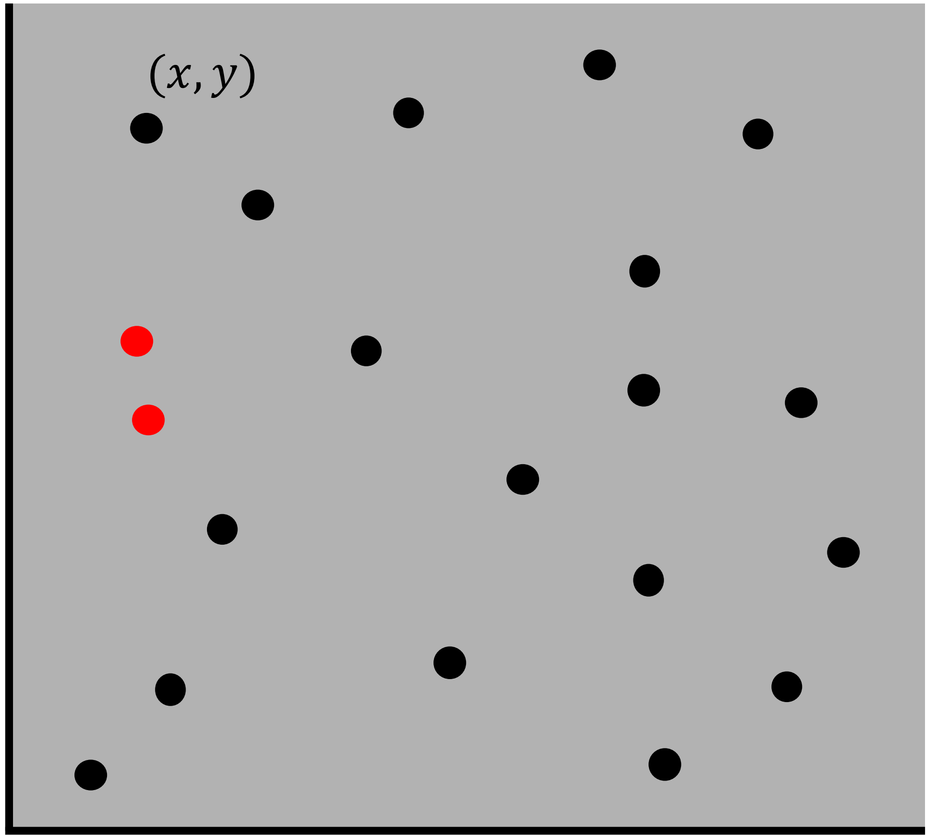
Closest Pair of Points

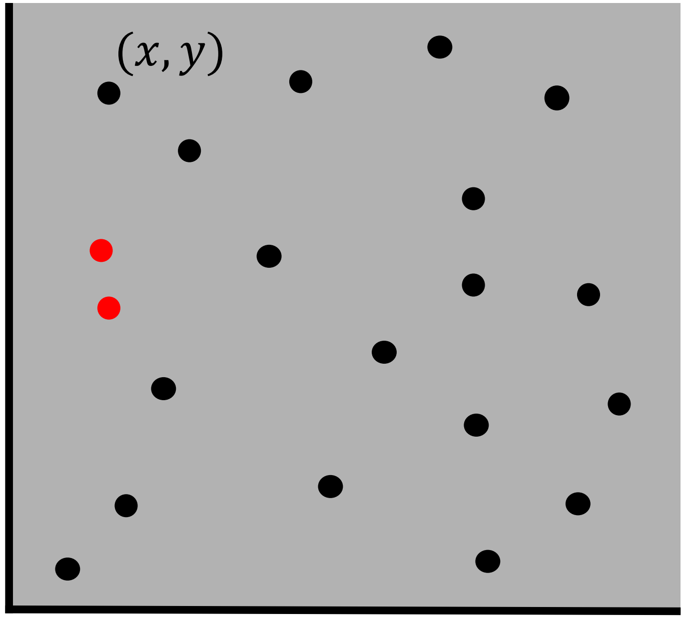
CSCI 532

Closest Pair Problem



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

Closest Pair Problem



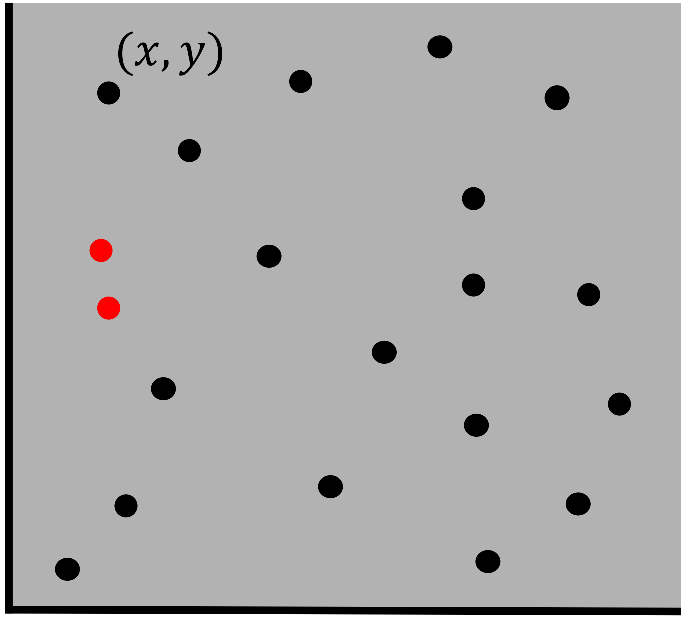
	P_1	P_2	…	P_n
P_1	/	$d_{1,2}$	…	$d_{1,n}$
P_2	$d_{2,1}$	/	…	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	…	/

Simple solution:

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

Running Time = ?

Closest Pair Problem



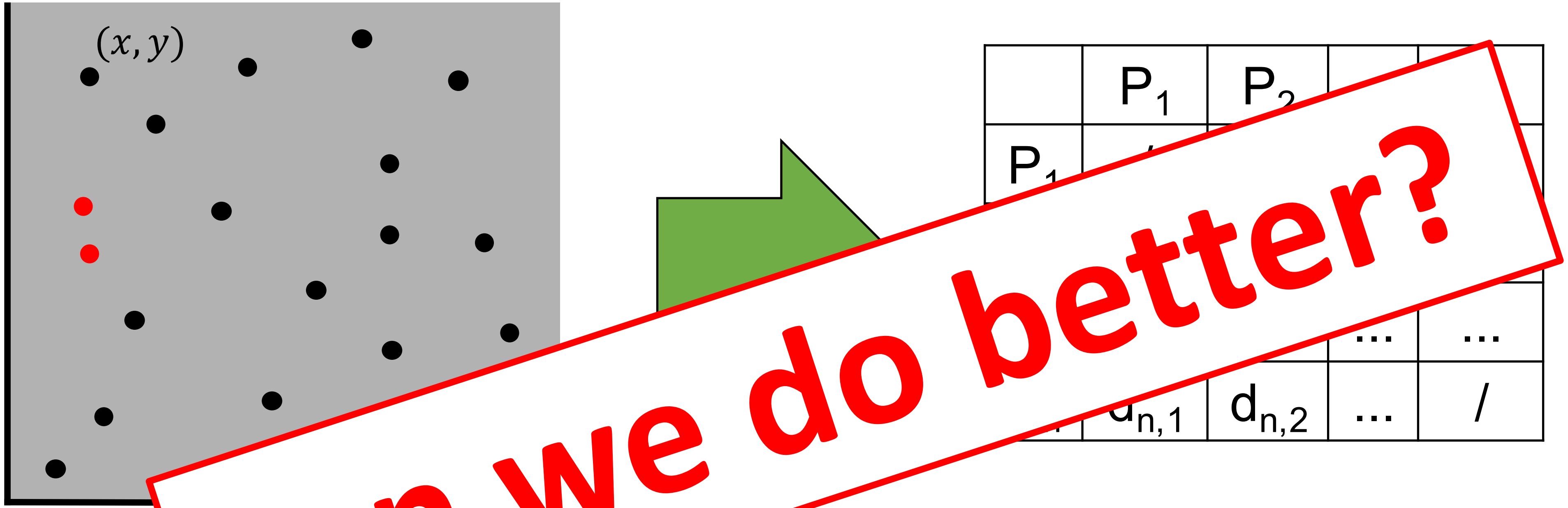
	P_1	P_2	...	P_n
P_1	/	$d_{1,2}$	...	$d_{1,n}$
P_2	$d_{2,1}$	/	...	$d_{2,n}$
...	...	...	...	...
P_n	$d_{n,1}$	$d_{n,2}$	...	/

Simple solution:

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

Running Time = $O(n^2)$

Closest Pair Problem



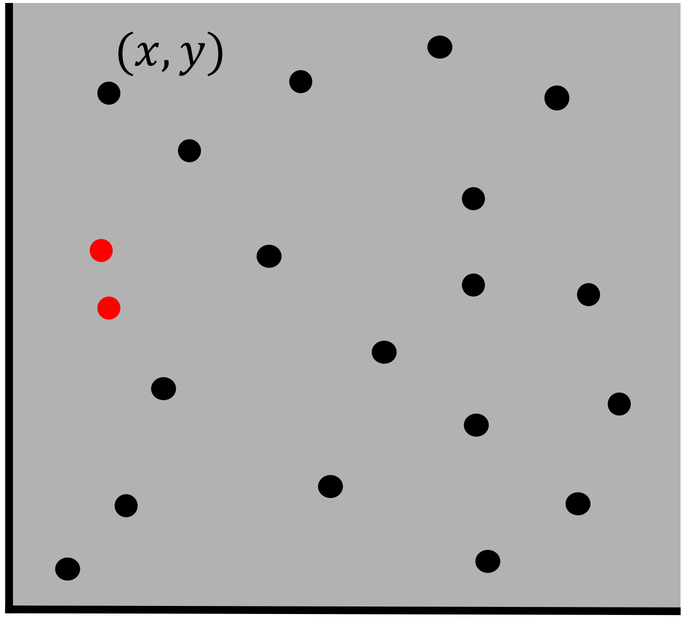
Can we do better?

solution:

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

Running Time = $O(n^2)$

Closest Pair Problem

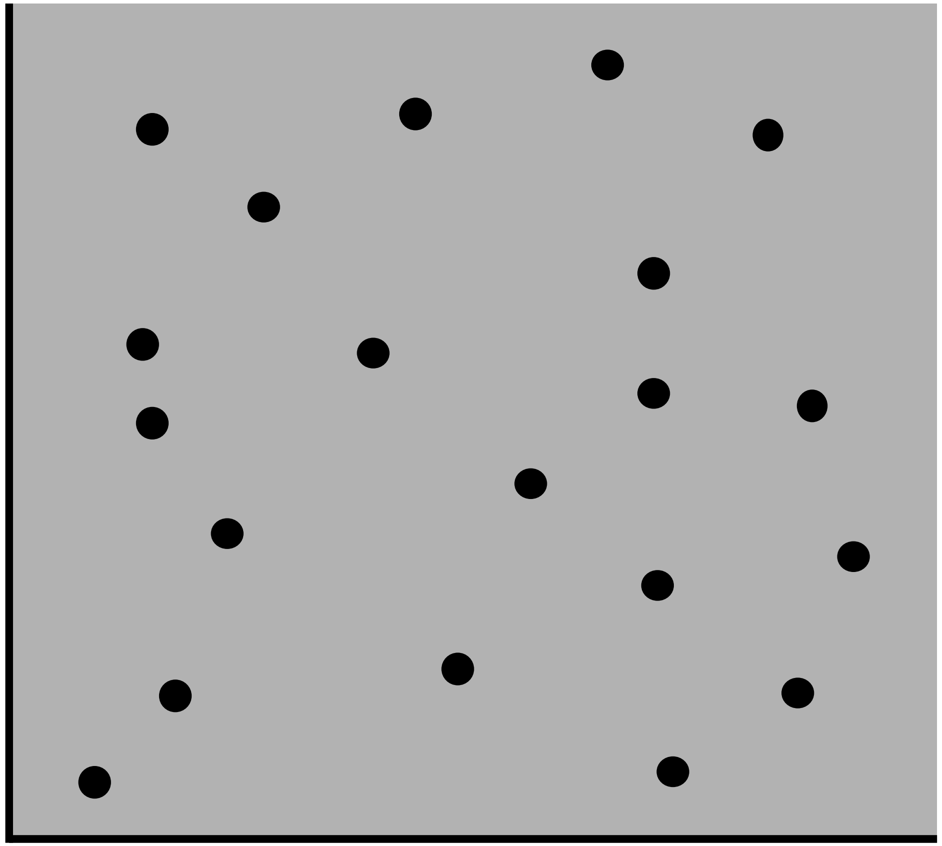


	P_1	P_2	...	P_n
P_1	/	$d_{1,2}$	...	$d_{1,n}$
P_2	$d_{2,1}$	/	...	$d_{2,n}$
...	...	...	...	...
P_n	$d_{n,1}$	$d_{n,2}$	...	/

Divide and Conquer Algorithms:

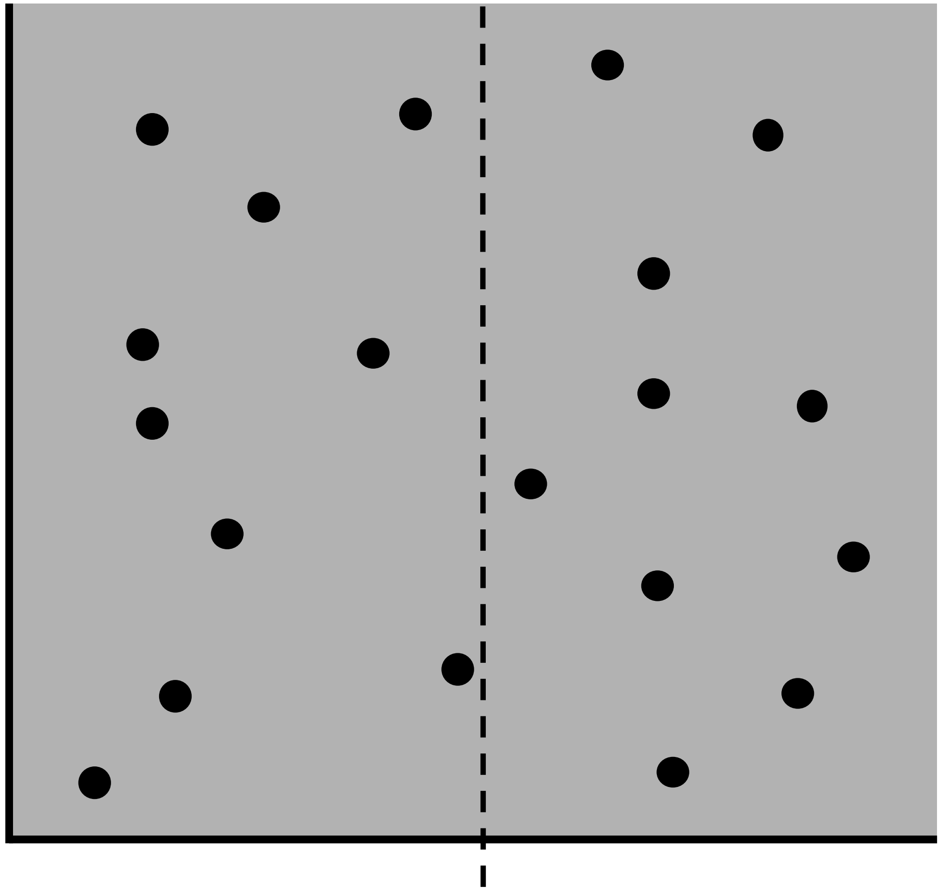
- Divide into subproblems that are smaller instances of the original.
- “Conquer” the subproblems by solving them recursively.
- Combine subproblem solutions into solution for original problem.

Closest Pair Problem – Divide and Conquer



How can we make the problem smaller and easier?

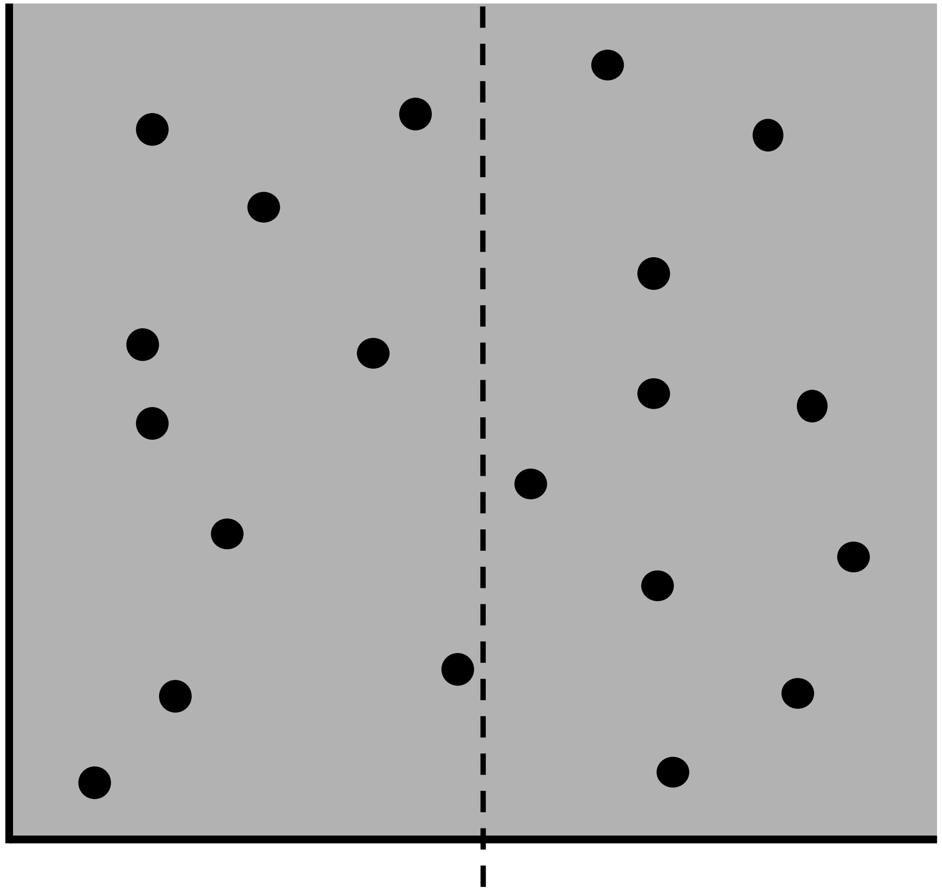
Closest Pair Problem – Divide and Conquer



How can we make the problem smaller and easier?

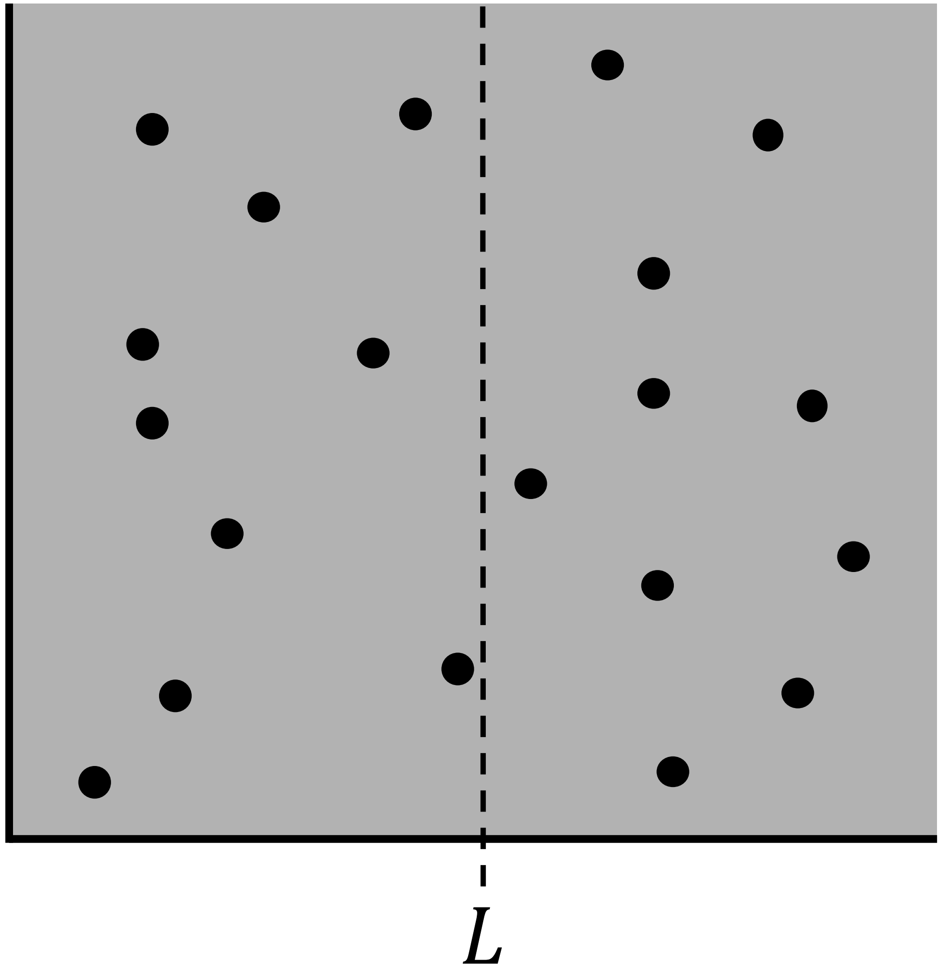
Split it up!

Closest Pair Problem – Divide and Conquer



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

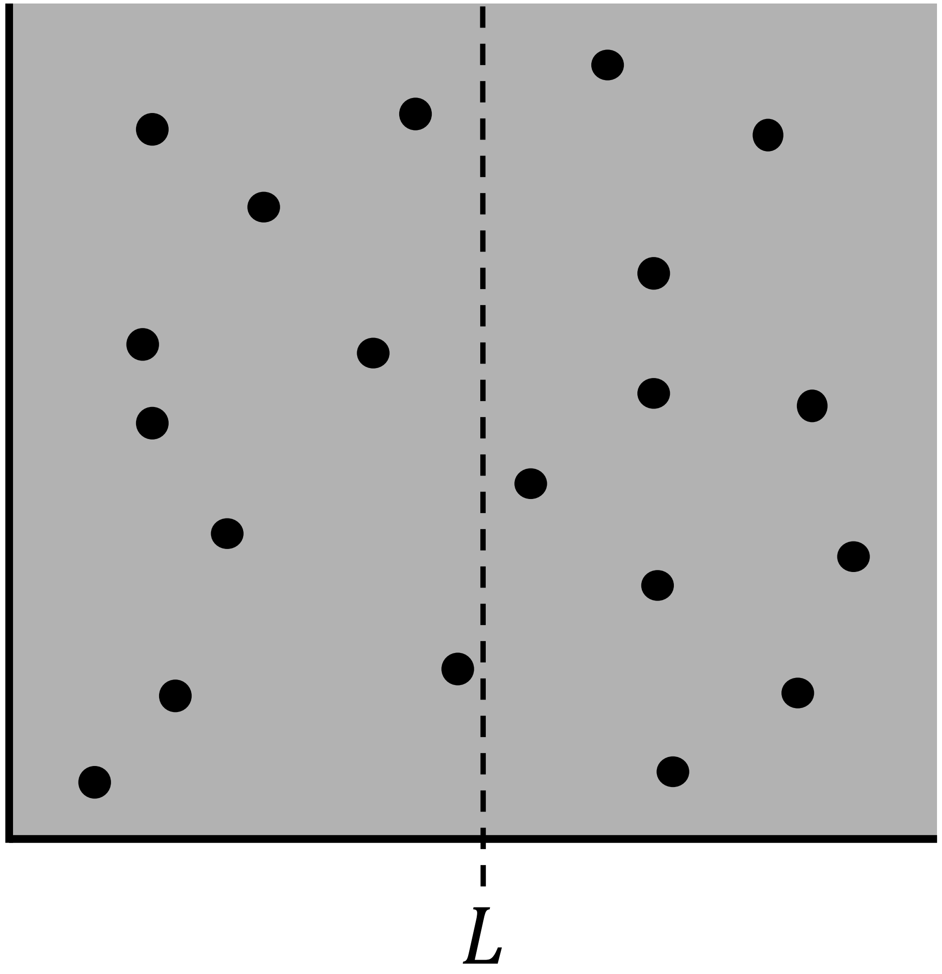
Closest Pair Problem – Divide and Conquer



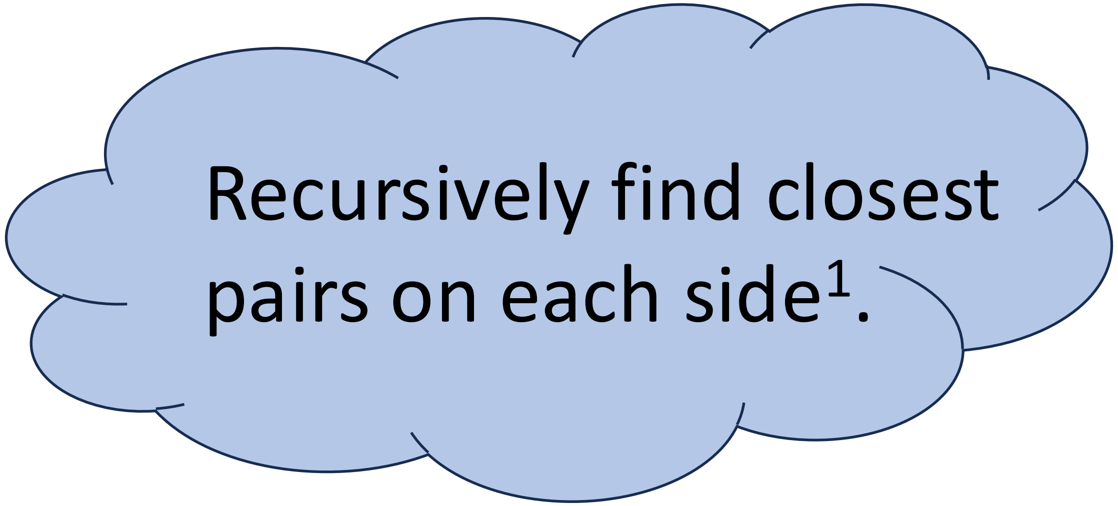
Divide: How can we draw line so that half of the points are on each side?

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

Closest Pair Problem – Divide and Conquer

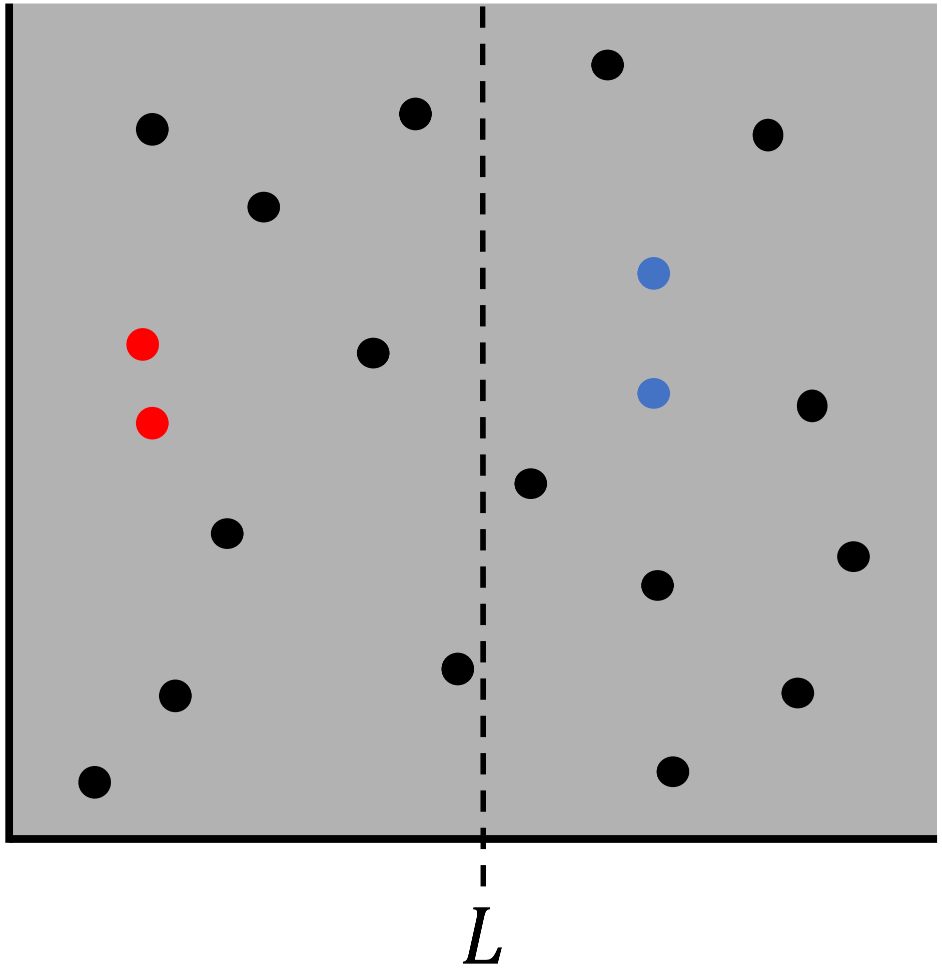


Conquer:



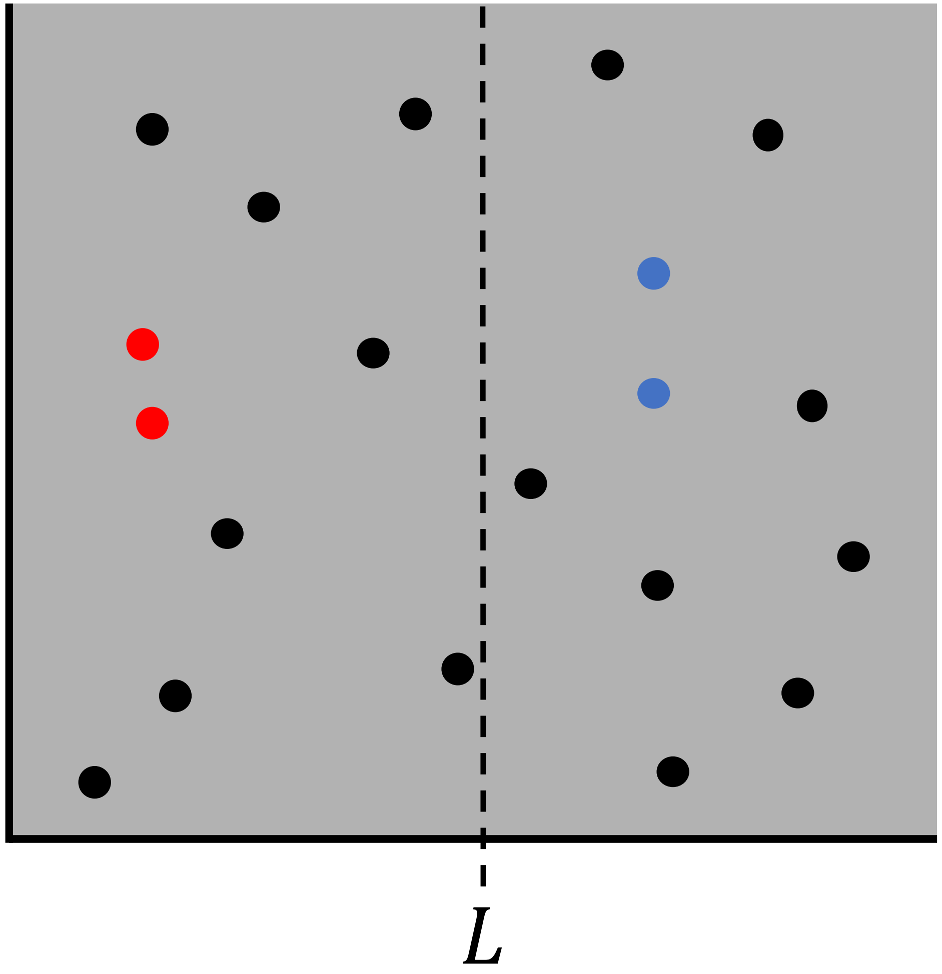
[1]Details to follow

Closest Pair Problem – Divide and Conquer



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

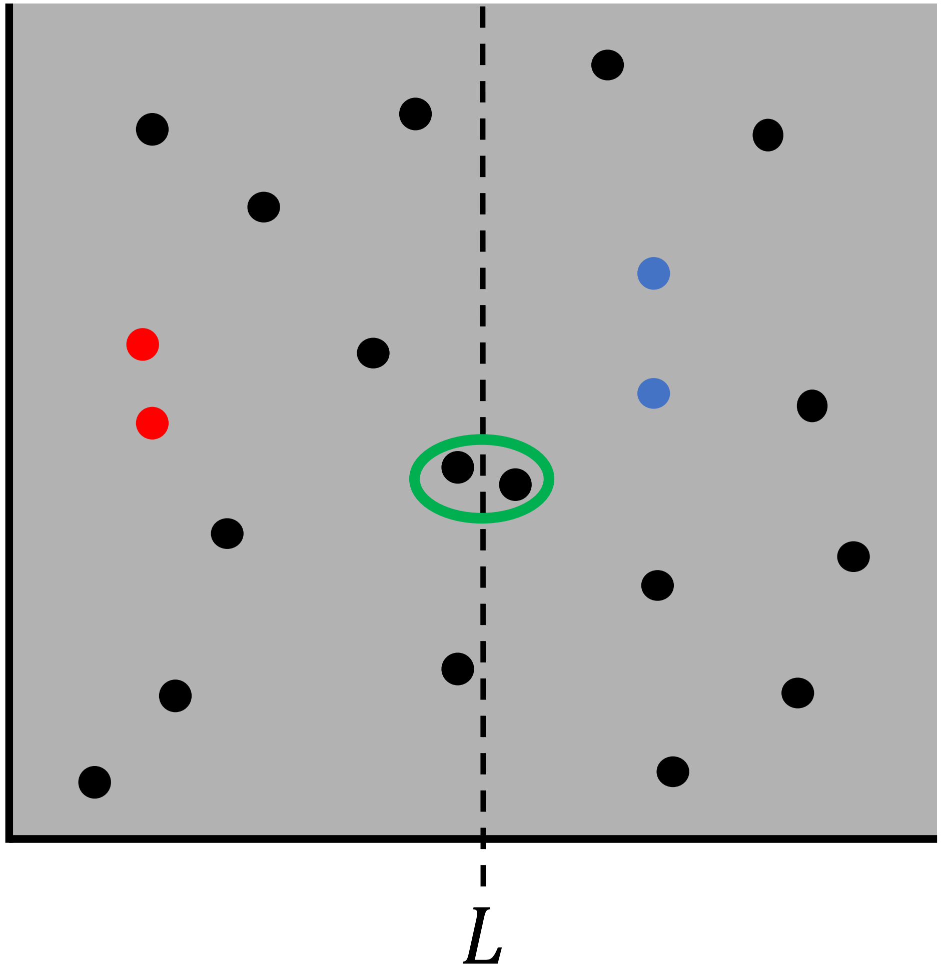
Closest Pair Problem – Divide and Conquer



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

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

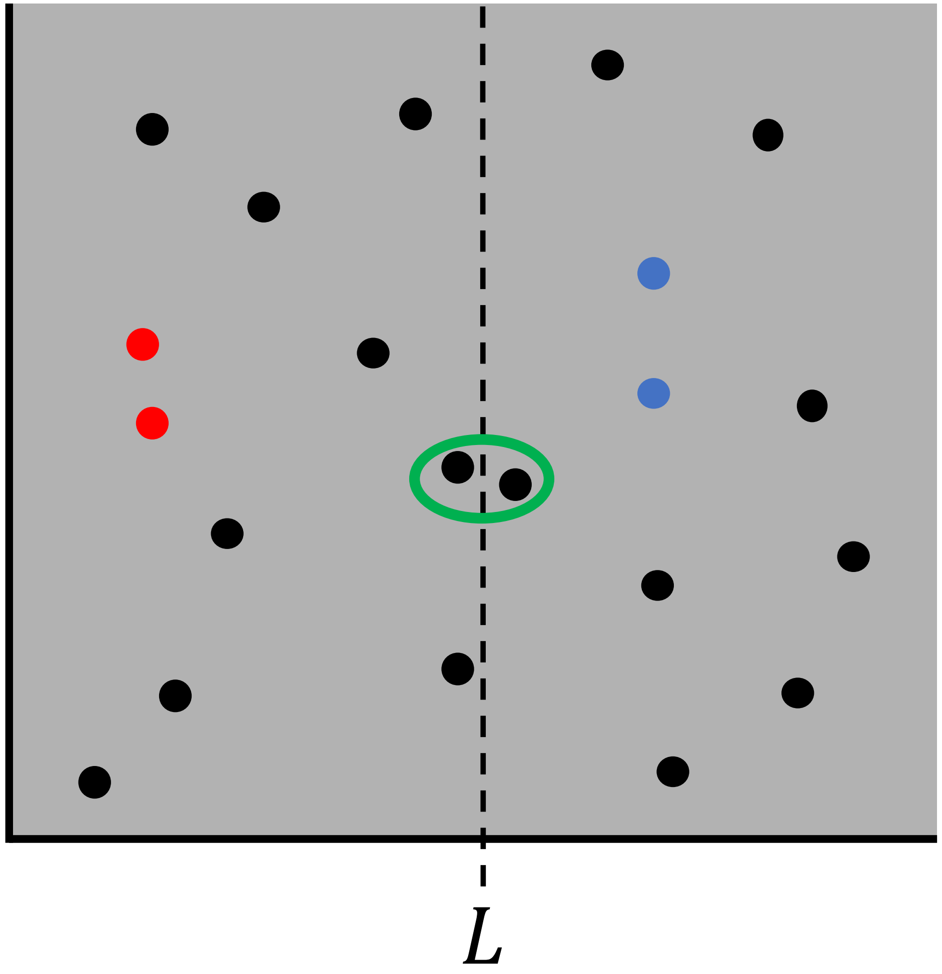
Closest Pair Problem – Divide and Conquer



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

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

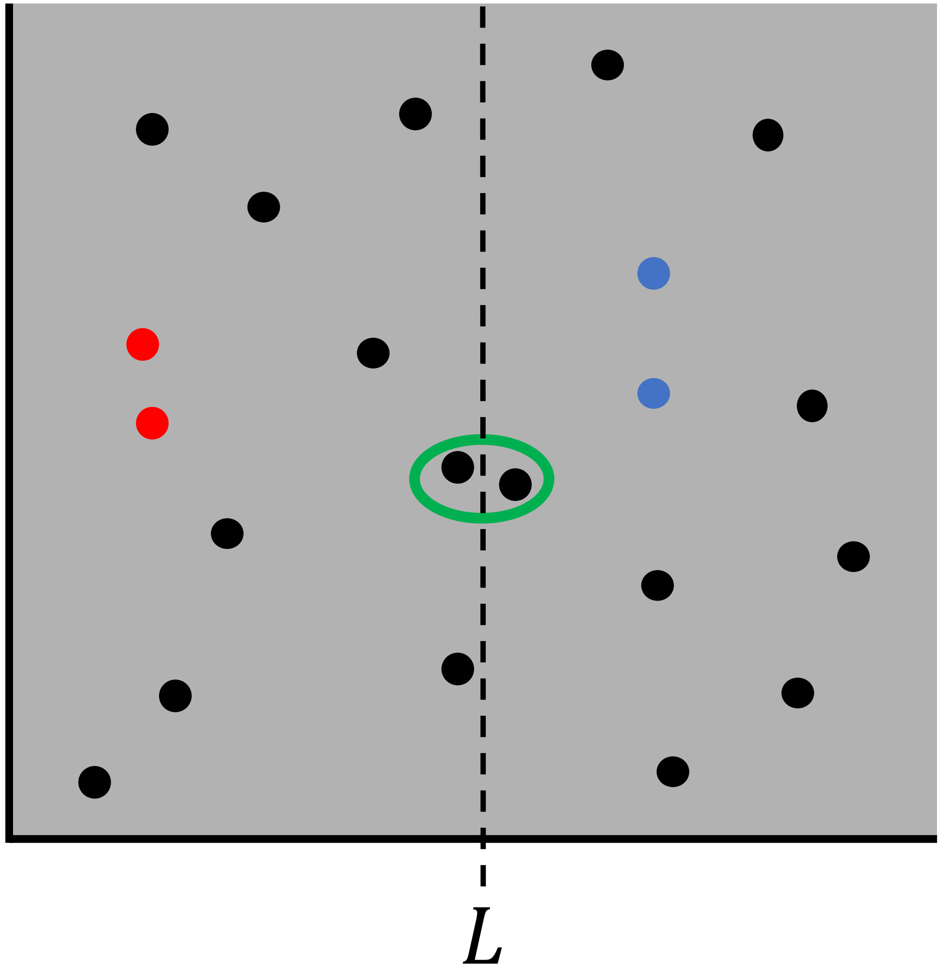
Closest Pair Problem – Divide and Conquer



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

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

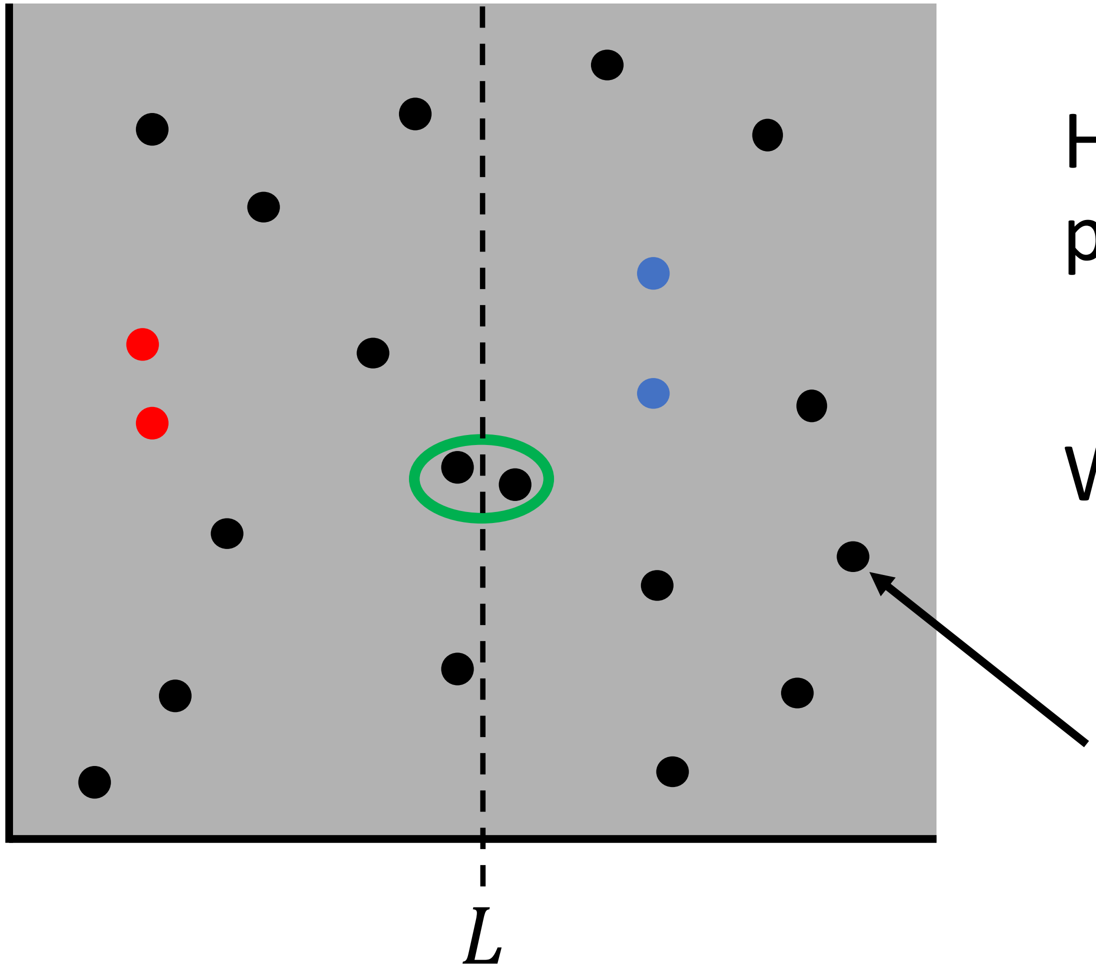
Closest Pair Problem – Divide and Conquer



How should we search for "straddle points"?

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

Closest Pair Problem – Divide and Conquer

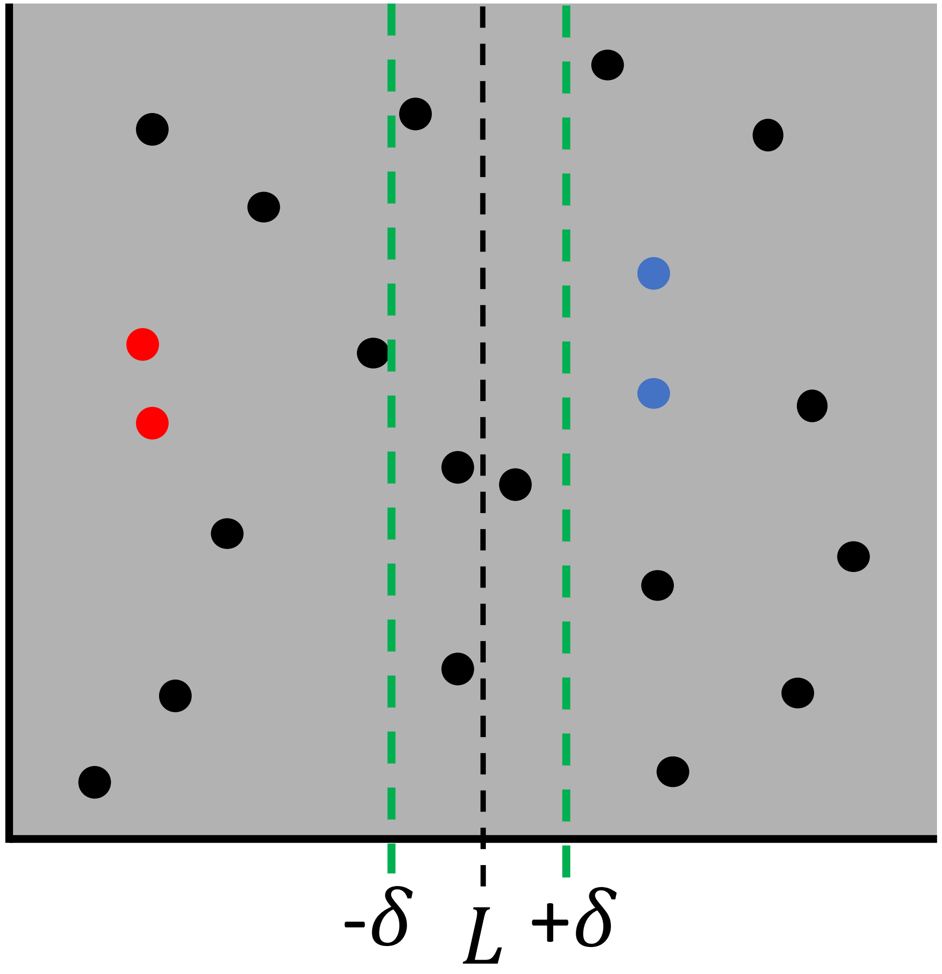


How should we search for “straddle points”?

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

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

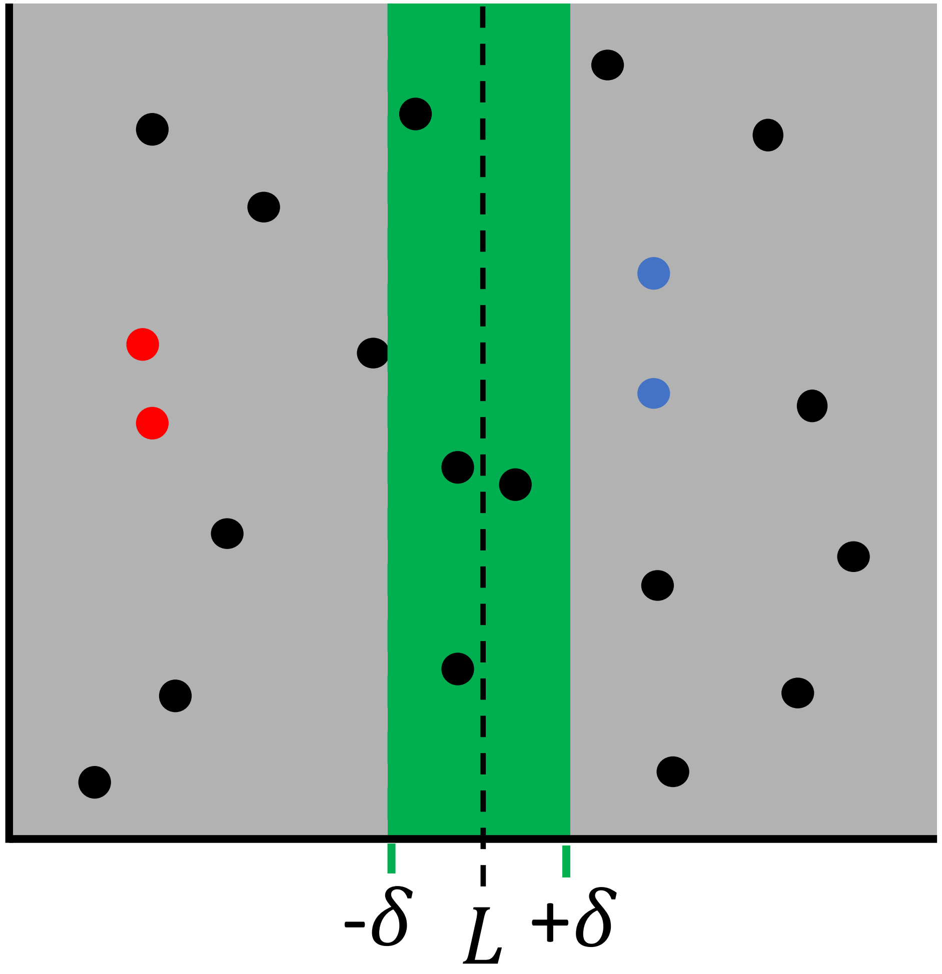
Closest Pair Problem – Divide and Conquer



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

Reason: Points outside $L \pm \delta$ cannot reach the other side in less than δ.

Closest Pair Problem – Divide and Conquer

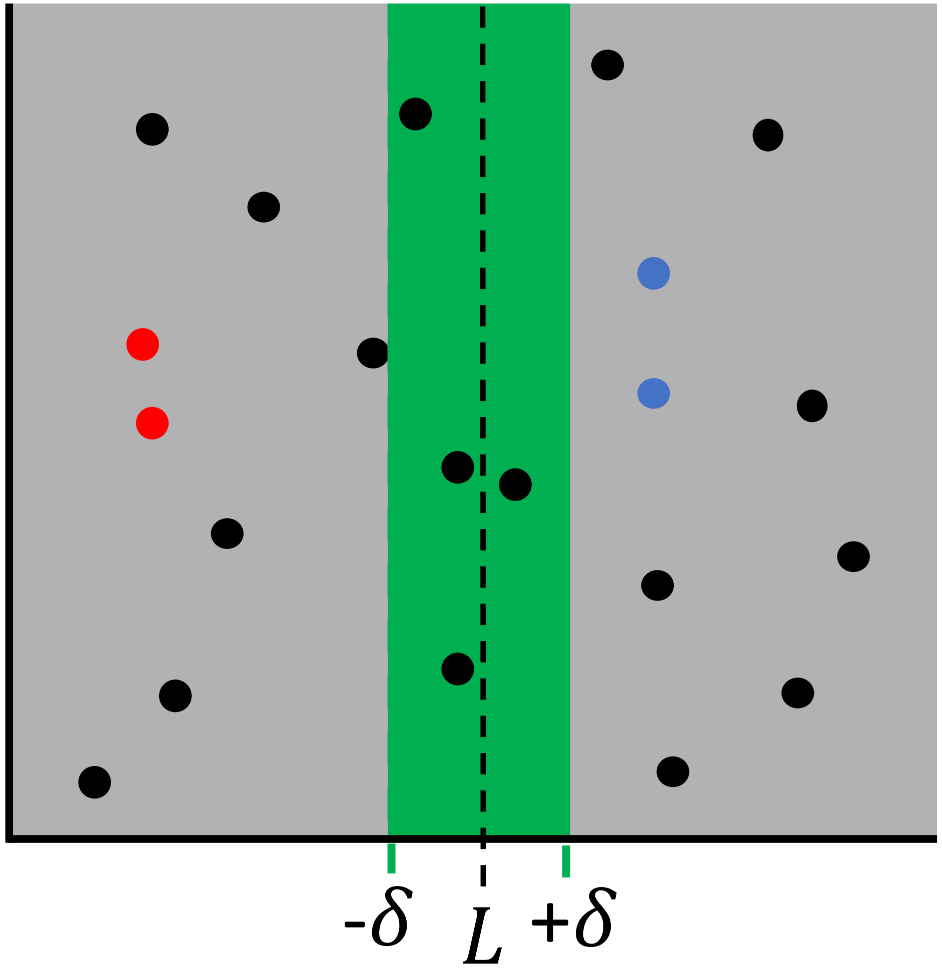


Rule: We only need to hunt for straddle points at most δ away from L.

Reason: Points outside $L \pm \delta$ cannot reach the other side in less than δ.

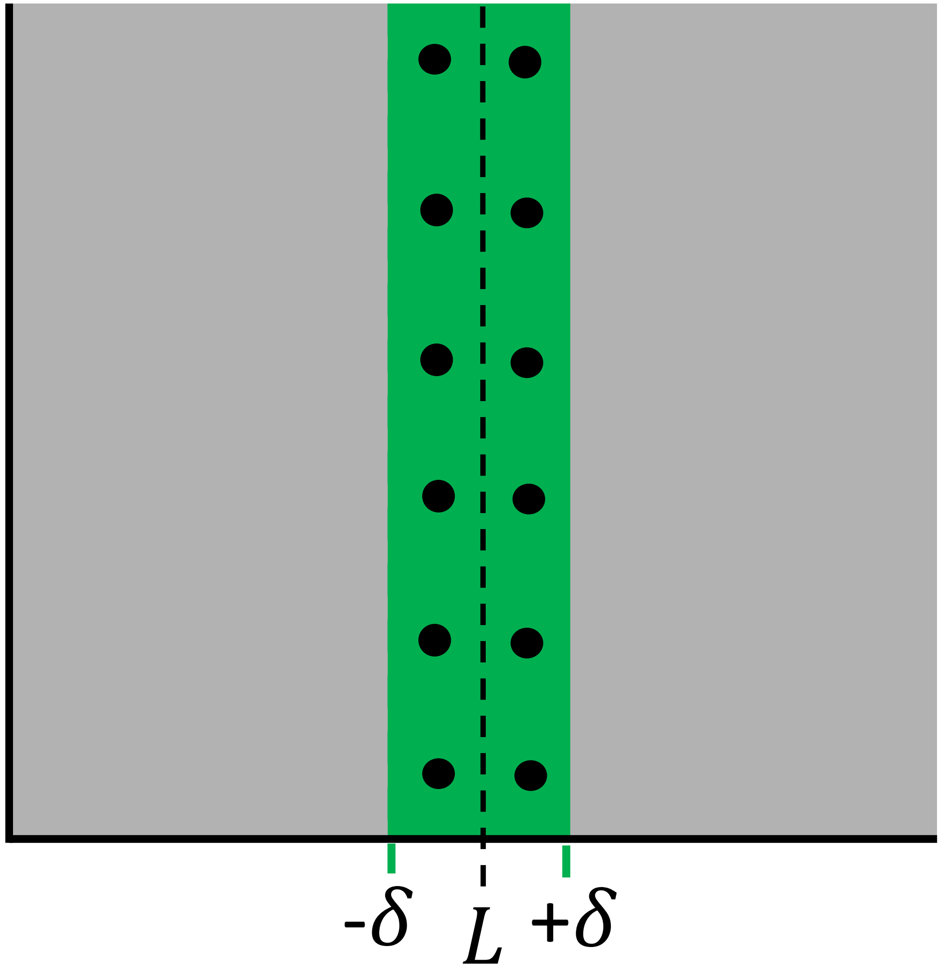
Let $\boldsymbol{S}$ be the set of straddle points.

Closest Pair Problem – Divide and Conquer



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

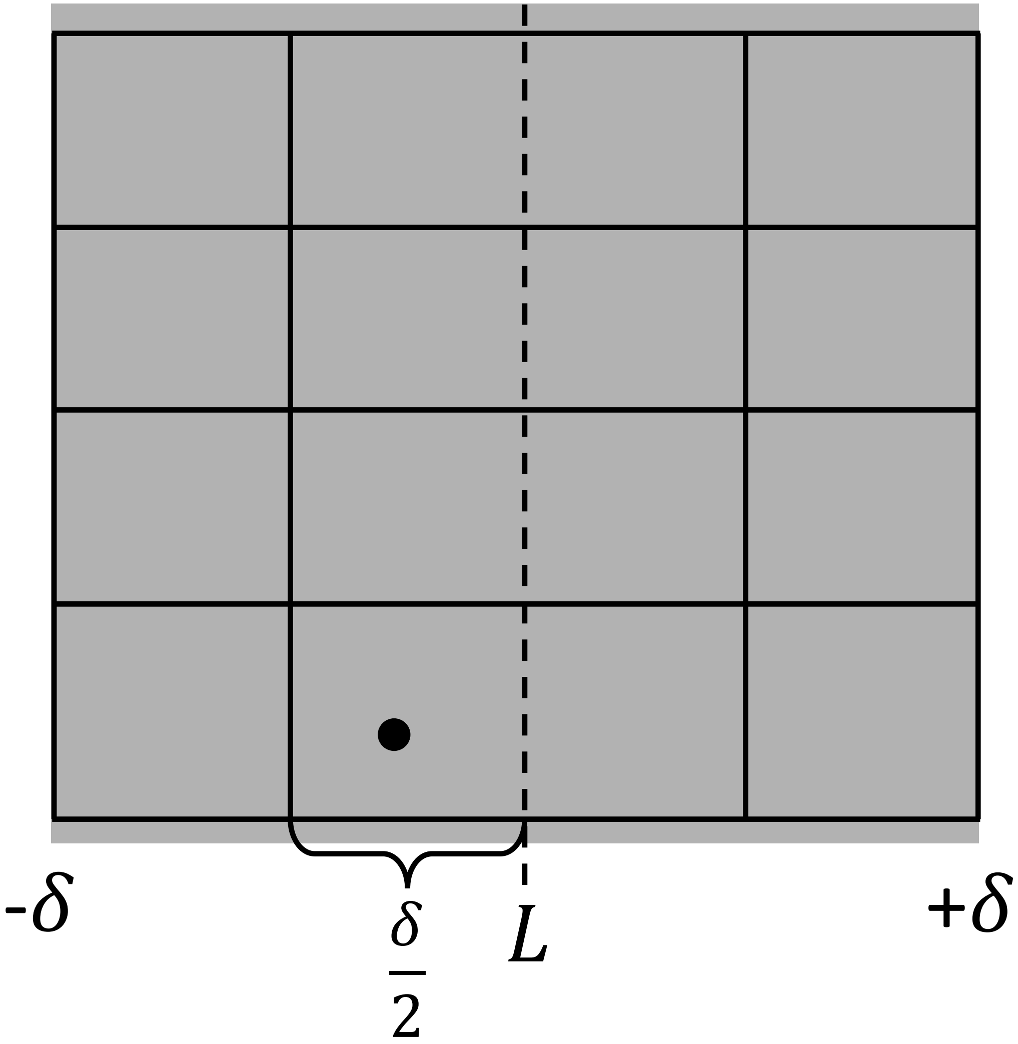
Closest Pair Problem – Divide and Conquer



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

So, we need to reduce the number of straddle points we have to consider.

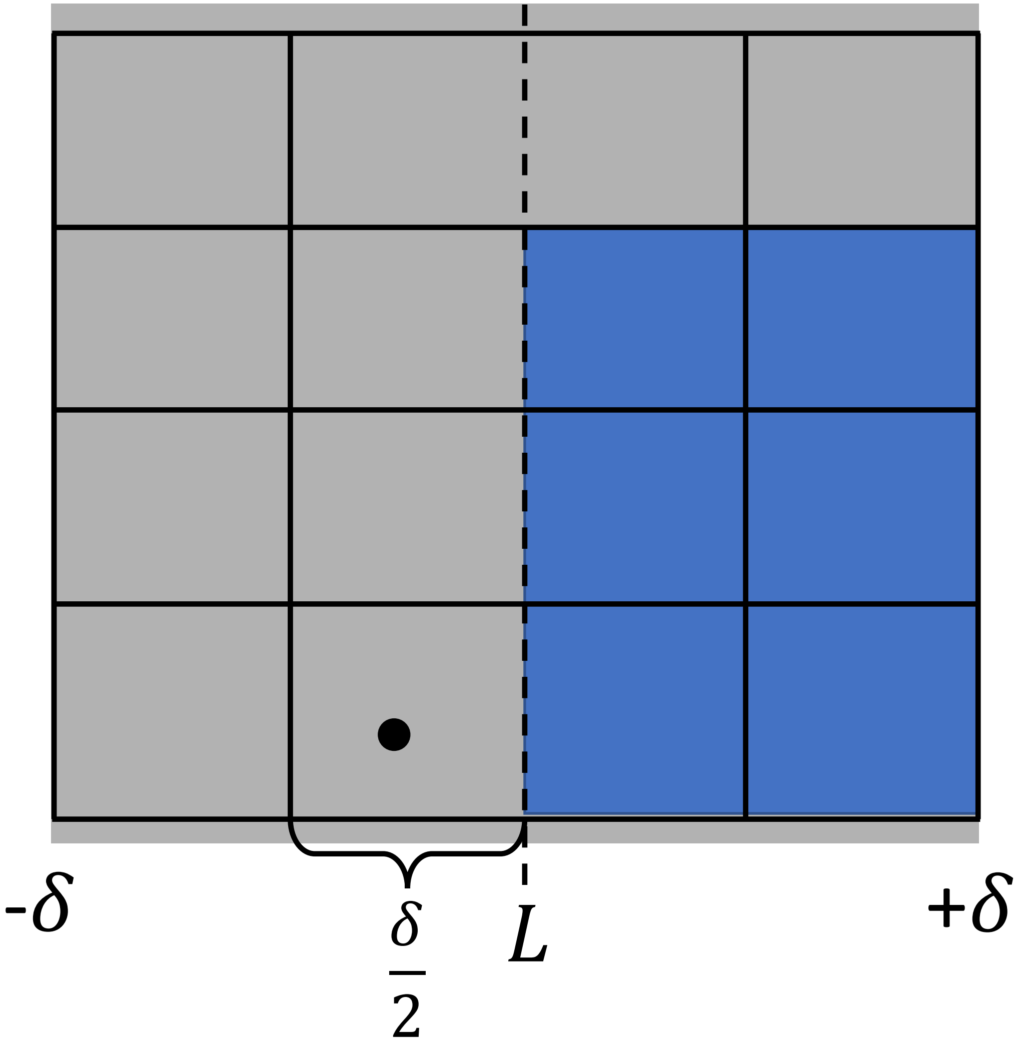
Closest Pair Problem – Divide and Conquer



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

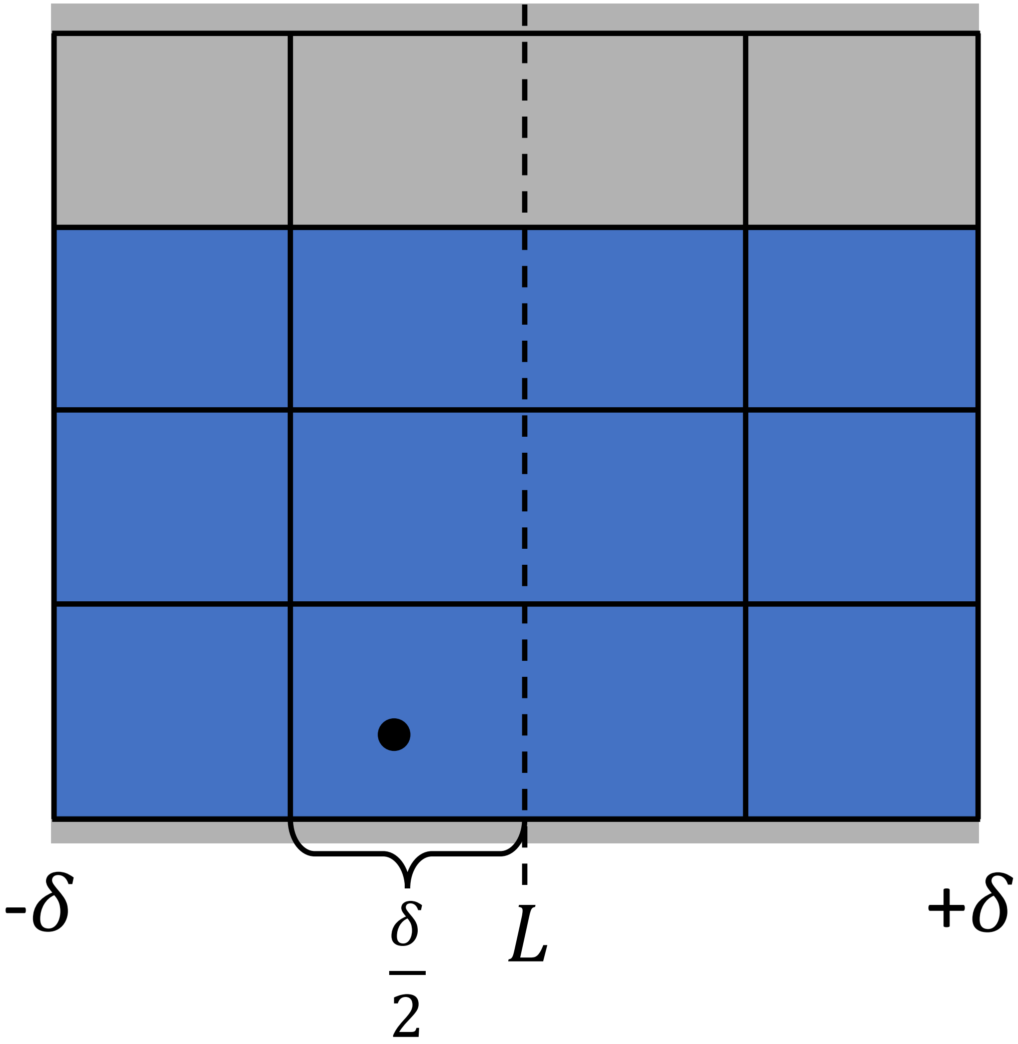


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points within δ.

Closest Pair Problem – Divide and Conquer

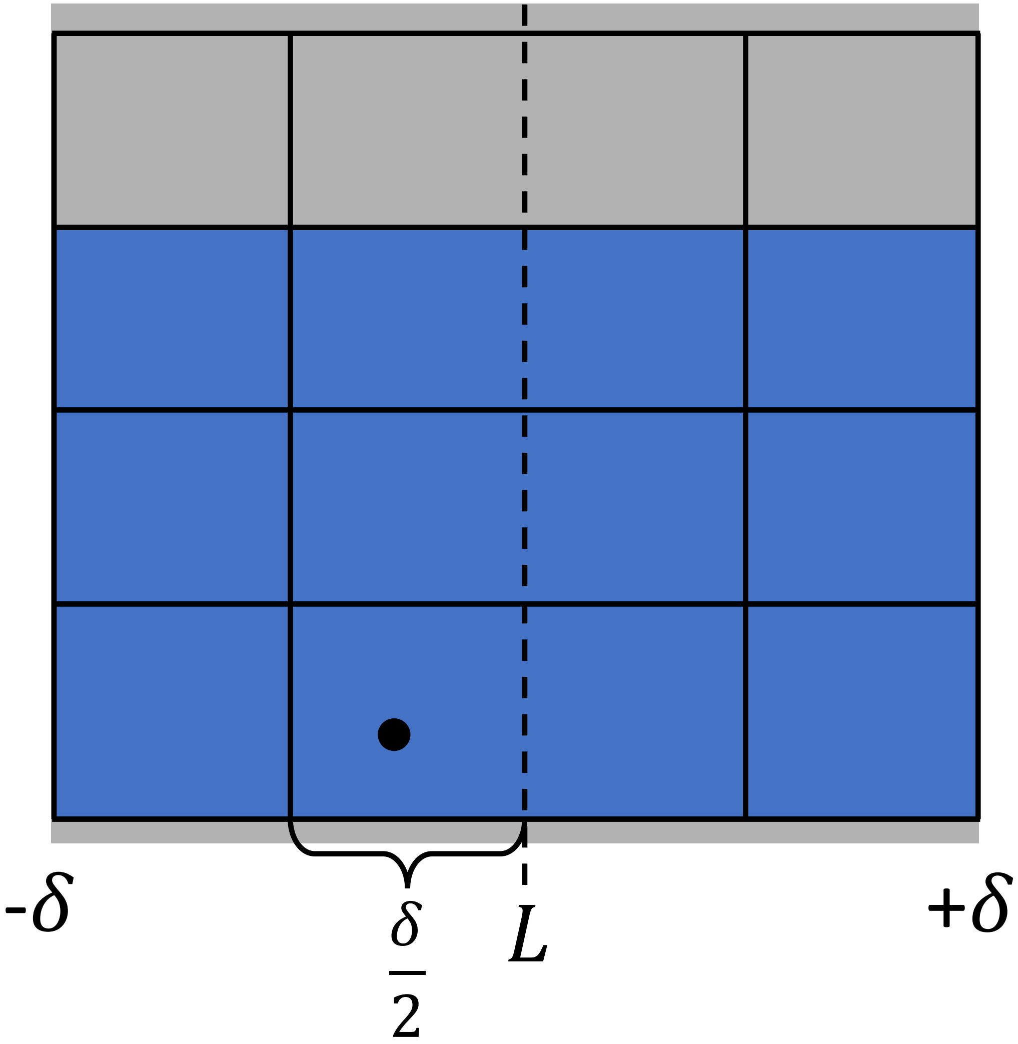


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points within δ.

Closest Pair Problem – Divide and Conquer



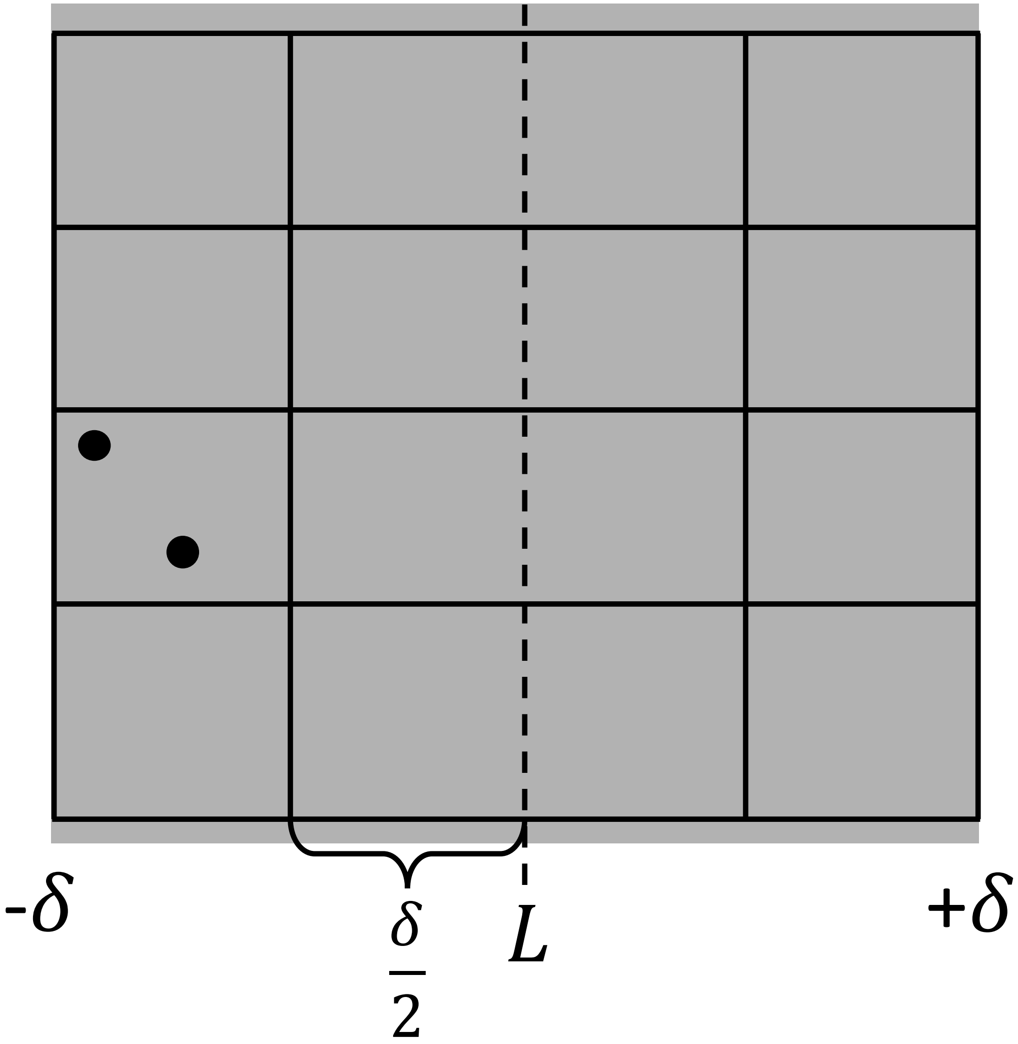
Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points within δ.

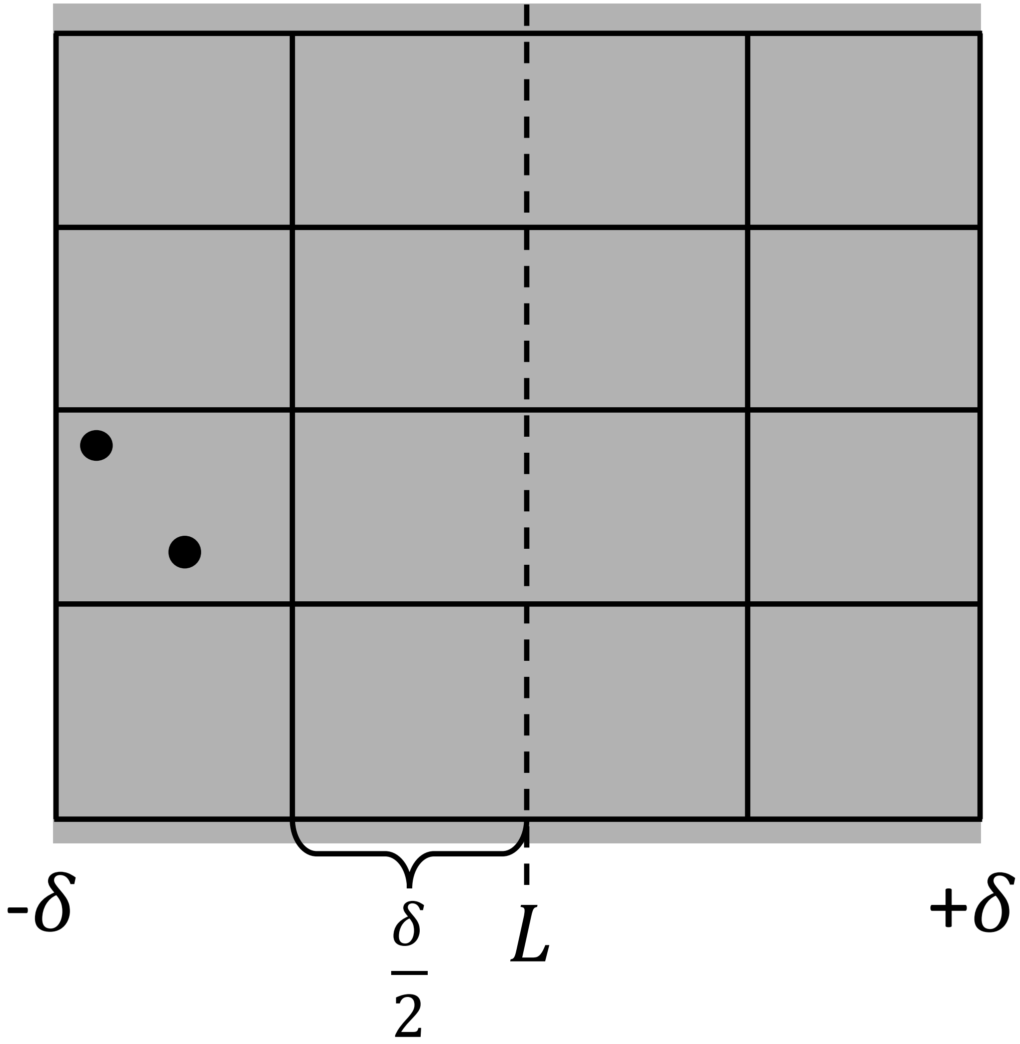
Does this reduce the number of points we need to consider?

Closest Pair Problem – Divide and Conquer



Can we have multiple points in one box?

Closest Pair Problem – Divide and Conquer

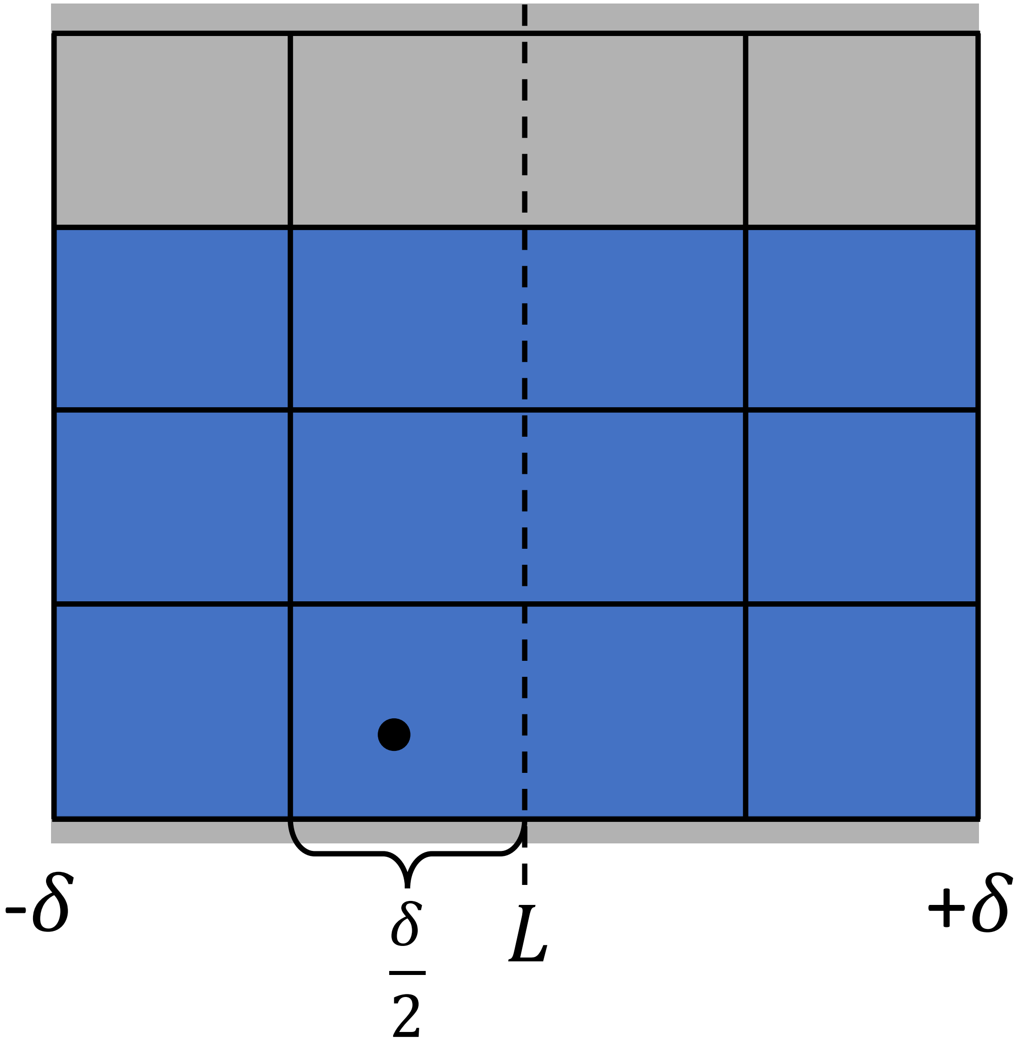


Can we have multiple points in one box?

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

$\Rightarrow$ at most one point per box.

Closest Pair Problem – Divide and Conquer



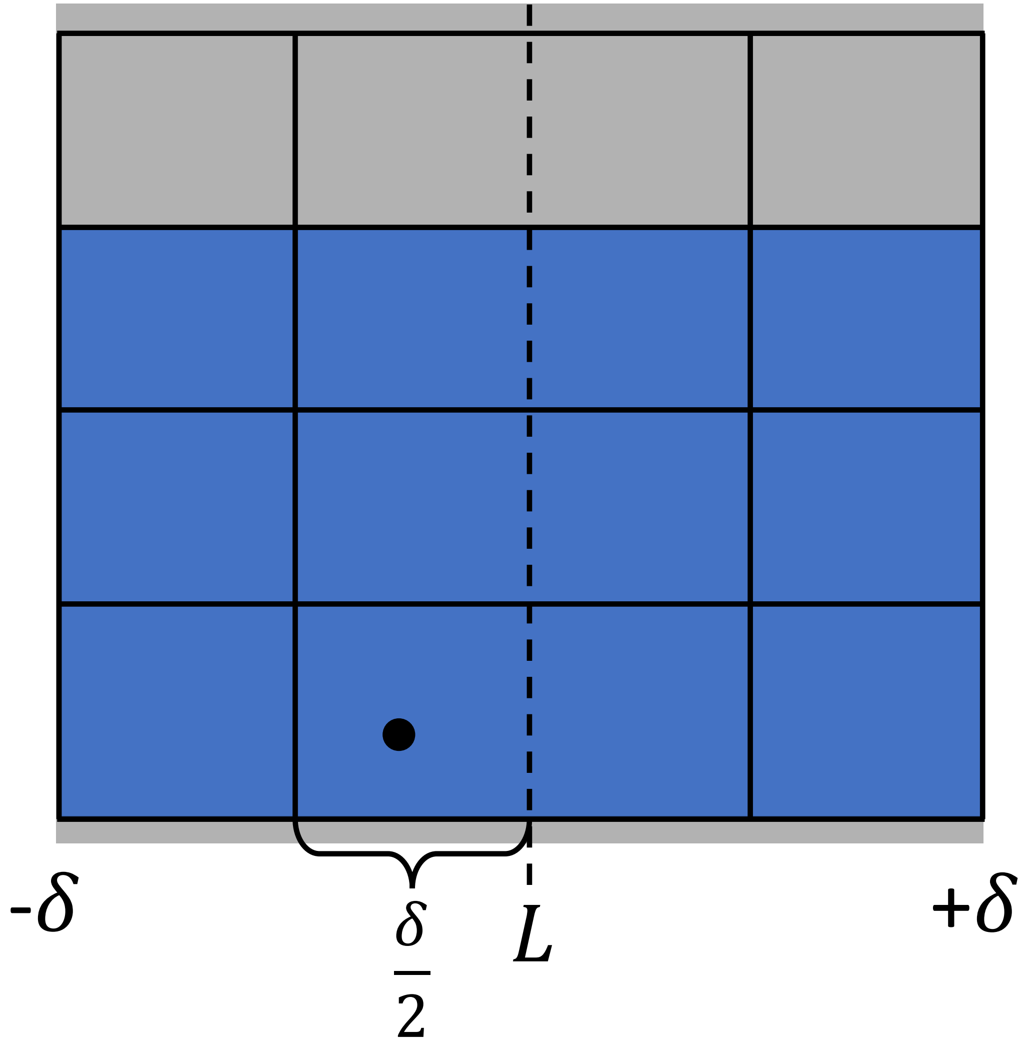
Only care about certain boxes

\+ At most one point per box

Fixed number of points to check

1. Sort straddle points by y coordinate.
2. Only possible "δ-busting" points are the 11 points after our point being considered.

Closest Pair Problem – Divide and Conquer



Only care about certain boxes
\+ At most one point per box

Fixed number of points to check

Straddle point hunting:

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

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

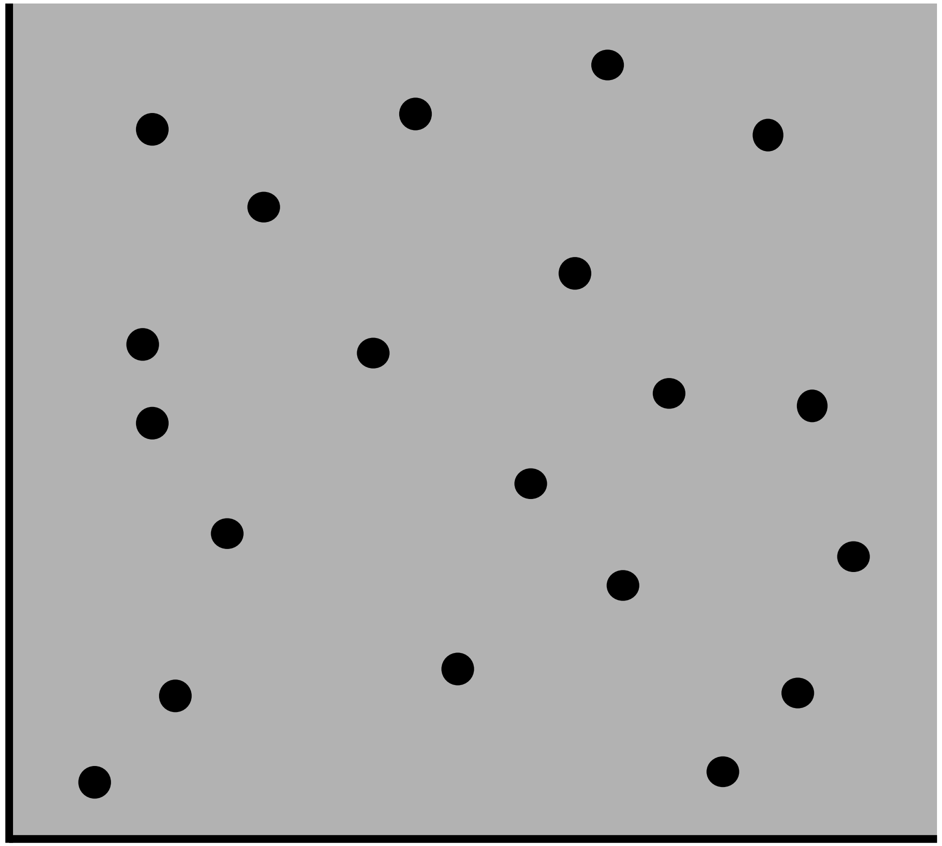
Closest Pair Problem – Algorithm

findClosestPair(P):

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. **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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Recursively find closest pairs on each side.

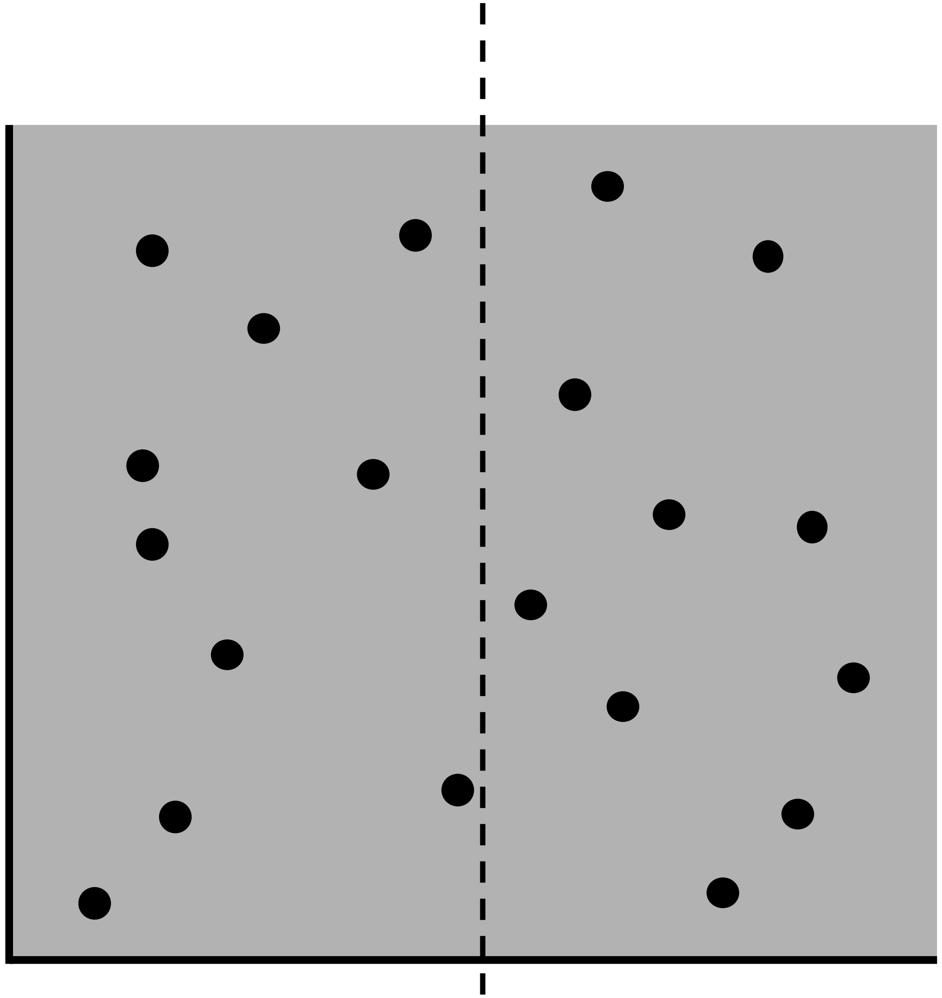
Closest Pair Problem – Divide and Conquer



Recursive Process:

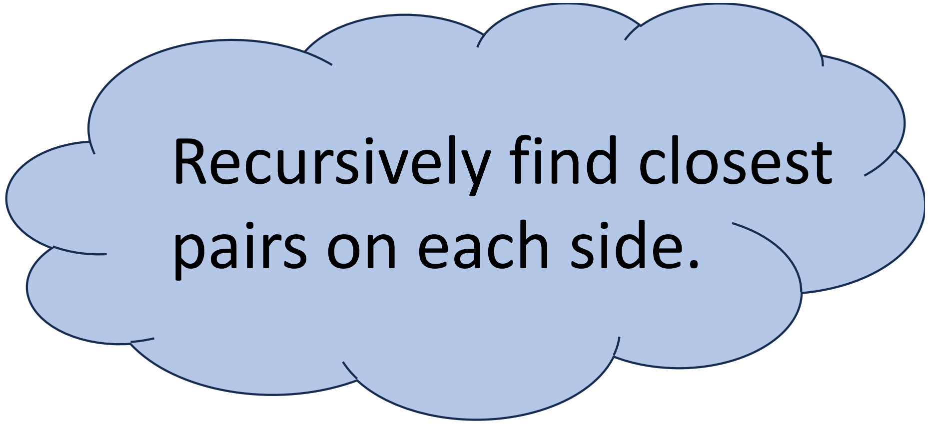
Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer

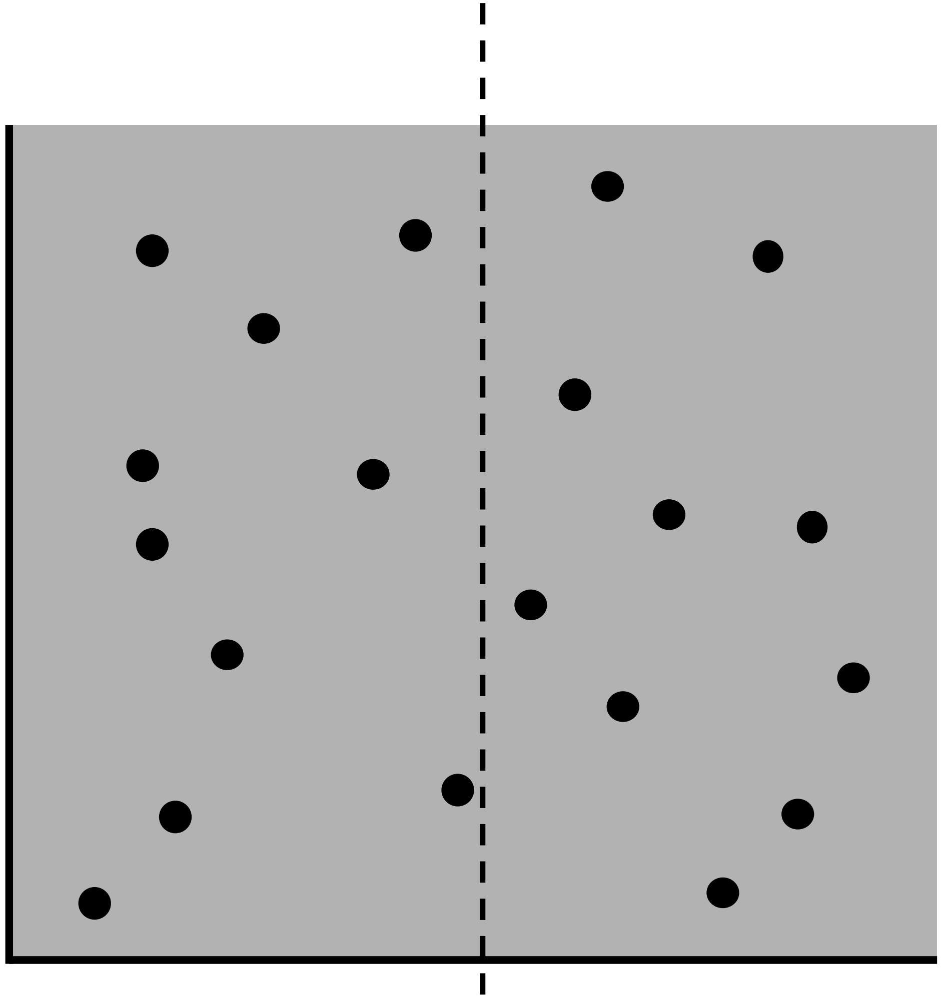


Recursive Process:

1. Divide points in half.



Closest Pair Problem – Divide and Conquer

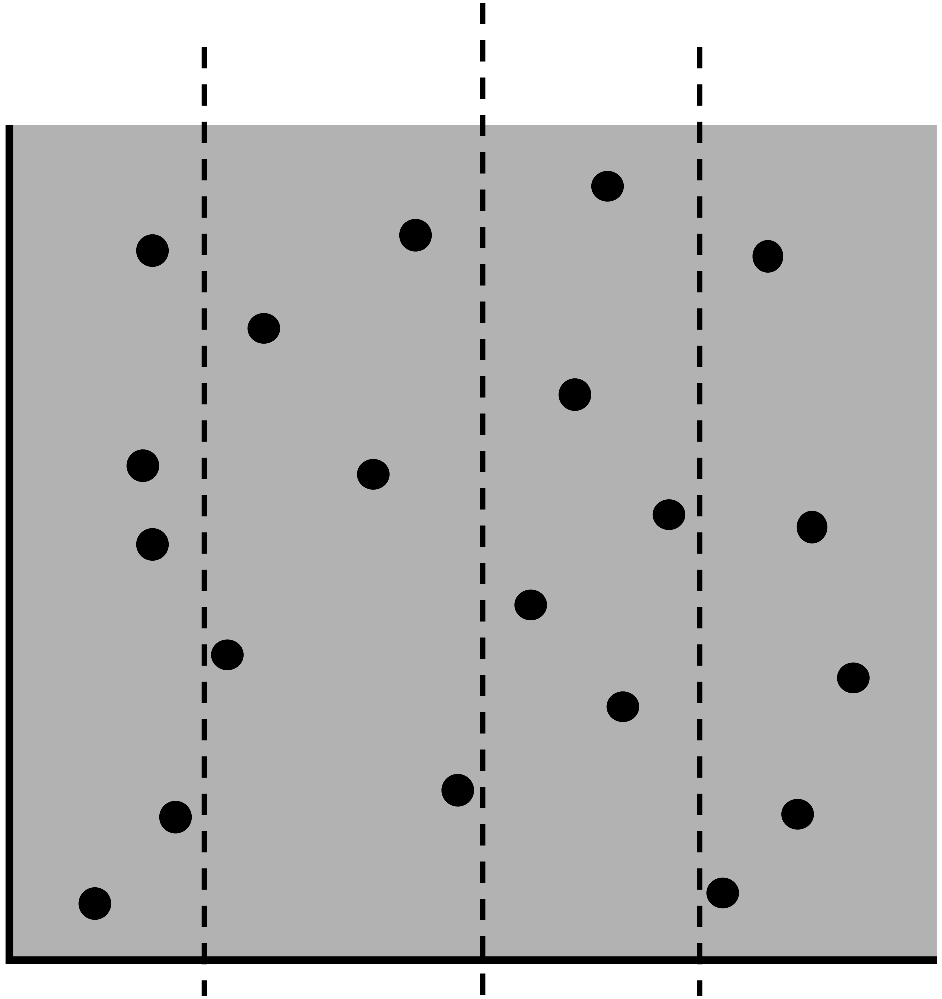


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until determining d_{left} and d_{right} is trivial.

Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer

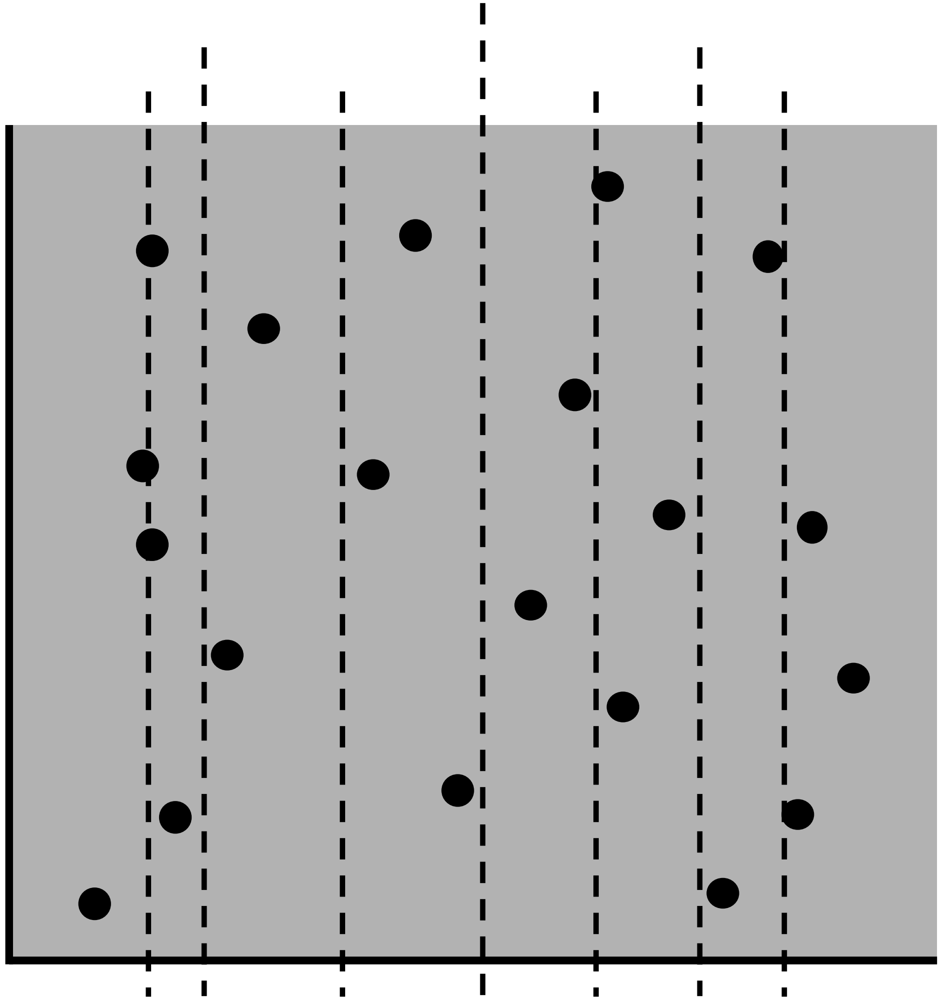


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until determining d_{left} and d_{right} is trivial.

Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer

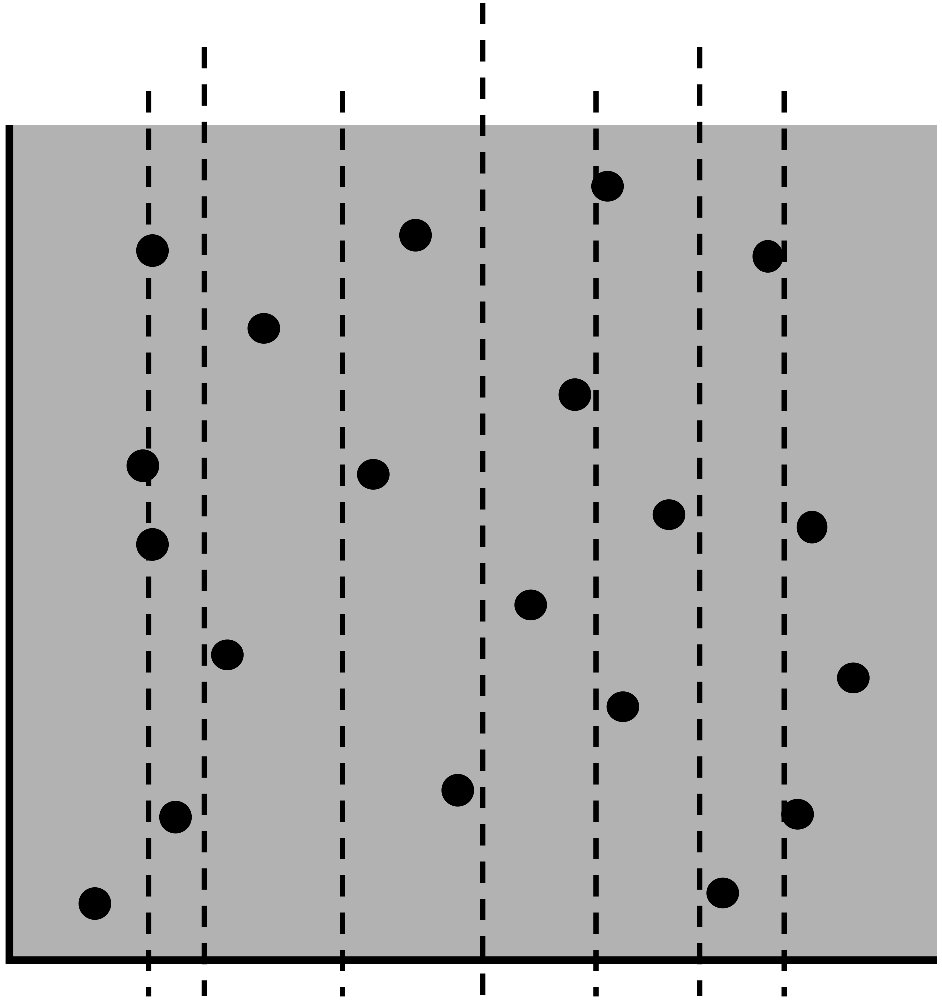


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until determining d_{left} and d_{right} is trivial.

Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer

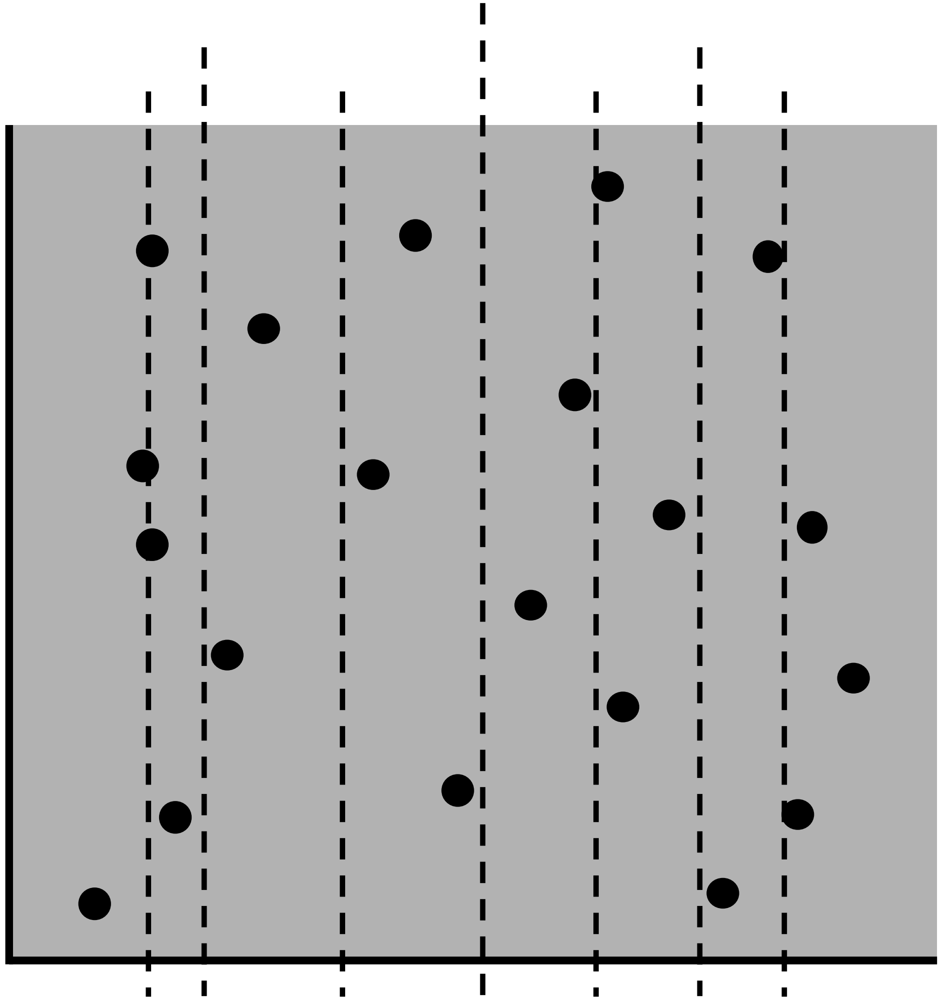


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until determining d_{left} and d_{right} is trivial.

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

Closest Pair Problem – Divide and Conquer



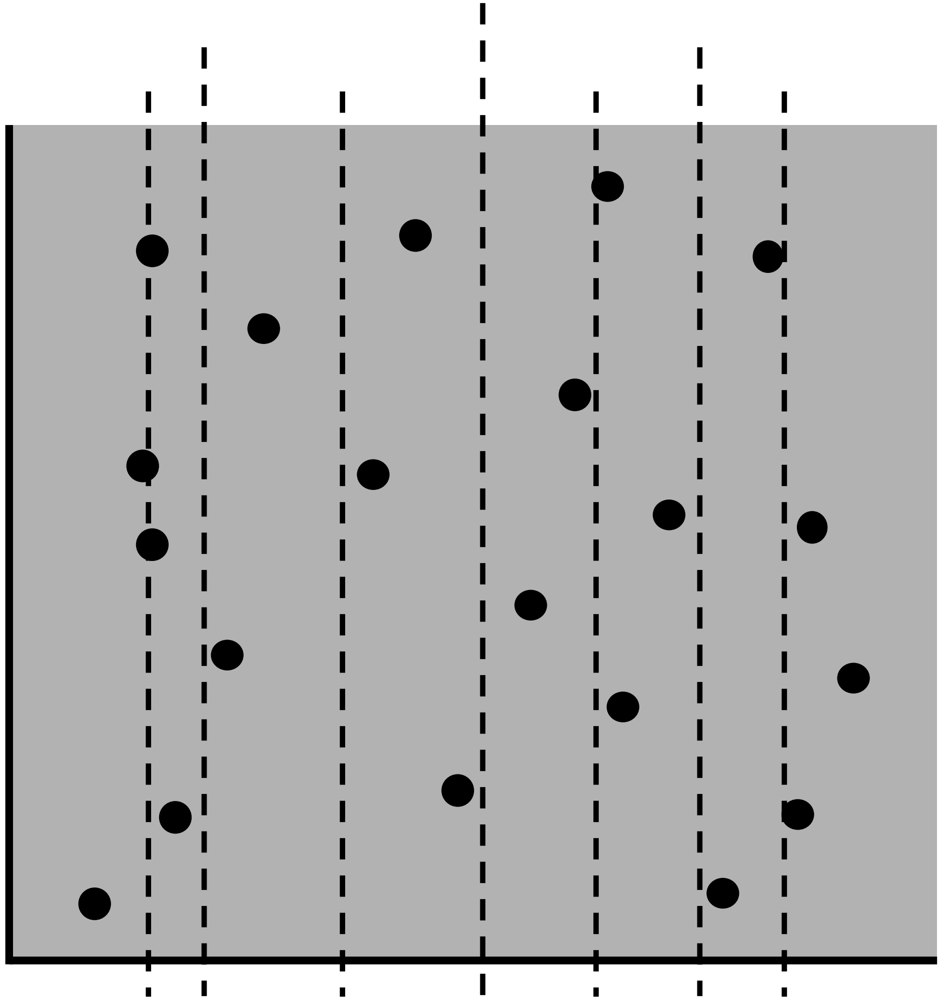
Recursive Process:

1. Divide points in half.
2. Repeat step 1 until determining d_{left} and d_{right} is trivial.

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 – Divide and Conquer



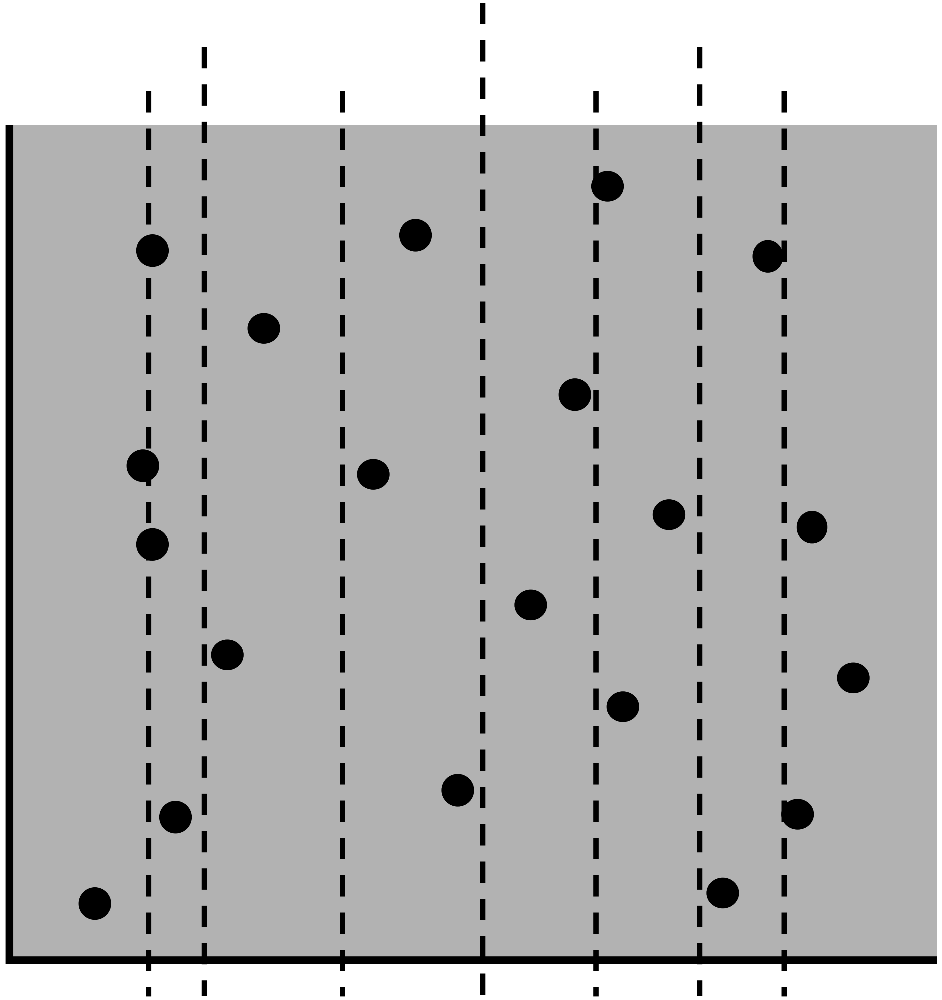
Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.

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

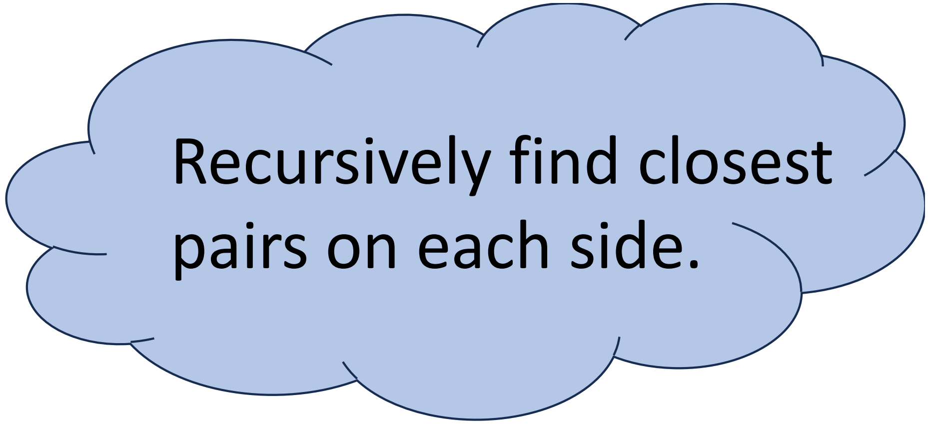
When there are one or two points on the left and right sides.

Closest Pair Problem – Divide and Conquer

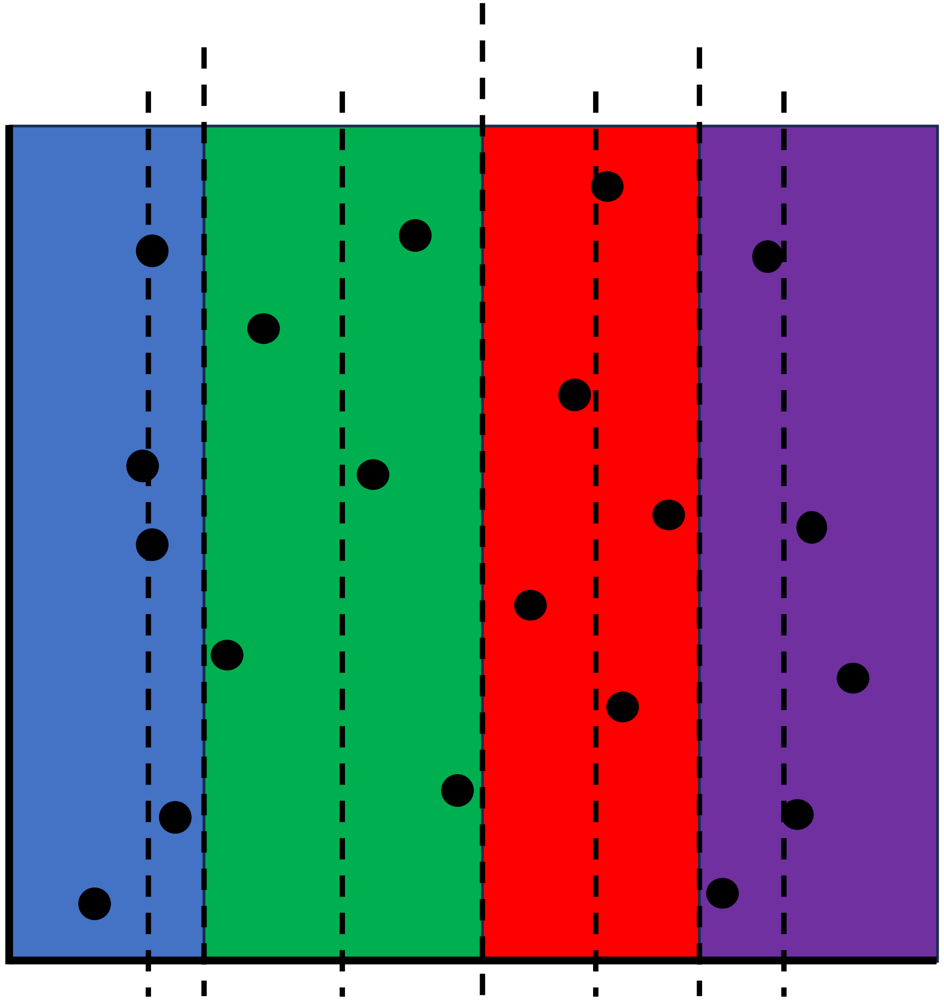


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.
3. Combine left and right sides to find closest of subproblems.



Closest Pair Problem – Divide and Conquer

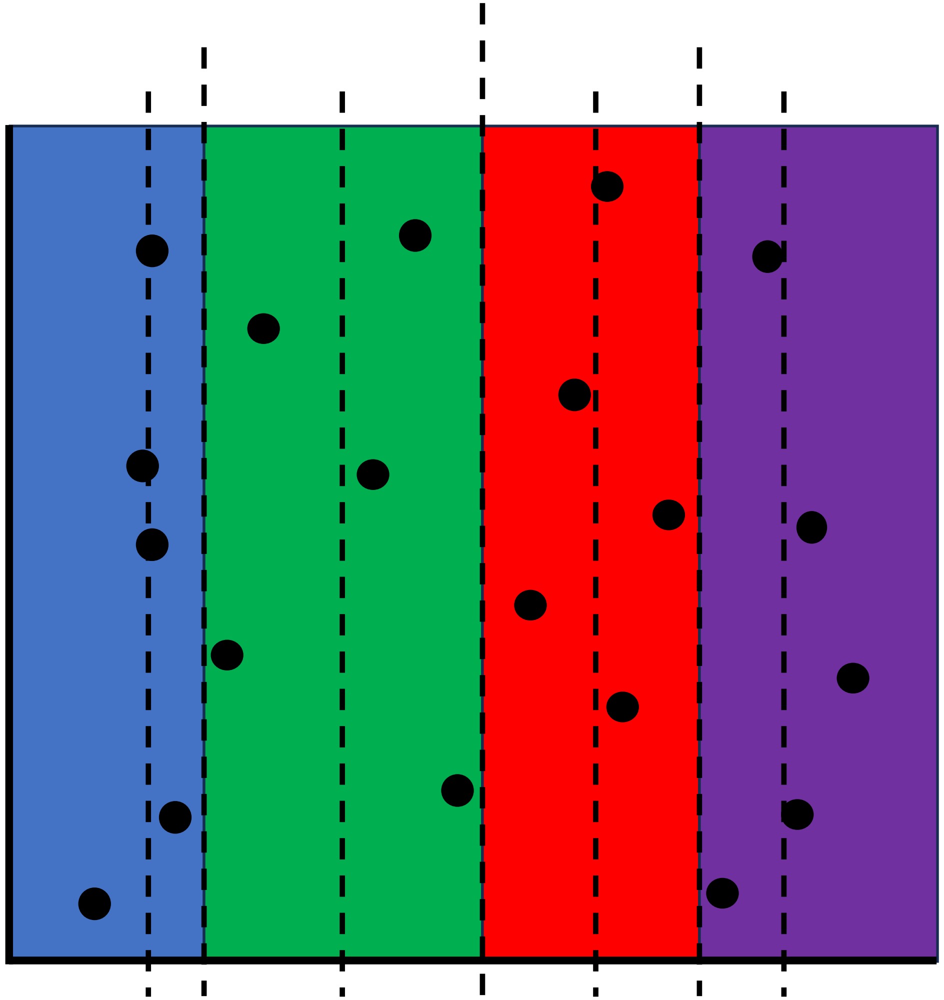


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.
3. Combine left and right sides to find closest of subproblems.

Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer

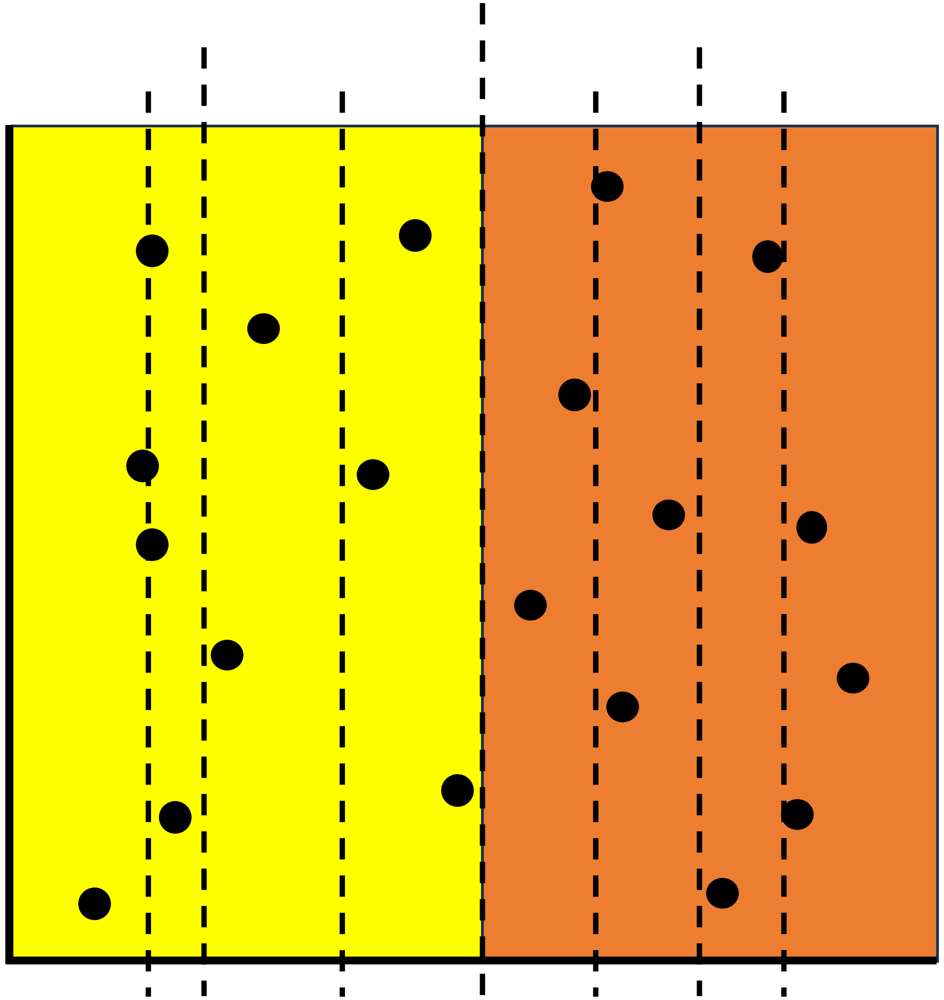


Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.
3. Combine left and right sides to find closest of subproblems.
4. Repeat until initial division is combined.

Recursively find closest pairs on each side.

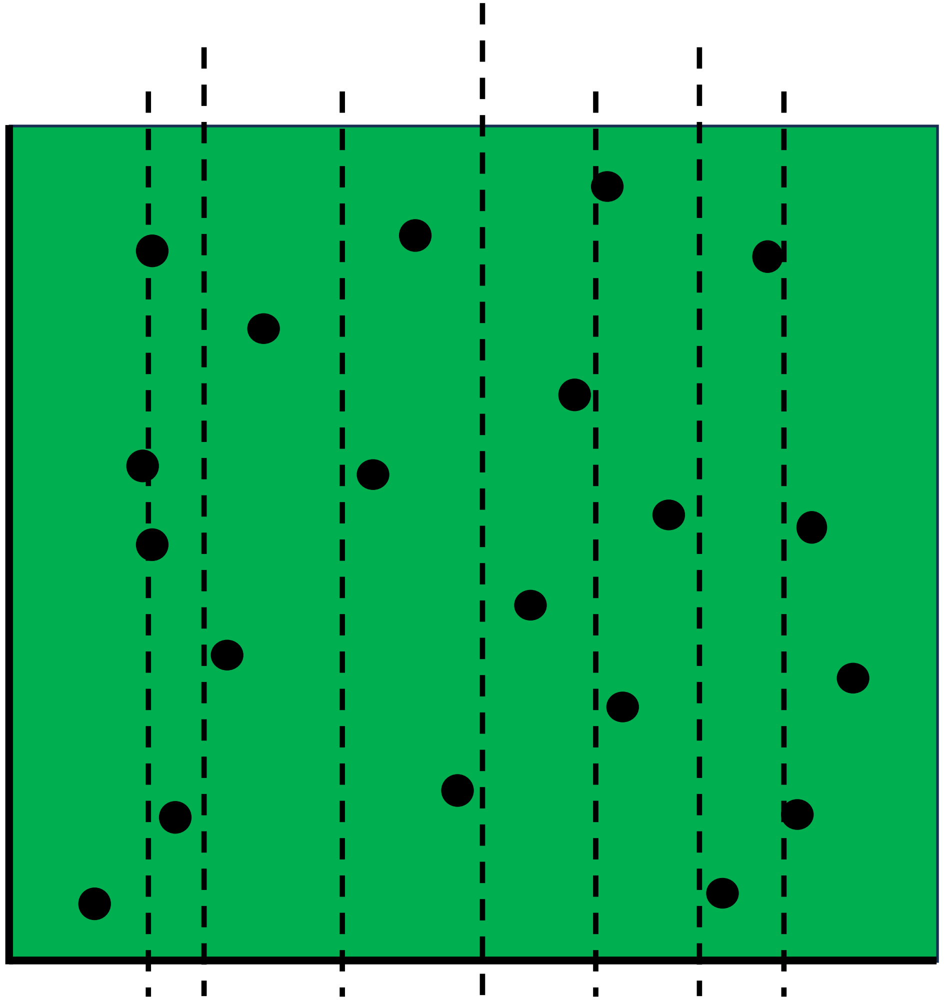
Closest Pair Problem – Divide and Conquer



Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.
3. Combine left and right sides to find closest of subproblems.
4. Repeat until initial division is combined.

Closest Pair Problem – Divide and Conquer



Recursive Process:

1. Divide points in half.
2. Repeat step 1 until there are only one or two points on each side.
3. Combine left and right sides to find closest of subproblems.
4. Repeat until initial division is combined.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. **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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

d_{left} = findClosestPair(P_{left})
d_{right} = findClosestPair(P_{right})

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Valid?

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Valid?

It's returning the distance between two points in P.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Optimal?

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Optimal?

If there was a closer pair, they would have been compared on the left side, right side, or as a straddle point.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}.
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Running Time?

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord.
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n\log n)}$
2. 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-coord. $\boldsymbol{O(n\log n)}$
6. Compare points in S to next 11 points and update δ.
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord. $\boldsymbol{O(n \log n)}$
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

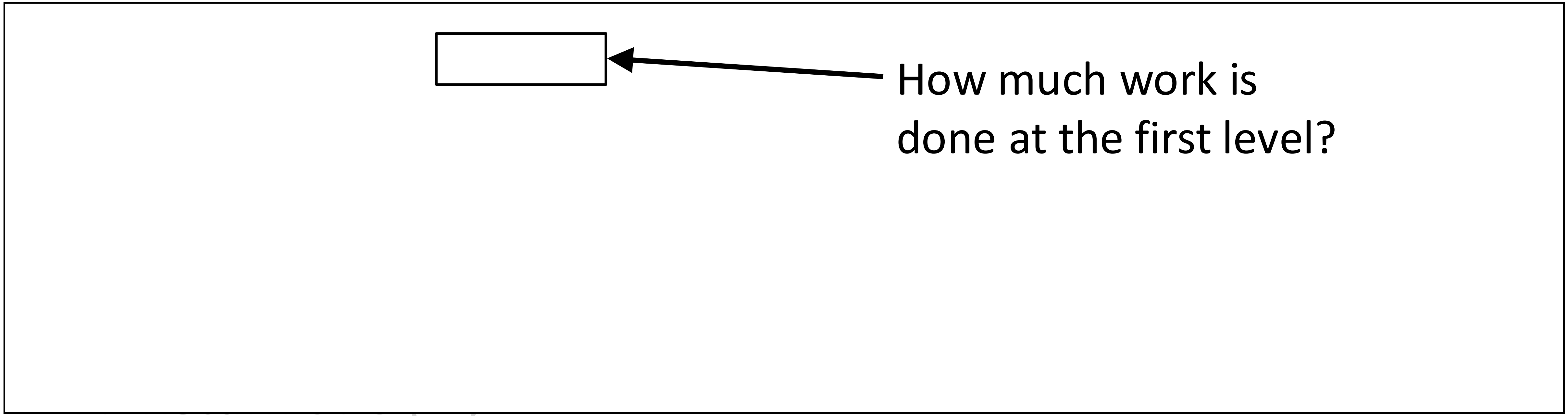
`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. 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-coord. $\boldsymbol{O(n \log n)}$
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

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n\log n)}$
2. 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-coord. $\boldsymbol{O(n\log n)}$
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

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**

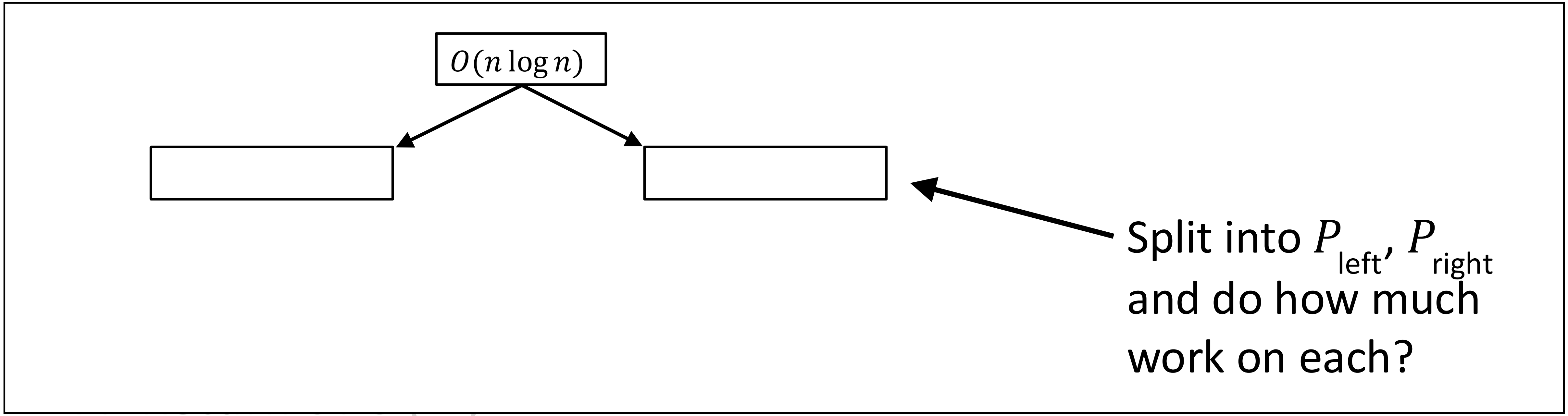
$O(n \log n)$

How much work is done at the first level?

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

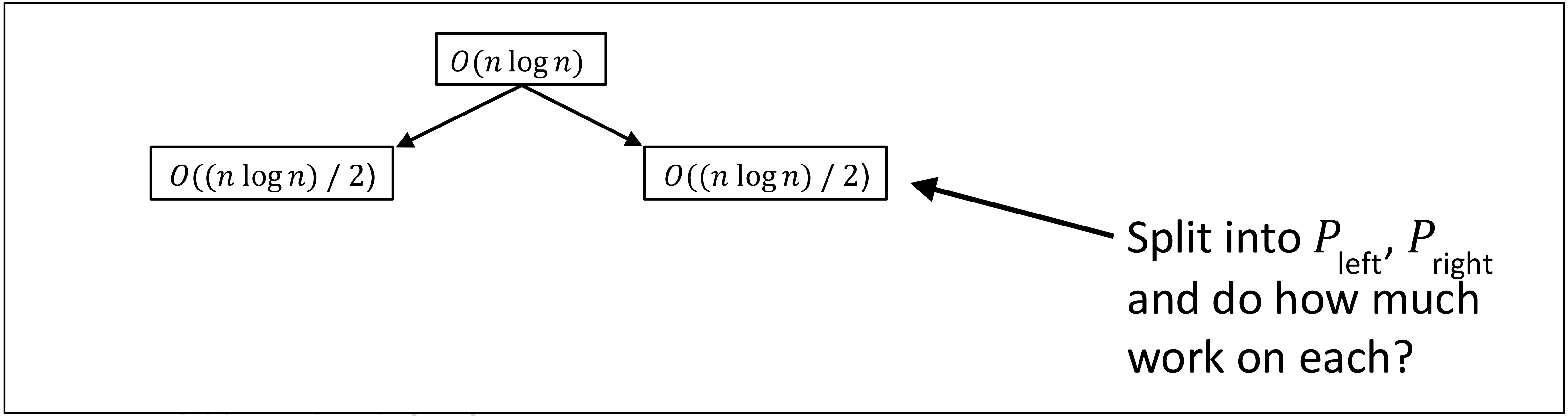
1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

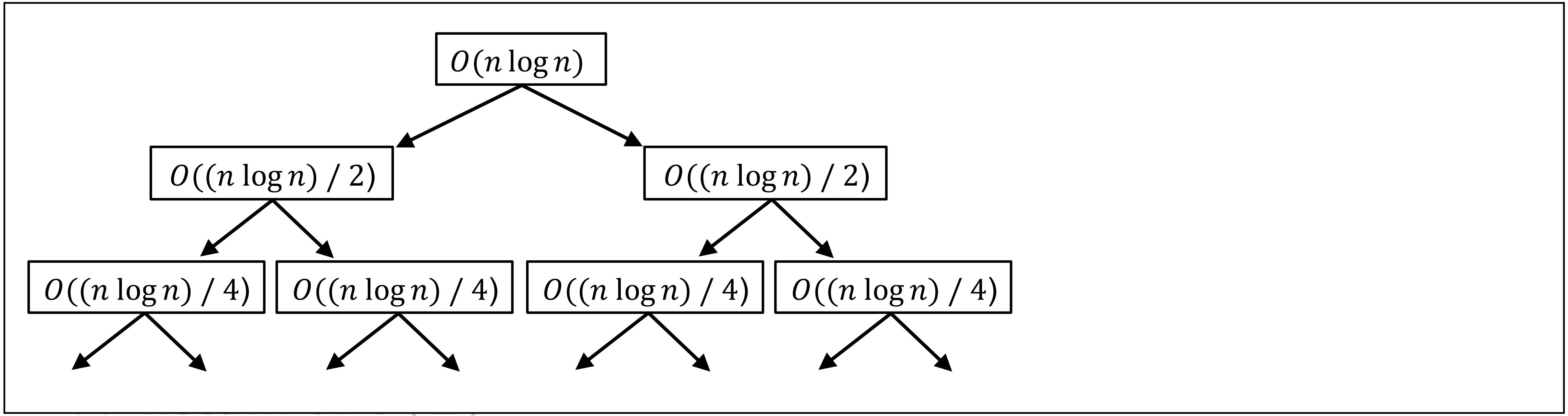
1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

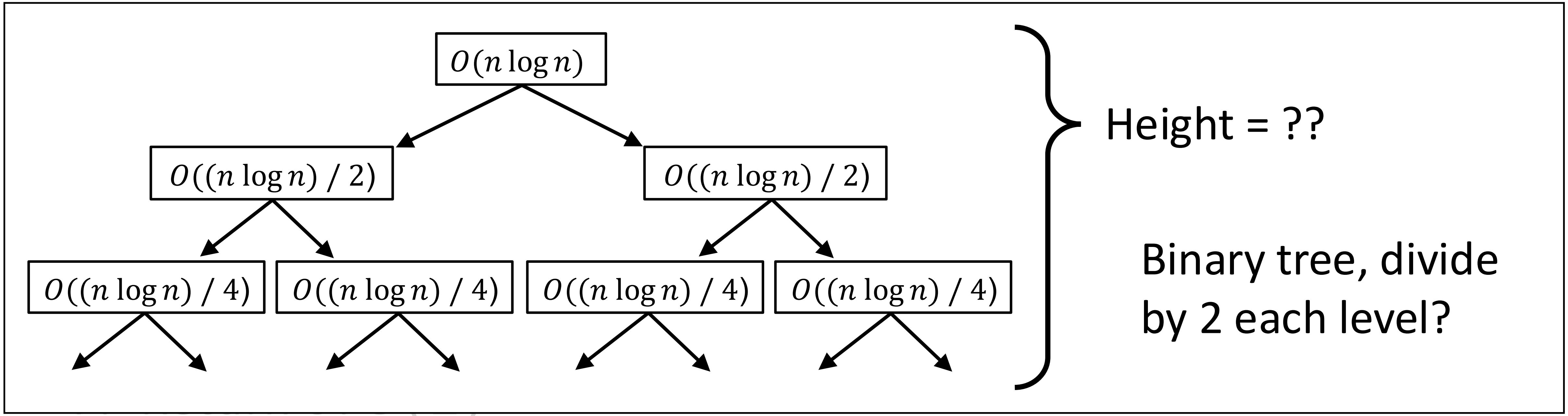
1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

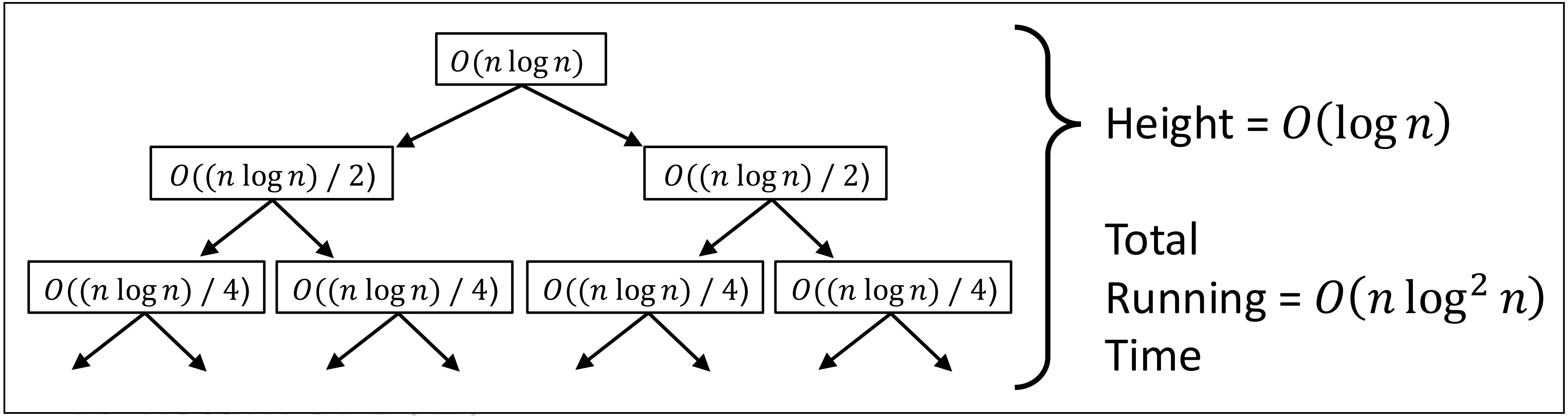
1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

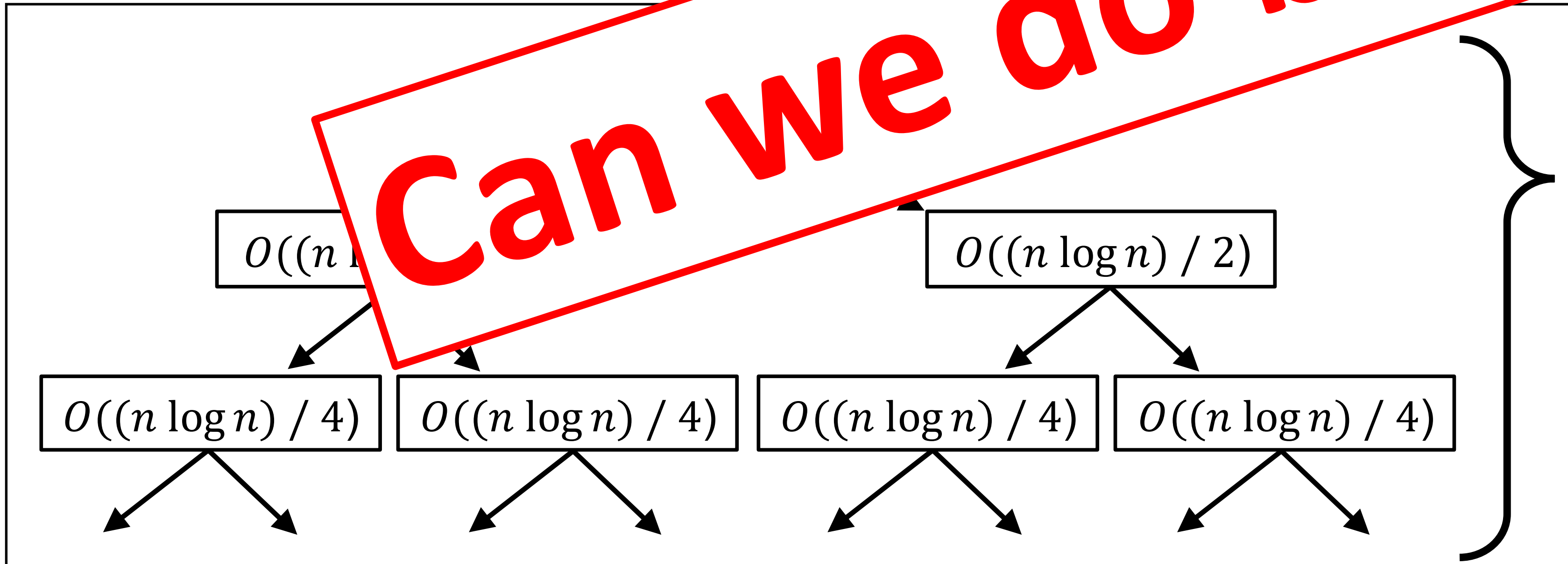
1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}. **TBD**



Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make … $\log n)$
2. Determine d_{left} and d …

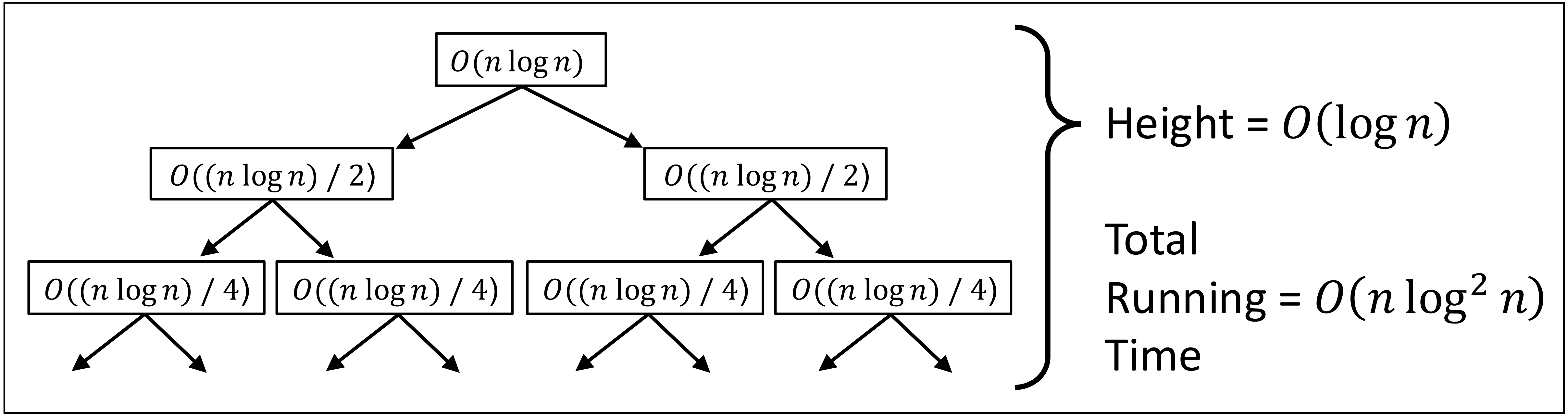


Height = $O(\log n)$

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

Can we do better?

Closest Pair Problem – Algorithm



Option 1: (Significantly) Reduce the height of the recursion tree.

Option 2: (Significantly) Reduce the amount of work done at each level.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

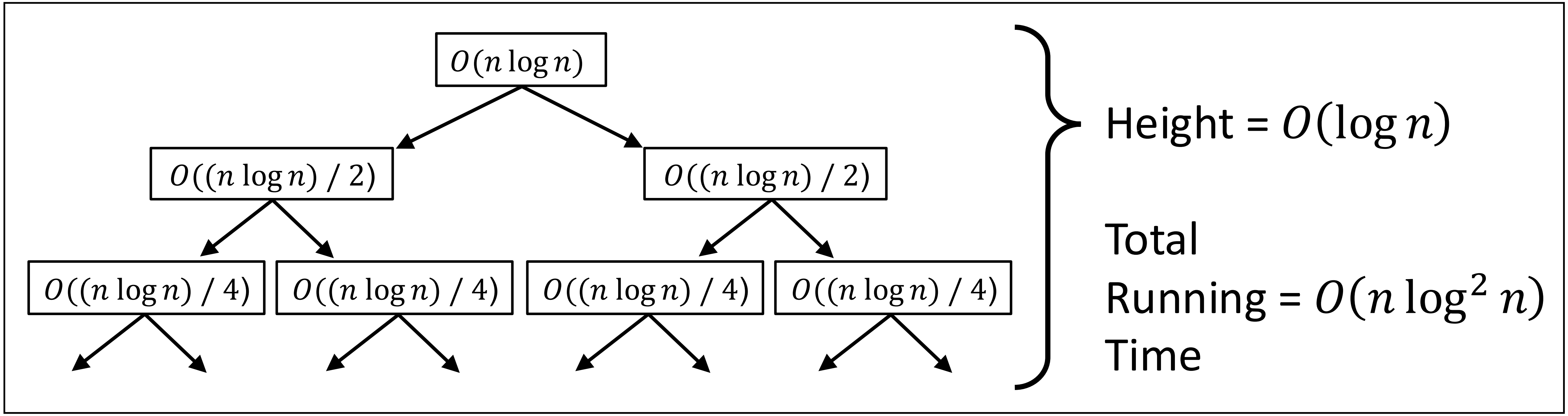
Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

Maybe we don't need to sort so often??

Closest Pair Problem – Algorithm



Plan:

- Presort by x-coordinate (X)
- Presort by y-coordinate (Y)
- Split X and Y by comparing to L.

Closest Pair Problem – Algorithm

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
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 points by x-coordinate (X) and y-coordinate (Y).

```
findClosestPair(P):
```

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
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 points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n\log n)}$

findClosestPair(P):

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n\log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n\log n)}$
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 points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$

`findClosestPair(`P`):`

1. Sort points by x-coord, find L, make P_{left}, P_{right}. $\boldsymbol{O(n \log n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
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 points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n\,log\,n)}$

`findClosestPair(`P`):`

1. ~~Sort points by x-coord,~~ find L, split X, Y. $\boldsymbol{O(n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
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 points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$

`findClosestPair(`P`):`

1. ~~Sort points by x-coord,~~ find L, split X, Y. $\boldsymbol{O(n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$
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 points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$

`findClosestPair(`P`):`

1. ~~Sort points by x-coord,~~ find L, split X, Y. $\boldsymbol{O(n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$~~
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

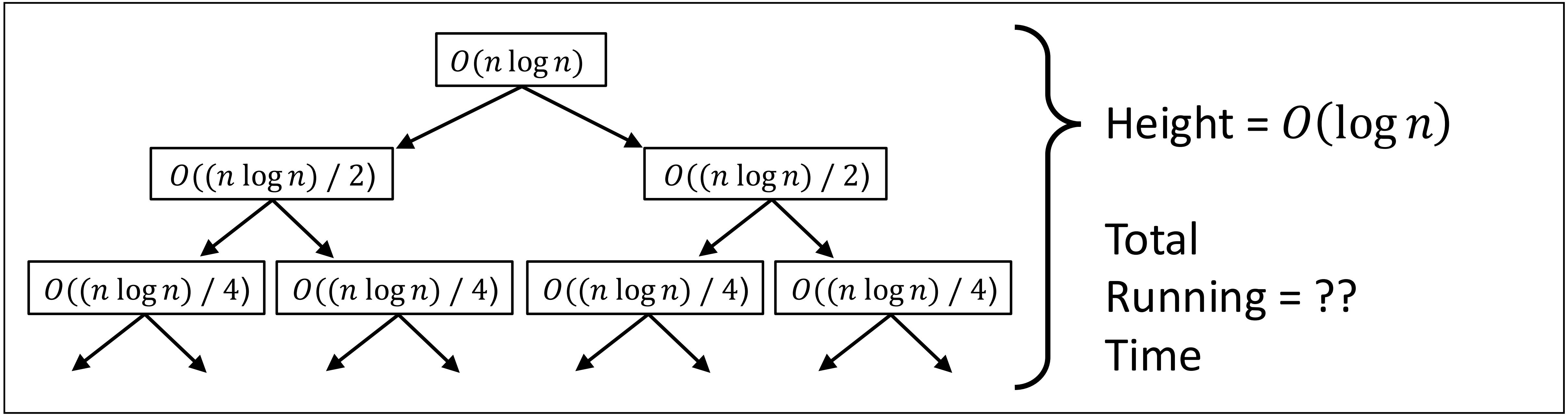
Closest Pair Problem – Algorithm

0. Sort points by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$

`findClosestPair(`P`):`

1. ~~Sort points by x-coord,~~ find L, split X, Y. $\boldsymbol{O(n)}$
2. Determine d_{left} and d_{right}.
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-coord. $\boldsymbol{O(n \log n)}$~~
6. Compare points in S to next 11 points and update δ. $\boldsymbol{O(n)}$
7. Return δ. $\boldsymbol{O(1)}$

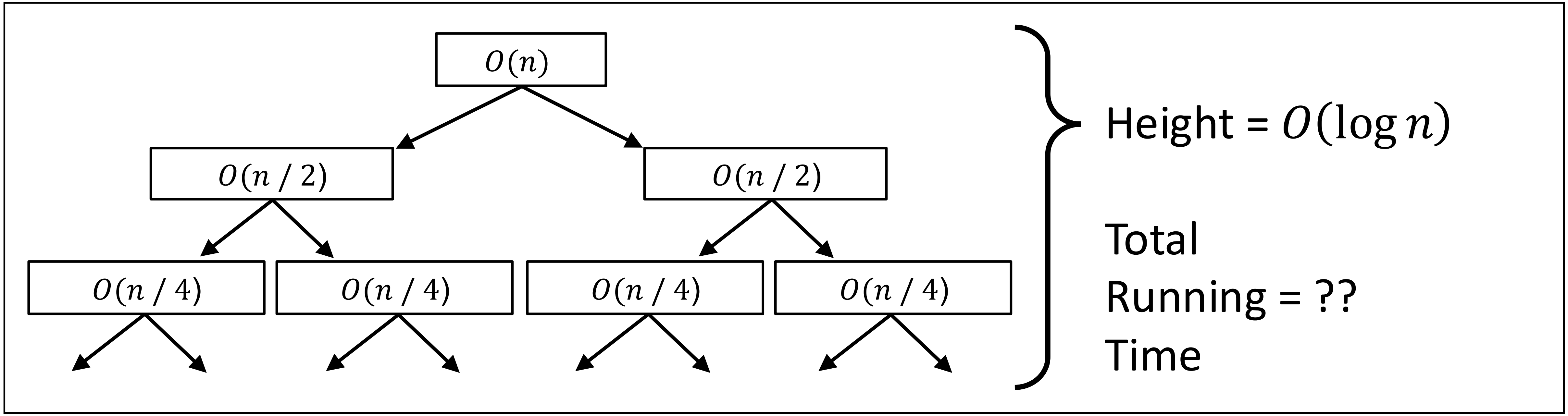
Closest Pair Problem – Algorithm



Plan:

- Presort by x-coordinate (X)
- Presort by y-coordinate (Y)
- Split X and Y by comparing to L.

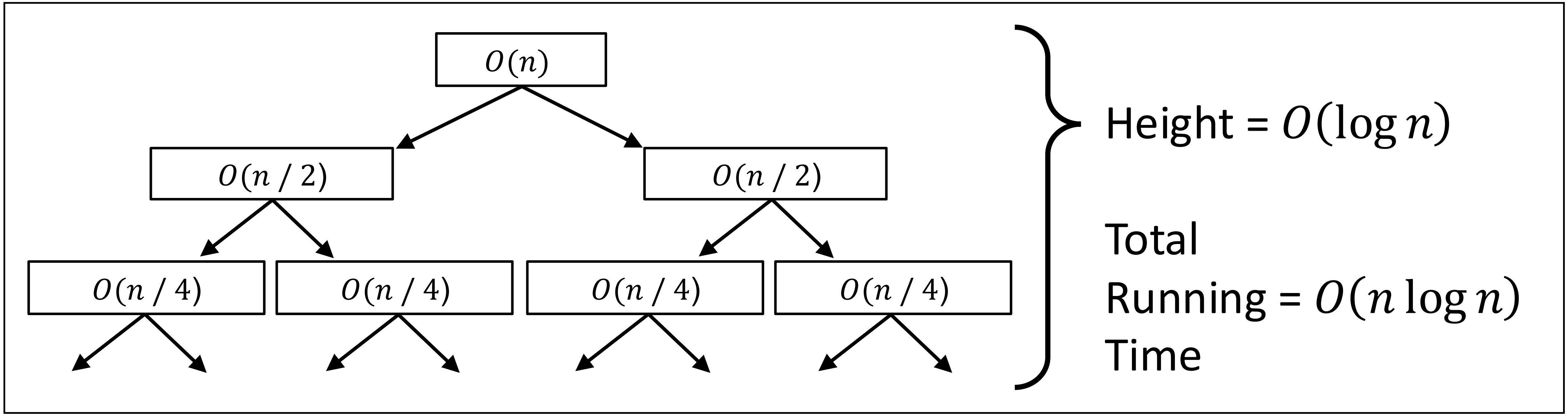
Closest Pair Problem – Algorithm



Plan:

- Presort by x-coordinate (X)
- Presort by y-coordinate (Y)
- Split X and Y by comparing to L.

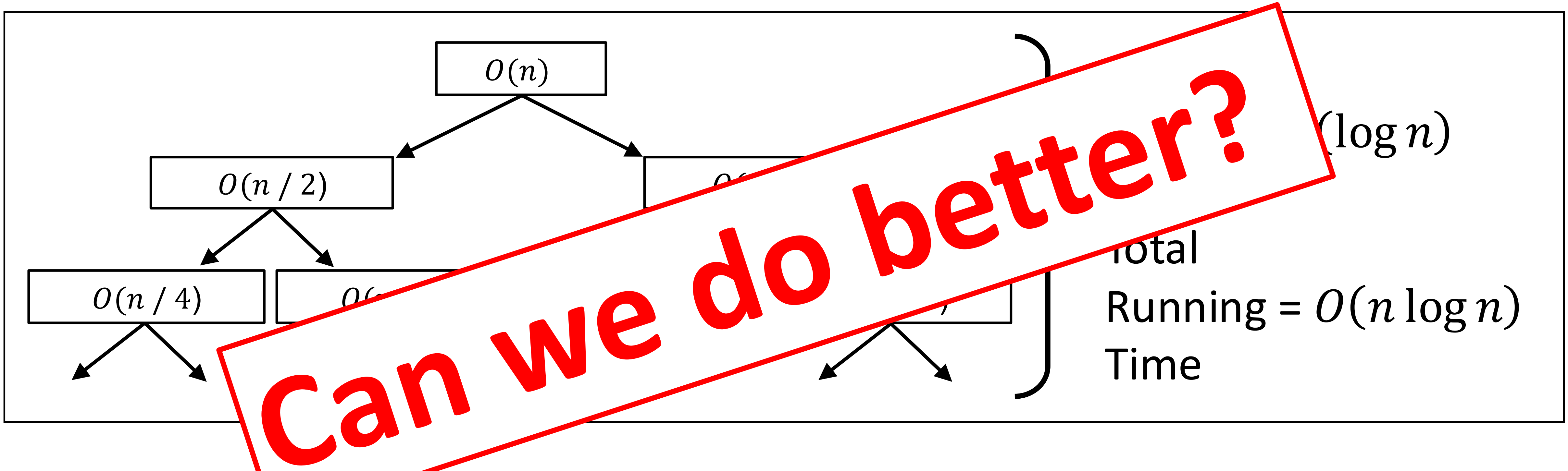
Closest Pair Problem – Algorithm



Plan:

- Presort by x-coordinate (X)
- Presort by y-coordinate (Y)
- Split X and Y by comparing to L.

Closest Pair Problem – Algorithm



Plan:

- Presort by x-coordinate (X)
- Presort by y-coordinate (Y)
- Split X and Y by comparing to L.