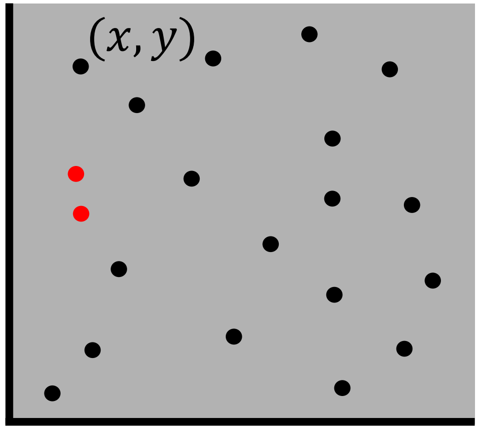
Closest Pair of Points

CSCI 232

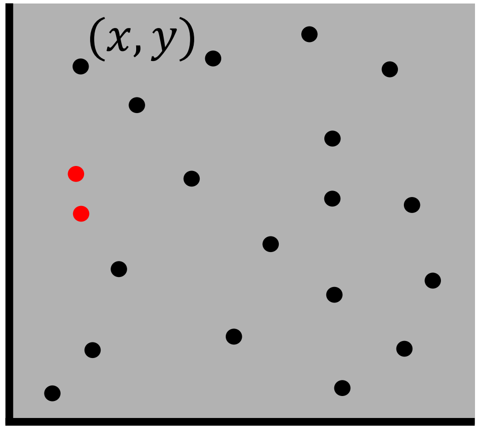
Closest Pair Problem



Given n points, find a pair of points with the smallest distance between them.

(Assume no points have the same x or y values).

Closest Pair Problem

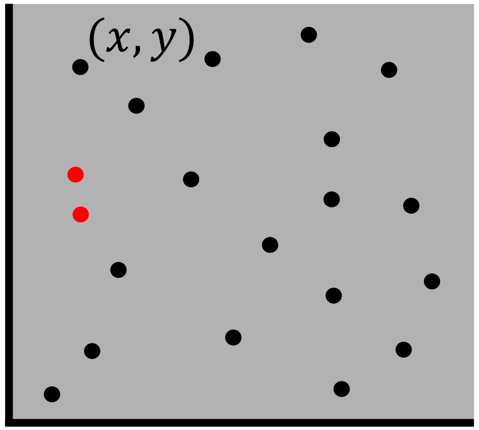


Given n points, find a pair of points with the smallest distance between them.

(Assume no points have the same x or y values).

Algorithm?

Closest Pair Problem



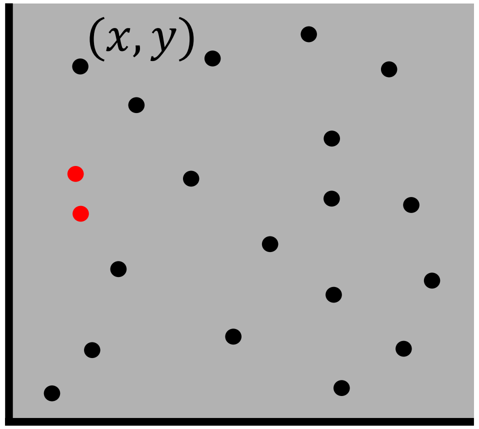
	P_1	P_2	...	P_n
P_1	/	$d_{1,2}$	...	$d_{1,n}$
P_2	$d_{2,1}$	/	...	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	...	/

Simple solution:

1. Compute distance for each pair.
2. Select smallest.

Running Time = ?

Closest Pair Problem



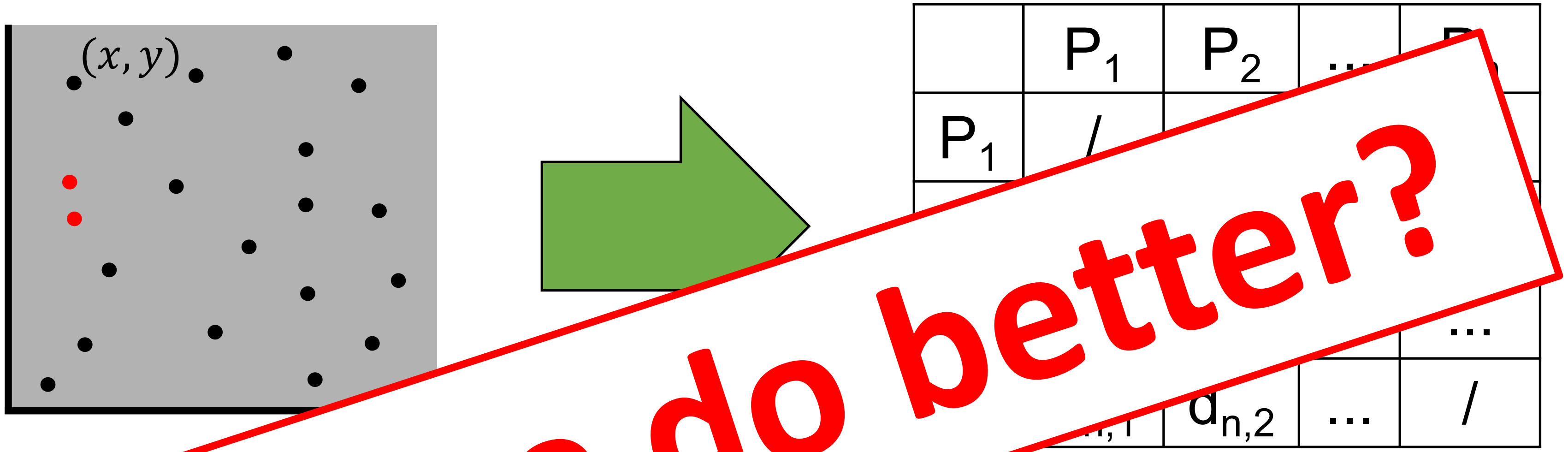
	P_1	P_2	…	P_n
P_1	/	$d_{1,2}$	…	$d_{1,n}$
P_2	$d_{2,1}$	/	…	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	…	/

Simple solution:

1. Compute distance for each pair.
2. Select smallest.

Running Time = $O(n^2)$

Closest Pair Problem



Can we do better?

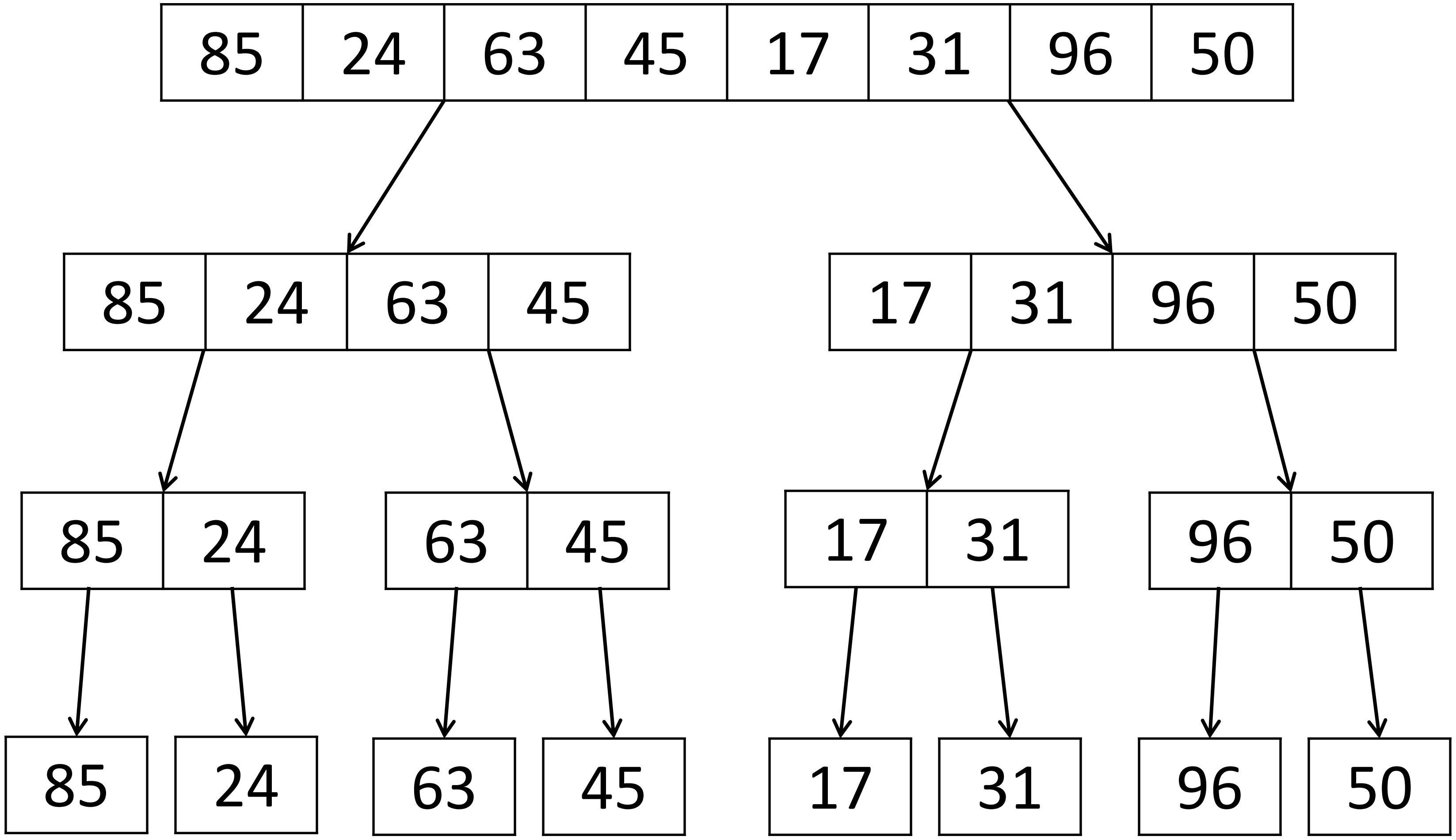
1. Compute distance for each pair.
2. Select smallest.

Running Time = $O(n^2)$

Divide and Conquer Battle Plan

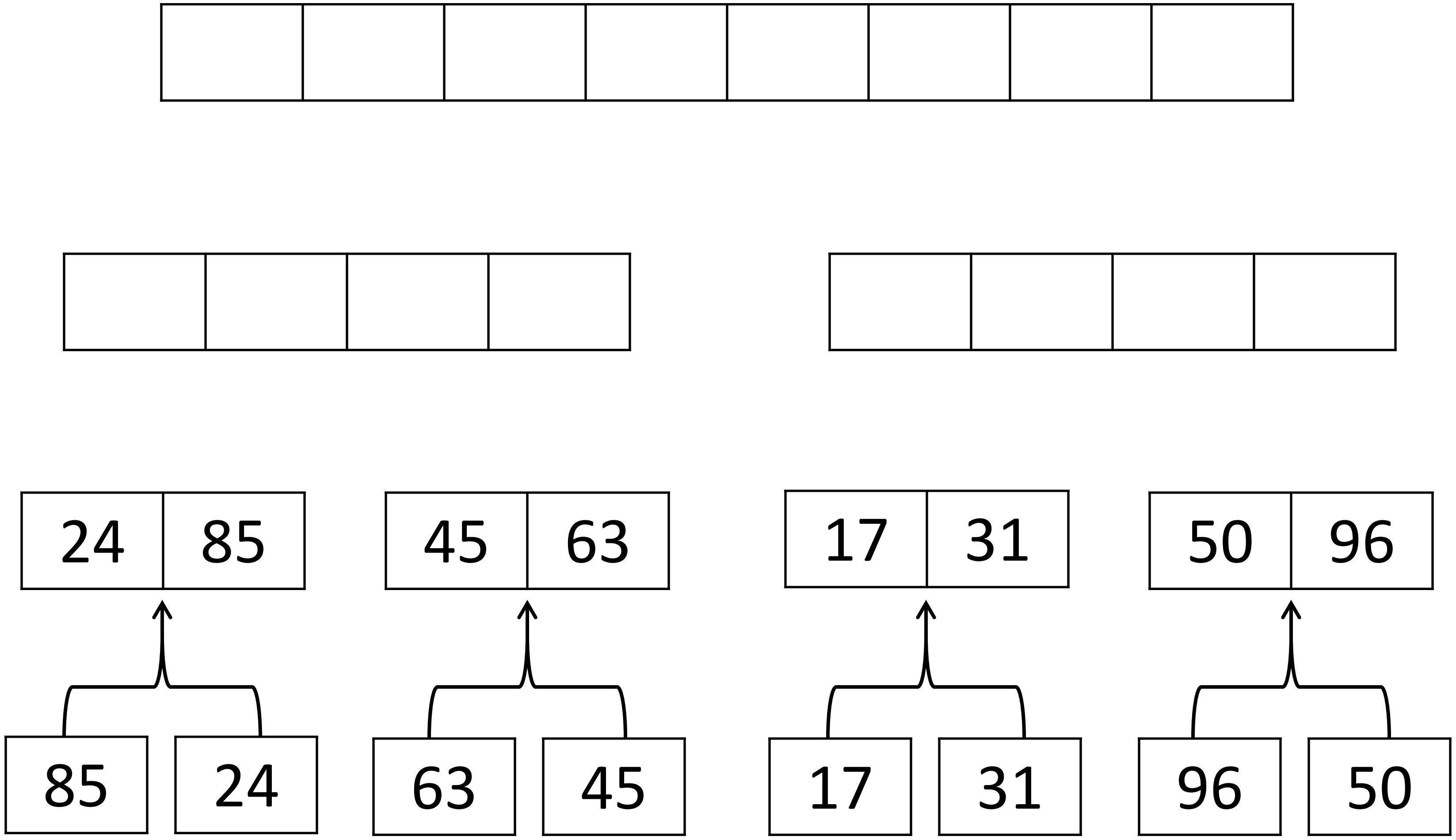
1. Divide problem into subproblems that are smaller instances of the same problem.
2. Conquer the subproblems by solving them recursively.
3. Combine the solutions to the subproblems into the solution for the original problem.

Divide and Conquer – Merge Sort



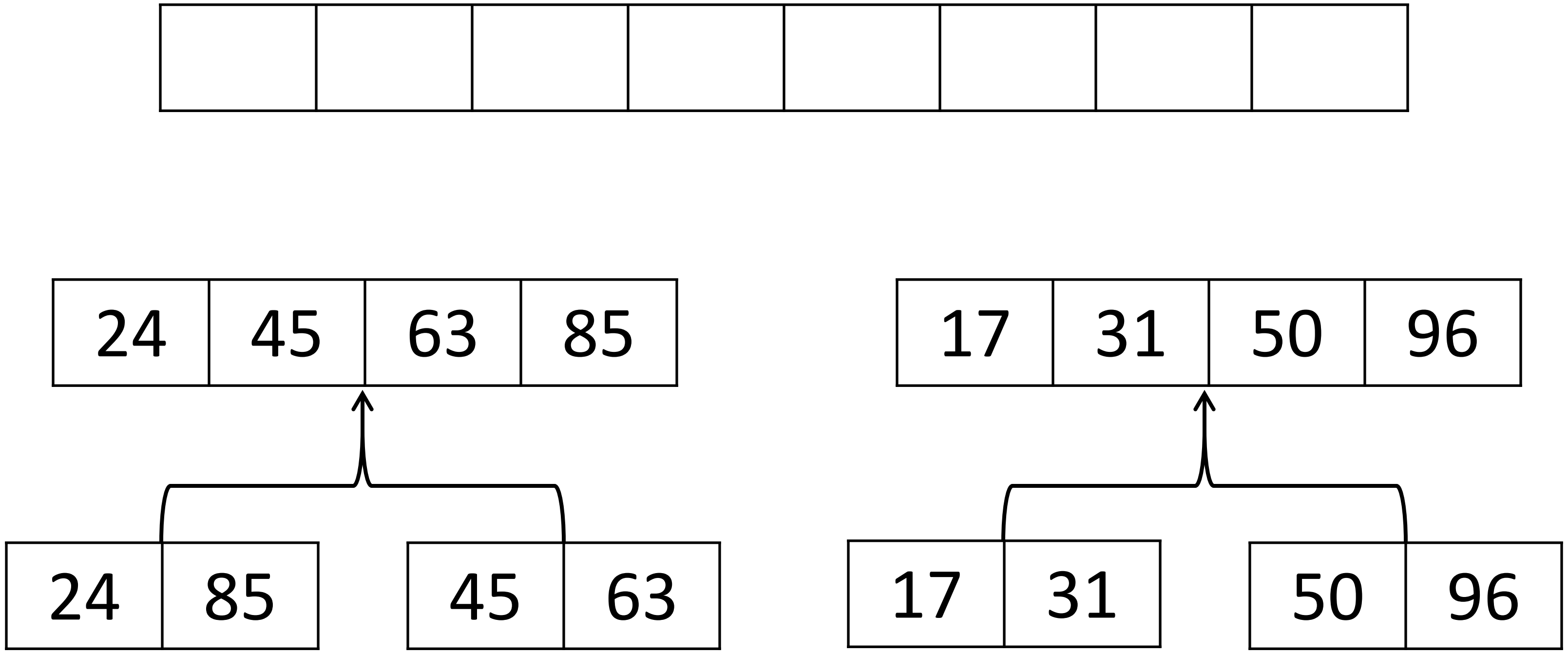
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Divide and Conquer – Merge Sort



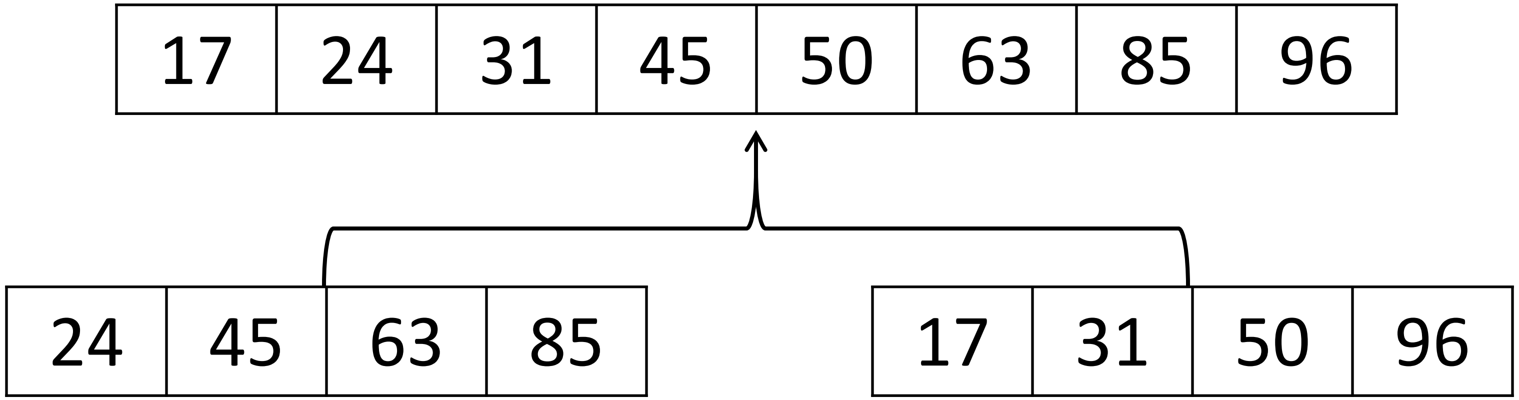
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Divide and Conquer – Merge Sort



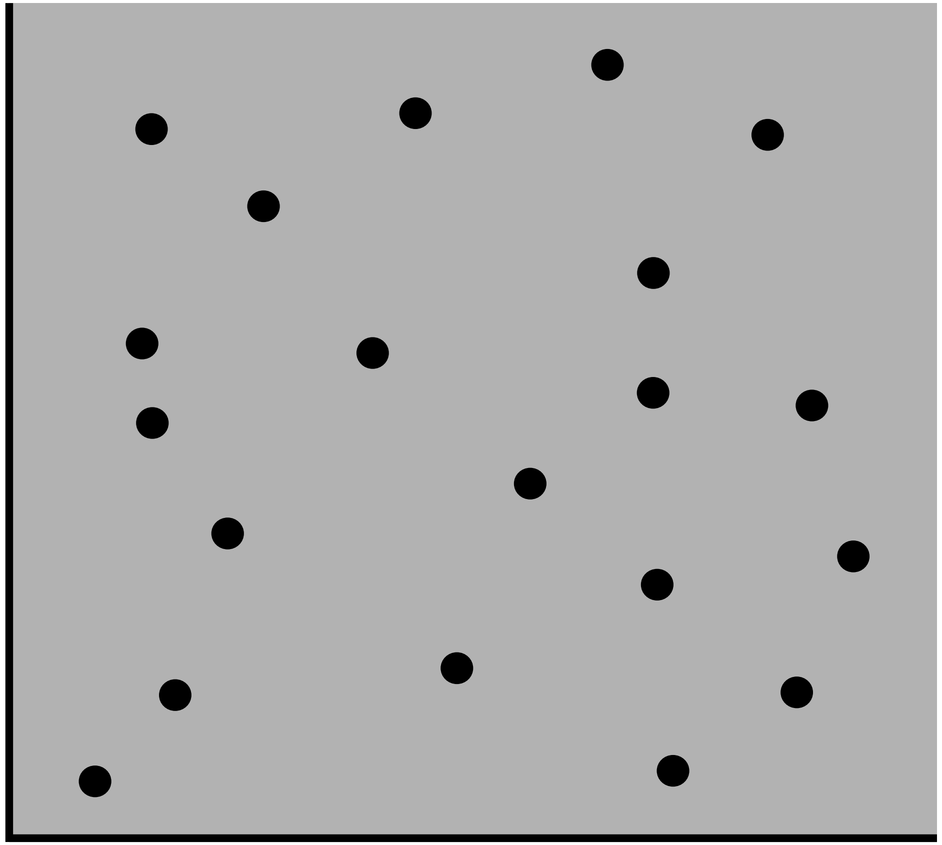
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Divide and Conquer – Merge Sort



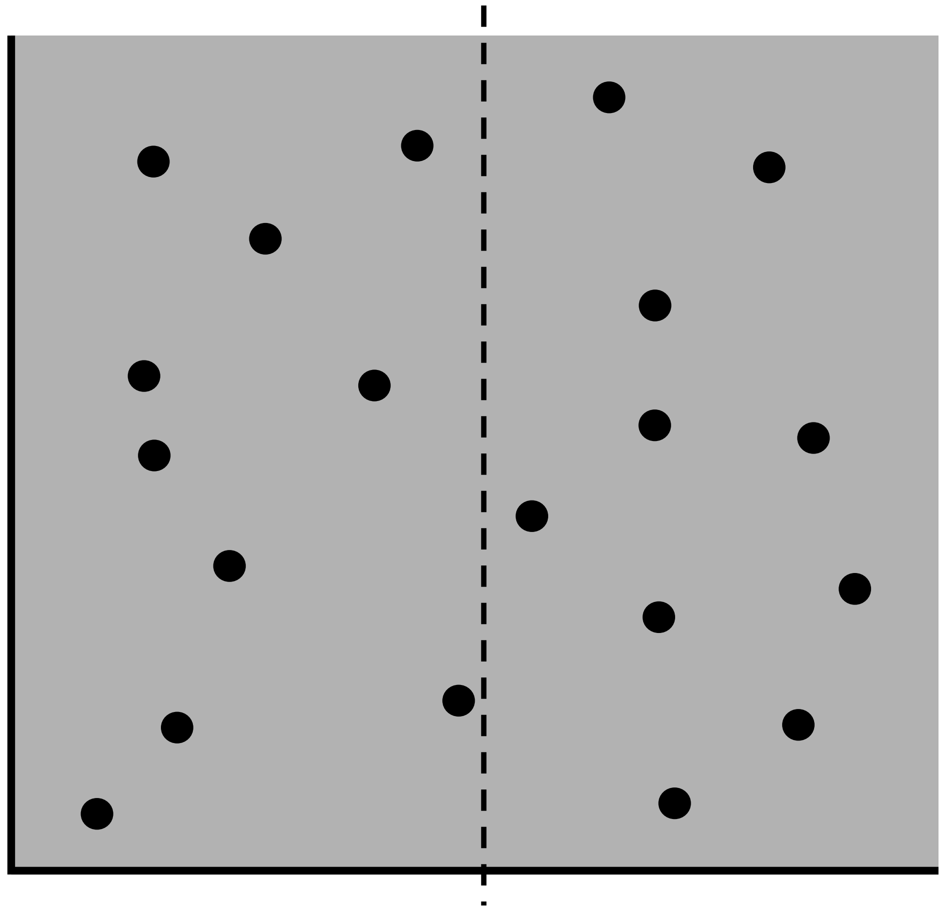
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Closest Pair Problem – Divide and Conquer



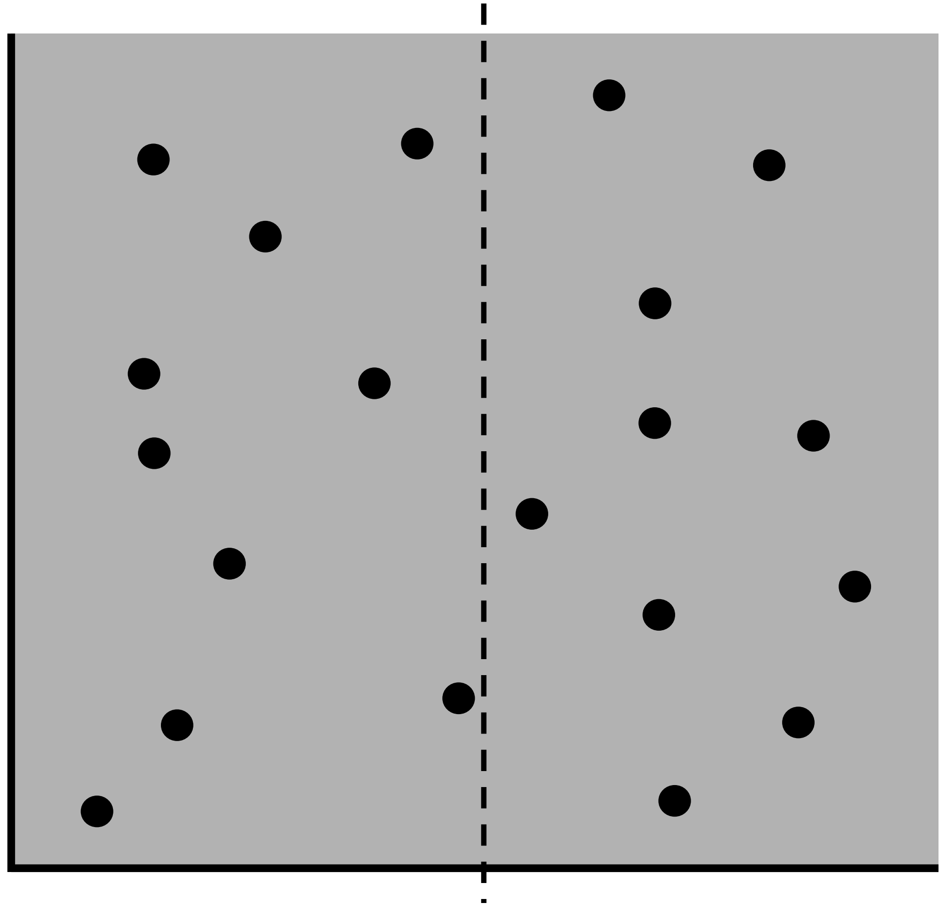
How can we make the problem smaller and easier?

Closest Pair Problem – Divide and Conquer



Divide: How can we draw line so that half of the points are on each side?

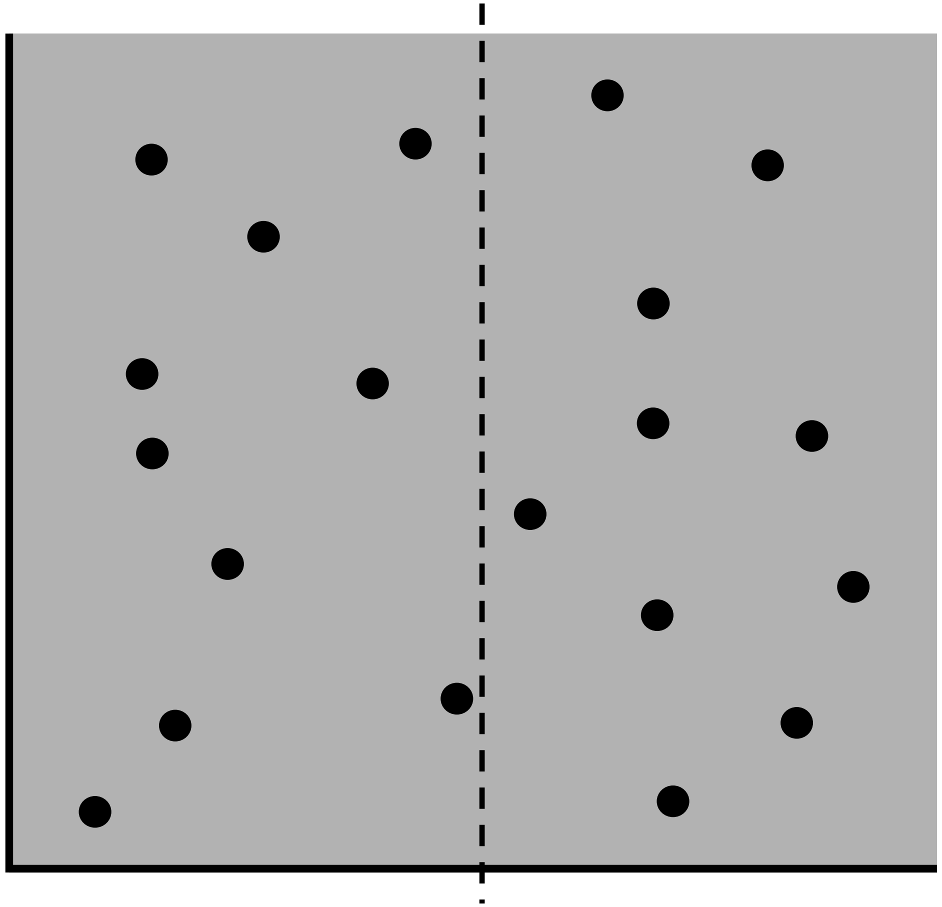
Closest Pair Problem – Divide and Conquer



Divide: How can we draw line so that half of the points are on each side?

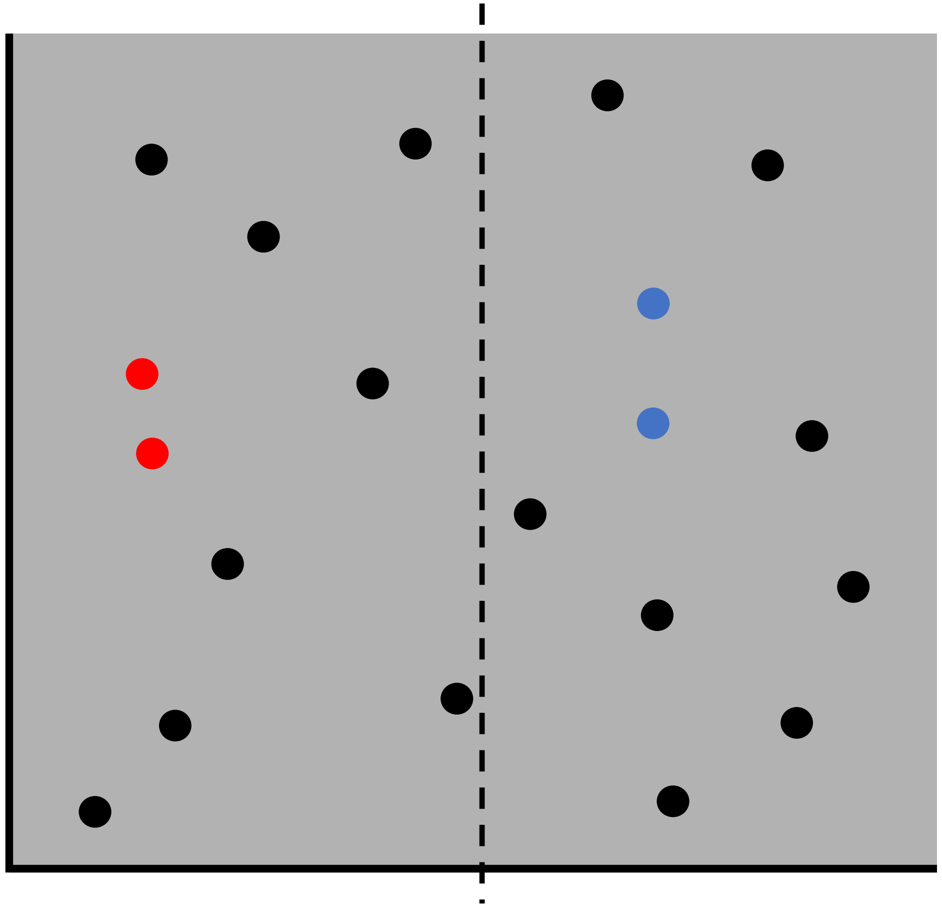
1. Sort by x-coordinate.
2. Put line at median value.

Closest Pair Problem – Divide and Conquer



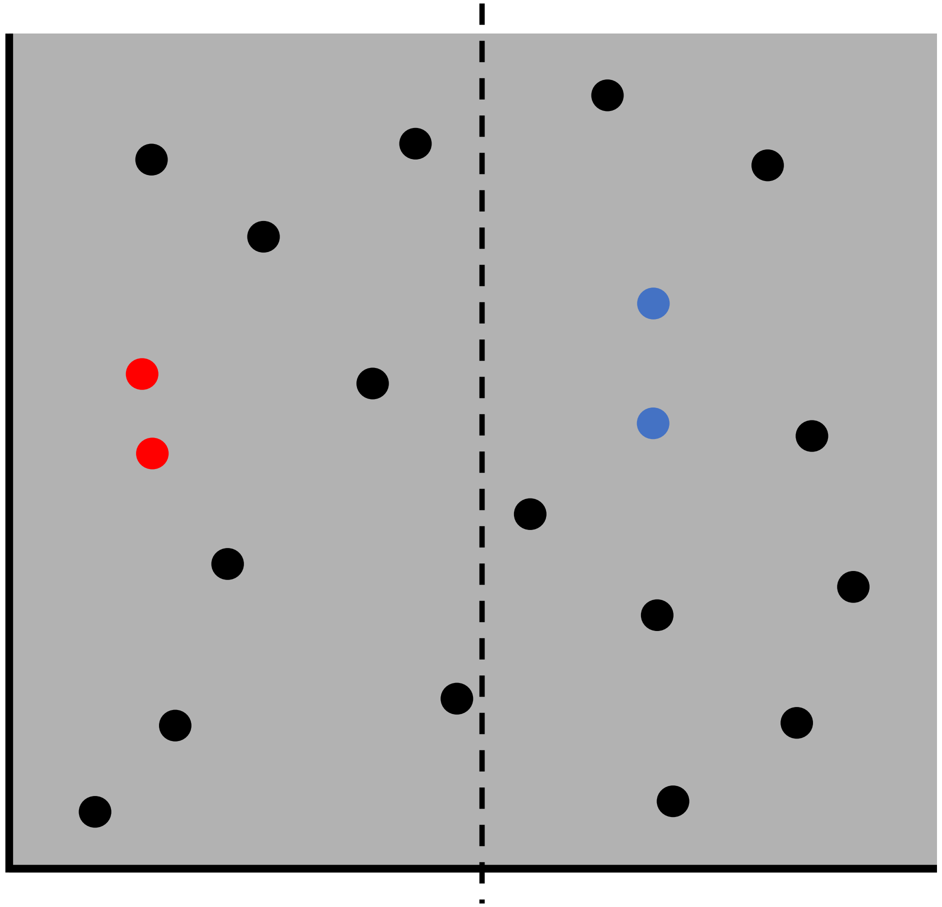
Conquer: Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer



Combine: If we had closest left and closest right, how do we determine closest?

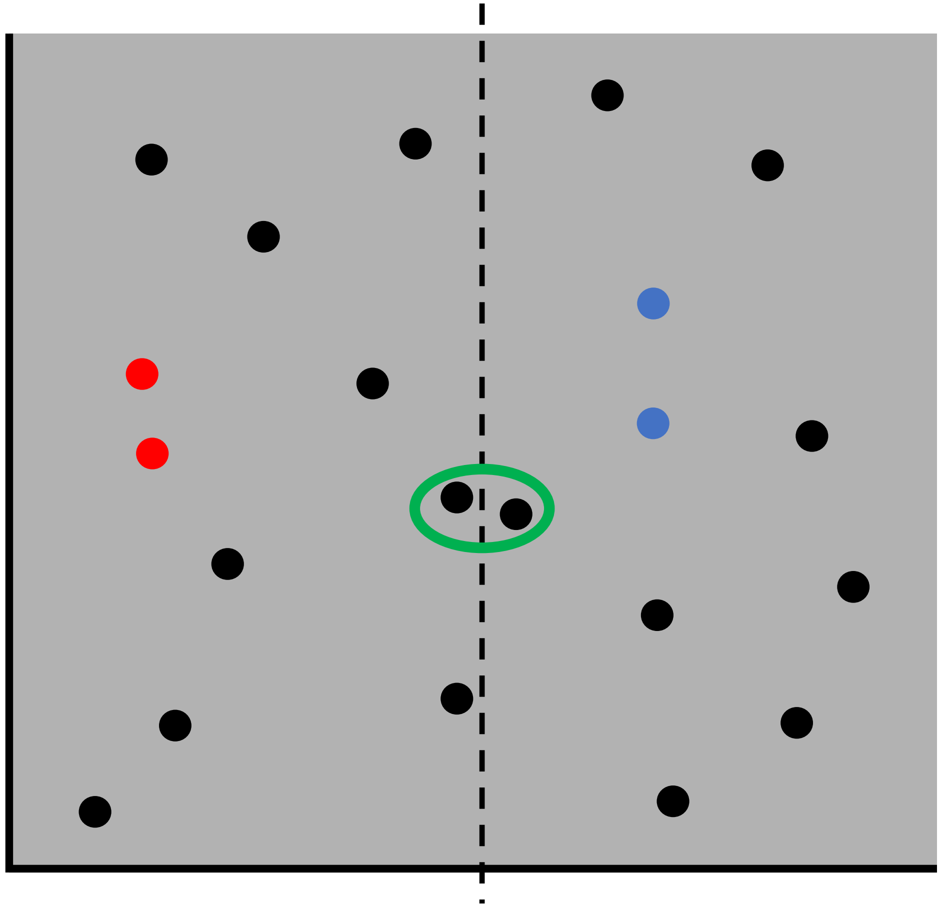
Closest Pair Problem – Divide and Conquer



Combine: If we had closest left and closest right, how do we determine closest?

1. Return minimum of: $d_{\text{left}}, d_{\text{right}}$.

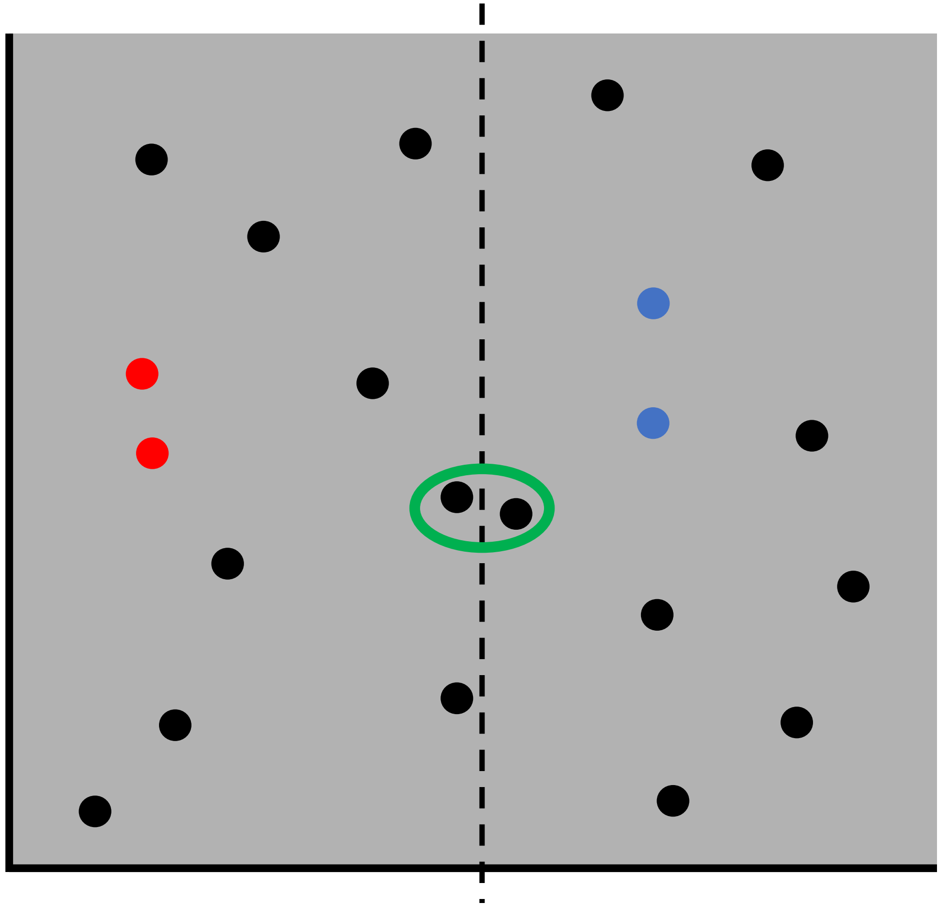
Closest Pair Problem – Divide and Conquer



Combine: If we had closest left and closest right, how do we determine closest?

1. Return minimum of: d_{left}, d_{right}.

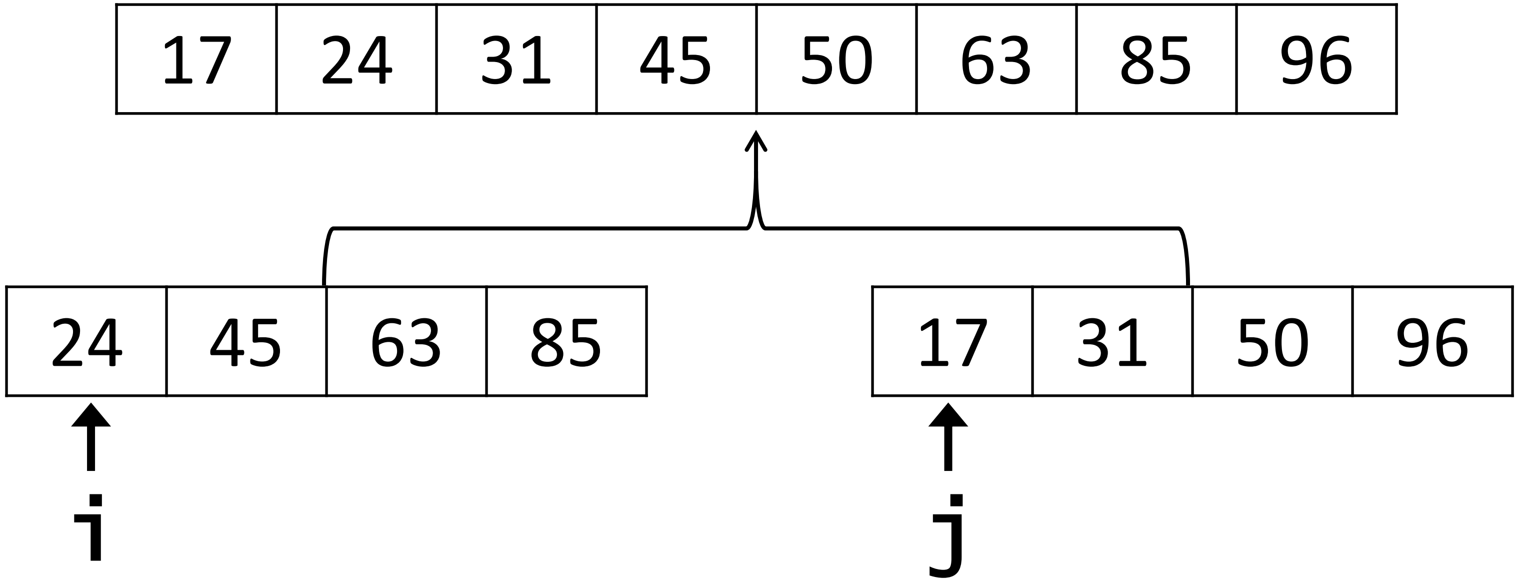
Closest Pair Problem – Divide and Conquer



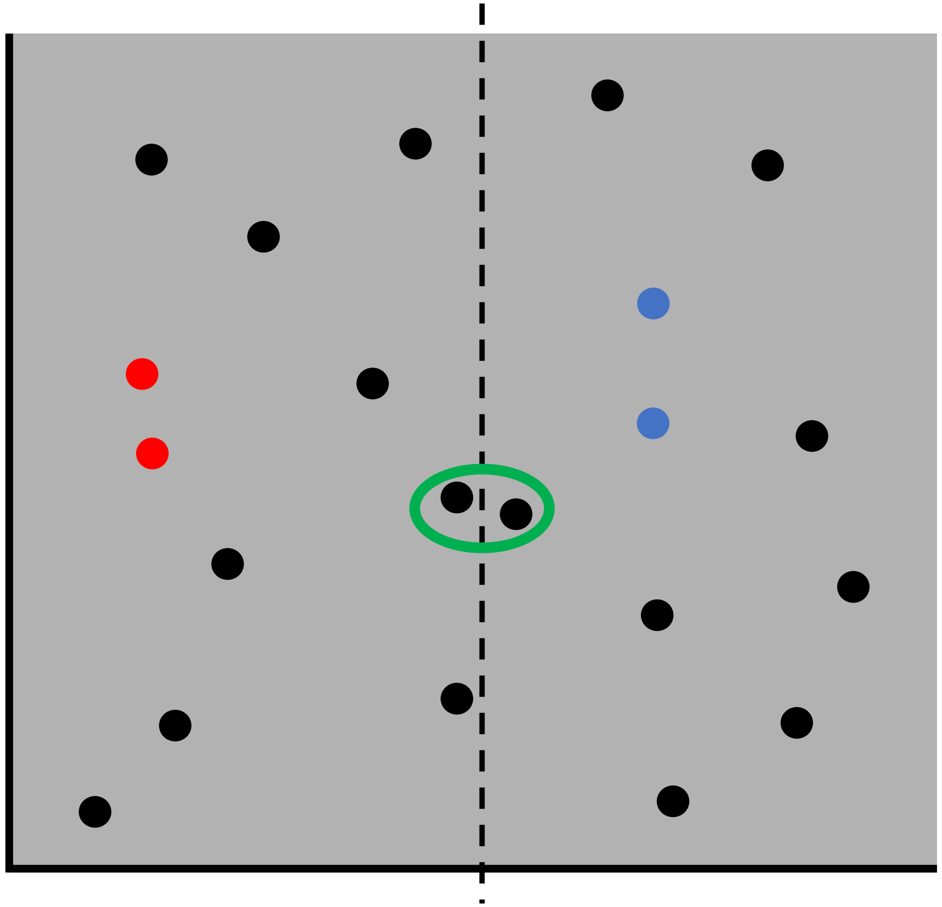
Combine: If we had closest left and closest right, how do we determine closest?

1. Return minimum of: d_{left}, d_{right}, $d_{\text{min\_straddle}}$.

Merge Sort – Combine



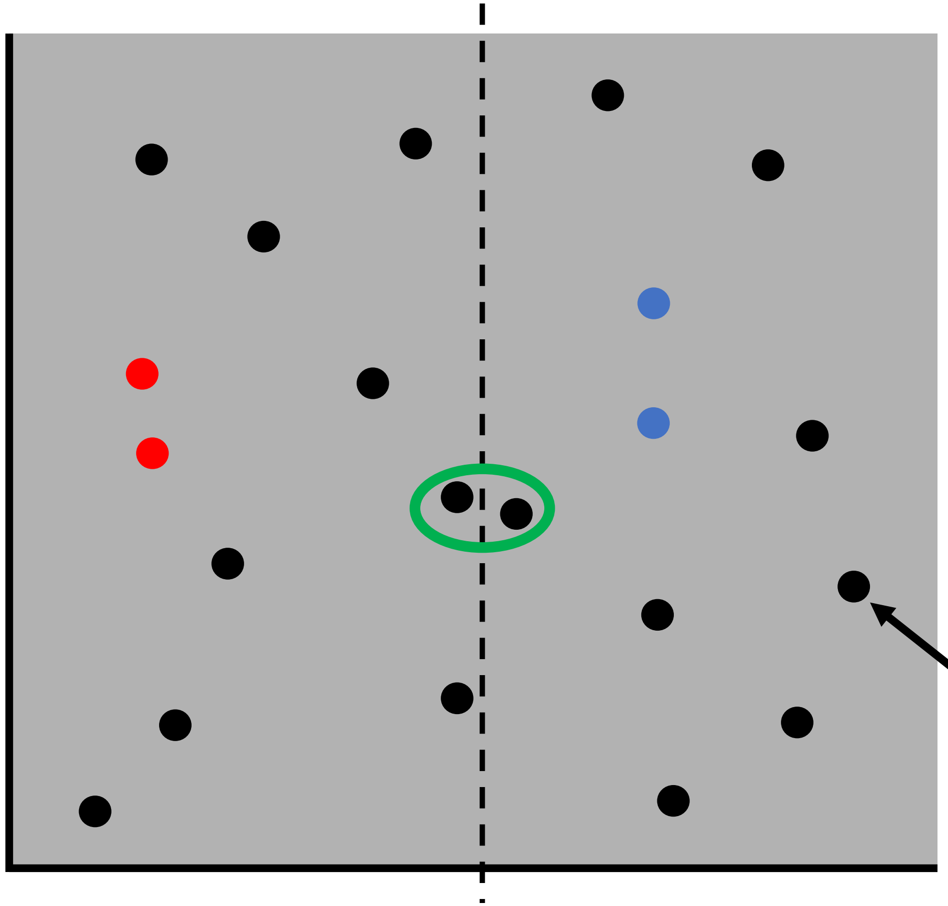
Closest Pair Problem – Divide and Conquer



How should we search for "straddle points"?

Suppose $\delta = \min(d_{\text{left}}, d_{\text{right}})$.

Closest Pair Problem – Divide and Conquer

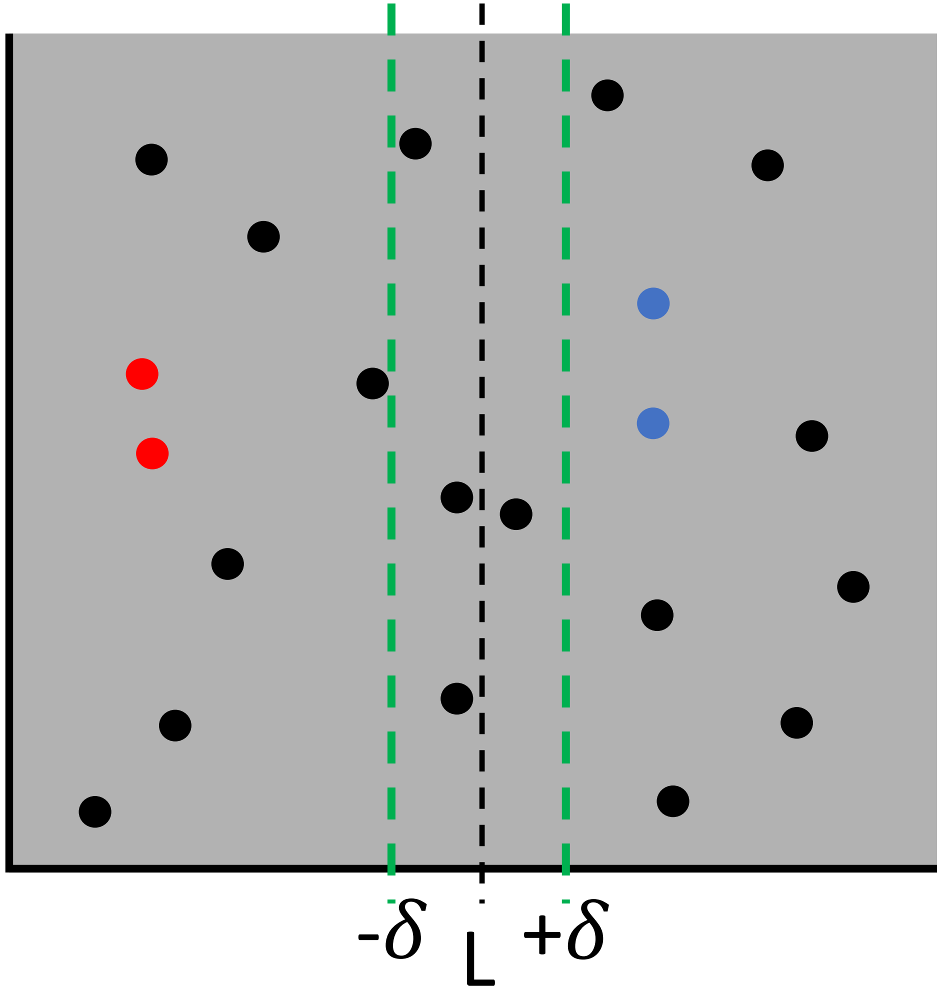


How should we search for “straddle points”?

Suppose $\delta = \min(d_{\text{left}}, d_{\text{right}})$.

Do we need to consider this point when looking for straddle points?

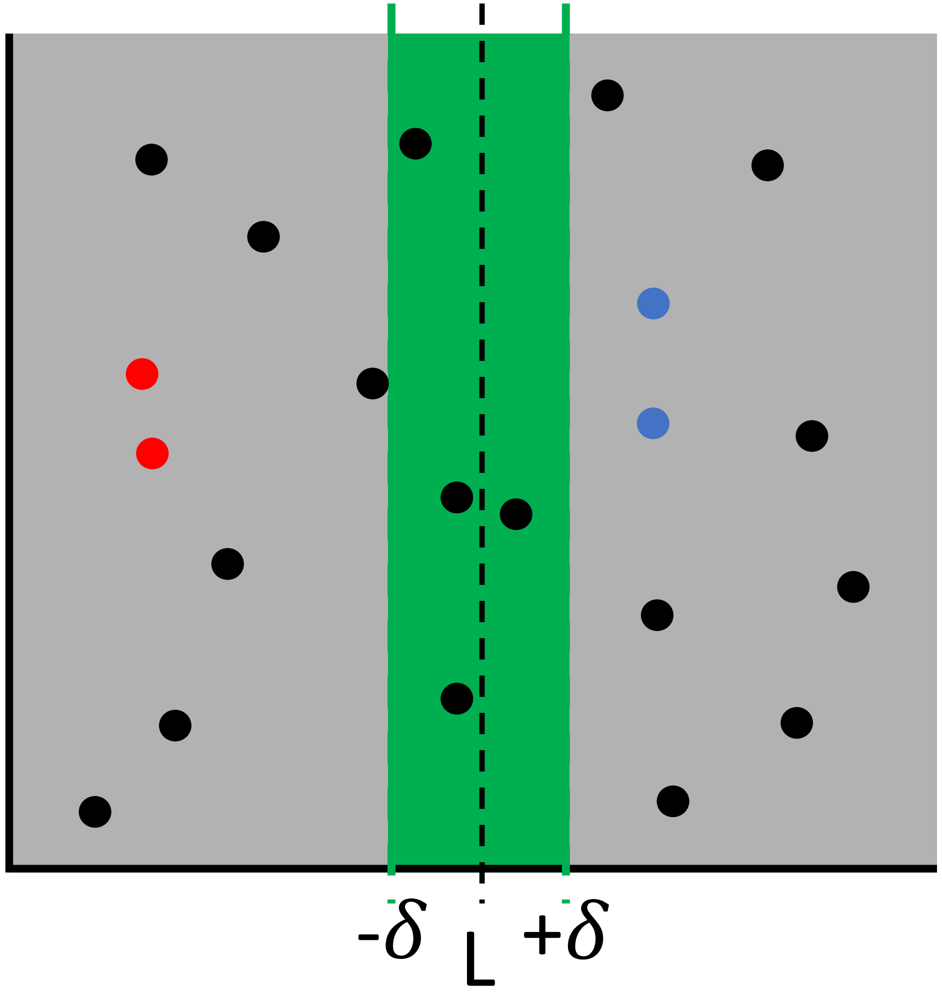
Closest Pair Problem – Divide and Conquer



Rule: We only need to hunt for straddle points at most δ away from L.

Reason: Points outside cannot reach the other side in less than δ.

Closest Pair Problem – Divide and Conquer

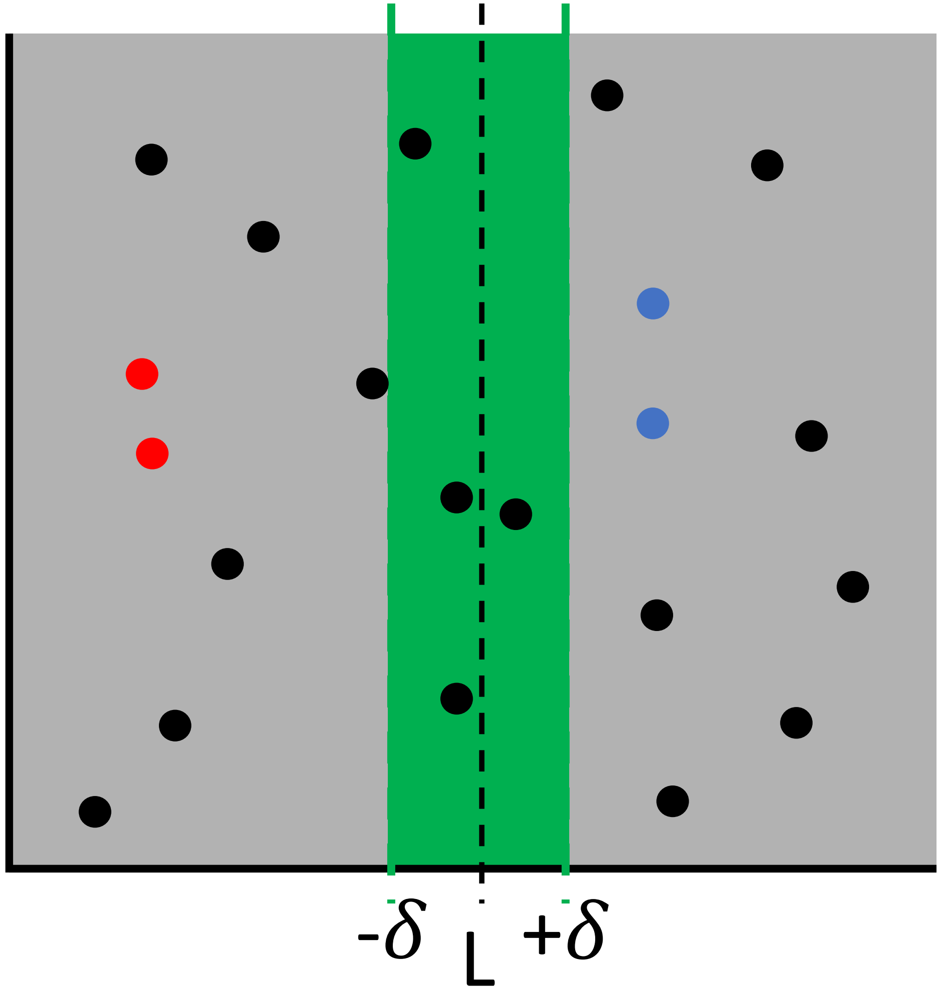


Rule: We only need to hunt for straddle points at most δ away from L.

Reason: Points outside cannot reach the other side in less than δ.

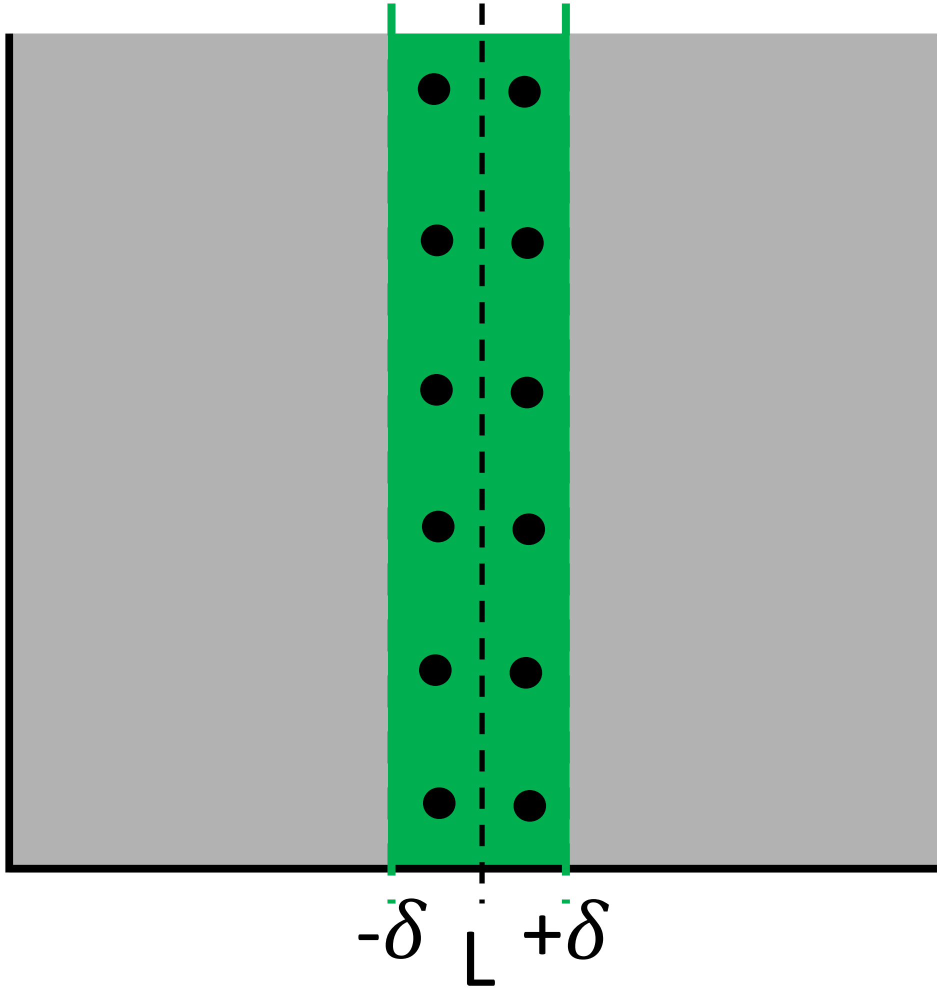
Let **S** be the set of straddle points.

Closest Pair Problem – Divide and Conquer



Can we just compare all left straddle points to all right straddle points?

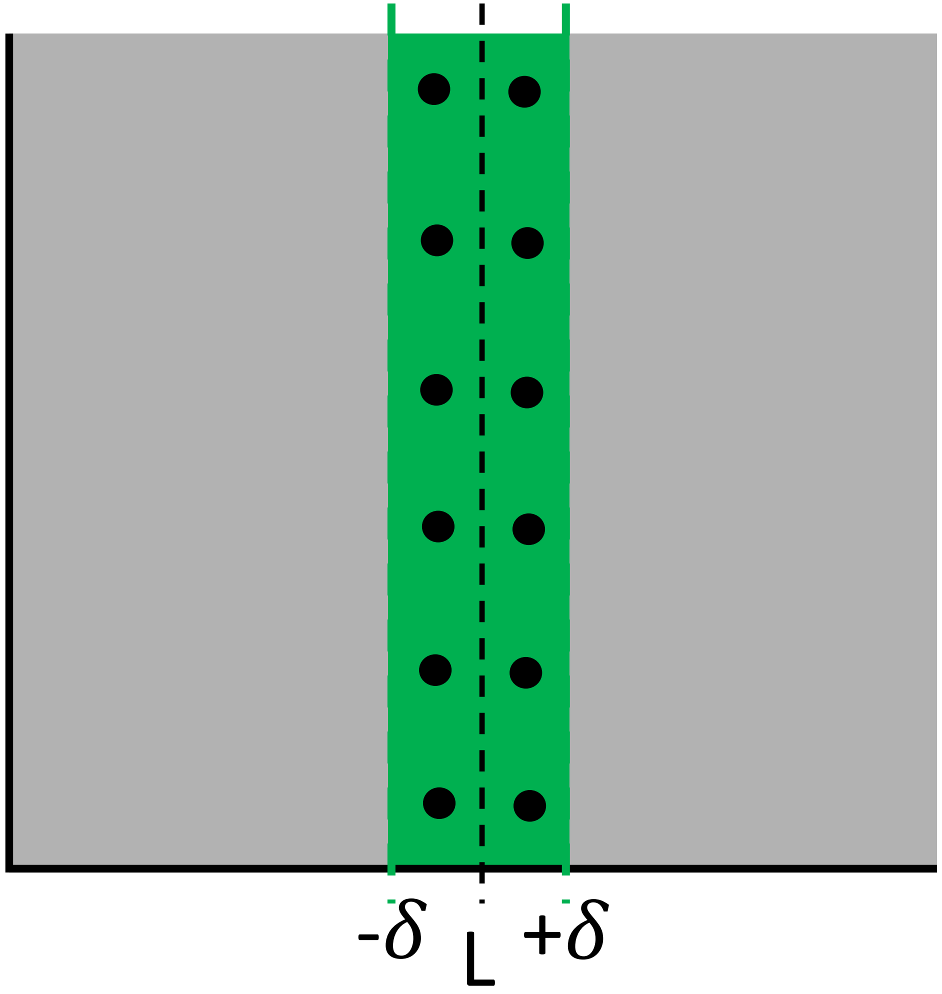
Closest Pair Problem – Divide and Conquer



Can we just compare all left straddle points to all right straddle points?

Yes, but running time could still be $O(n^2)$.

Closest Pair Problem – Divide and Conquer

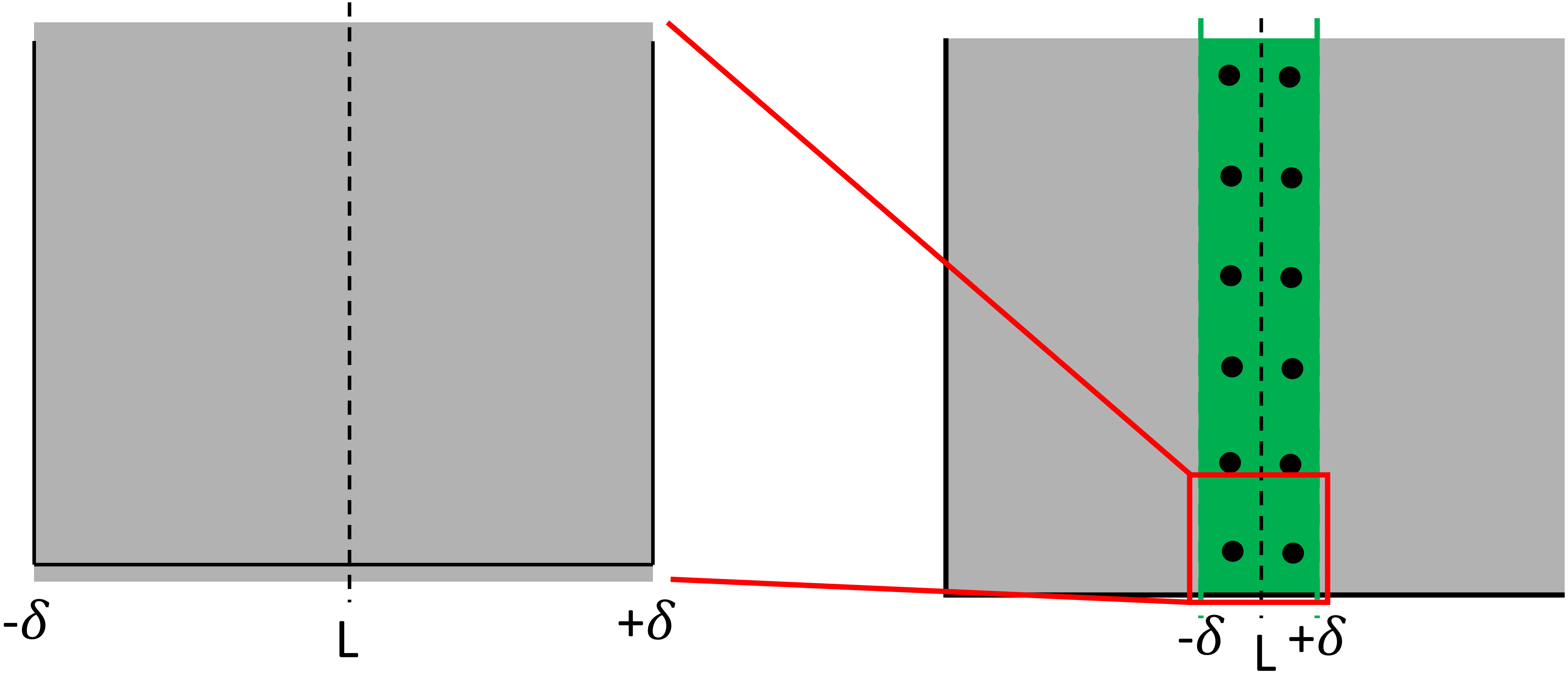


Can we just compare all left straddle points to all right straddle points?

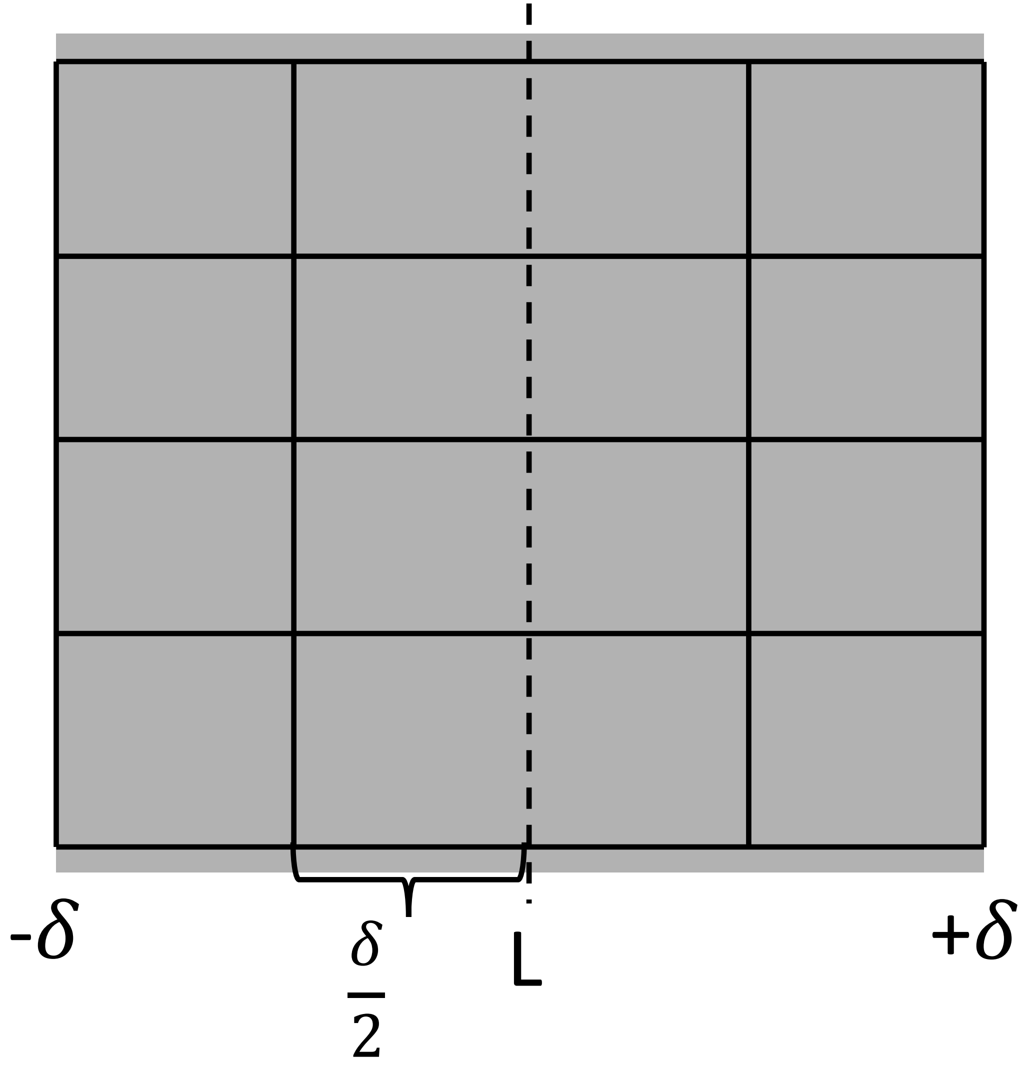
Yes, but running time could still be $O(n^2)$.

We need to reduce the number of straddle point comparisons we consider.

Closest Pair Problem – Divide and Conquer

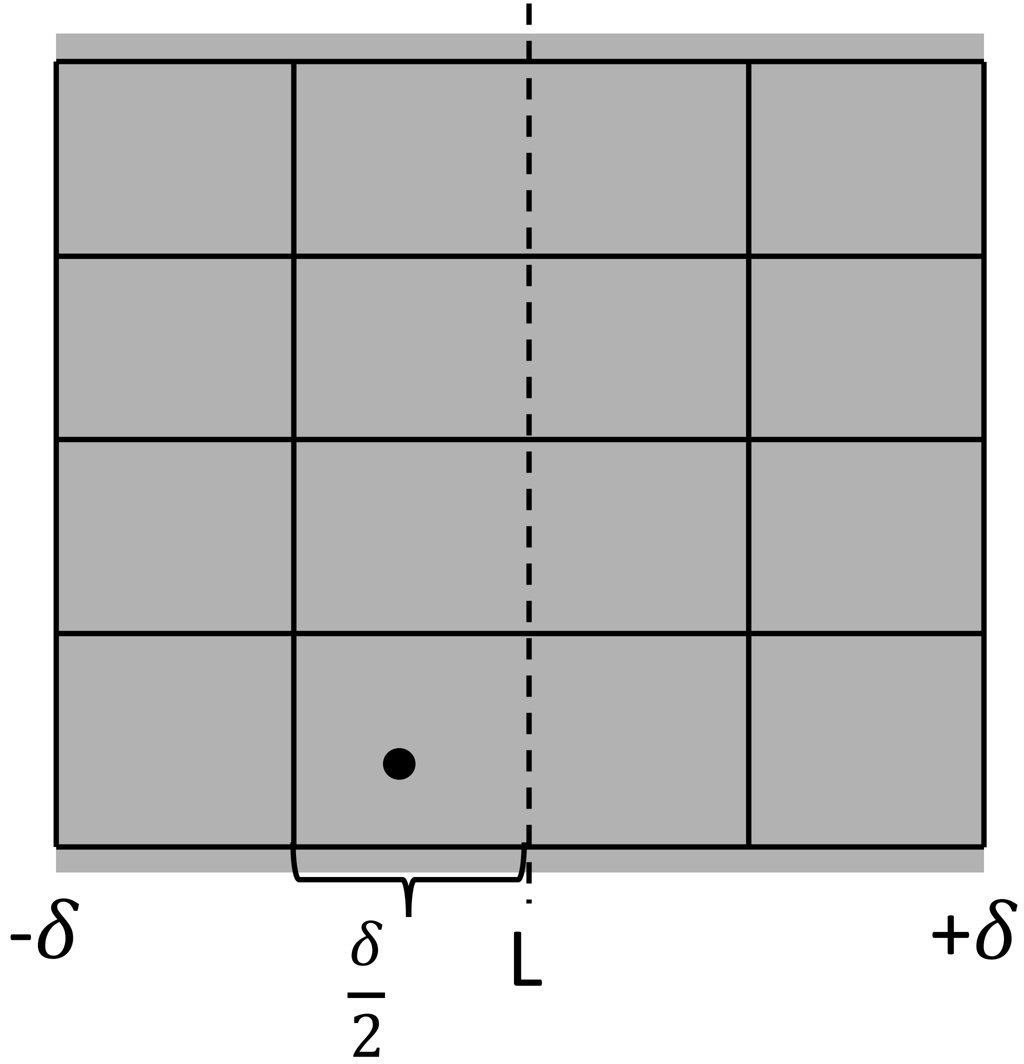


Closest Pair Problem – Divide and Conquer



Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

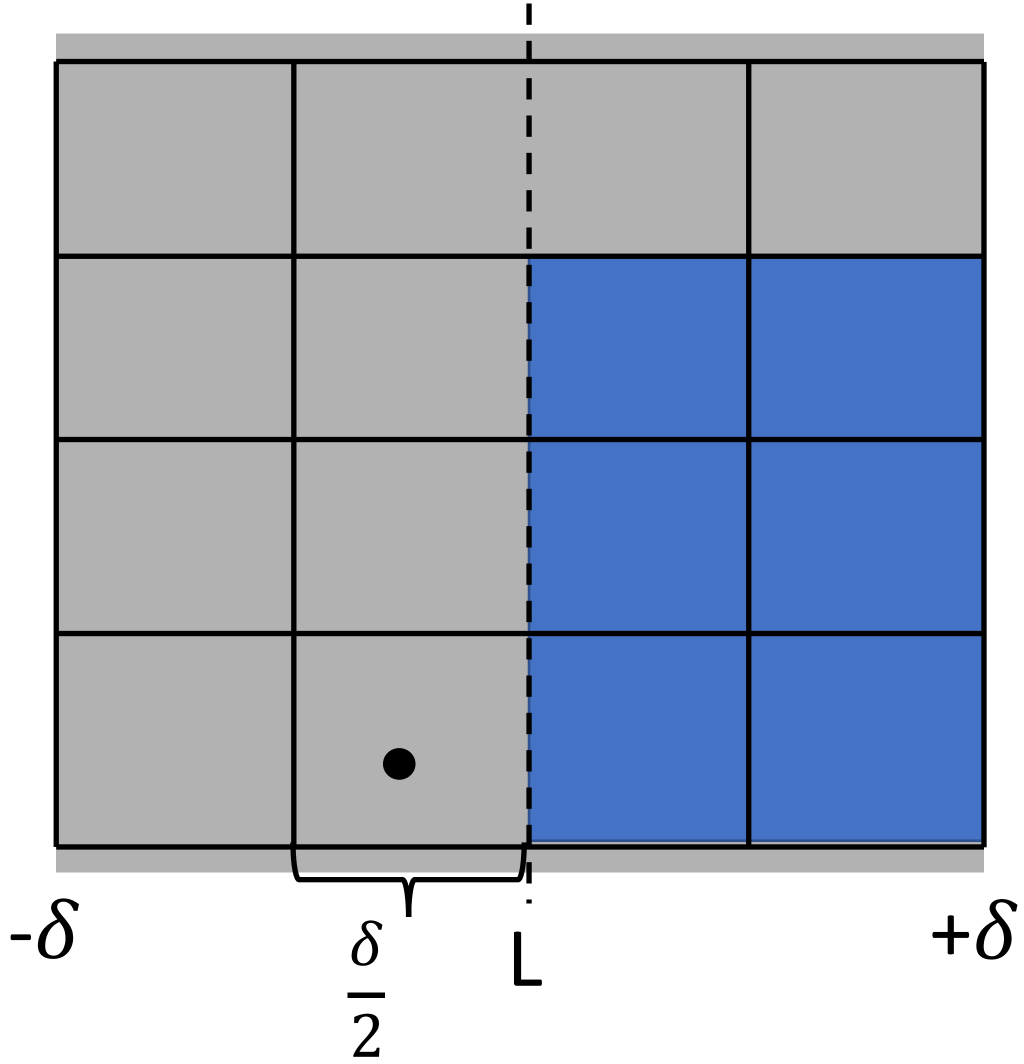
Closest Pair Problem – Divide and Conquer



Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Closest Pair Problem – Divide and Conquer

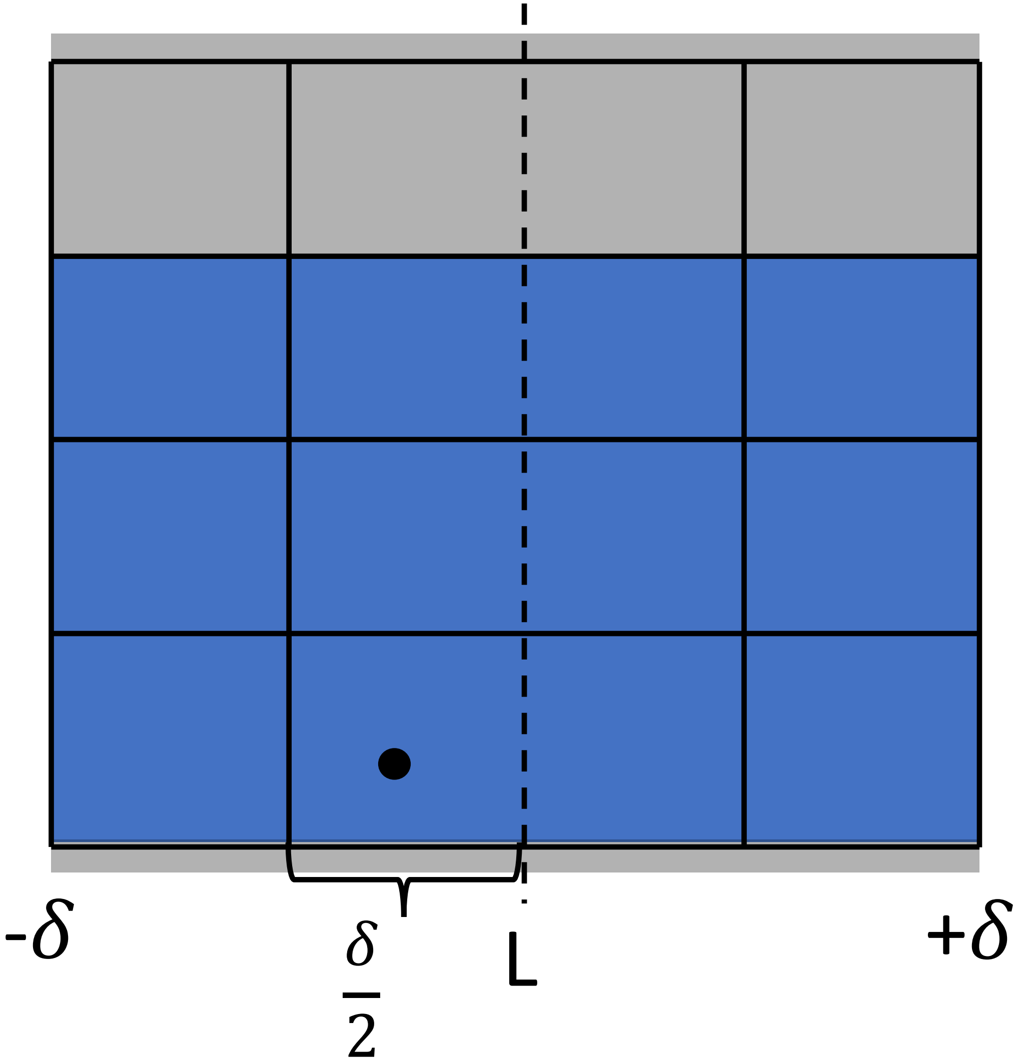


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points on other side within δ.

Closest Pair Problem – Divide and Conquer

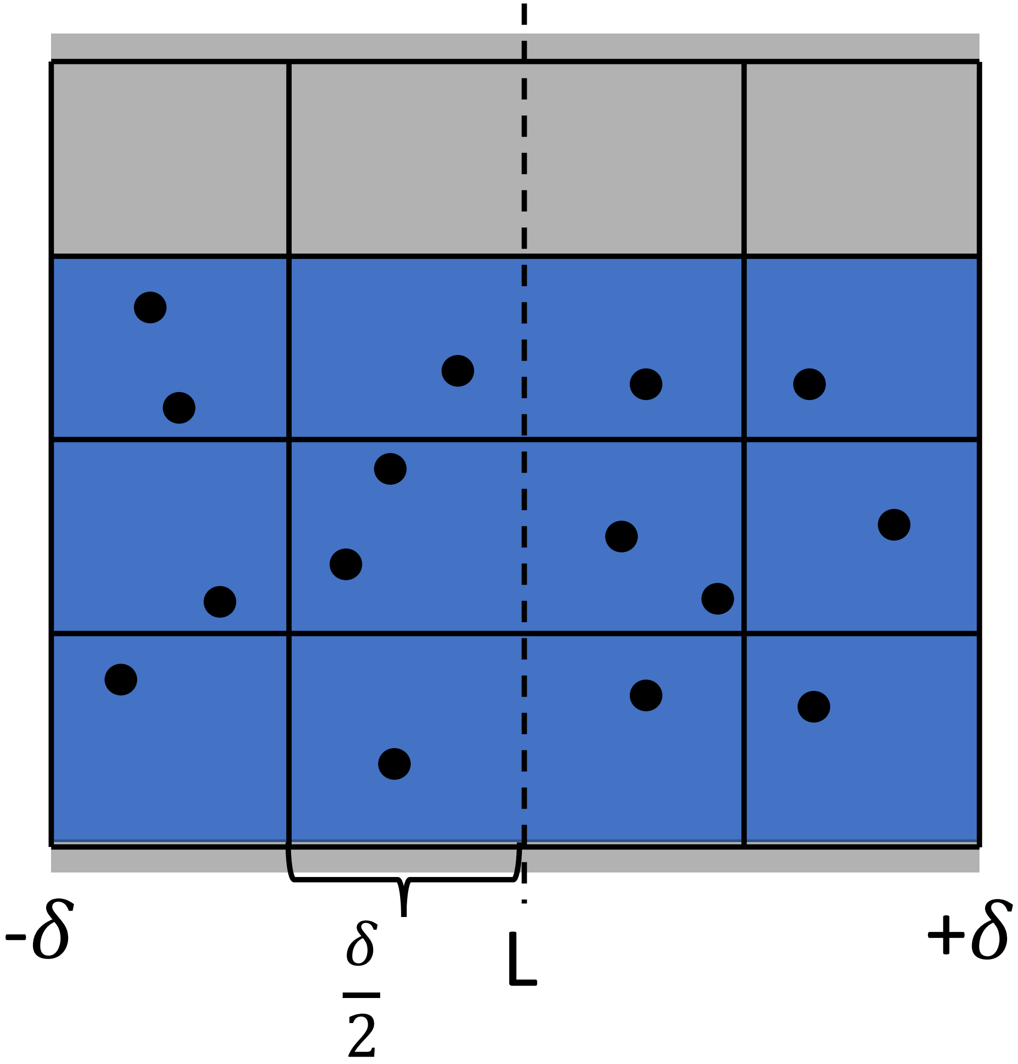


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points on other side within δ.

Closest Pair Problem – Divide and Conquer



Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

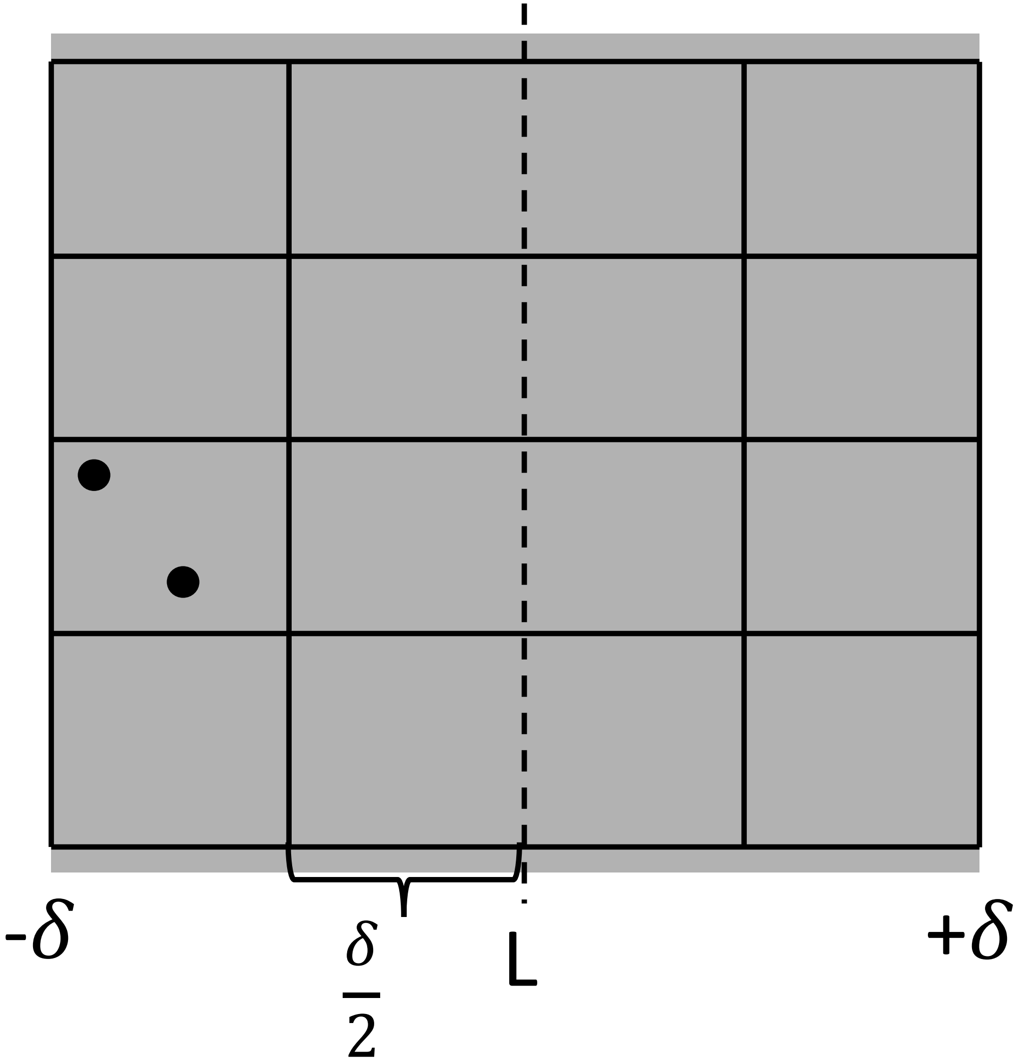
Can we focus our search to certain boxes?

Yes – we only care about points on other side within δ.

What if all of the points are in this region?

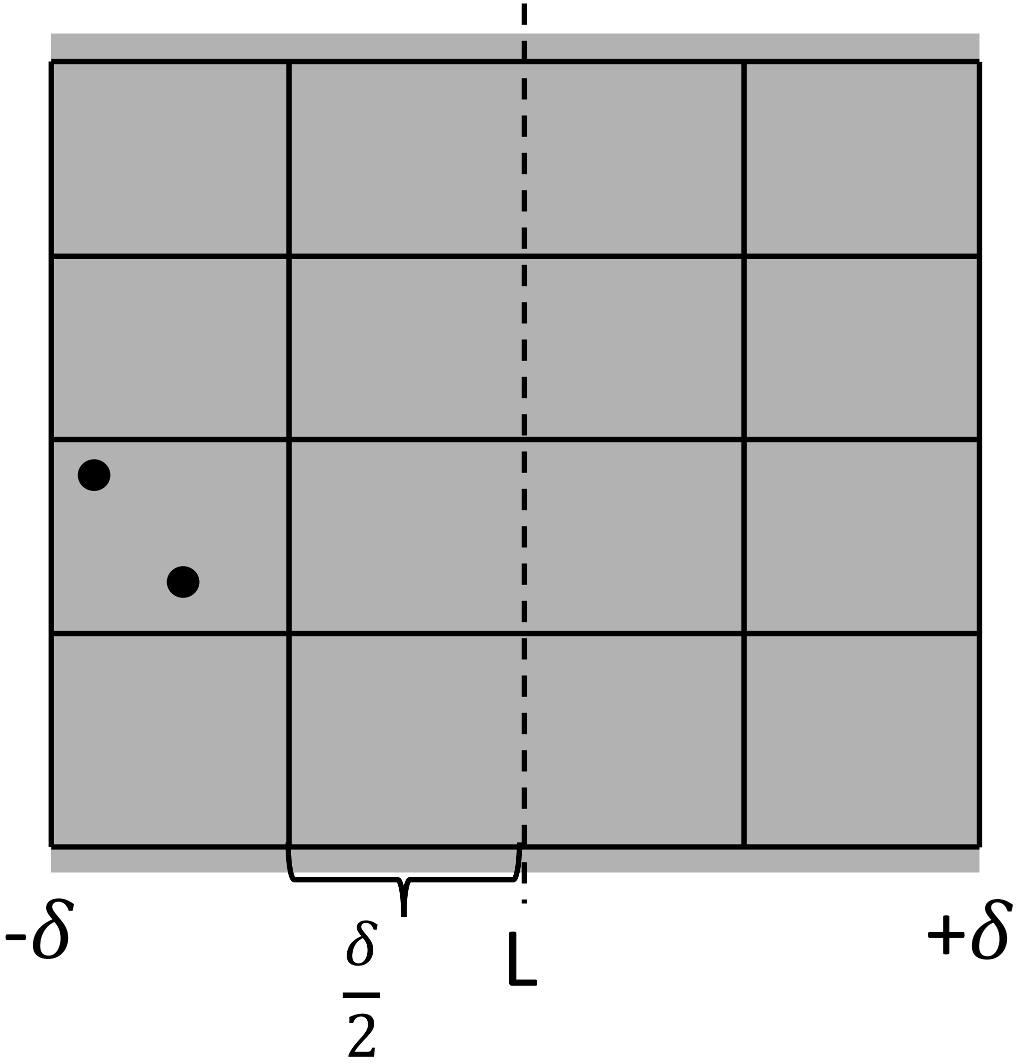
This still gives us possibly lots of points to look at.

Closest Pair Problem – Divide and Conquer



Can we have multiple points in one box?

Closest Pair Problem – Divide and Conquer

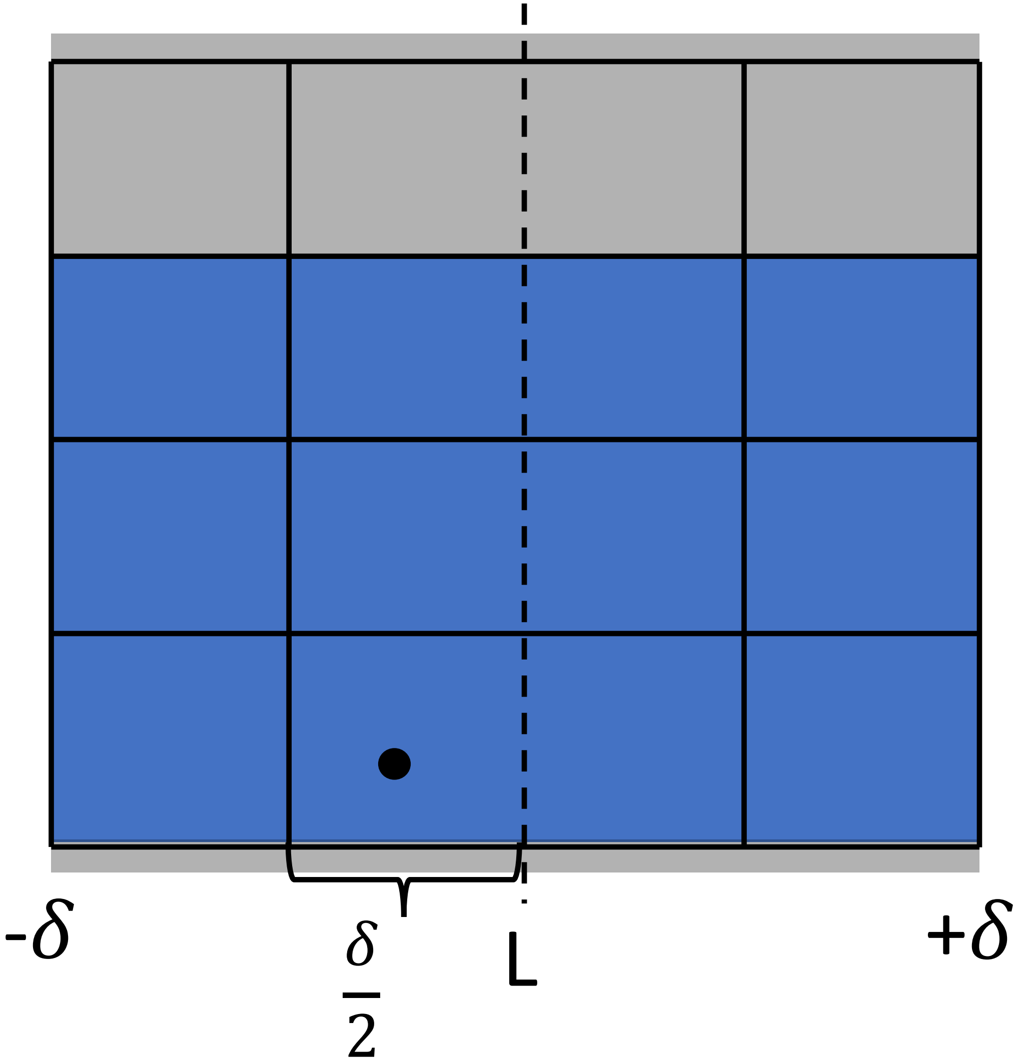


Can we have multiple points in one box?

No. δ is the smallest distance on either side of L.

$\Rightarrow$ at most one point per box.

Closest Pair Problem – Divide and Conquer

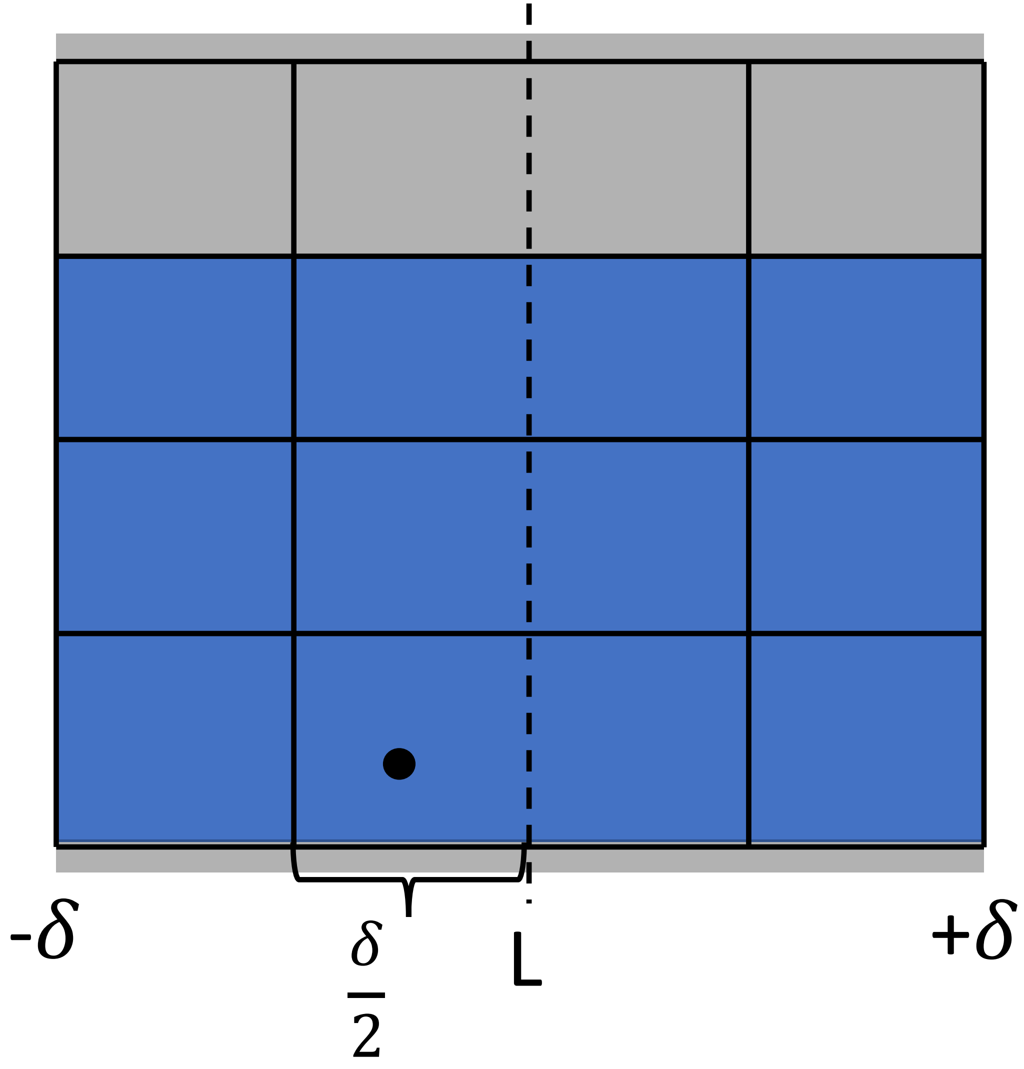


Only care about 11 boxes

\+ At most one point per box

At most 11 points to check

Closest Pair Problem – Divide and Conquer



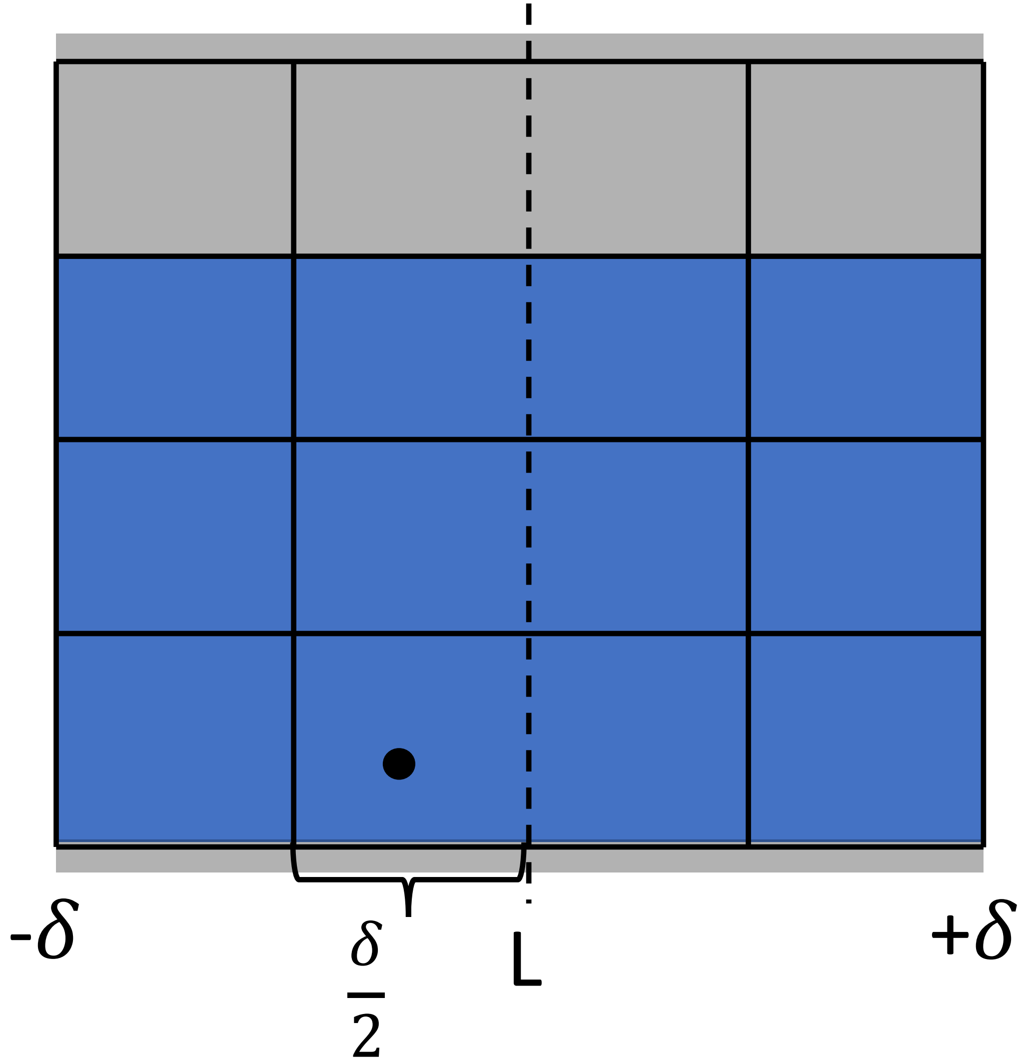
Only care about 11 boxes

$+$ At most one point per box

At most 11 points to check

1. Sort straddle points by y coordinate.

Closest Pair Problem – Divide and Conquer



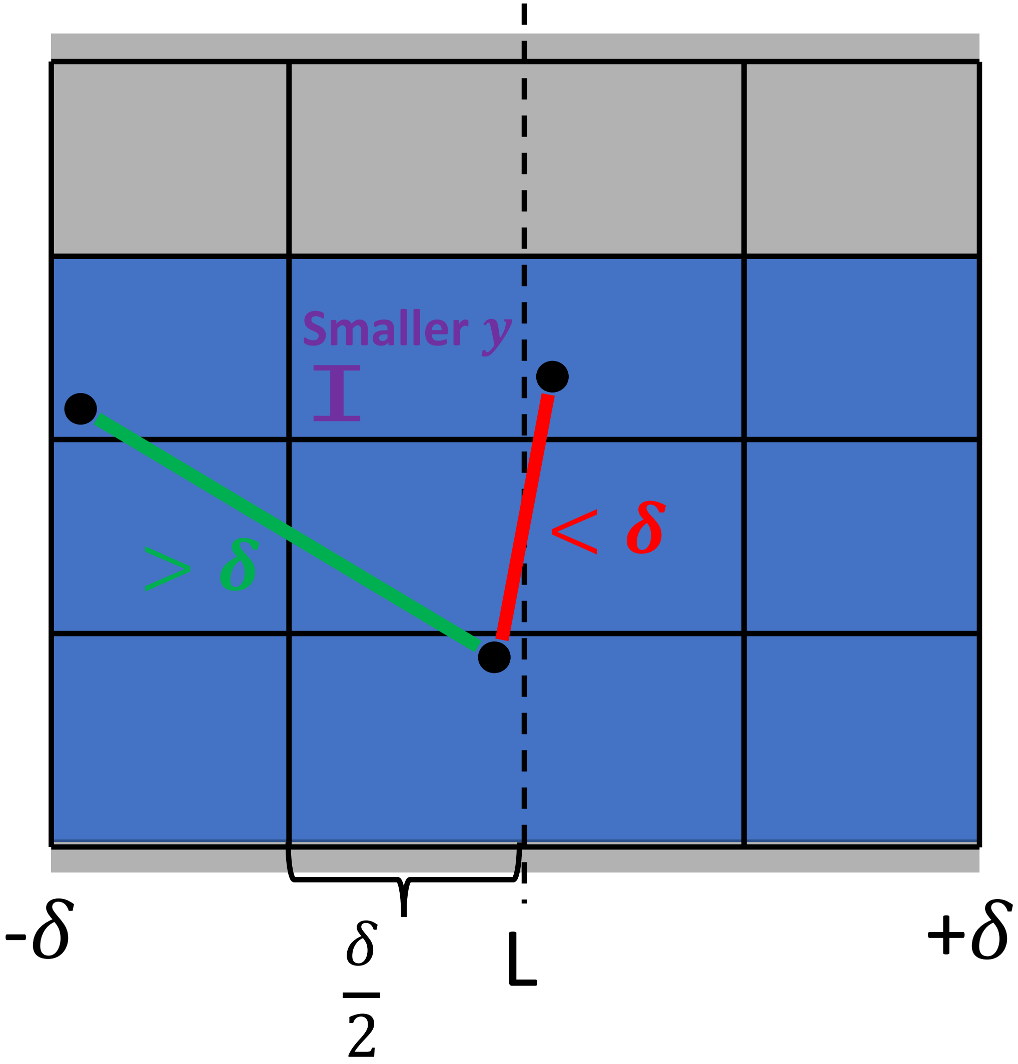
Only care about 11 boxes

\+ At most one point per box

At most 11 points to check

1. Sort straddle points by y coordinate.
2. For each point, check next 11 points to see if distance is less than δ.

Closest Pair Problem – Divide and Conquer

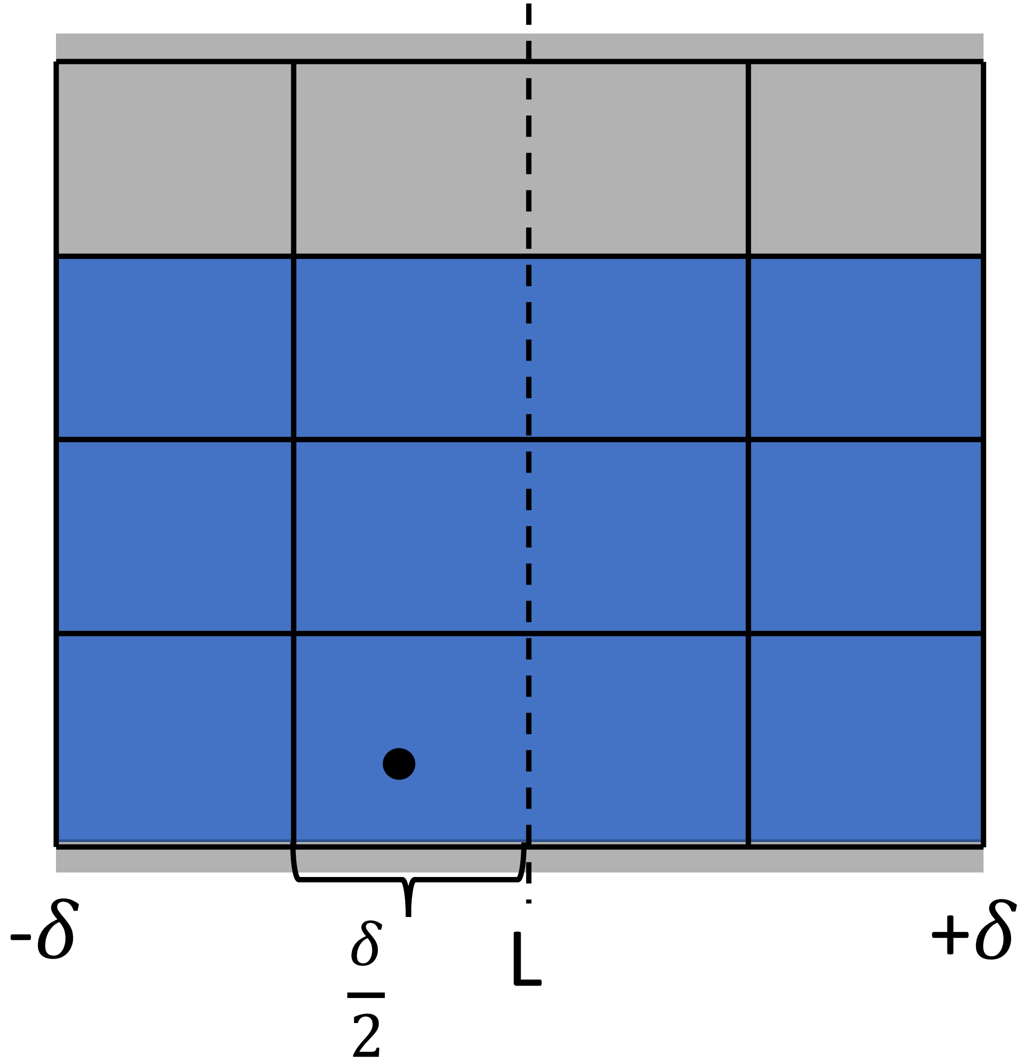


Only care about 11 boxes
$+$ At most one point per box

At most 11 points to check

1. Sort straddle points by y coordinate.
2. For each point, check next 11 points to see if distance is less than δ.

Closest Pair Problem – Divide and Conquer



Only care about 11 boxes
+ At most one point per box

At most 11 points to check

Straddle point hunting:

$$O(n^2) \longrightarrow O(n \log n)$$

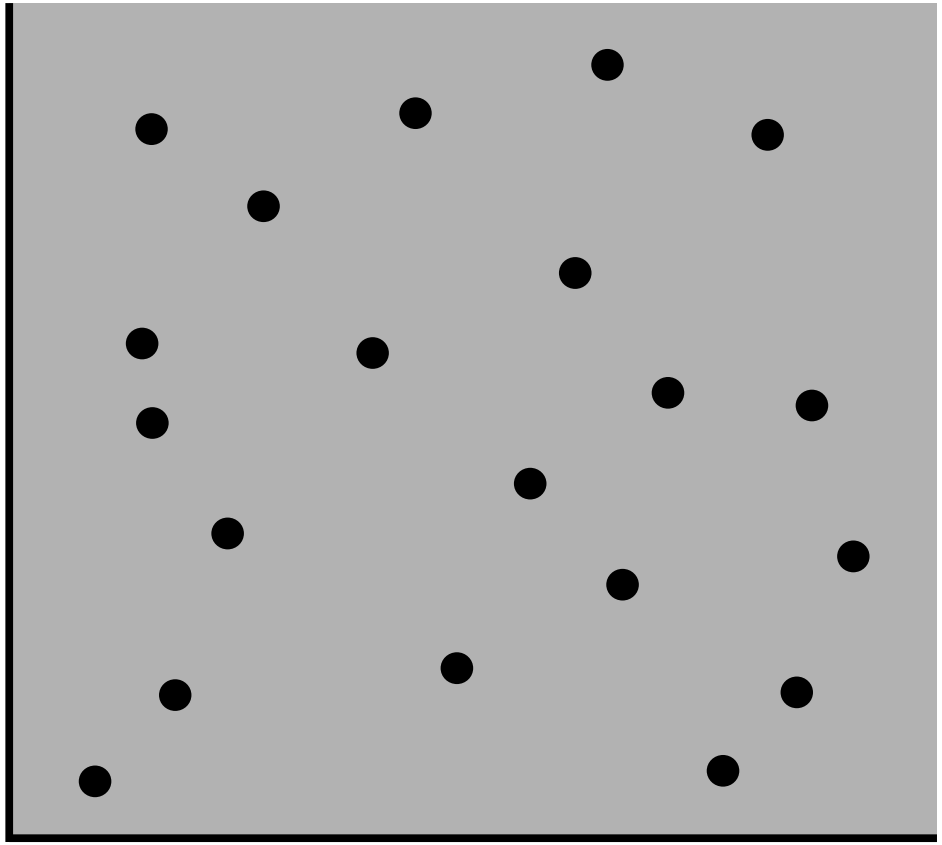
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.

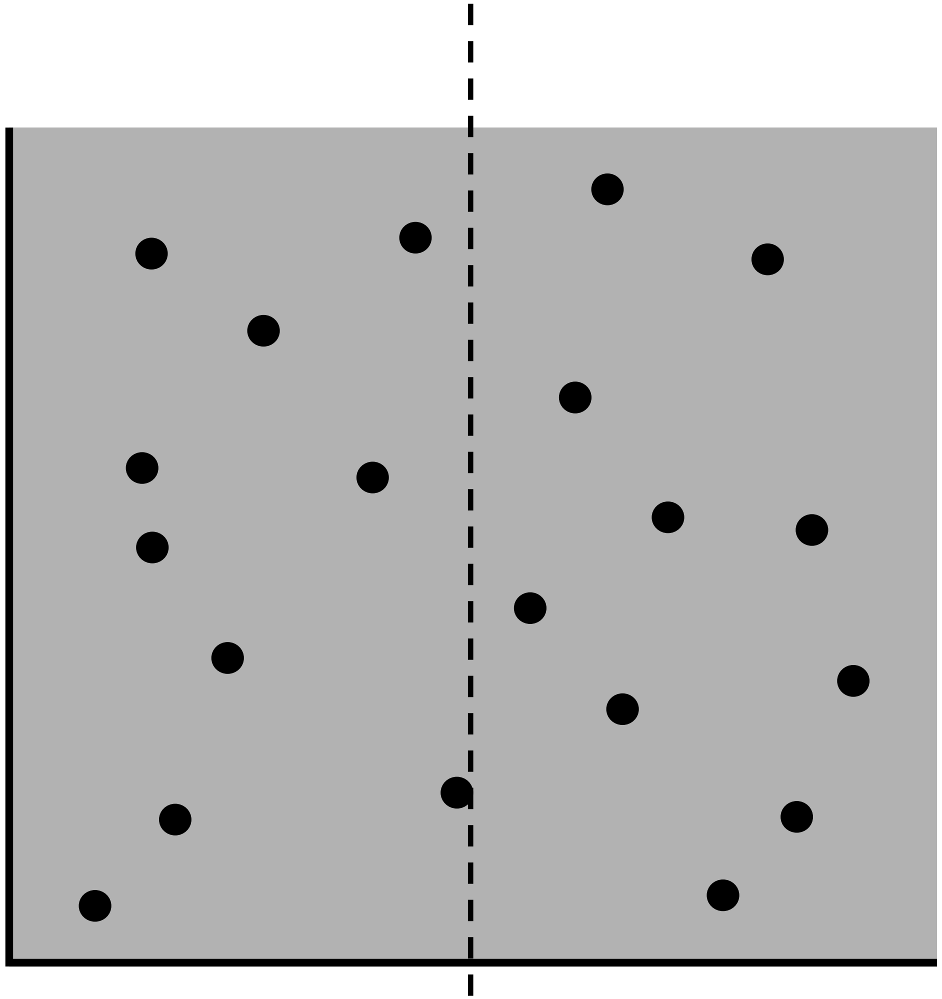
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.

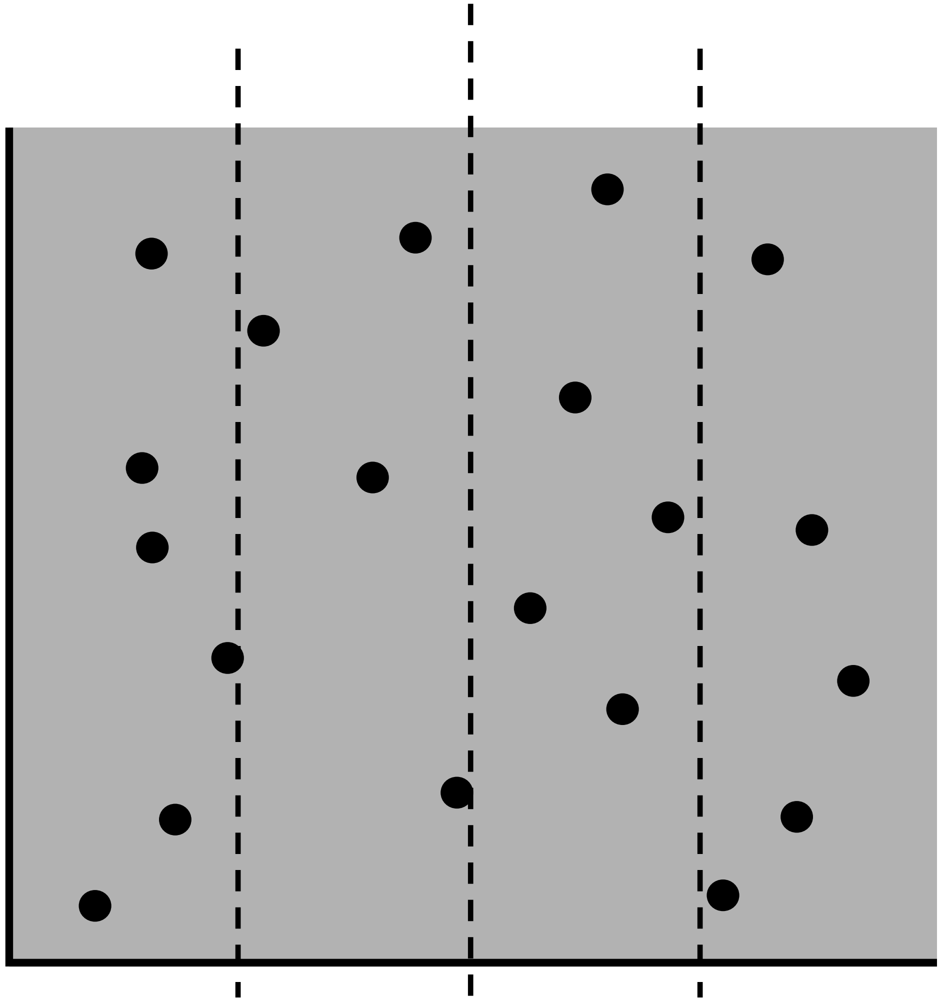
Closest Pair Problem – Divide and Conquer



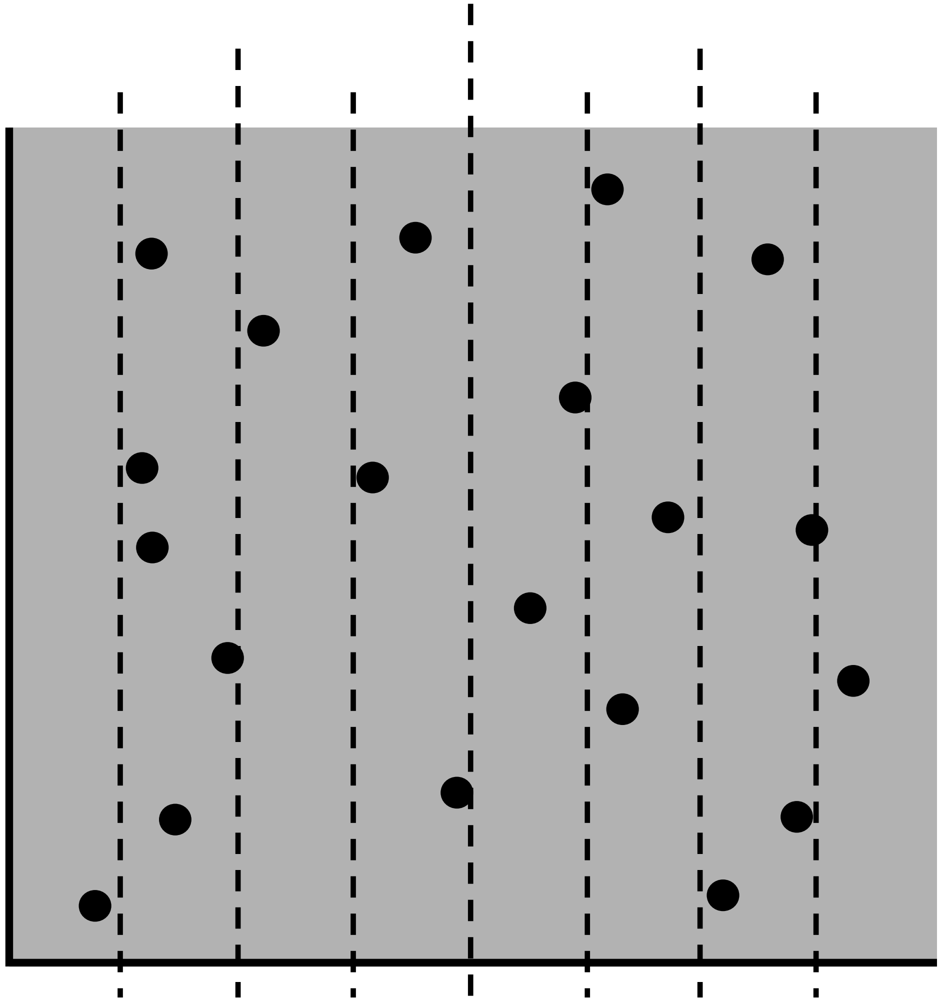
Closest Pair Problem – Divide and Conquer



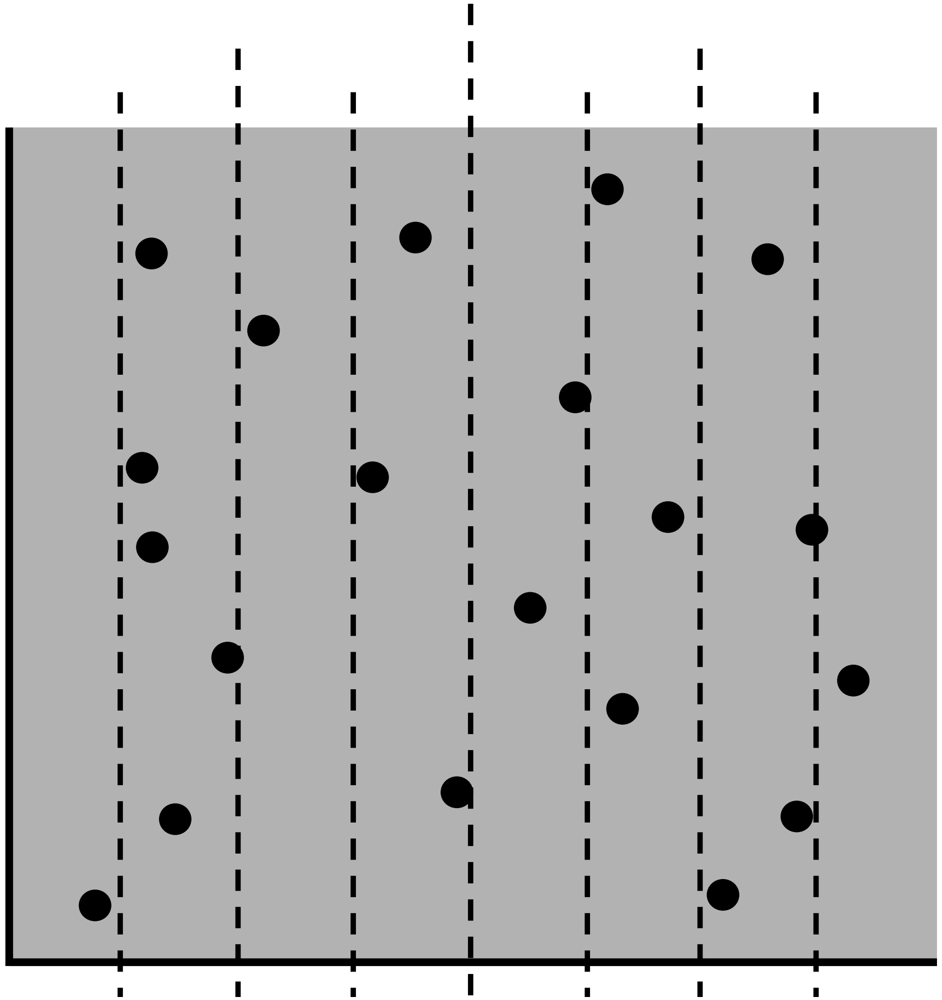
Closest Pair Problem – Divide and Conquer



Closest Pair Problem – Divide and Conquer

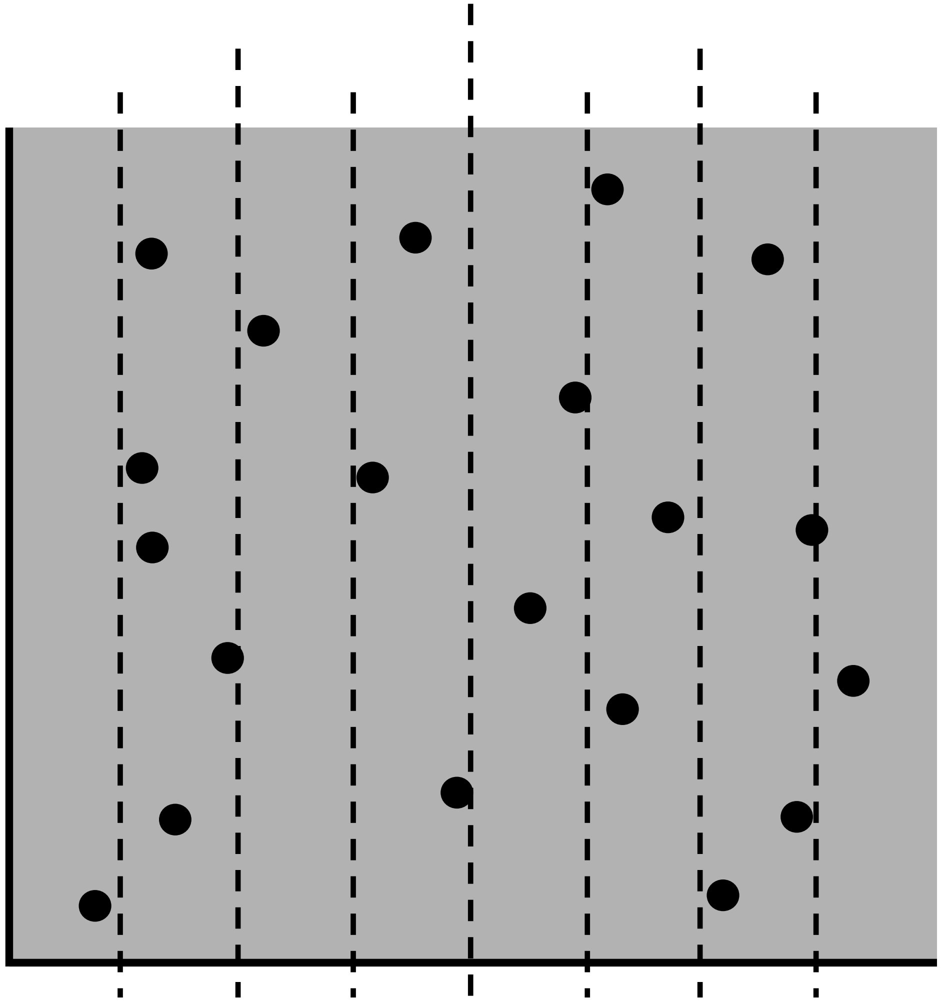


Closest Pair Problem – Divide and Conquer



When is finding d_{left} and d_{right} trivial?

Closest Pair Problem – Divide and Conquer



When is finding d_{left} and d_{right} trivial?

When there are one or two points on the left and right sides.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L.
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Running Time?

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n\log n)}$
2. Recursively determine d_{left} and d_{right}.
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L.
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n\log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n\log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $O(1)$
4. Let S be straddle points within δ of L. $O(n)$
5. Sort S by y-coordinate. $O(n \log n)$
6. Compare points in S to next 11 points and update δ. $O(n)$
7. Return δ. $O(1)$

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret

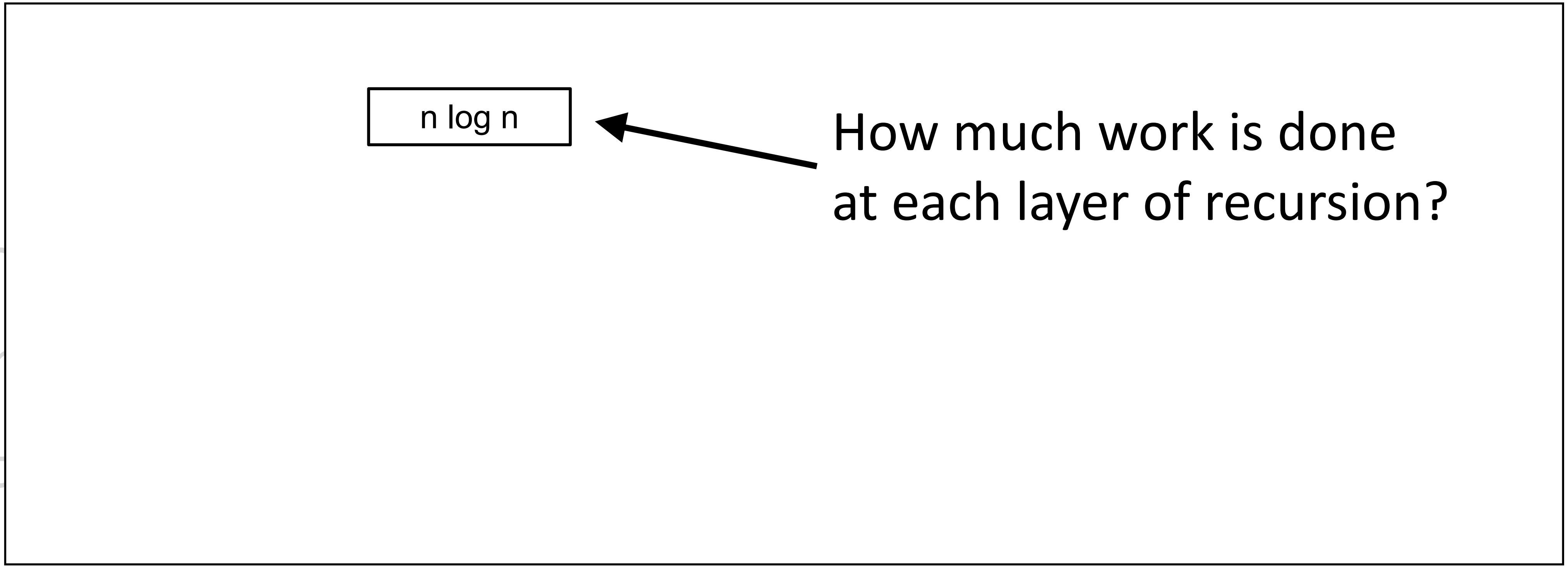
How much work is done
at each layer of recursion?

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

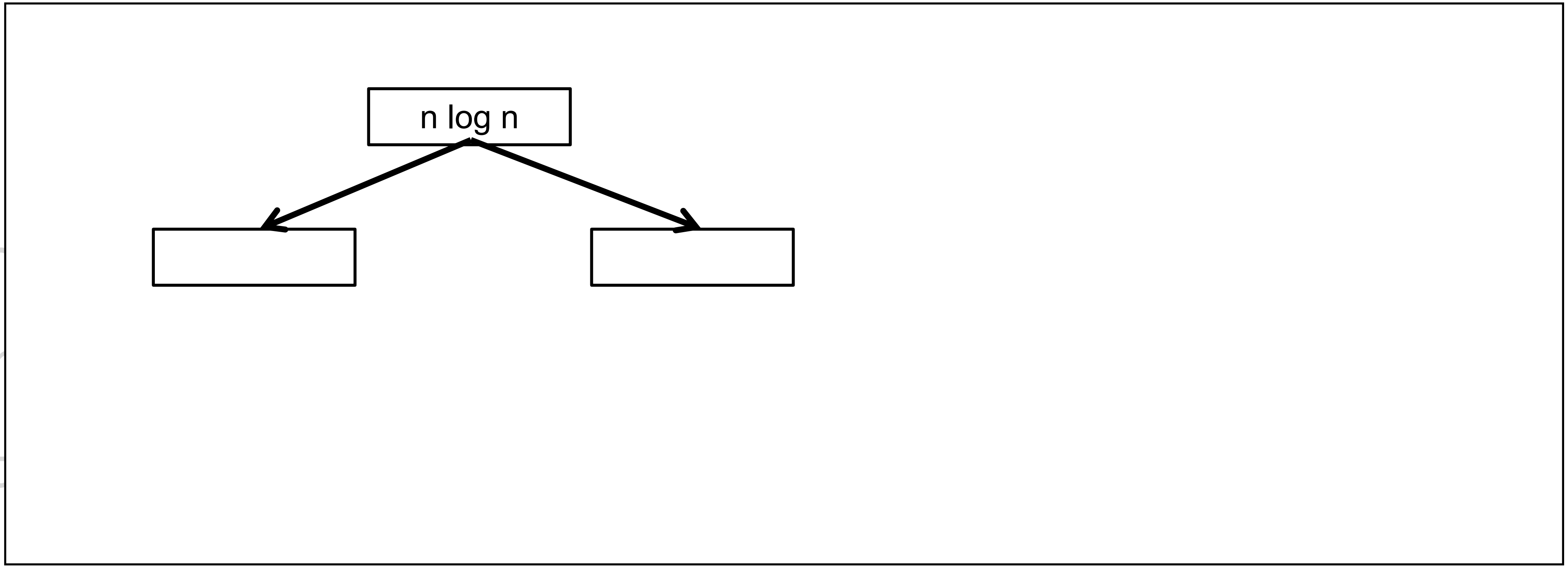
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



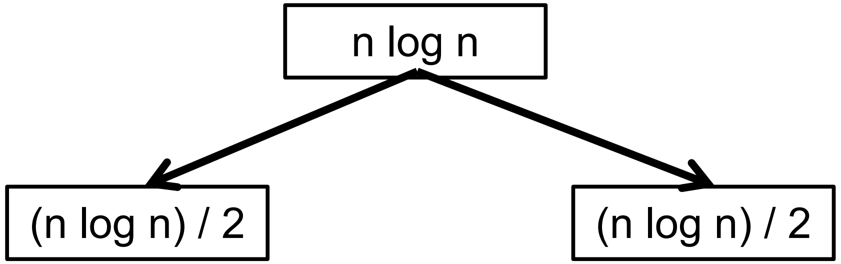
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



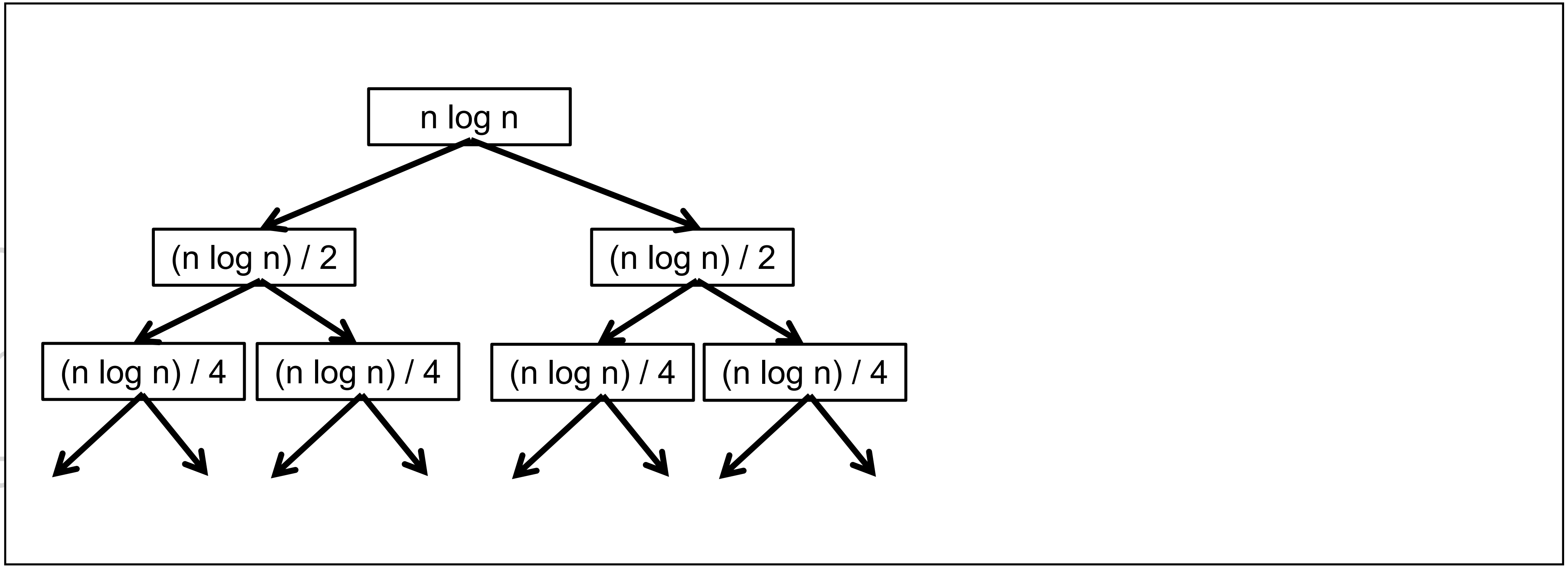
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



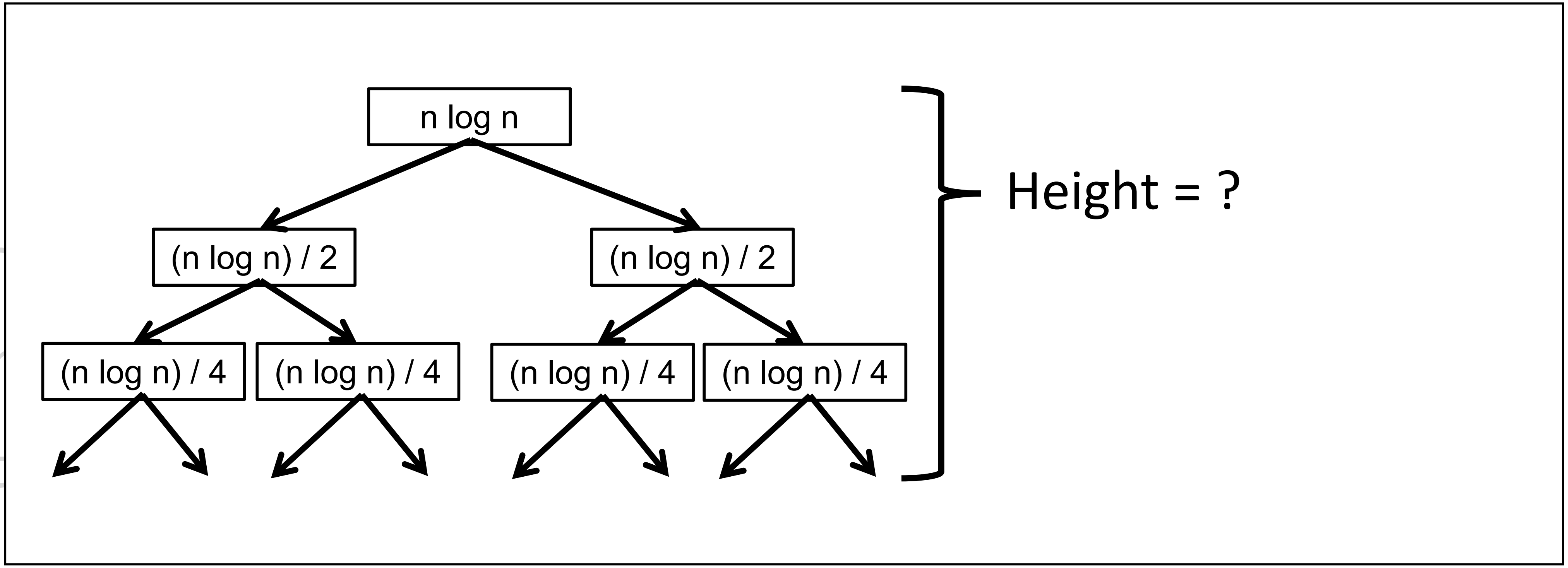
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



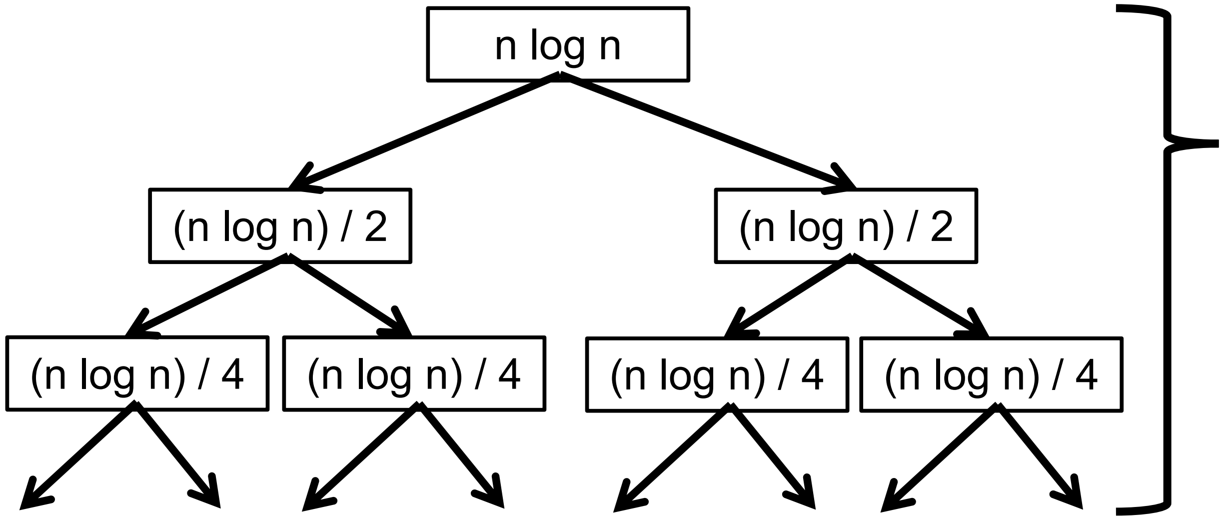
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret

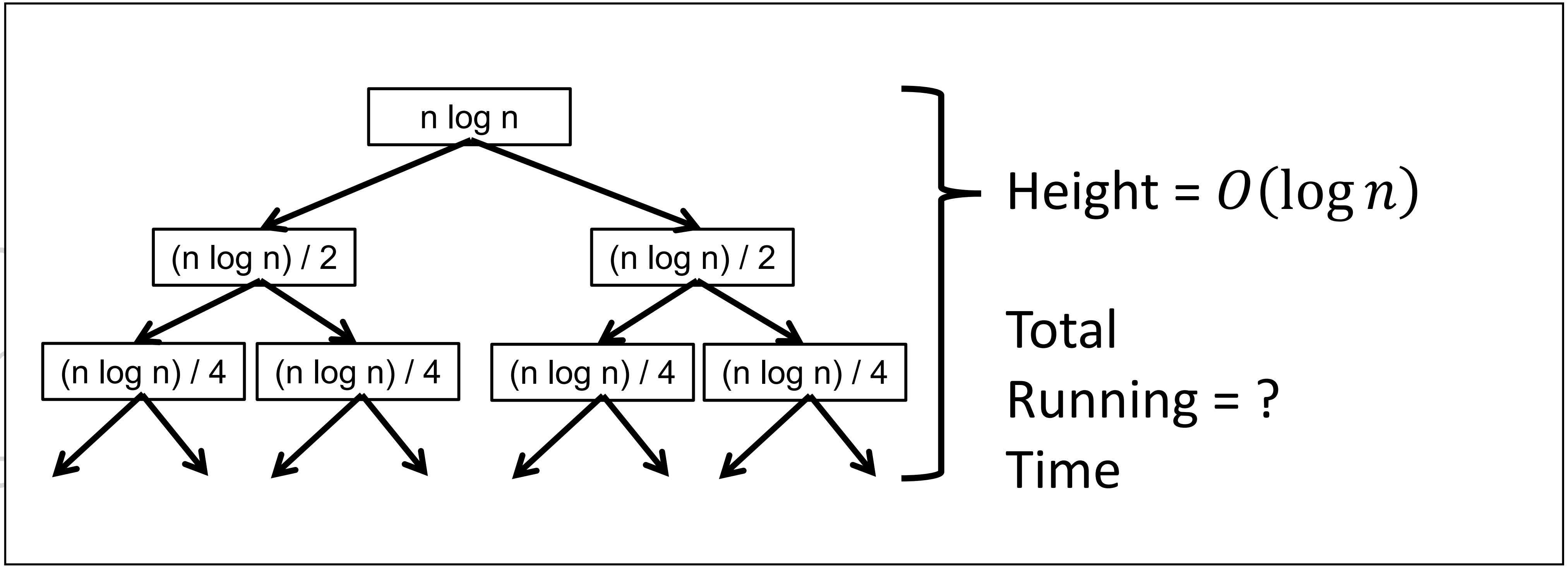


Height = ?

Binary tree, divide by 2 each time?

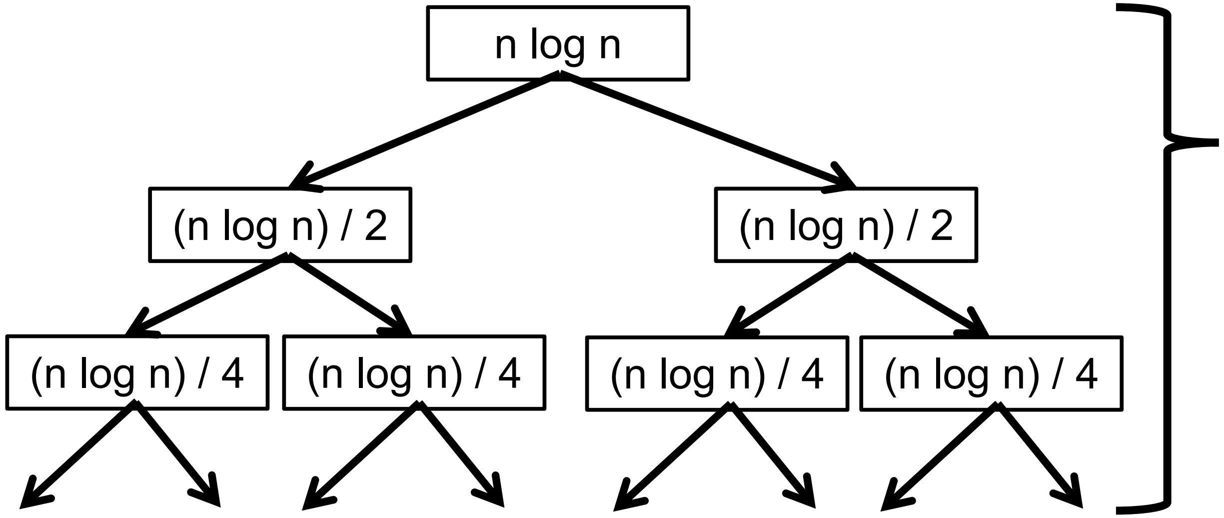
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret

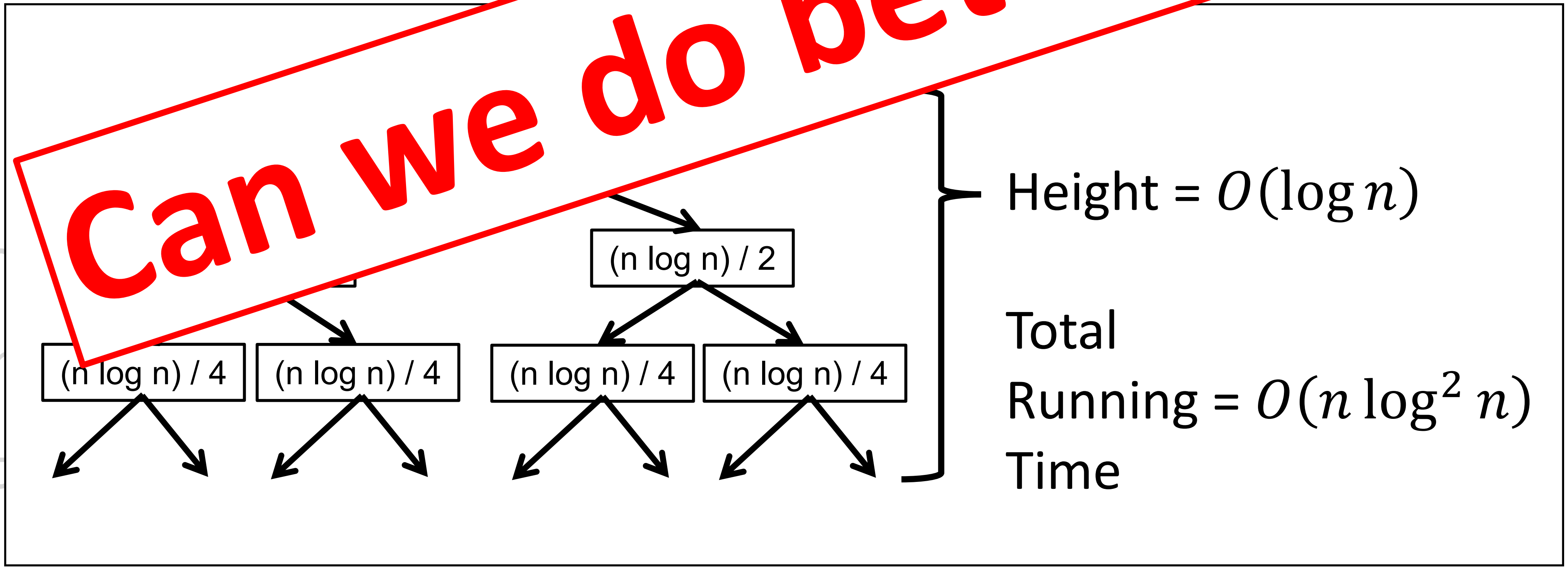


Height = $O(\log n)$

Total Running Time = $O(n \log^2 n)$

Closest Pair Problem – Algorithm

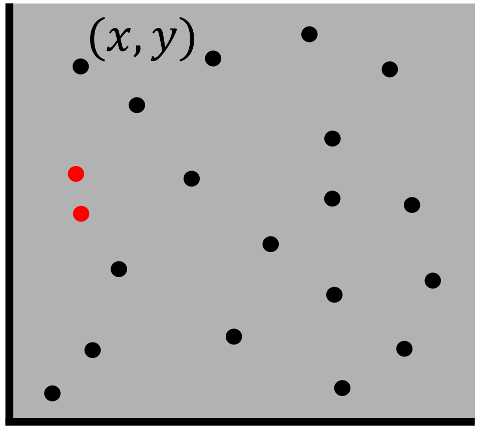
1. Sort points by x-coordinate and make L. $O($
2. Recursively determine d_{left} and
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Can we do better?

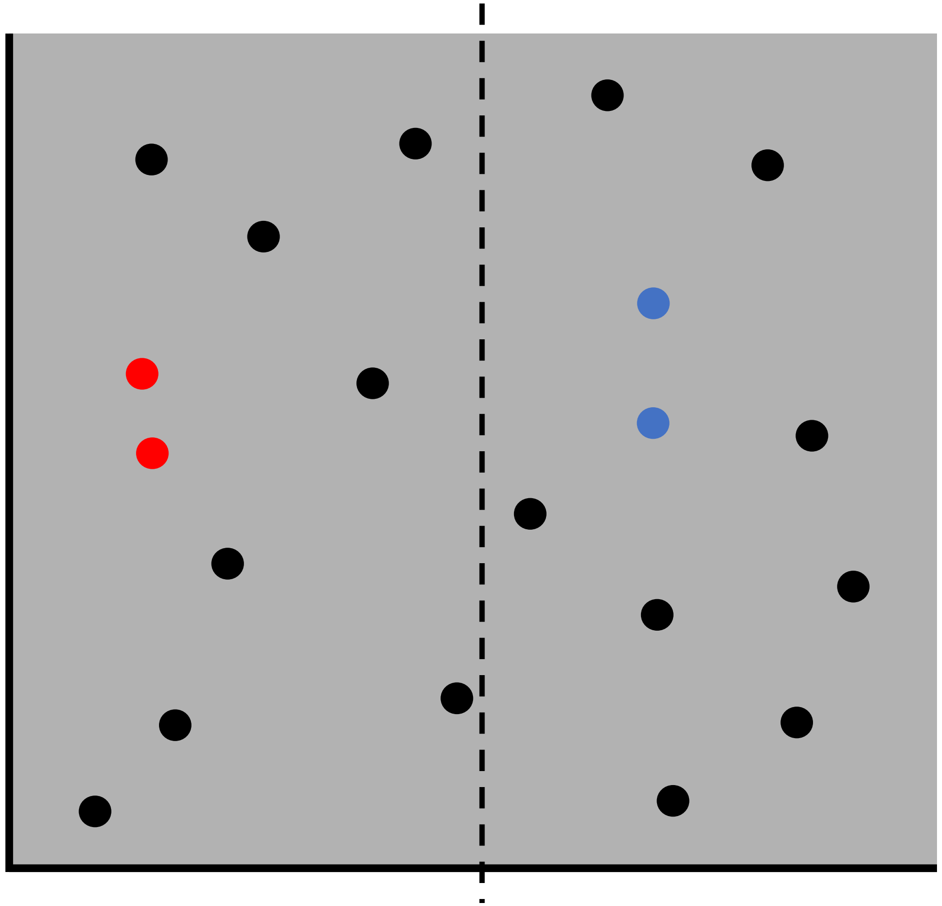
Closest Pair of Points
CSCI 232

Closest Pair Problem



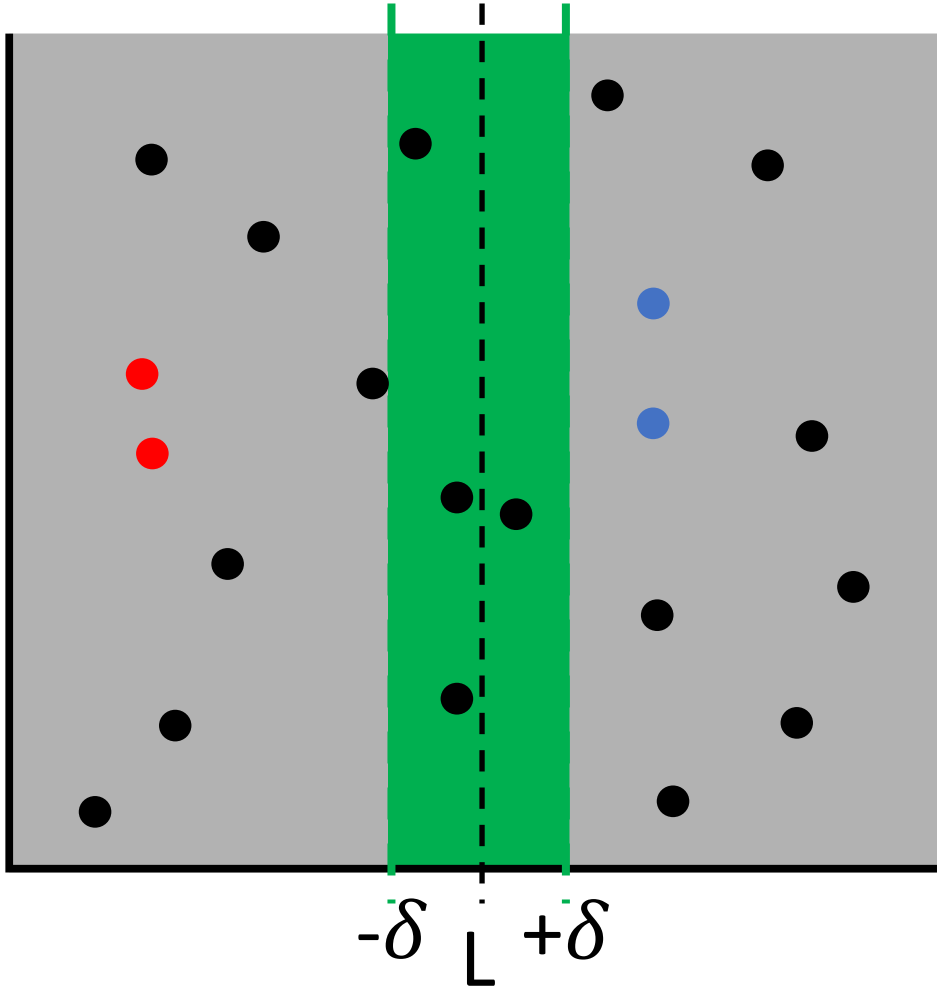
Given n points, find a pair of points with the smallest distance between them.

Closest Pair Problem – Divide and Conquer



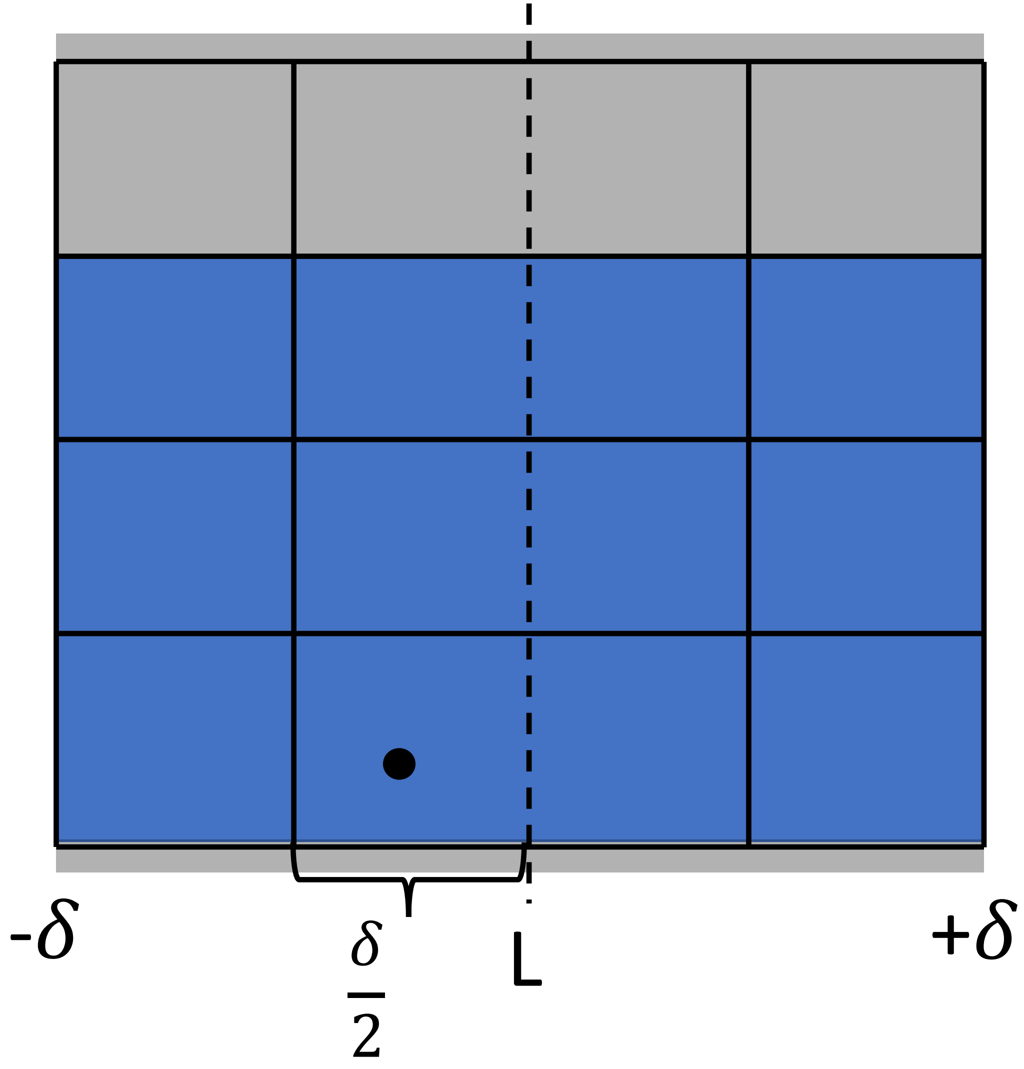
1. Recursively find closest on left and right.

Closest Pair Problem – Divide and Conquer



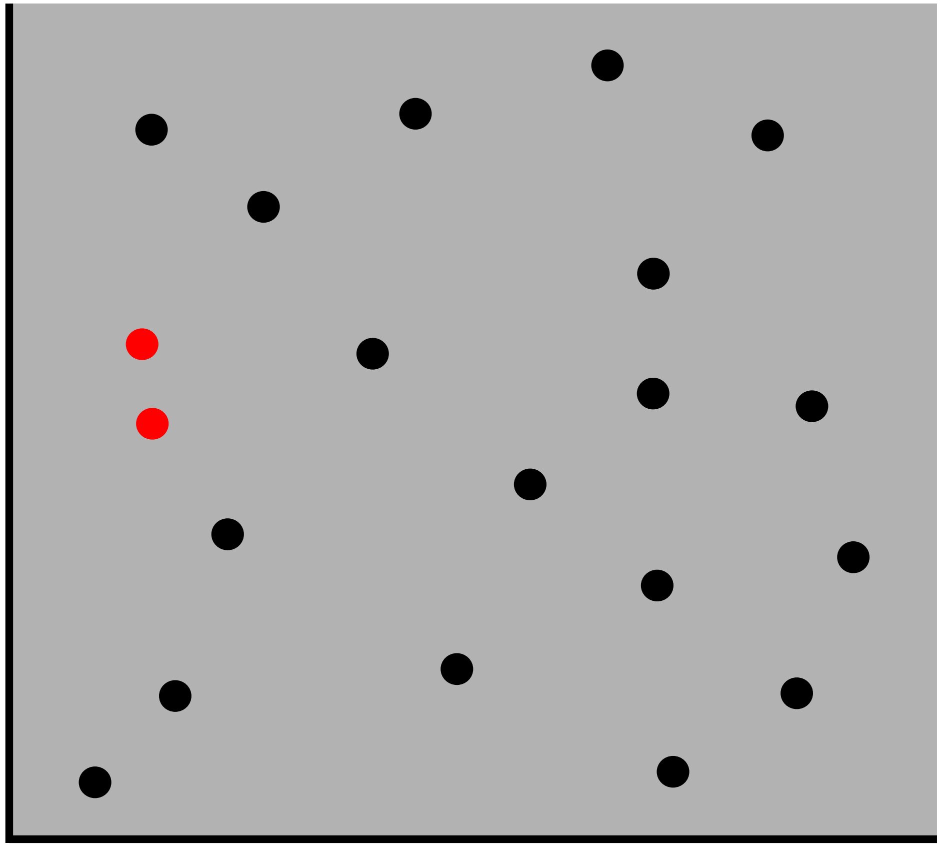
1. Recursively find closest on left and right.
2. Find closest overall by considering "straddle points":

Closest Pair Problem – Divide and Conquer



1. Recursively find closest on left and right.
2. Find closest overall by considering "straddle points":
   2.1 Compare each straddle point to the next 11 points in sorted (y-coordinate) order.

Closest Pair Problem – Divide and Conquer



1. Recursively find closest on left and right.
2. Find closest overall by considering "straddle points":
   2.1 Compare each straddle point to the next 11 points in sorted (y-coordinate) order.
3. Return closest.

Closest Pair Problem – Algorithm

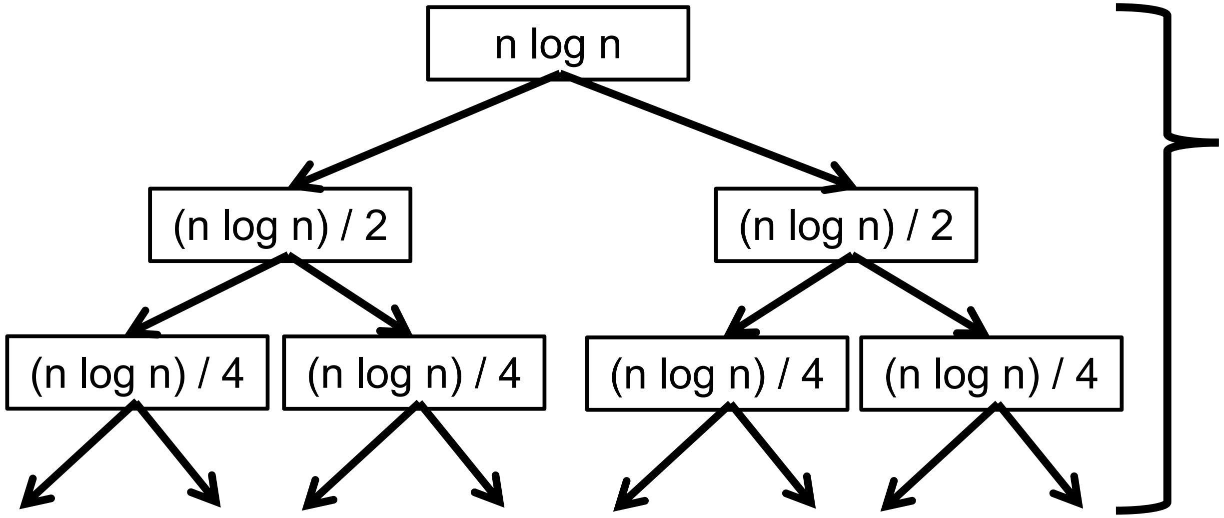
1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $O(1)$
4. Let S be straddle points within δ of L. $O(n)$
5. Sort S by y-coordinate. $O(n \log n)$
6. Compare points in S to next 11 points and update δ. $O(n)$
7. Return δ. $O(1)$

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret

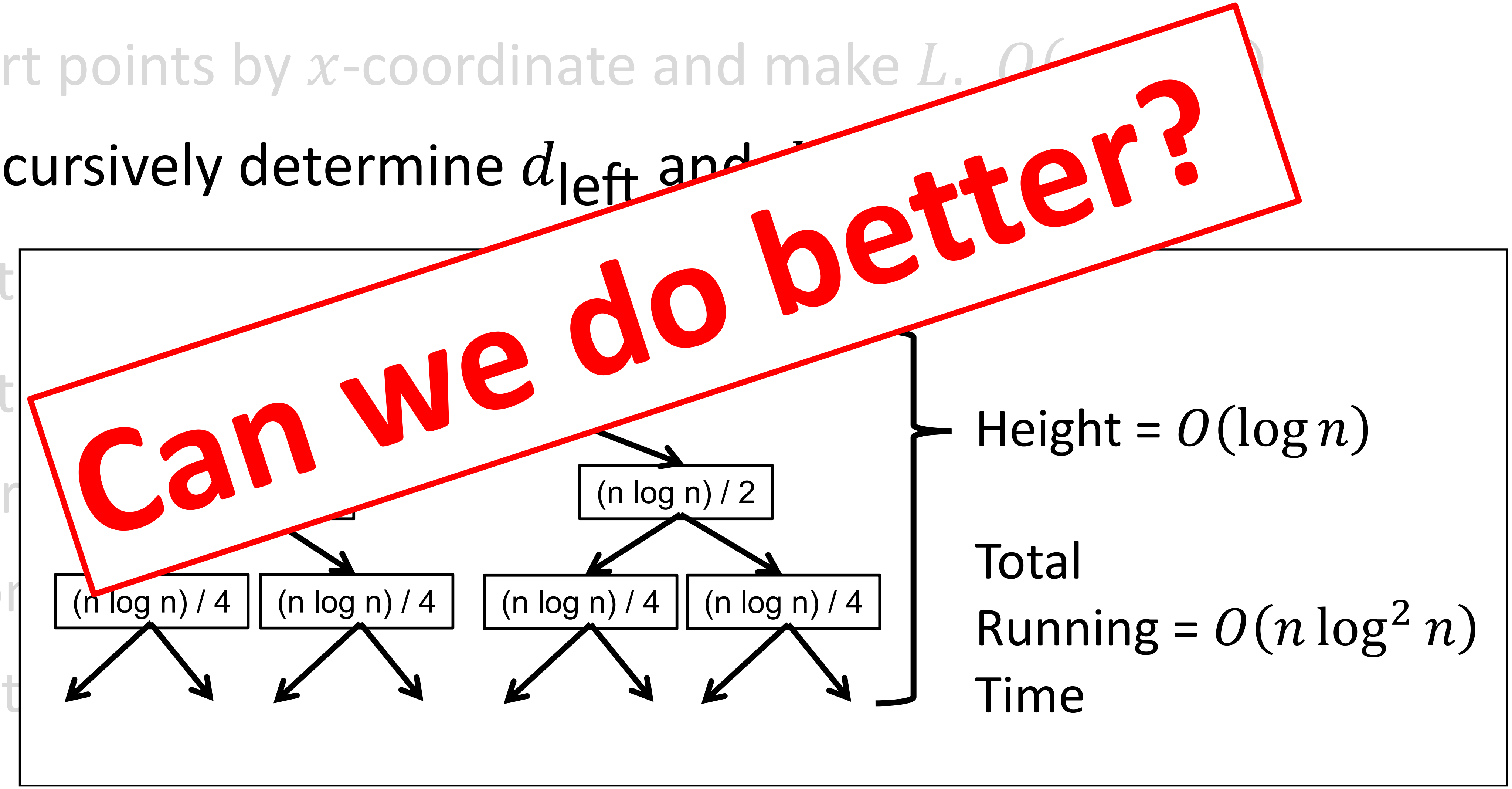


Height = $O(\log n)$

Total Running Time = $O(n \log^2 n)$

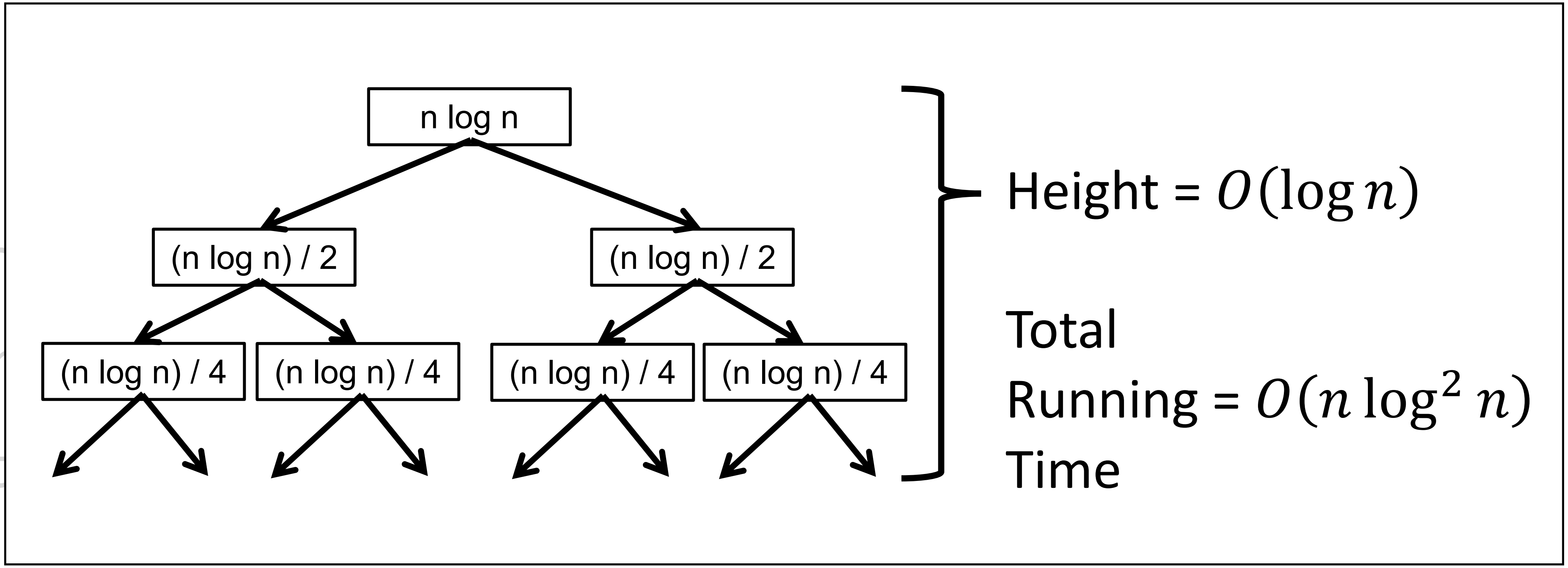
Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $O($
2. Recursively determine d_{left} and
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Closest Pair Problem – Algorithm

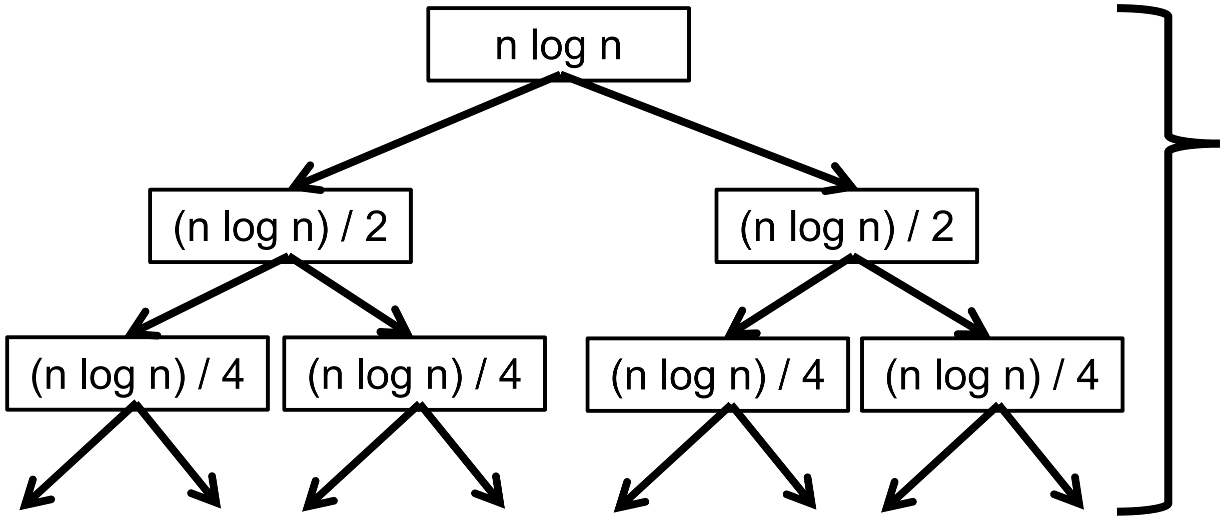
1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Closest Pair Problem – Algorithm

What is driving our complexity?

1. S
2. Recursively determine d_{left} and d_{right}: TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Height = $O(\log n)$

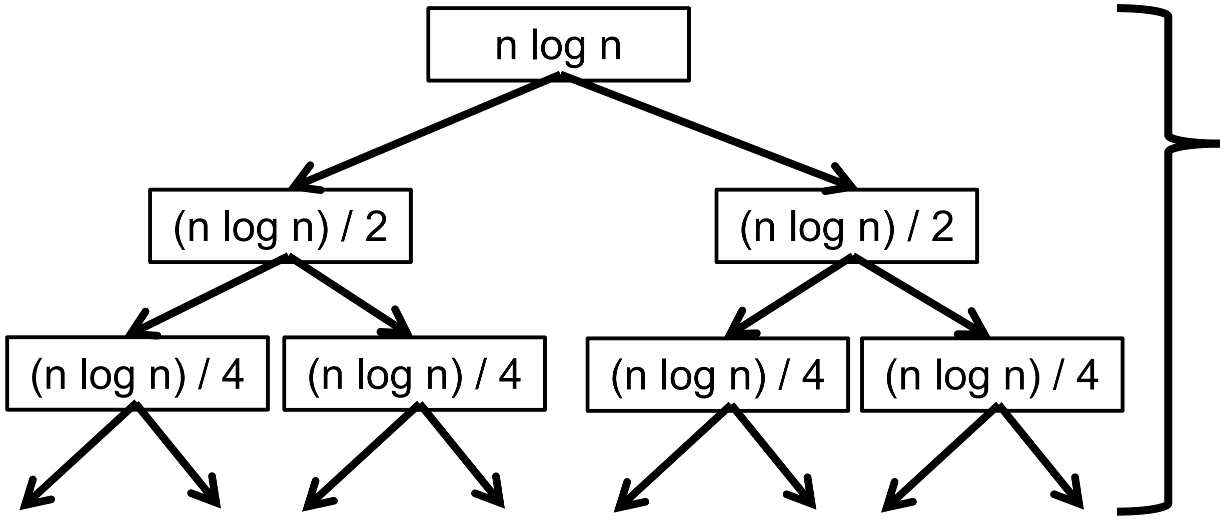
Total Running Time = $O(n \log^2 n)$

Closest Pair Problem – Algorithm

What is driving our complexity?

The work being done ***<u>in</u>*** each recursive call.

1. S
2. Recursively determine d_{left} and d_{right}: TBD
3. Let
4. Let
5. Sor
6. Co
7. Ret



Height = $O(\log n)$

Total Running Time = $O(n \log^2 n)$

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

What is driving our complexity in each recursive call?

1.
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

What is driving our complexity in each recursive call?

Sorting S by y-coordinate.

1.
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

How can we reduce our complexity?

Closest Pair Problem – Algorithm

What is driving our complexity in each recursive call?

Sorting S by y-coordinate.

1.
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

How can we reduce our complexity?

Sort once, before the recursive calls.

Closest Pair Problem – Algorithm

1. Sort points by x-coordinate and make L. $\boldsymbol{O(n\log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n\log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. **Sort by x-coordinate (X) and y-coordinate (Y).**
1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. **Sort by x-coordinate (X) and y-coordinate (Y).** $\boldsymbol{O(n \log n)}$
1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. **Make L and split X and Y.**
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. **Make L and split X and Y.**
2. Recursively determine d
3. Let $\delta = \min(d_{\text{left}}, d_{\text{righ}}$
4. Let S be straddle points
5. Sort S by y-coordinate.
6. Compare points in S to r $\boldsymbol{n)}$
7. Return δ. $\boldsymbol{O(1)}$

```
L = X[ceiling(X.length / 2 - 1)].x

for each (x,y) in X:
   if (x <= L):
      X_left.add((x,y))
   else:
      X_right.add((x,y))

for each (x,y) in Y:
   if (x <= L):
      Y_left.add((x,y))
   else:
      Y_right.add((x,y))
```

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. **Make L and split X and Y.** $\boldsymbol{O(n)}$
2. Recursively determine d
3. Let $\delta = \min(d_{\text{left}}, d_{\text{righ}}$
4. Let S be straddle points
5. Sort S by y-coordinate.
6. Compare points in S to r $\boldsymbol{n)}$
7. Return δ. $\boldsymbol{O(1)}$

```
L = X[ceiling(X.length / 2 - 1)].x

for each (x,y) in X:
   if (x <= L):
      X_left.add((x,y))
   else:
      X_right.add((x,y))

for each (x,y) in Y:
   if (x <= L):
      Y_left.add((x,y))
   else:
      Y_right.add((x,y))
```

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. **Recursively determine d_{left} and d_{right}.** **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. **Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$.** $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. **Let S be straddle points within δ of L.** $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n\log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. **Let $\boldsymbol{S}$ be straddle points within $\boldsymbol{\delta}$ of $\boldsymbol{L}$.** $\boldsymbol{O(n)}$
5. Sort S by y-coord
6. Compare points i
7. Return δ. $\boldsymbol{O(1)}$

```
for each (x,y) in Y:
    if (x >= L - δ && x <= L + δ):
        S.add((x,y))
```

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. **Sort S by y-coordinate.** $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. ~~**Sort S by y-coordinate. $\boldsymbol{O(n \log n)}$**~~
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. **Compare points in S to next 11 points and update δ.** $\boldsymbol{O(n)}$
6. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

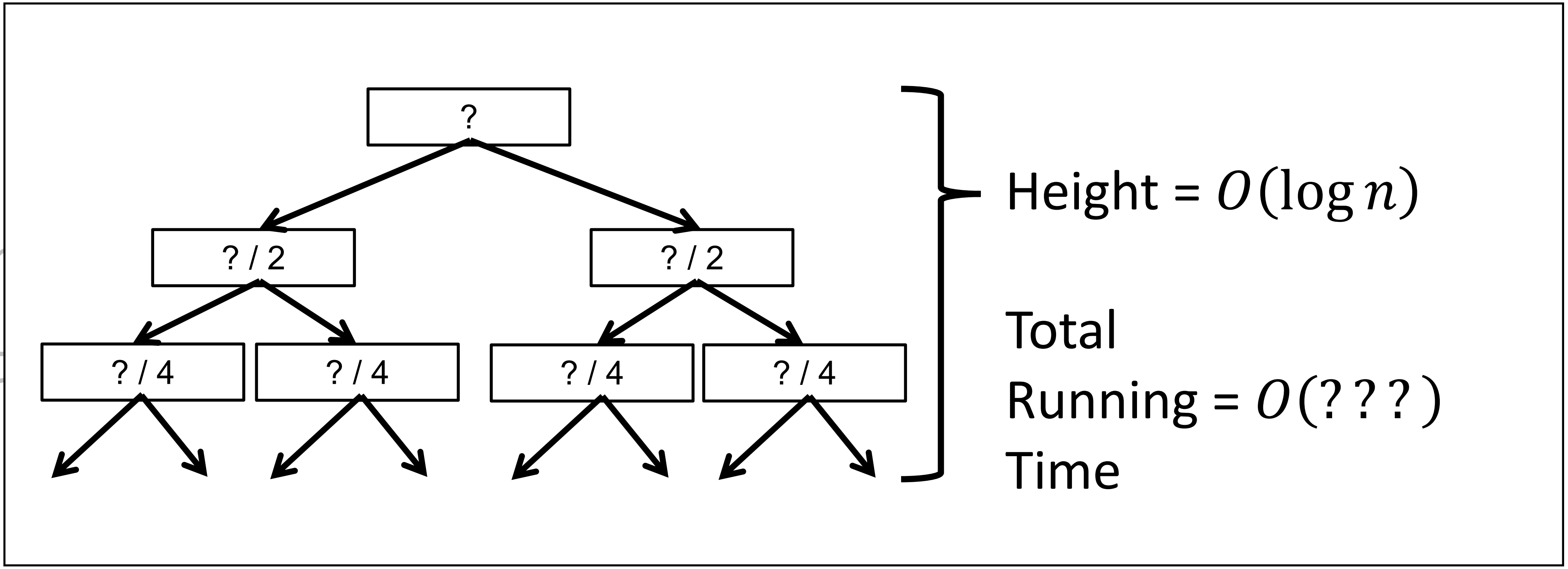
0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
6. **Return $\boldsymbol{\delta}$.** $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
6. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let
4. Let
5. Cor
6. Ret

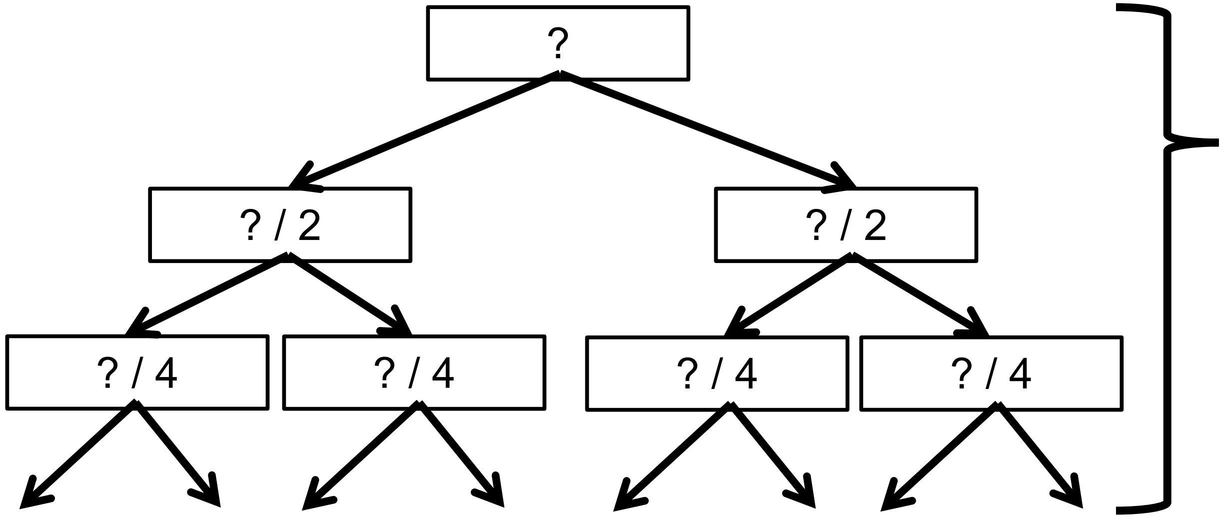


Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
6. Return δ. $\boldsymbol{O(1)}$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let
4. Let
5. Cor
6. Ret

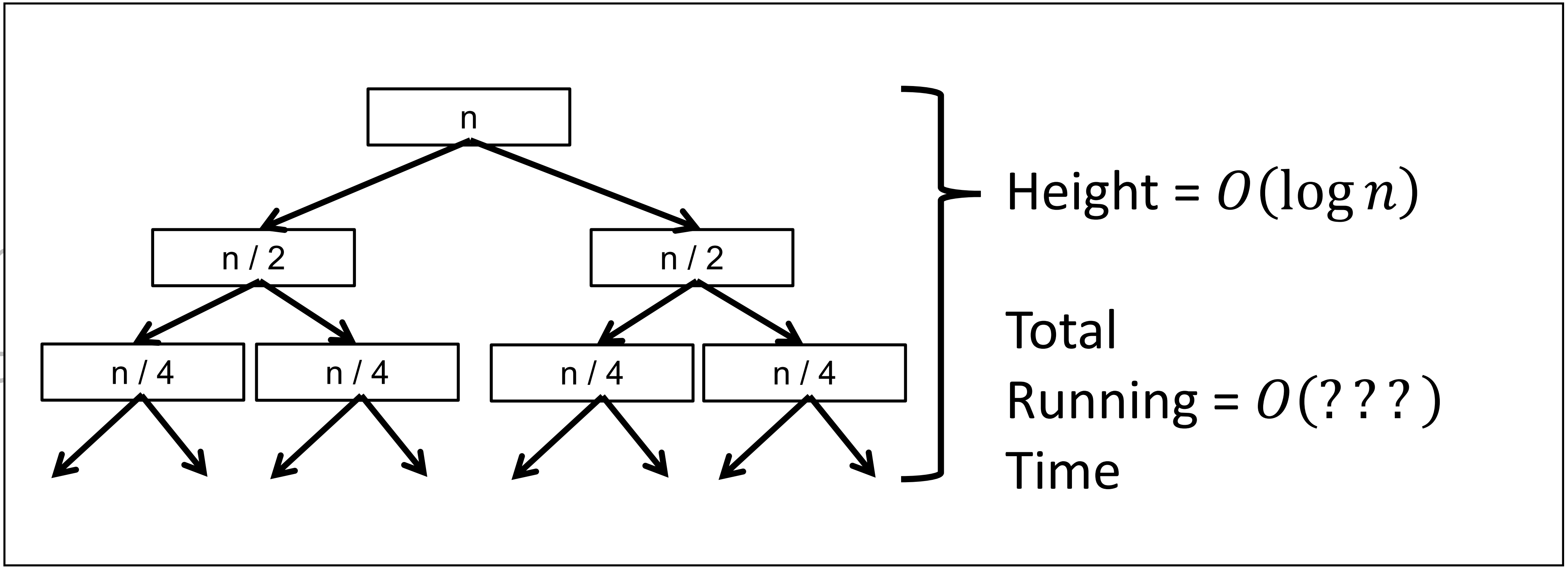


Height = $O(\log n)$

Total Running Time = $O(???)$

Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let
4. Let
5. Cor
6. Ret



Closest Pair Problem – Algorithm

0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let
4. Let
5. Cor
6. Ret