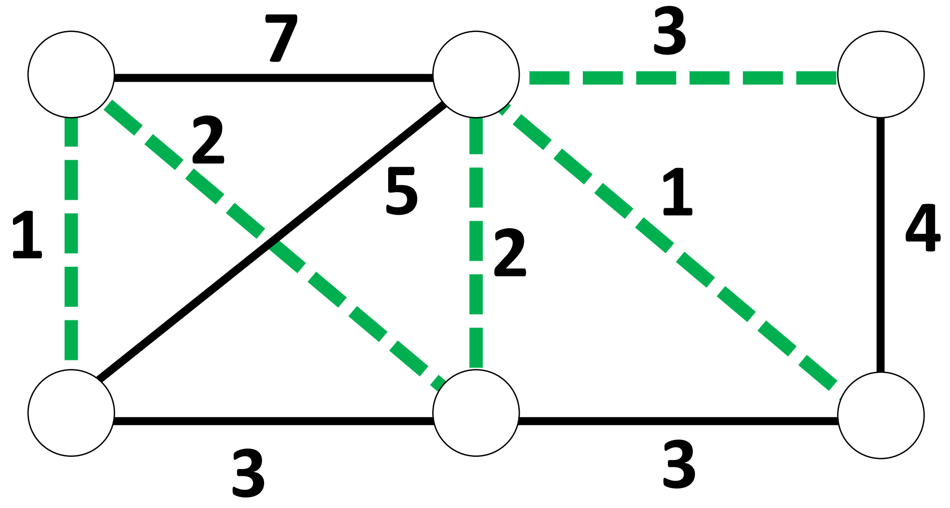
Closest Pair of Points

CSCI 532

Kruskal's MST Algorithm

<u>Algorithm:</u> Add the edge with smallest weight, that does not create a cycle.



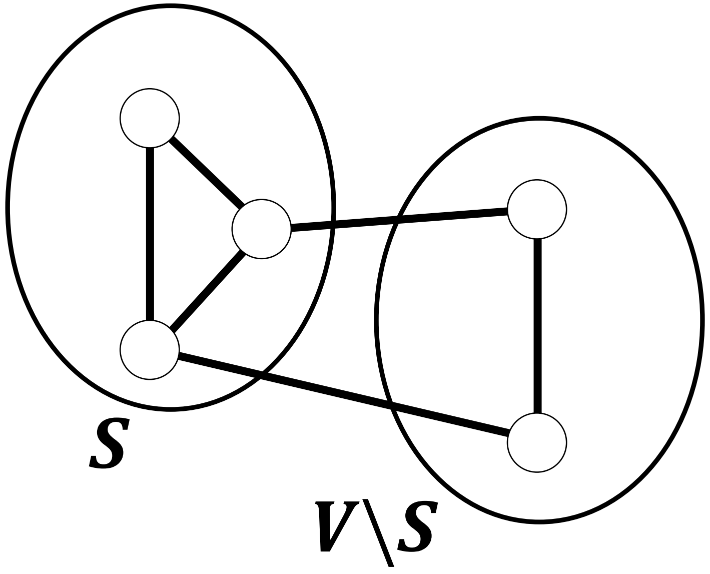
What are some questions we may have about the algorithm?

1. ~~Is the solution valid? (Does it actually find a spanning tree?)~~
2. ~~What is the running time?~~
3. Is the solution optimal?

MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\backslash S$ is part of every MST.

Proof:



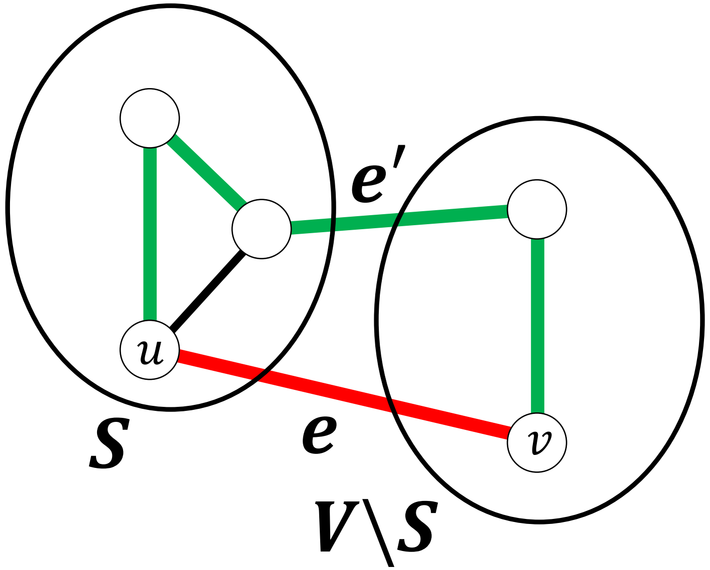
MST Cut Property

<u>Lemma:</u> Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

<u>Proof:</u> Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let $\boldsymbol{e}$ be the cheapest edge between S and $V \backslash S$.

Suppose $\boldsymbol{T}$ is a spanning tree that does not include $\boldsymbol{e}$. $\boldsymbol{T} \cup \{\boldsymbol{e}\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.



MST Cut Property

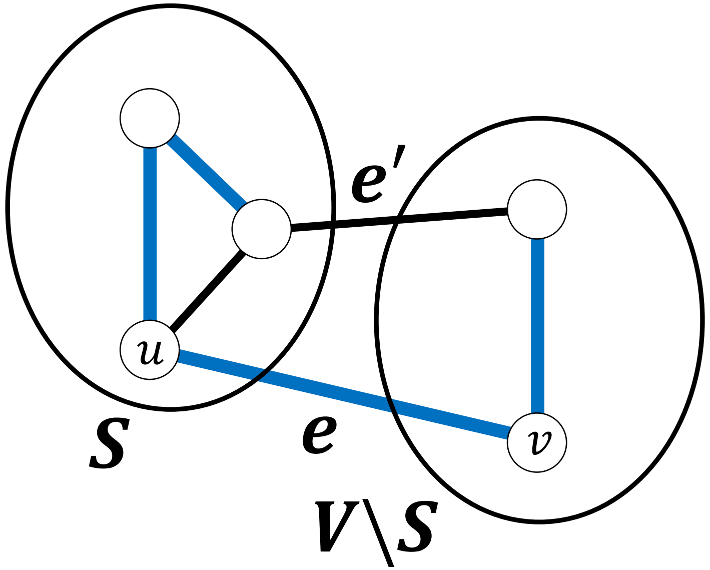
Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \setminus S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \setminus S$ (otherwise it would not be a tree).

Let e be the cheapest edge between S and $V \setminus S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V \setminus S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\} \setminus \{e'\}$.



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

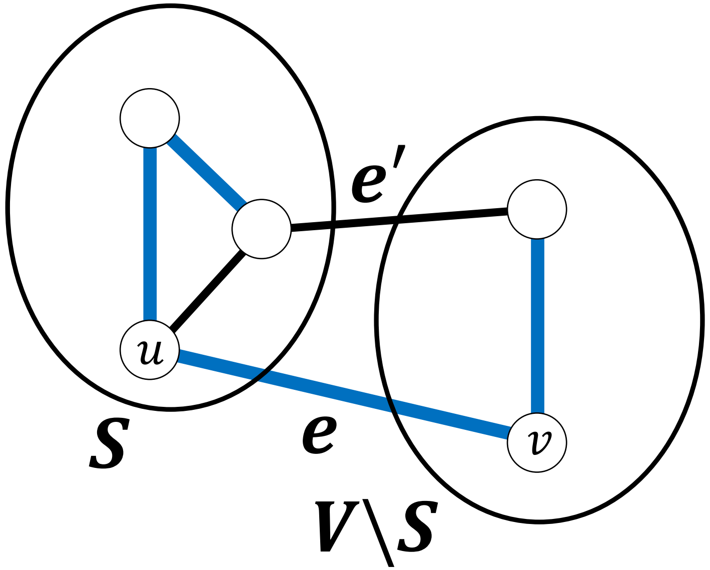
Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let e be the cheapest edge between S and $V \backslash S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\} \backslash \{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:



MST Cut Property

<u>Lemma:</u> Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\backslash S$ is part of every MST.

<u>Proof:</u> Any MST of G must include some edge between S and $V\backslash S$ (otherwise it would not be a tree).

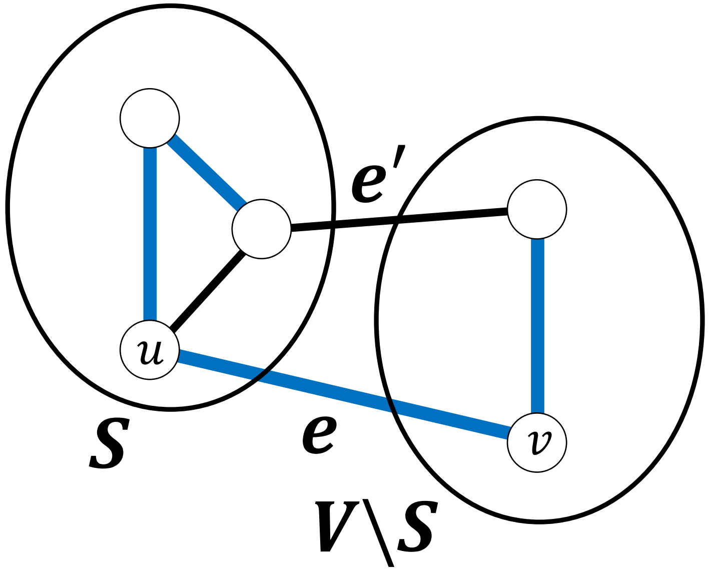
Let e be the cheapest edge between S and $V\backslash S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V\backslash S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\}\backslash\{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:

- $\boldsymbol{T'}$ is a tree (breaking cycle doesn't disconnect graph)



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\setminus S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V\setminus S$ (otherwise it would not be a tree).

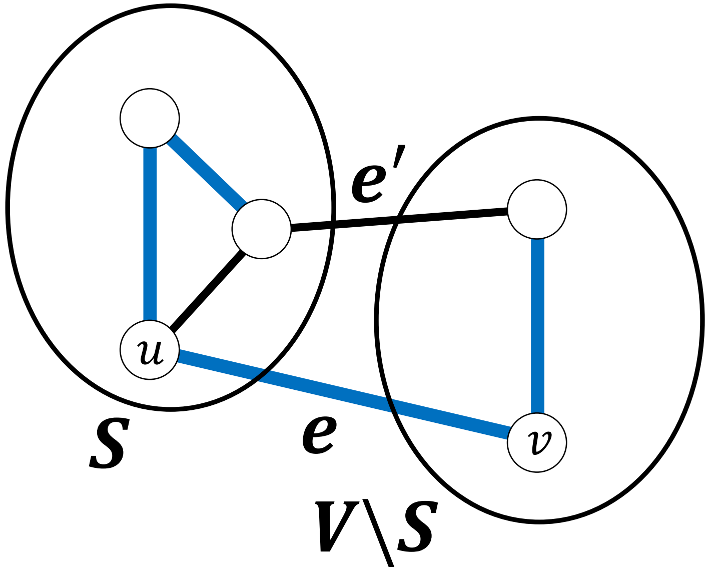
Let e be the cheapest edge between S and $V\setminus S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V\setminus S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\}\setminus\{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:

- $\boldsymbol{T'}$ is a tree (breaking cycle doesn't disconnect graph)
- $\boldsymbol{T'}$ spans V (same number of edges as spanning tree T)



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

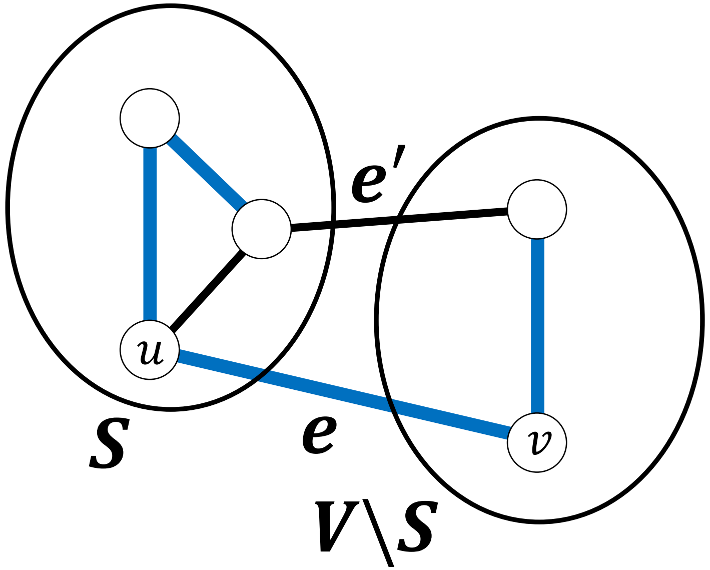
Let e be the cheapest edge between S and $V \backslash S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\} \backslash \{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:

- $\boldsymbol{T'}$ is a tree (breaking cycle doesn't disconnect graph)
- $\boldsymbol{T'}$ spans V (same number of edges as spanning tree T)
- cost($\boldsymbol{T'}$) < cost(T) since e' was replaced by the cheaper e.



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V \backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V \backslash S$ (otherwise it would not be a tree).

Let e be the cheapest edge between S and $V \backslash S$.

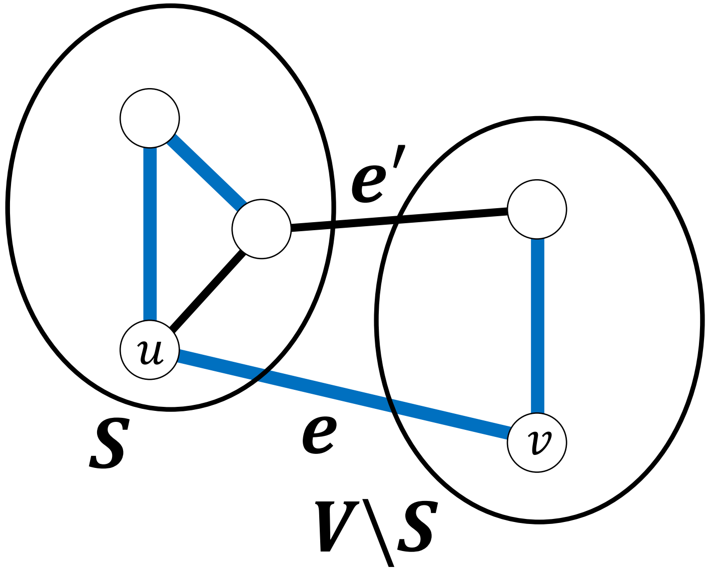
Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V \backslash S$.

Remove e' to form $\boldsymbol{T'} = T \cup \{e\} \backslash \{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:

- $\boldsymbol{T'}$ is a tree (breaking cycle doesn't disconnect graph)
- $\boldsymbol{T'}$ spans V (same number of edges as spanning tree T)
- cost($\boldsymbol{T'}$) < cost(T) since e' was replaced by the cheaper e.

Thus, $\boldsymbol{T'}$ is a cheaper spanning tree



MST Cut Property

Lemma: Suppose that S is a subset of nodes from $G = (V, E)$. Then, the cheapest edge e between S and $V\backslash S$ is part of every MST.

Proof: Any MST of G must include some edge between S and $V\backslash S$ (otherwise it would not be a tree).

Let e be the cheapest edge between S and $V\backslash S$.

Suppose T is a spanning tree that does not include e. $T \cup \{e\}$ must have a cycle and that cycle must have another edge e' between S and $V\backslash S$.

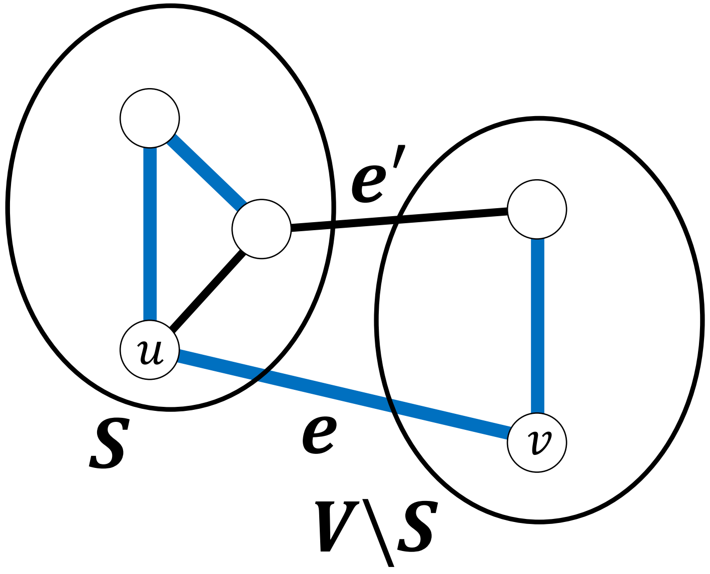
Remove e' to form $\boldsymbol{T'} = T \cup \{e\}\backslash\{e'\}$.

$\boldsymbol{T'}$ is a cheaper spanning tree because:

- $\boldsymbol{T'}$ is a tree (breaking cycle doesn't disconnect graph)
- $\boldsymbol{T'}$ spans V (same number of edges as spanning tree T)
- cost($\boldsymbol{T'}$) < cost(T) since e' was replaced by the cheaper e.

Thus, $\boldsymbol{T'}$ is a cheaper spanning tree

⇒ Every MST must include e.



Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

??

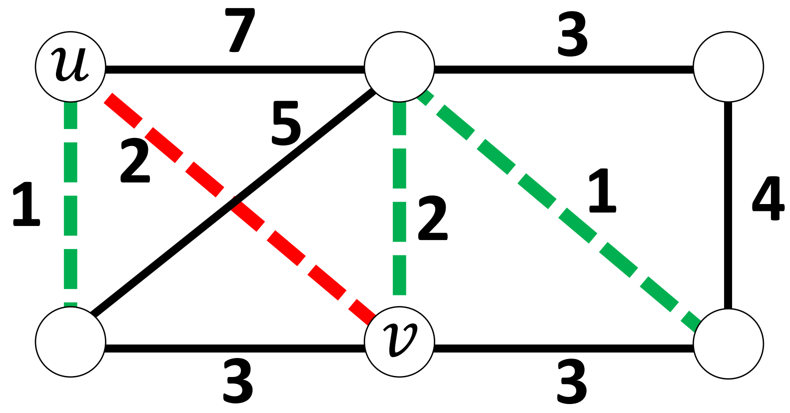
Lemma: The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

Kruskal’s MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal’s algorithm.



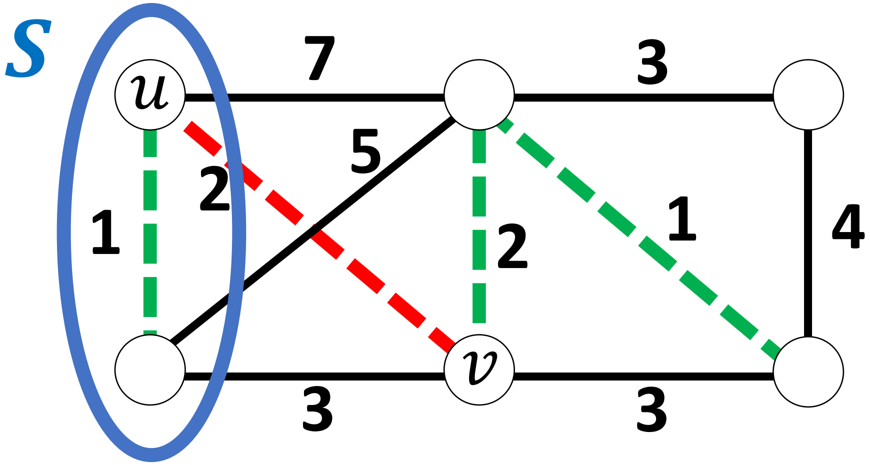
Lemma: The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

Kruskal's MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal's algorithm.

Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal's algorithm. Let $\boldsymbol{S}$ be the set of u and all nodes already connected to u. (or v and all nodes connected to v)



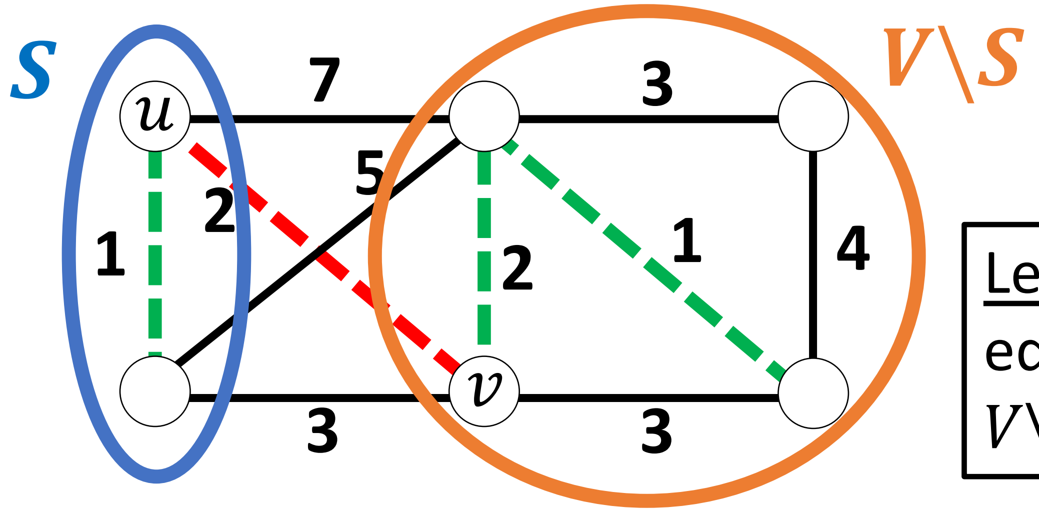
Lemma: The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

Kruskal’s MST Algorithm

<u>Algorithm:</u> Add the edge with smallest weight, that does not create a cycle.

<u>Proof of optimality:</u> Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal’s algorithm. Let $\boldsymbol{S}$ be the set of u and all nodes already connected to u. Clearly $u \in \boldsymbol{S}$ and $v \in \boldsymbol{V \backslash S}$ (otherwise adding $\boldsymbol{e}$ would have created a cycle).



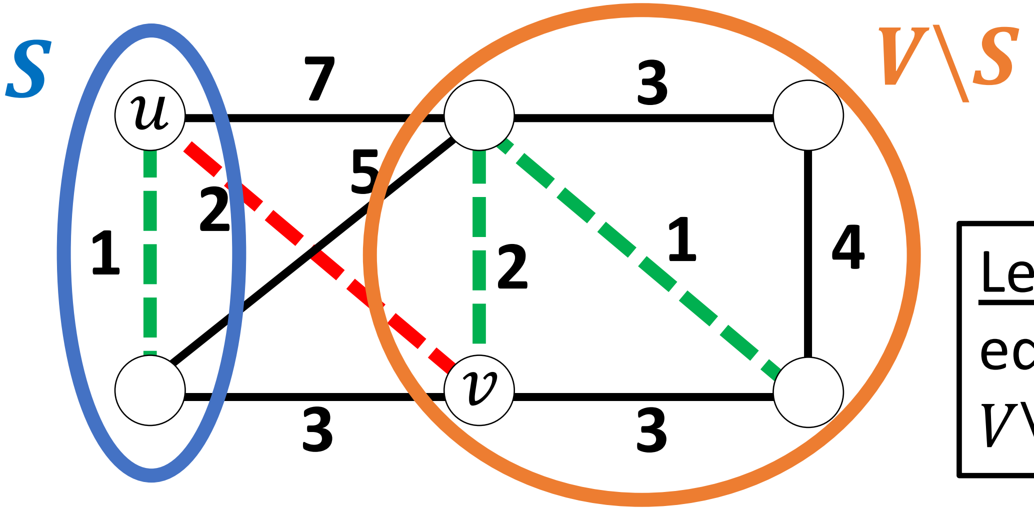
<u>Lemma:</u> The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

Kruskal's MST Algorithm

<u>Algorithm:</u> Add the edge with smallest weight, that does not create a cycle.

<u>Proof of optimality:</u> Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal's algorithm.

Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal's algorithm. Let $\boldsymbol{S}$ be the set of u and all nodes already connected to u. Clearly $u \in \boldsymbol{S}$ and $v \in \boldsymbol{V \backslash S}$ (otherwise adding $\boldsymbol{e}$ would have created a cycle). We are picking the cheapest edge that crosses the cut (otherwise the cheaper edge would have been selected since it would not have created a cycle either).



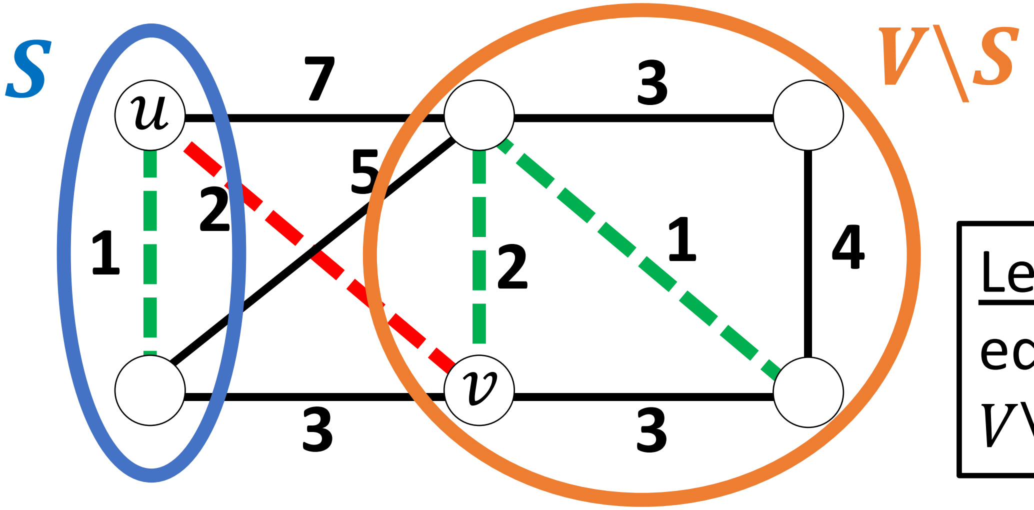
<u>Lemma:</u> The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

Kruskal's MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal's algorithm.

Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal's algorithm. Let $\boldsymbol{S}$ be the set of u and all nodes already connected to u. Clearly $u \in \boldsymbol{S}$ and $v \in \boldsymbol{V \backslash S}$ (otherwise adding $\boldsymbol{e}$ would have created a cycle). We are picking the cheapest that crosses the cut (otherwise the cheaper edge would have been selected since it would not have created a cycle either). By the cut property lemma, this edge must be part of the MST.



Lemma: The cheapest edge between $S \subseteq V$ and $V \backslash S$ is part of every MST.

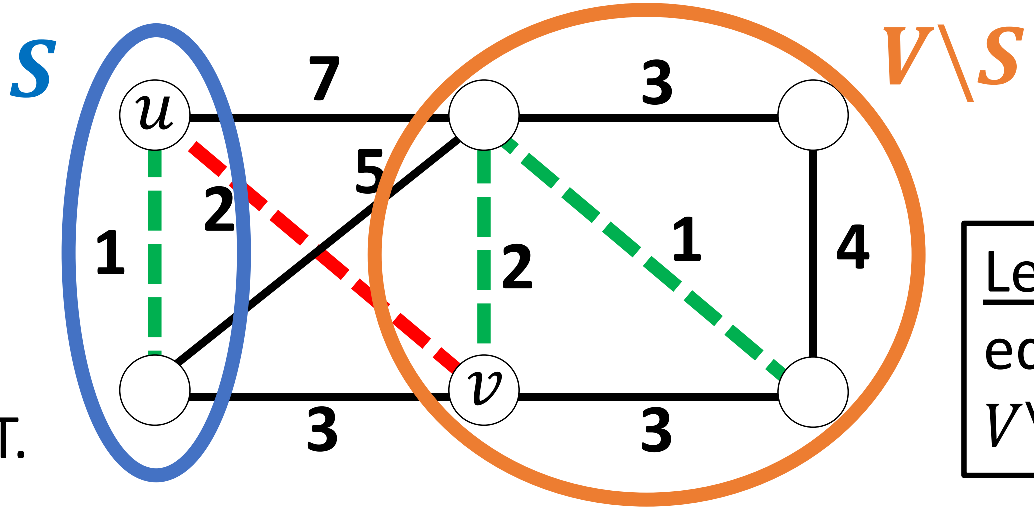
Kruskal's MST Algorithm

Algorithm: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal's algorithm.

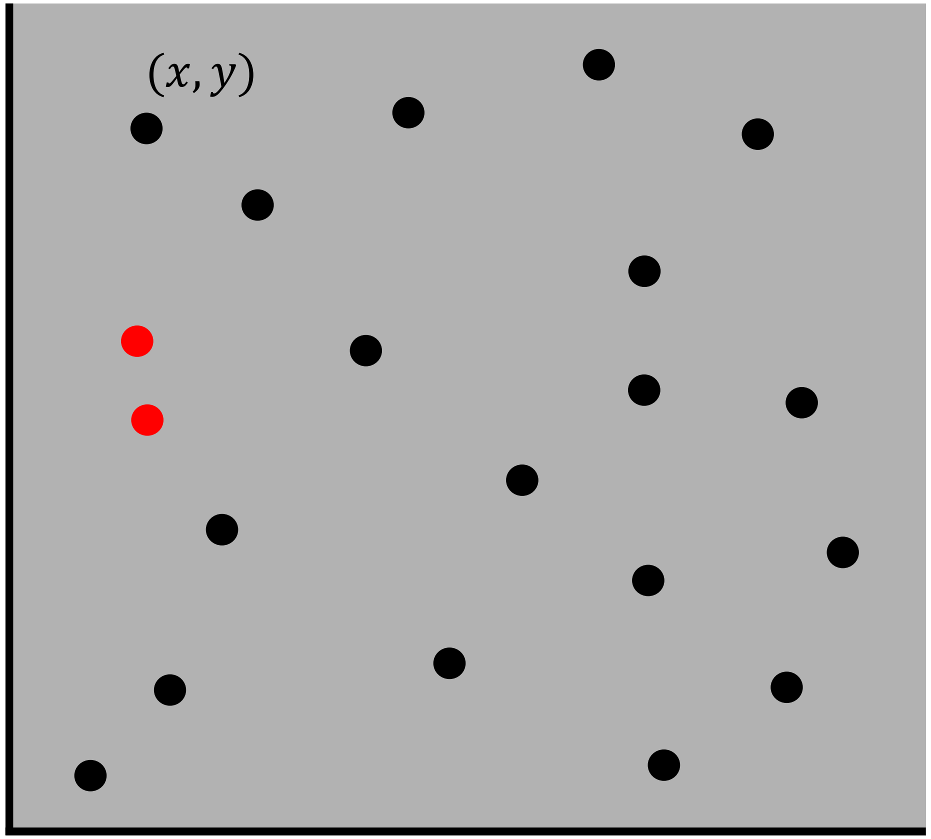
Consider the interation that some edge $\boldsymbol{e} = (u, v)$ is added by Kruskal's algorithm. Let $\boldsymbol{S}$ be the set of u and all nodes already connected to u. Clearly $u \in \boldsymbol{S}$ and $v \in \boldsymbol{V\backslash S}$ (otherwise adding $\boldsymbol{e}$ would have created a cycle). We are picking the cheapest that crosses the cut (otherwise the cheaper edge would have been selected since it would not have created a cycle either). By the cut property lemma, this edge must be part of the MST.

Thus, every edge found by Kruskal's algorithm is part of the MST, and since the edges found form a spanning tree, it is the MST.



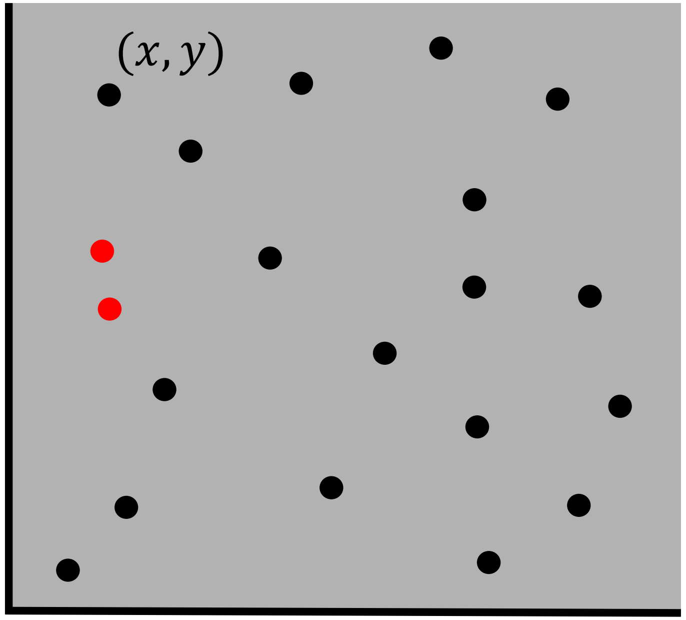
Lemma: The cheapest edge between $S \subseteq V$ and $V\backslash S$ is part of every MST.

Closest Pair Problem



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

Closest Pair Problem



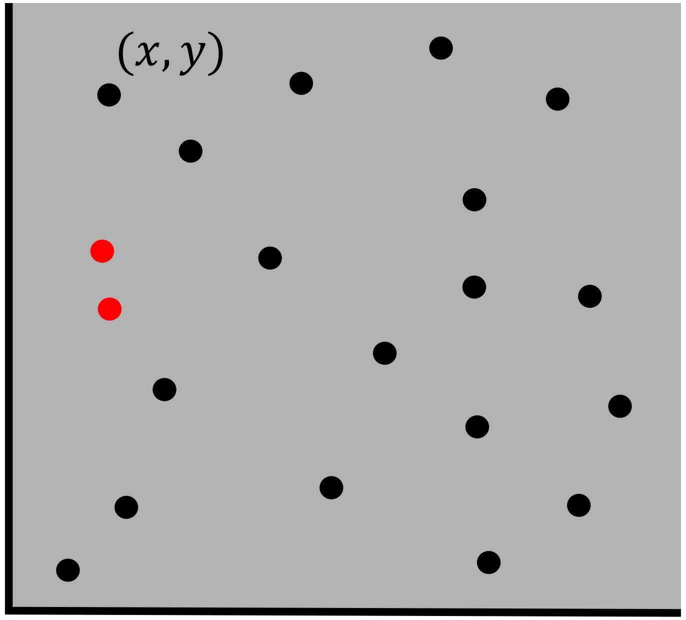
	P_1	P_2	…	P_n
P_1	/	$d_{1,2}$	…	$d_{1,n}$
P_2	$d_{2,1}$	/	…	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	…	/

Simple solution:

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

Running Time = ?

Closest Pair Problem



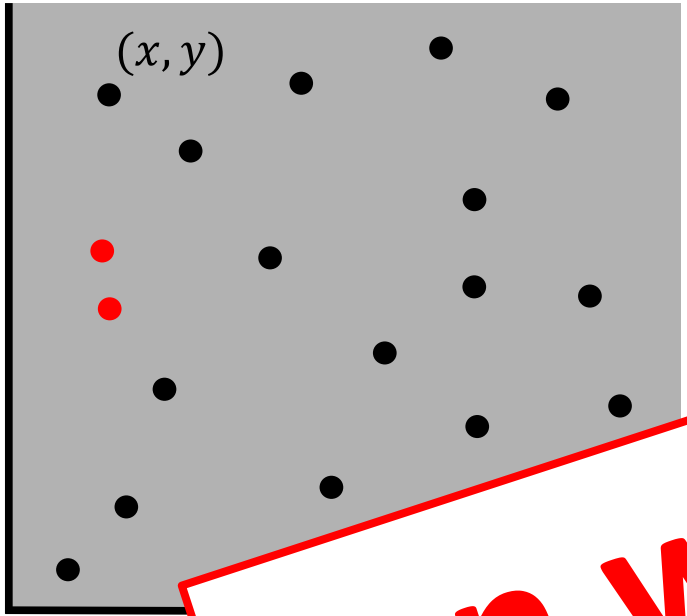
	P_1	P_2	…	P_n
P_1	/	$d_{1,2}$	…	$d_{1,n}$
P_2	$d_{2,1}$	/	…	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	…	/

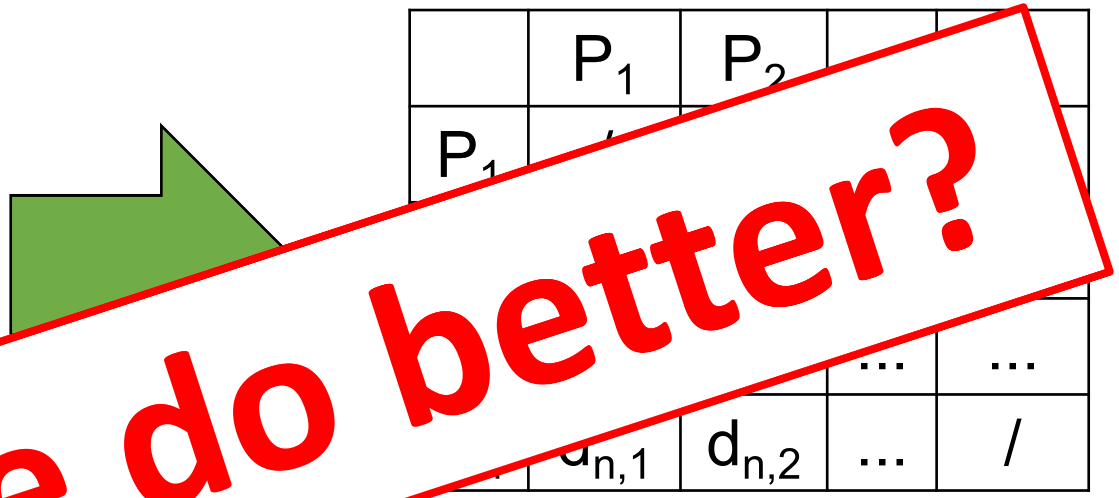
Simple solution:

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

Running Time = $O(n^2)$

Closest Pair Problem





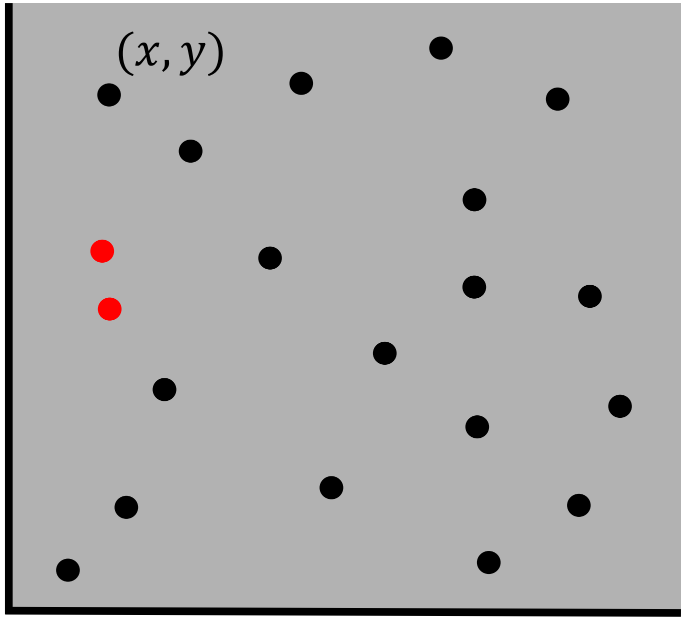
Can we do better?

solution:

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

Running Time = $O(n^2)$

Closest Pair Problem

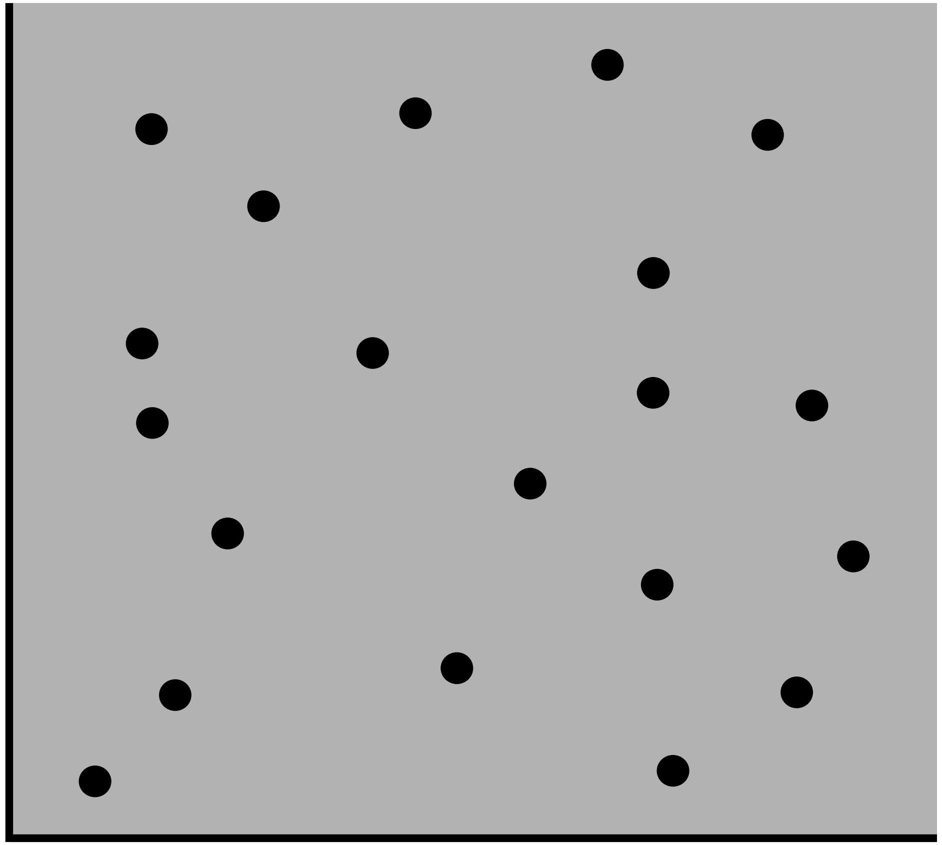


	P_1	P_2	…	P_n
P_1	/	$d_{1,2}$	…	$d_{1,n}$
P_2	$d_{2,1}$	/	…	$d_{2,n}$
…	…	…	…	…
P_n	$d_{n,1}$	$d_{n,2}$	…	/

Divide and Conquer Algorithms:

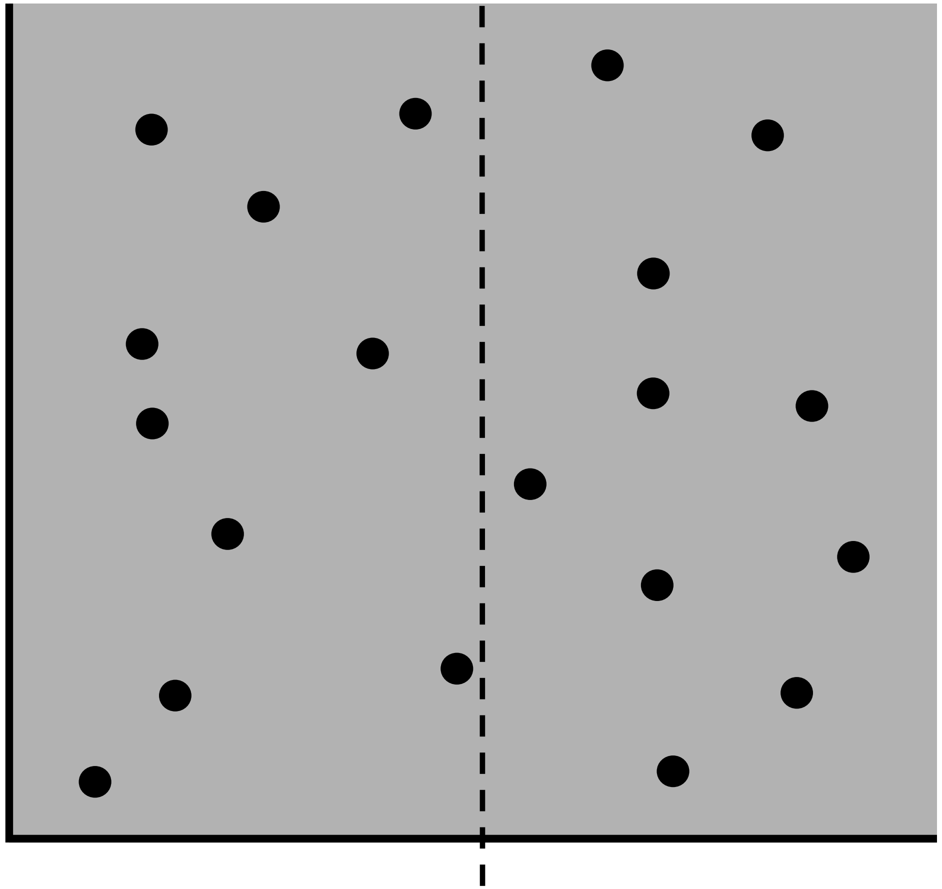
- Divide into subproblems that are smaller instances of the original.
- "Conquer" the subproblems by solving them recursively.
- Combine subproblem solutions into solution for original problem.

Closest Pair Problem – Divide and Conquer



How can we make the problem smaller and easier?

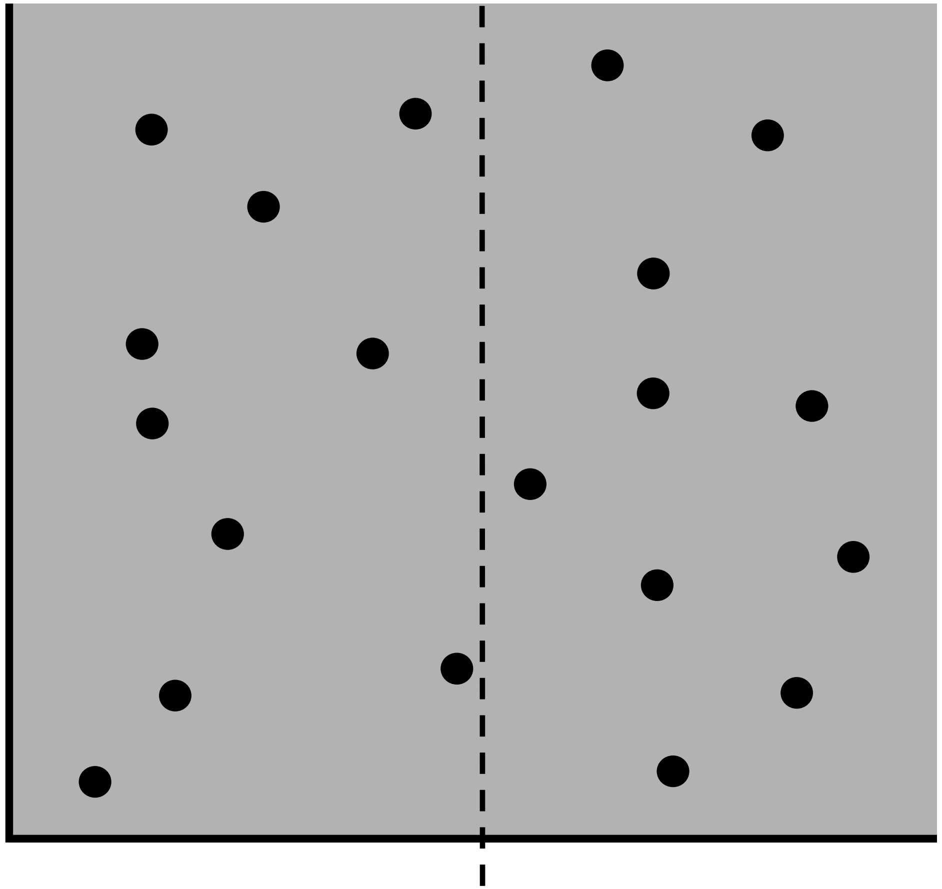
Closest Pair Problem – Divide and Conquer



How can we make the problem smaller and easier?

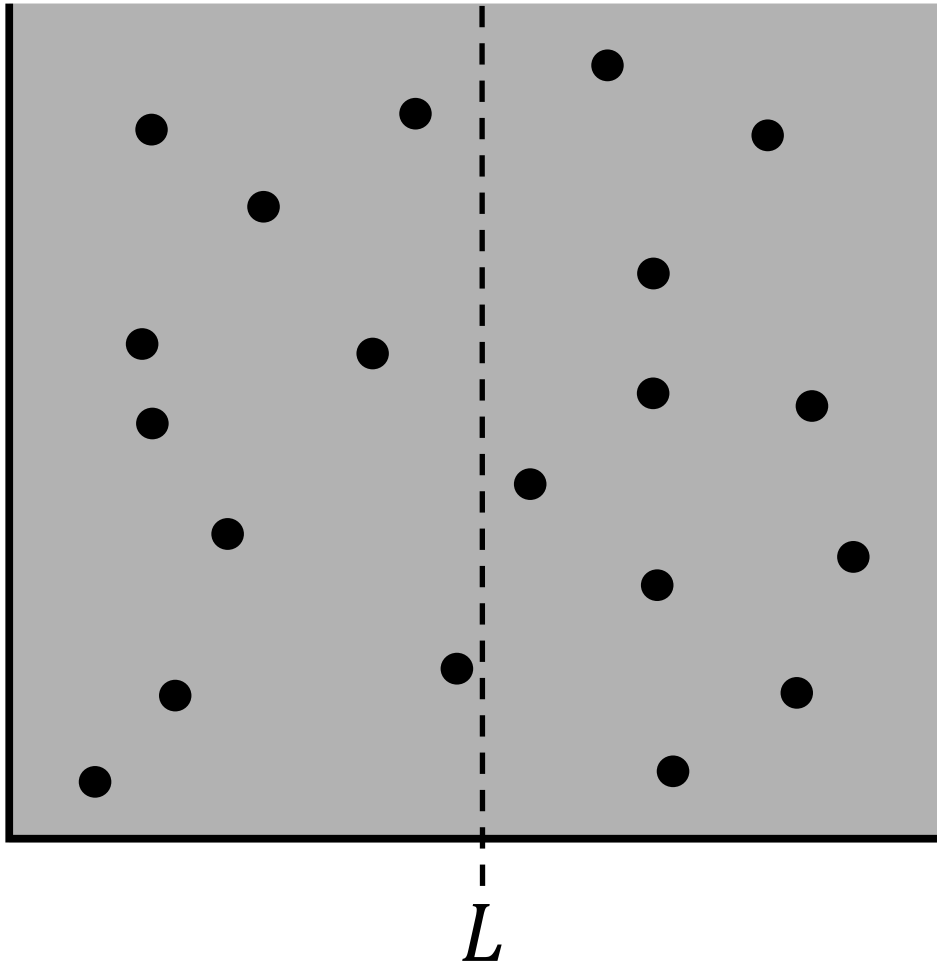
Split it up!

Closest Pair Problem – Divide and Conquer



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

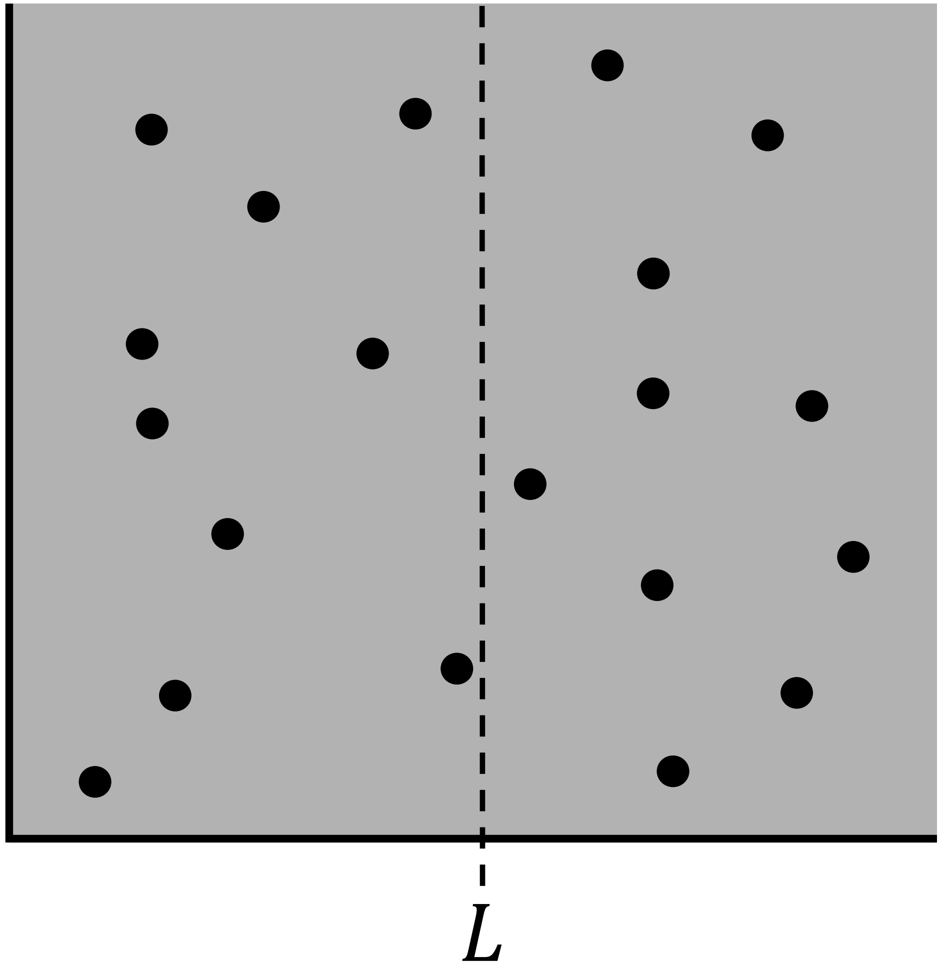
Closest Pair Problem – Divide and Conquer



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

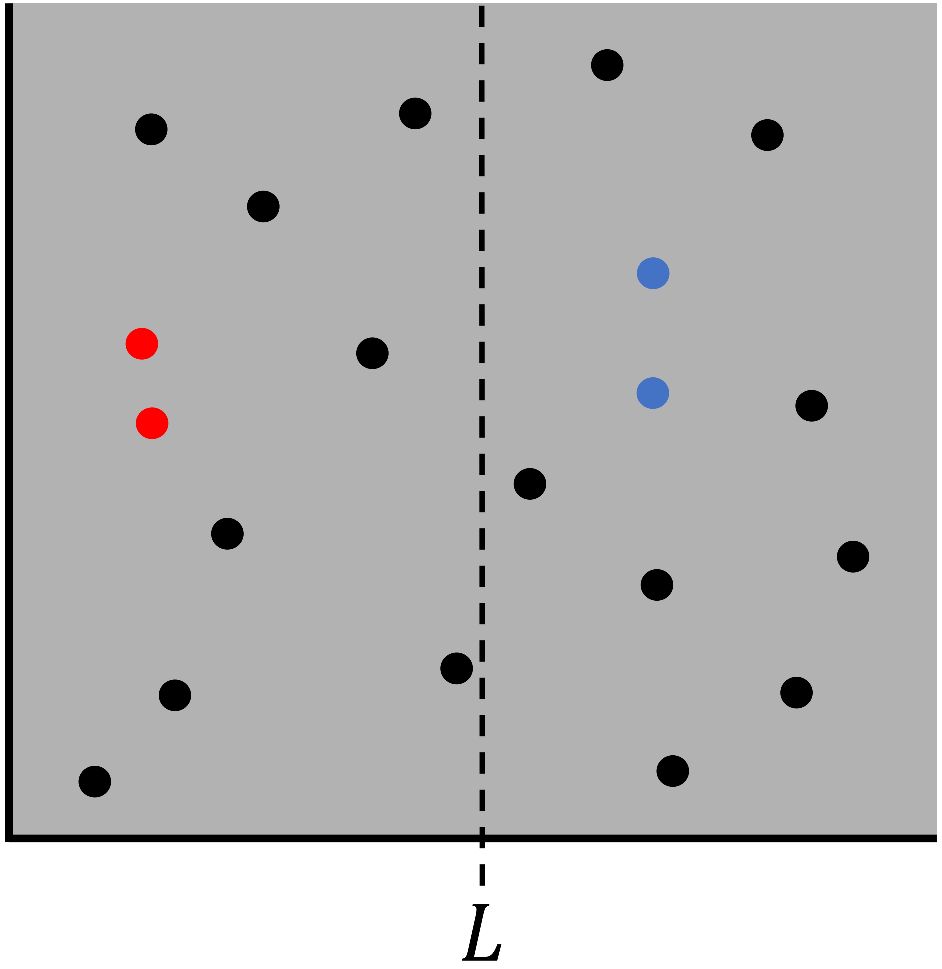


Conquer:

Recursively find closest pairs on each side[1].

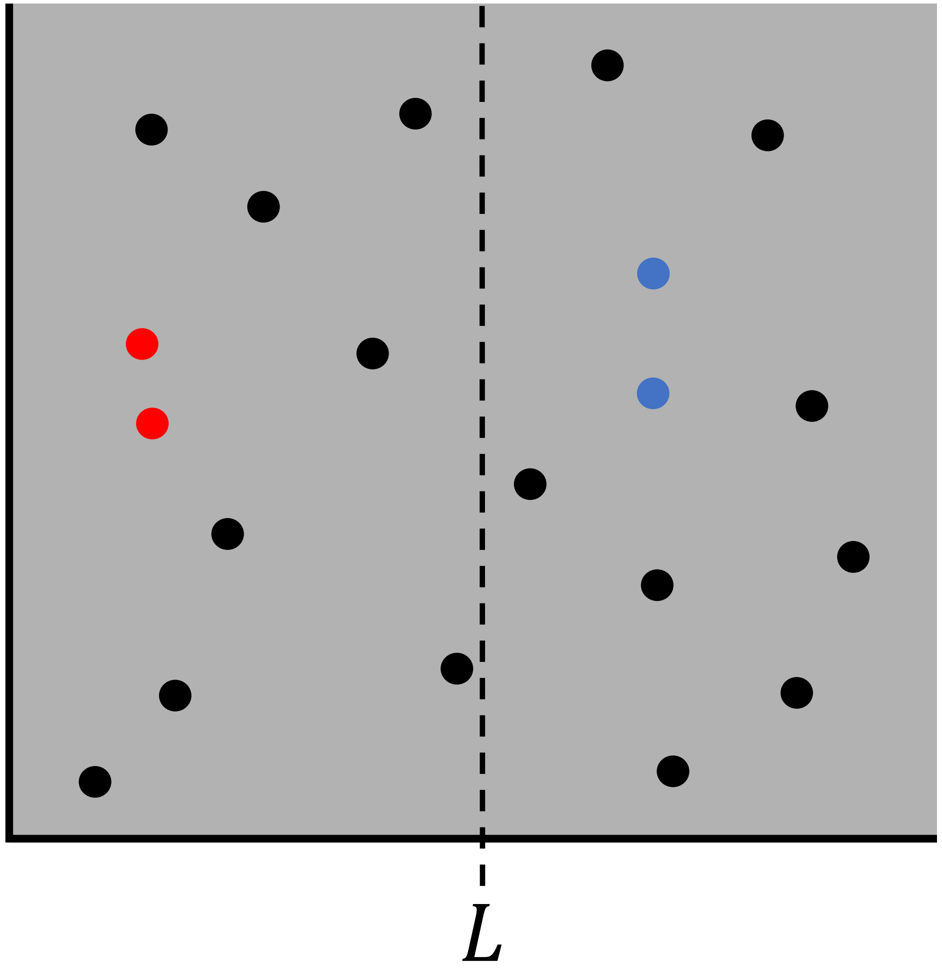
[1]Details to follow

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?

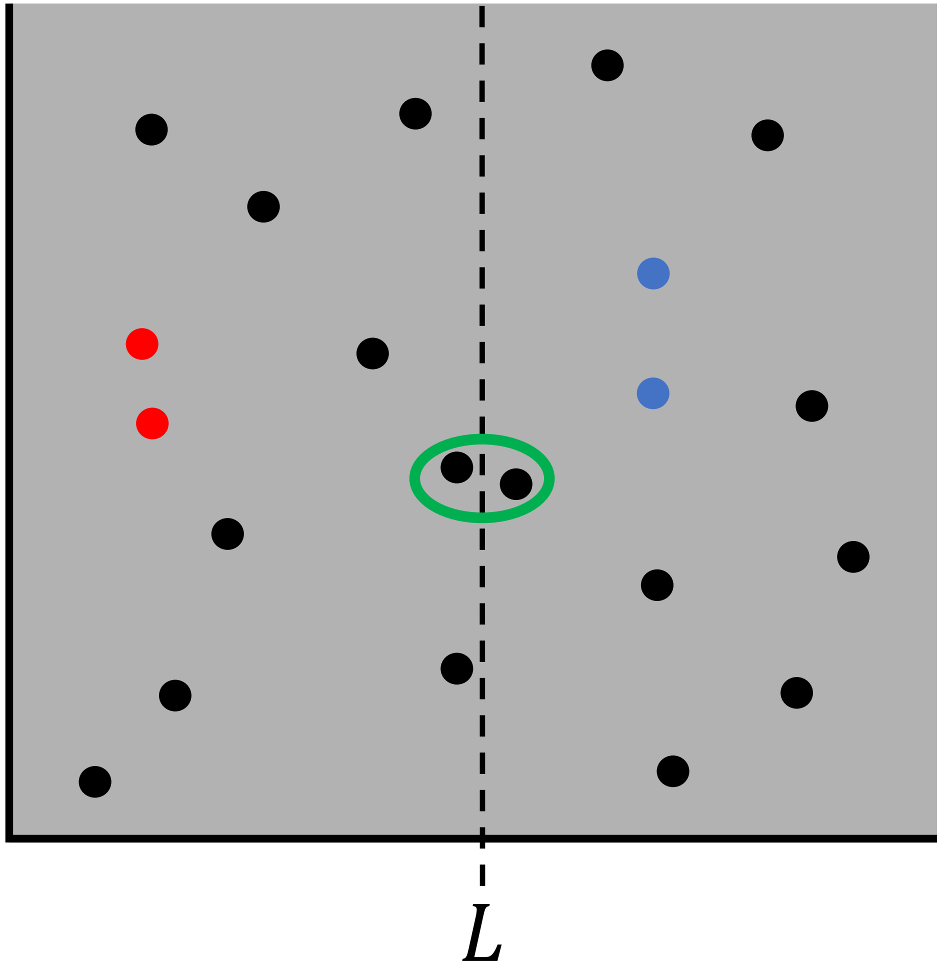
Closest Pair Problem – Divide and Conquer



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}.

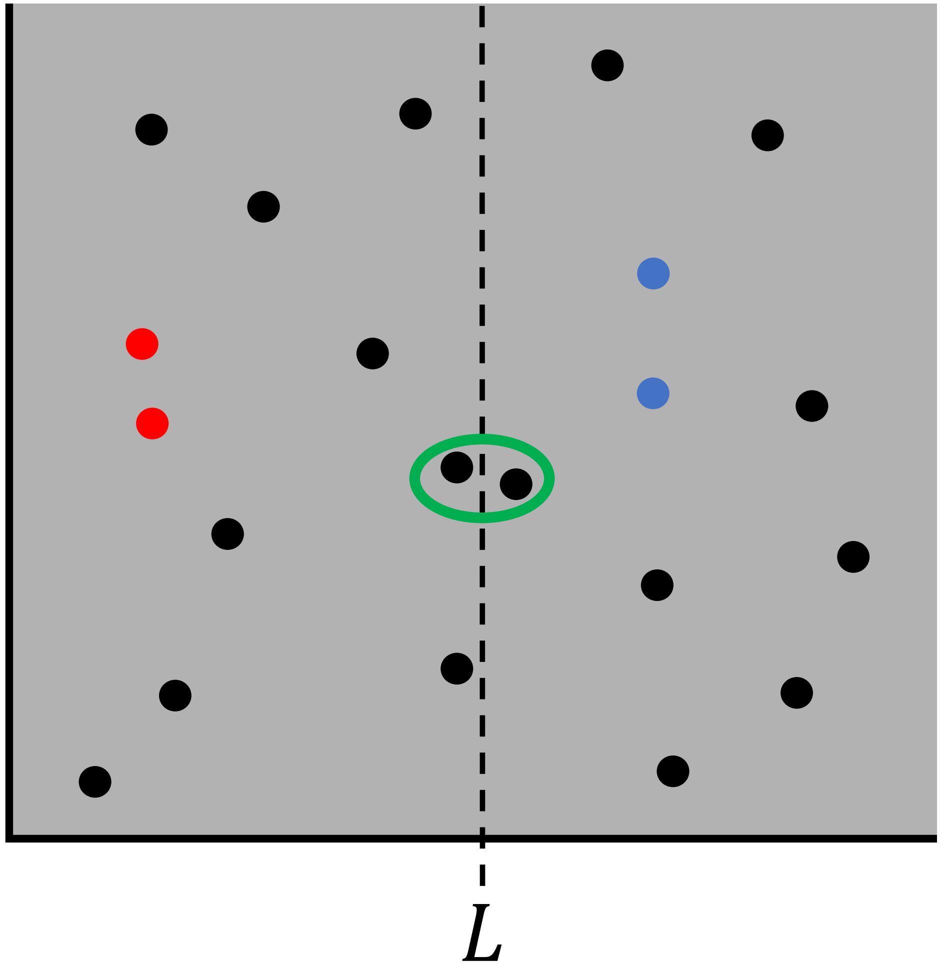
Closest Pair Problem – Divide and Conquer



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

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

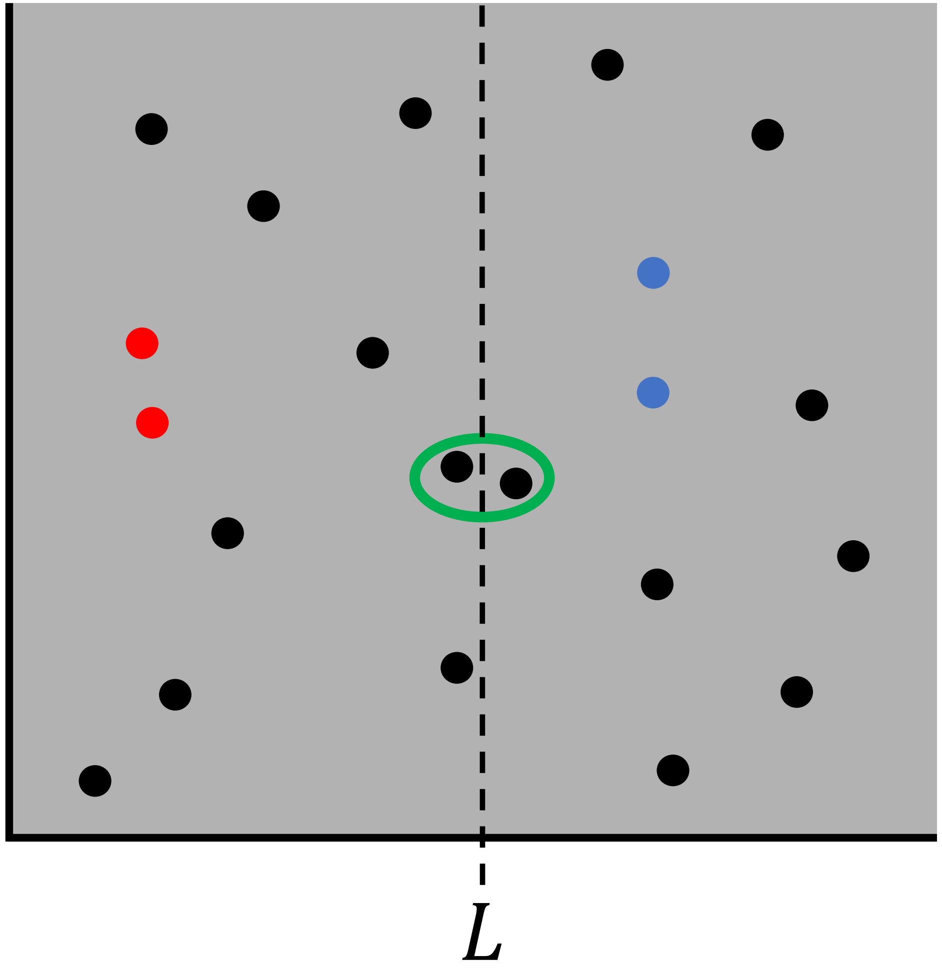
Closest Pair Problem – Divide and Conquer



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

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

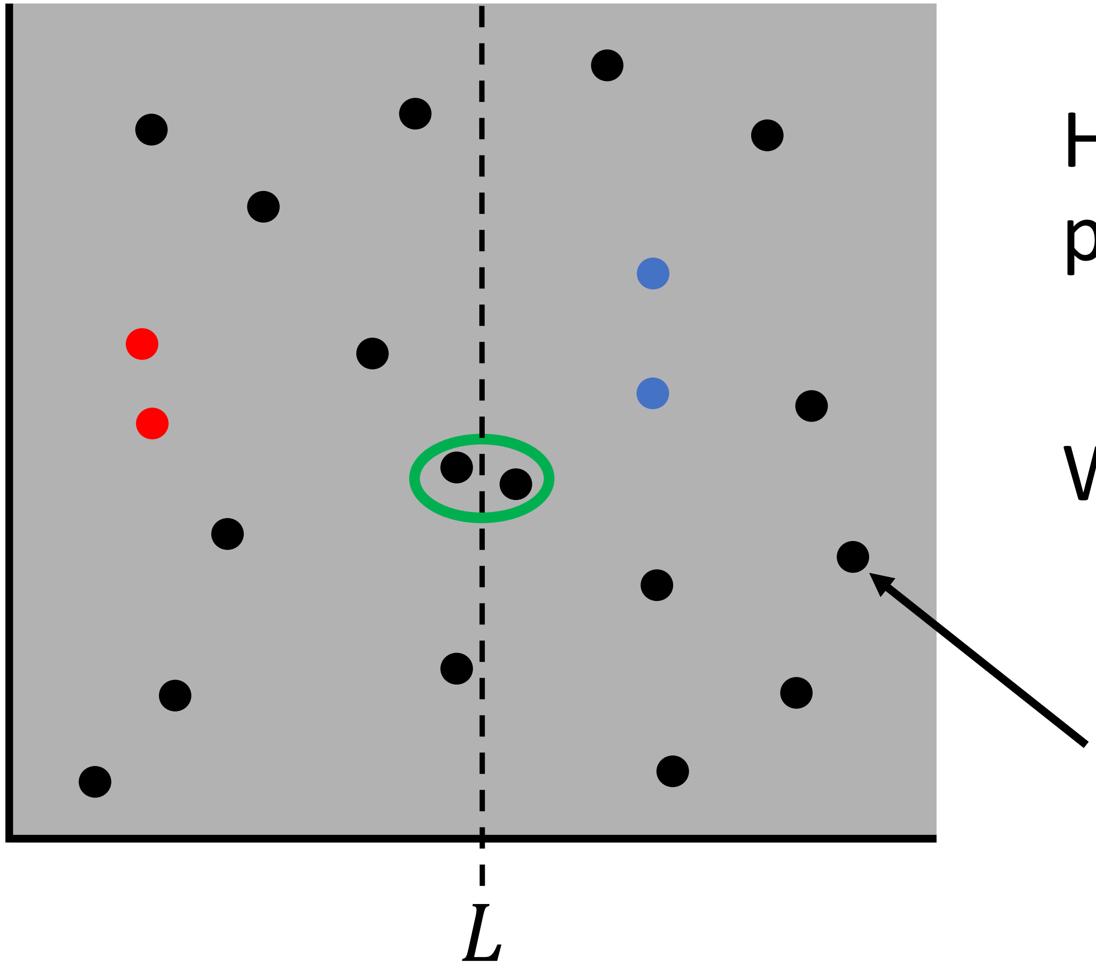
Closest Pair Problem – Divide and Conquer



How should we search for "straddle points"?

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

Closest Pair Problem – Divide and Conquer

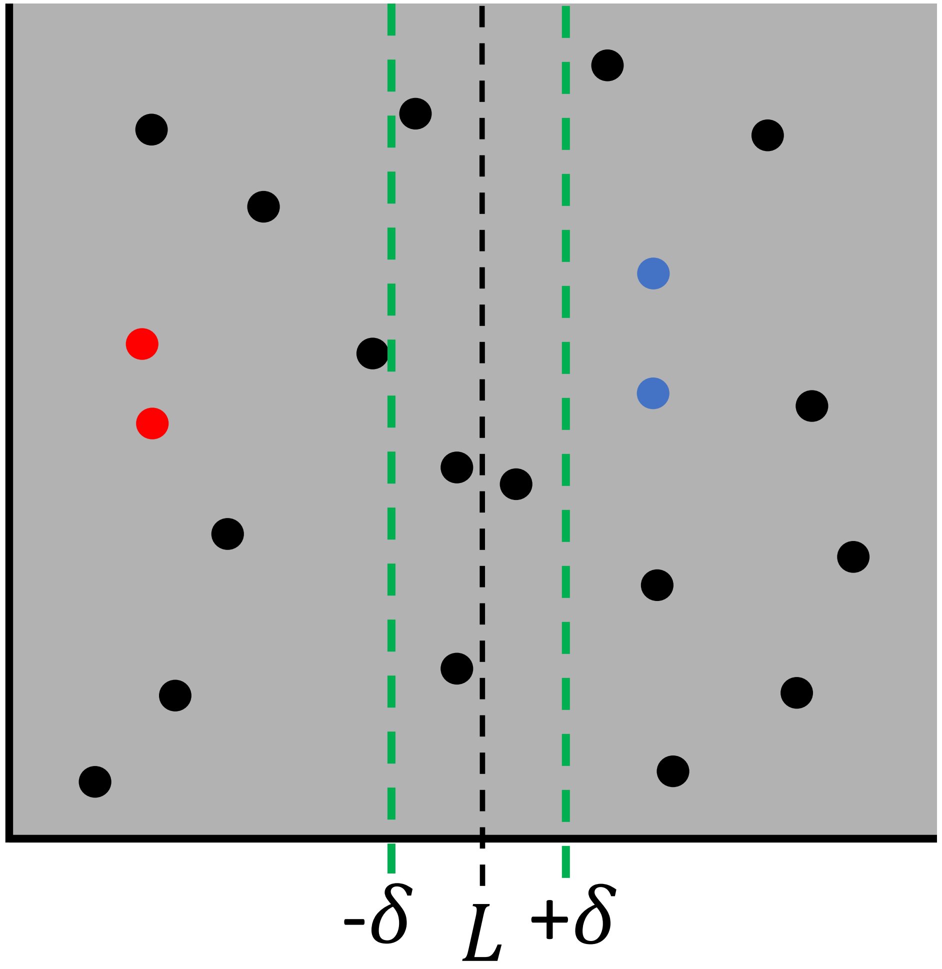


How should we search for "straddle points"?

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

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

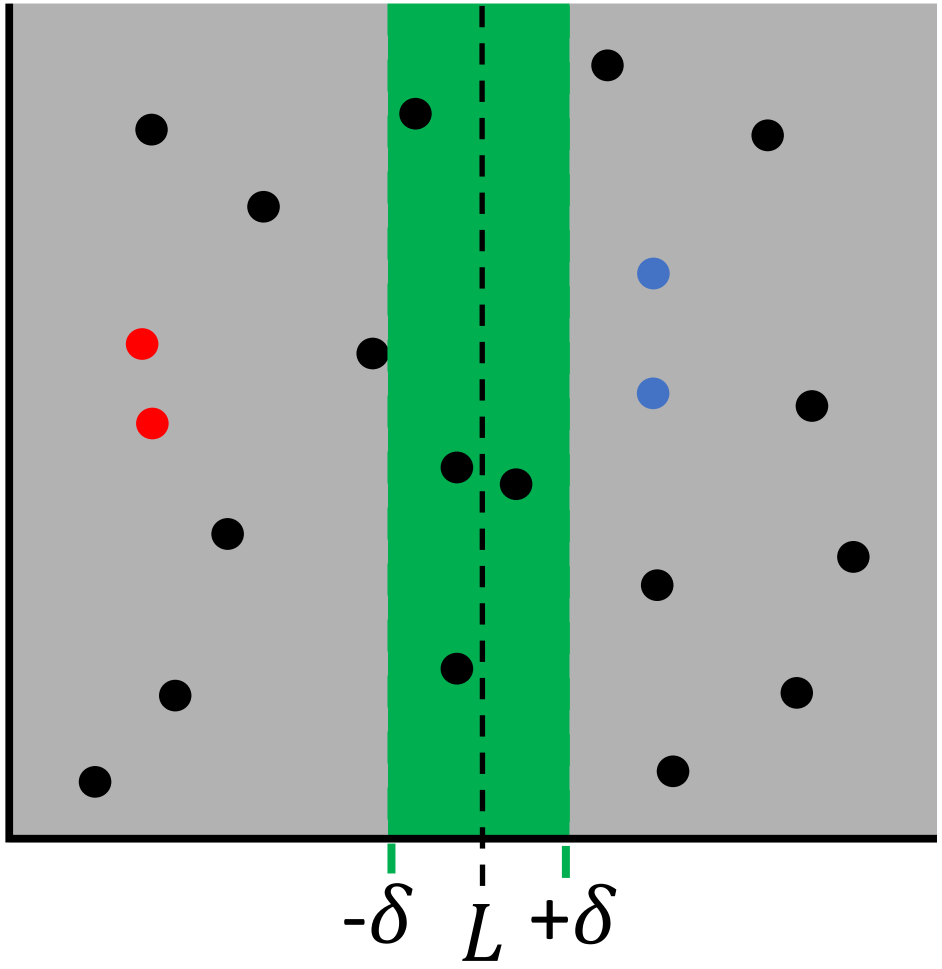
Closest Pair Problem – Divide and Conquer



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

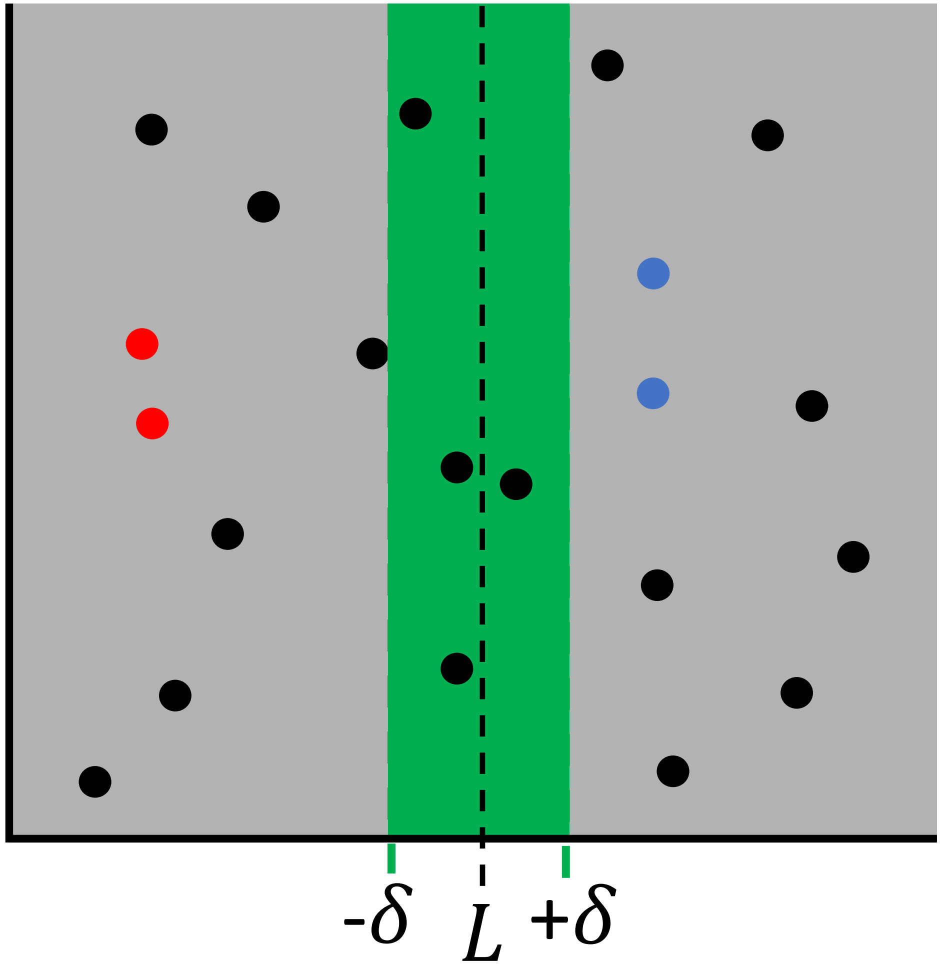


Rule: We only need to hunt for straddle points at most δ away from L.

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

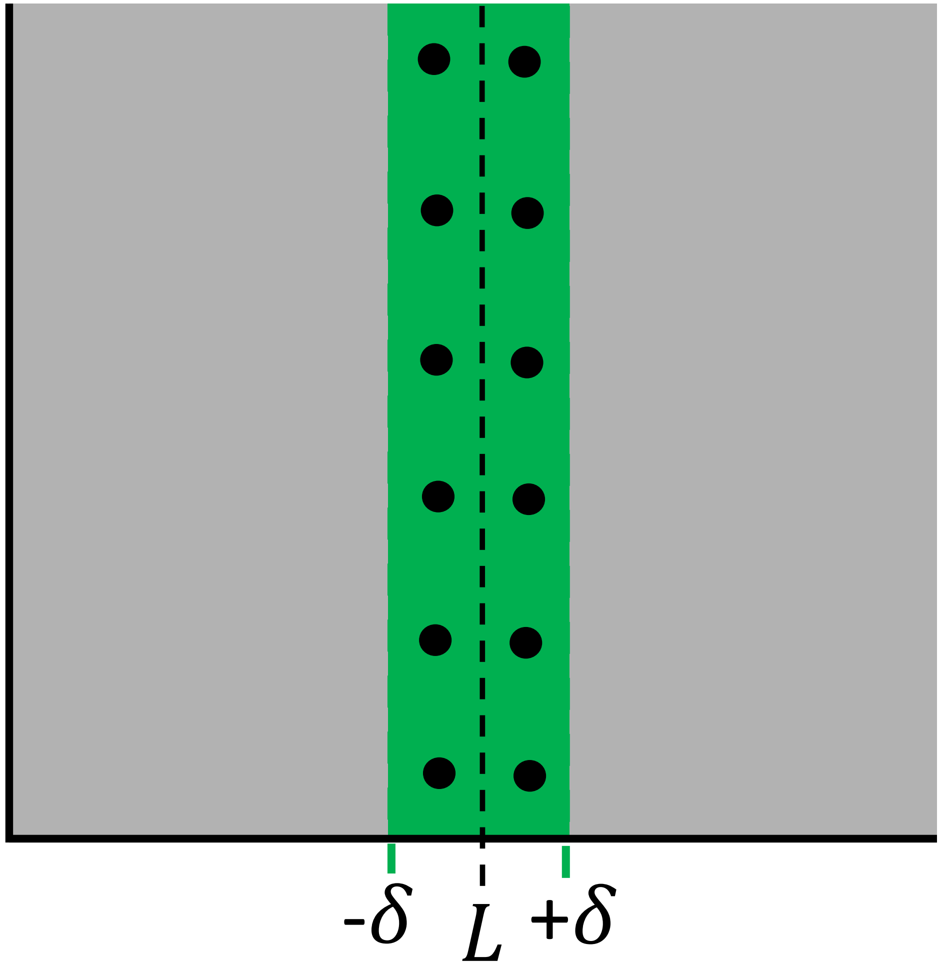
Let $\boldsymbol{S}$ be the set of straddle points.

Closest Pair Problem – Divide and Conquer



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

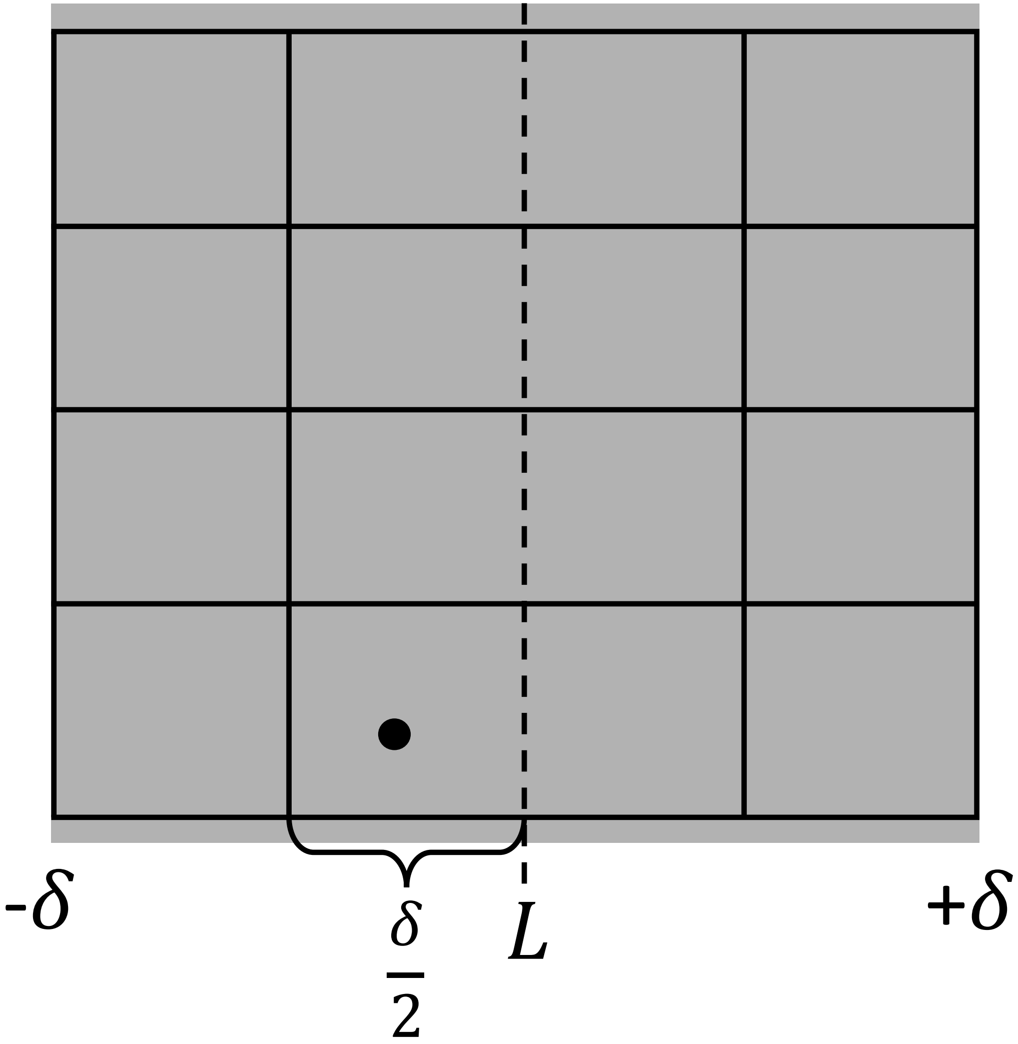
Closest Pair Problem – Divide and Conquer



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.

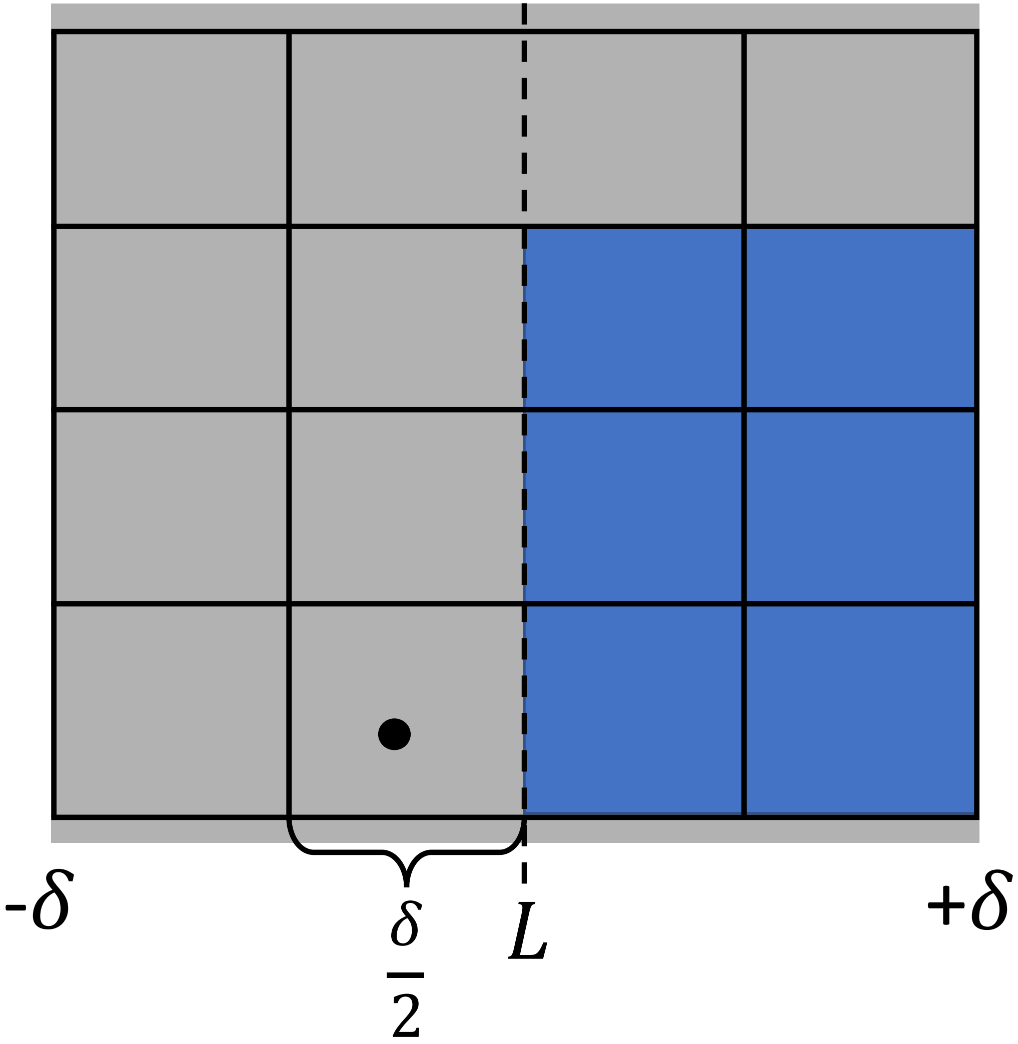
Closest Pair Problem – Divide and Conquer



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

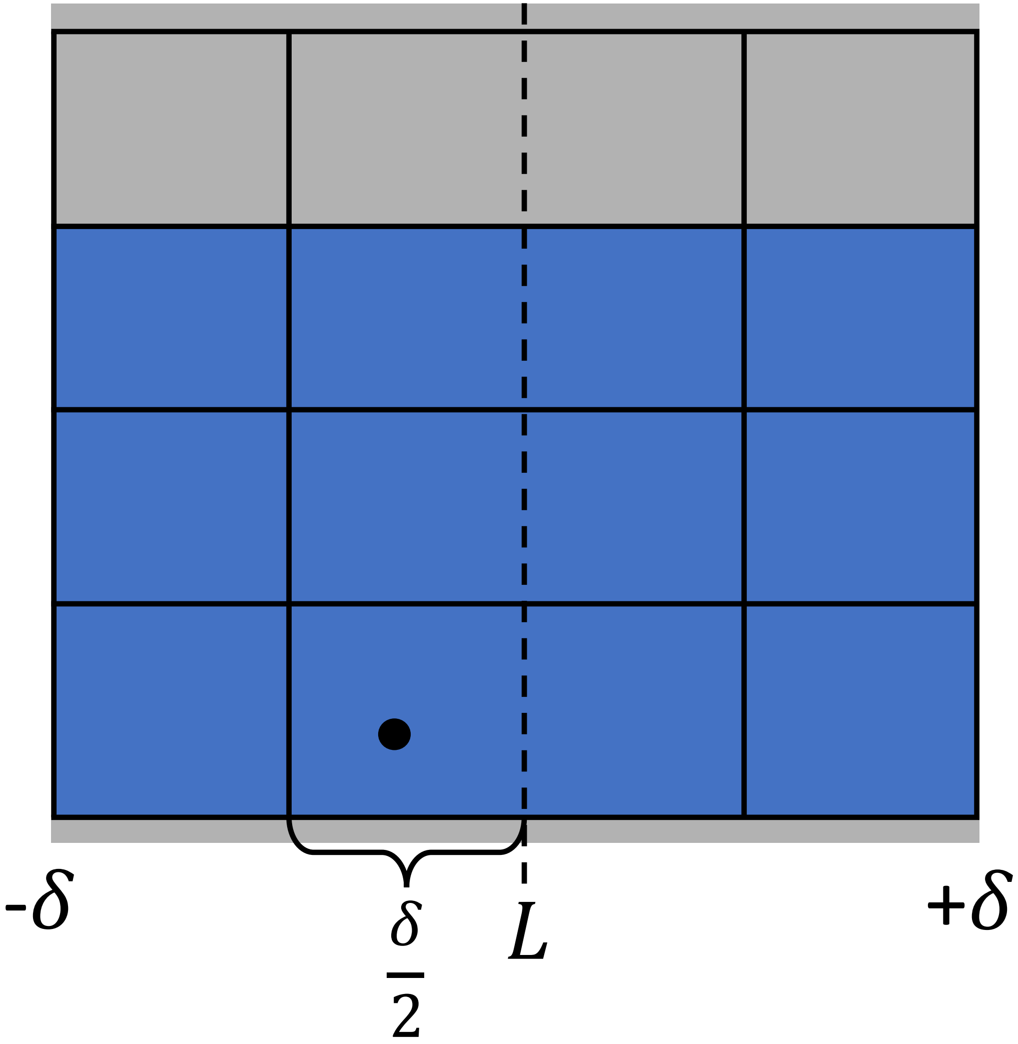


Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

Yes – we only care about points within δ.

Closest Pair Problem – Divide and Conquer



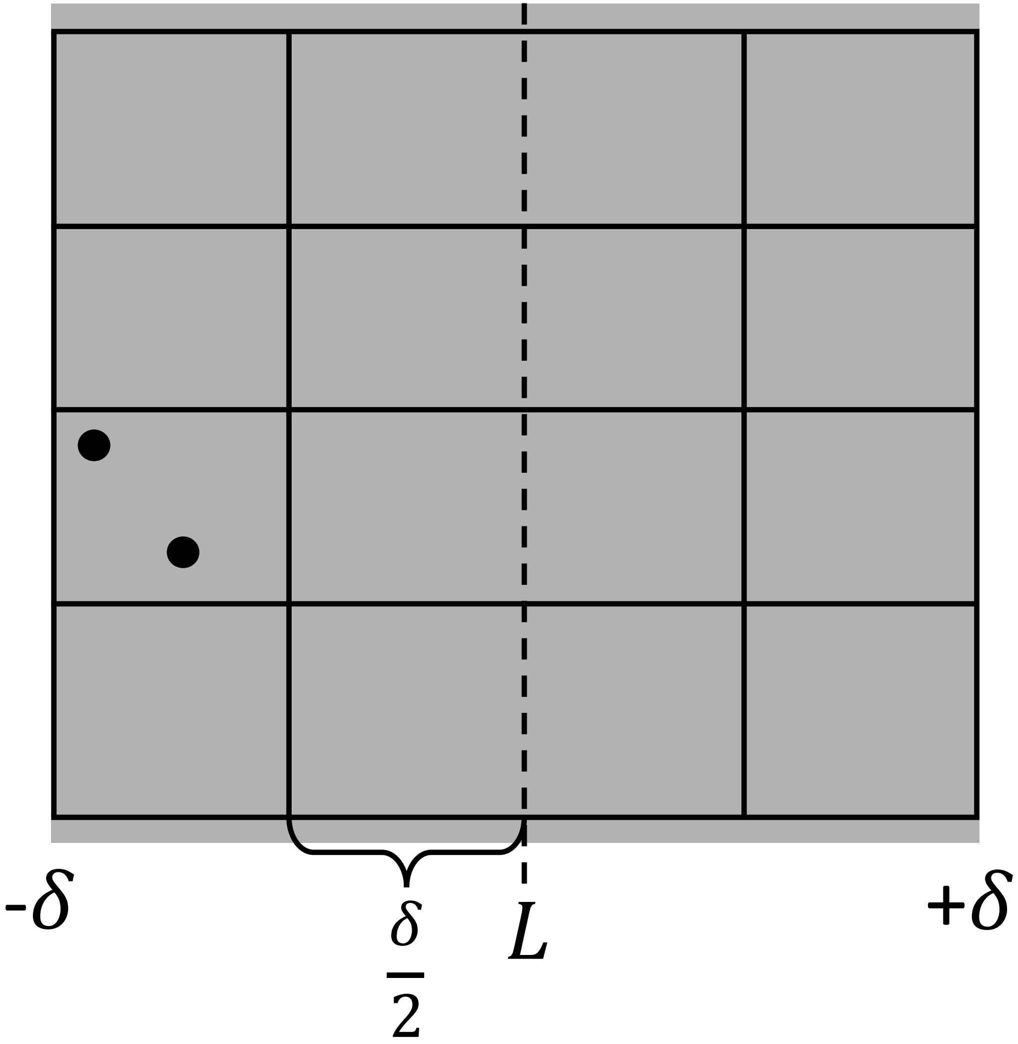
Divide S into $\frac{\delta}{2} \times \frac{\delta}{2}$ boxes.

Can we focus our search to certain boxes?

> Yes – we only care about points within δ.

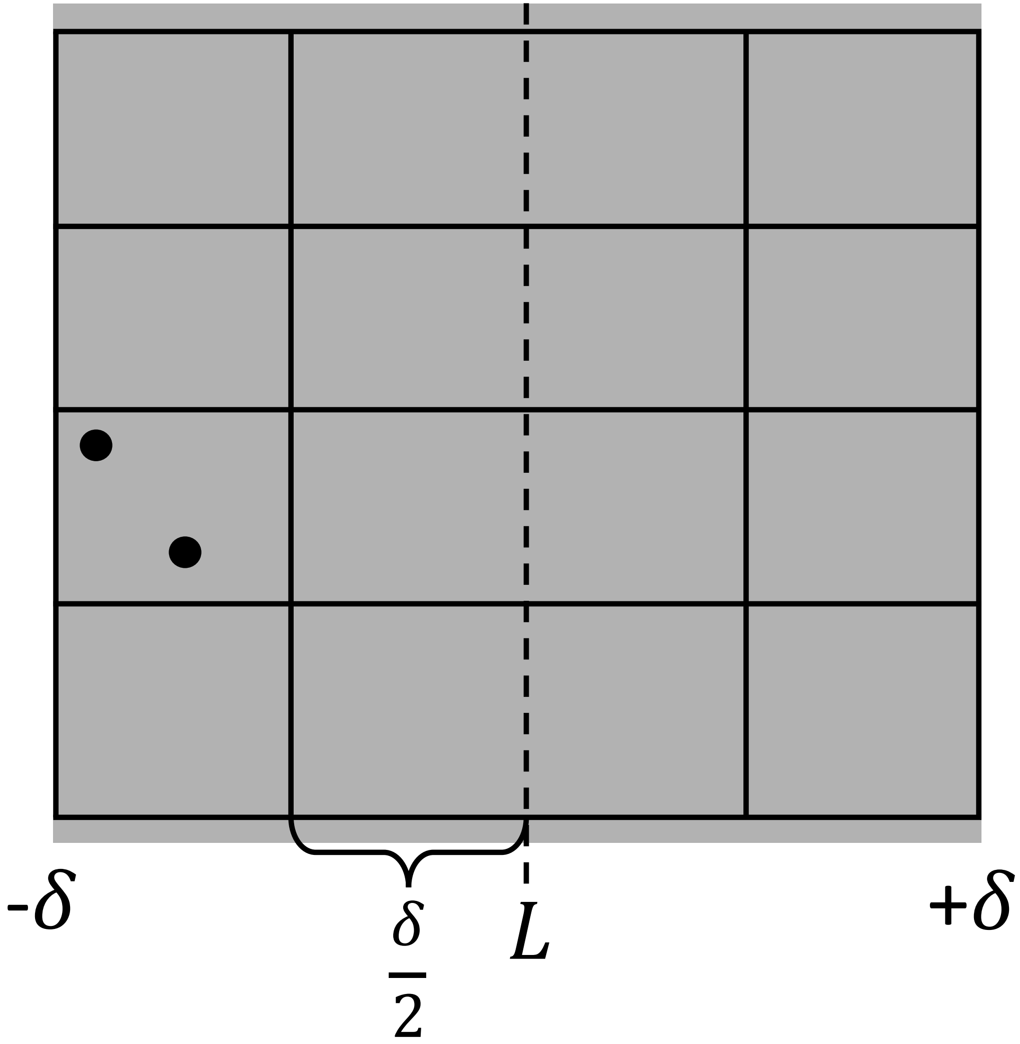
This still gives us possibly lots of points to look at.

Closest Pair Problem – Divide and Conquer



Can we have multiple points in one box?

Closest Pair Problem – Divide and Conquer

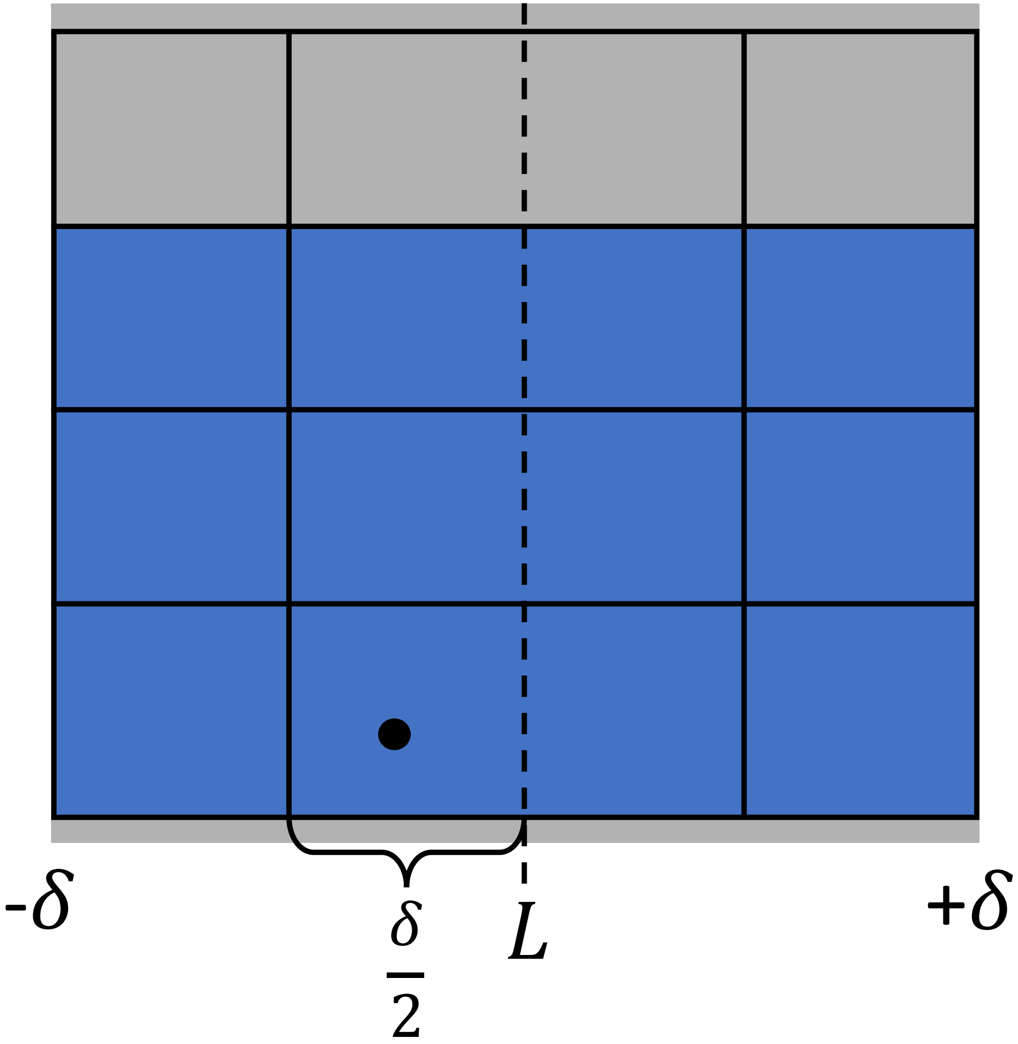


Can we have multiple points in one box?

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

⇒ at most one point per box.

Closest Pair Problem – Divide and Conquer

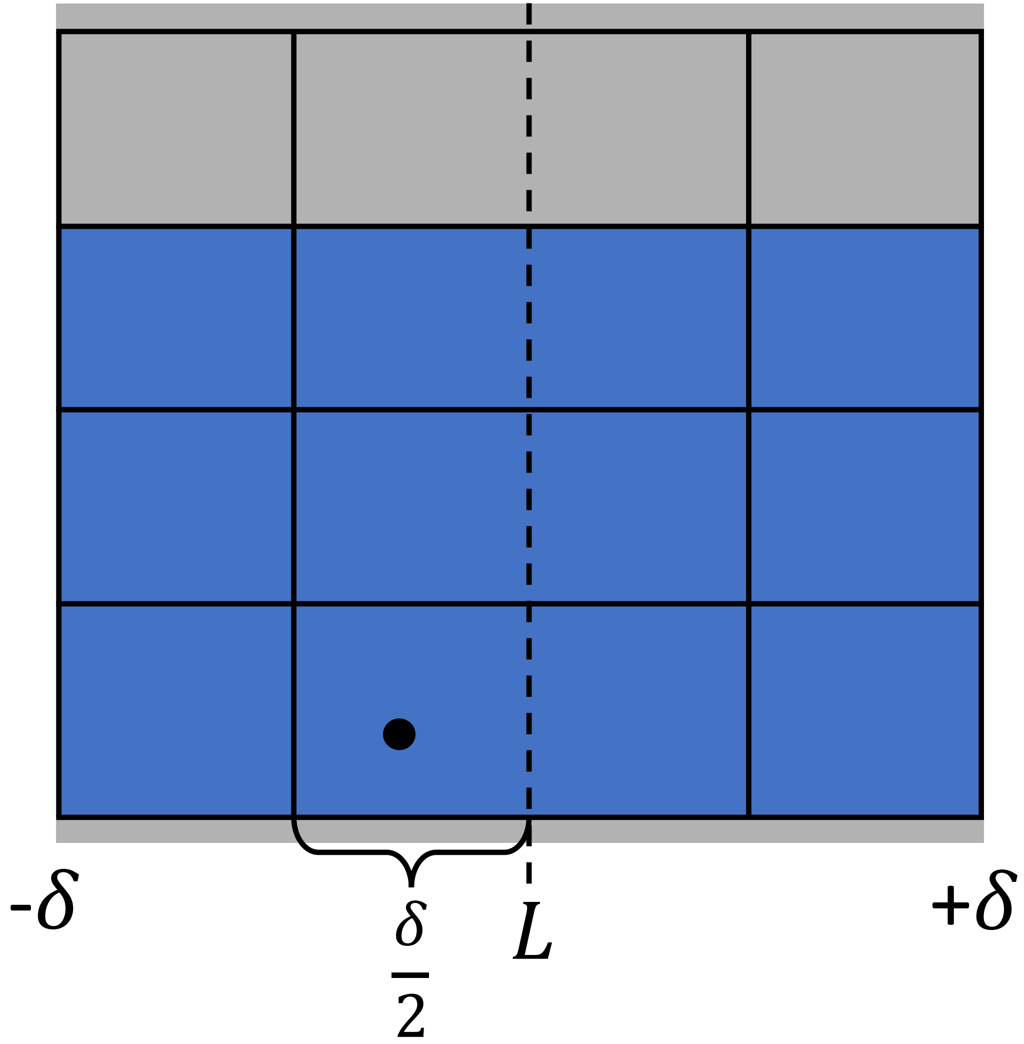


Only care about certain boxes
+ At most one point per box

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:

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