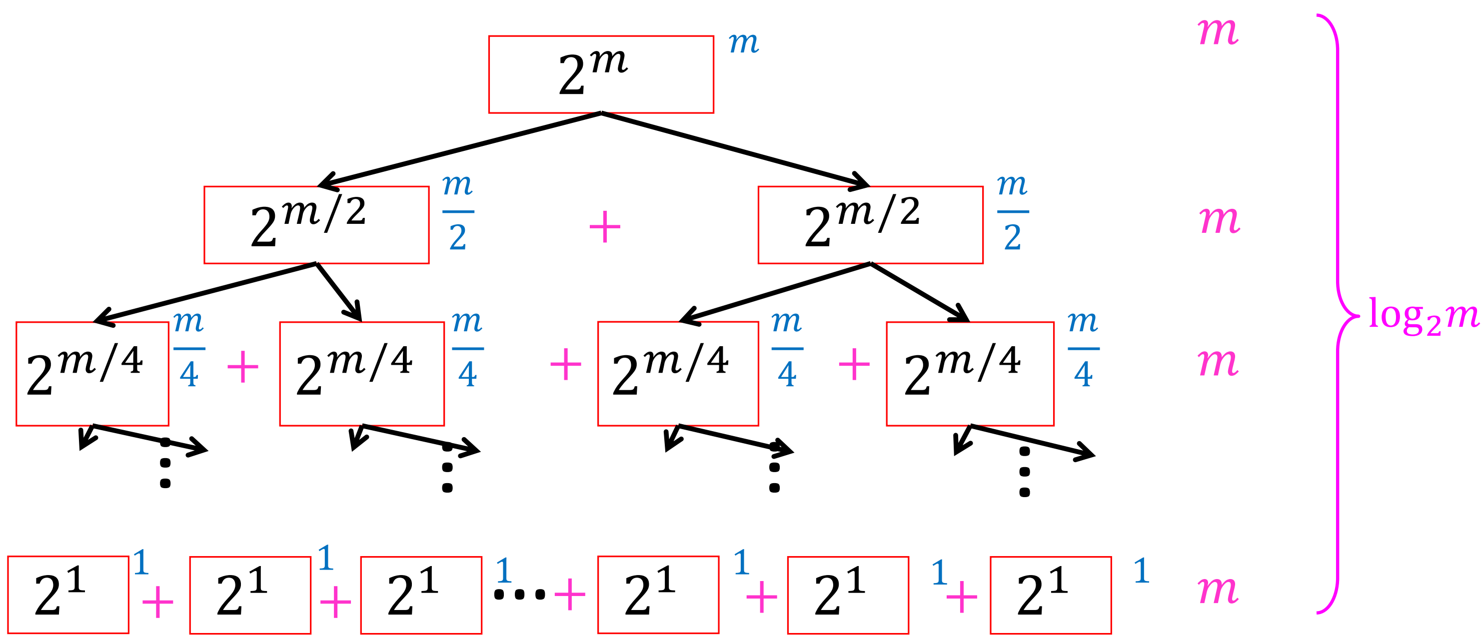
$$n = 2^m \qquad T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$$



Tree method

$n = 2^m$

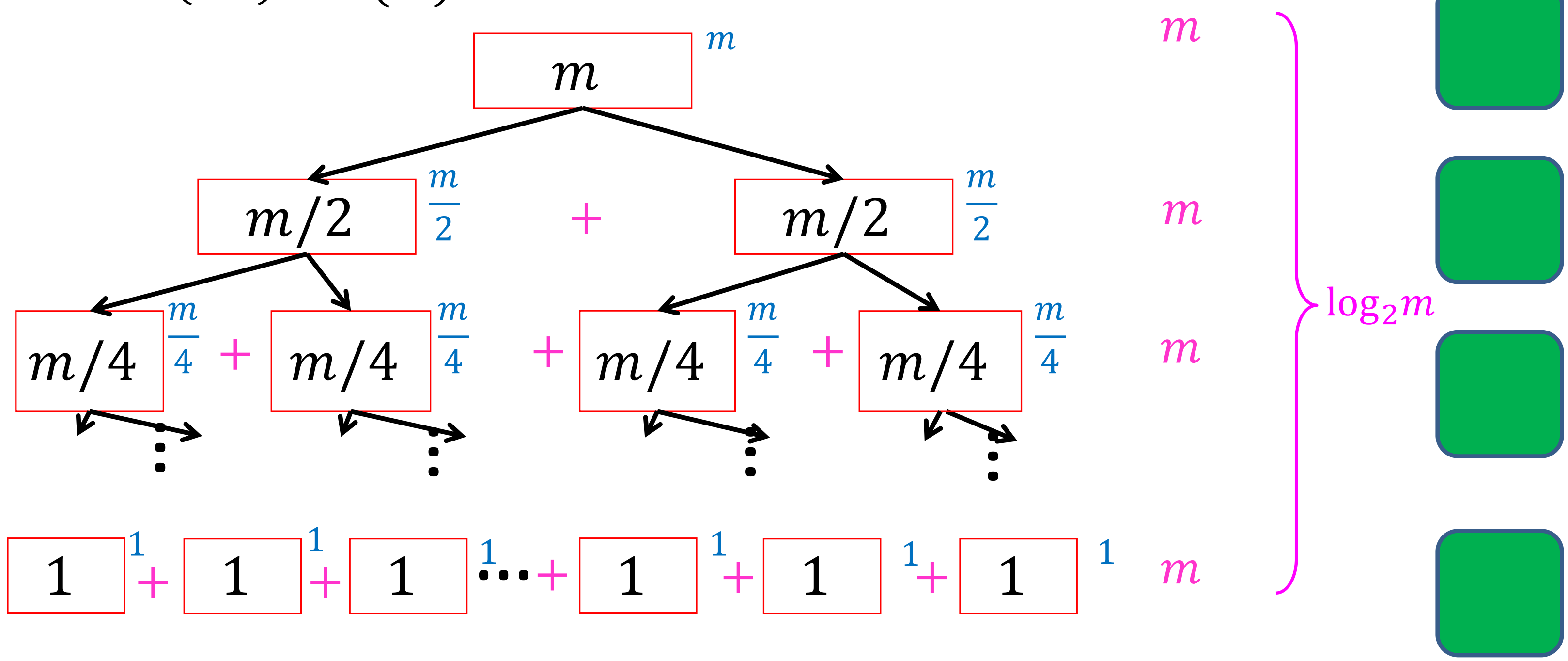
$$T(2^m) = 2T(2^{m/2}) + m$$



Tree method

$$n = 2^m$$
$$T(2^m) = S(m)$$

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

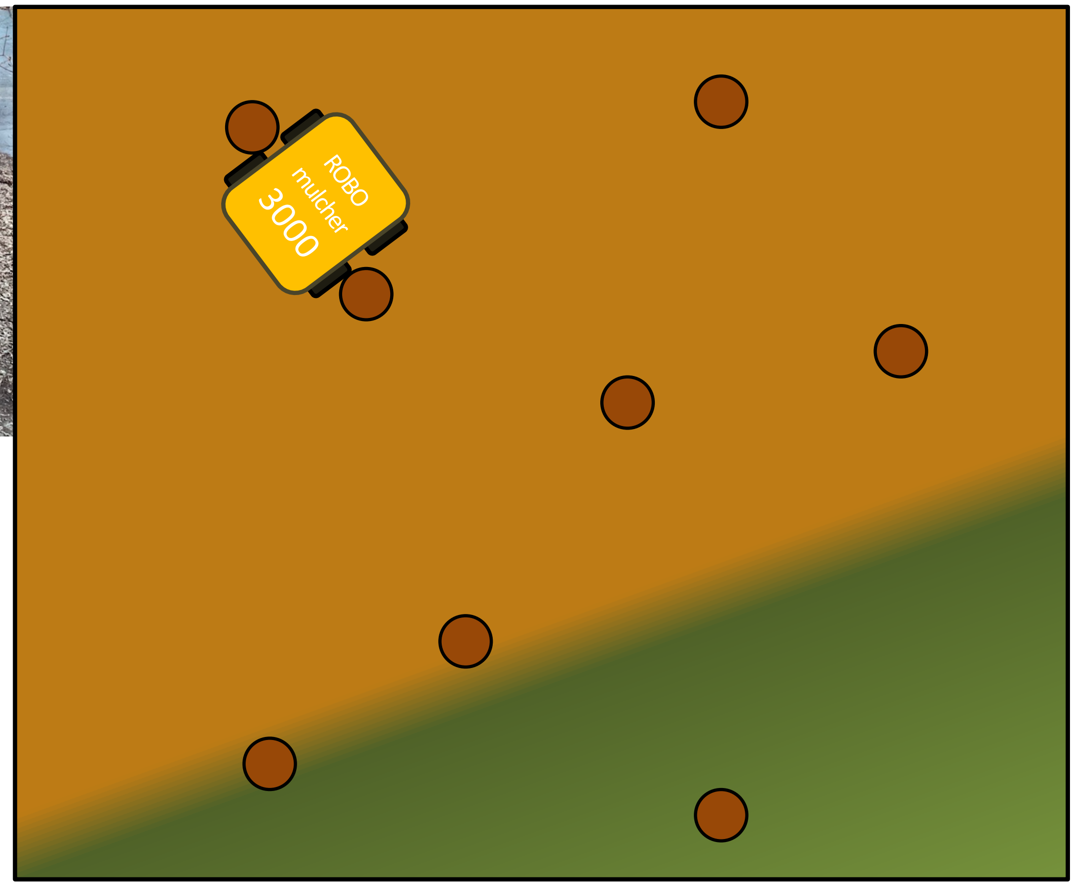


$$T(n) = O(m \cdot \log_2 m) = O(\log_2 n \cdot \log_2 \log_2 n)$$

Robbie's Yard

There has to be an easier way!

Constraints: Trees and Plants

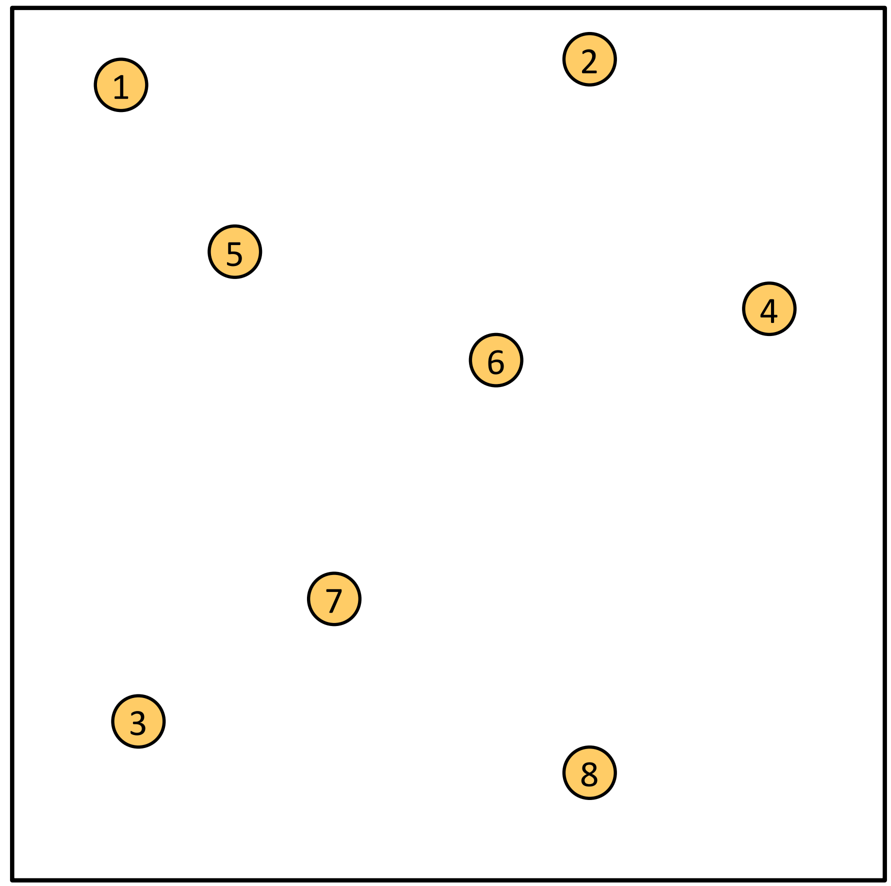


Need to find:
Closest Pair of Trees - how wide can the robot be?

Closest Pair of Points

Given:
A list of points

Return:
Pair of points with smallest distance apart



Closest Pair of Points: Naïve

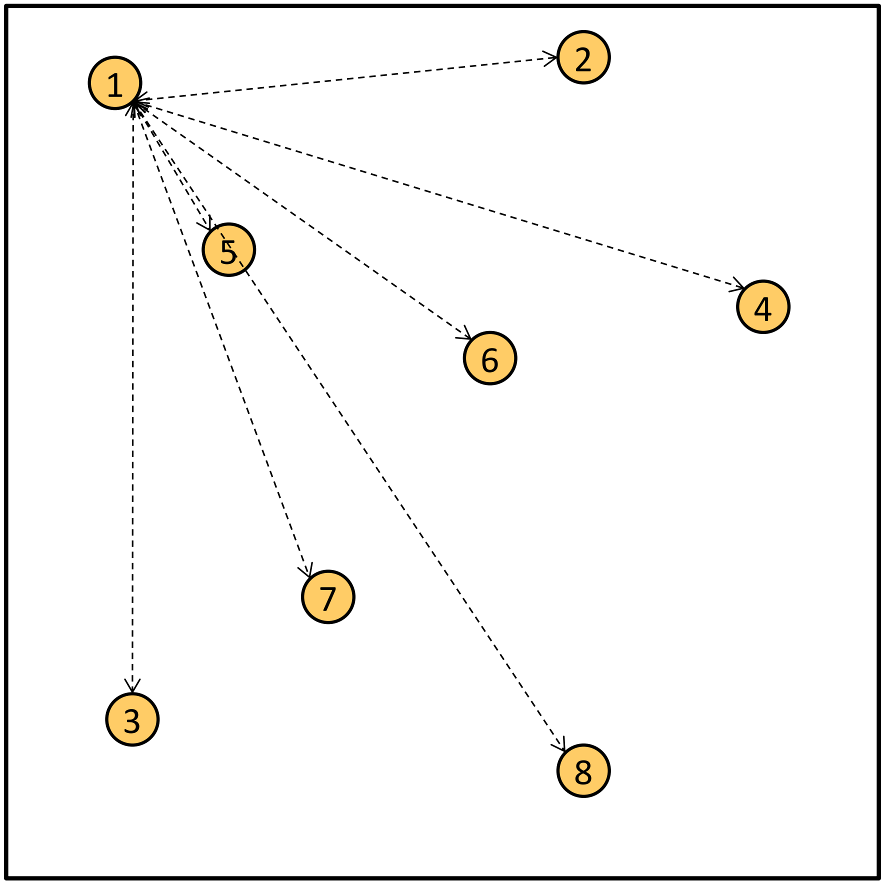
Given:
A list of points

Return:
Pair of points with smallest distance apart

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

We can do better!

$\Theta(n \log n)$

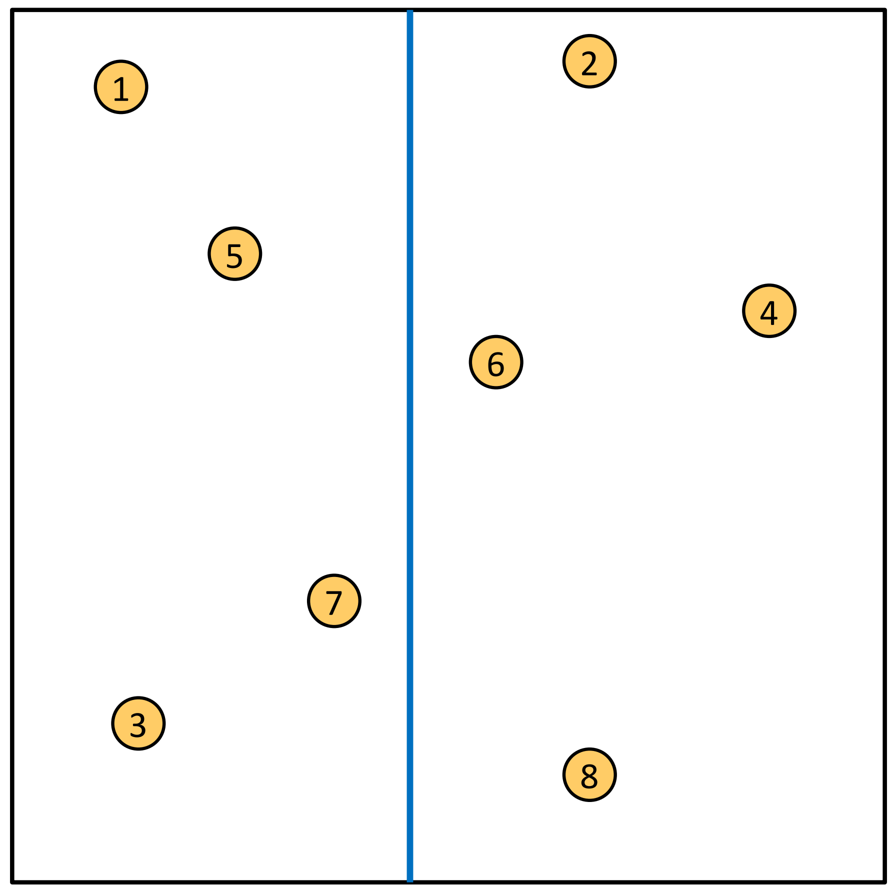


Closest Pair of Points: D&C

Divide: How?

At median x coordinate

Conquer:



Closest Pair of Points: D&C

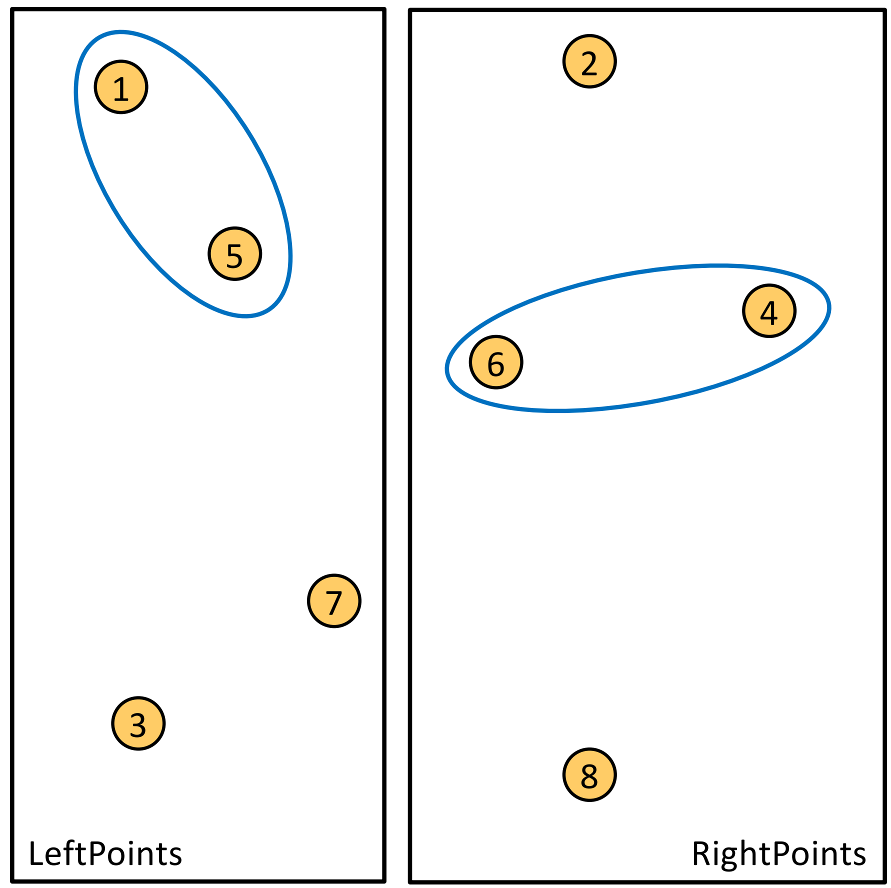
Divide:

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

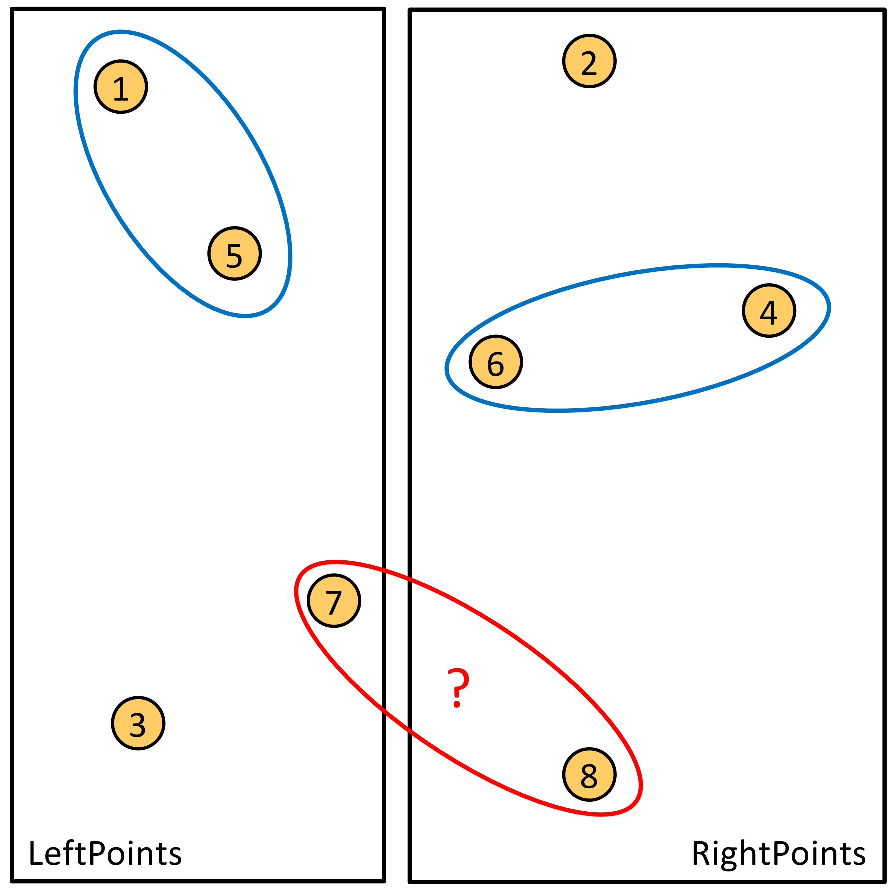
Conquer:

Recursively find closest pairs from Left and Right

Combine:

Return min of Left and Right pairs

Problem?



Closest Pair of Points: D&C

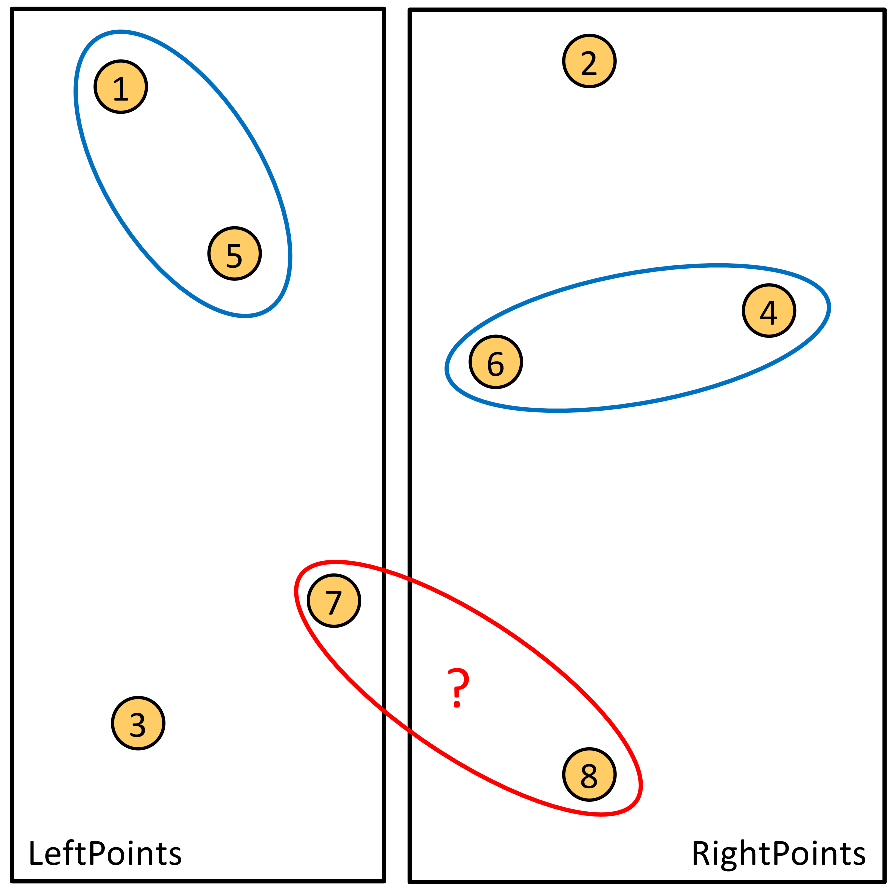
Combine:

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

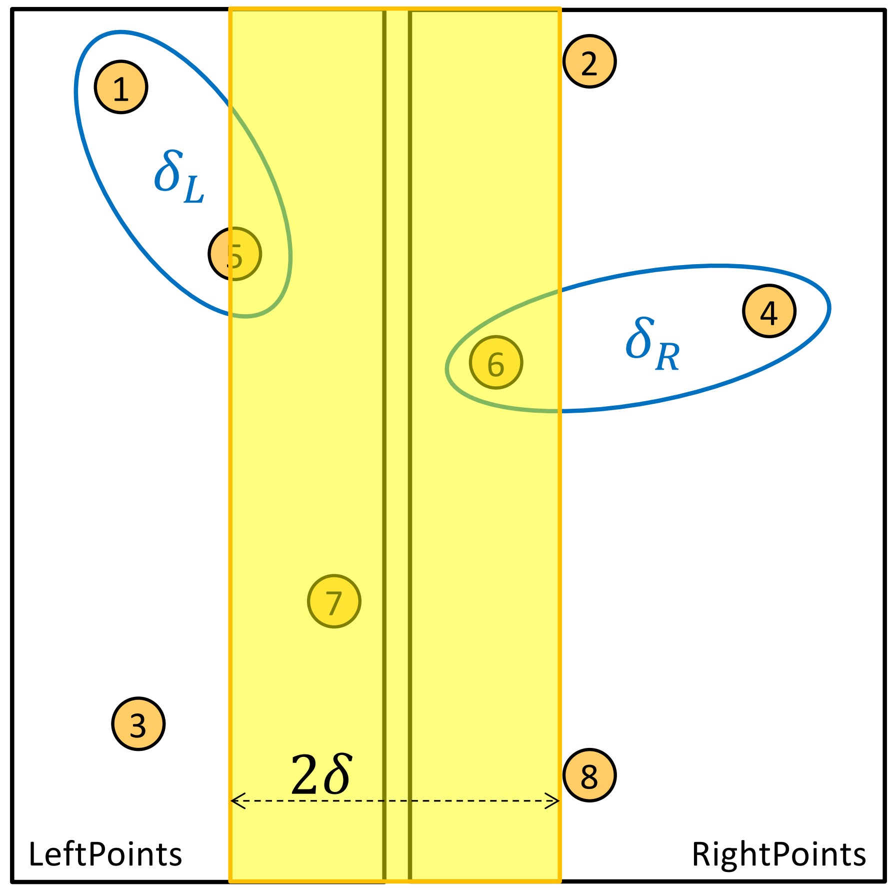
Combine:

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:

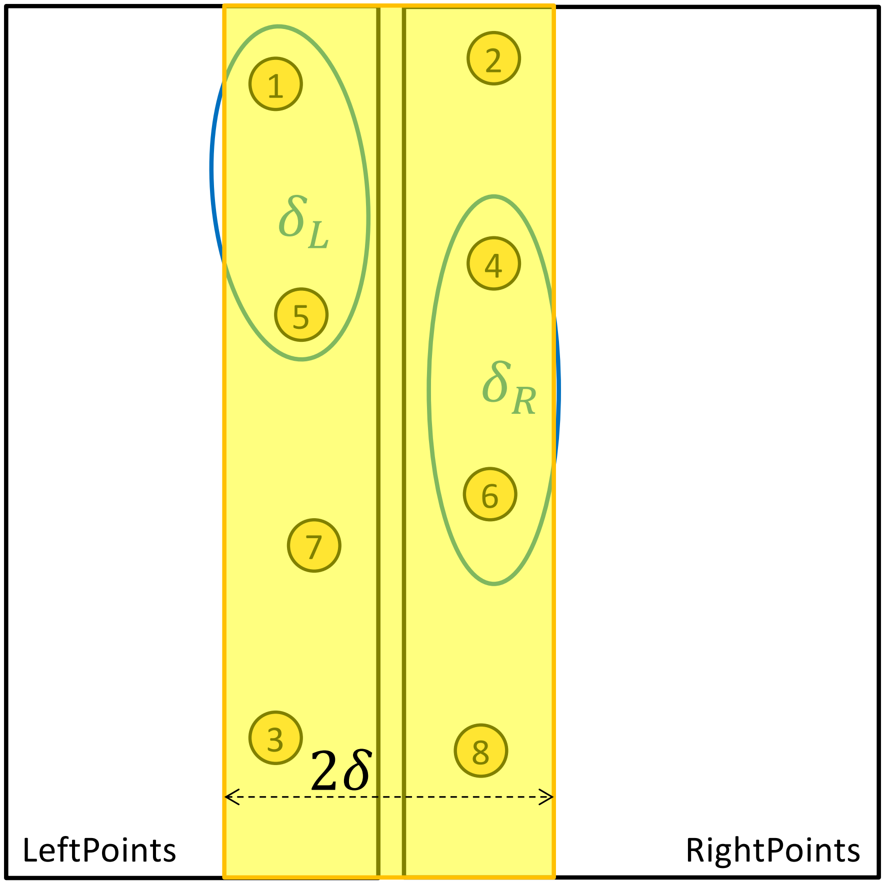
2. Closest Pair Spanned our "Cut"

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

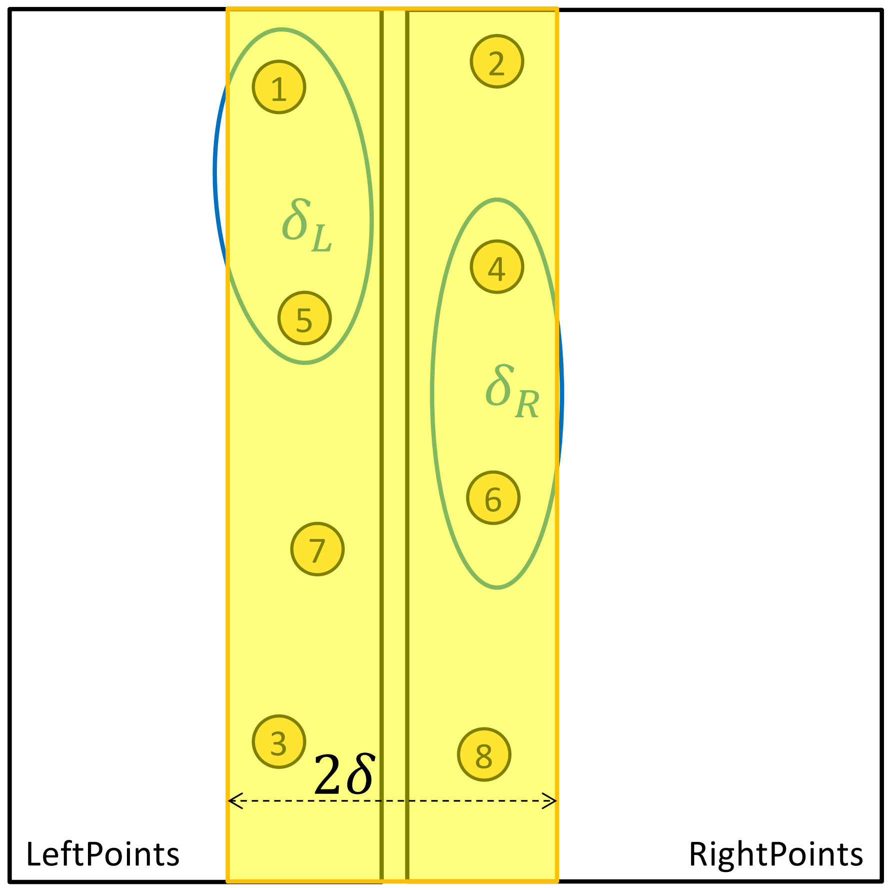
Combine:

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

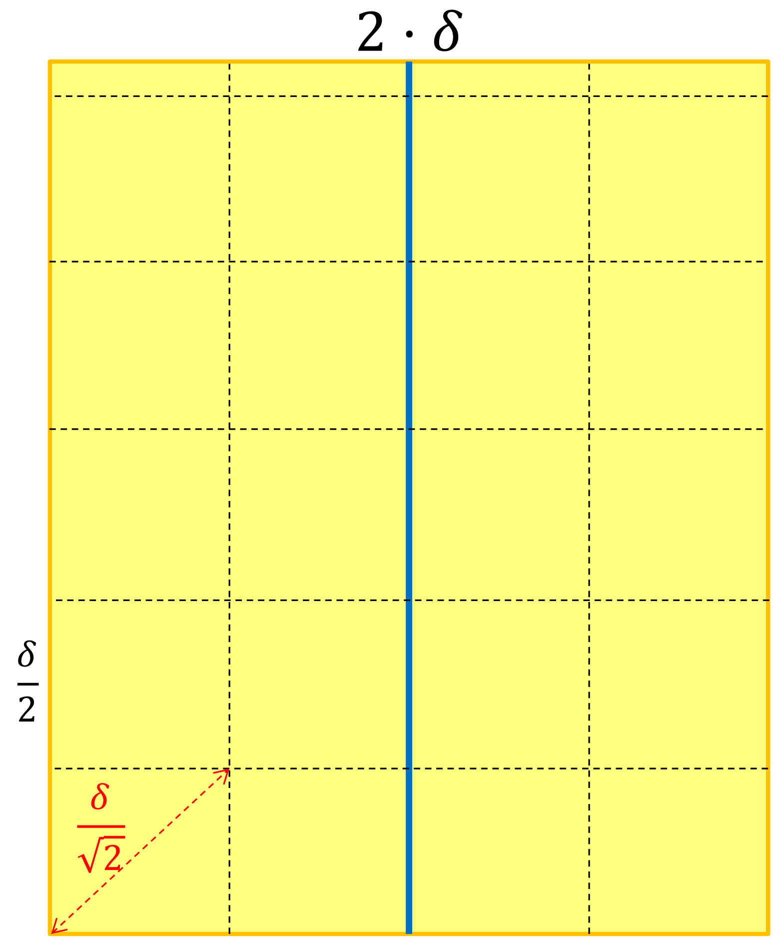
Combine:

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:

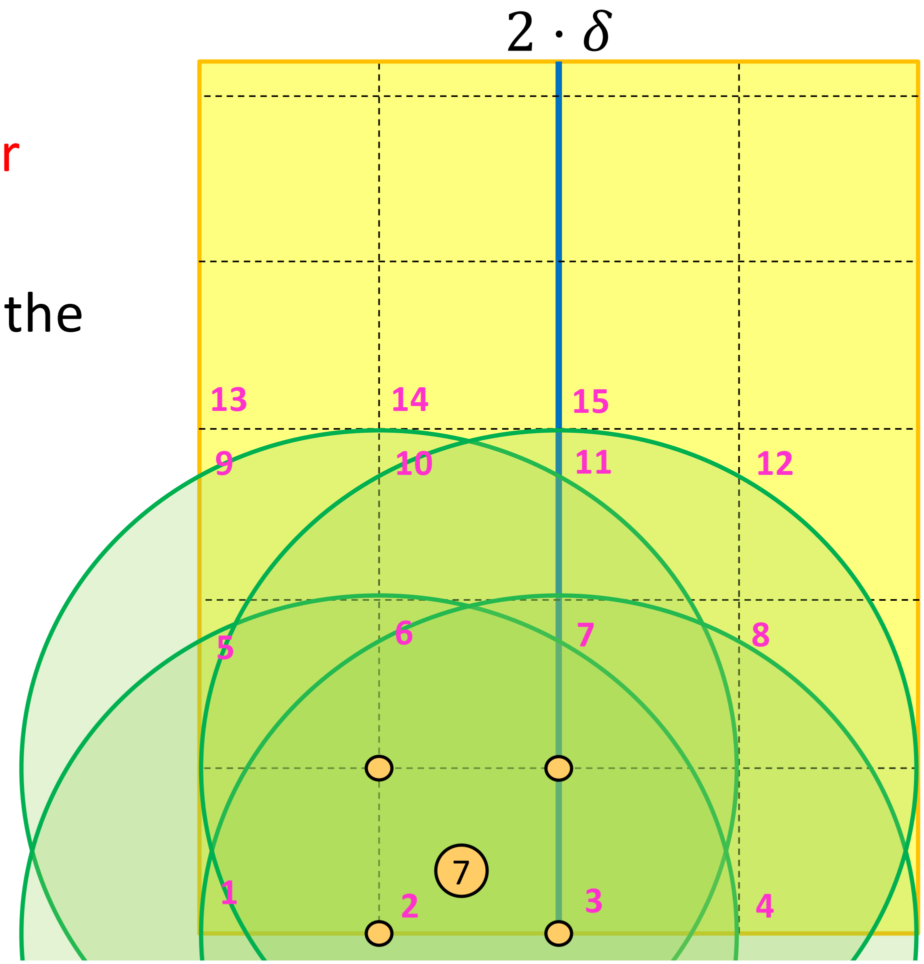
2. Closest Pair Spanned our "Cut"

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

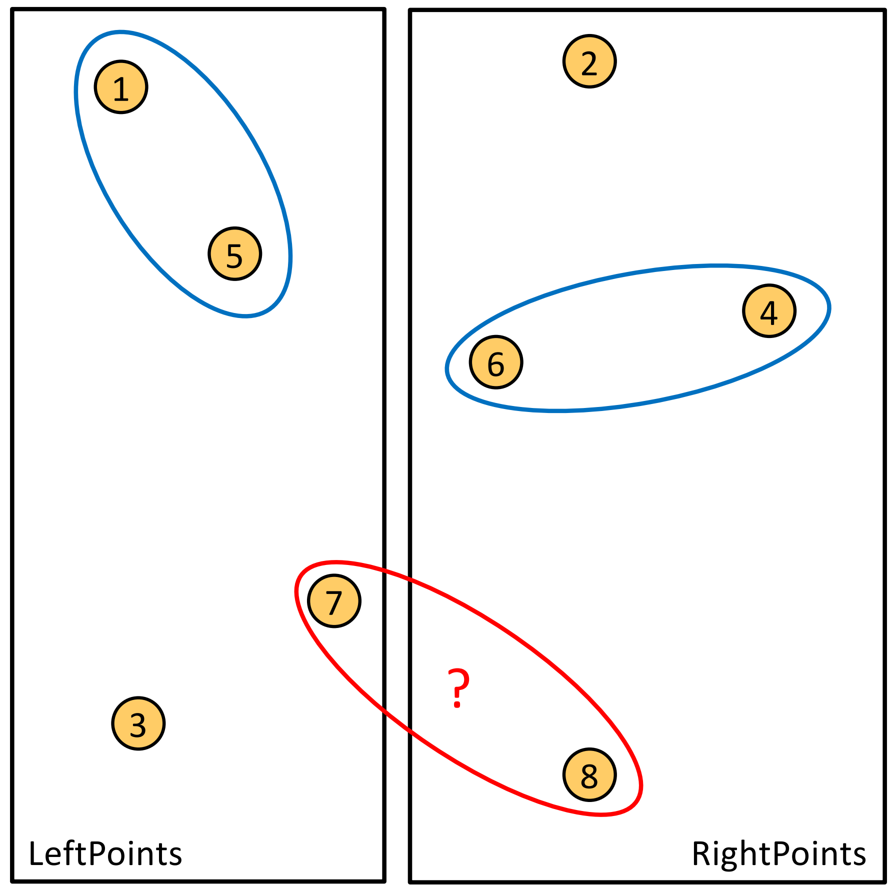
Divide: Partition points into two lists of points based on x-coordinate (split at the median x)

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



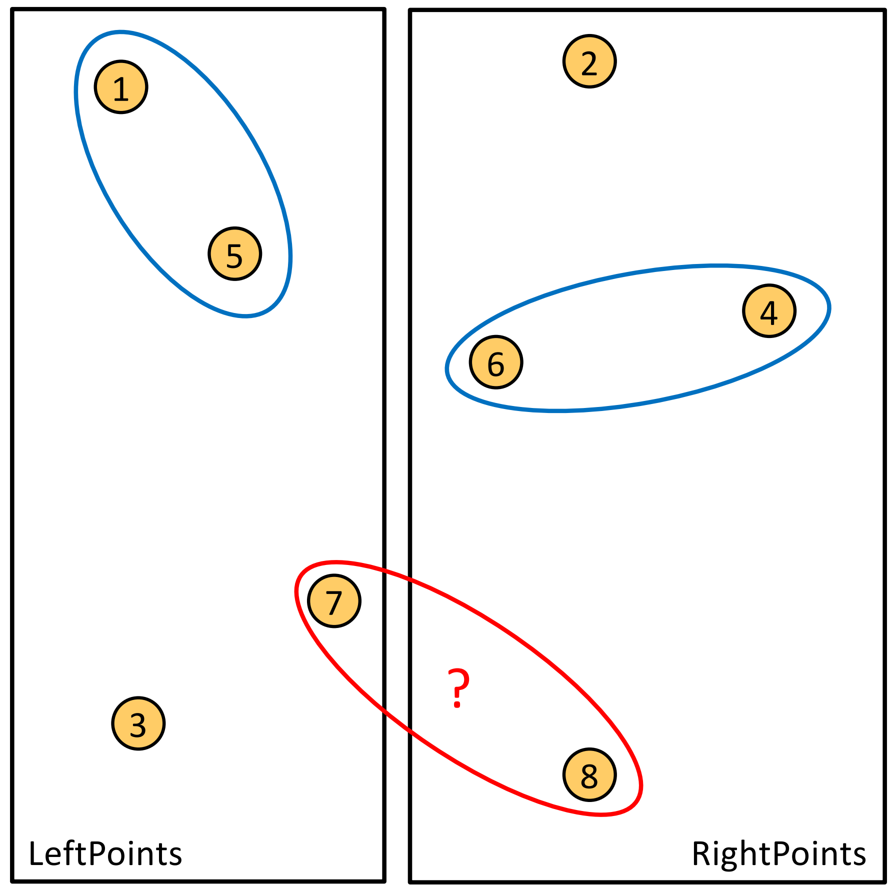
Closest Pair of Points: Divide and Conquer

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 in … way (x-coordinate within distan… 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:

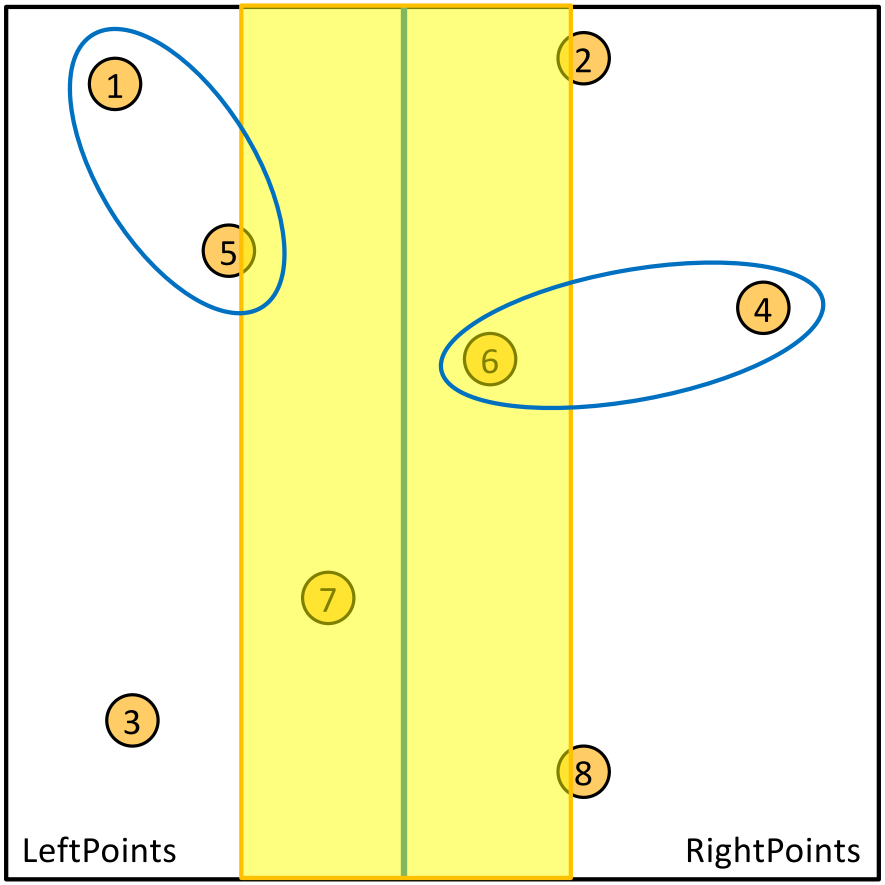
Closest Pair: $(4,6)$, $\delta_{4,6}$

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?

$\Theta(n \log n)$

$$T(n) = \begin{cases} \Theta(1) \\ 2T(n/2) \\ \Theta(n) \\ \Theta(n) \\ \Theta(1) \end{cases}$$

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

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

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

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

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

Run time? $O(n^3)$

Matrix Multiplication D&C

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 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

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

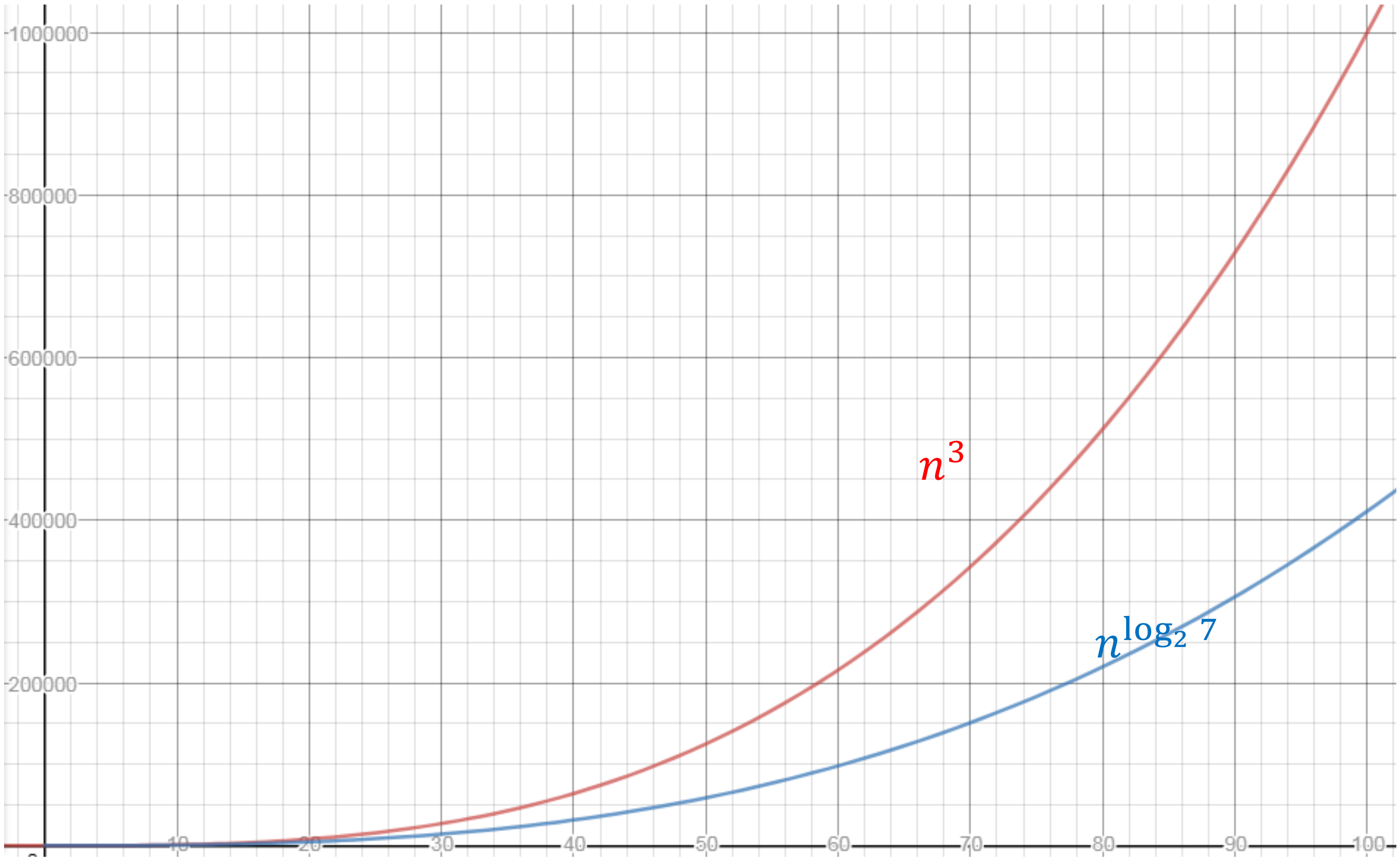
$$a = 7, b = 2, f(n) = \frac{9}{2}n^2$$

Case 1!

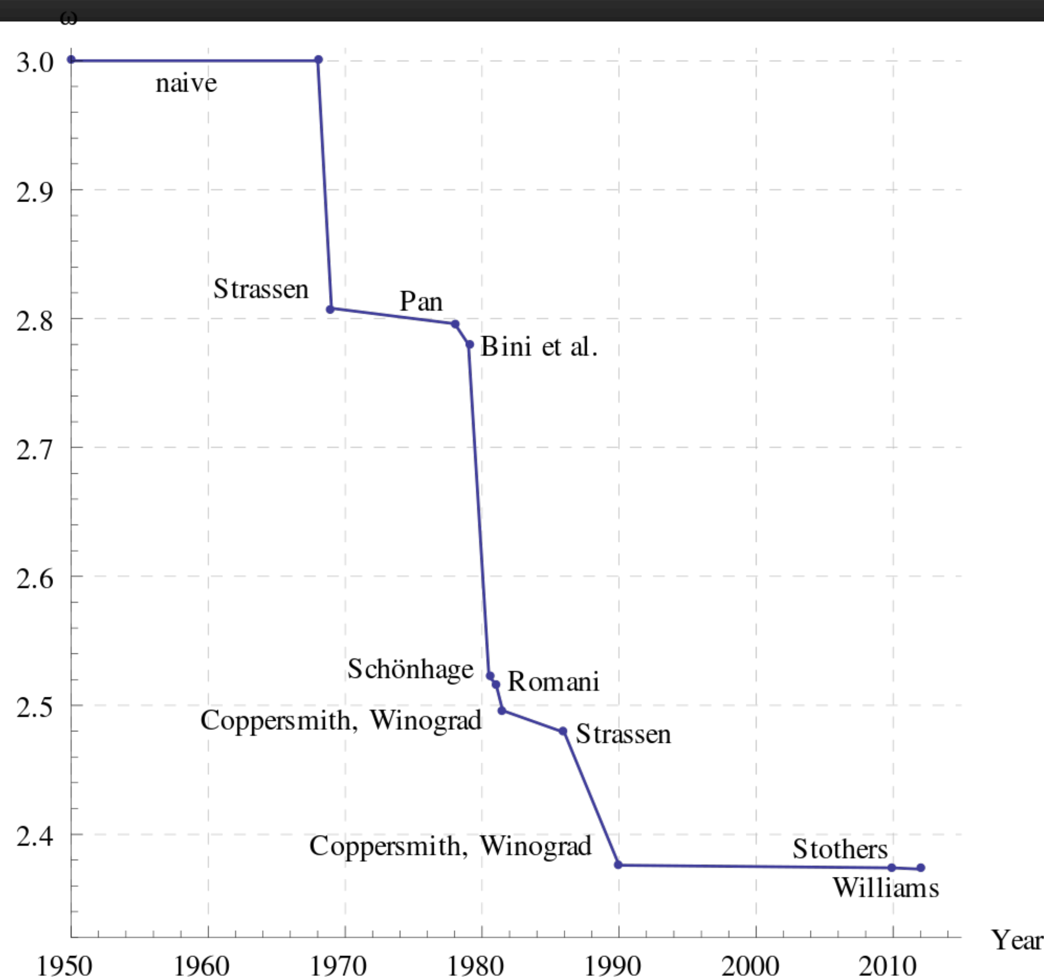
$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.807}$$

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

Strassen's Algorithm



Is this the fastest?



Best possible
is unknown

May not even
exist!