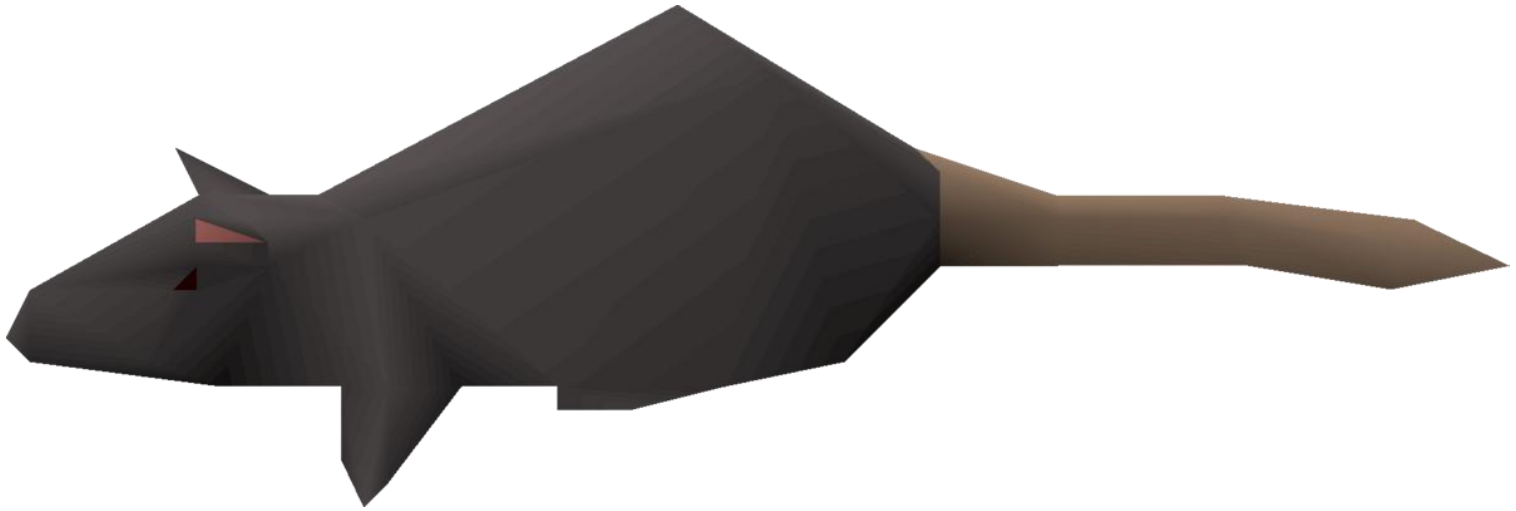
CSCI 232:
Data Structures and Algorithms

Divide and Conquer

Reese Pearsall

Spring 2025

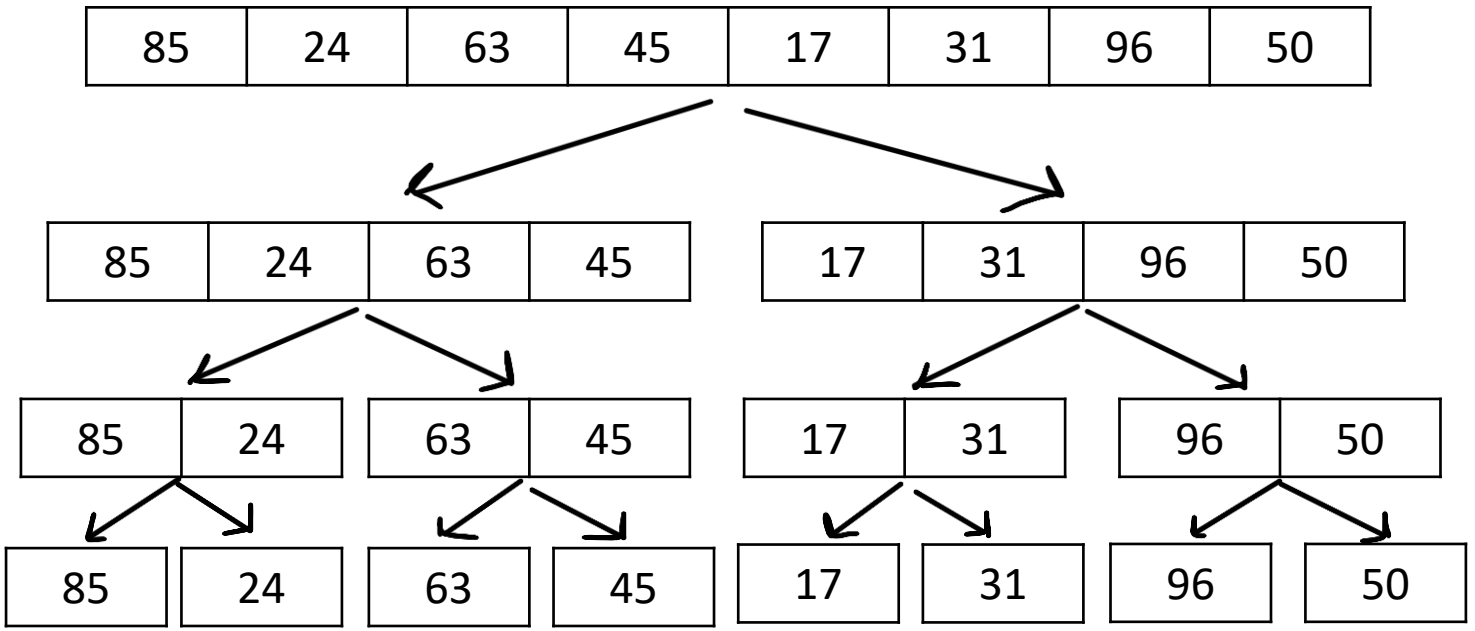


Would you rather fight 1000 rats all at once, or 1000 rats one after another?

Divide and Conquer is an algorithm technique that involves breaking down the problem into smaller subproblems (*divide*), which are solved independently, and then combined to solve the original problem (*conquer*)

In some cases, it easier to solve several smaller subproblems, and combine their results instead of solving one big problem

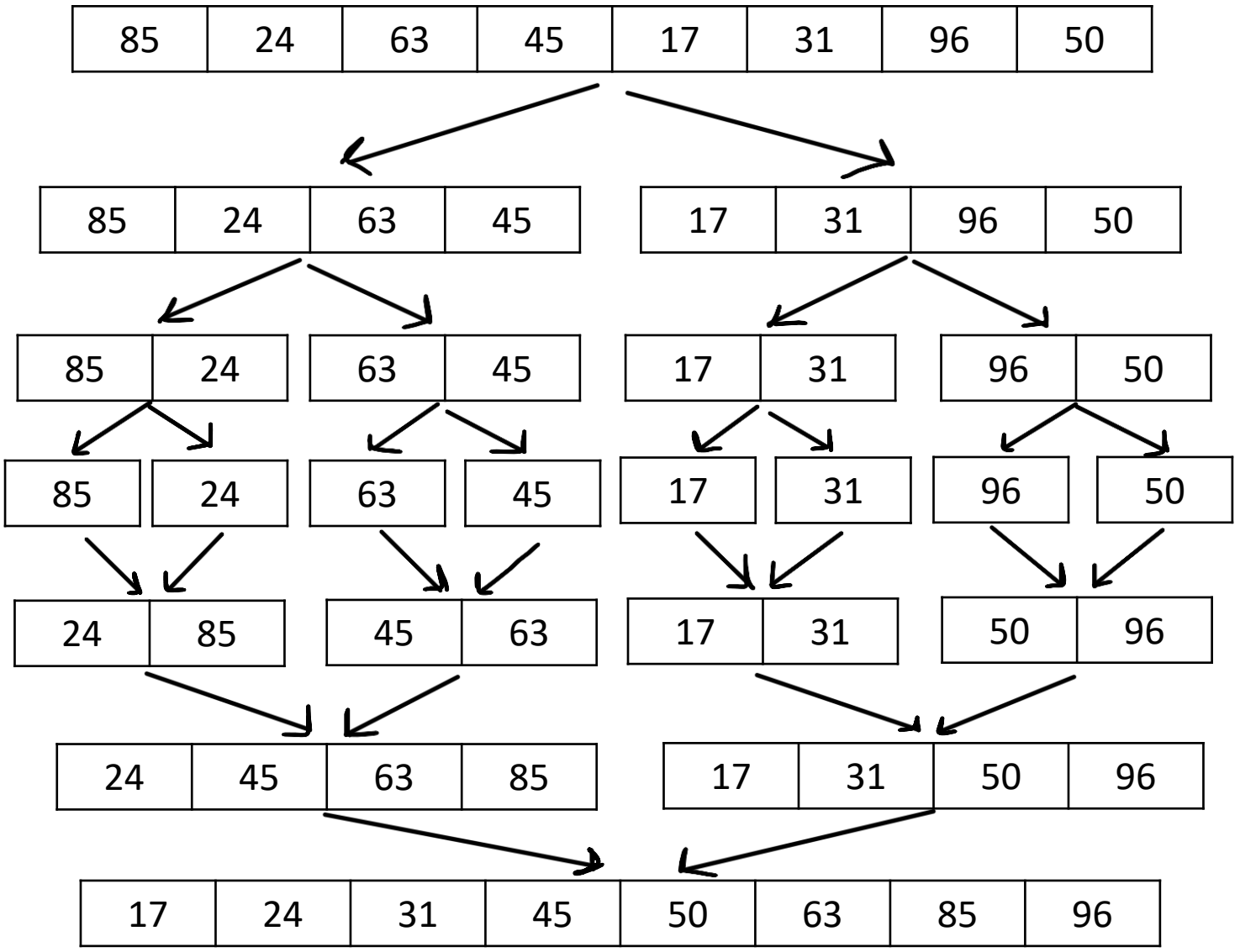
Merge sort is a prime example of divide and conquer



1. Divide: Split problems into two (roughly) equal parts

2. Conquer: sort the parts

Merge sort is a prime example of divide and conquer

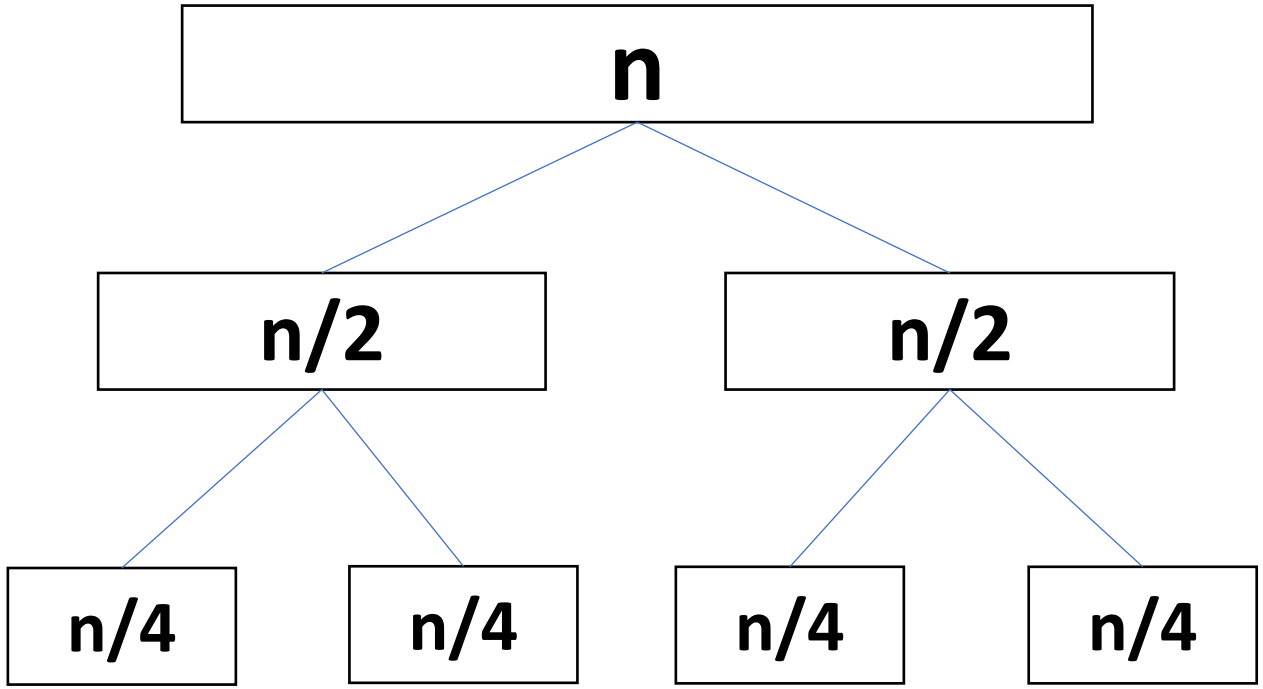


1. Divide: Split problems into two (roughly) equal parts

2. Conquer: sort the parts

3. Combine: merge the sorted parts

Recursion tree



Work done at each level → O(n)

Height of tree → log(n)

Total running time: **O(nlogn)**

Divide and Conquer Running time

Recursive divide and conquer running time can be characterized as:

$$\mathbf{T(n) = aT(n/b) + D(n) + C(n)}$$

T(n) → total running time of divide and conquer algorithm

Divide and Conquer Running time

Recursive divide and conquer running time can be characterized as:

$$T(n) = aT(n/b) + D(n) + C(n)$$

T(n) → total running time of divide and conquer algorithm

Running time T(n) references another instance of T(n) → **Recurrence relation**

Divide and Conquer Running time

Recursive divide and conquer running time can be characterized as:

$$\mathbf{T(n) = aT(n/b) + D(n) + C(n)}$$

$\mathbf{T(n)}$ → total running time of divide and conquer algorithm

$\mathbf{a}$ – number of subproblems relative to previous

$\mathbf{n/b}$ – size of subproblem relative to previous

$\mathbf{D(n)}$ – running time to divide problems

$\mathbf{C(n)}$ – running time to combine problems

Divide and Conquer Running time

Recursive divide and conquer running time can be characterized as:

$$\mathbf{T(n) = aT(n/b) + D(n) + C(n)}$$

$\mathbf{T(n)}$ → total running time of divide and conquer algorithm

$\mathbf{a}$ – number of subproblems relative to previous

$\mathbf{n/b}$ – size of subproblem relative to previous

$\mathbf{D(n)}$ – running time to divide problems

$\mathbf{C(n)}$ – running time to combine problems

Master theorem: If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$ for constants $a \geq 1$, $b > 1$, $d \geq 0$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

Divide and Conquer Running time

Recursive divide and conquer running time can be characterized as:

$$\mathbf{T(n) = aT(n/b) + D(n) + C(n)}$$

$\mathbf{T(n)}$ → total running time of divide and conquer algorithm

$\mathbf{a}$ – number of subproblems relative to previous

$\mathbf{n/b}$ – size of subproblem relative to previous

$\mathbf{D(n)}$ – running time to divide problems

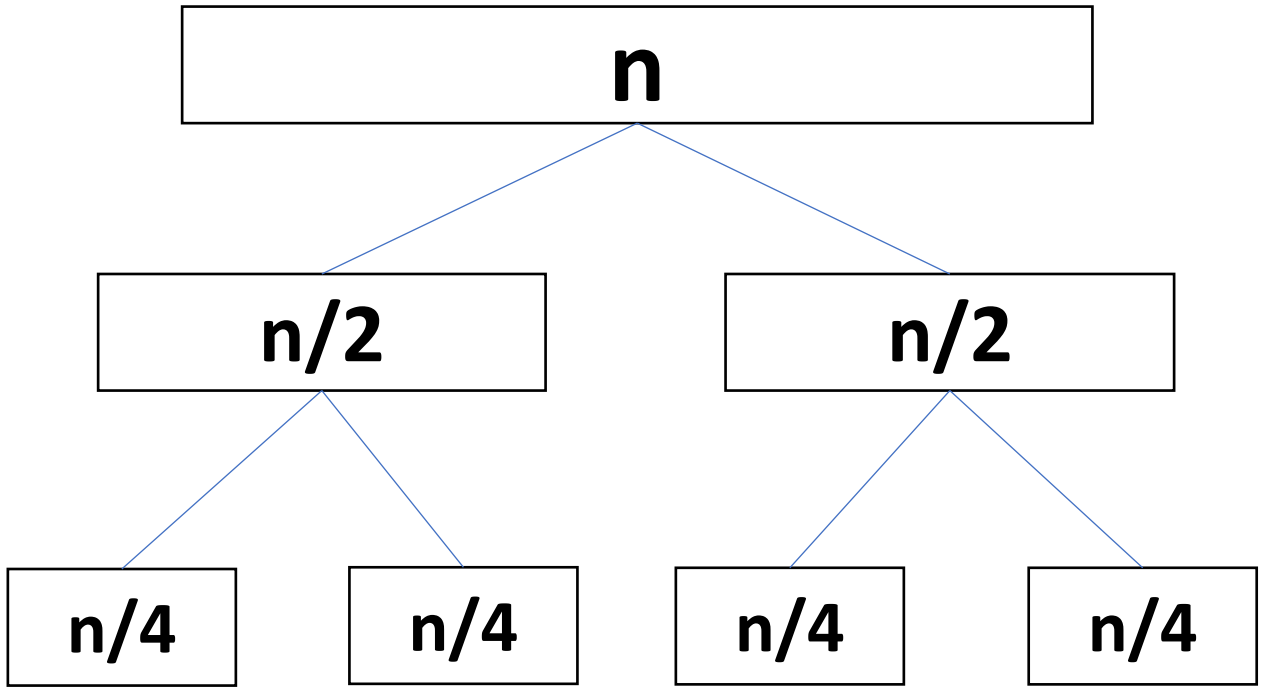
$\mathbf{C(n)}$ – running time to combine problems

Master theorem: If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$ for constants $a \geq 1, b > 1, d \geq 0$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

"If the time used outside of recursion is greater than the work done recursively at each level, the running time is dominated by the work for splitting/merging/combining"

Merge Sort Running Time



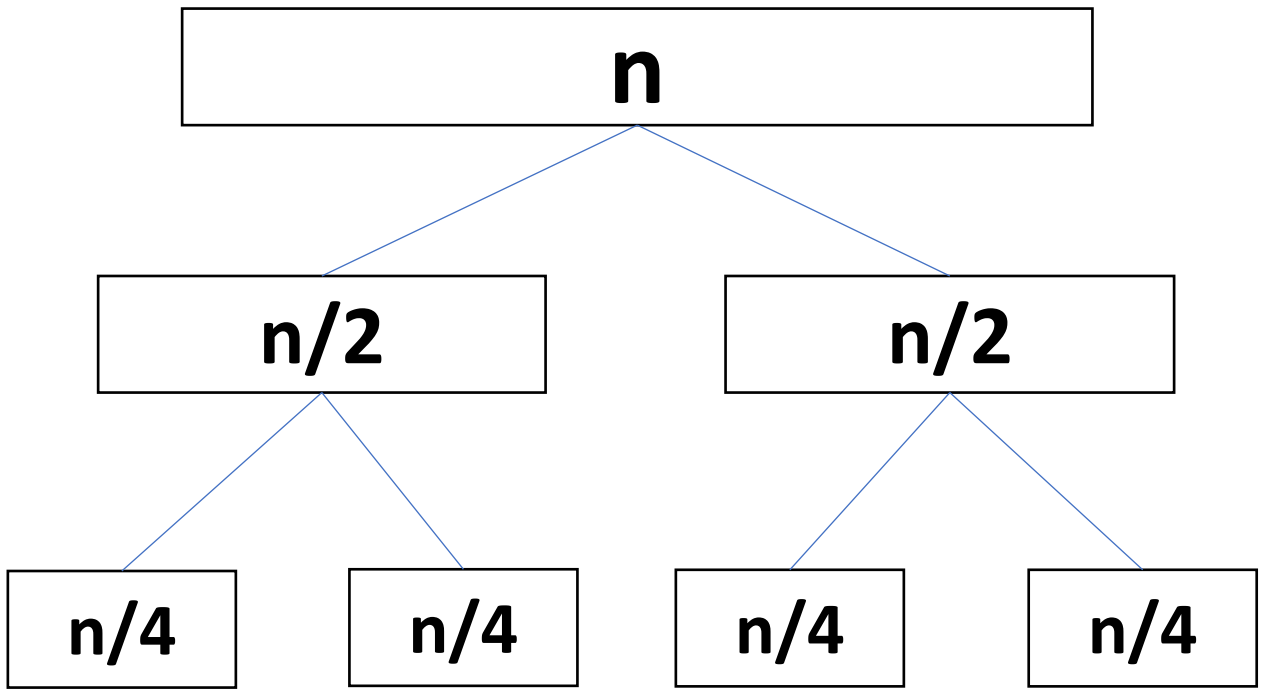
a – number of subproblems relative to previous

n/b – size of subproblem relative to previous

D(n) – running time to divide problems

C(n) – running time to combine problems

Merge Sort Running Time



a – number of subproblems relative to previous

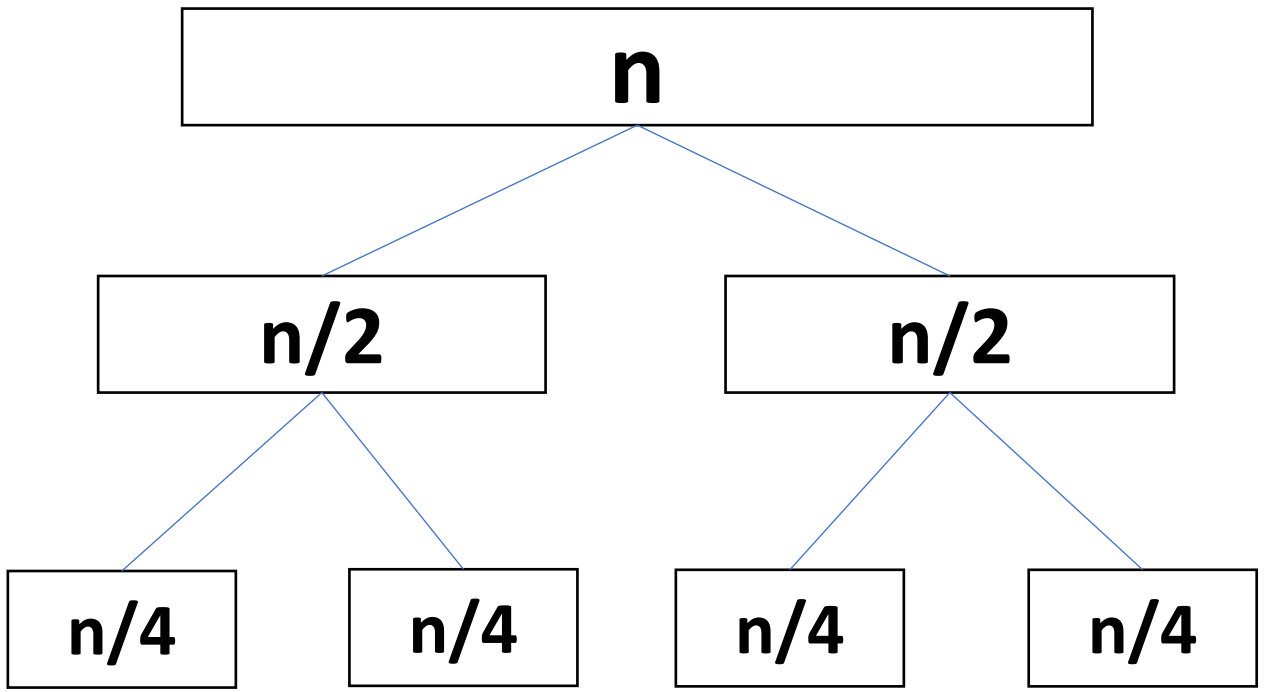
2

n/b – size of subproblem relative to previous

D(n) – running time to divide problems

C(n) – running time to combine problems

Merge Sort Running Time



a – number of subproblems relative to previous

2

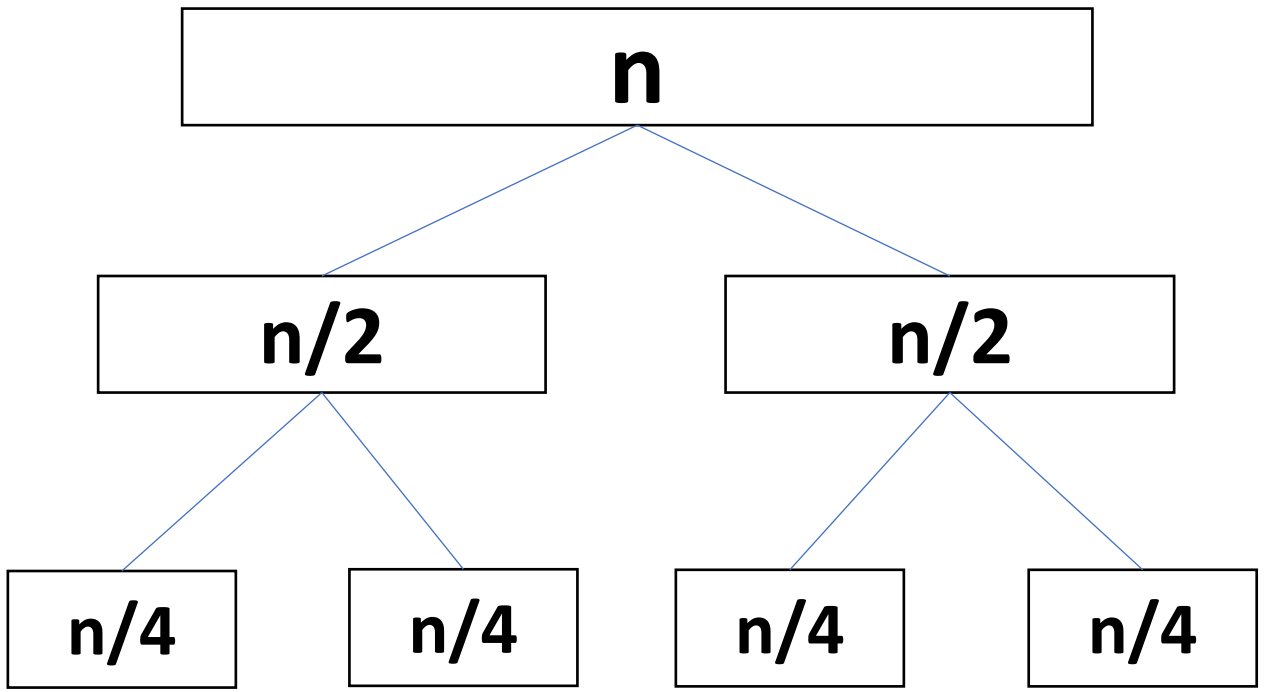
n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

C(n) – running time to combine problems

Merge Sort Running Time



a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

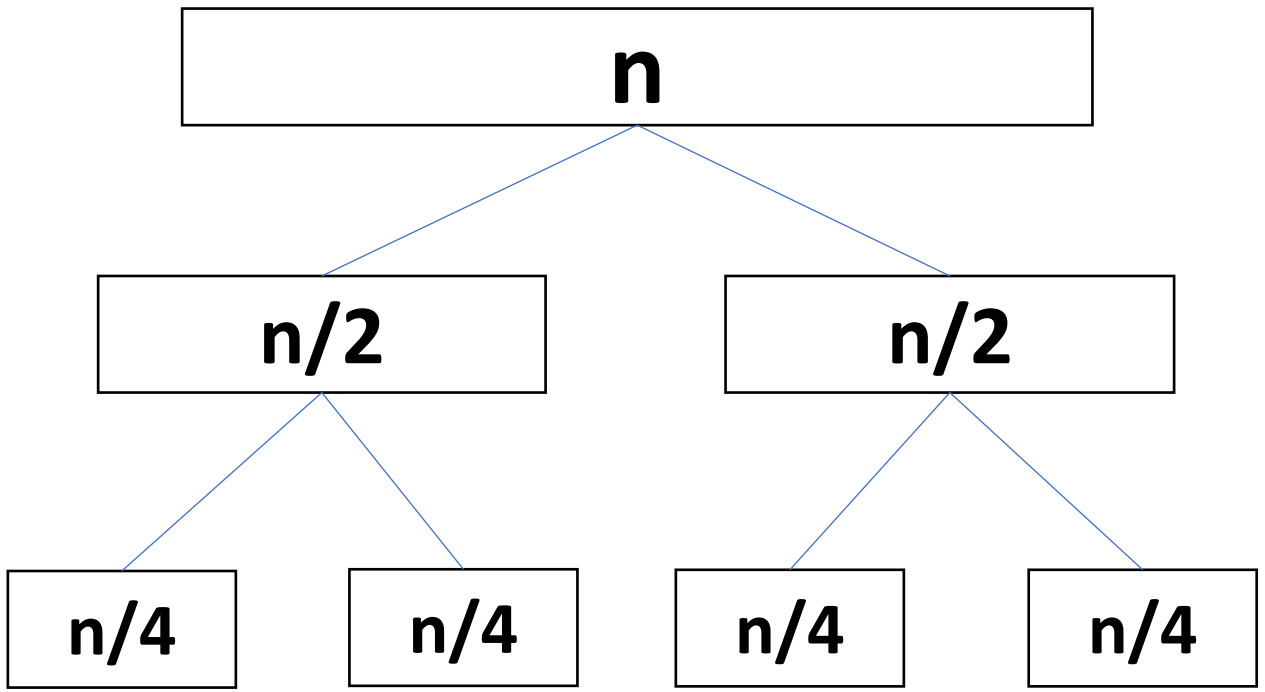
n/2

D(n) – running time to divide problems

O(n)*

C(n) – running time to combine problems

Merge Sort Running Time



a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

O(n)*

C(n) – running time to combine problems

O(n)

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

O(n)*

C(n) – running time to combine problems

O(n)

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

O(n)*

C(n) – running time to combine problems

O(n)

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

   $\log_b a = \log_2 2 = 1$

"Rate of growth"

How deep and bushy the recursion gets

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

O(n) *

C(n) – running time to combine problems

O(n)

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$
2. Determine $O(n^{\mathrm{d}})$

$\log_b a = \log_2 2 = 1$

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

$n/2$

D(n) – running time to divide problems

$O(n)^*$

C(n) – running time to combine problems

$O(n)$

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

$\log_b a = \log_2 2 = 1$

2. Determine $O(n^{\mathrm{d}})$

$O(n) + O(n) \in O(n^1)$

d = 1

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

$n/2$

D(n) – running time to divide problems

$O(n)^*$

C(n) – running time to combine problems

$O(n)$

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

   $\log_b a = \log_2 2 = 1$

2. Determine $O(n^{\mathrm{d}})$

   $O(n) + O(n) \in O(n^1)$

   d = 1

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

O(n) *

C(n) – running time to combine problems

O(n)

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

   $\log_b a = \log_2 2 = 1$

2. Determine $O(n^{\mathrm{d}})$

   $O(n) + O(n) \in O(n^1)$

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

$n/2$

D(n) – running time to divide problems

$O(n)^*$

C(n) – running time to combine problems

$O(n)$

Merge Sort Running Time

Master theorem:

If $T(n) = aT(n/b) + O(n^{\mathrm{d}})$, then:

$$T(n) \in \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

1. Calculate $\log_b a$

$\log_b a = \log_2 2 = 1$

2. Determine $O(n^{\mathrm{d}})$

$O(n) + O(n) \in O(n^1)$

"Running time of merge sort"

$T(n) \in O(n^{\mathrm{d}} \log n)$

$T(n) \in O(n \log n)$ ✓

a – number of subproblems relative to previous

2

n/b – size of subproblem relative to previous

n/2

D(n) – running time to divide problems

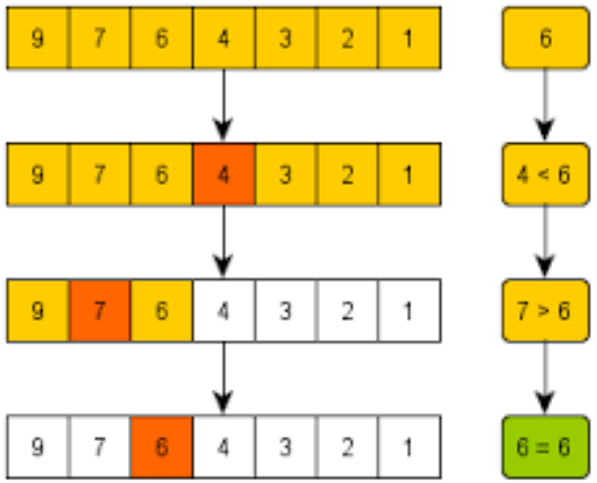
O(n) *

C(n) – running time to combine problems

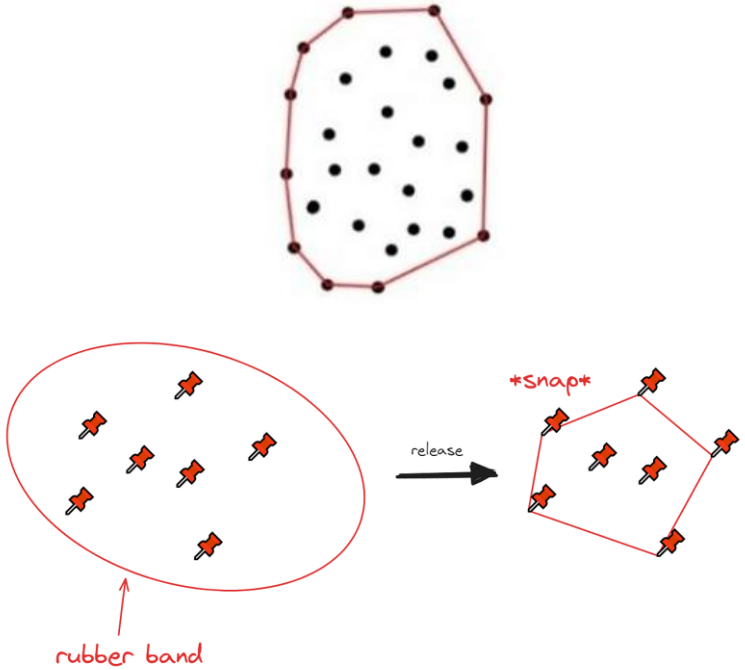
O(n)

Divide and Conquer examples

Binary Search



Convex Hull



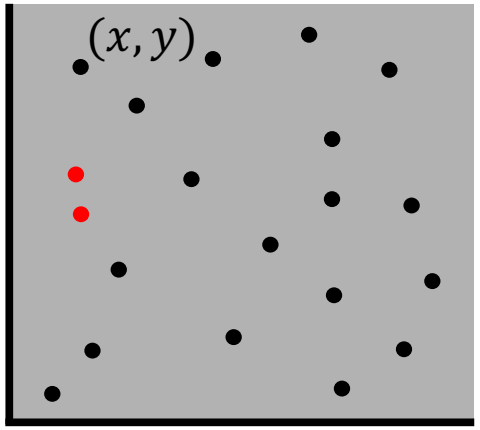
Matrix Multiplication

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

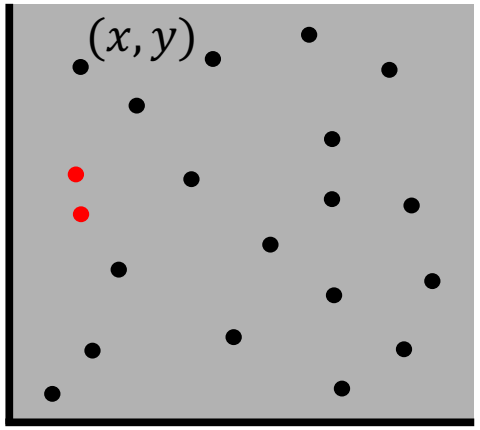
Closest Pair Problem



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

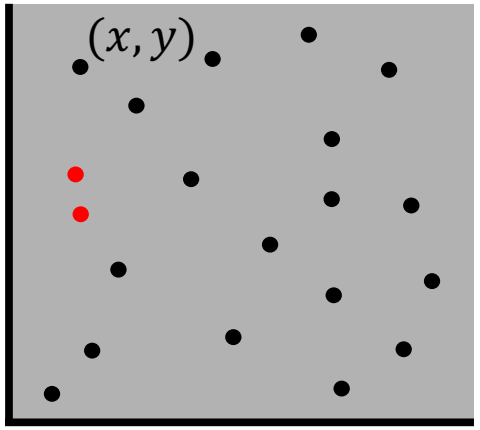


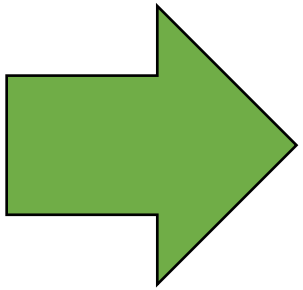
Given n points, find a pair of points with the smallest distance between them.

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

Algorithm?

Closest Pair Problem





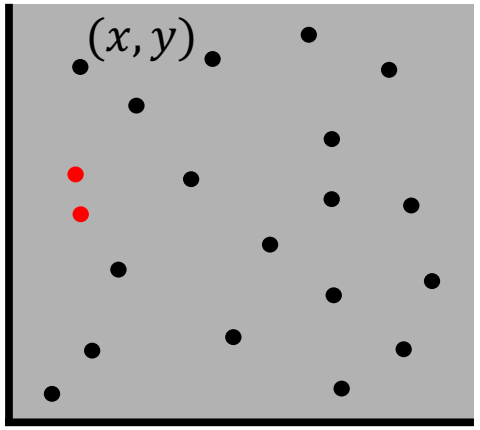
	P_1	P_2	...	P_n
P_1	/	$d_{1,2}$	...	$d_{1,n}$
P_2	$d_{2,1}$	/	...	$d_{2,n}$
...	...	...	...	...
P_n	$d_{n,1}$	$d_{n,2}$	...	/

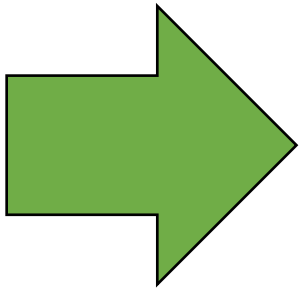
Simple solution:

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

Running Time = ?

Closest Pair Problem





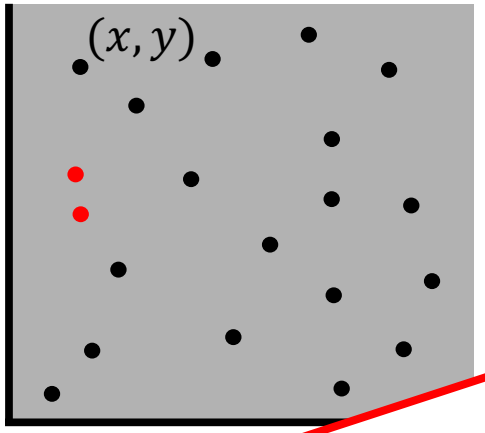
	P_1	P_2	...	P_n
P_1	/	$d_{1,2}$	...	$d_{1,n}$
P_2	$d_{2,1}$	/	...	$d_{2,n}$
...	...	...	...	...
P_n	$d_{n,1}$	$d_{n,2}$	...	/

Simple solution:

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

Running Time = $O(n^2)$

Closest Pair Problem



Can we do better?

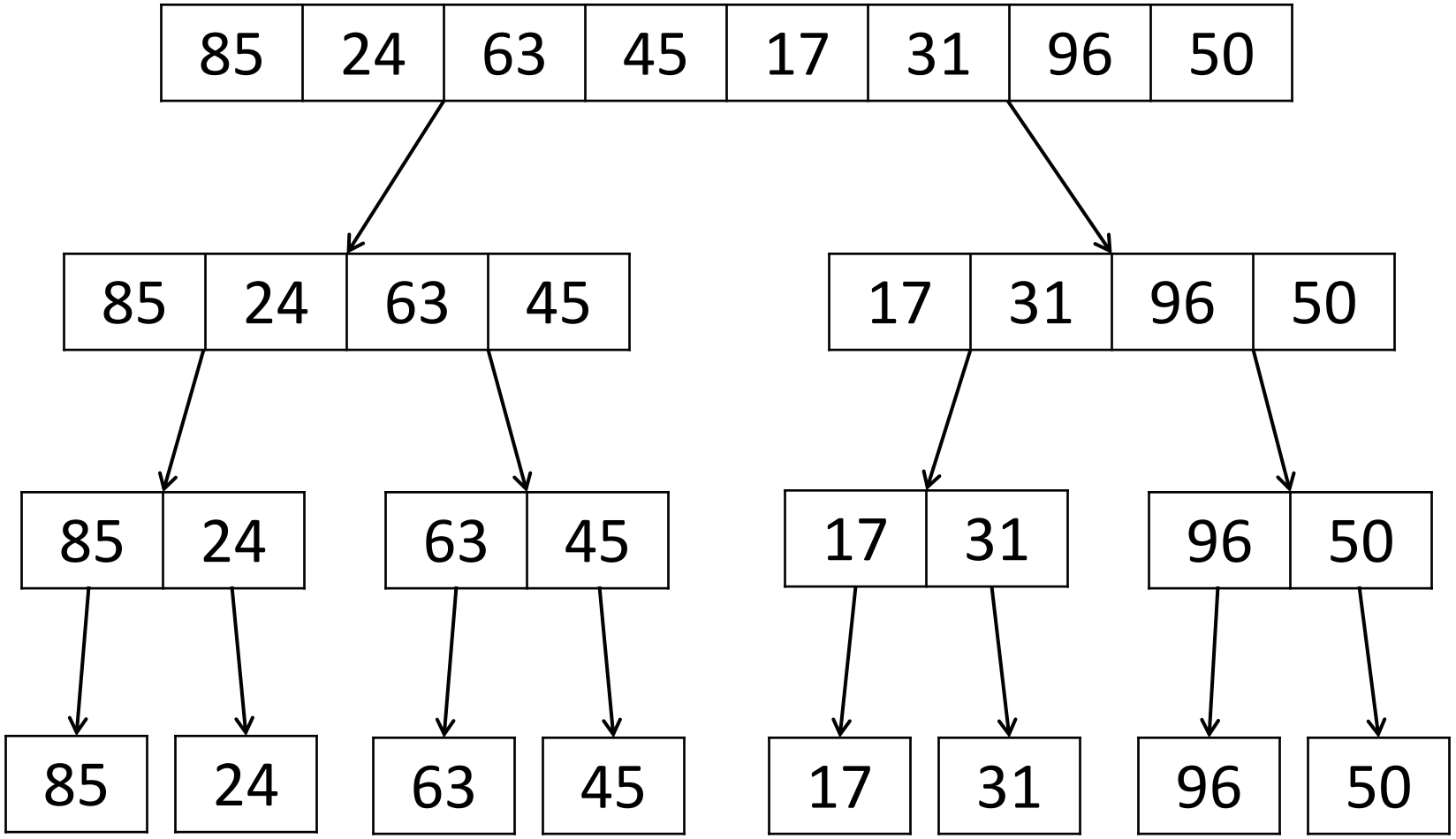
1. Compute distance for each pair.
2. Select smallest.

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.

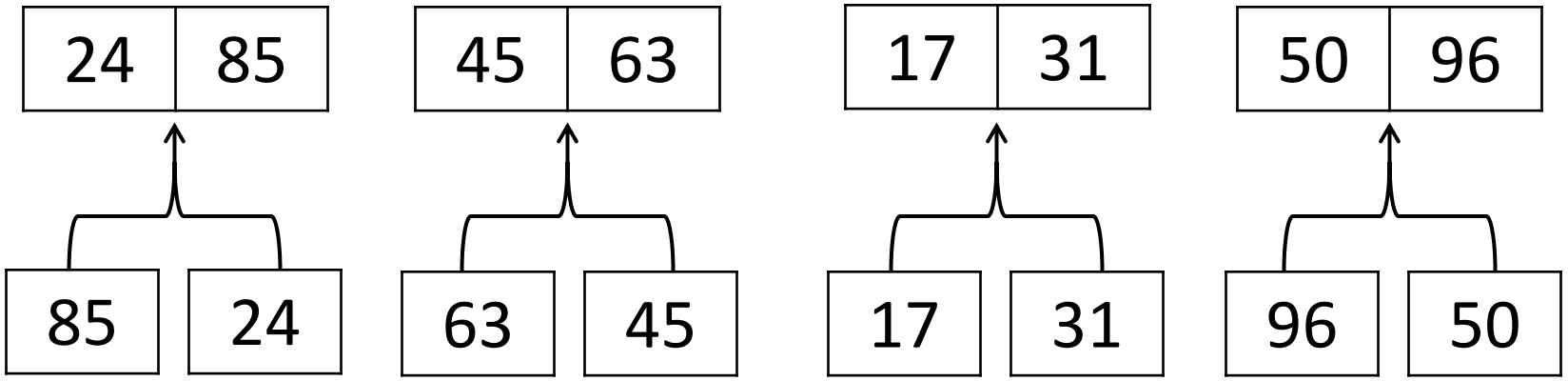
Divide and Conquer – Merge Sort



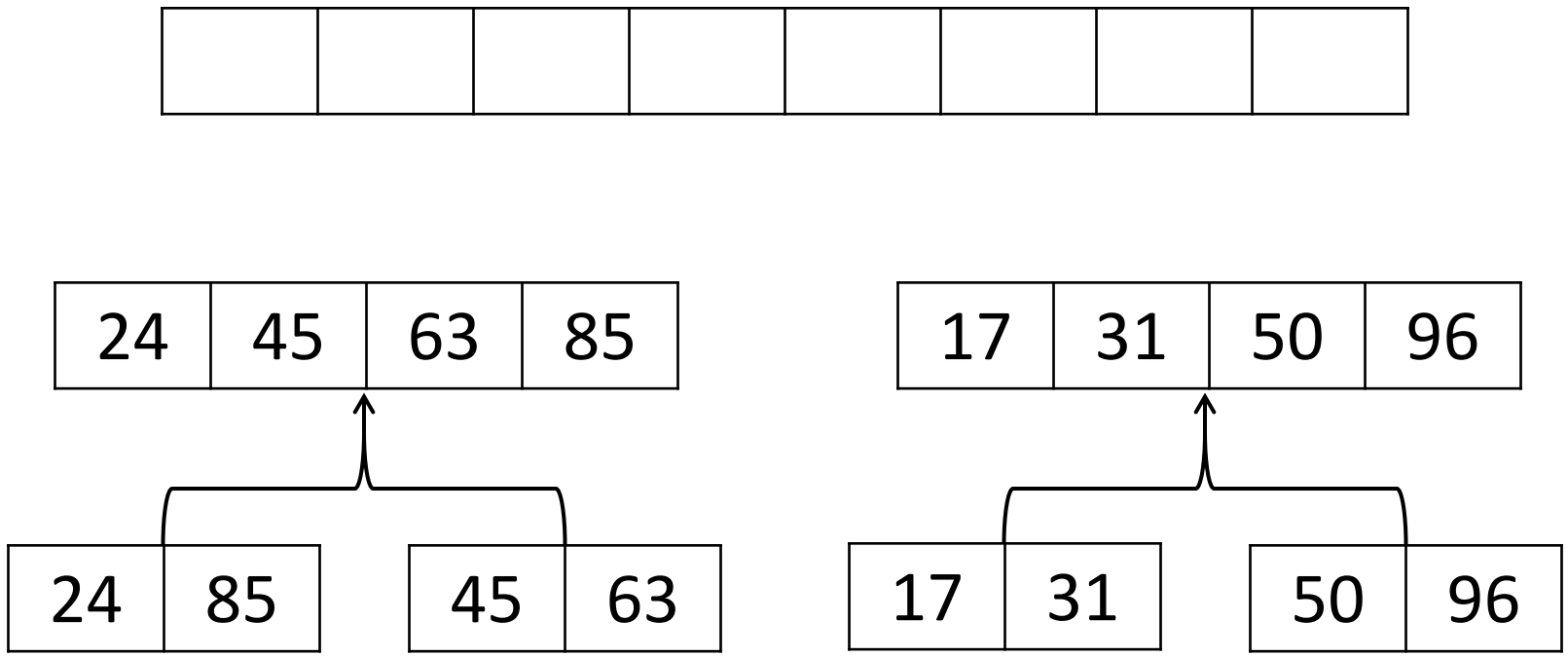
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Divide and Conquer – Merge Sort

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

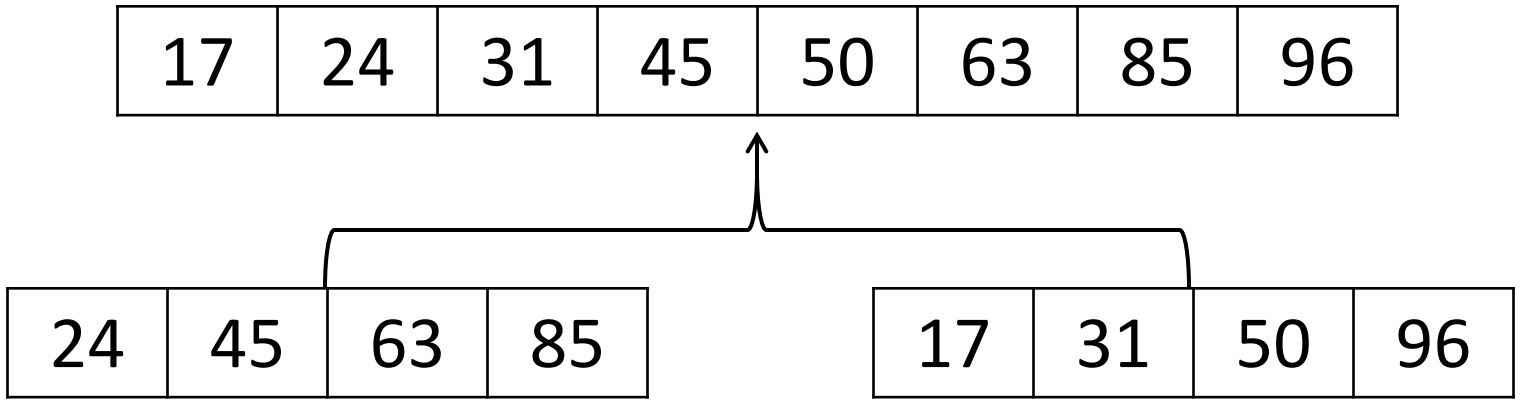


Divide and Conquer – Merge Sort



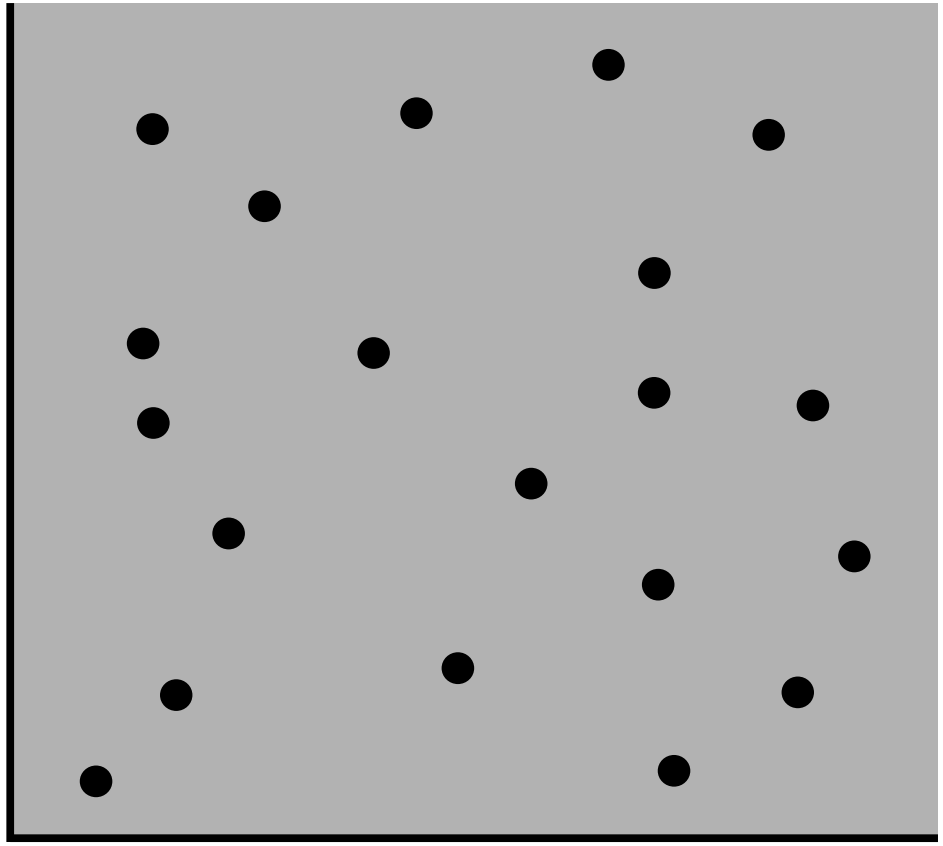
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Divide and Conquer – Merge Sort



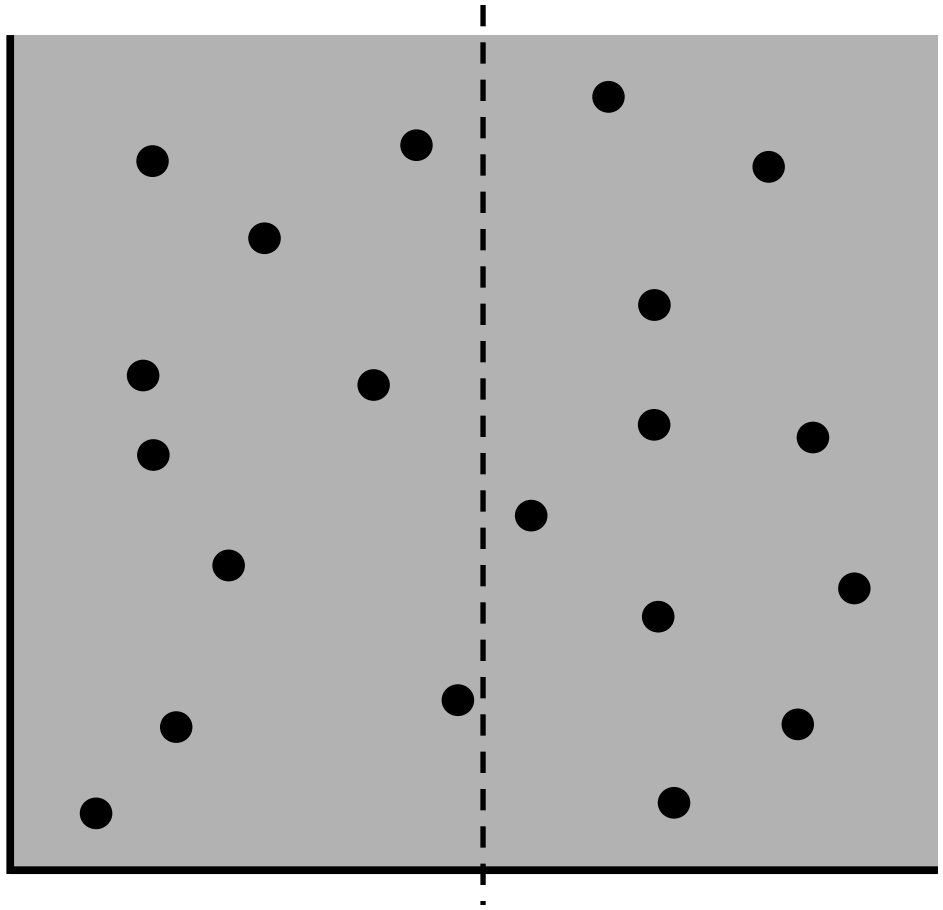
1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.

Closest Pair Problem – Divide and Conquer



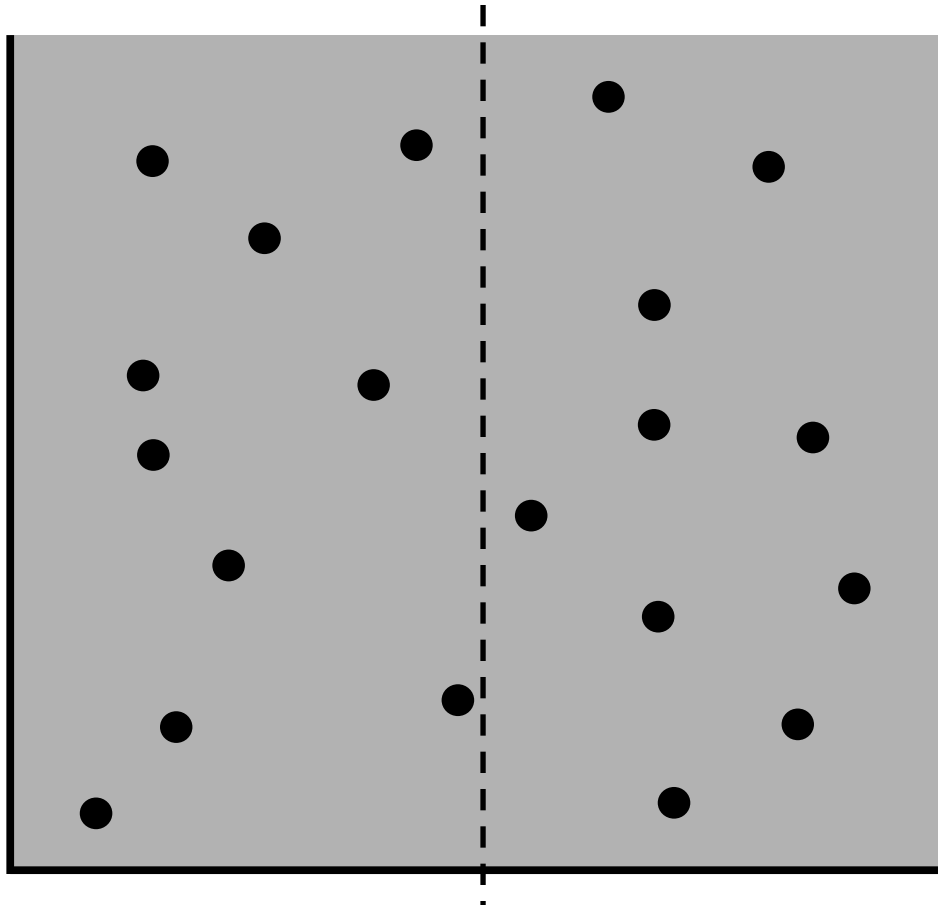
How can we make the problem smaller and easier?

Closest Pair Problem – Divide and Conquer



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

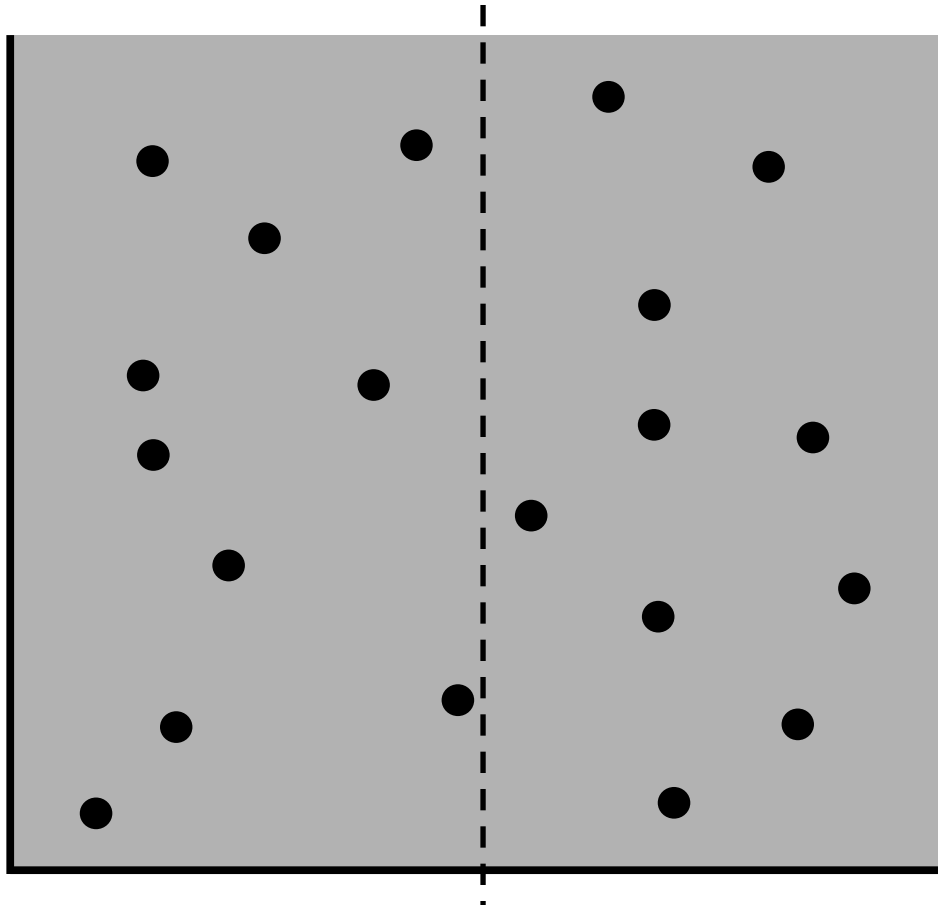
Closest Pair Problem – Divide and Conquer



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

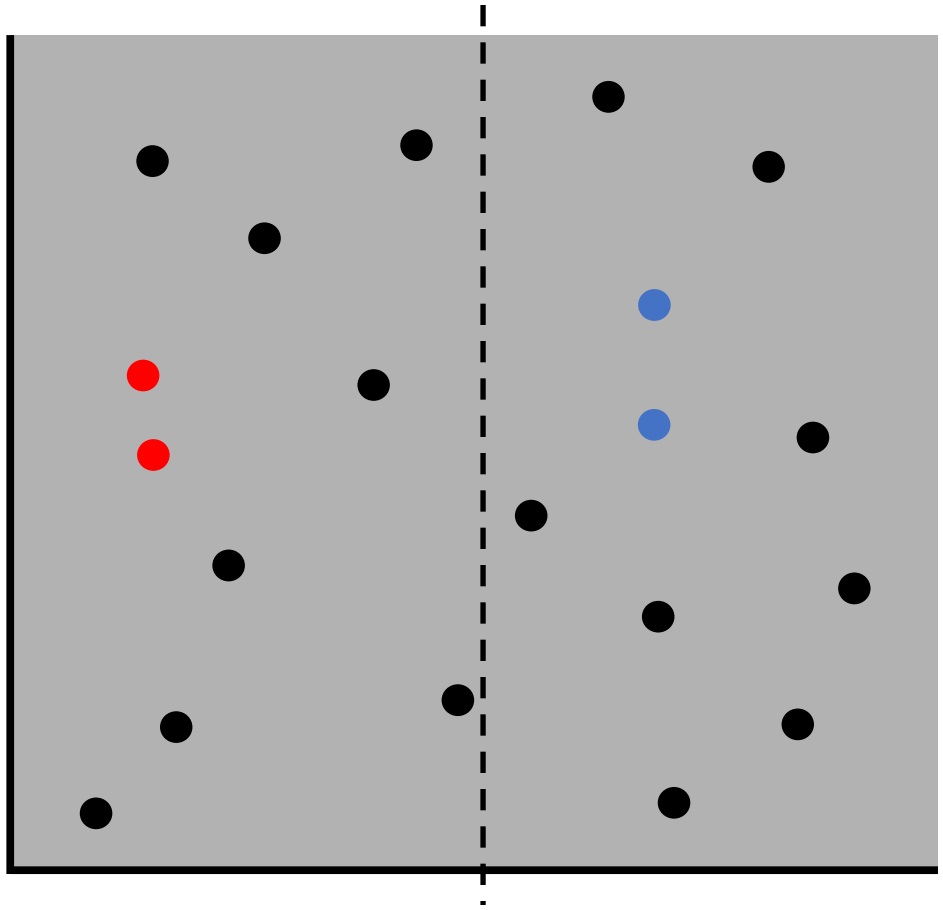
1. Sort by x-coordinate.
2. Put line at median value.

Closest Pair Problem – Divide and Conquer



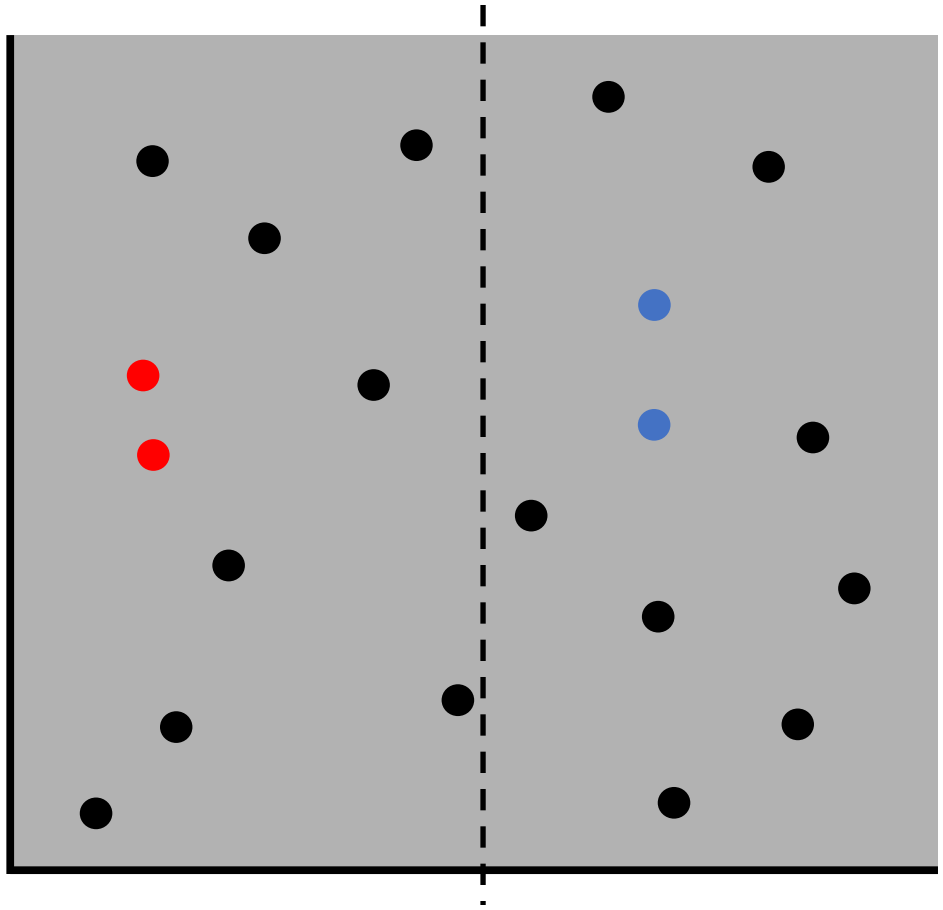
Conquer: Recursively find closest pairs on each side.

Closest Pair Problem – Divide and Conquer



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

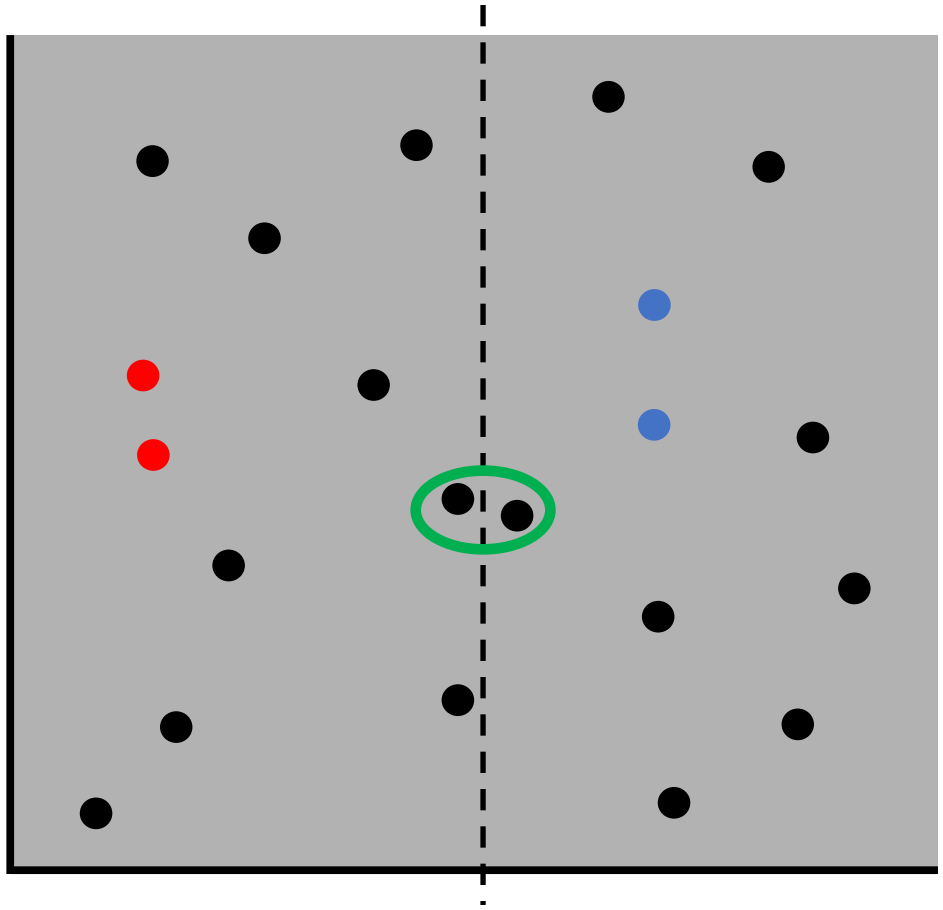
Closest Pair Problem – Divide and Conquer



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

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

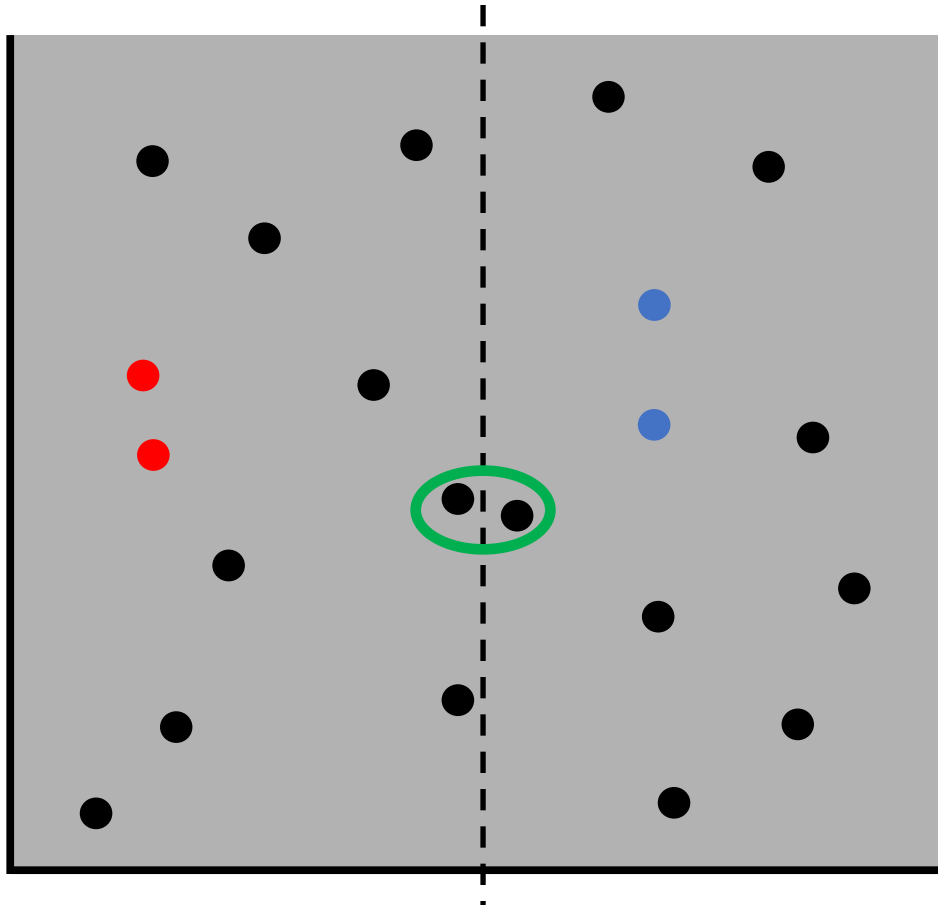
Closest Pair Problem – Divide and Conquer



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

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

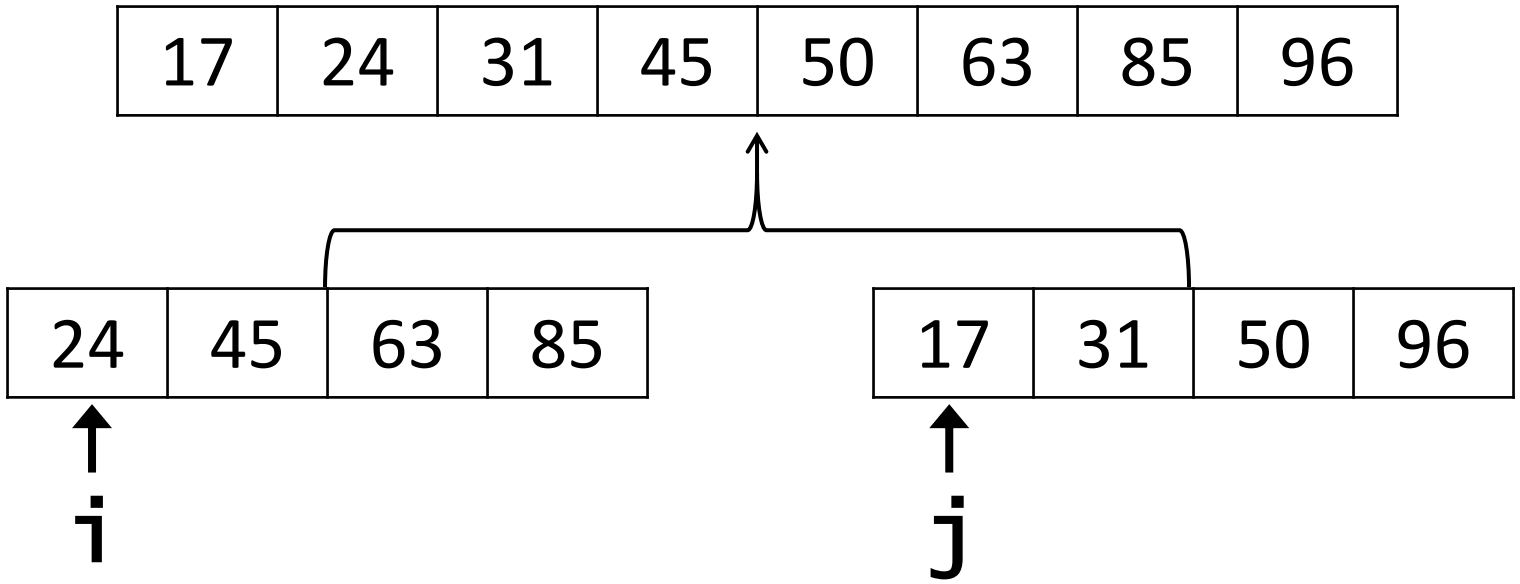
Closest Pair Problem – Divide and Conquer



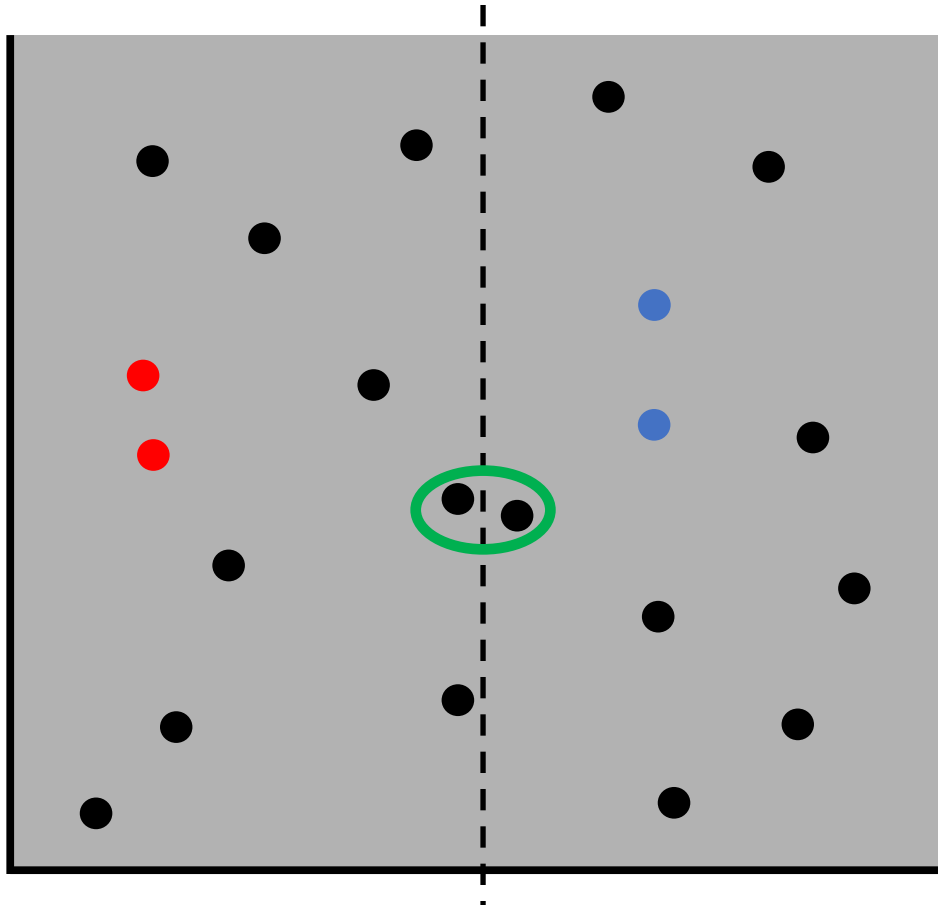
Combine: If we had closest left and closest right, how do we determine closest?

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

Merge Sort – Combine



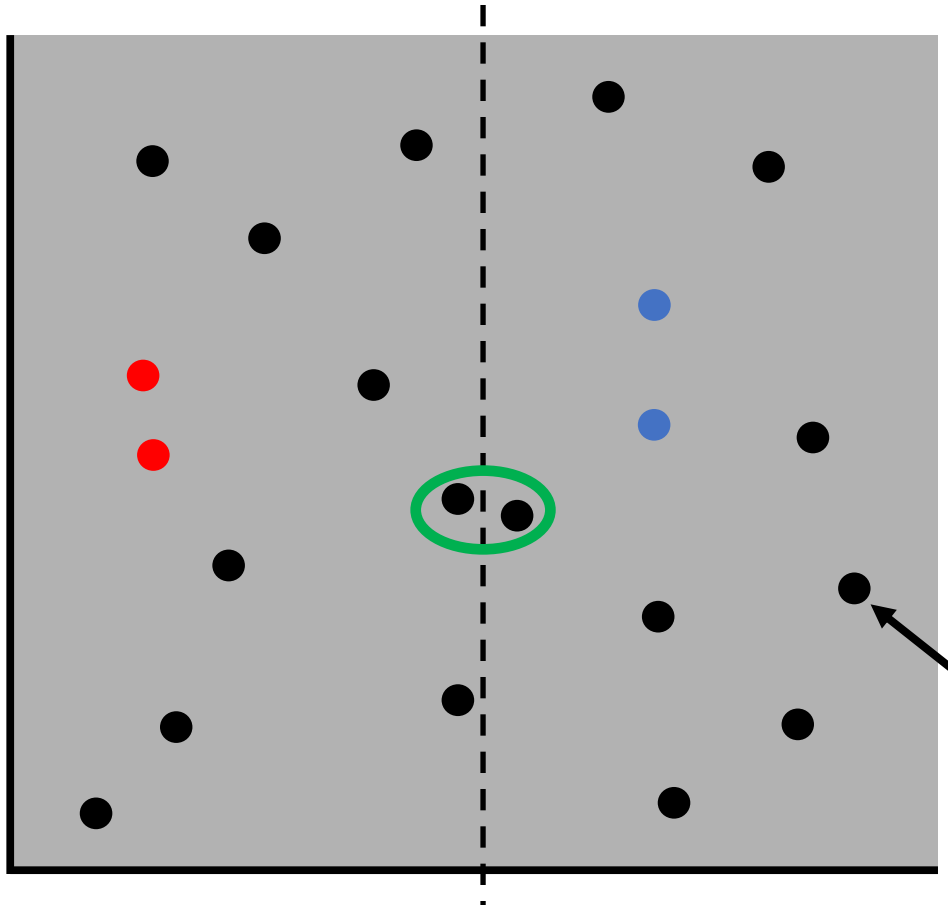
Closest Pair Problem – Divide and Conquer



How should we search for “straddle points”?

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

Closest Pair Problem – Divide and Conquer

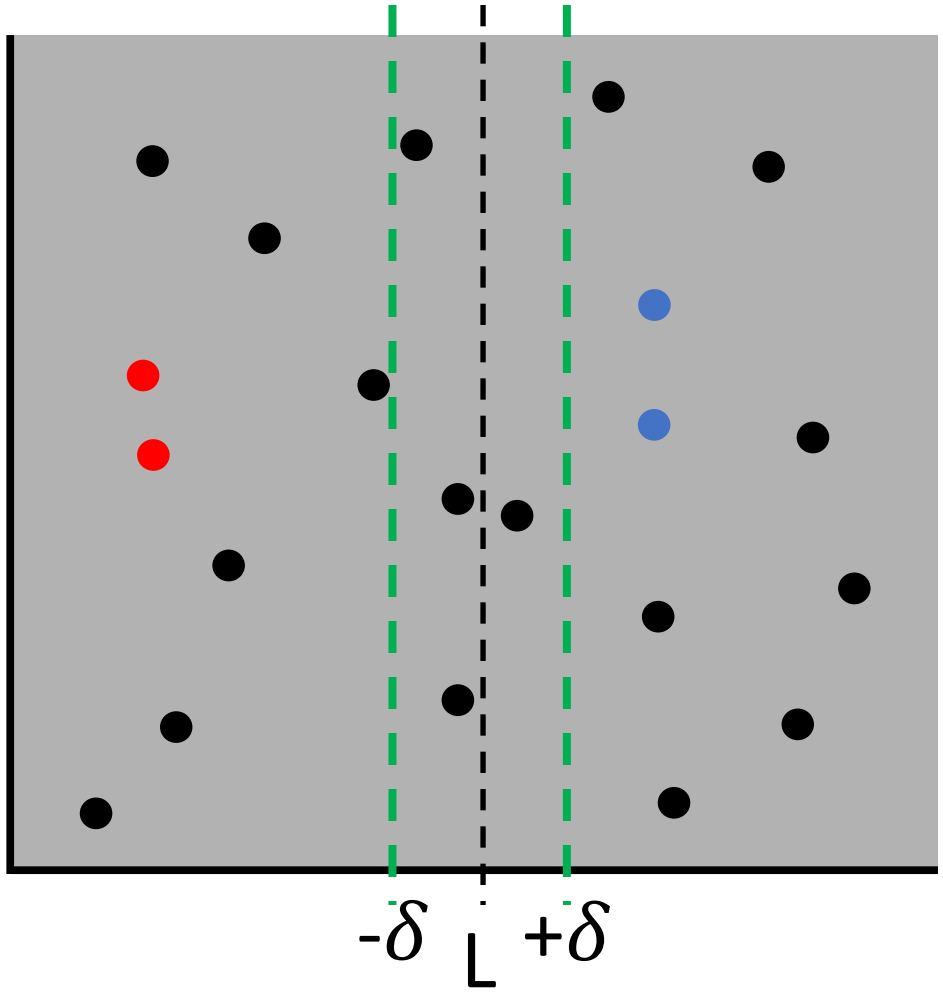


How should we search for “straddle points”?

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

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

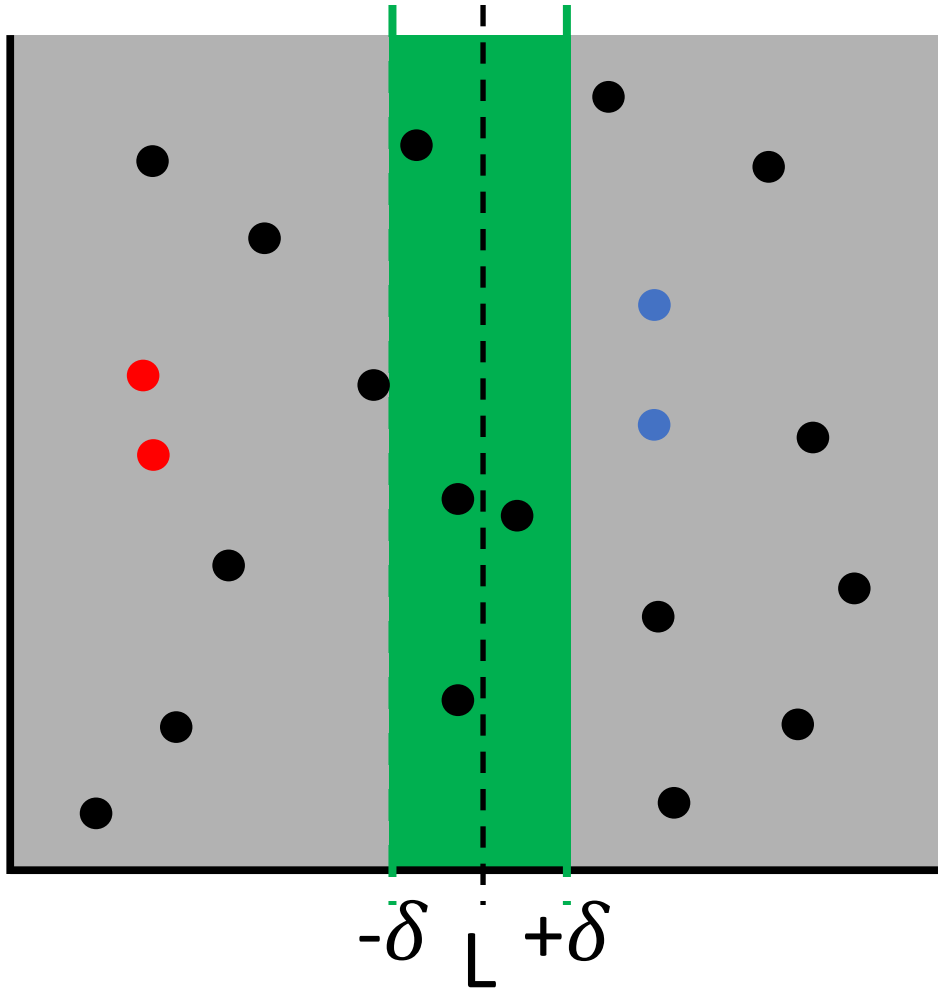
Closest Pair Problem – Divide and Conquer



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

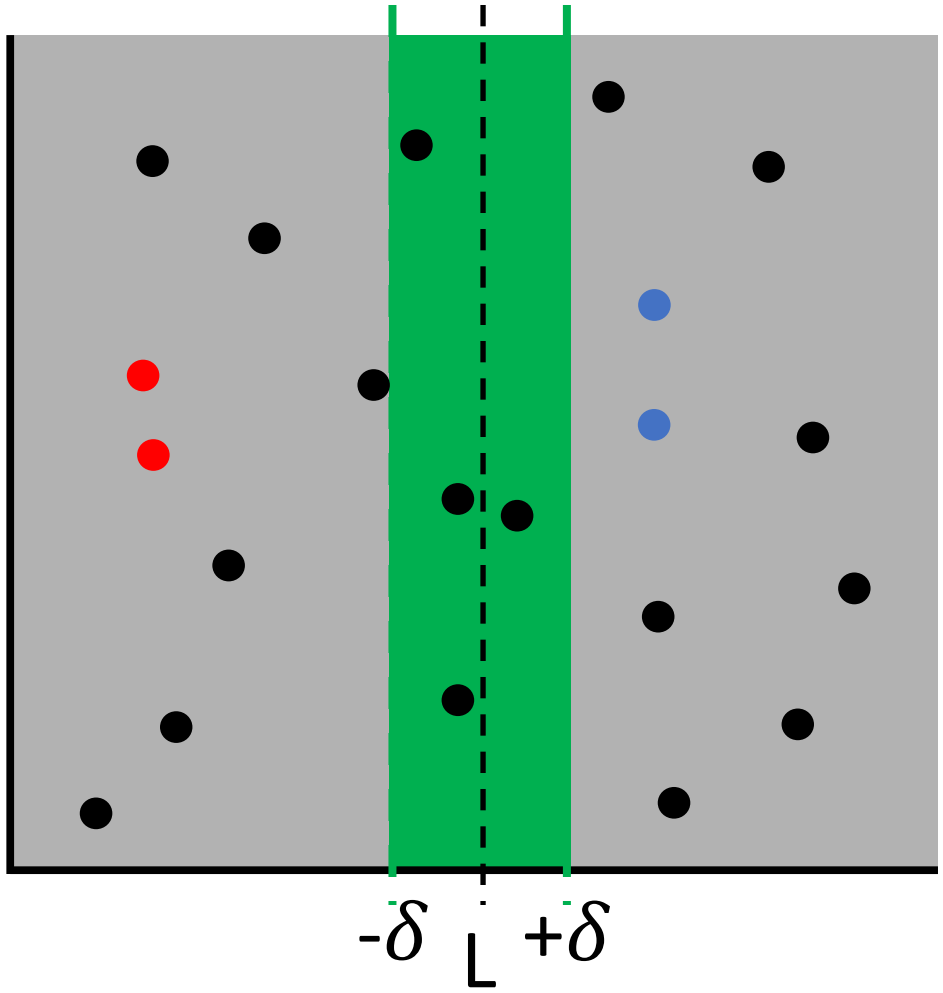


Rule: We only need to hunt for straddle points at most δ away from L.

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

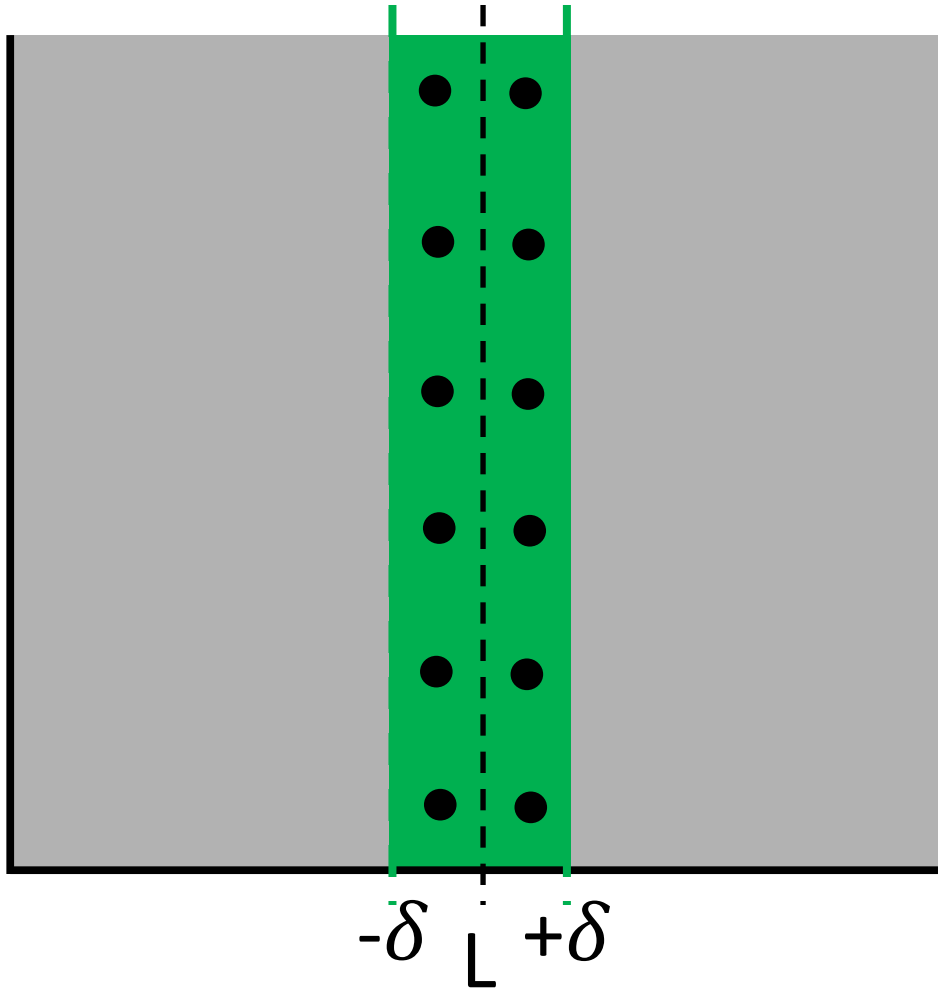
Let **S** be the set of straddle points.

Closest Pair Problem – Divide and Conquer



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

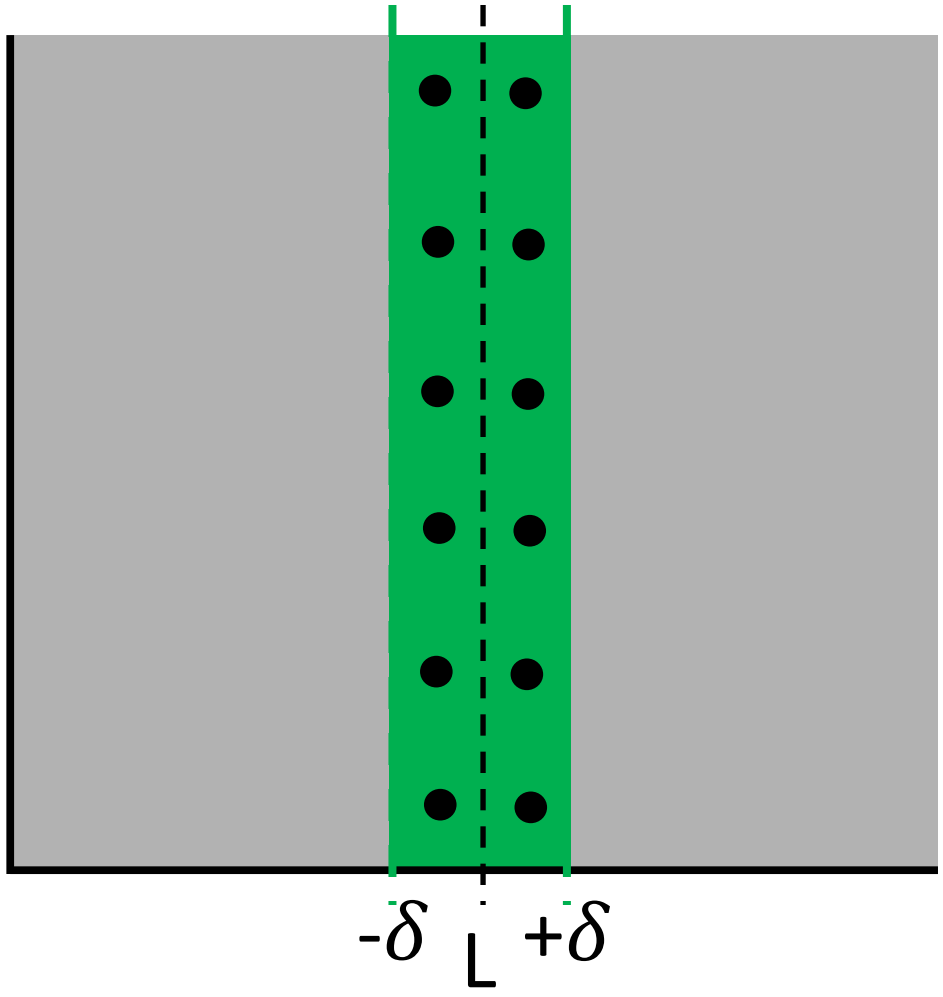
Closest Pair Problem – Divide and Conquer



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

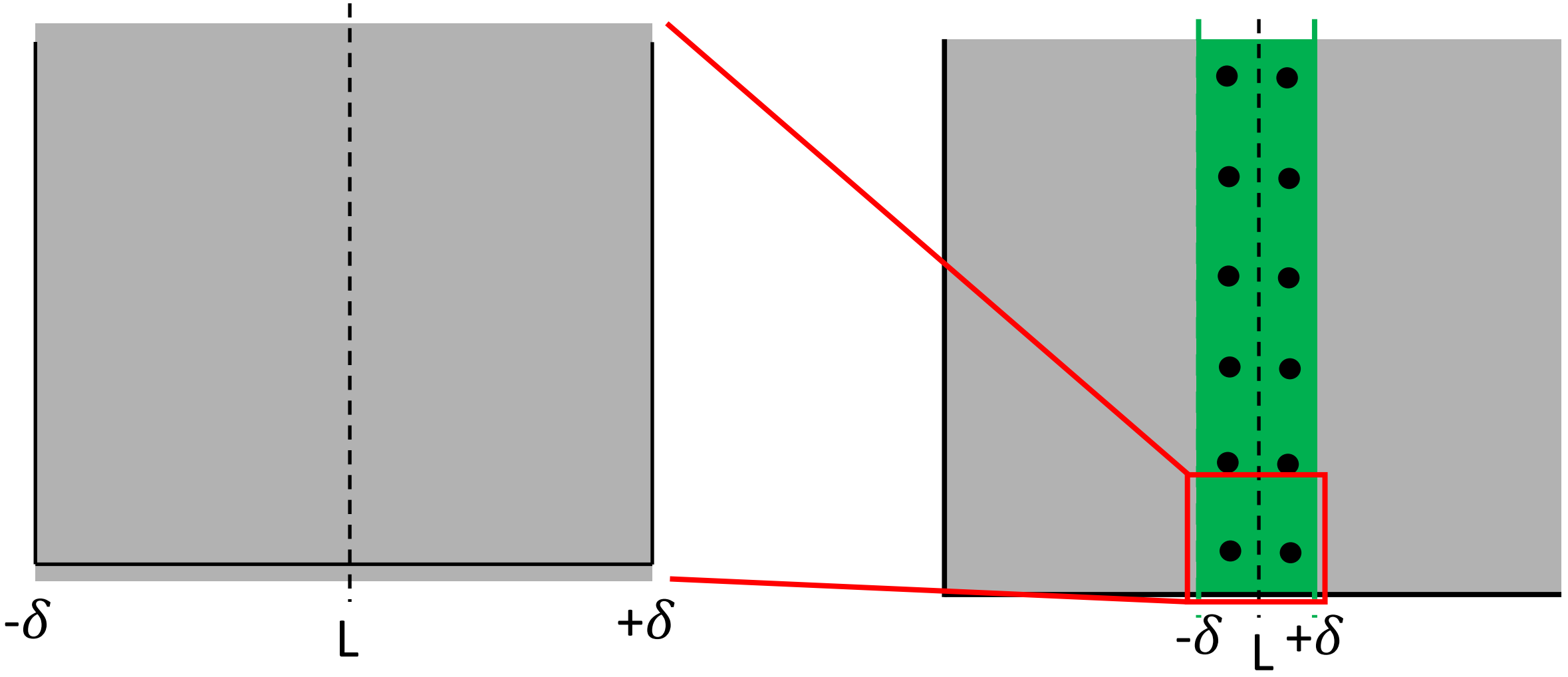


Can we just compare all left straddle points to all right straddle points?

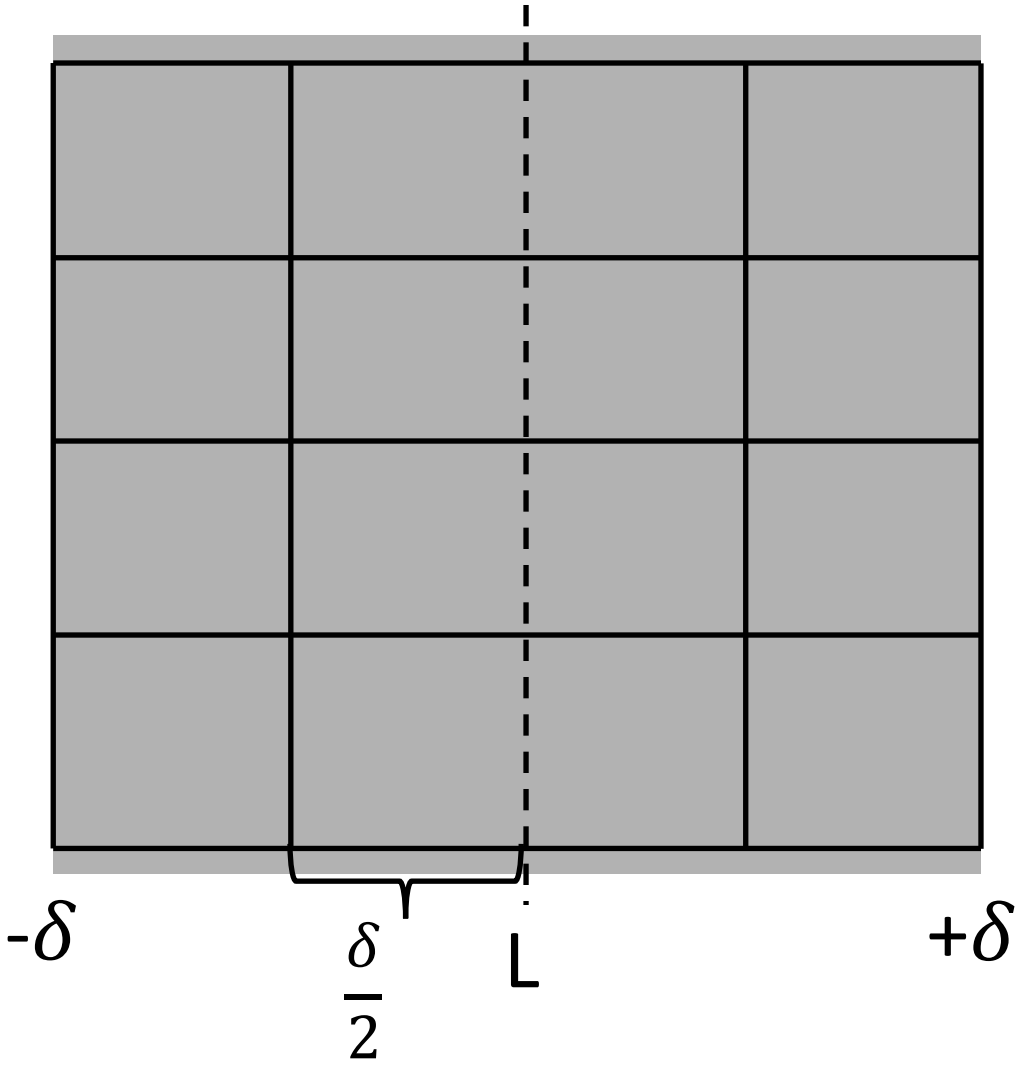
Yes, but running time could still be $O(n^2)$.

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

Closest Pair Problem – Divide and Conquer

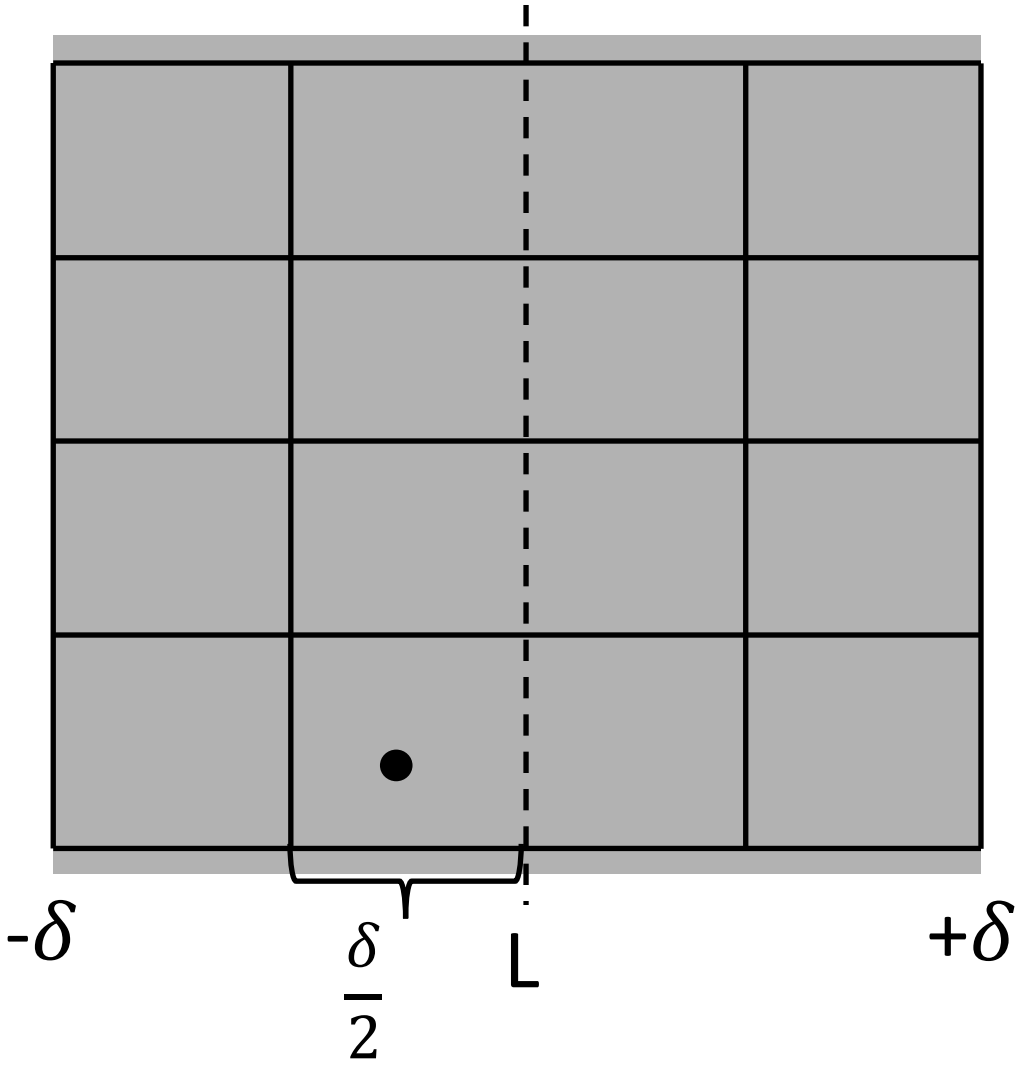


Closest Pair Problem – Divide and Conquer



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

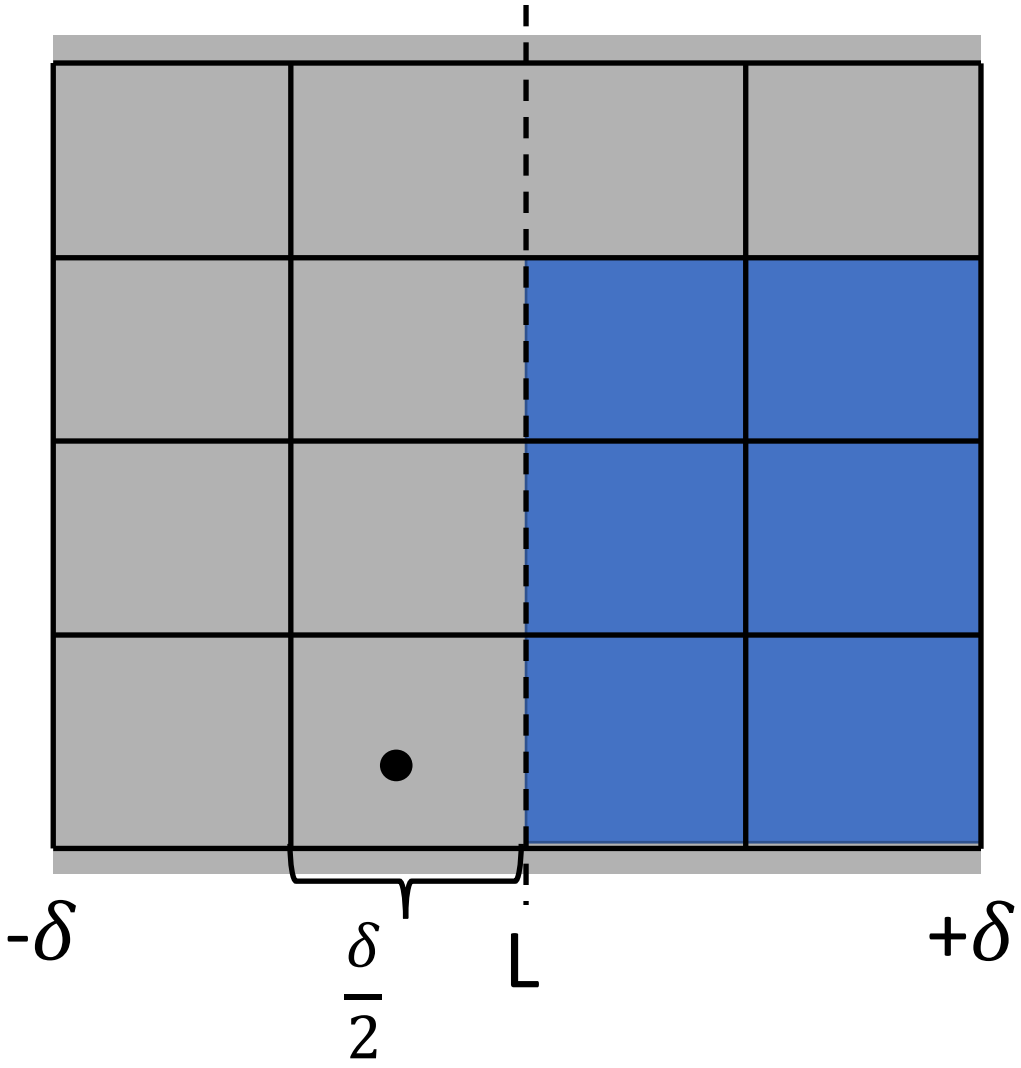
Closest Pair Problem – Divide and Conquer



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

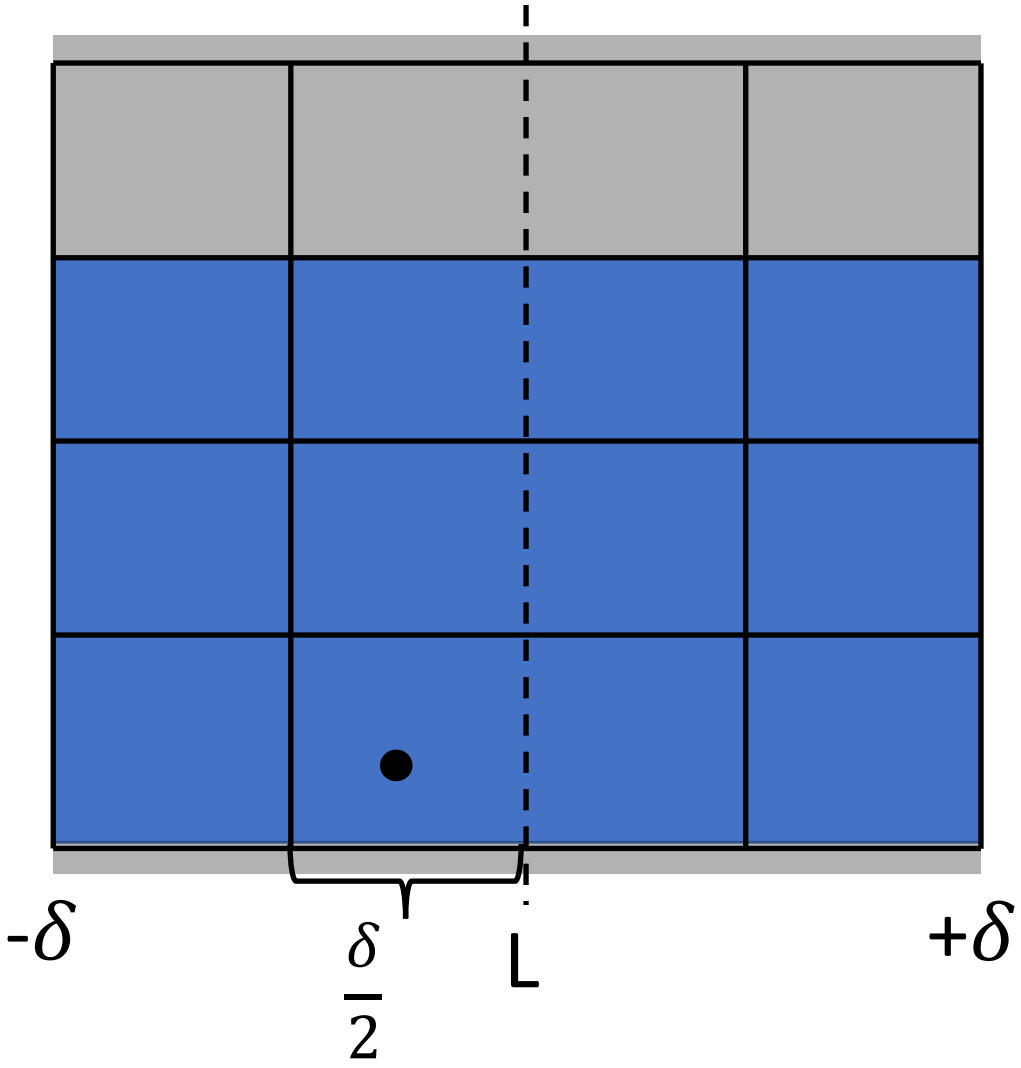


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

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

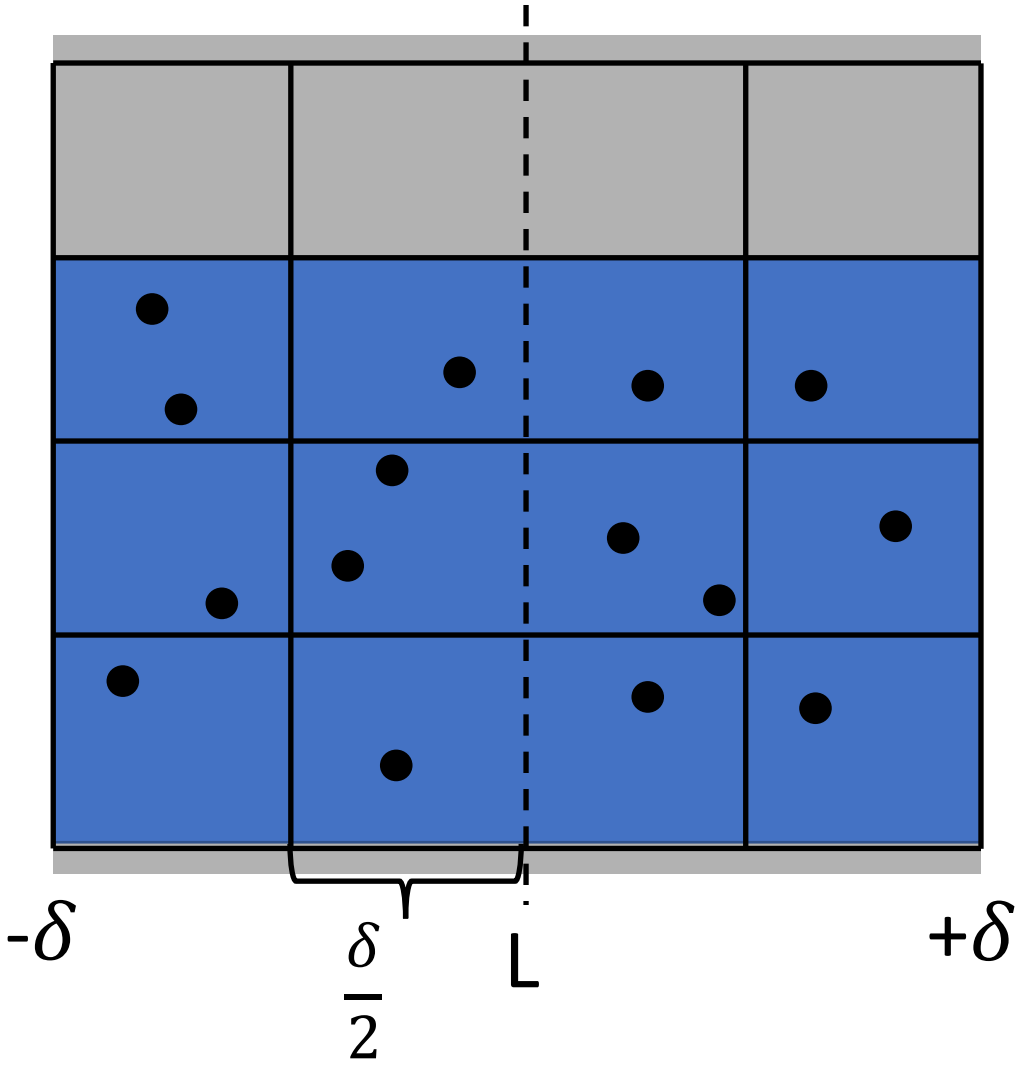


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

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.

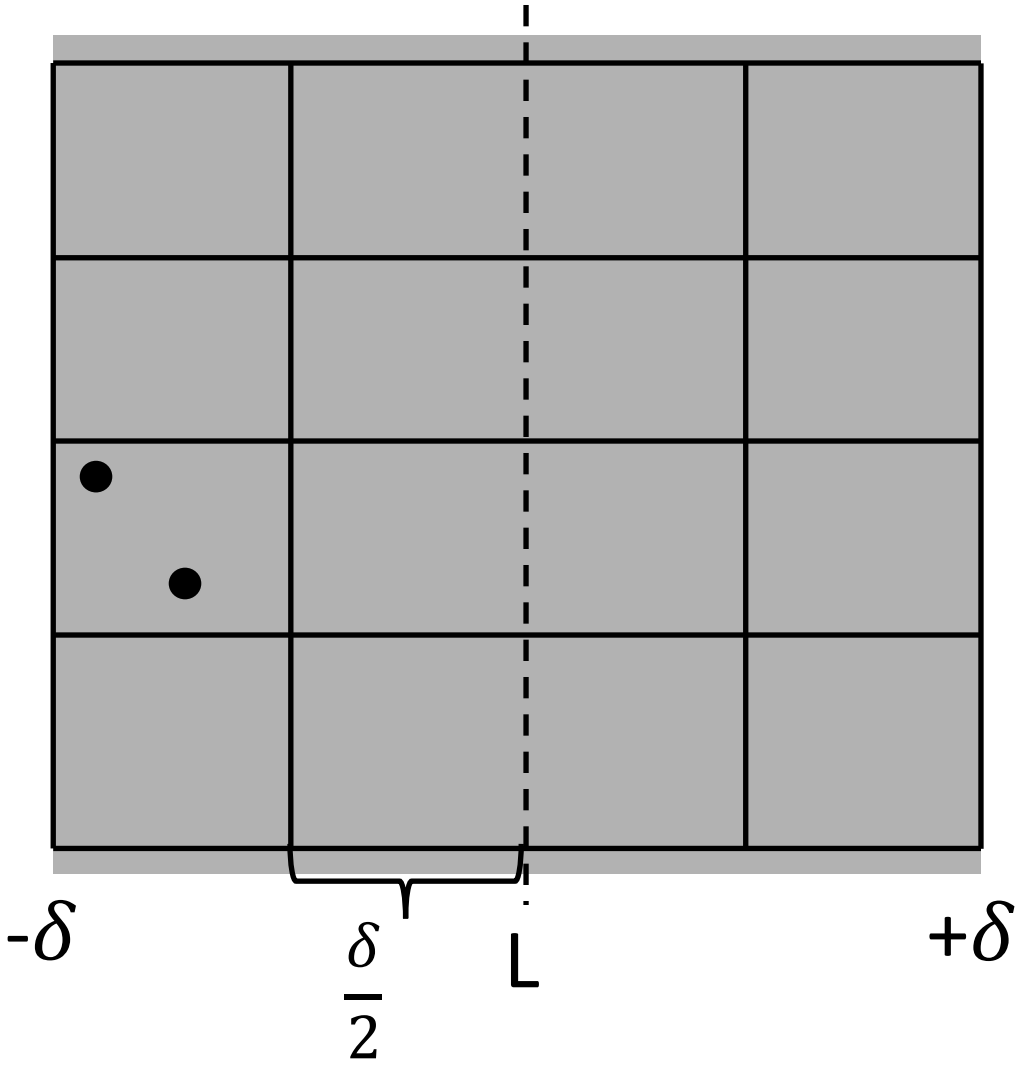
Can we focus our search to certain boxes?

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

What if all of the points are in this region?

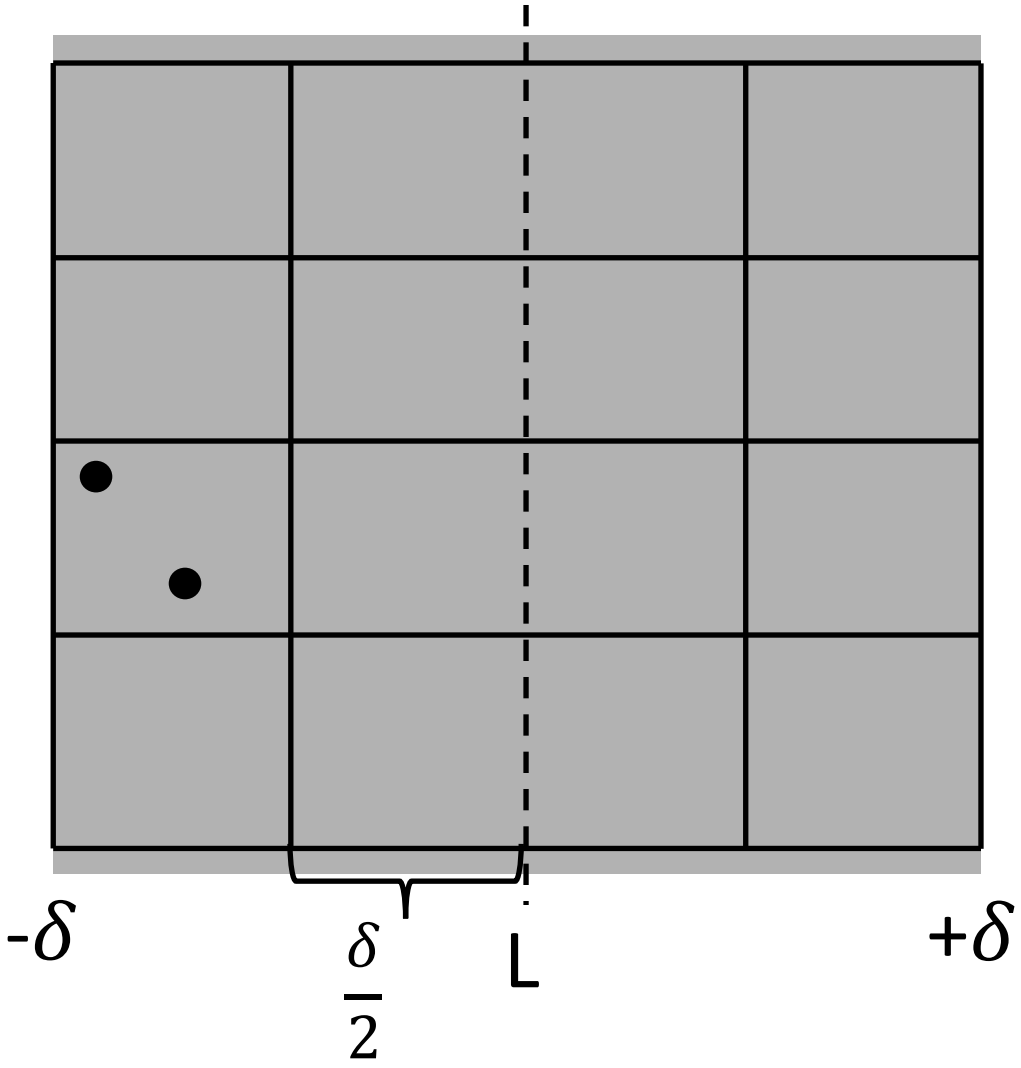
This still gives us possibly lots of points to look at.

Closest Pair Problem – Divide and Conquer



Can we have multiple points in one box?

Closest Pair Problem – Divide and Conquer

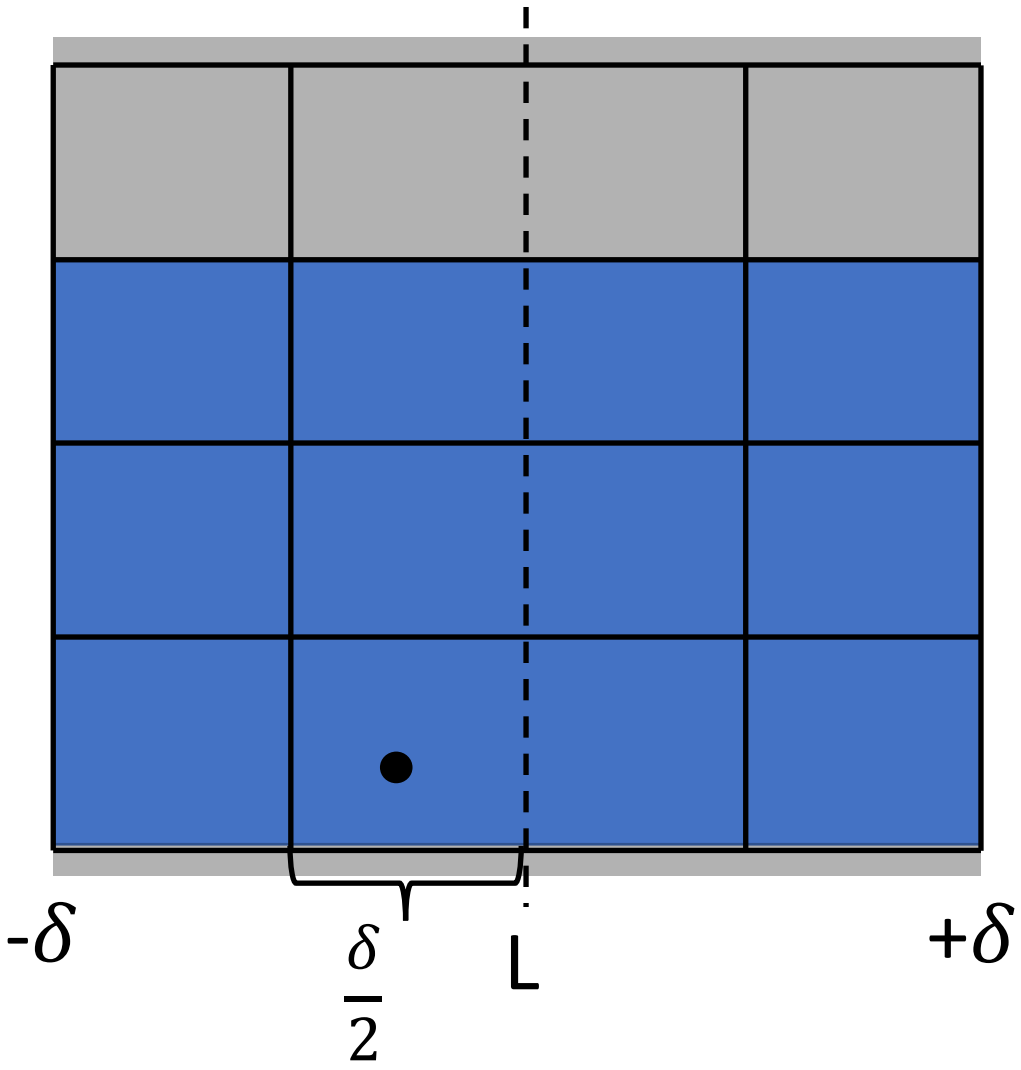


Can we have multiple points in one box?

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

$\Rightarrow$ at most one point per box.

Closest Pair Problem – Divide and Conquer

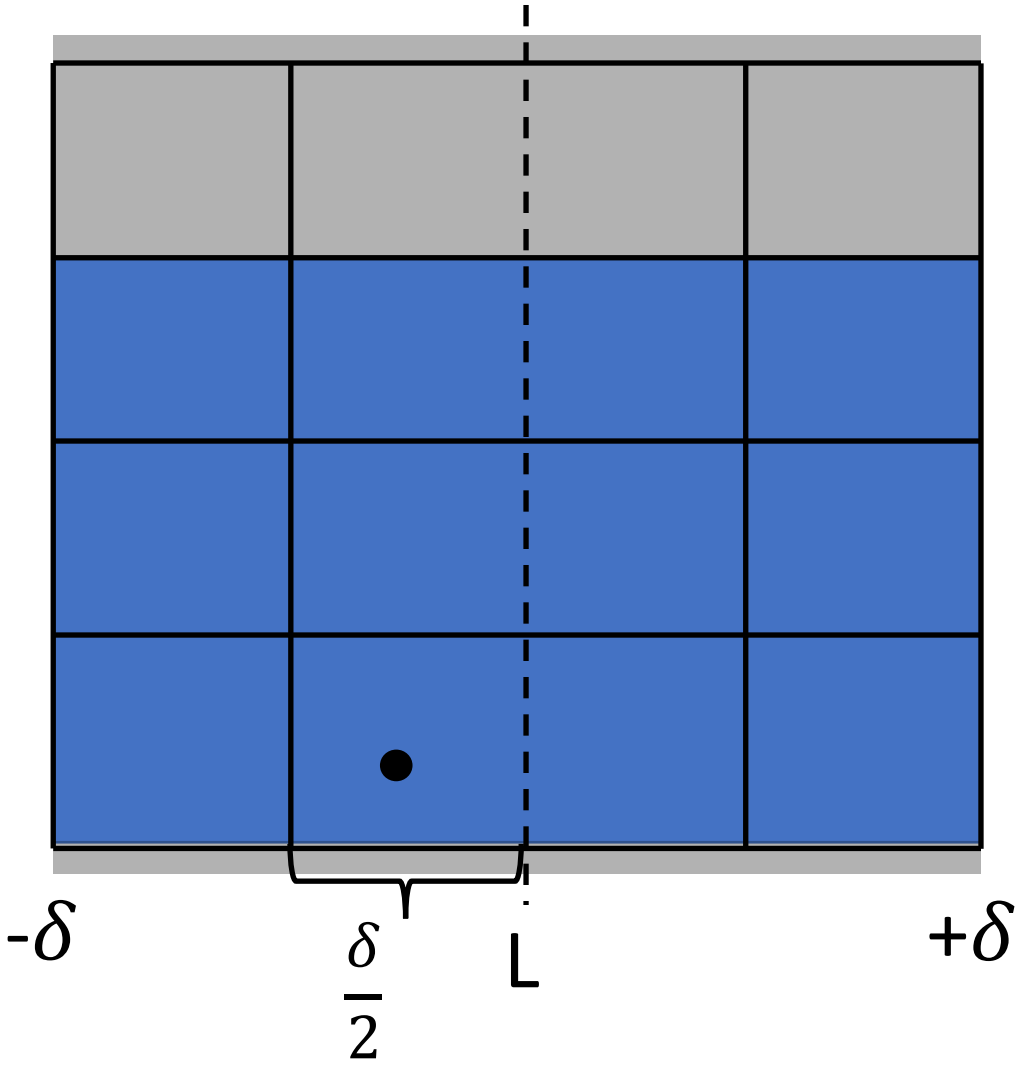


Only care about 11 boxes

\+ At most one point per box

At most 11 points to check

Closest Pair Problem – Divide and Conquer



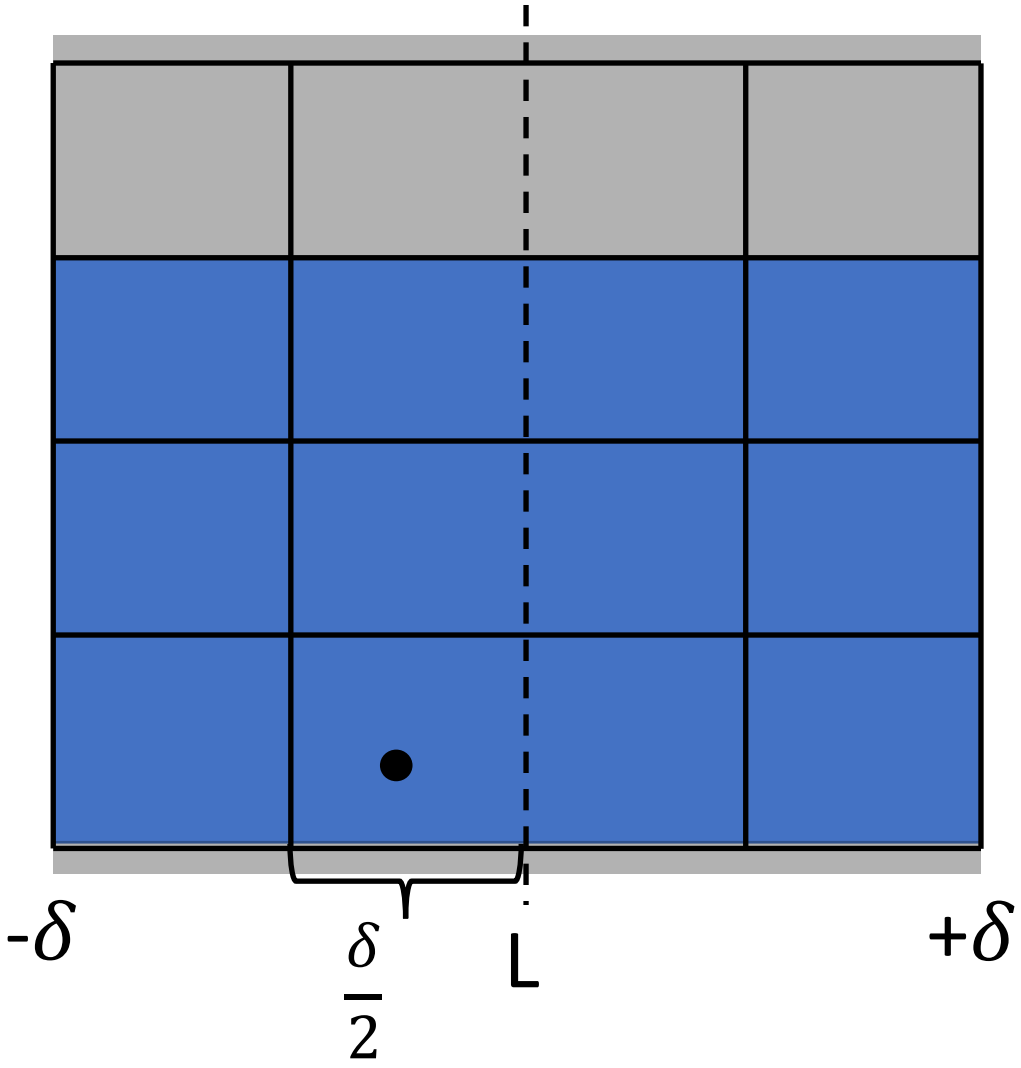
Only care about 11 boxes

\+ 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



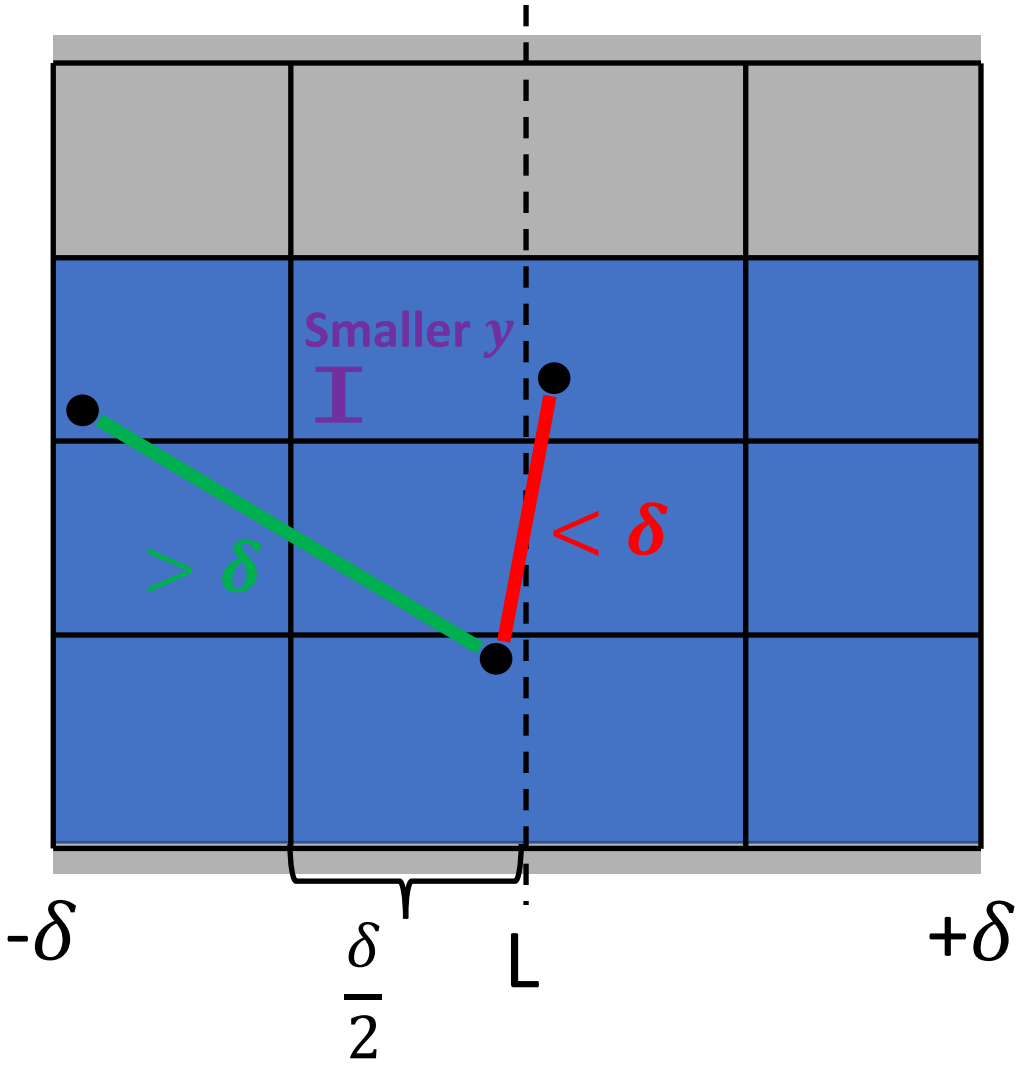
Only care about 11 boxes

\+ 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



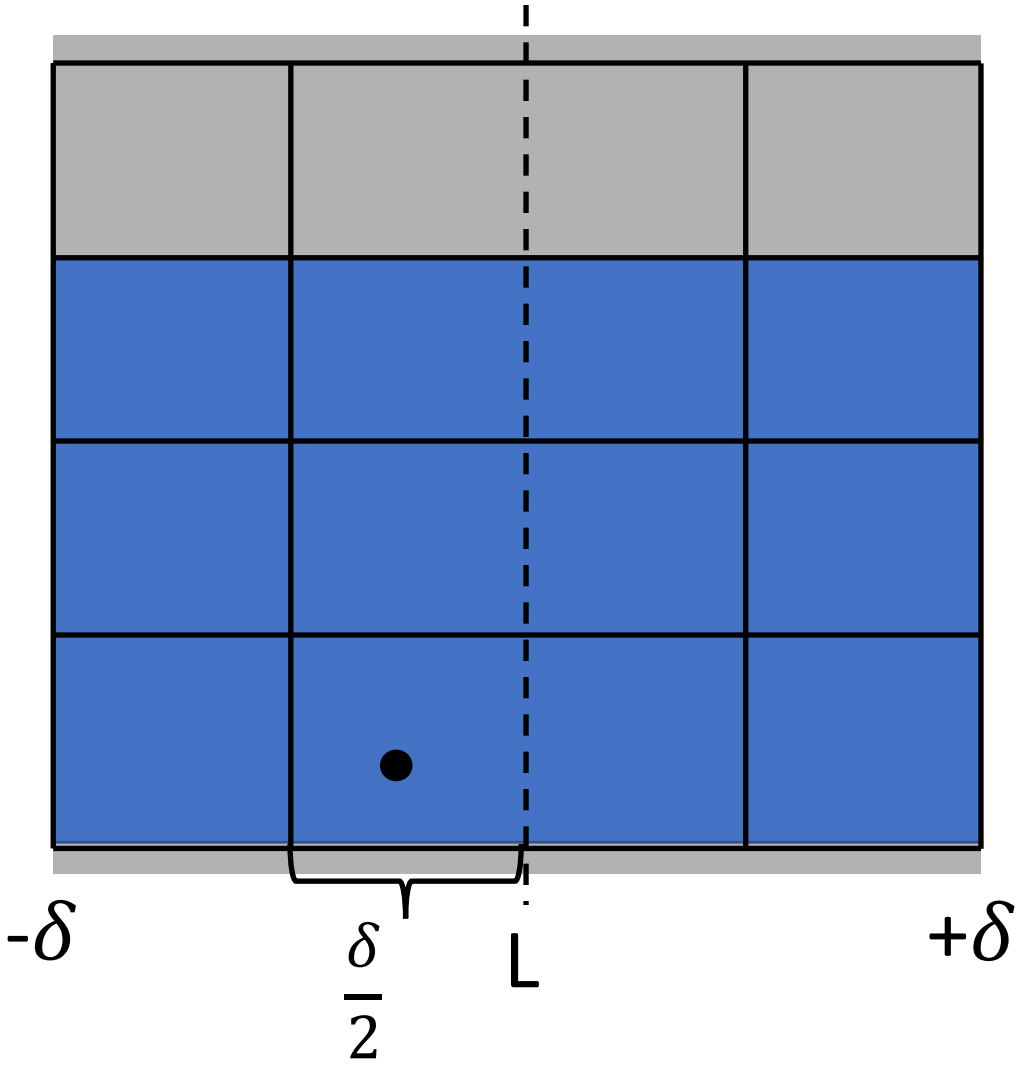
Only care about 11 boxes

\+ 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:

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

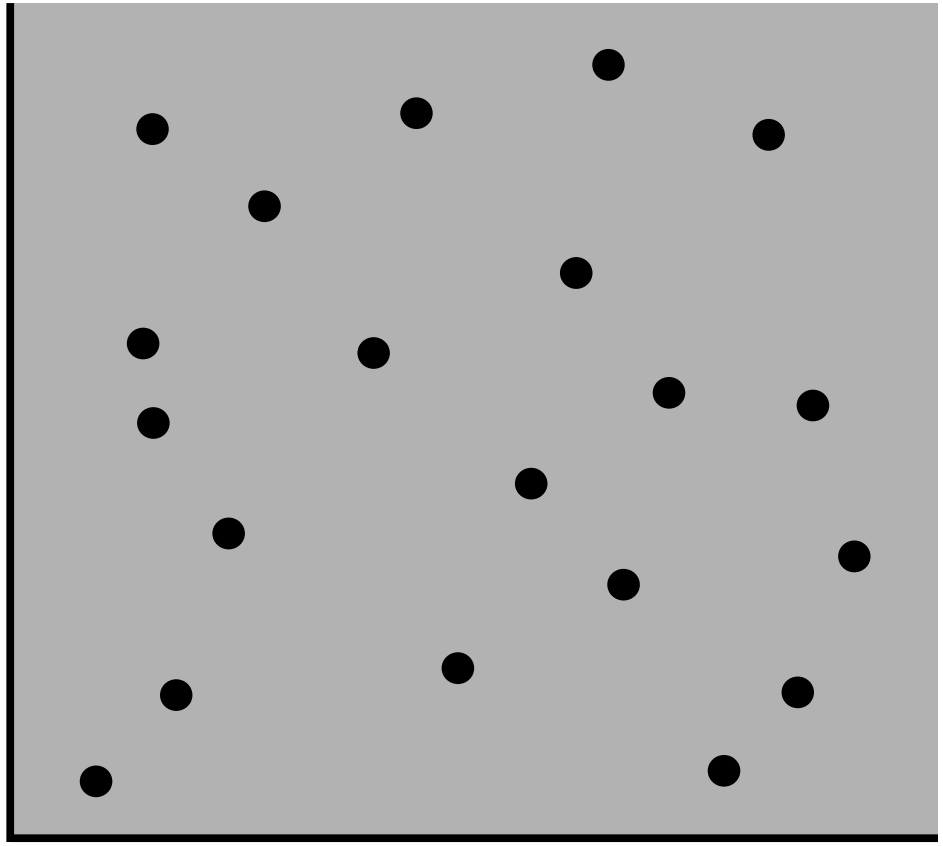
Closest Pair Problem – Algorithm

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

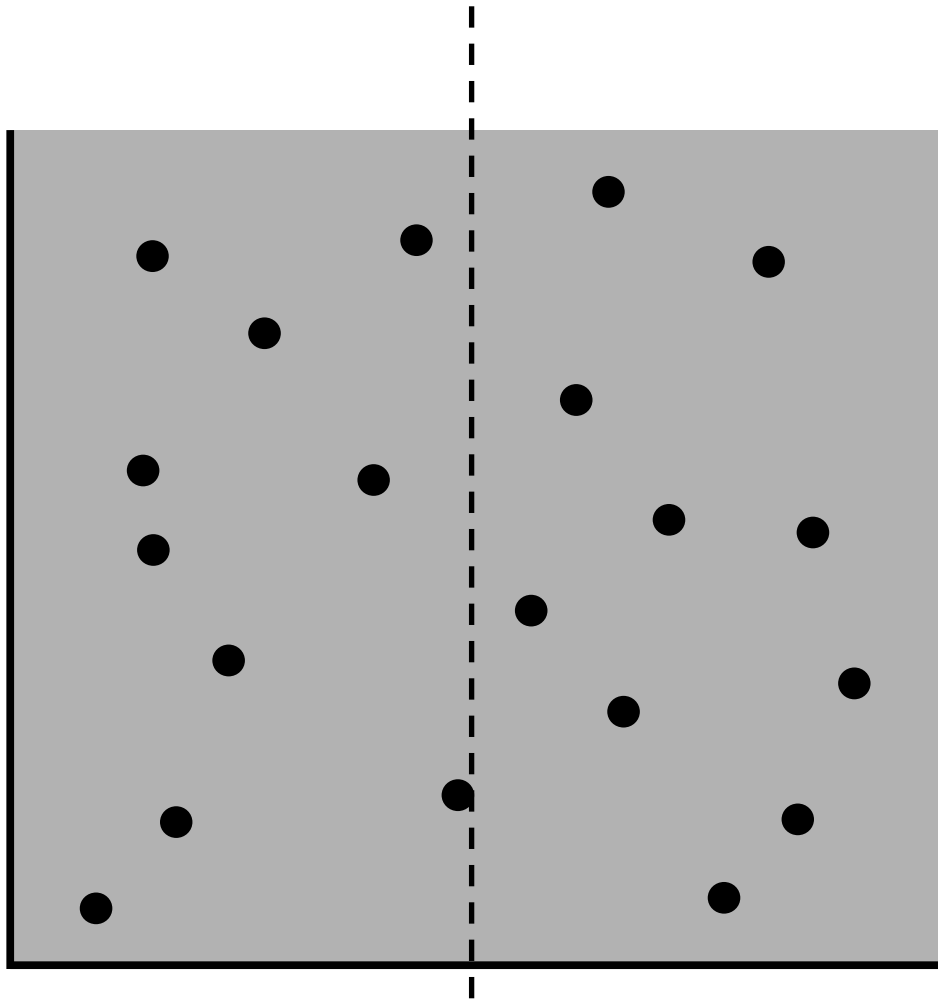
Closest Pair Problem – Algorithm

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

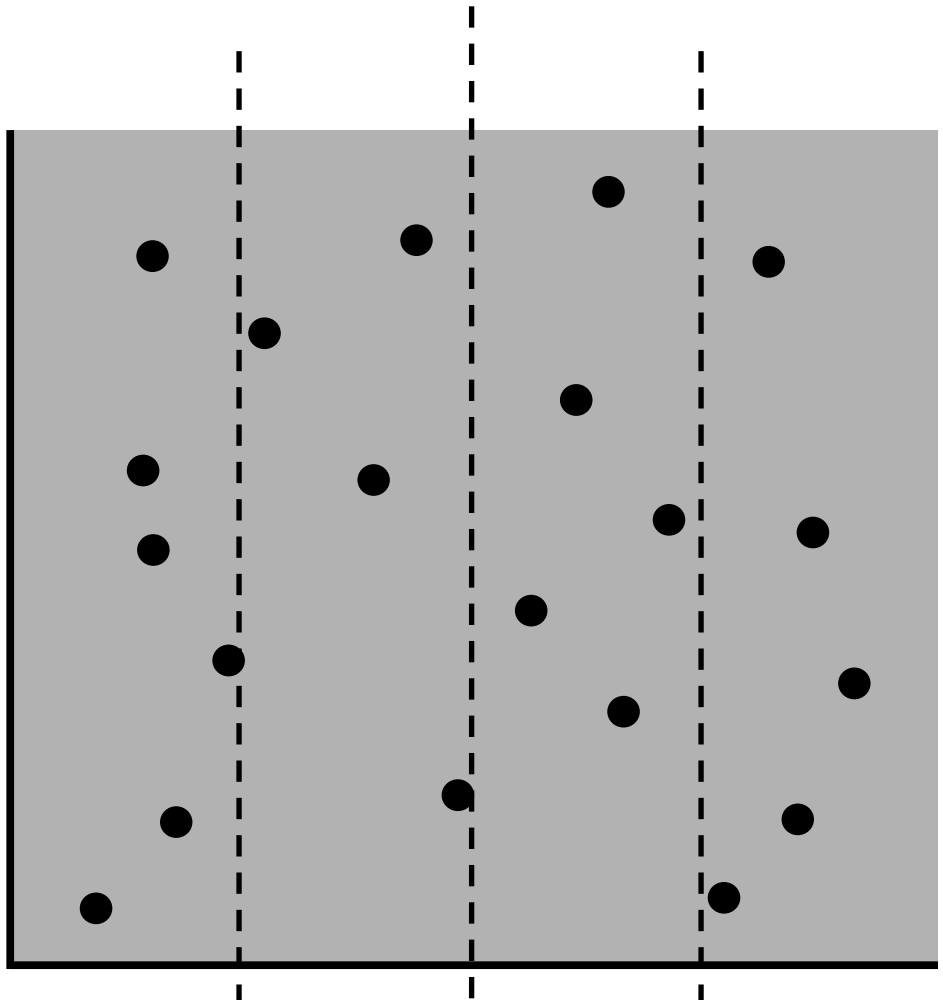
Closest Pair Problem – Divide and Conquer



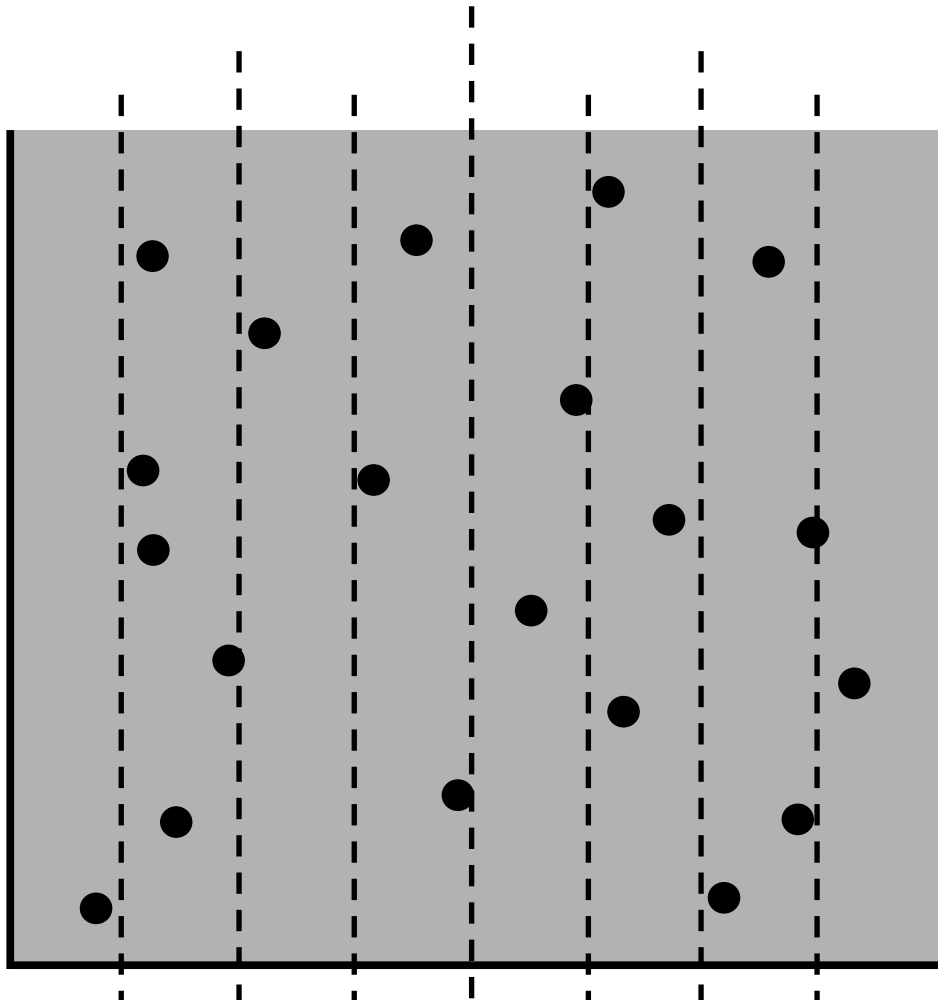
Closest Pair Problem – Divide and Conquer



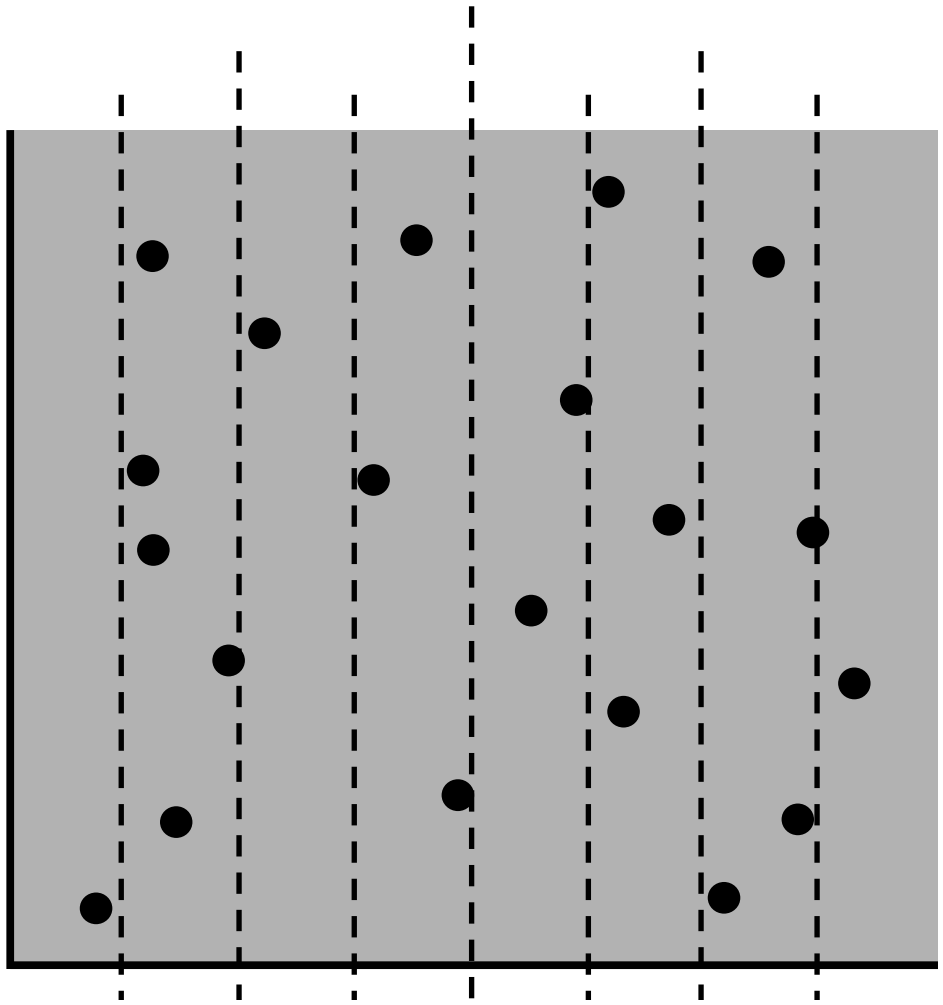
Closest Pair Problem – Divide and Conquer



Closest Pair Problem – Divide and Conquer

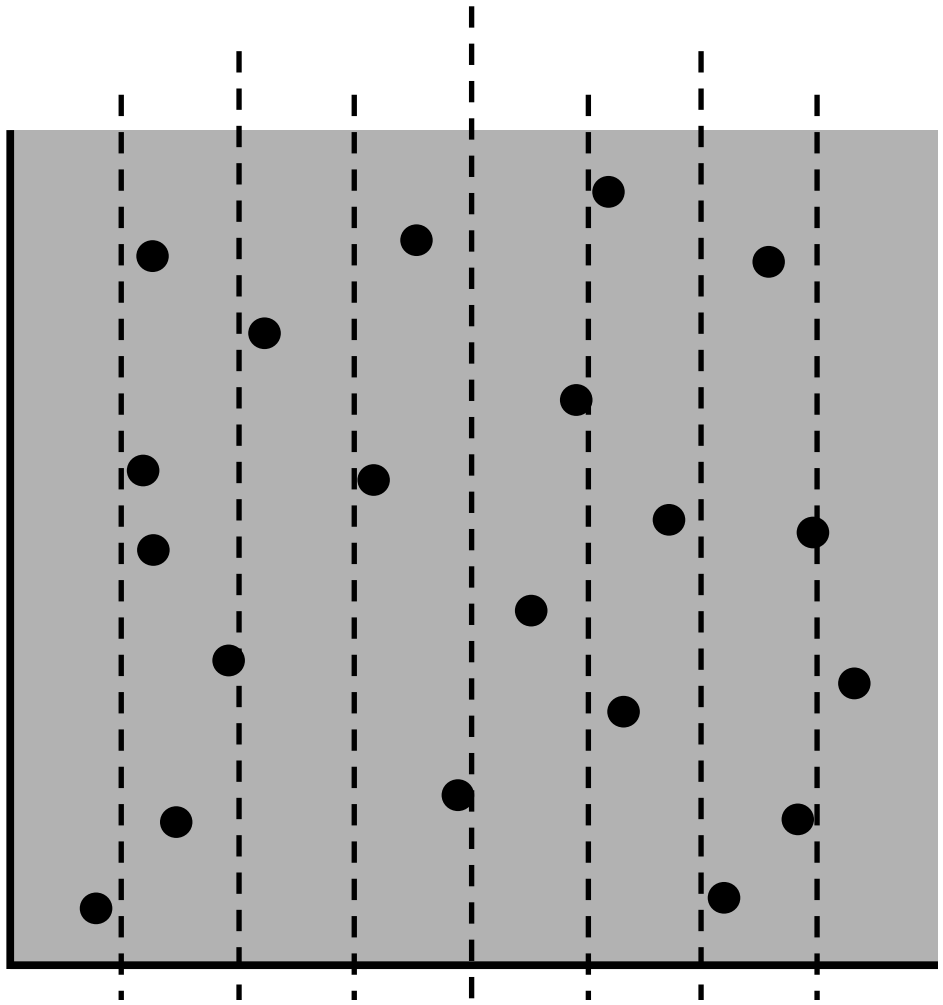


Closest Pair Problem – Divide and Conquer



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

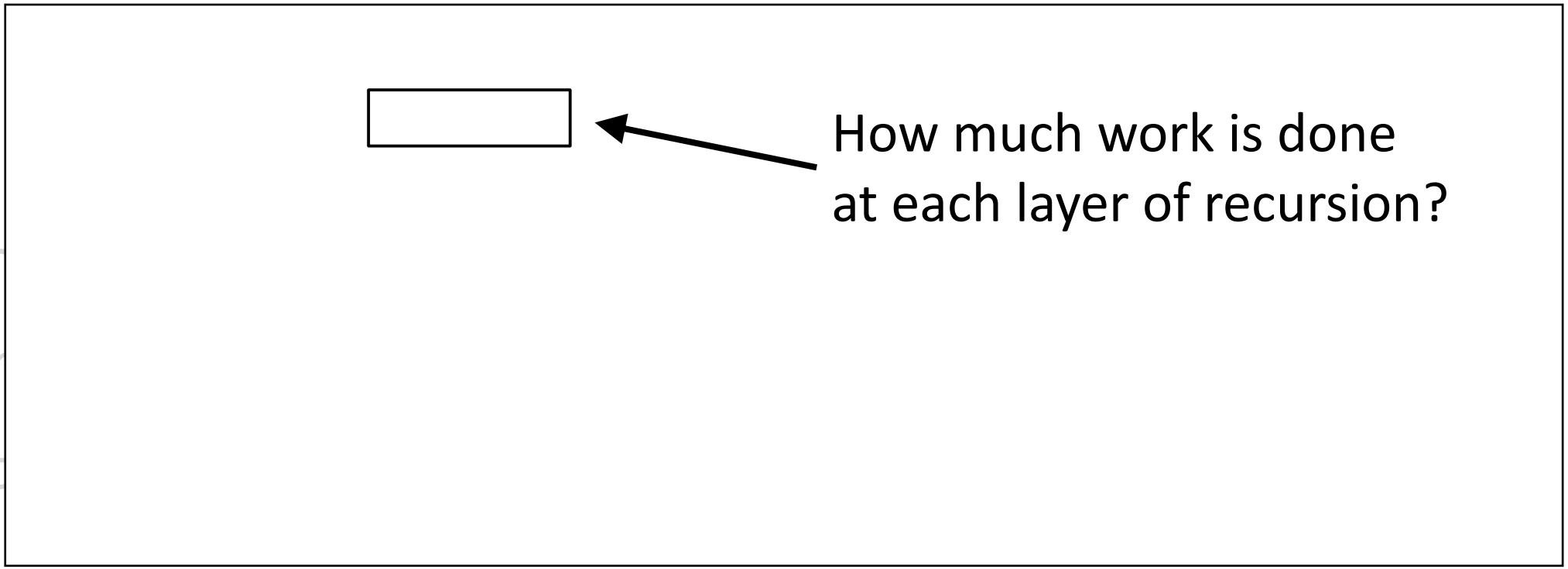
1. Sort points by x-coordinate and make L. $\boldsymbol{O(n \log n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of L. $\boldsymbol{O(n)}$
5. Sort S by y-coordinate. $\boldsymbol{O(n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

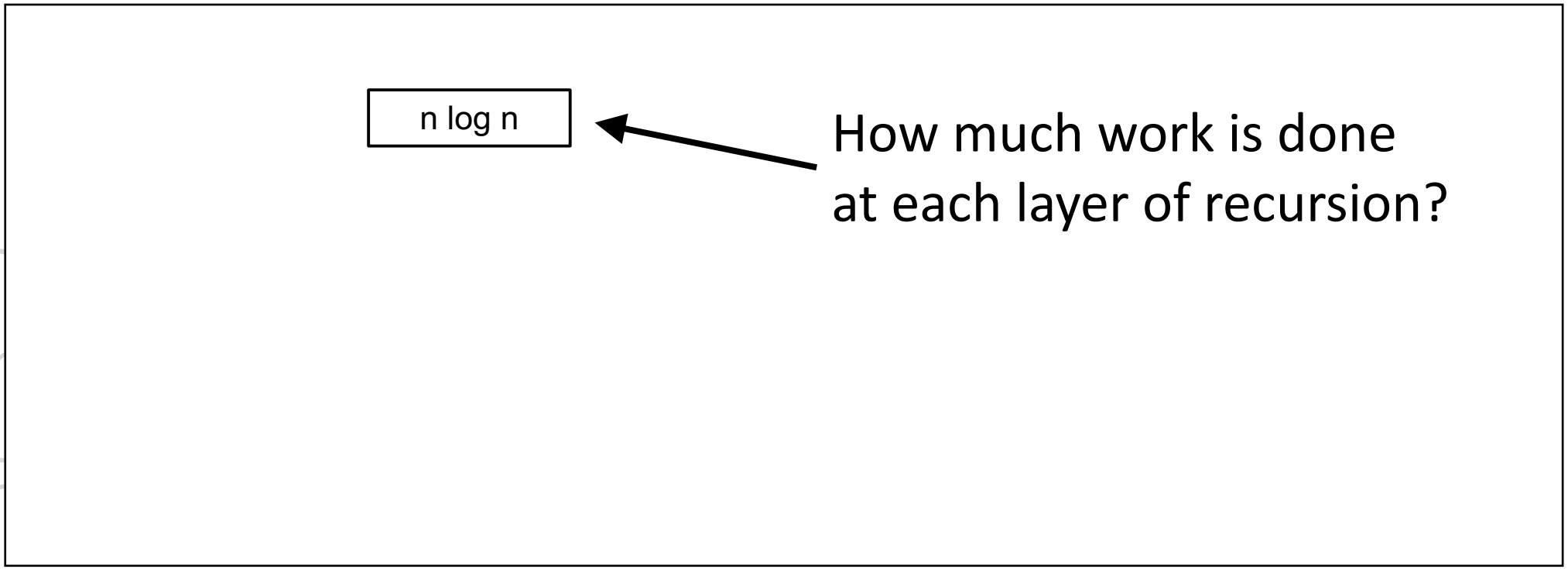


Closest Pair Problem – Algorithm

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. $\boldsymbol{O(1)}$

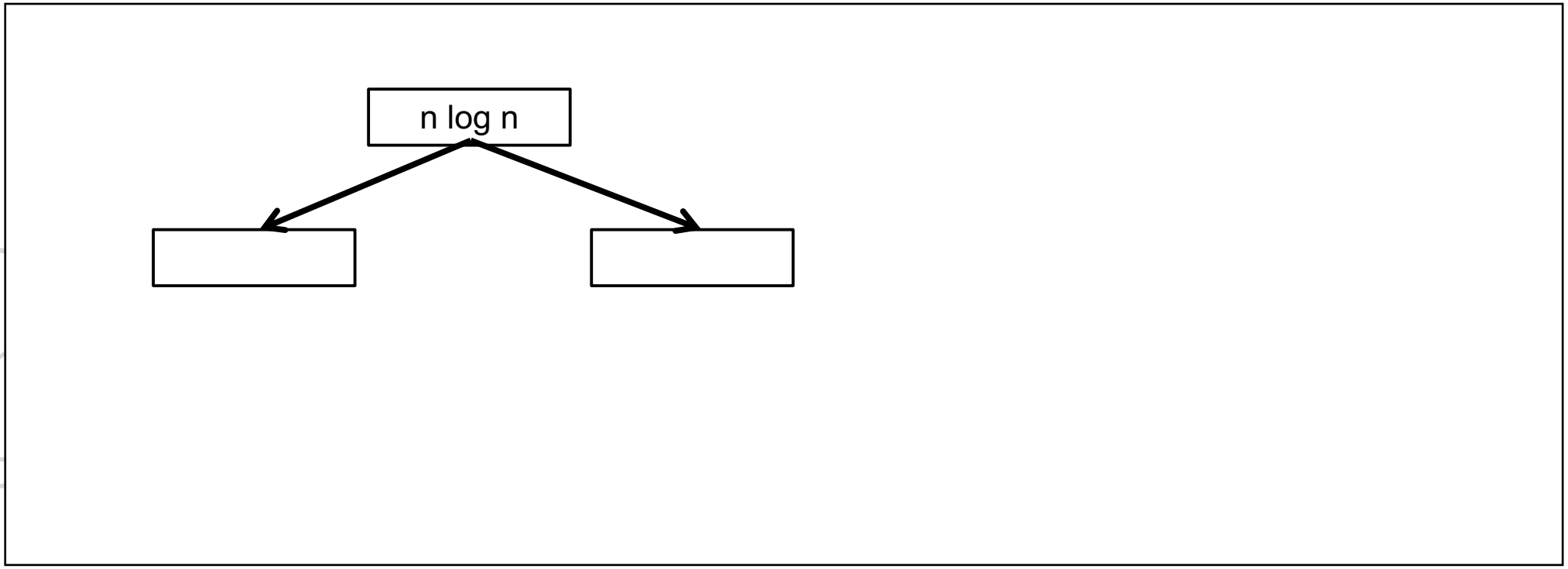
Closest Pair Problem – Algorithm

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



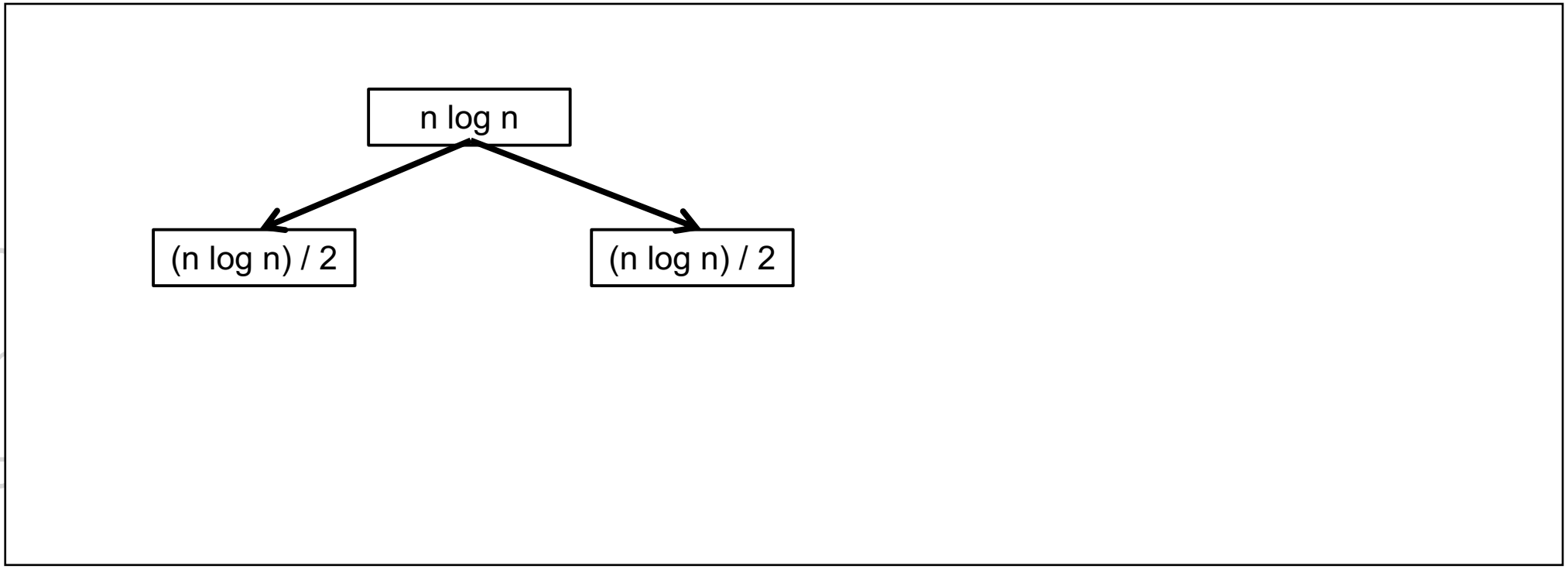
Closest Pair Problem – Algorithm

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



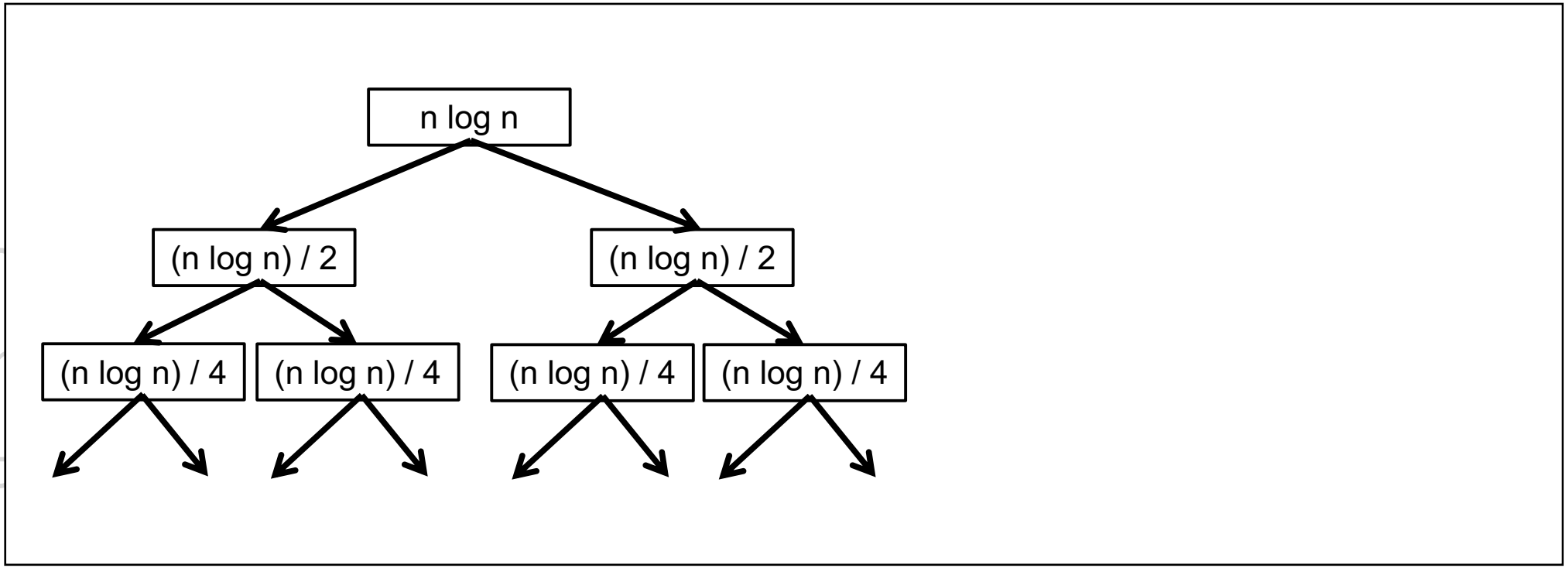
Closest Pair Problem – Algorithm

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



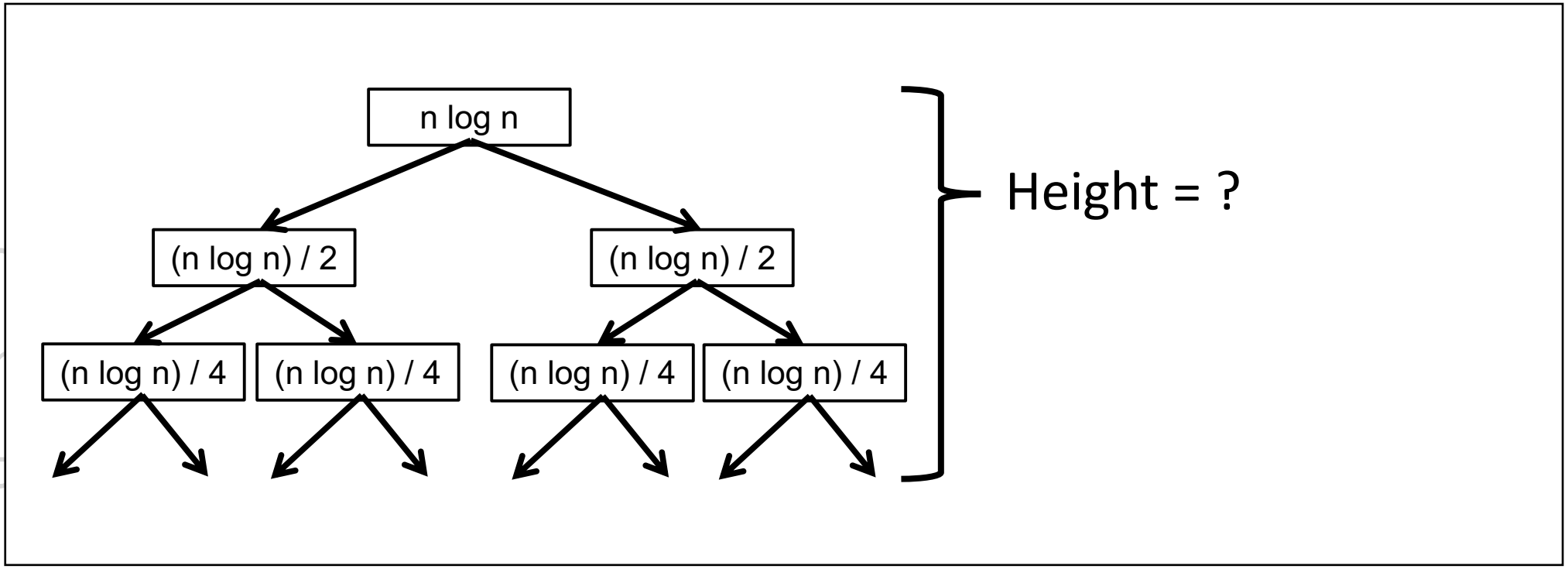
Closest Pair Problem – Algorithm

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



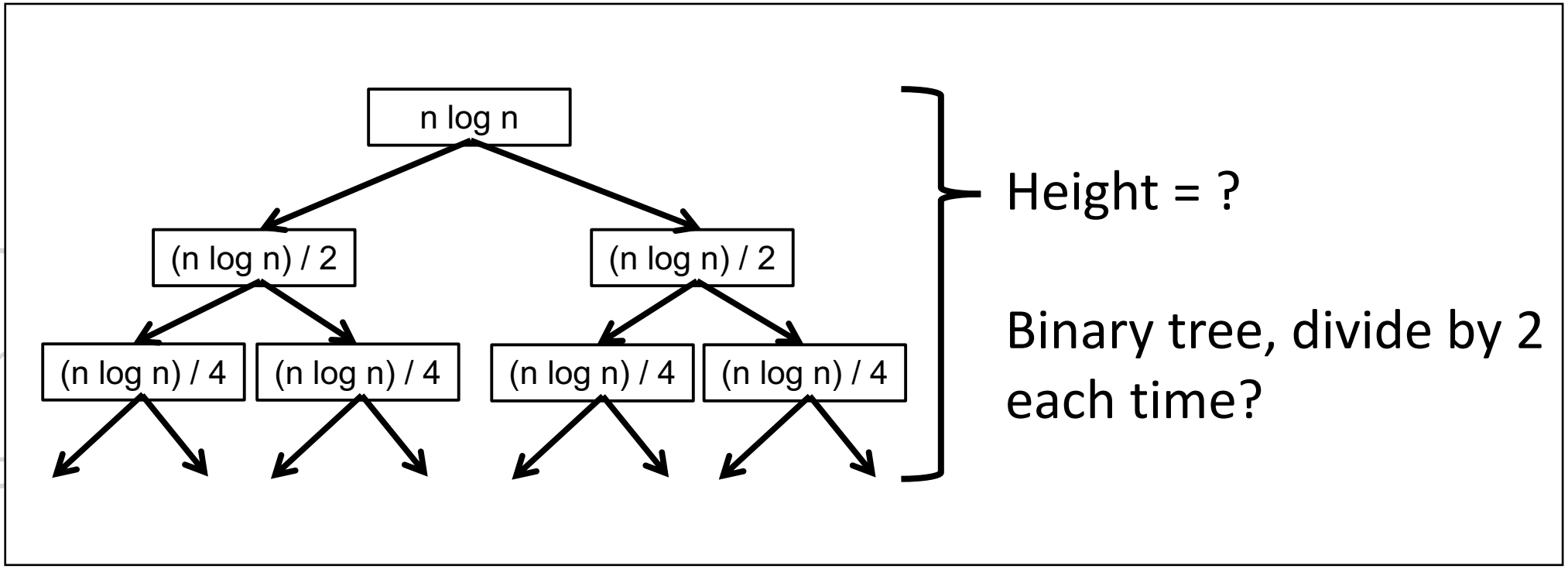
Closest Pair Problem – Algorithm

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



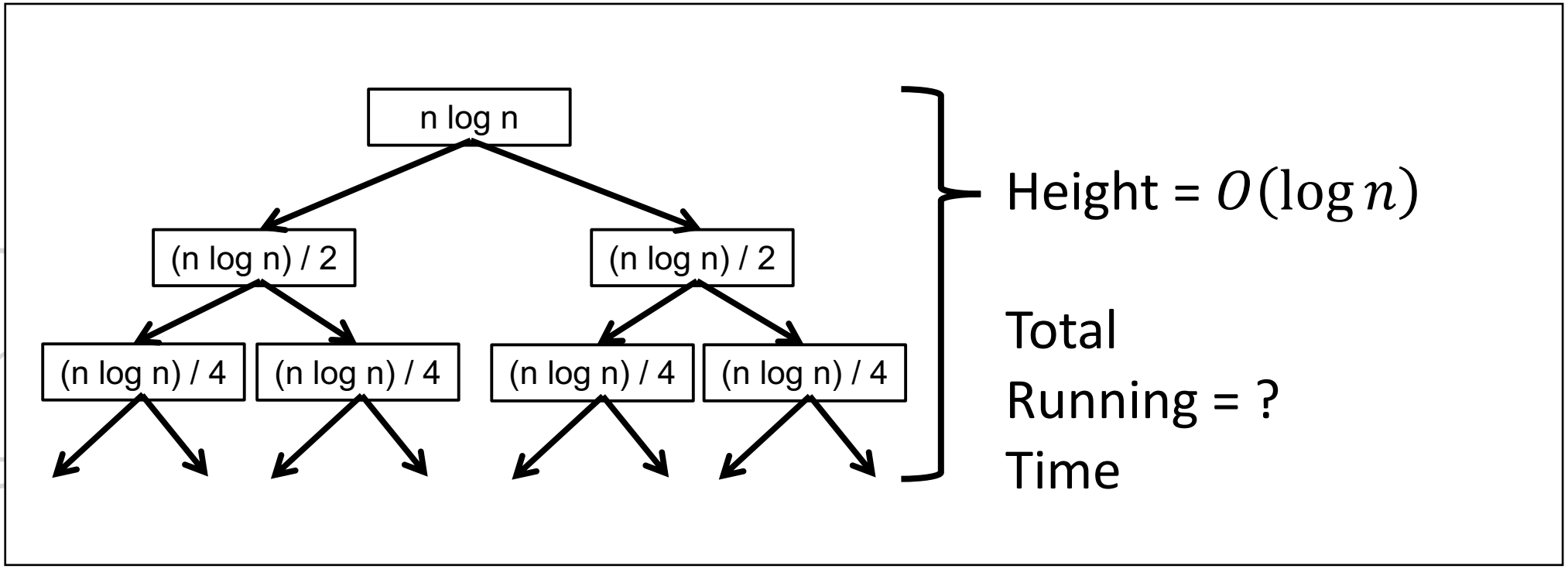
Closest Pair Problem – Algorithm

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



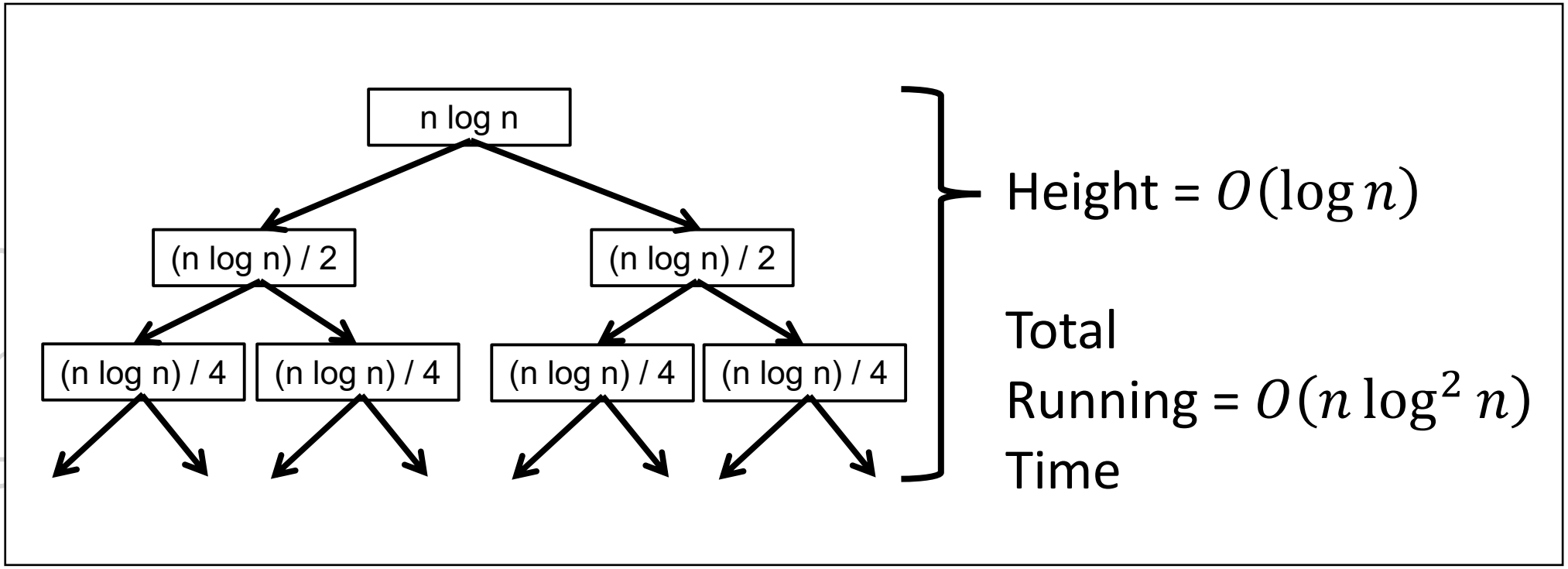
Closest Pair Problem – Algorithm

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

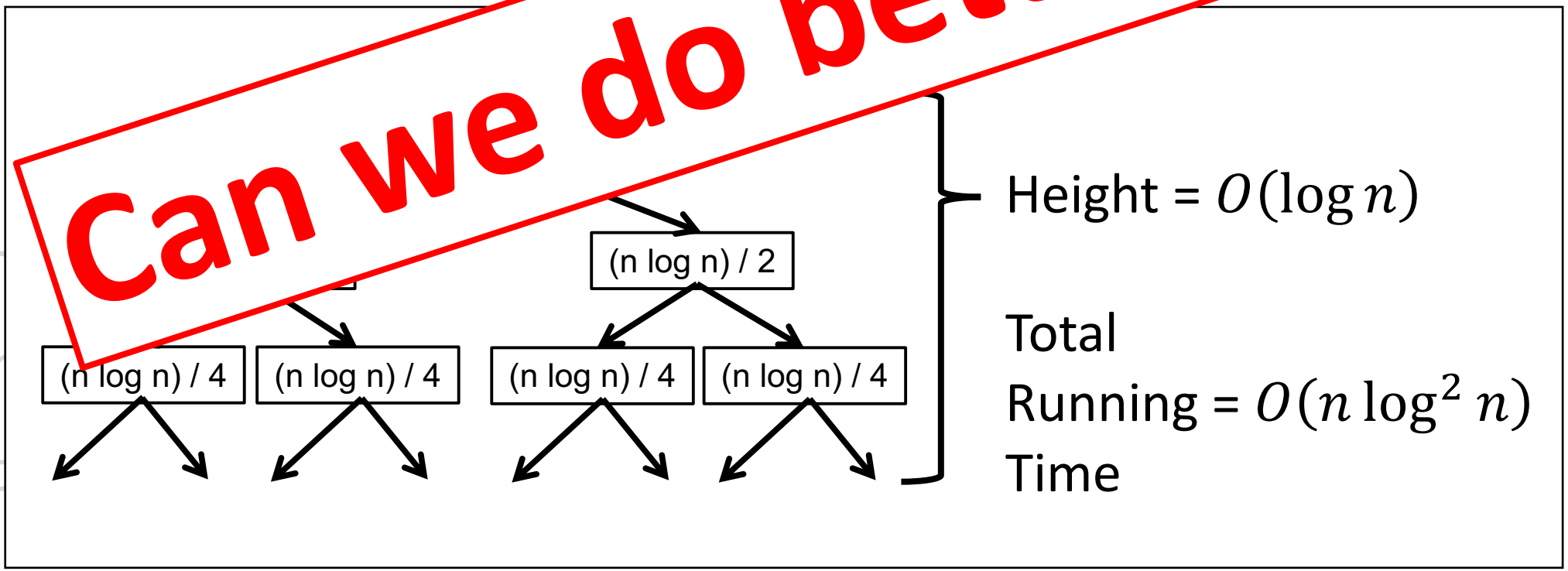
1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



Closest Pair Problem – Algorithm

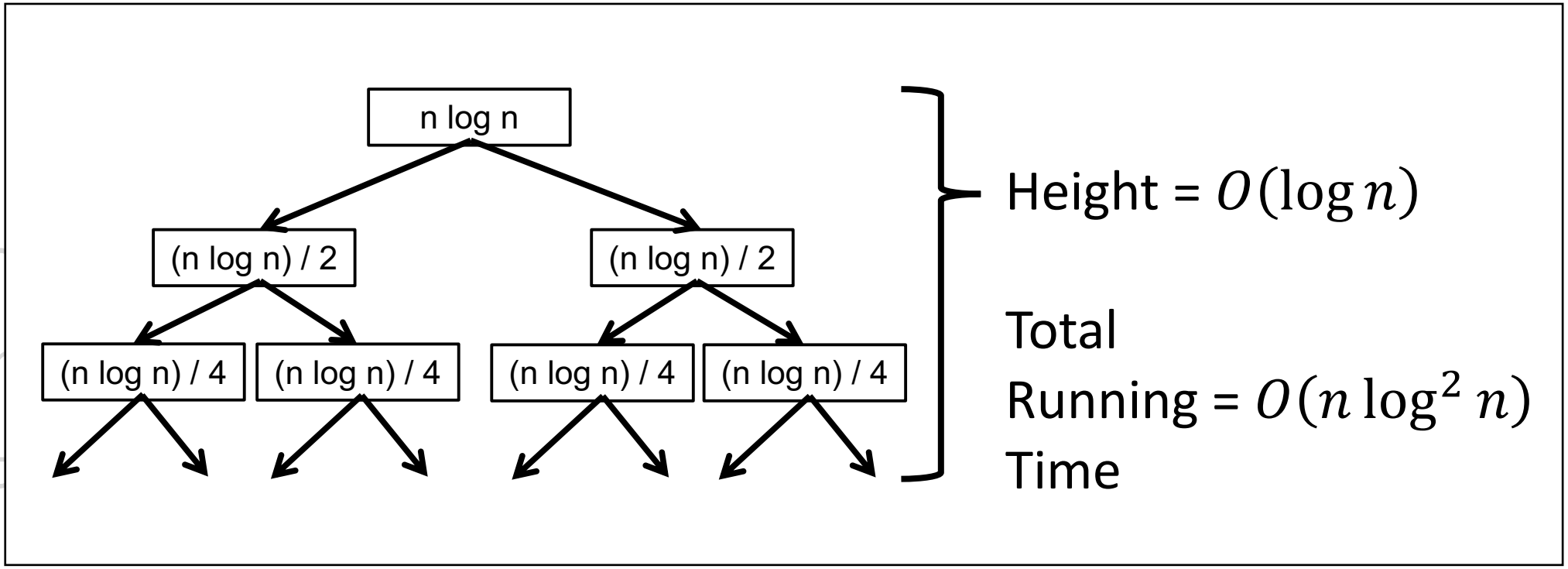
1. Sort points by x-coordinate and make L. $O($ $)$
2. Recursively determine d_{left} and
3. Let
4. Let
5. Sor
6. Cor
7. Ret

Can we do better?



Closest Pair Problem – Algorithm

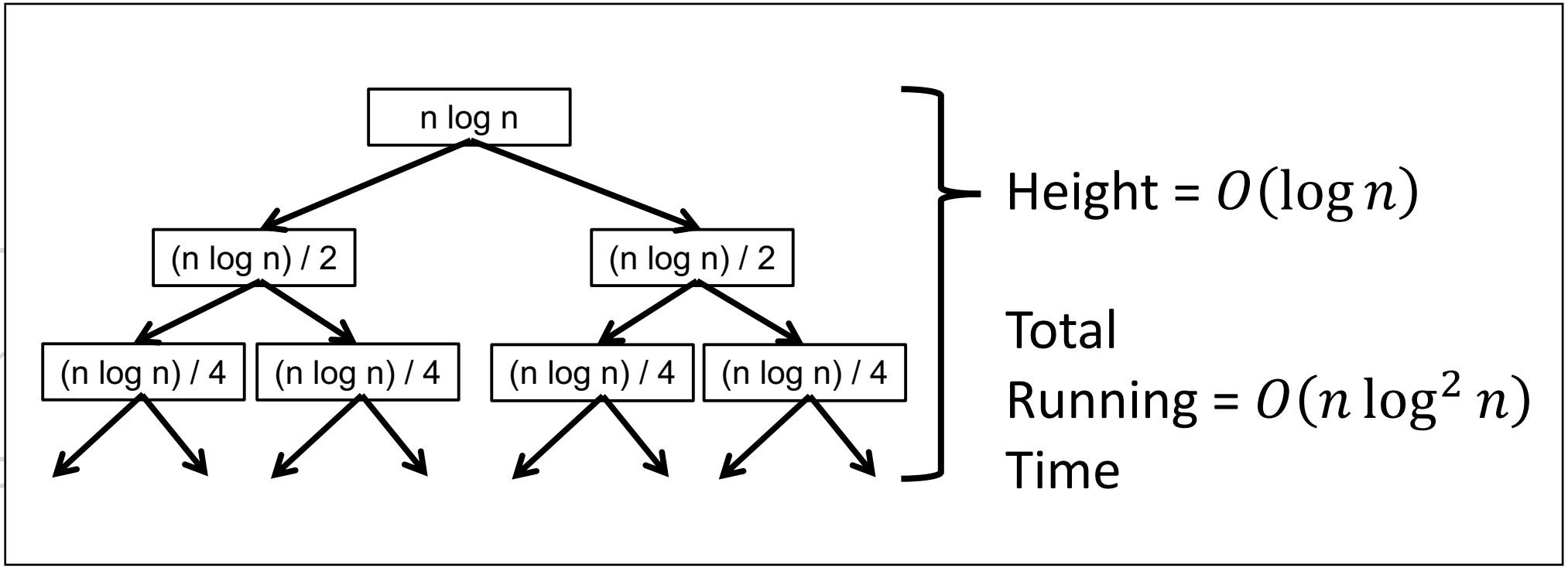
1. Sort points by x-coordinate and make L. $O(n \log n)$
2. Recursively determine d_{left} and d_{right}. TBD
3. Let
4. Let
5. Sor
6. Cor
7. Ret



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

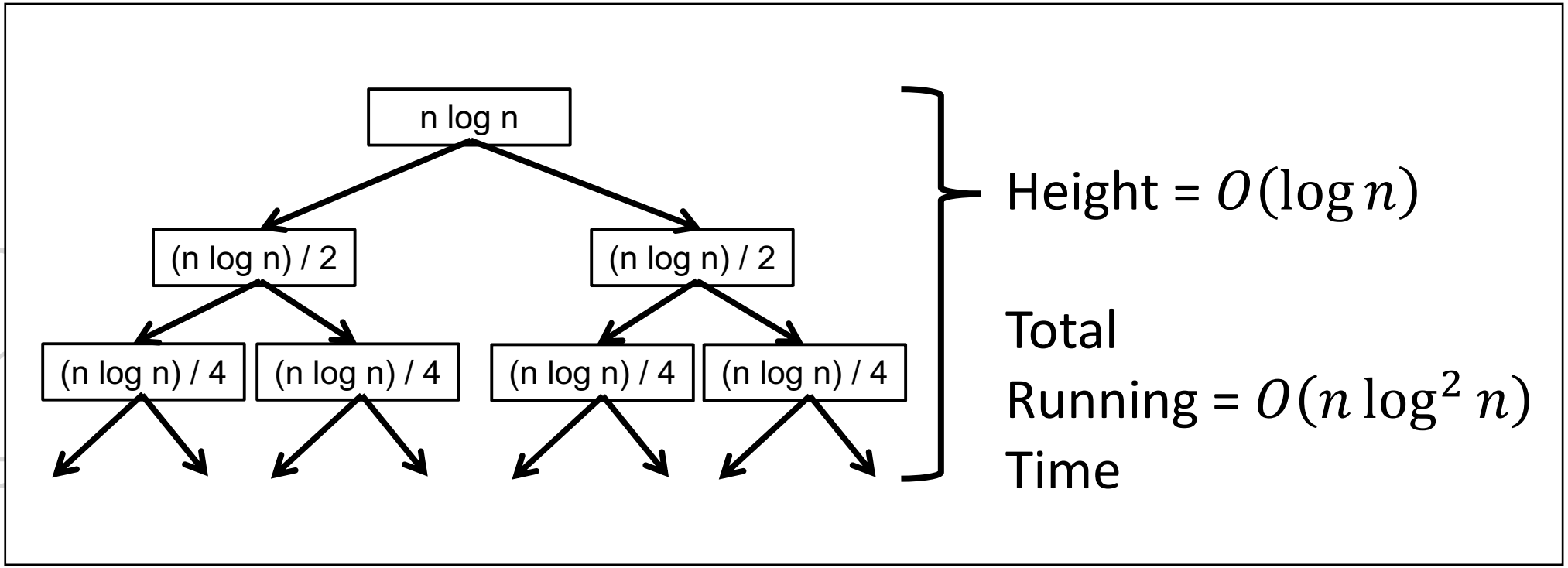


Closest Pair Problem – Algorithm

What is driving our complexity?

The work being done ***in*** each recursive call.

1. S
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. $\boldsymbol{O(n \log n)}$
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. What is driving our complexity in each recursive call?
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. What is driving our complexity in each recursive call?
   Sorting S by y-coordinate.
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 $n)$
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 n) $\boldsymbol{n)}$
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 S be straddle points within δ of 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. $O(n \log n)$**~~
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

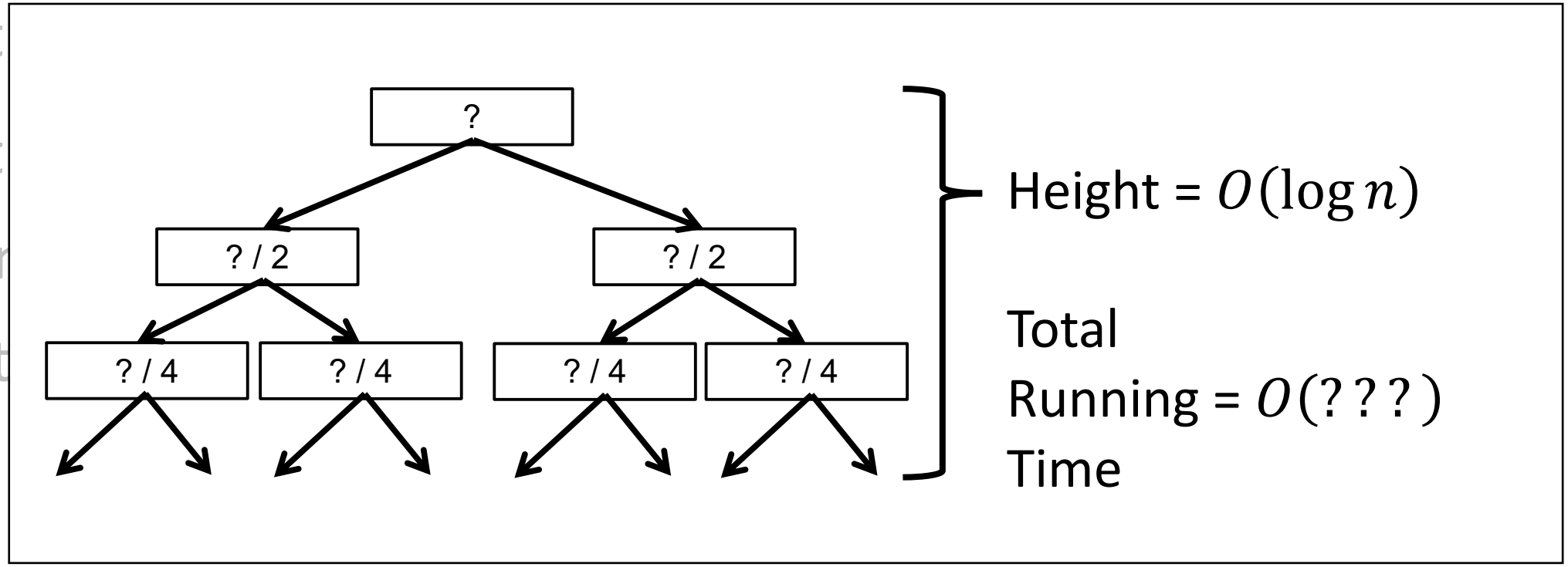
0. Sort by x-coordinate (X) and y-coordinate (Y). $\boldsymbol{O(n \log n)}$
1. Make L and split X and Y. $\boldsymbol{O(n)}$
2. Recursively determine d_{left} and d_{right}. **TBD**
3. Let $\delta = \min(d_{\text{left}}, d_{\text{right}})$. $\boldsymbol{O(1)}$
4. Let S be straddle points within δ of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

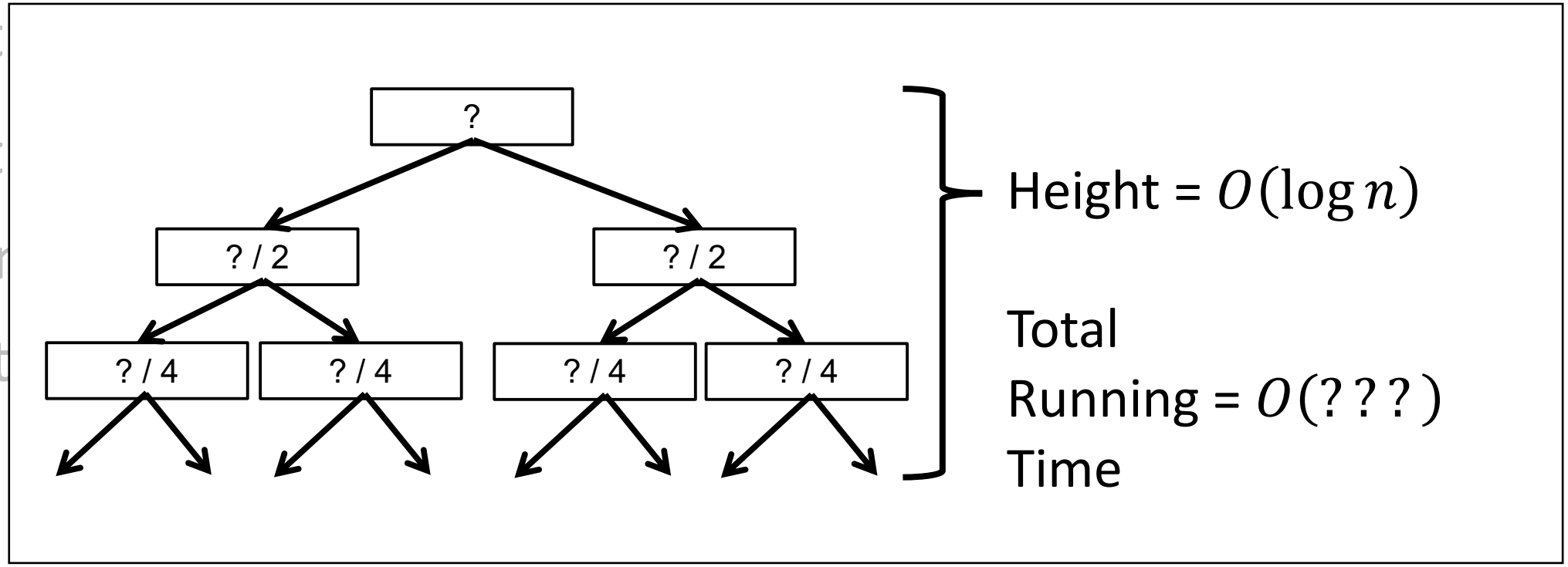


Closest Pair Problem – Algorithm

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. $\boldsymbol{O(1)}$

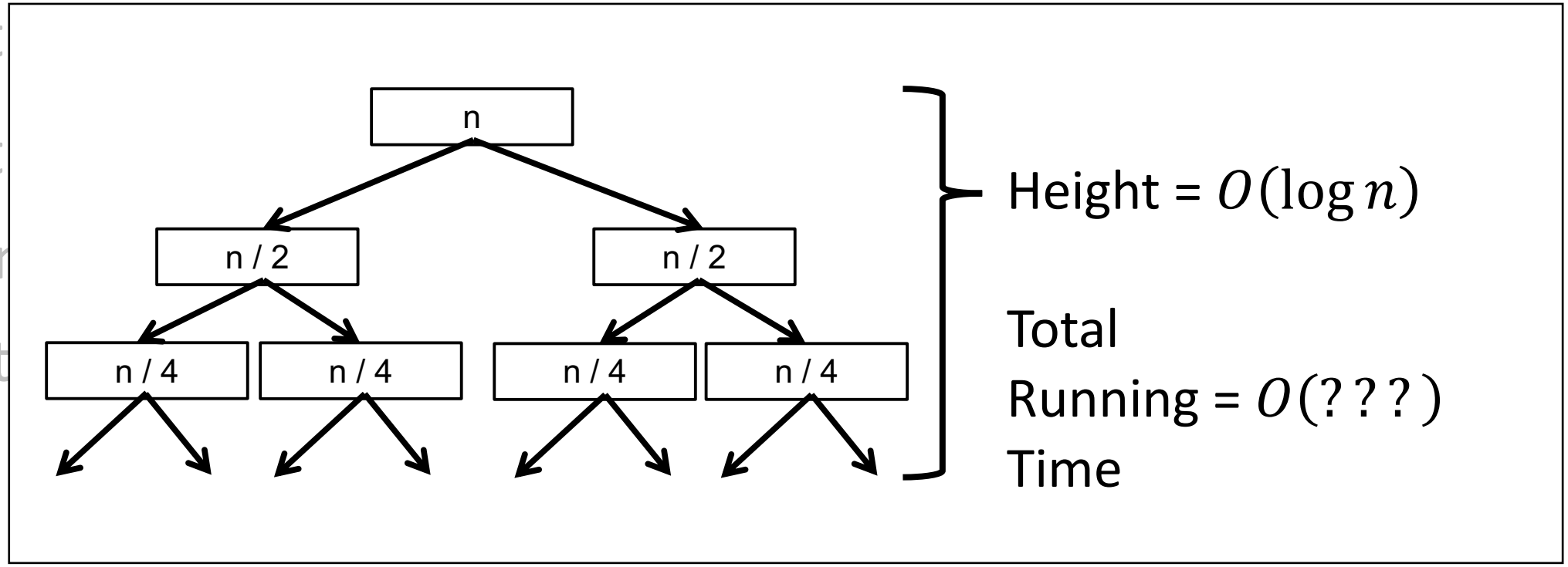
Closest Pair Problem – Algorithm

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



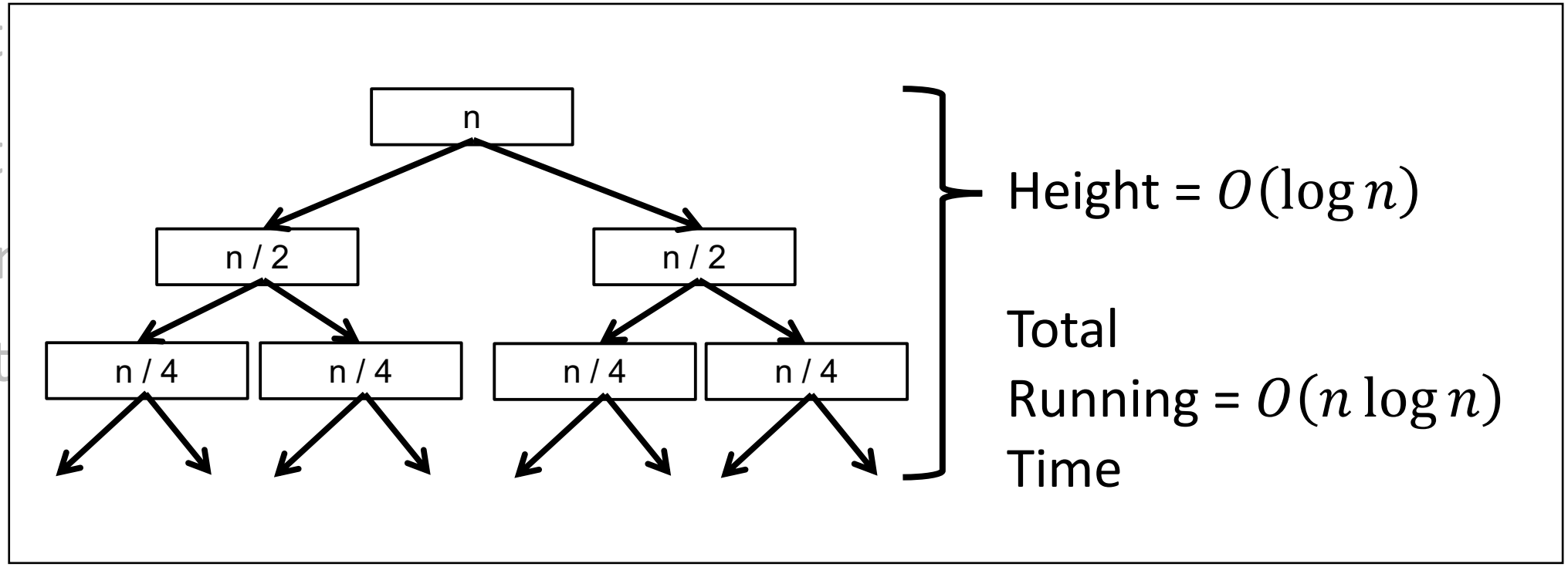
Closest Pair Problem – Algorithm

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



Final Quiz

Same format as Quiz 1 and Quiz 2

- 6AM – 11:59PM

You can use any notes, assignments, slides, lecture recordings, documentation, but just no accessing external resources or AI tools

I'd recommend bringing a scratch piece of paper, notebook, or something you can write on

Quiz Contents

- Graphs
- Graph Algorithms
- Minimum Spanning Tree
- Shortest Path
- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Possibly some stuff about BST and Hash tables

CSCI 232- Data Structures and Algorithms

“Tools”

- Arrays
- Linked Lists
- Stacks/Queues
- **Hash Tables**
- **Trees**
- **Graphs**

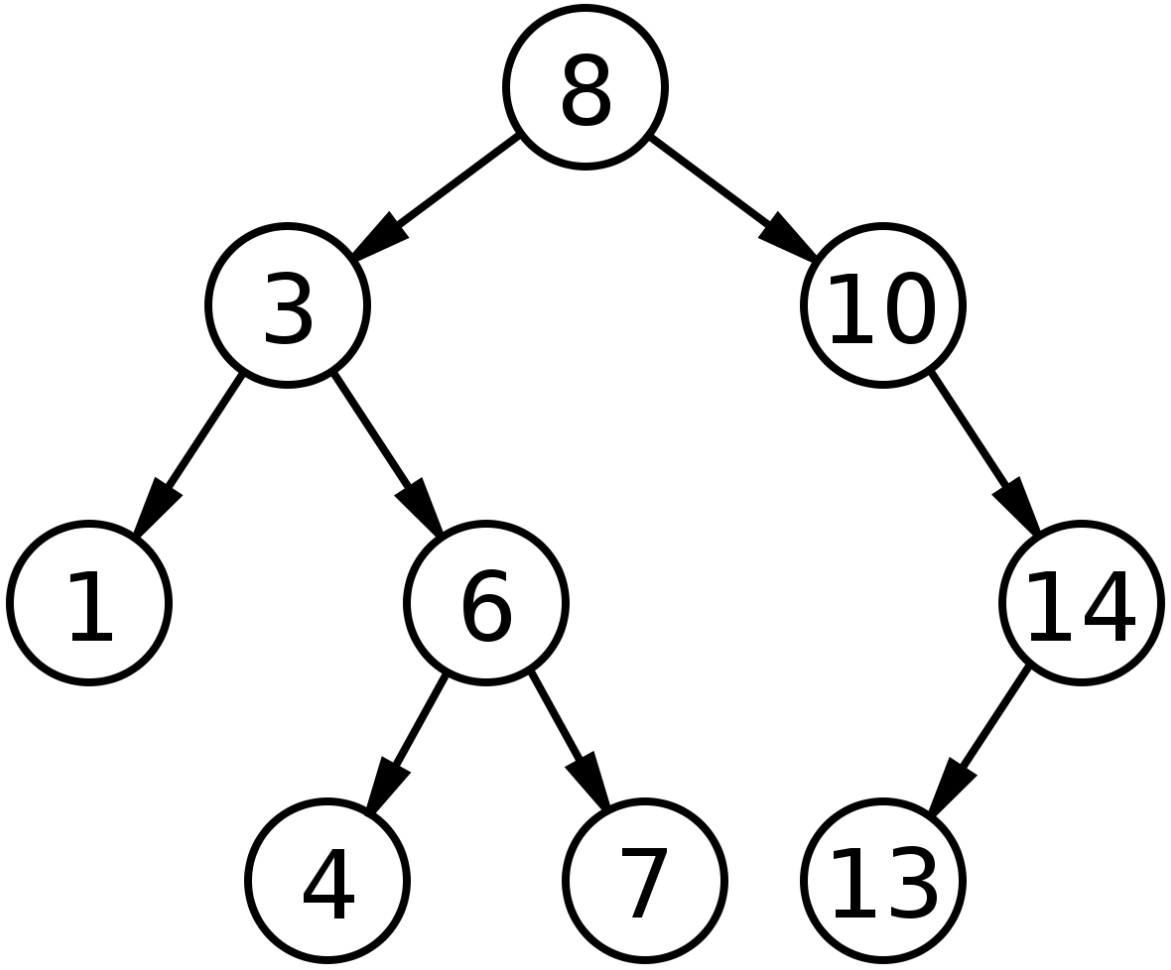
“Use of tools”

- Sorting
- Searching
- Routing
- Optimization

A **data structure** is a mechanism for storing and organizing data

An **algorithm** is a series of instructions to be followed to solve some problem

(Balanced) Binary Search Trees



Class TreeSet<E> Class TreeMap<K,V>

- **O(logn)** Addition Time
- **O(logn)** Removal Time
- **O(logn)** Search Time

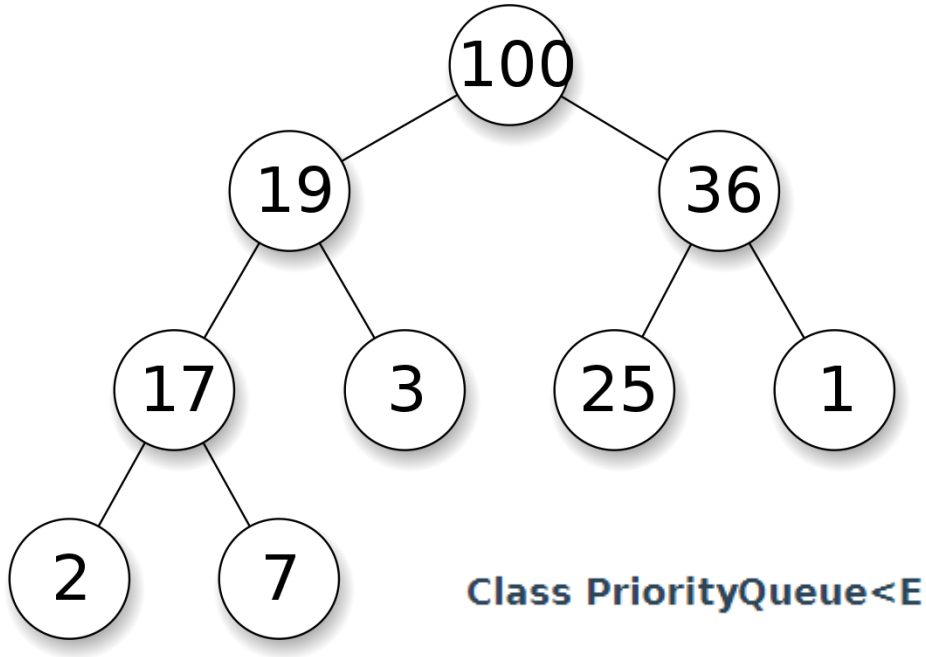
Not as efficient for adding/removing, but much more efficient to search through

How to guarantee balance?
→ Red/Black Trees!

More complex operations, especially in a self-balancing tree (rotations, replacements, etc)

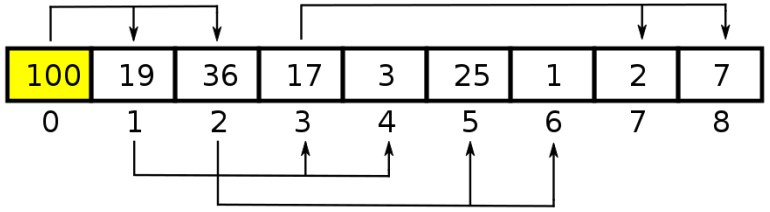
Heap

Tree representation



Class PriorityQueue<E>

Array representation



- **O(logn)** Addition Time
- **O(logn)** Removal Time
- **O(n)** Search Time
- **O(1)** Retrieving Highest Priority Element

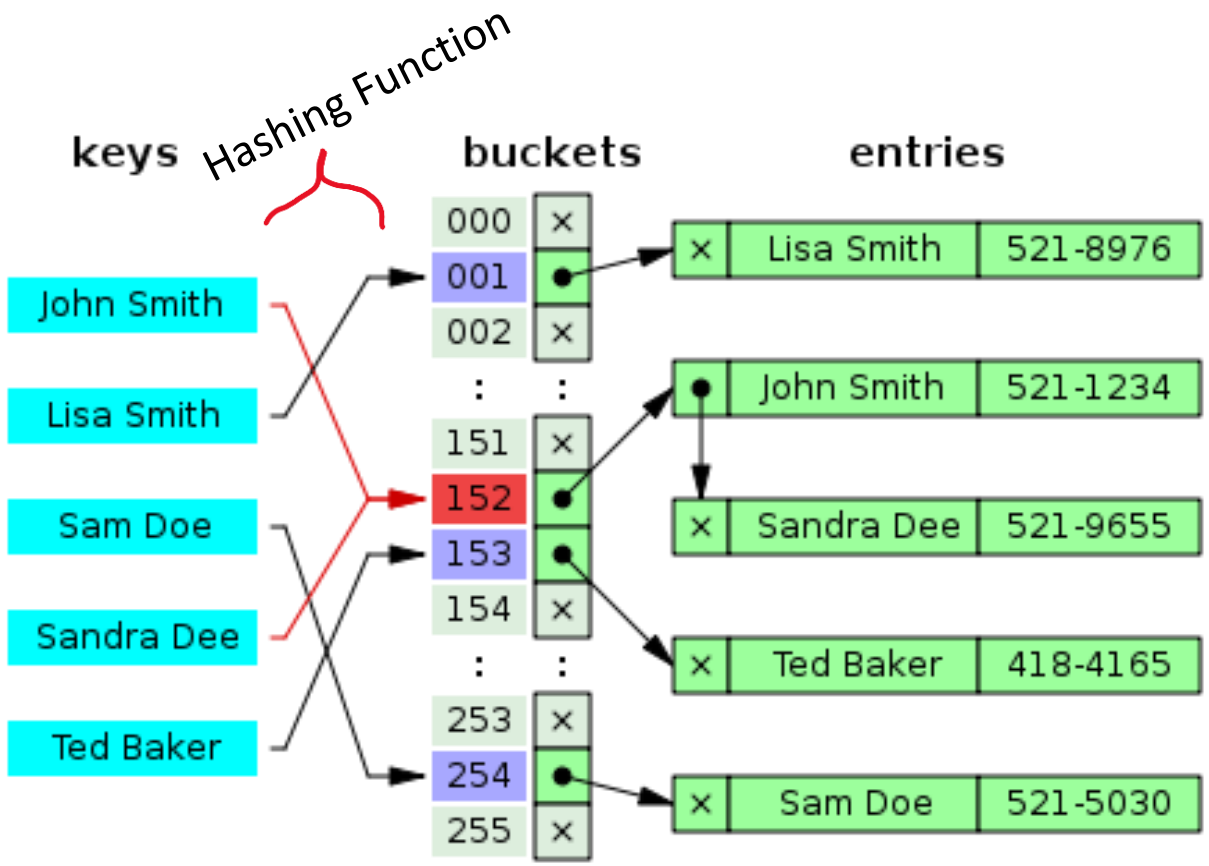
A priority queue!!

Efficient at retrieving the highest priority element

General searches are not as efficient

Creating a heap from an (unsorted) array is also efficient

Hash Tables



Class HashMap<K,V> Class HashSet<E>

(These libraries are very optimized to avoid collisions ☺)

- **O(1)** Addition
- **O(1)** Removal
- **O(1)** Retrieval/Contains
- **O(n)** searching if you don't have key
  (n = # of keys)

<u>Downsides</u>

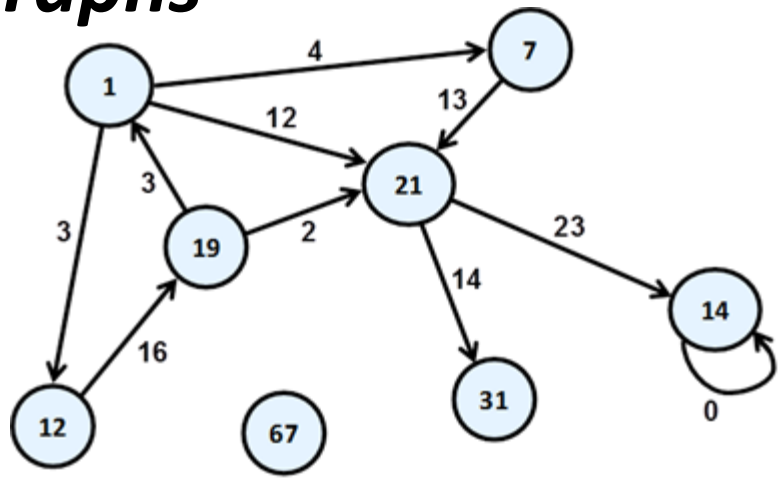
No Duplicate Keys

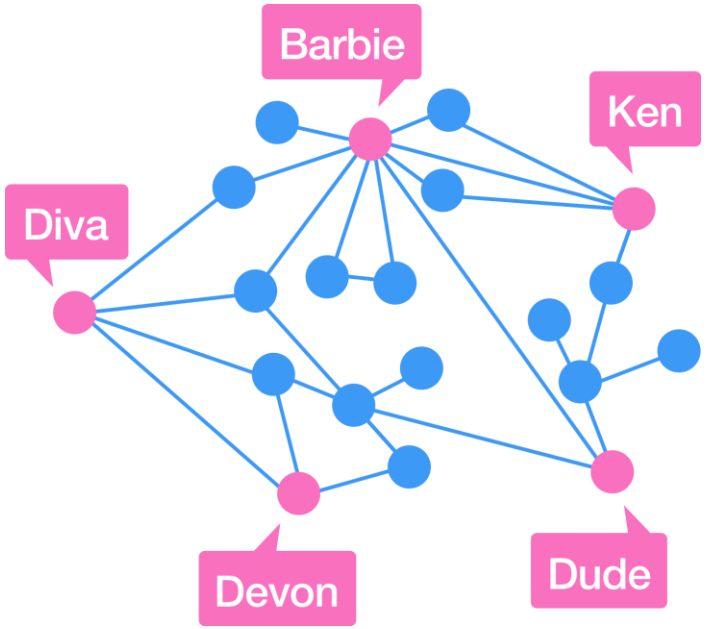
Unordered

Difficult to Sort

Collisions can be spooky

Graphs





A fundamental data structure that can be applied to *many* problems

Many problems that don't seem like a graph problem can be restructured to a graph problem

Many graph algorithms can be done in polynomial time (shortest path, searching, MST)

JGraphT

a Java library of graph theory data structures and algorithms

Algorithms

- Breadth-First
- Depth-First

- Heap Sort

- Hashing Function
- Collision Resolution

- Kruskal's Algorithm (MST)
- Primm's Algorithm (MST)

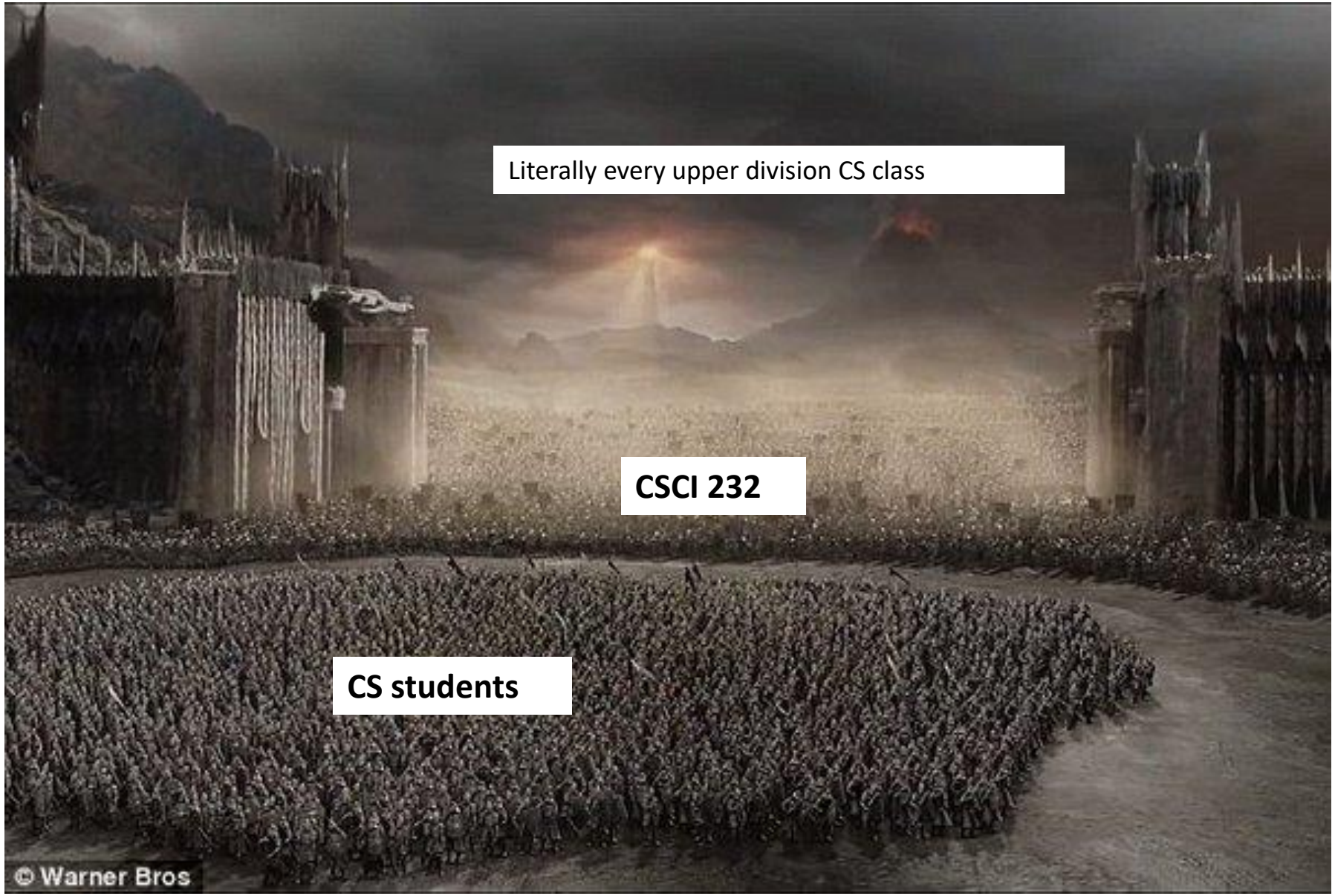
- **Dijkstra's Algorithm** (Shortest Path)
- A* (Shortest Path)

- Greedy Algorithms
  - Knapsack Problem
  - Traveling Salesman

- Dynamic Programming
  - Change Making
  - Rod Cutting
  - Edit Distance

- Divide and Conquer
  - Closest Pair

There are some problems that we don't have an efficient algorithm for!


Literally every upper division CS class
CSCI 232
CS students
© Warner Bros

Thank You!

This class has been fun to teach. I think this class is much more enjoyable and interesting than CSCI 132. *(you built some pretty cool things in this class!)*

I really enjoyed being able to have you for both CSCI 132 *and* CSCI 232

I hope you enjoyed this class, and I hope the stuff you learned will be helpful in your career/future classes (this is one of the **most important** classes you take!)

If I can be of assistance to you for anything in the future (reference, advising, support), please let me know!

I will be teaching CSCI 466, CSCI 476, and CSCI 232 next semester

Connect with me on LinkedIn!

Reese Pearsall (He/Him)
Instructor at Montana State University
Bozeman, Montana, United States · Contact info