CSE 417 Autumn 2025

Lecture 8: Sorting as a Subroutine

Nathan Brunelle

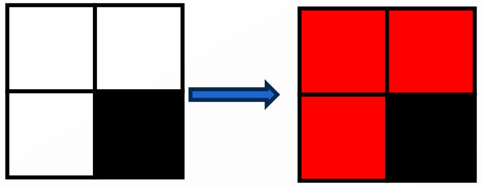
Homeworks

HW 1 feedback released yesterday

HW 2 out, due Friday (today) 11:59pm.

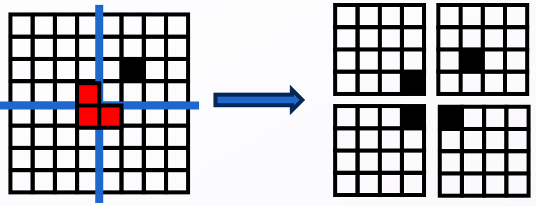
Divide and Conquer Review

Divide and Conquer (Trominoes)



Base Case:

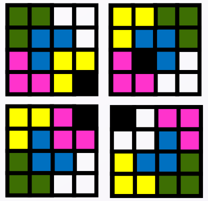
For a 2×2 board, the empty cells will be exactly a tromino



Divide:

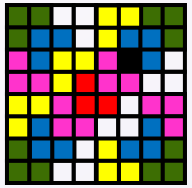
Break of the board into quadrants of size $2^{n-1} \times 2^{n-1}$ each

Put a tromino at the intersection such that all quadrants have one occupied cell



Conquer:

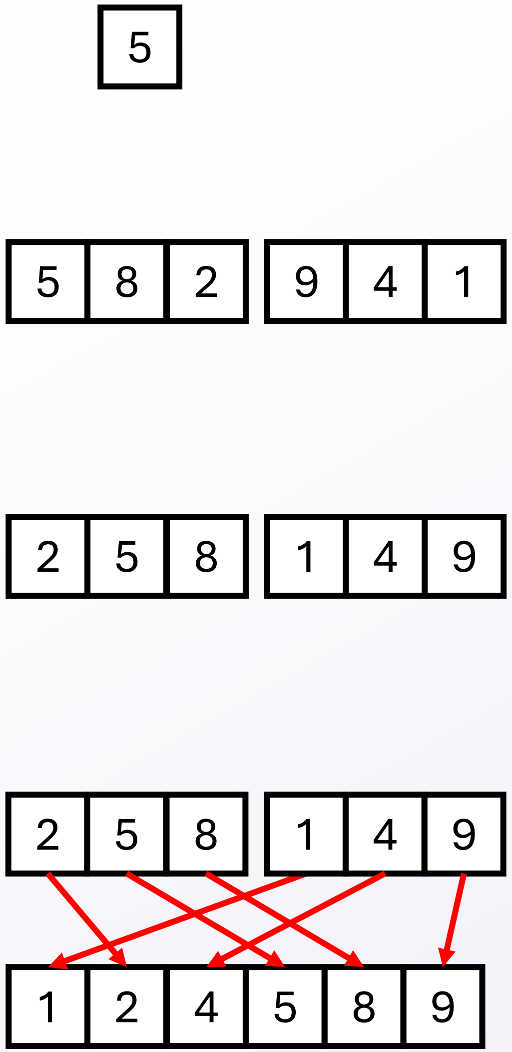
Cover each quadrant



Combine:

Reconnect quadrants

Divide and Conquer (Merge Sort)



Base Case:

If the list is of length 1 or 0, it's already sorted, so just return it

(Alternative: when length is ≤ 15, use insertion sort)

Divide:

Split the list into two "sublists" of (roughly) equal length

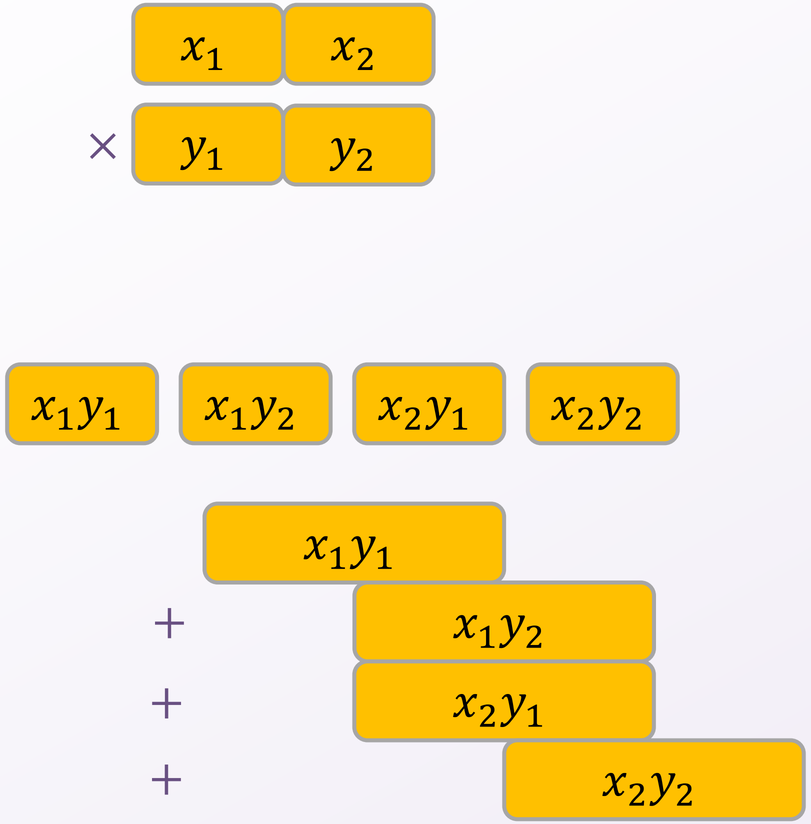
Conquer:

Sort both lists recursively

Combine:

Merge sorted sublists into one sorted list

Divide and Conquer (Integer Multiplication)



Base Case:

If there is only 1 place value, just multiply them

Divide:

Break the operands into 4 values:

- x_1 is the most significant $\frac{n}{2}$ digits of x
- x_2 is the least significant $\frac{n}{2}$ digits of x
- y_1 is the most significant $\frac{n}{2}$ digits of y
- y_2 is the most significant $\frac{n}{2}$ digits of y

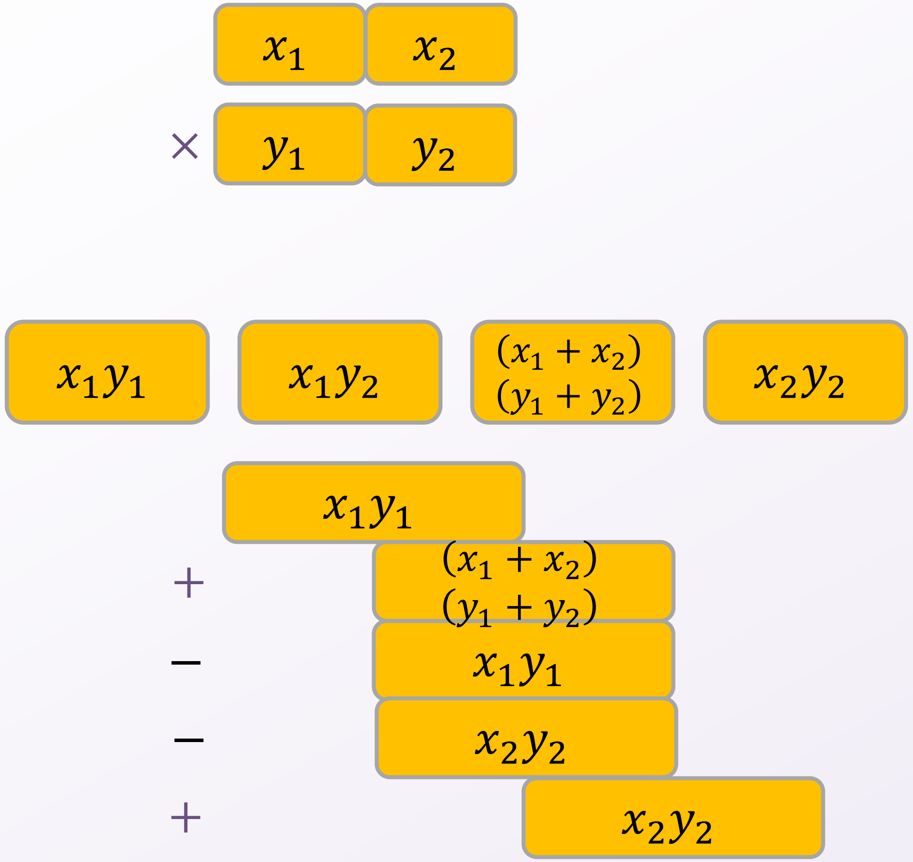
Conquer:

Compute each of x_1y_1, x_1y_2, x_2y_1, and x_2y_2

Combine:

Return $2^n(x_1y_1) + 2^{\frac{n}{2}}(x_1y_2 + x_2y_1) + (x_2y_2)$

Divide and Conquer (Karatsuba Method)



Base Case:

If there is only 1 place value, just multiply them

Divide:

Break the operands into 4 values:

- x_1 is the most significant $\frac{n}{2}$ digits of x
- x_2 is the least significant $\frac{n}{2}$ digits of x
- y_1 is the most significant $\frac{n}{2}$ digits of y
- y_2 is the most significant $\frac{n}{2}$ digits of y

Conquer:

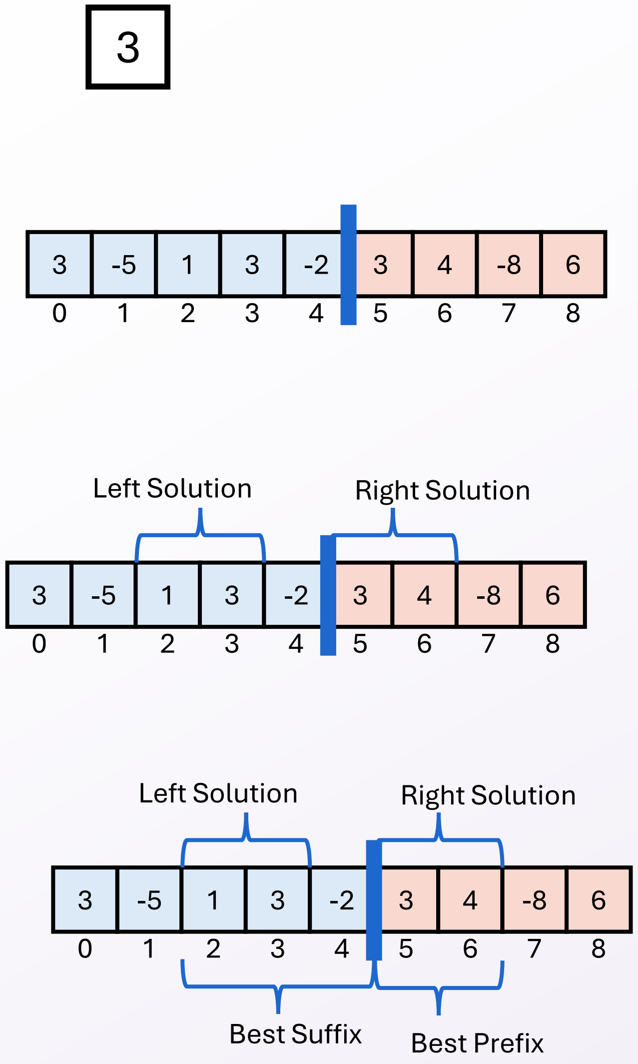
Compute each of x_1y_1, $(x_1 + x_2)(y_1 + y_2)$, and x_2y_2

Combine:

Return

$$2^n(x_1y_1) + 2^{\frac{n}{2}}\big((x_1 + x_2)(y_1 + y_2) - x_1y_1 - x_2y_2\big) + (x_2y_2)$$

Maximum Sum Subarray (D&C from reading)



Base Case:

If $i = j$ then return $i, i, arr[i]$ as the start, end, sum respectively

Divide:

Split the list into two "sublists" of (roughly) equal length. So the left is i to $\frac{i+j}{2}$ and the right is $\frac{i+j}{2} + 1$ to j

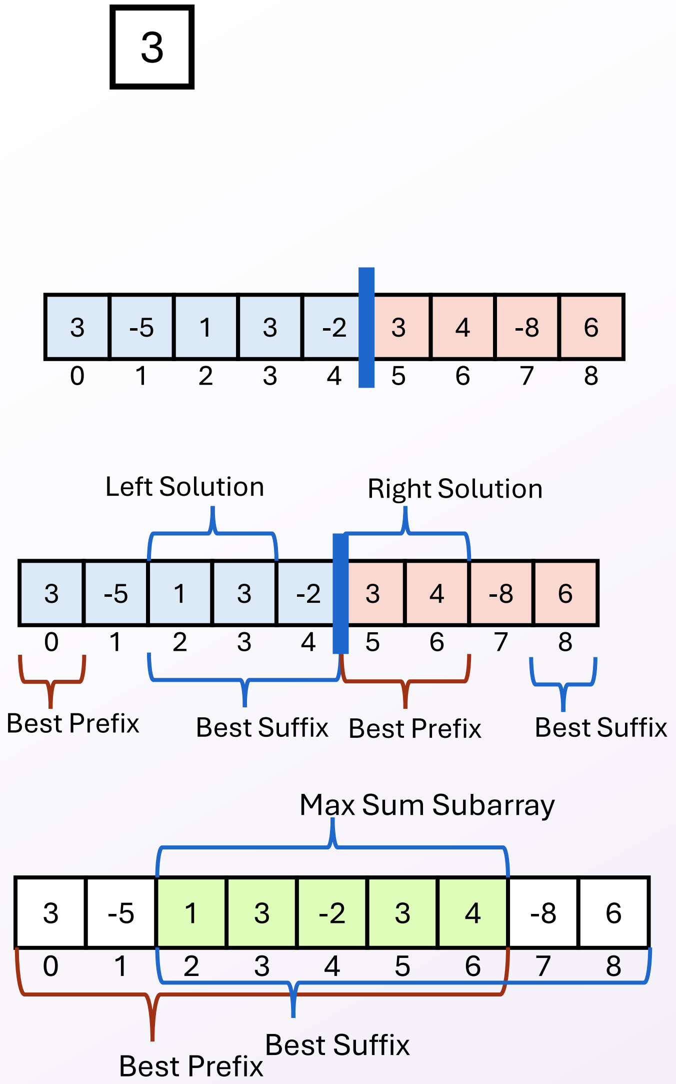
Conquer:

Find the start, end and sum of each subarray. Call these $leftStart, leftEnd, leftSum, rightStart, rightEnd, rightSum$

Combine:

Find the best suffix of the left subarray and best prefix of the right subarray. Return depending on which of $leftSum$, $rightSum$, and $middleSum$ is largest

Maximum Sum Subarray (Improved D&C)



Base Case:

If $i = j$ then: start=i, end =i, max sum=$arr[i]$, suffix start =i, suffix sum=$arr[i]$, prefix start =i, prefix sum=$arr[i]$, and total sum=$arr[i]$

Divide:

Split the list into two "sublists" of (roughly) equal length. So the left is i to $\frac{i+j}{2}$ and the right is $\frac{i+j}{2} + 1$ to j

Conquer:

Find all 8 return values for each half, we'll have a $left$ and $right$ version of each

Combine:

Use the 16 return values from the conquer step to identify the 8 return values for this step (details on the next slide)

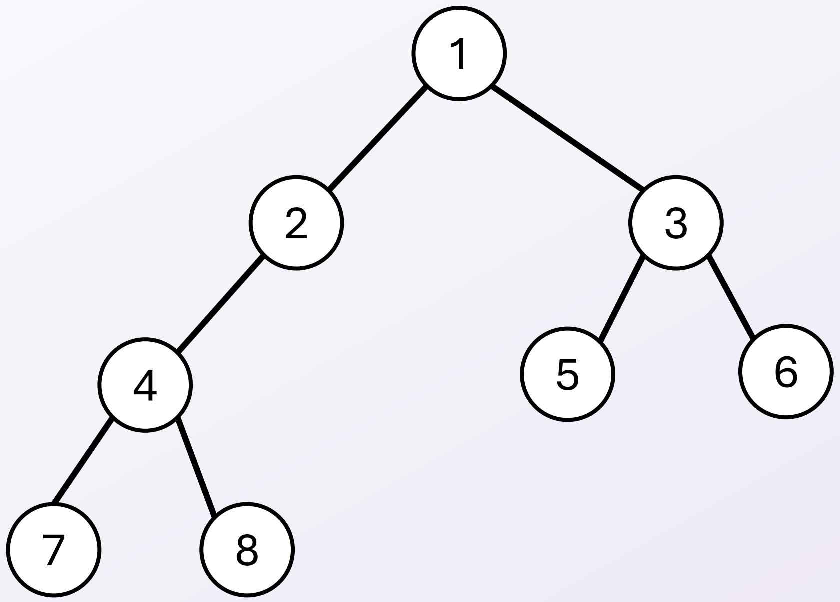
The “Technique of Computing More”

Sometimes, it’s helpful to perform more tasks in your combine and conquer algorithm. We’ll see 2 examples:

1) More tasks give better running time
2) More tasks enable correctness

Binary Tree Diameter

Binary Trees – Vocab Review



Nodes: Objects in the tree (labelled 1-8 here). They contain a value and may have a link to up to two other nodes

Child Node: a node linked to by some other node, that node is called its “parent”. E.g. 4 is the child of 2

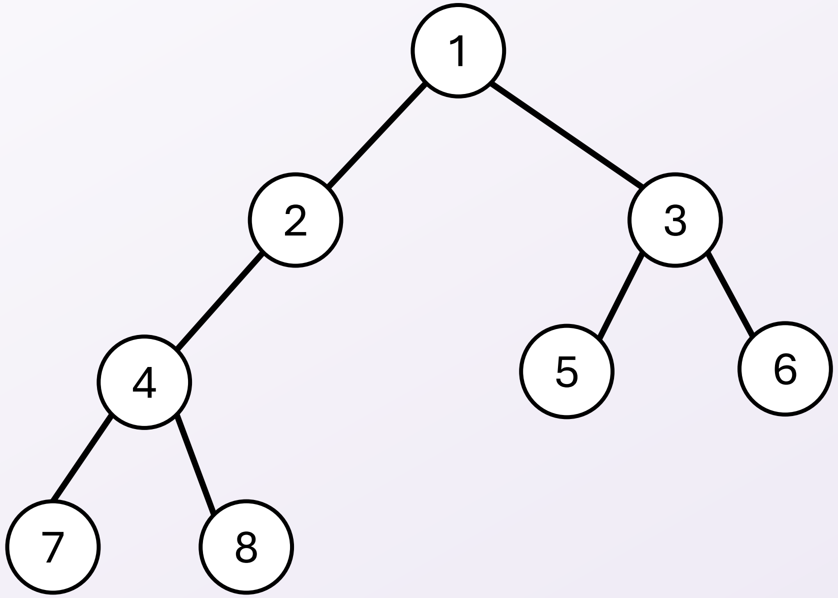
Sibling Nodes: two nodes that share a parent. E.g. 2 and 3 are siblings

Root Node: The unique node which has no parent. Node 1 is the root

Leaf Nodes: Nodes that have no children. 5,6,7, and 8 here

Binary Tree Height - Definition

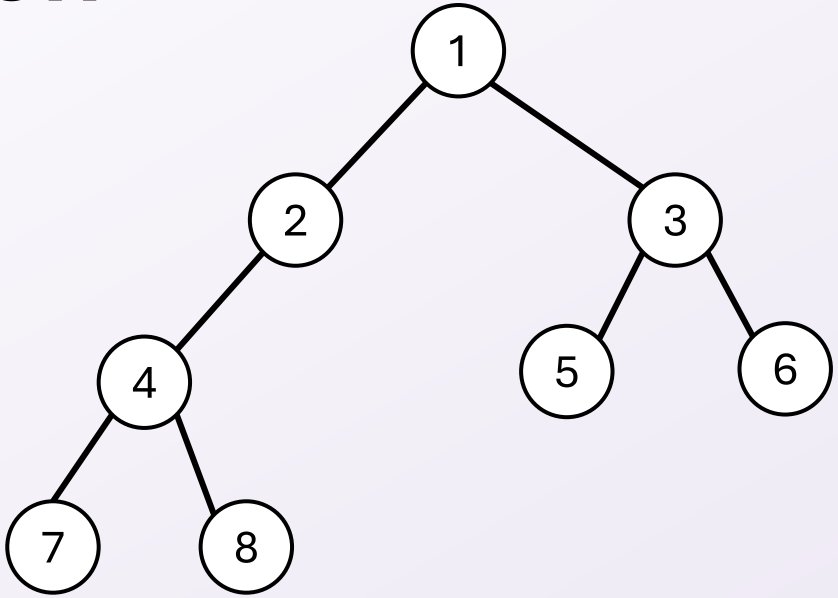
Distance: The distance between two nodes is the number of links you must follow to get from one to the other. E.g. the distance from 2 to 8 is 2, the distance from 2 to 6 is 3.



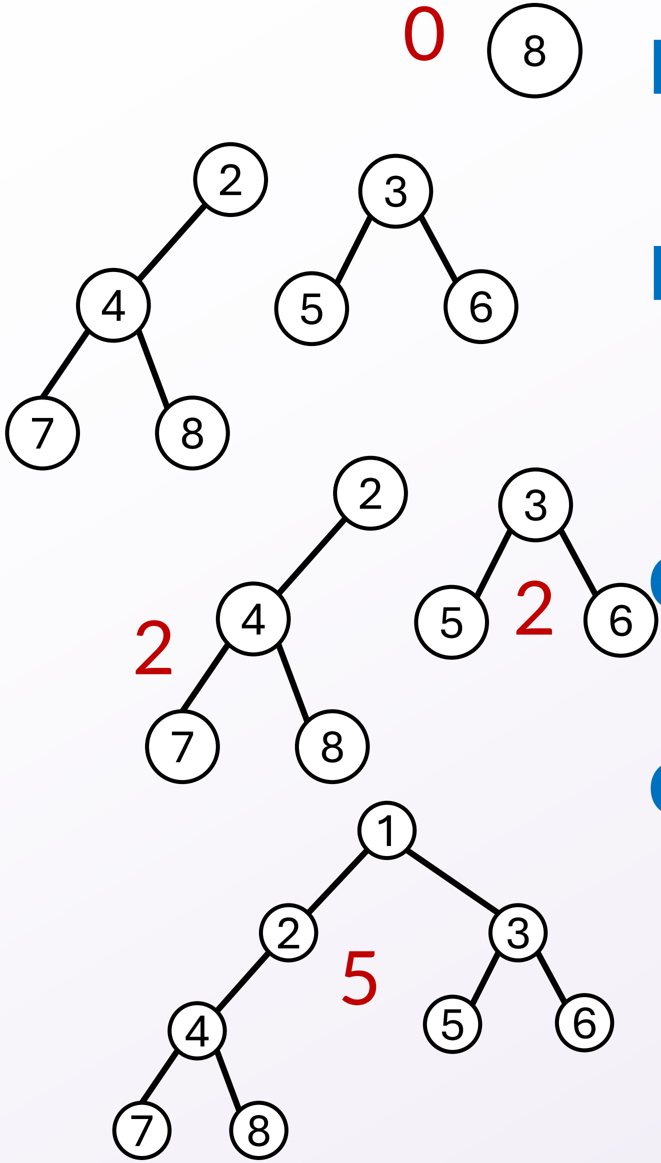
Height: The height of a binary tree is the largest distance from the root to some leaf. The height of this tree is 3 (1 is 3 away from 7)

Binary Tree Diameter - Definition

Diameter: The maximum distance between two nodes in a binary tree. The diameter of this tree is 5, because 7 is distance 5 from node 6.



Binary Tree Diameter – Incorrect Algorithm



Base Case:

If the current node is a leaf, the diameter is 0

Divide:

Split the tree into the left subtree and the right subtree

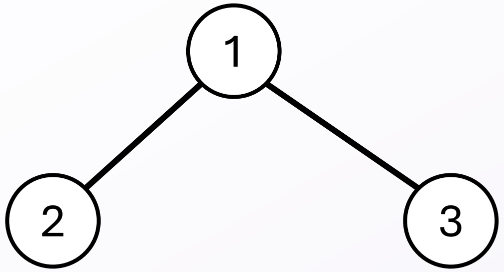
Conquer:

Find the diameter of each subtree

Combine:

Return the diameter of the left subtree + the diameter of the right subtree + 1

Incorrect Algorithm - Counterexample



Diameter of the left subtree: 0

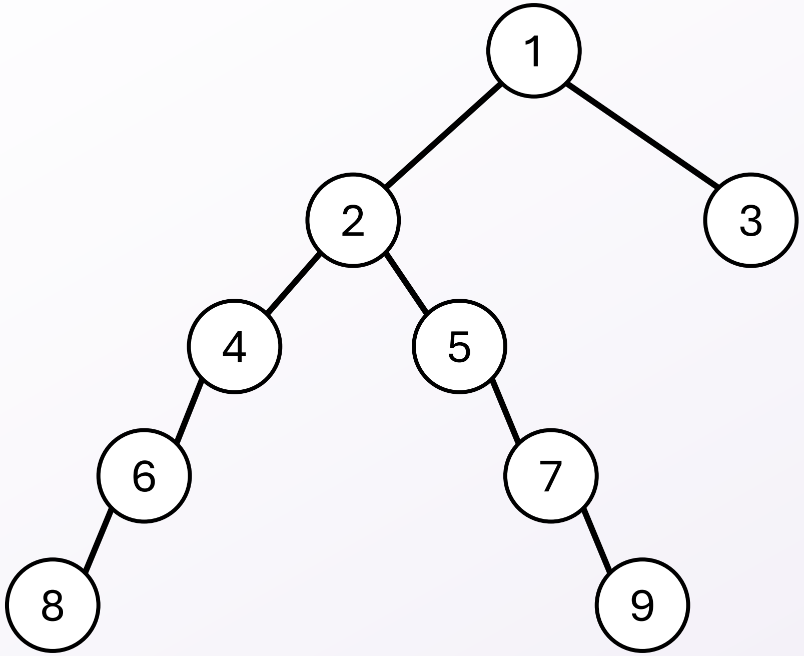
Diameter of the right subtree: 0

Diameter of the whole tree: 2

Diameter ended up being:

the distance to a left leaf + distance to a right leaf

Incorrect Algorithm - Counterexample



Diameter of the left subtree: 6

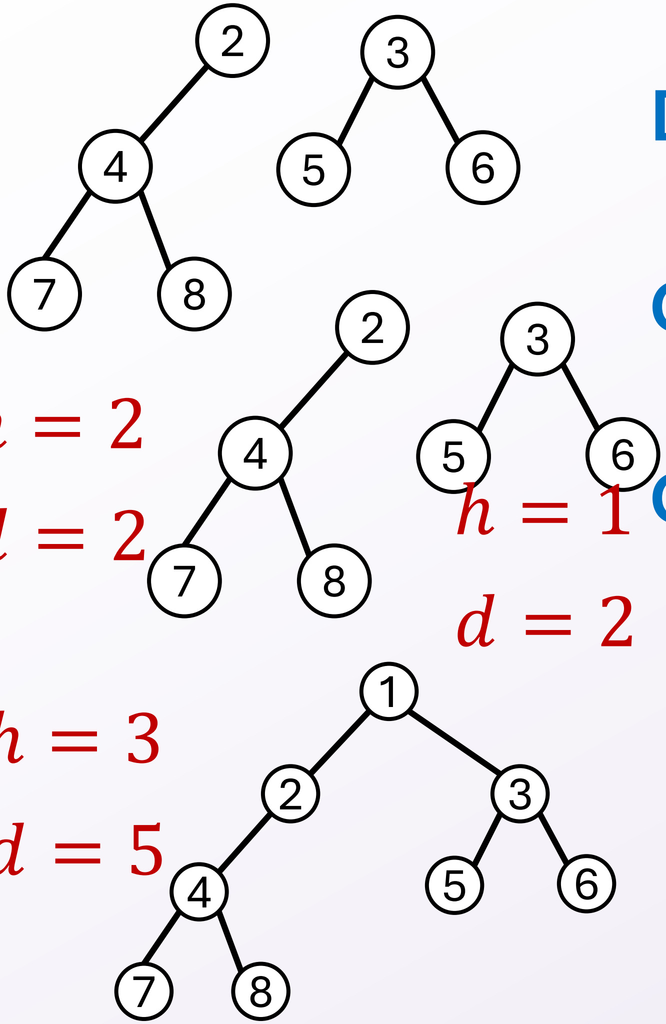
Diameter of the right subtree: 0

Diameter of the whole tree: 6

Diameter ended up being:

The diameter of a subtree

Binary Tree Diameter – Correct Algorithm



$h = 2$
$d = 2$

$h = 1$
$d = 2$

$h = 3$
$d = 5$

Base Case:

If the node is null the diameter and height are -1.

Divide:

Split the tree into the left subtree and the right subtree

Conquer:

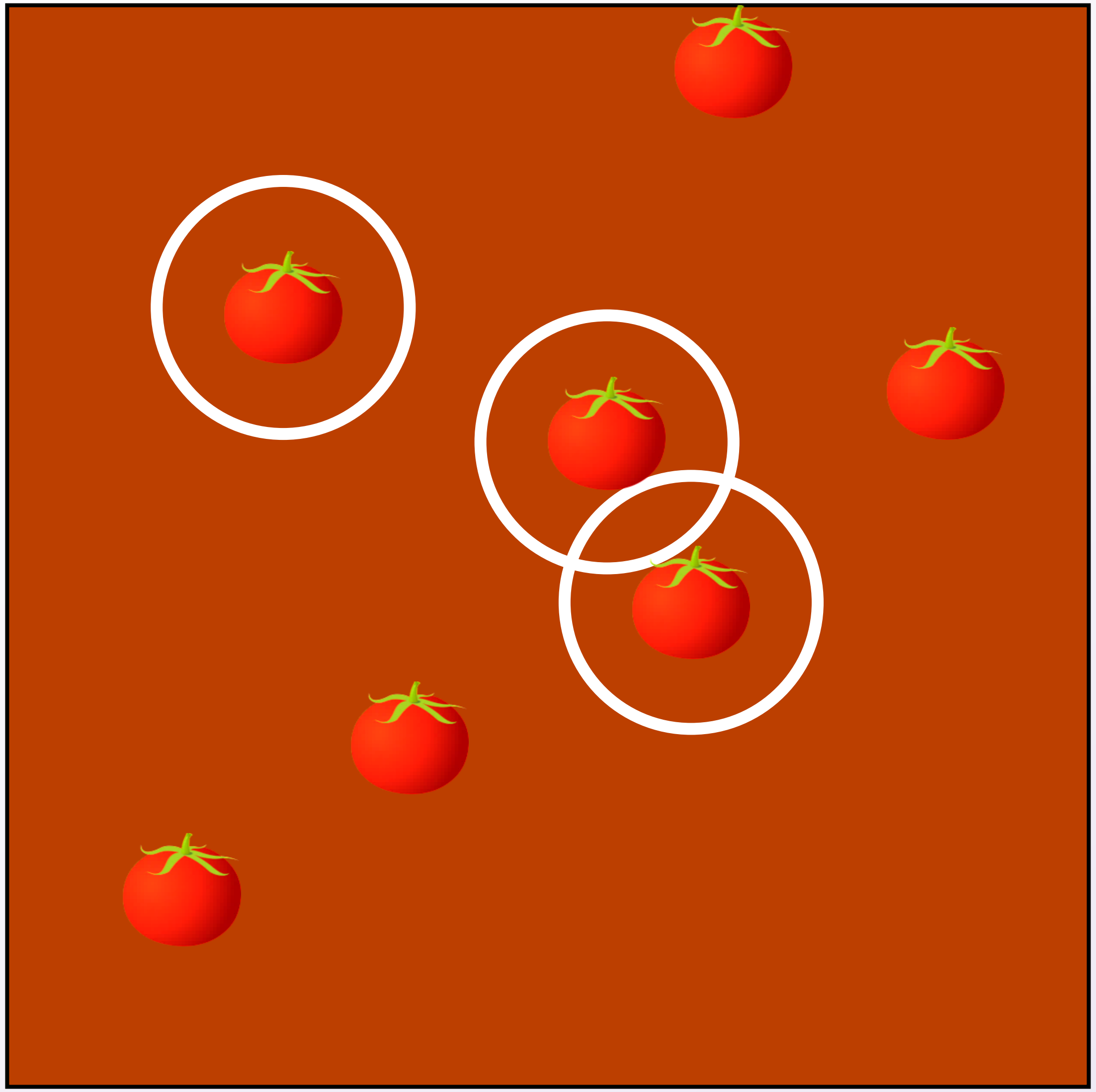
Find the diameter and height of each subtree

Combine:

Height = 1 + max(left height, right height)

Diameter = max(left diameter,
right diameter,
left height + right height +2)

Closest Pair of Tomatoes



Closest Pair of Points

Given:

- A sequence of $\boldsymbol{n}$ points $\boldsymbol{p_1, \ldots, p_n}$ with real coordinates in 2 dimensions ($\mathbb{R}^2$)

Find:

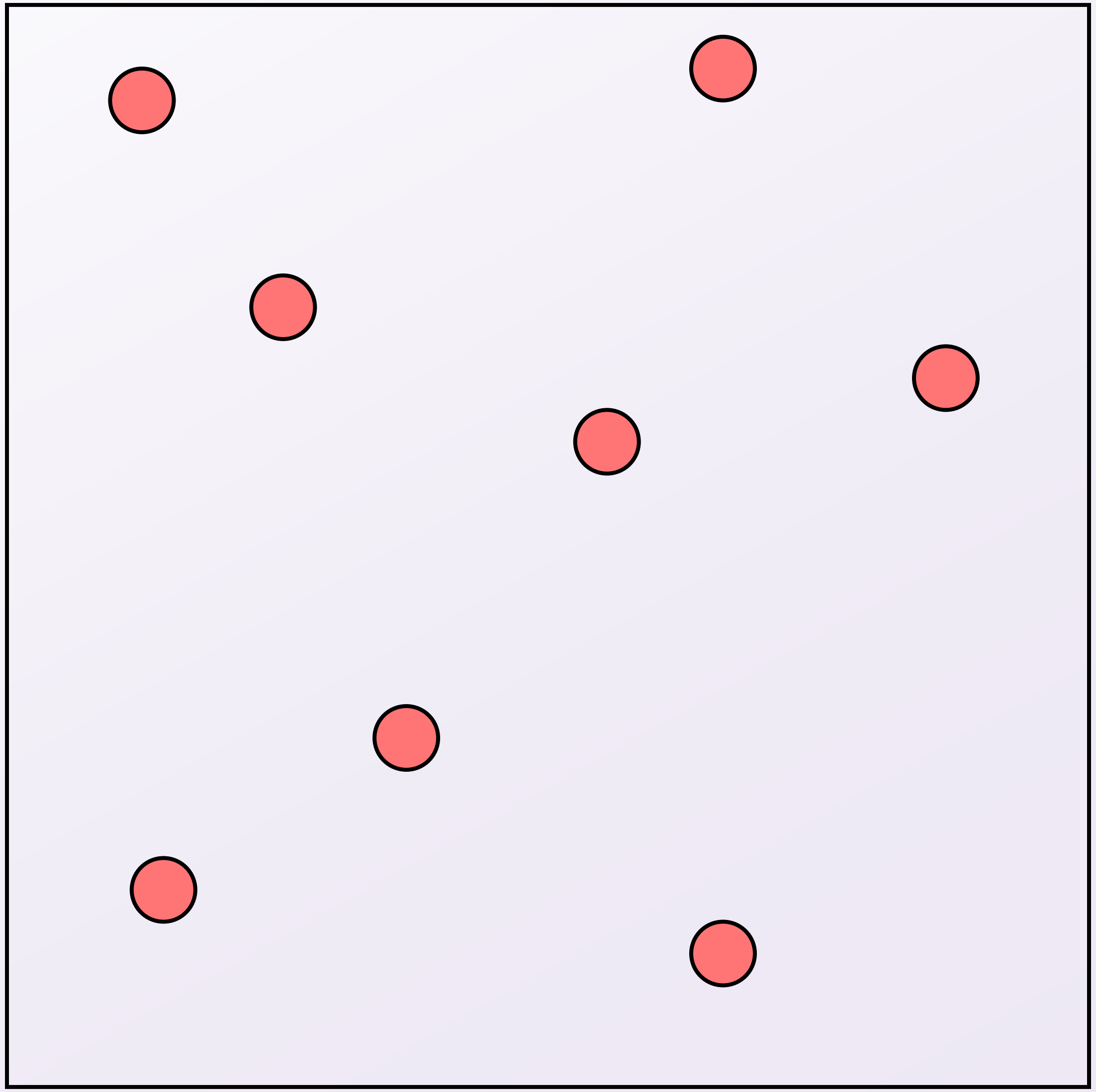
- A pair of points $\boldsymbol{p_i, p_j}$ s.t. the Euclidean distance $\boldsymbol{d(p_i, p_j)}$ is minimized

How about a $\Theta(\boldsymbol{n^2})$ algorithm?

- Try all possible pairs, keeping the smallest

Our goal:

- Use D&C to create a $\Theta(n \log n)$ algorithm



Closest Pair of Point D&C Idea

To get $\boldsymbol{\Theta(n\ log\ n)}$, we will aim for $\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + n}$

Base Case:

If the number of points is small, do use a naïve solution

Divide:

Otherwise partition the points into 2 subsets

Running time "budget" $O(n)$

Conquer:

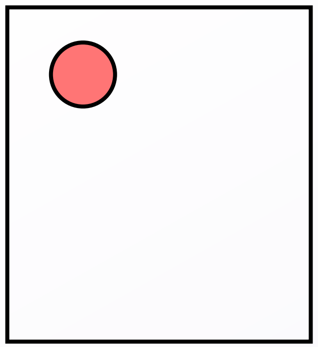
Find the closest pair of points in each subset

Combine:

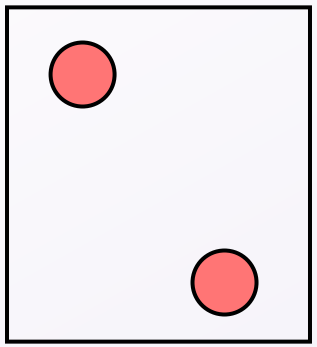
Use those closest pairs of points to find the closest overall

Running time "budget" $O(n)$

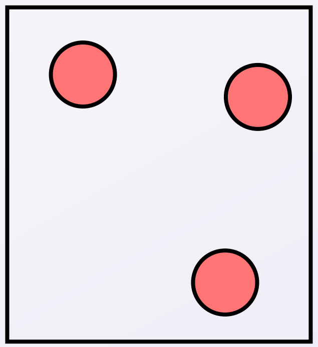
Closest Pair: Base Cases



If $n = 1$
return ∞

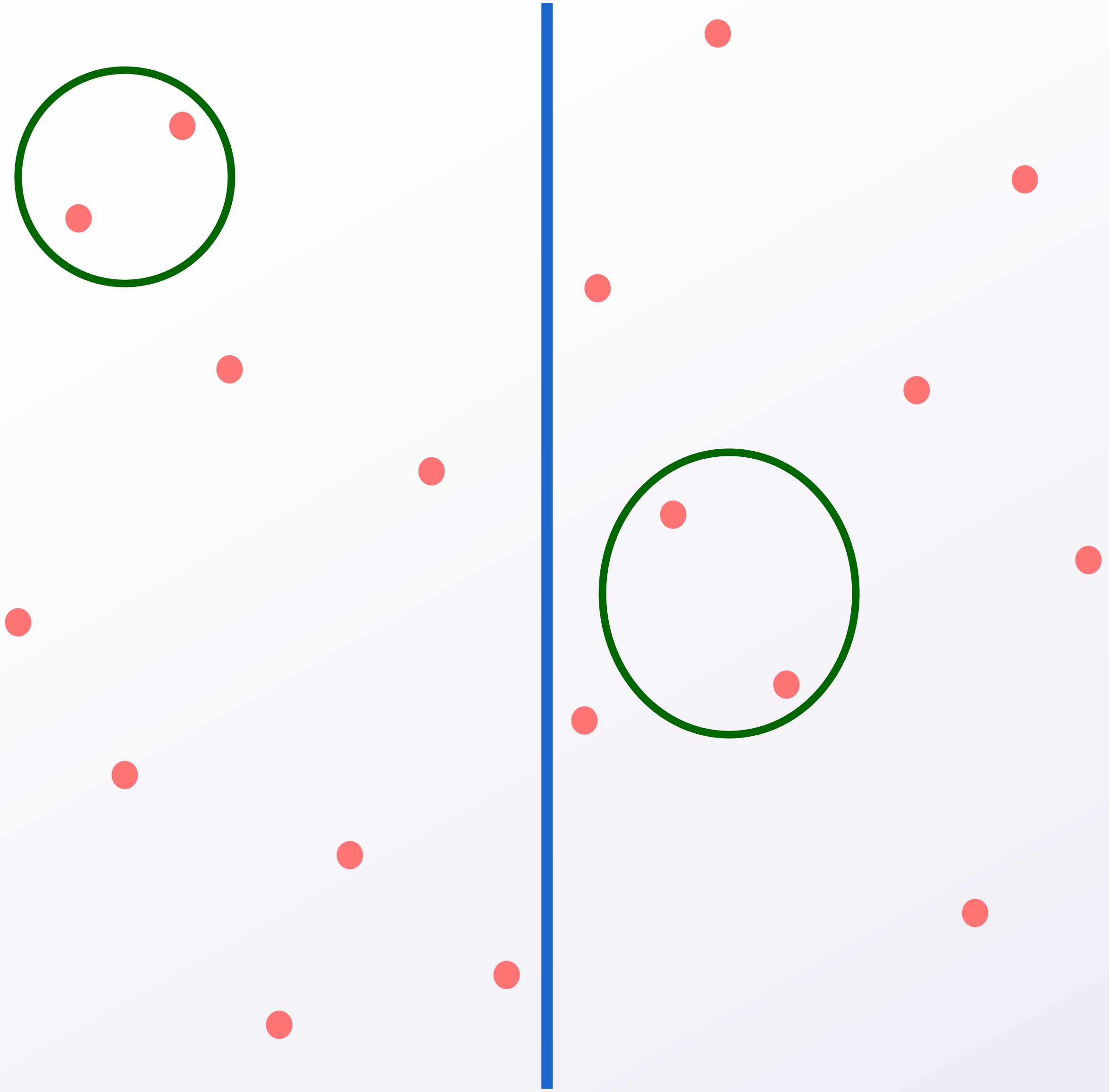


If $n = 2$
return the distance



If $n = 3$
check all 3 pairs
return the closest

Closest Pair: First Idea



Divide:

- Split using **median** $\boldsymbol{x}$-coordinate
- each subpart has size $\boldsymbol{n/2}$.

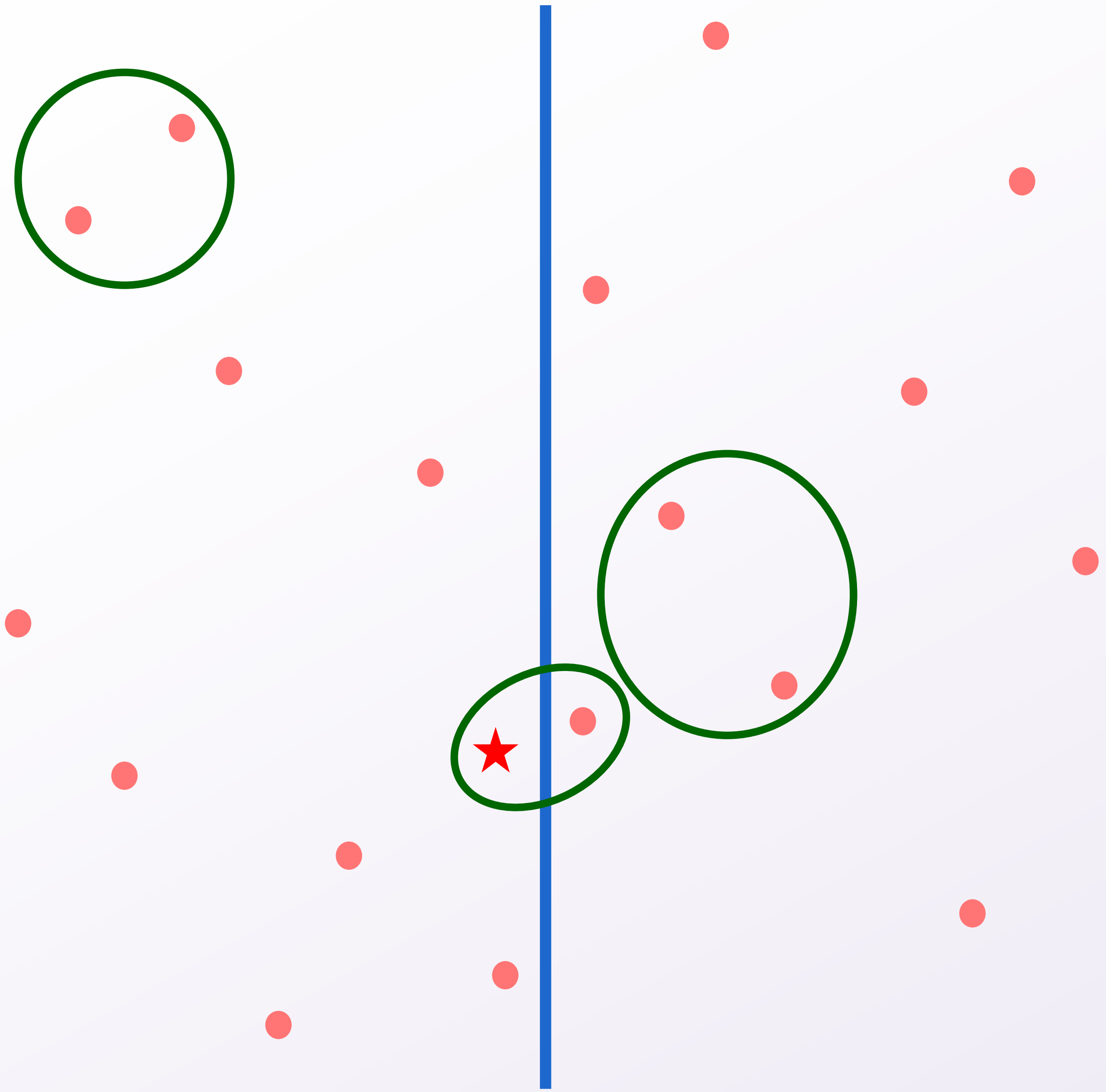
Conquer:

- Solve both size $\boldsymbol{n/2}$ subproblems
- We now have the closest pair from the left and from the right

Combine:

- Return the closer of the left pair and the right pair

Closest Pair: First Idea - Problem

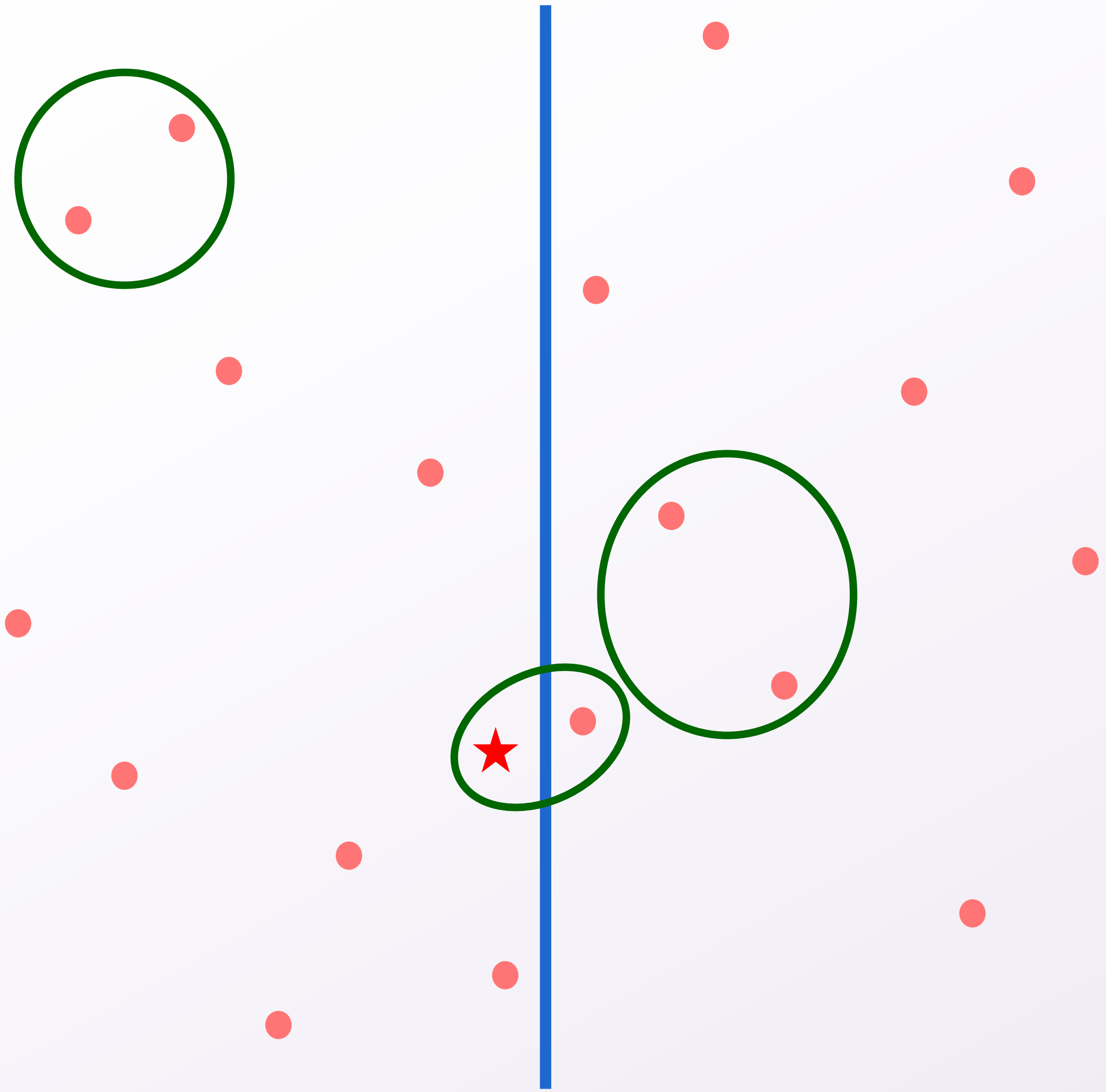


Divide:
- Split using **median** x-coordinate
- each subpart has size $n/2$.

Conquer:
- Solve both size $n/2$ subproblems
- We now have the closest pair from the left and from the right

Combine:
- Find the closest pair crossing the middle
- Return the closest of the left, right, and crossing pairs

Finding the Closest Crossing Pair – 1st Idea



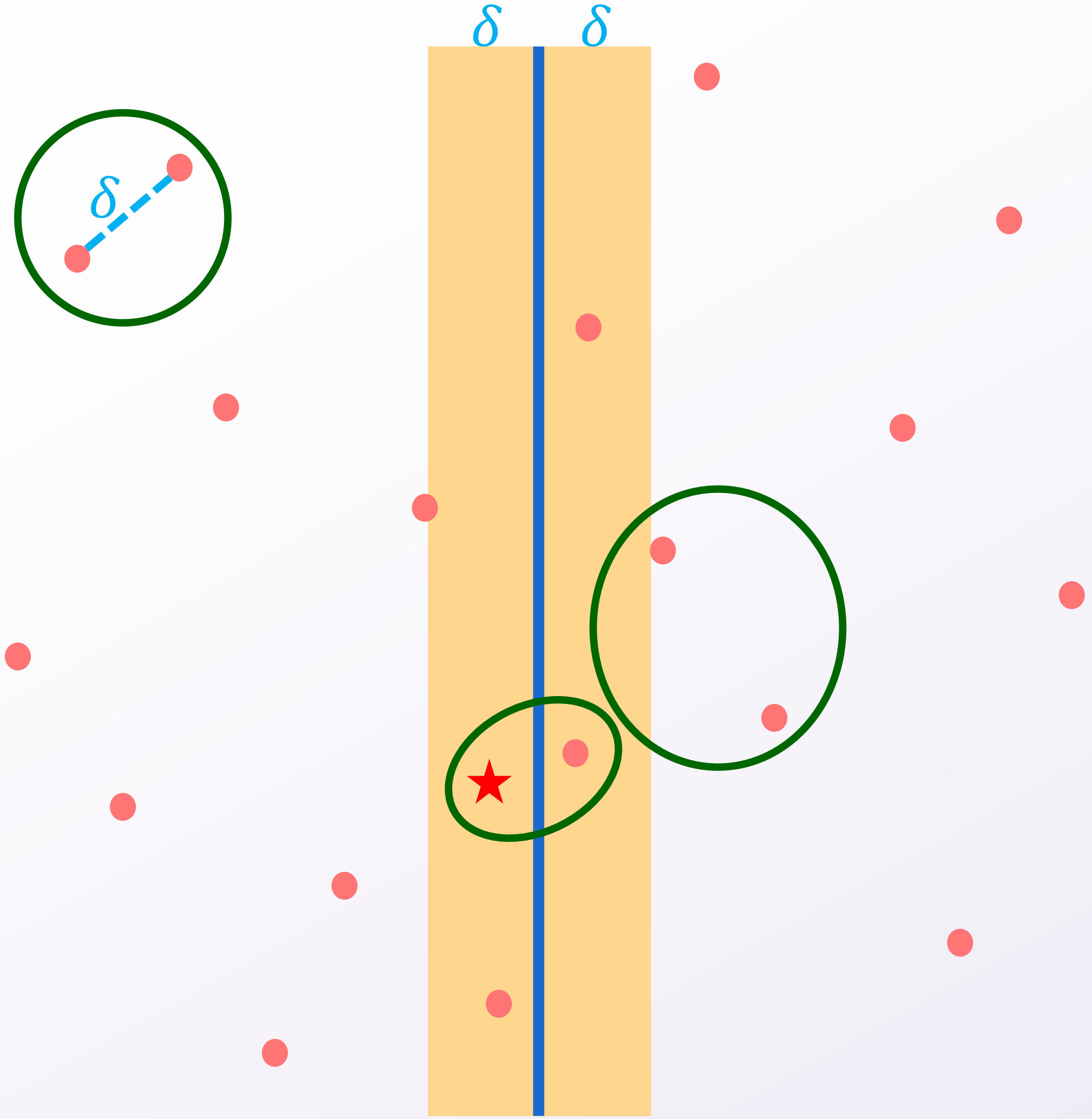
Combine:
- Find the closest pair crossing the middle
- Return the closest of the left, right, and crossing pairs

Procedure:
- For each point on the left, find its closest point on the right
- Save the closest seen as the crossing pair

Problem?

Running time is $\left(\frac{n}{2}\right)^2$

Finding the Closest Crossing Pair – 2nd Idea



Combine:

- Find the closest pair crossing the middle
- Return the closest of the left, right, and crossing pairs

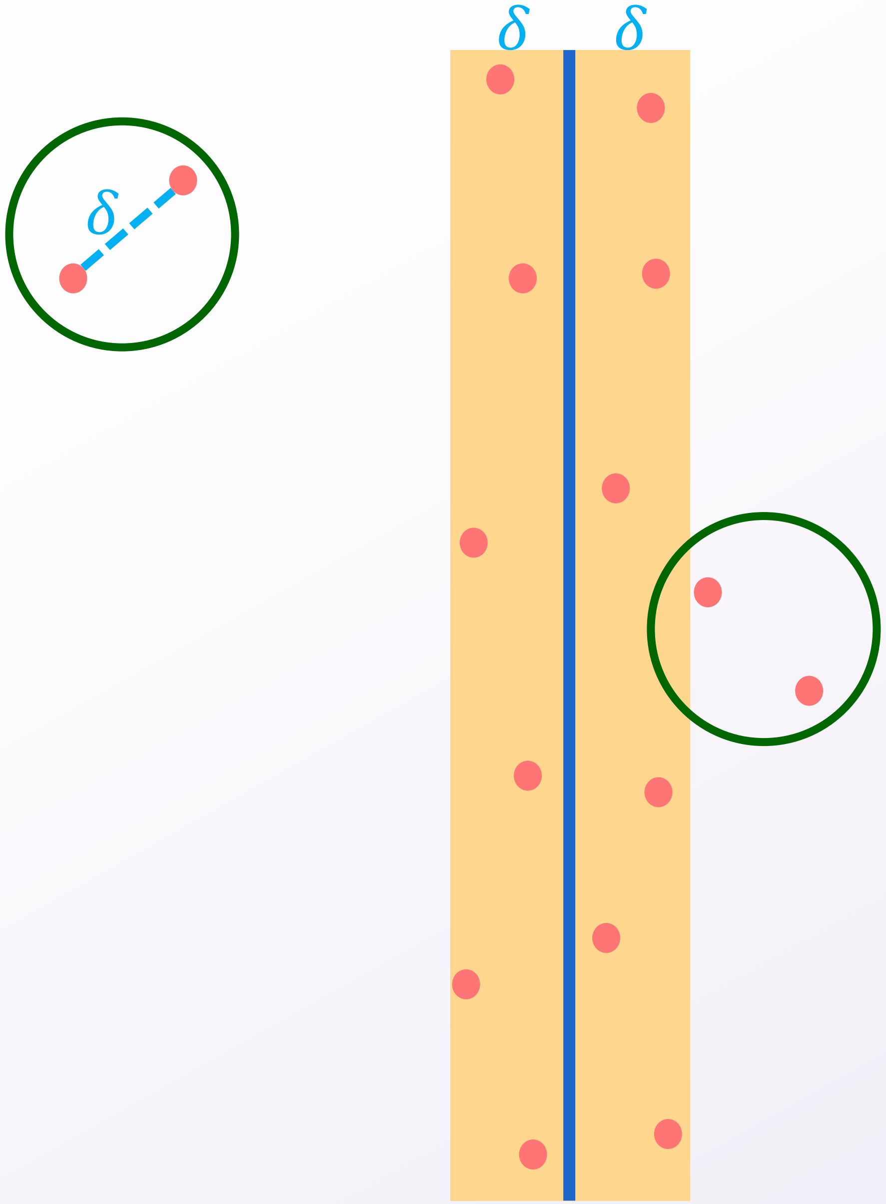
Observation:

- We only care about crossing pairs that might be closer than left and right
- Ignore points too far from the divide

Procedure:

- Let δ be the closest distance from left and right
- For each point on the left that's within δ of the divide, find its closest match from among points within δ on the right

Problem with the 2nd Idea



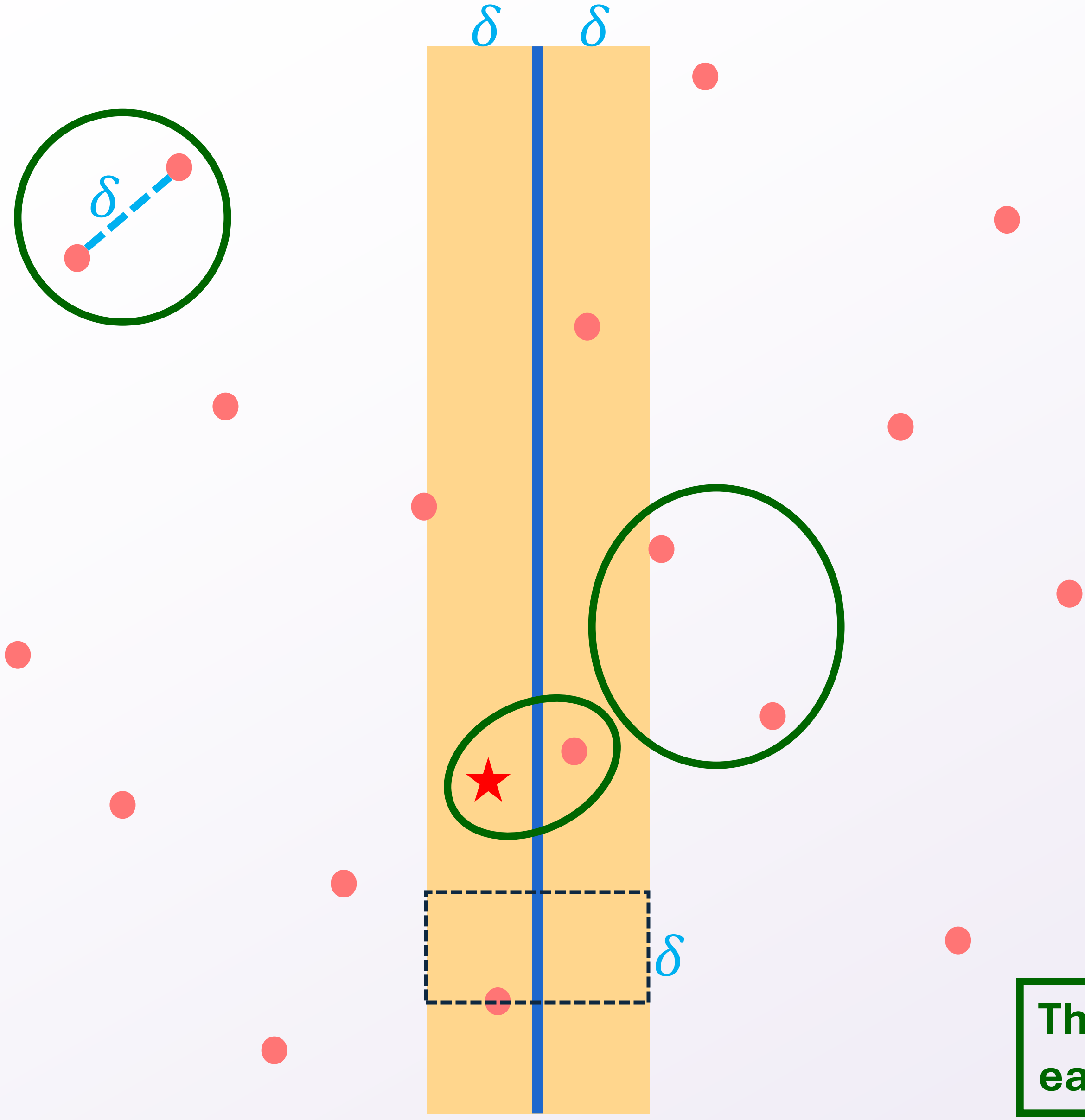
Combine:
- Find the closest pair crossing the middle
- Return the closest of the left, right, and crossing pairs

Observation:
- We only care about crossing pairs that might be closer than left and right
- Ignore points too far from the divide

Problem:
- We could still exceed our budget!

Solution:
- Re-apply the observation vertically!
- We only need to consider points within δ above the current point as well!

Finding the Closest Crossing Pair – 3rd Idea



Combine:

- Find the closest pair crossing the middle
- Return the closest of the left, right, and crossing pairs

Procedure:

- Let δ be the closest distance from left and right
- From bottom to top, for each point p_l on the left that's within δ of the divide on the left:
    - compare it to each point on the right that is within δ of the divide and no more than δ above p_l

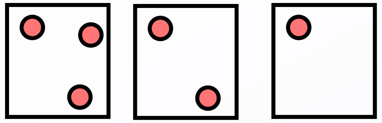
This will only fit within our budget if we compare each p_l to a constant number of other points

Divide and Conquer (Closest Pair of Points)

Preprocessing:

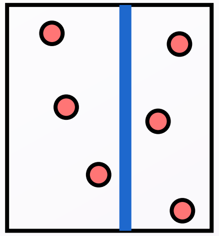
Sort the points by x coordinate (call this list L_x)

Make a copy of the points and sort by y coordinate (call this list L_y)



Base Case:

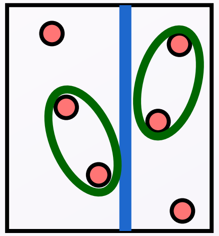
If there's 1 point then return ∞, If there's 2 or 3 points, solve naively



Divide:

Find the median x coordinate

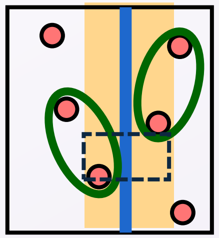
Partition L_x and L_y into the points on the left vs. right of the median



Conquer:

Recursively find the closest pair from among the left and right of the median

Combine:



Let δ be the closest from the left and the right solutions

Filter L_y to include only the points within δ of the median x

For each point p still in L_y:

- For each point within δ of p vertically:
  - Compare p with that point and save if the distance is less than δ

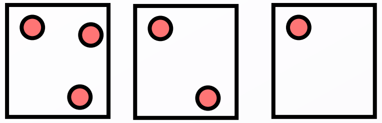
Return minimum of the saved pair and the one used for δ

Surprisingly, This works!

Preprocessing:

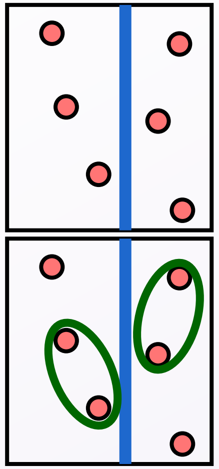
Sort the points by x coordinate (call this list L_x)

Make a copy of the points and sort by y coordinate (call this list L_y)



Base Case:

If there's 1 point then return ∞, If there's 2 or 3 points, solve naively



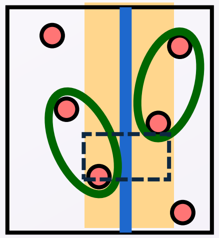
Divide:

Find the median x coordinate

Partition L_x and L_y into the points on the left vs. right of the median

Conquer:

Recursively find the closest pair from among the left and right of the median



Combine:

Let δ be the closest from the left and the right solutions

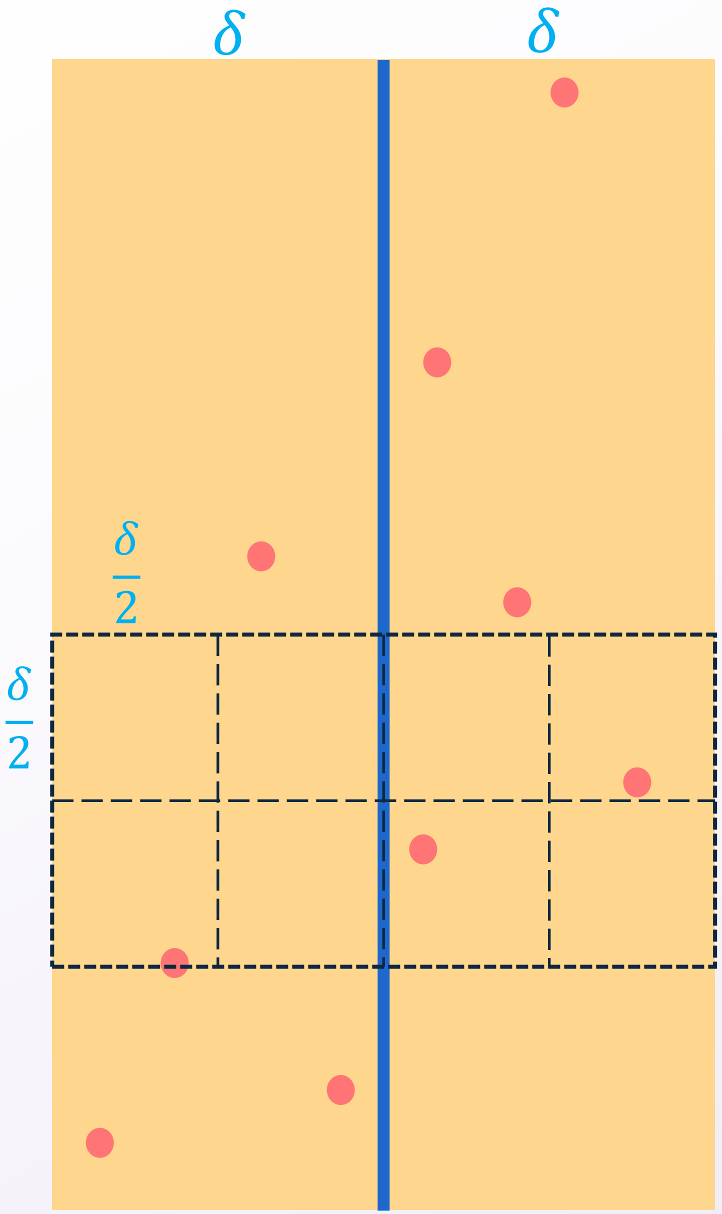
Filter L_y to include only the points within δ of the median x

For each point p still in L_y:

- **For the next 7 points vertically:**
  - Compare p with that point and save if the distance is less than δ

Return minimum of the saved pair and the one used for δ

Why is 7 enough?

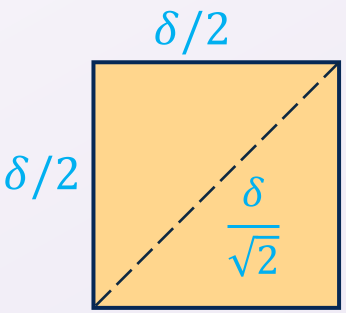


Claim:

- For any point p in the "strip", the 8th point above it is guaranteed to be more than δ away.

Proof:

- Consider a grid of $\frac{\delta}{2} \times \frac{\delta}{2}$ squares starting from p
- Any two points within the same square are at most $\frac{\delta}{\sqrt{2}}$ apart.



- Because $\sqrt{2} > 1$, we know that $\frac{\delta}{\sqrt{2}} < \delta$
- Therefore, there is at most one point per square
- Besides the one which contains p there are only 7 other squares within range δ

Full Algorithm

ClosestPair(L):
  $L_x = L$ sorted by x coordinate
  $L_y = L$ sorted by y coordinate
  return ClosestPairRec(L_x, L_y)

ClosestPairRec(L_x, L_y):
  # Base cases omitted
  $m =$ median x coordinate
  $P_{x1} =$ the points from L_x to the left of the median
  $P_{y1} =$ the points from L_y to the left of the median
  $P_{x2} =$ the points from L_x to the right of the median
  $P_{y2} =$ the points from L_y to the right of the median
  $a_1 =$ ClosestPairRec(P_{x1}, P_{y1})
  $a_2 =$ ClosestPairRec(P_{x2}, P_{y2})
  $a =$ closer of a_1 and a_2
  $\delta =$ distance(a)
  for each p in L_y:
    if p's x coordinate is more than δ from m:
      remove p from L_y
  for each p in L_y:
    for each of the next 7 points q in L_y:
      if distance(p, q):
        $a = (p, q)$
  return a