Advanced Algorithmic Techniques (COMP523)

Recursion and Divide and Conquer Techniques #3

Recap and plan

Recap and plan

- **Last lecture:**
  - Sorting with the **MergeSort** algorithm.
  - Sorting with the **QuickSort** algorithm.
  - The limitations of comparison-based sorting.

Recap and plan

- **Last lecture:**
  - Sorting with the **MergeSort** algorithm.
  - Sorting with the **QuickSort** algorithm.
  - The limitations of comparison-based sorting.
- **This lecture:**
  - Finding the closest pair of points.
  - Integer Multiplication.

Quick Recap

- **Searching:**
  - LinearSearch: Time O(n), (Aux.) Memory O(1)
  - BinarySearch: Time O(log n), (Aux.) Memory O(log n)
- **Sorting:**
  - InsertionSort: Time $O(n^2)$, (Aux.) Memory O(1)
  - MergeSort: Time O(n) log n, (Aux.) Memory **?**
  - QuickSort: Time $O(n^2)$, (Aux.) Memory **?**
- **Majority:**
  - General array: Time O(n), (Aux.) Memory **?**
  - Sorted array: Time O(log n), (Aux.) Memory **?**

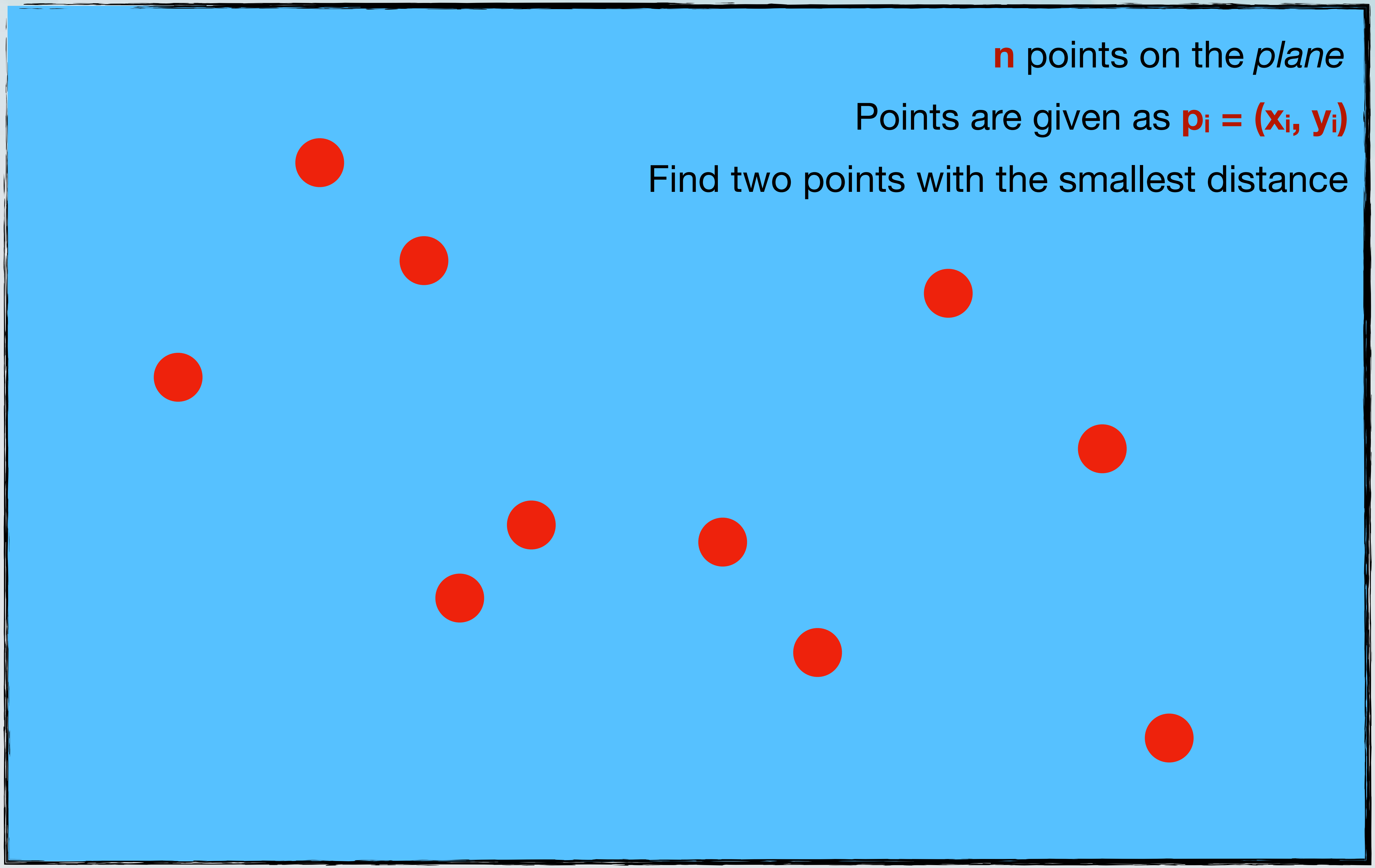
Finding the closest pair of points

Finding the closest pair of points

n points on the *plane*

Points are given as $p_i = (x_i, y_i)$

Find two points with the smallest distance

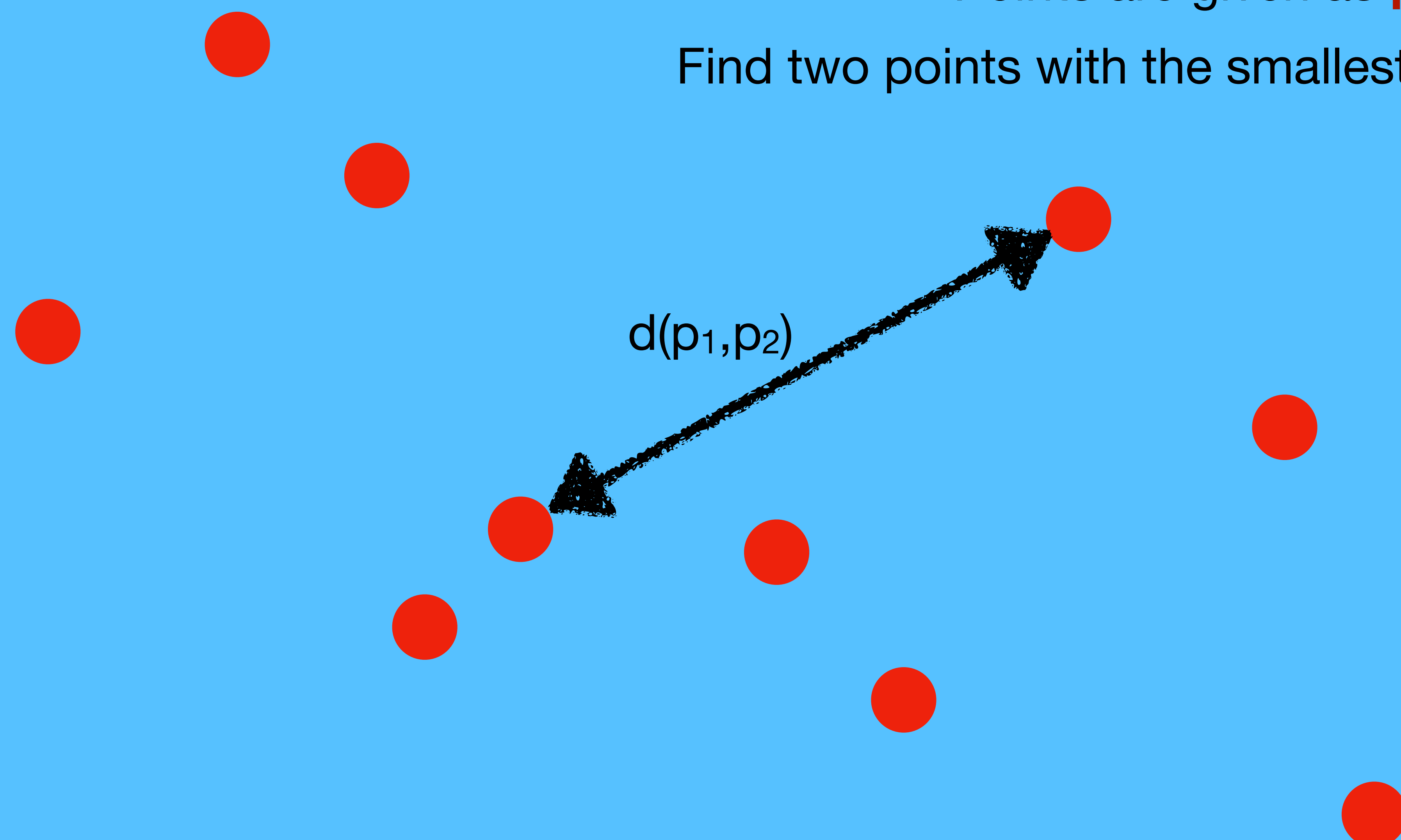


Finding the closest pair of points

n points on the *plane*

Points are given as $\mathbf{p_i = (x_i, y_i)}$

Find two points with the smallest distance



Naive solution

Naive solution

- For every point, find the distance to each other point.

Naive solution

- For every point, find the distance to each other point.
- Output two points that have the smallest distance.

Naive solution

- For every point, find the distance to each other point.
- Output two points that have the smallest distance.
- Running time?

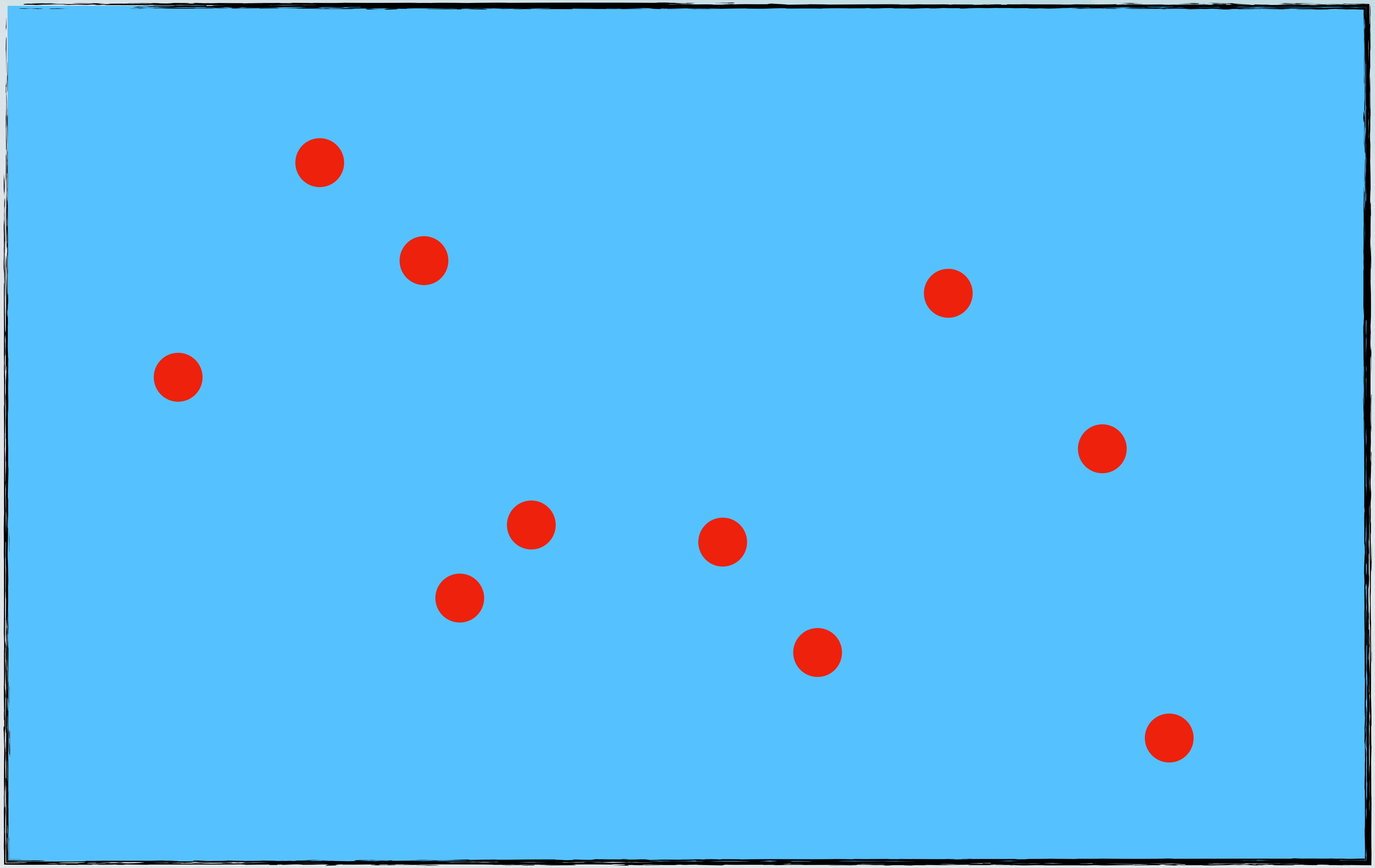
Naive solution

- For every point, find the distance to each other point.
- Output two points that have the smallest distance.
- Running time?
  - **$\Omega(n^2)$**

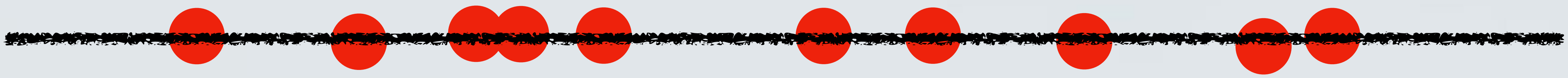
Naive solution

- For every point, find the distance to each other point.
- Output two points that have the smallest distance.
- Running time?
  - **$\Omega(n^2)$**
- Can we do better?

Warmup: Points on the line

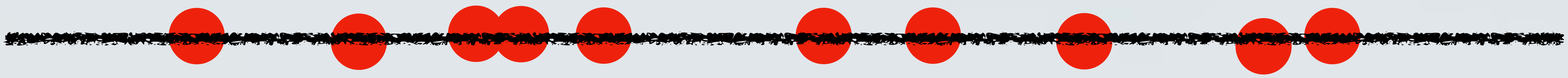


Warmup: Points on the line



Warmup: Points on the line

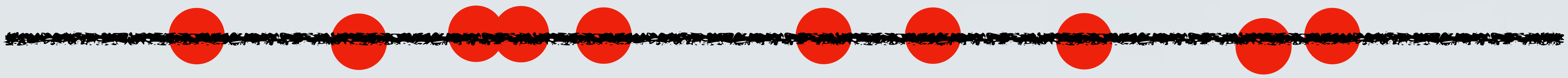
Points are now $x_1, x_2, \ldots, x_n$



Warmup: Points on the line

Points are now $x_1, x_2, \ldots, x_n$

Sort the points $x_1, x_2, \ldots, x_n$

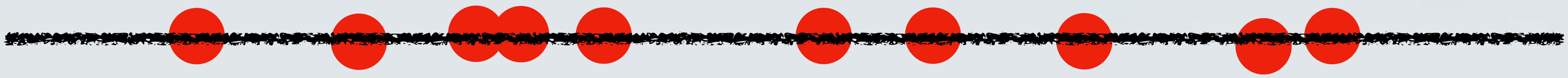


Warmup: Points on the line

Points are now $x_1, x_2, \ldots, x_n$

Sort the points $x_1, x_2, \ldots, x_n$

Consider only distances **between consecutive points**

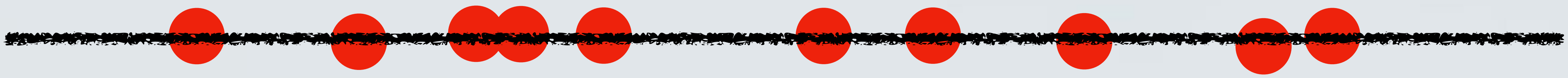


Warmup: Points on the line

Points are now $x_1, x_2, \ldots, x_n$

Sort the points $x_1, x_2, \ldots, x_n$

Consider only distances **between consecutive points**



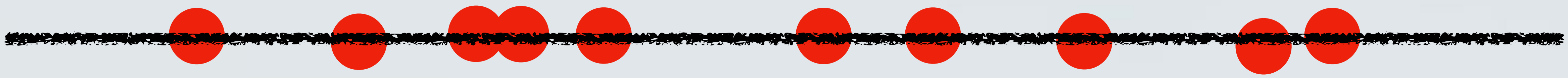
What is the worst-case running time?

Warmup: Points on the line

Points are now $x_1, x_2, \ldots, x_n$

Sort the points $x_1, x_2, \ldots, x_n$

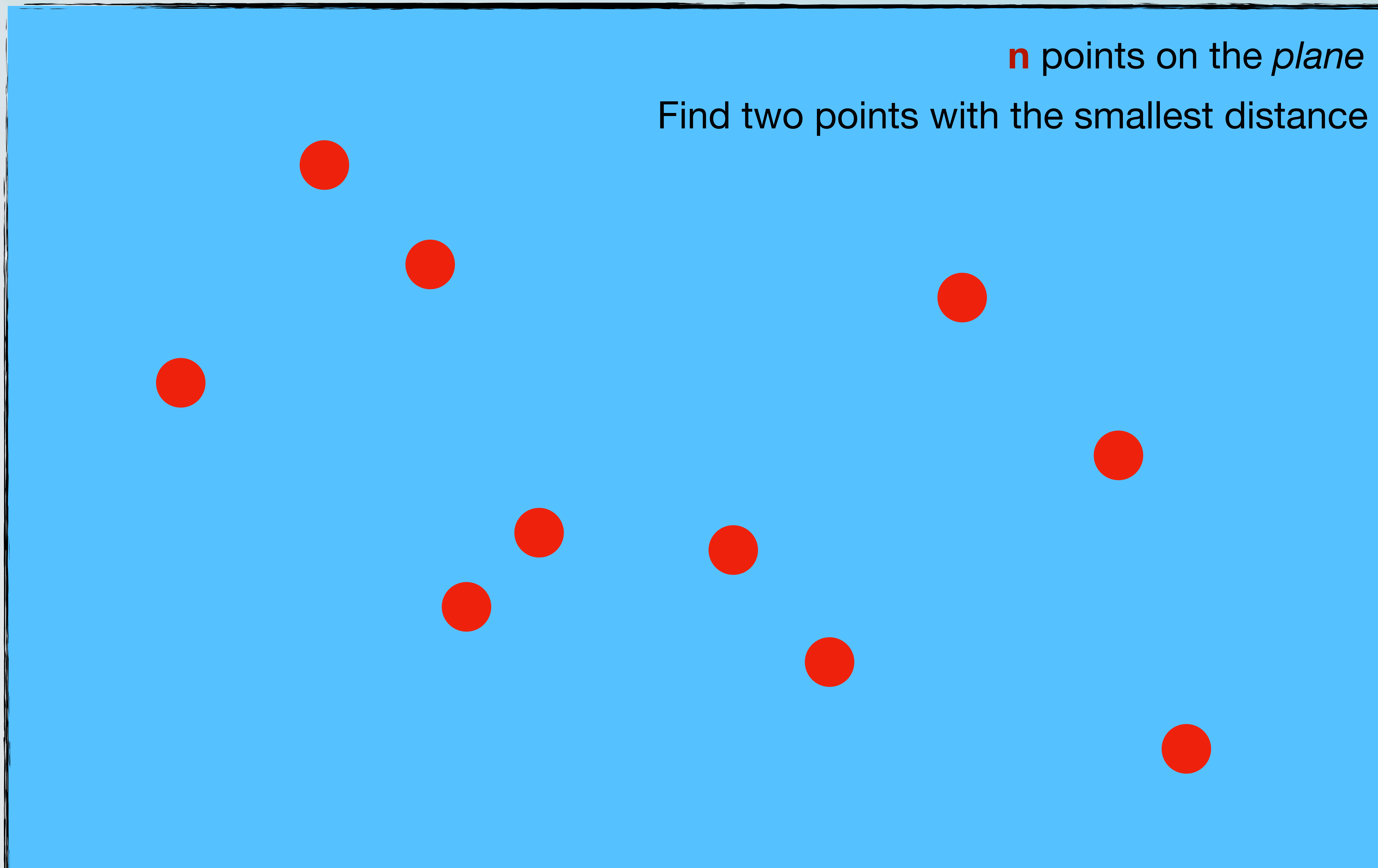
Consider only distances **between consecutive points**



What is the worst-case running time?

Can be done in **O(n log n)**

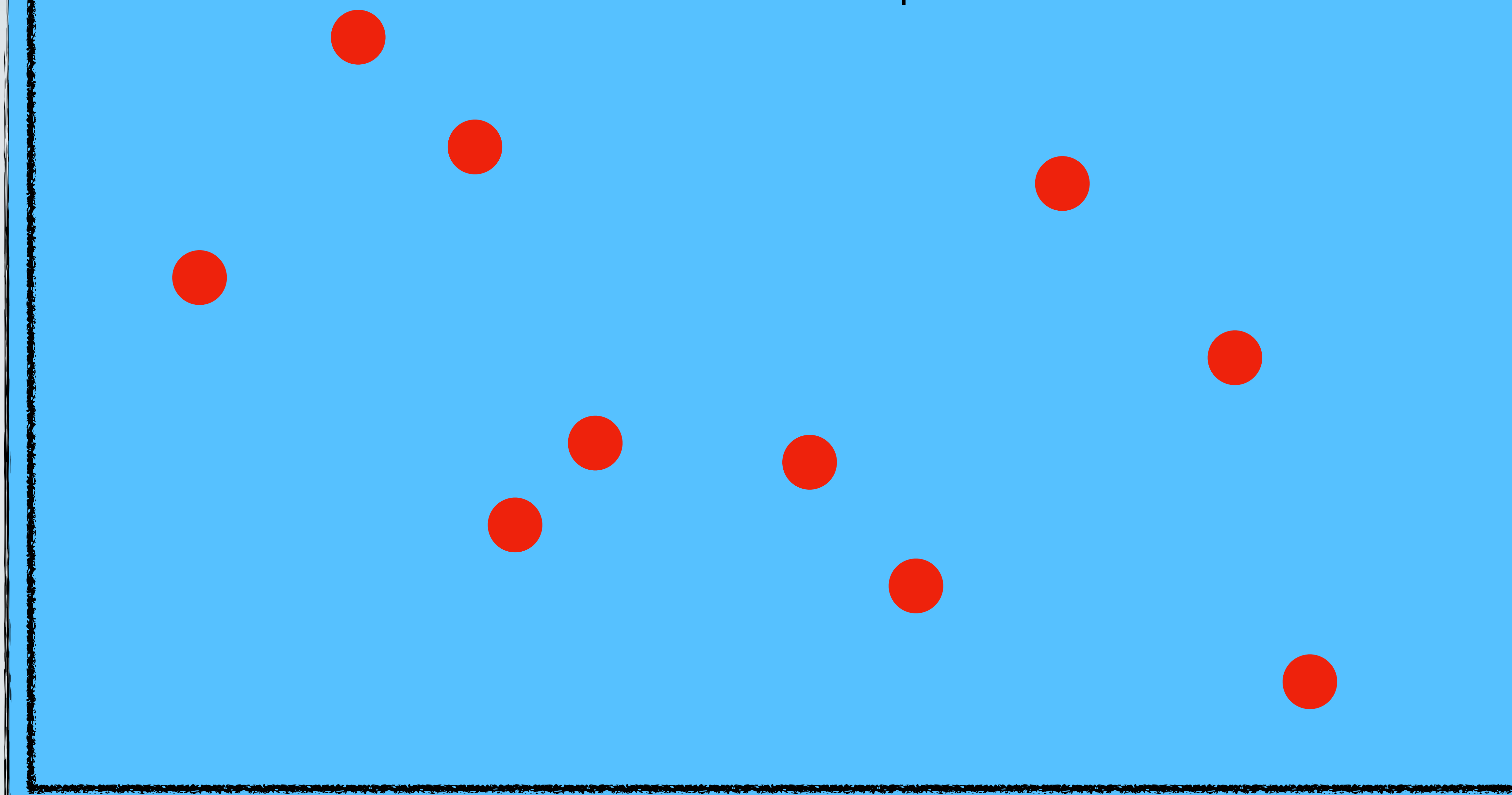
Finding the closest pair of points



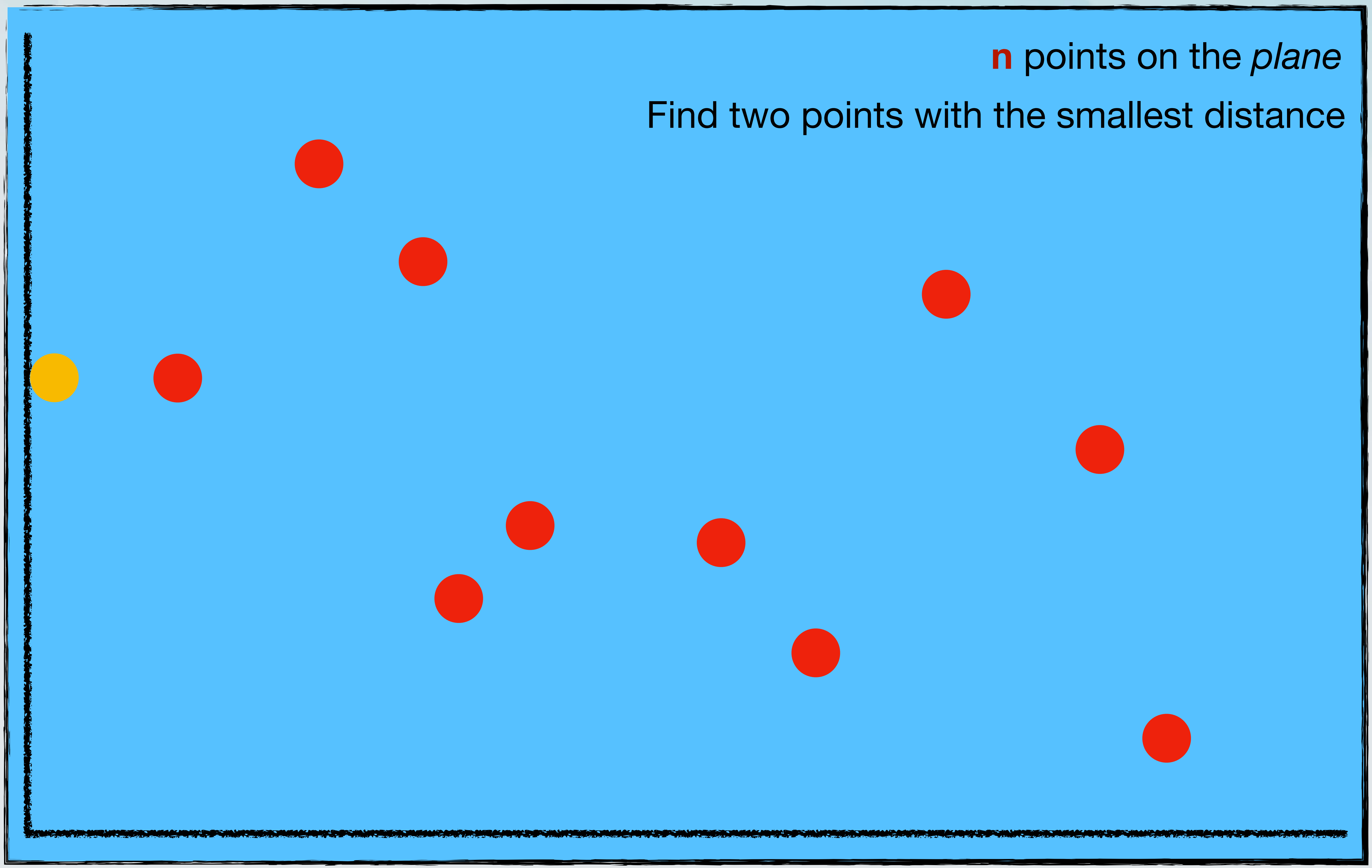
Finding the closest pair of points

n points on the *plane*

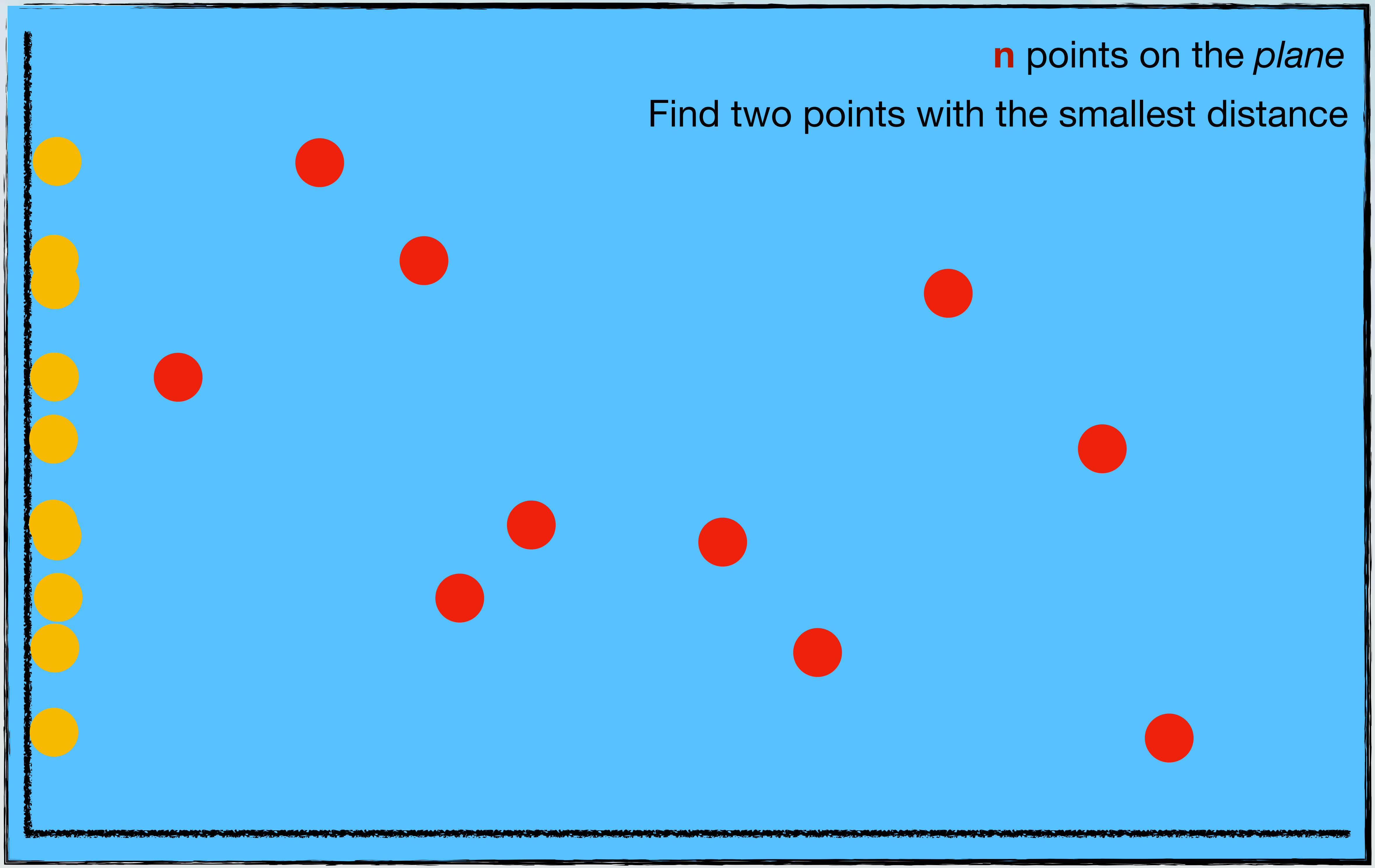
Find two points with the smallest distance



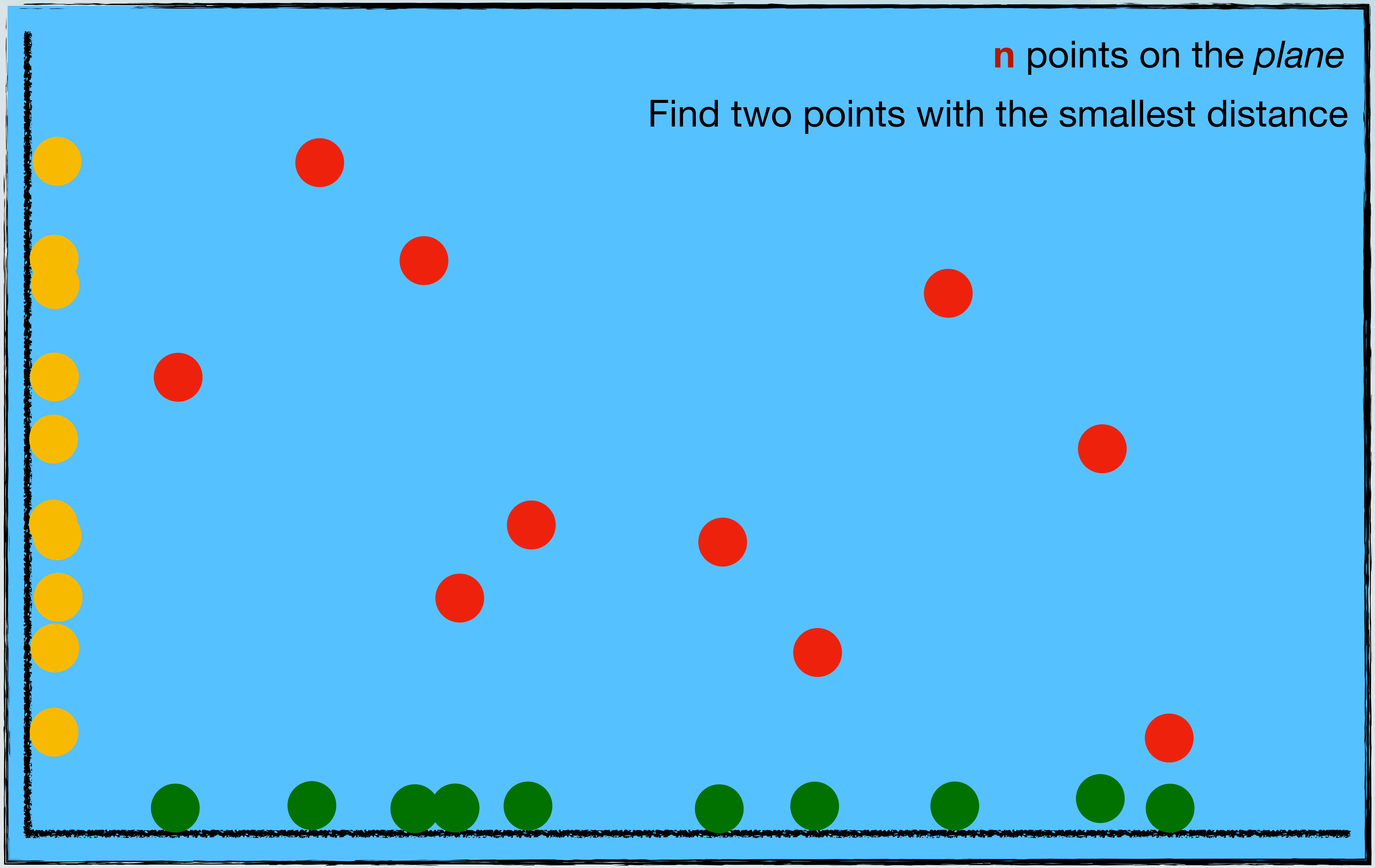
Finding the closest pair of points



Finding the closest pair of points



Finding the closest pair of points



Finding the closest pair of points

Finding the closest pair of points

Preprocessing:

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.
- To populate **H** and **V**, simply run through the given points $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, … , $p_n=(x_n,y_n)$ and put $x_1, x_2, \ldots, x_n$ into **H** and $y_1, y_2, \ldots, y_n$ into **V**.

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.
- To populate **H** and **V**, simply run through the given points $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, … , $p_n=(x_n,y_n)$ and put $x_1, x_2, \dots, x_n$ into **H** and $y_1, y_2, \dots, y_n$ into **V**.
- Sort the points in **H** and in **V**, using some sorting algorithm.

Finding the closest pair of points

Finding the closest pair of points

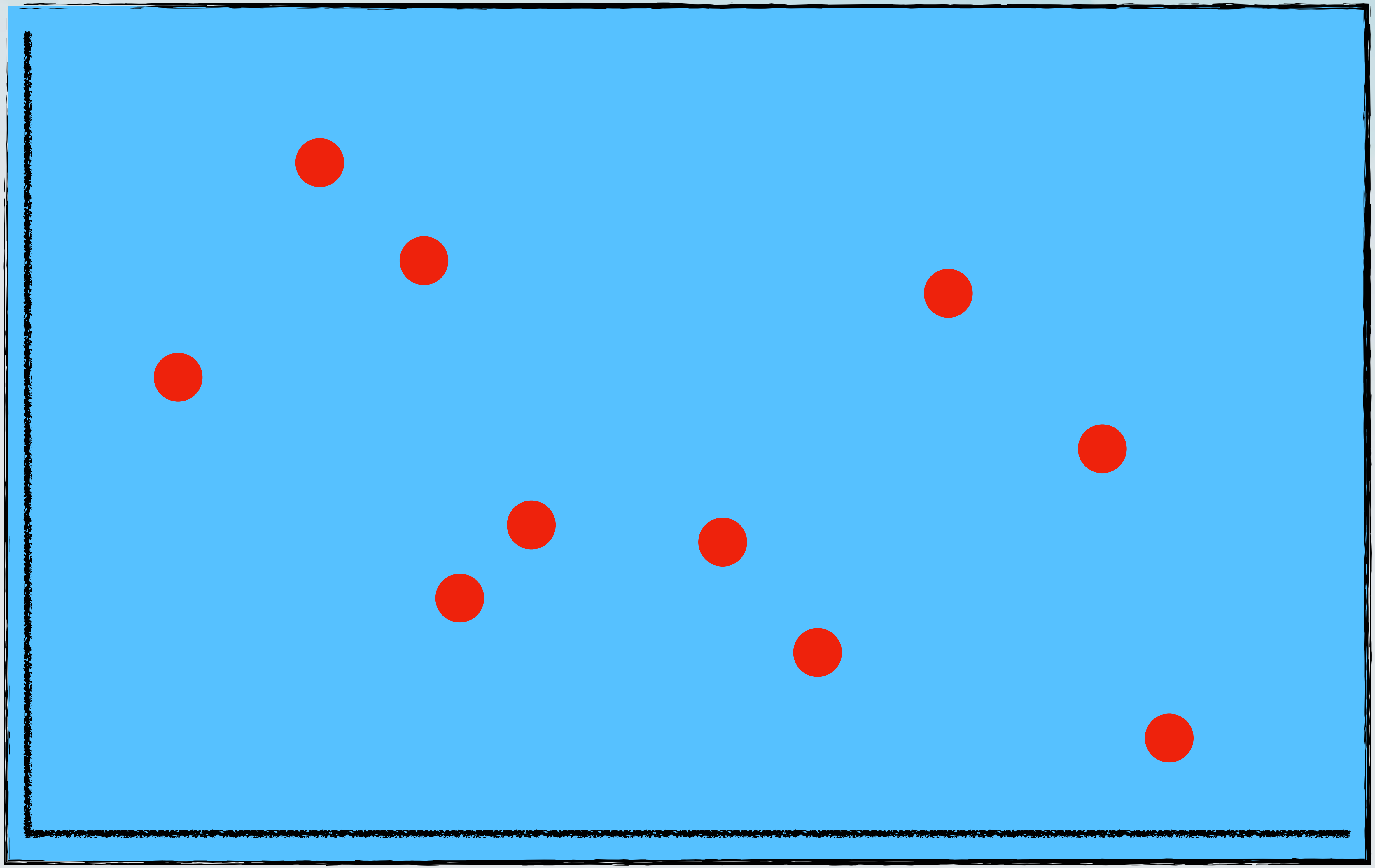
Main algorithm:

Finding the closest pair of points

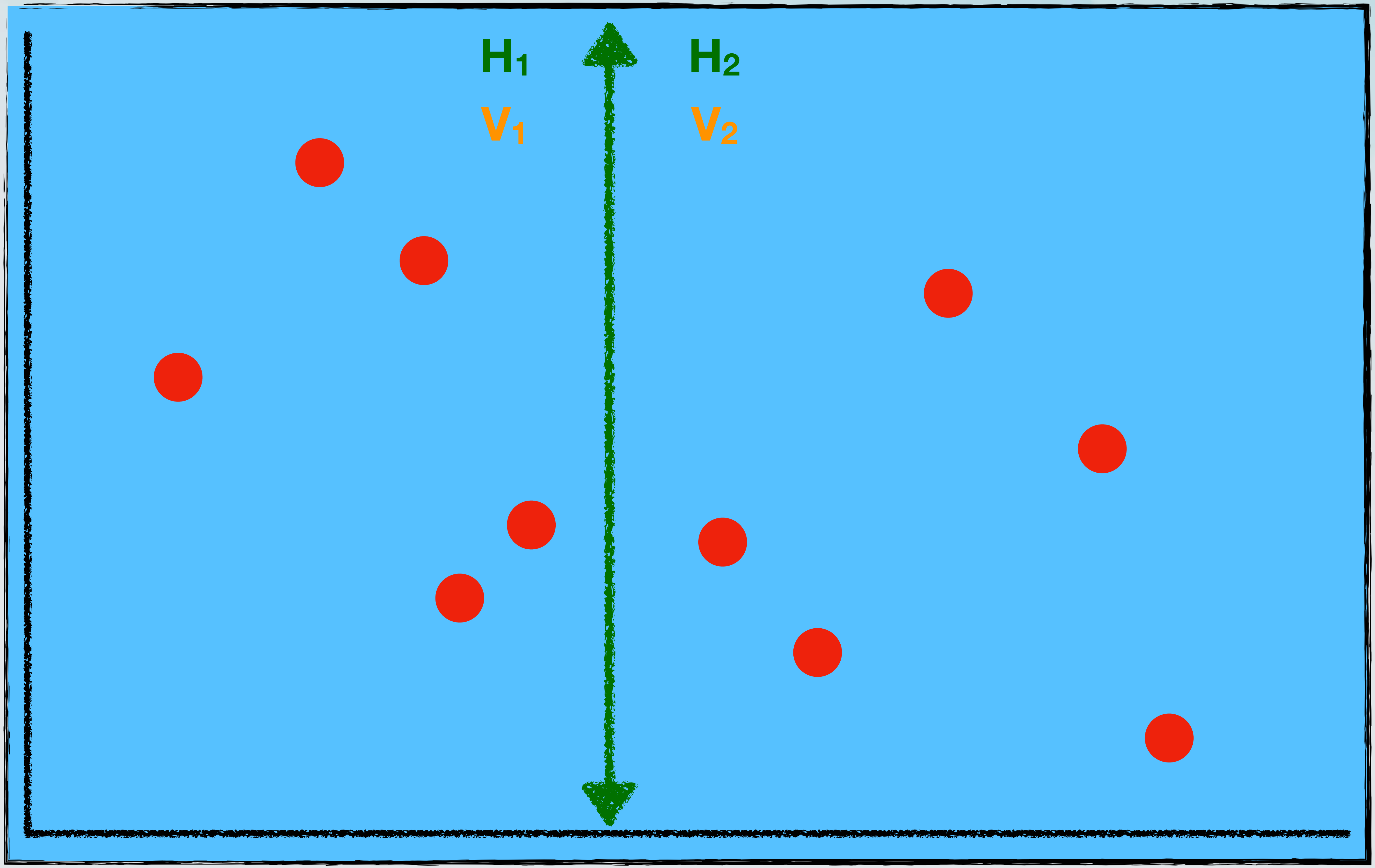
Main algorithm:

- Partition array **H** into two halves H_1 and H_2, according to the sorted order.

Finding the closest pair of points



Finding the closest pair of points



Finding the closest pair of points

Finding the closest pair of points

Main algorithm:

Finding the closest pair of points

Main algorithm:

- Partition array **H** into two halves **H_1** and **H_2**, according to the sorted order.

Finding the closest pair of points

Main algorithm:

- Partition array **H** into two halves **H_1** and **H_2**, according to the sorted order.
- For each element in **H_i**, put the element in **V_i** , for *$i=1,2$*.

Finding the closest pair of points

Main algorithm:

- Partition array **H** into two halves **H_1** and **H_2**, according to the sorted order.
- For each element in **H_i**, put the element in **V_i** , for *$i=1,2$*.
- Call the algorithm recursively on the two halves (with access to the sub-arrays **H_i** and **V_i** , for *$i=1,2$*.

Finding the closest pair of points

Main algorithm:

- Partition array **H** into two halves **H_1** and **H_2**, according to the sorted order.
- For each element in **H_i**, put the element in **V_i** , for *$i=1,2$*.
- Call the algorithm recursively on the two halves (with access to the sub-arrays **H_i** and **V_i** , for *$i=1,2$*.
- Let (l_1,l_2) and $(r1,r2)$ be the set of points returned by the runs of the algorithm on the two halves.

Finding the closest pair of points

Main algorithm:

- Partition array **H** into two halves **H_1** and **H_2**, according to the sorted order.
- For each element in **H_i**, put the element in **V_i** , for $i=1,2$.
- Call the algorithm recursively on the two halves (with access to the sub-arrays **H_i** and **V_i** , for $i=1,2$.
- Let (l_1,l_2) and $(r1,r2)$ be the set of points returned by the runs of the algorithm on the two halves.
    - We haven't really developed that part yet!

ClosestPair Pseudocode

Algorithm ClosestPair($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

Procedure ClosestPairRec(**H**,**V**)

Construct **H**$_1$, **H**$_2$, **V**$_1$, **V**$_2$

(l_1,l_2) = ClosestPairRec(**H**$_1$,**V**$_1$)

(r_1,r_2) = ClosestPairRec(**H**$_2$,**V**$_2$)

ClosestPair Pseudocode

Algorithm **ClosestPair**($p_1,\dots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = **ClosestPairRec**(**H**,**V**)

Procedure **ClosestPairRec**(**H**,**V**)

If $|H| = |V| \leq 3$

Check all pairwise distances

Construct H_1, H_2, V_1, V_2

(l_1,l_2) = **ClosestPairRec**(H_1,V_1)

(r_1,r_2) = **ClosestPairRec**(H_2,V_2)

Divide and Conquer

Divide and Conquer

- We have successfully divided the problem into smaller parts.

Divide and Conquer

- We have successfully divided the problem into smaller parts.

- How do we combine these parts to get a solution to the original problem?

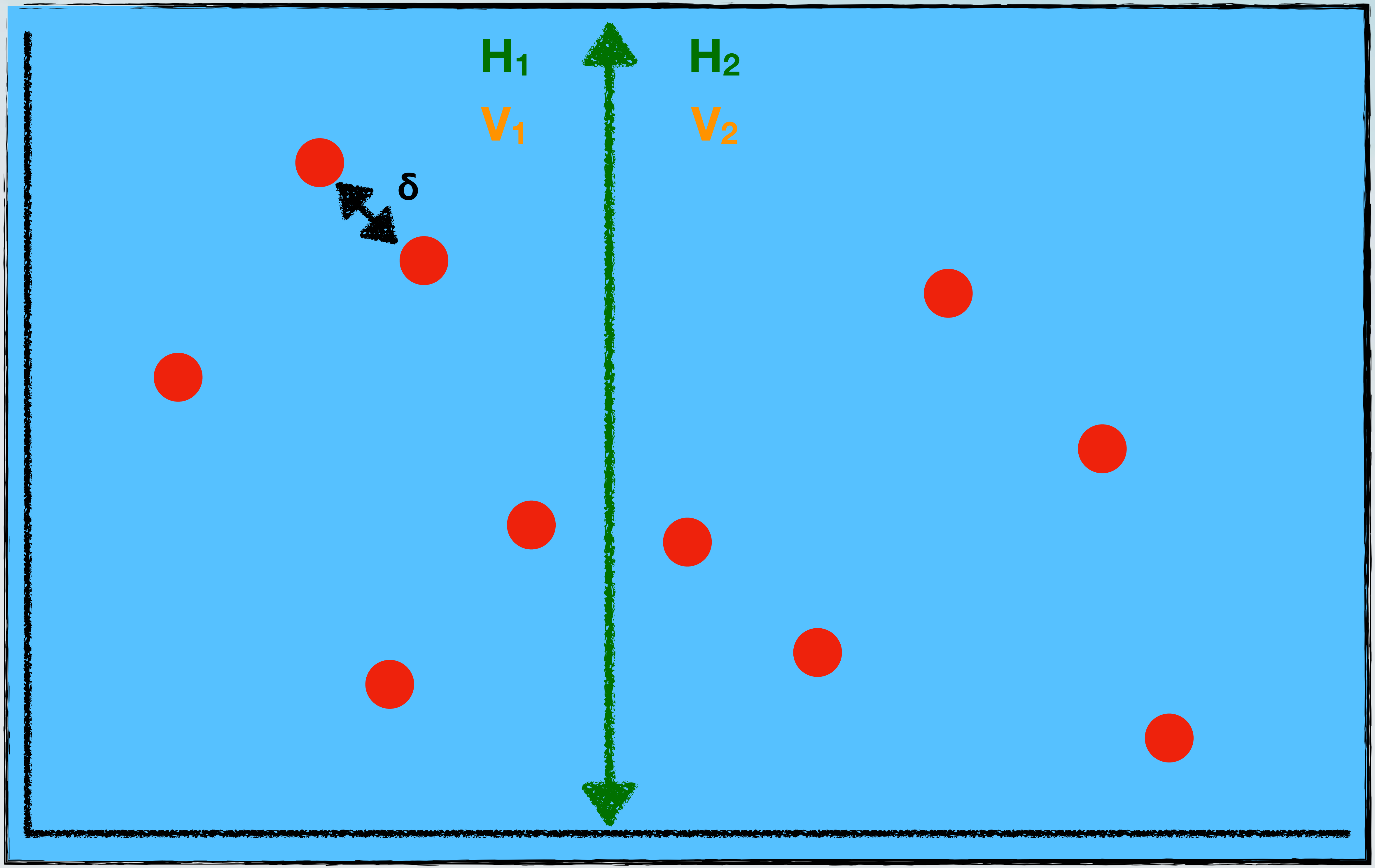
Divide and Conquer

- We have successfully divided the problem into smaller parts.
- How do we combine these parts to get a solution to the original problem?
- What might be the problem here?

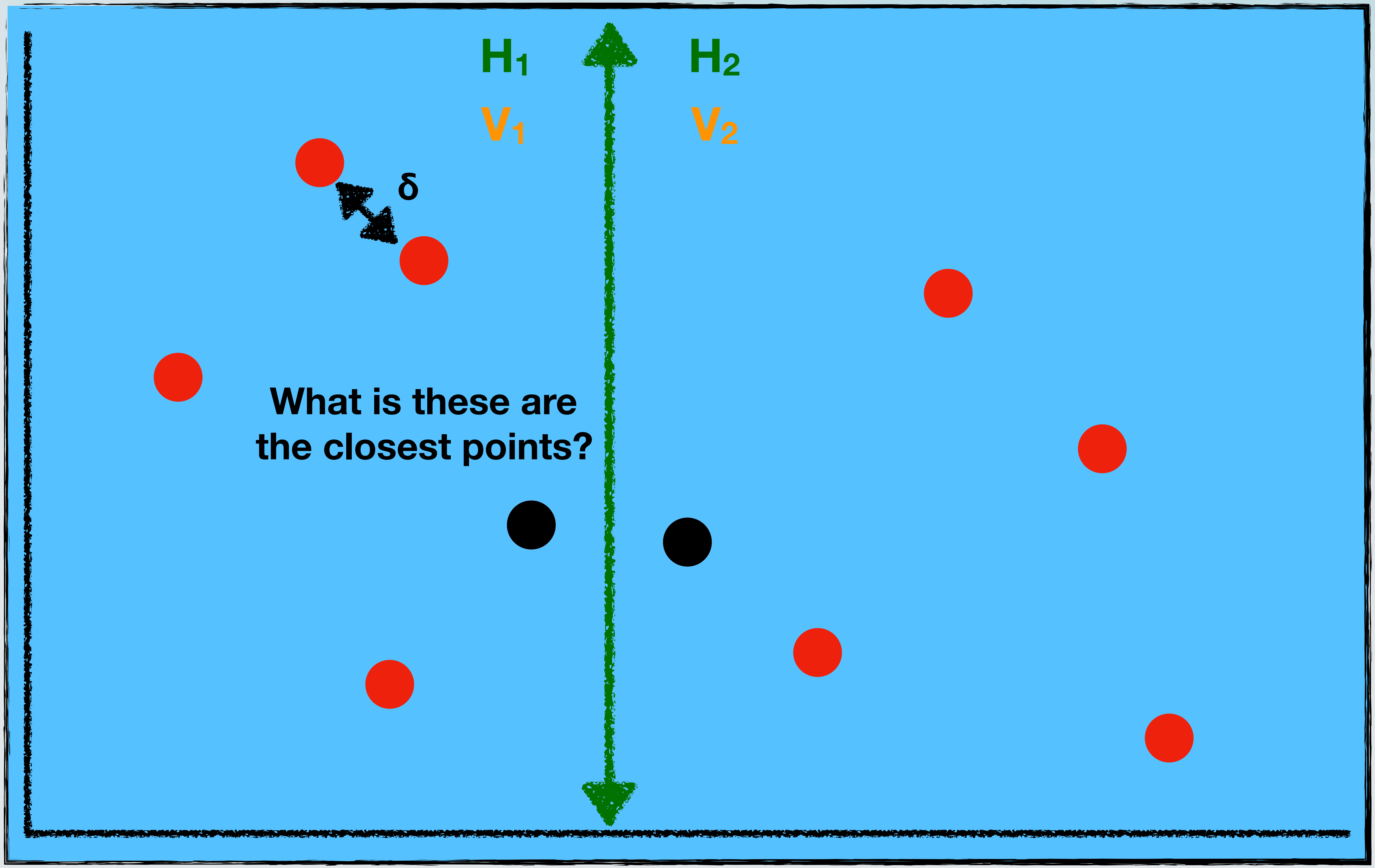
Divide and Conquer

- We have successfully divided the problem into smaller parts.
- How do we combine these parts to get a solution to the original problem?
- What might be the problem here?
- What if the smallest distance is between points in (**H_1**,**V_1**) and (**H_2**,**V_2**)?

Finding the closest pair of points



Finding the closest pair of points

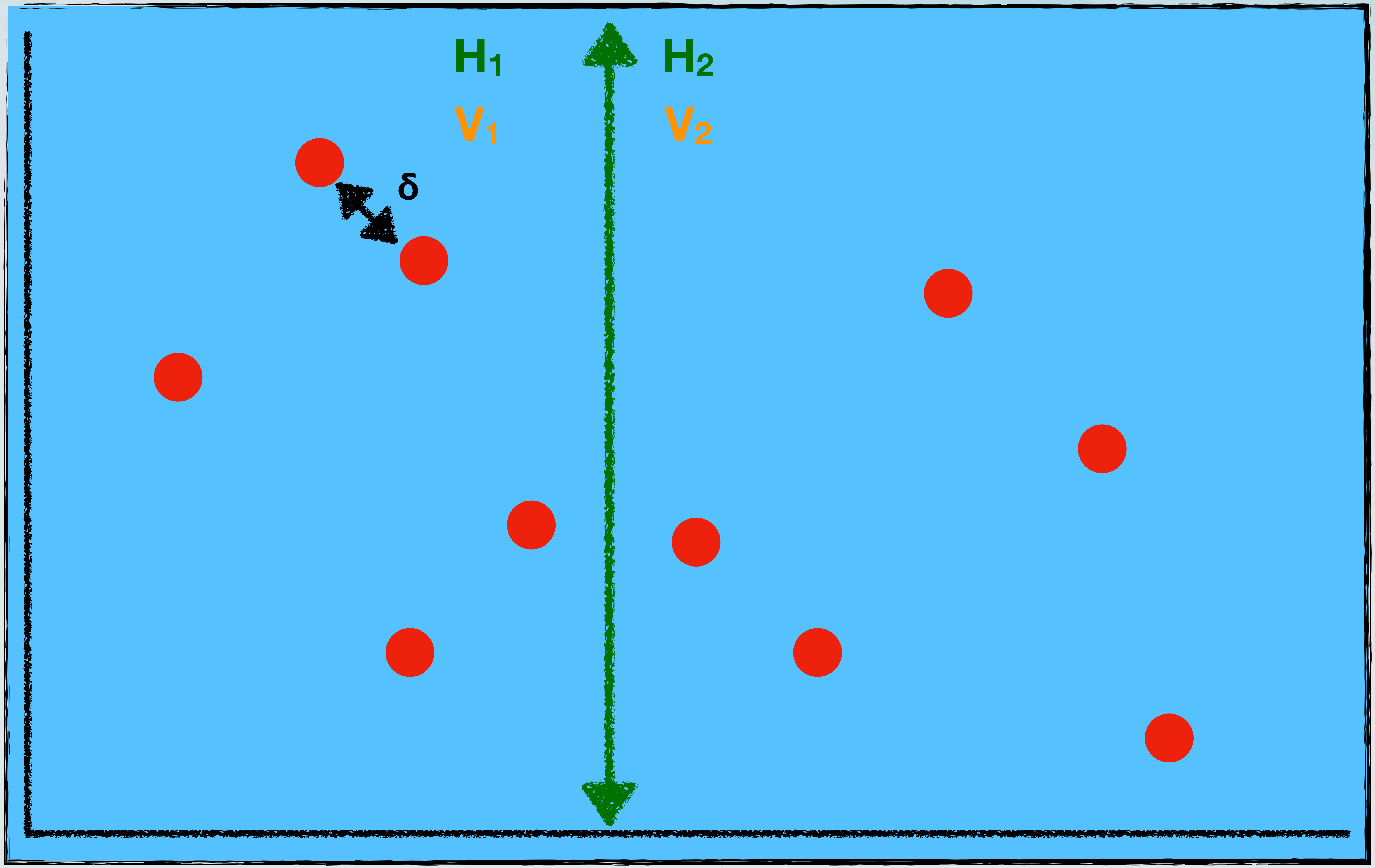


Combining the solutions

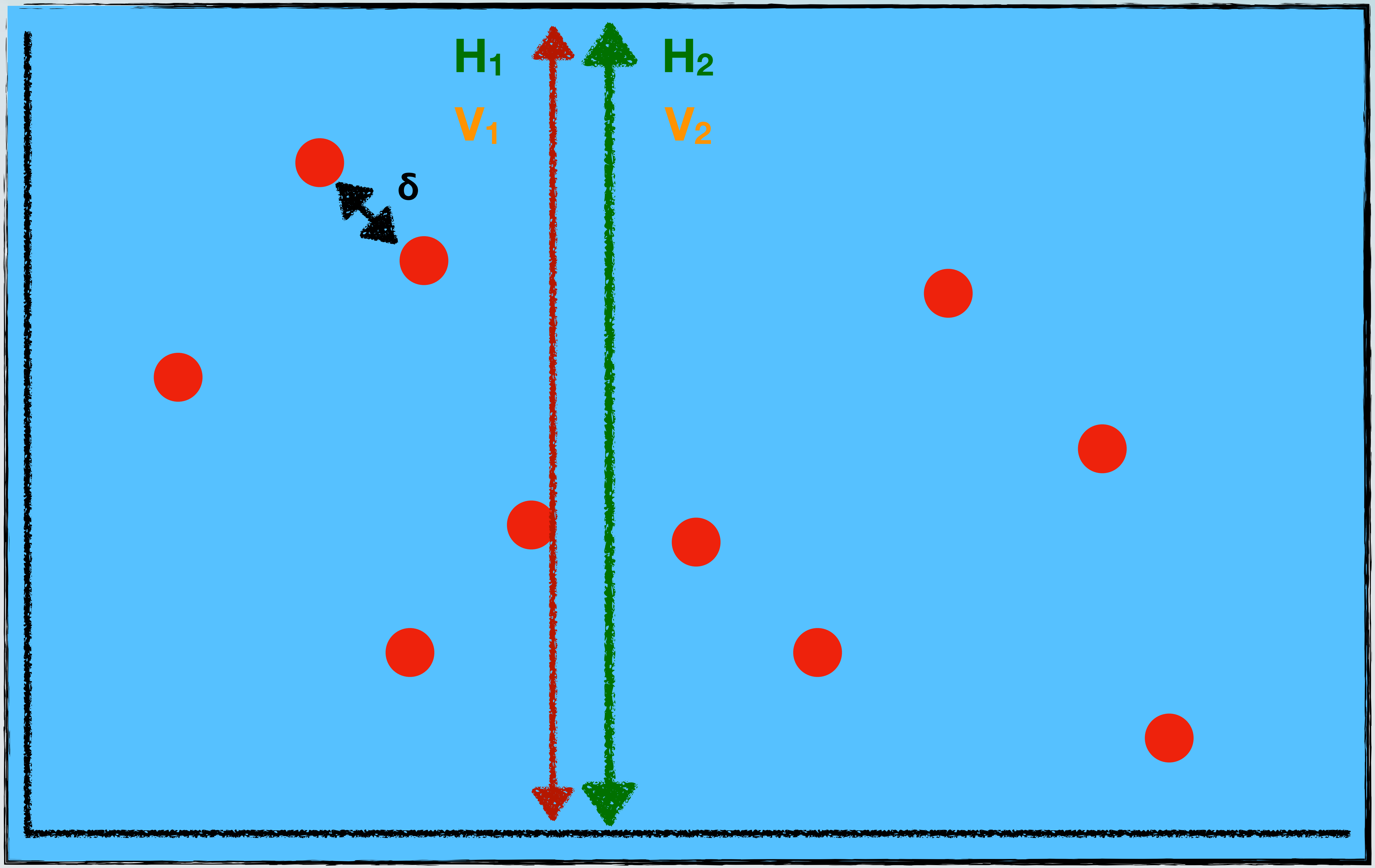
- Let δ be the $\min(d(l_1,l_2), d(r_1,r_2))$ be the minimum distance among the two solutions provided.

- Draw a vertical line **L** over the rightmost point of the set **H_1**.

Finding the closest pair of points



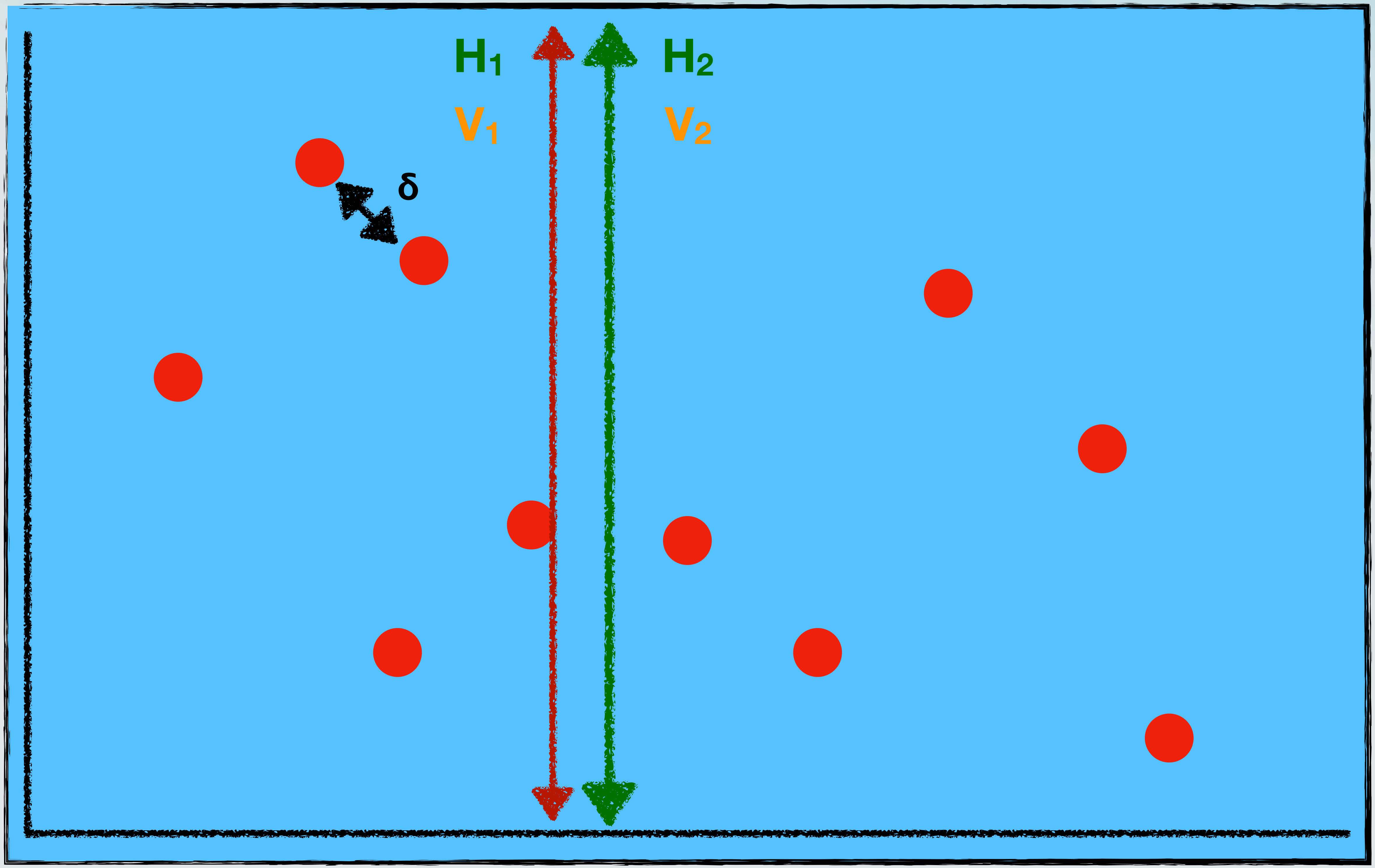
Finding the closest pair of points



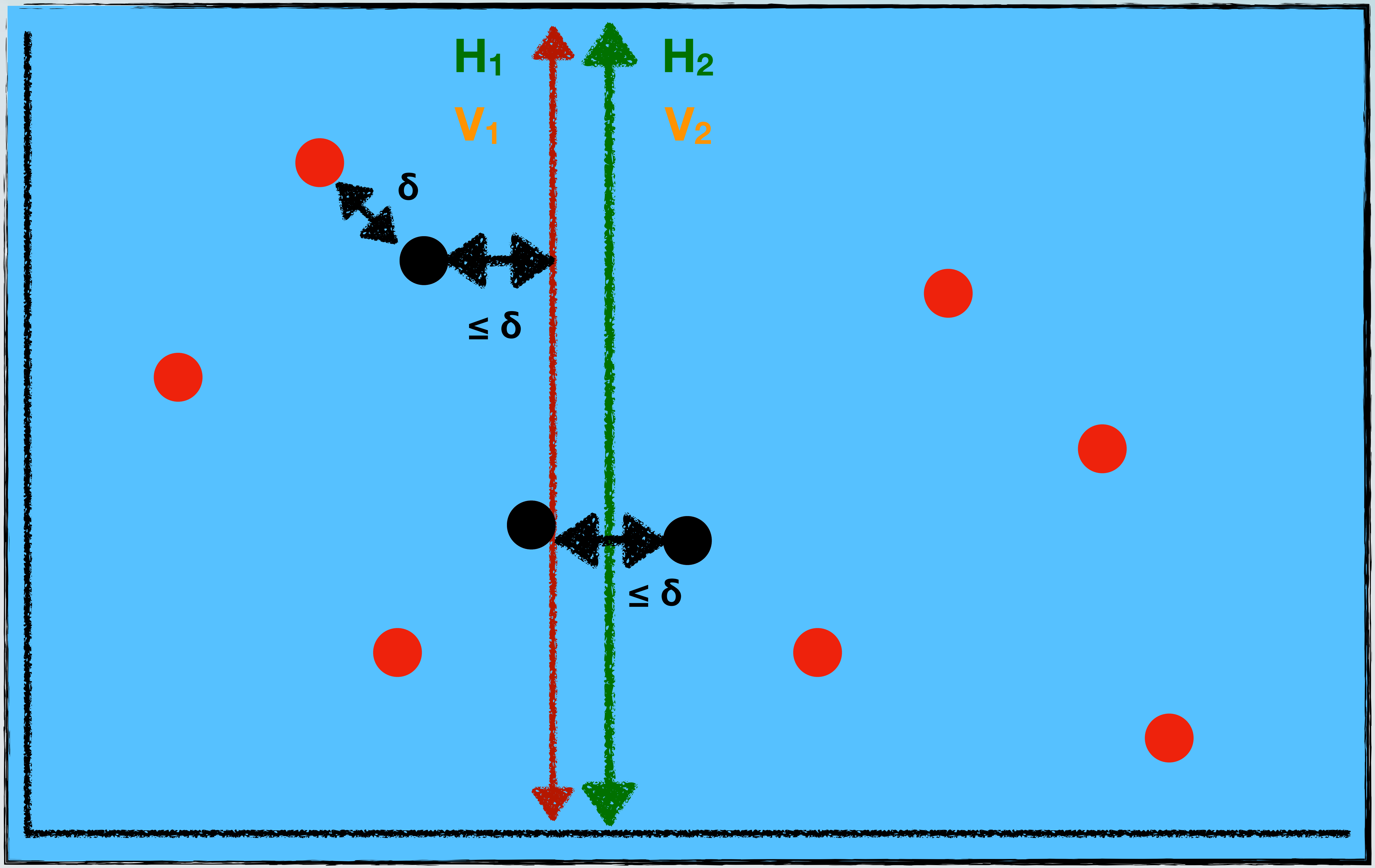
Combining the solutions

- Let δ be the $\min(d(l_1,l_2), d(r_1,r_2))$ be the minimum distance among the two solutions provided.
- Draw a vertical line **L** over the rightmost point of the set **H_1**.
    - This basically means that the separating line is a “tight” as possible.
- Let **S** be the set of points of distance within δ of **L**.

Finding the closest pair of points



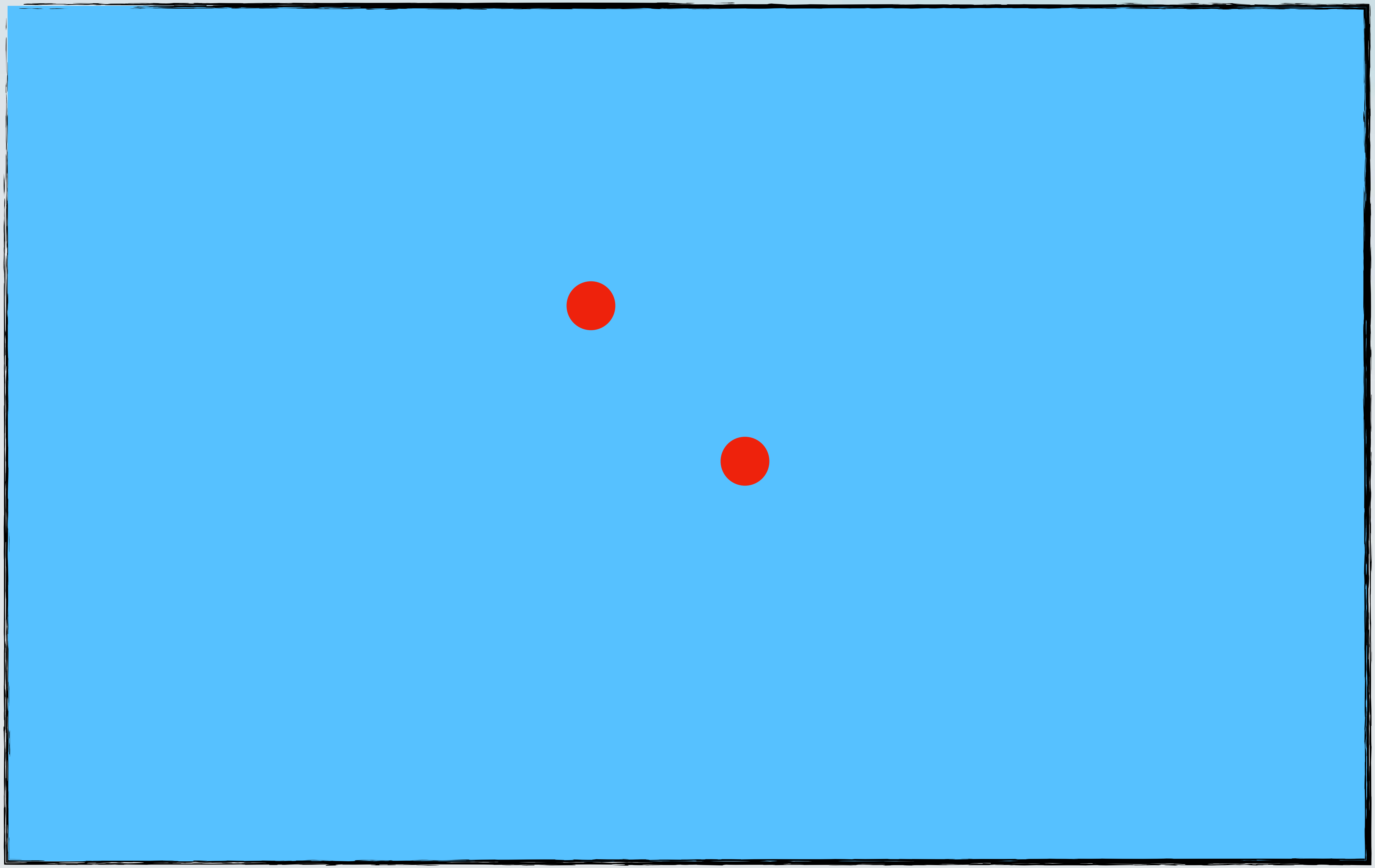
Finding the closest pair of points



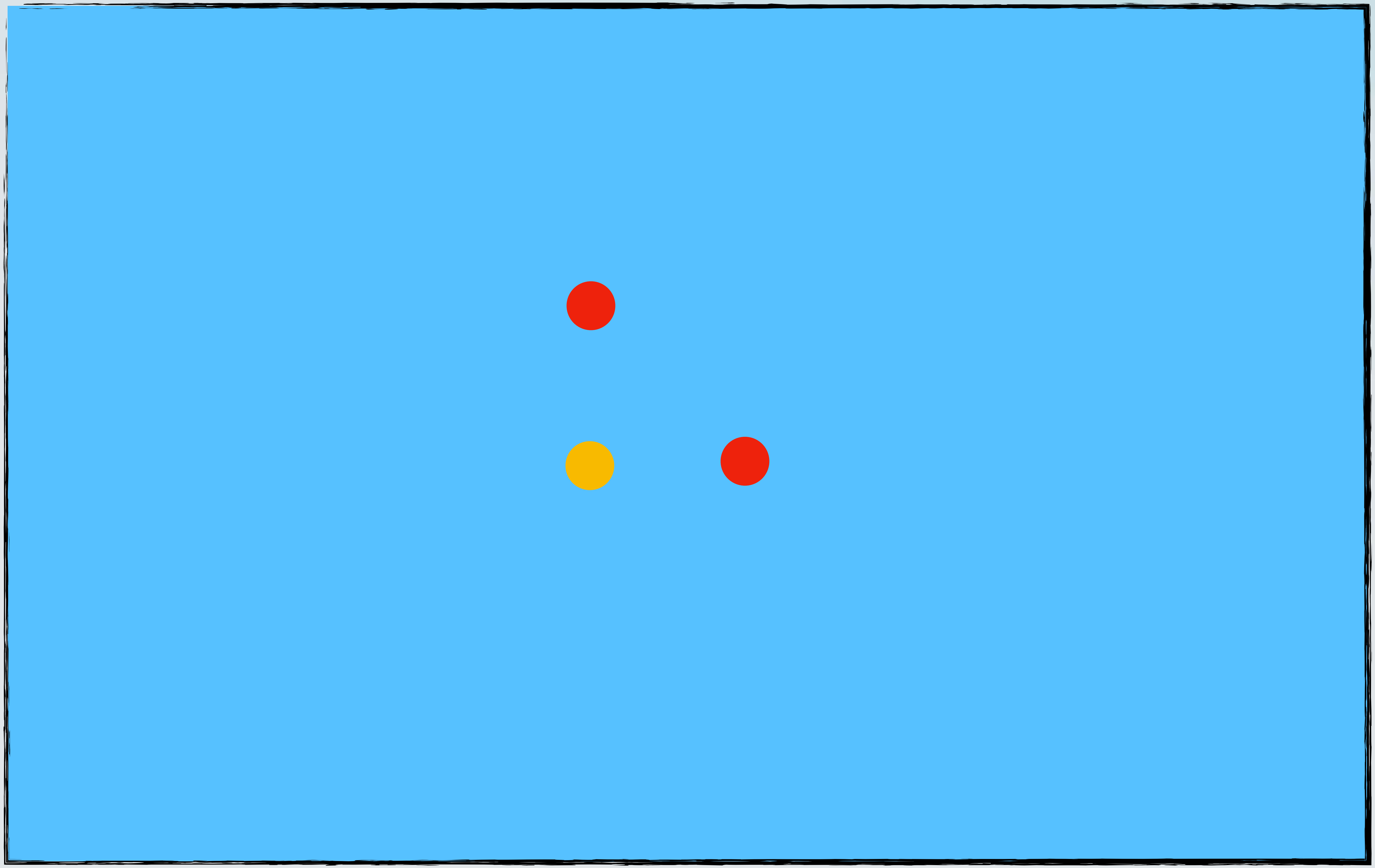
Combining the solutions

- Let δ be the $\min(d(l_1,l_2), d(r_1,r_2))$ be the minimum distance among the two solutions provided.
- Draw a vertical line **L** over the rightmost point of the set **H_1.**
  - This basically means that the separating line is a "tight" as possible.
- Let **S** be the set of points of distance within δ of **L**.

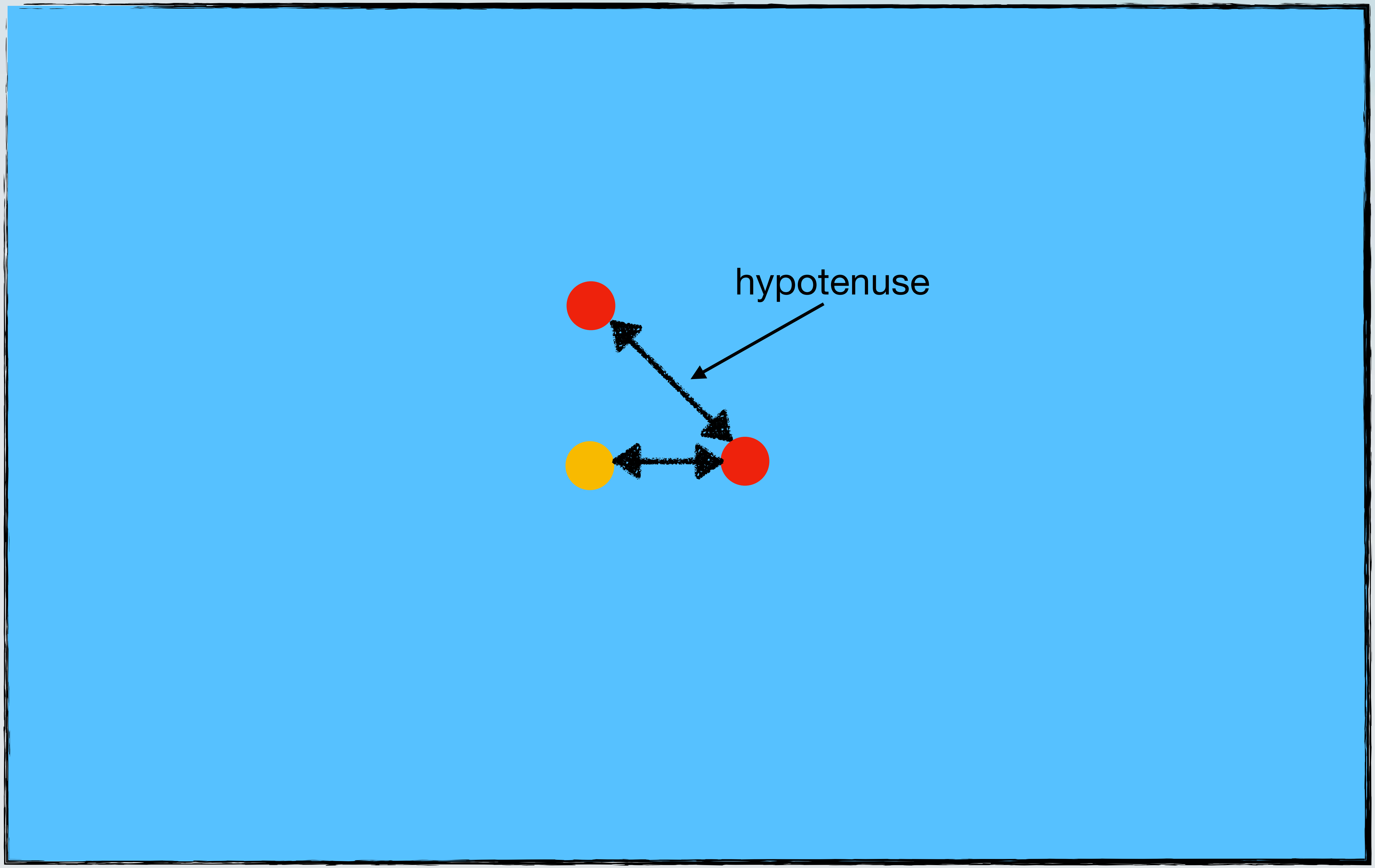
Is it safe to only consider the points in S?



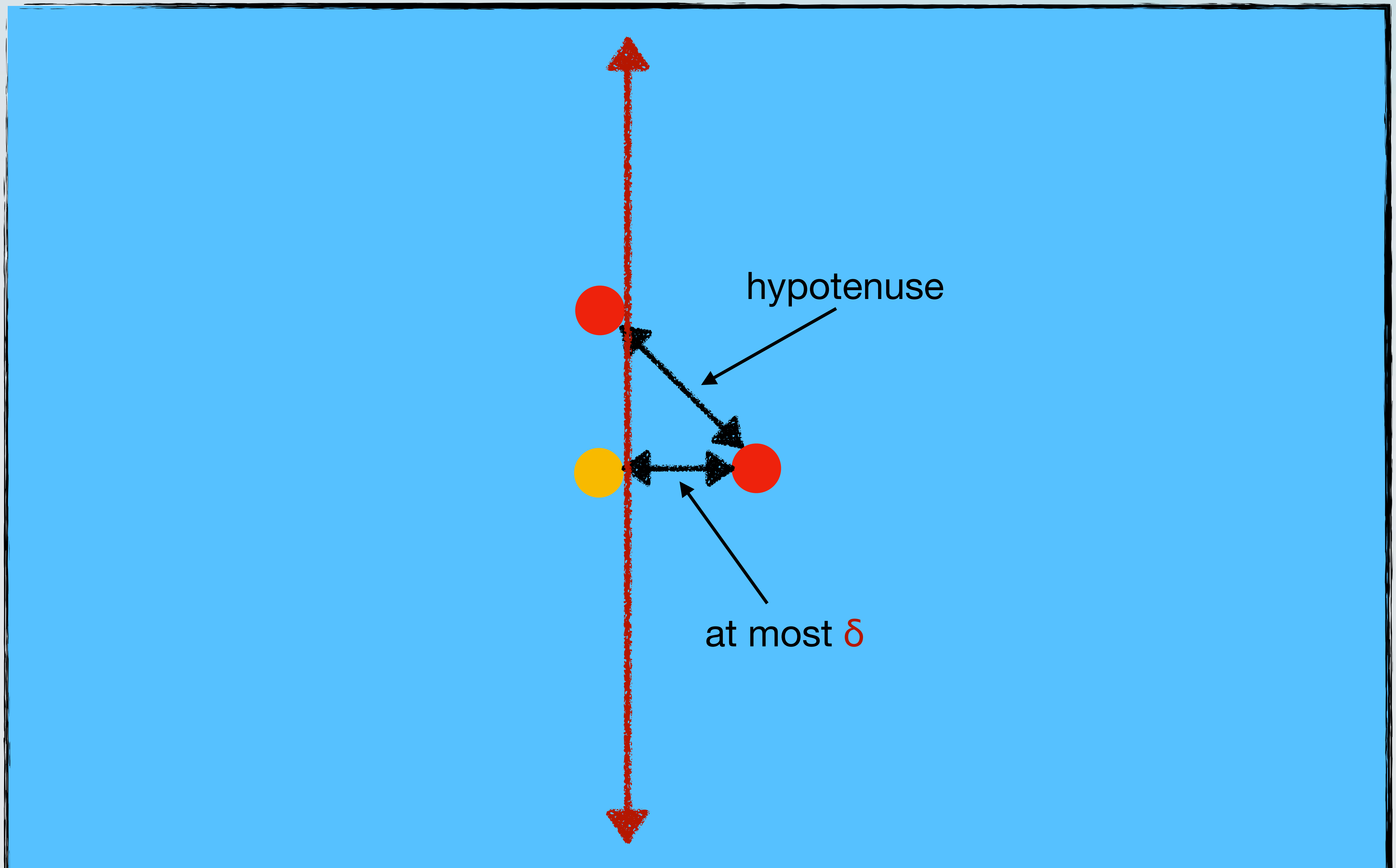
Is it safe to only consider the points in S?



Is it safe to only consider the points in S?



Is it safe to only consider the points in S?



Constructing the set S

Array V

1	2	4	6	10	14	17	19	21	24

Constructing the set S

Array V

1	2	4	6	10	14	17	19	21	24

Constructing the set S

Array V

1	2	4	6	10	14	17	19	21	24

Do we get more than a set?

Constructing the set **S**

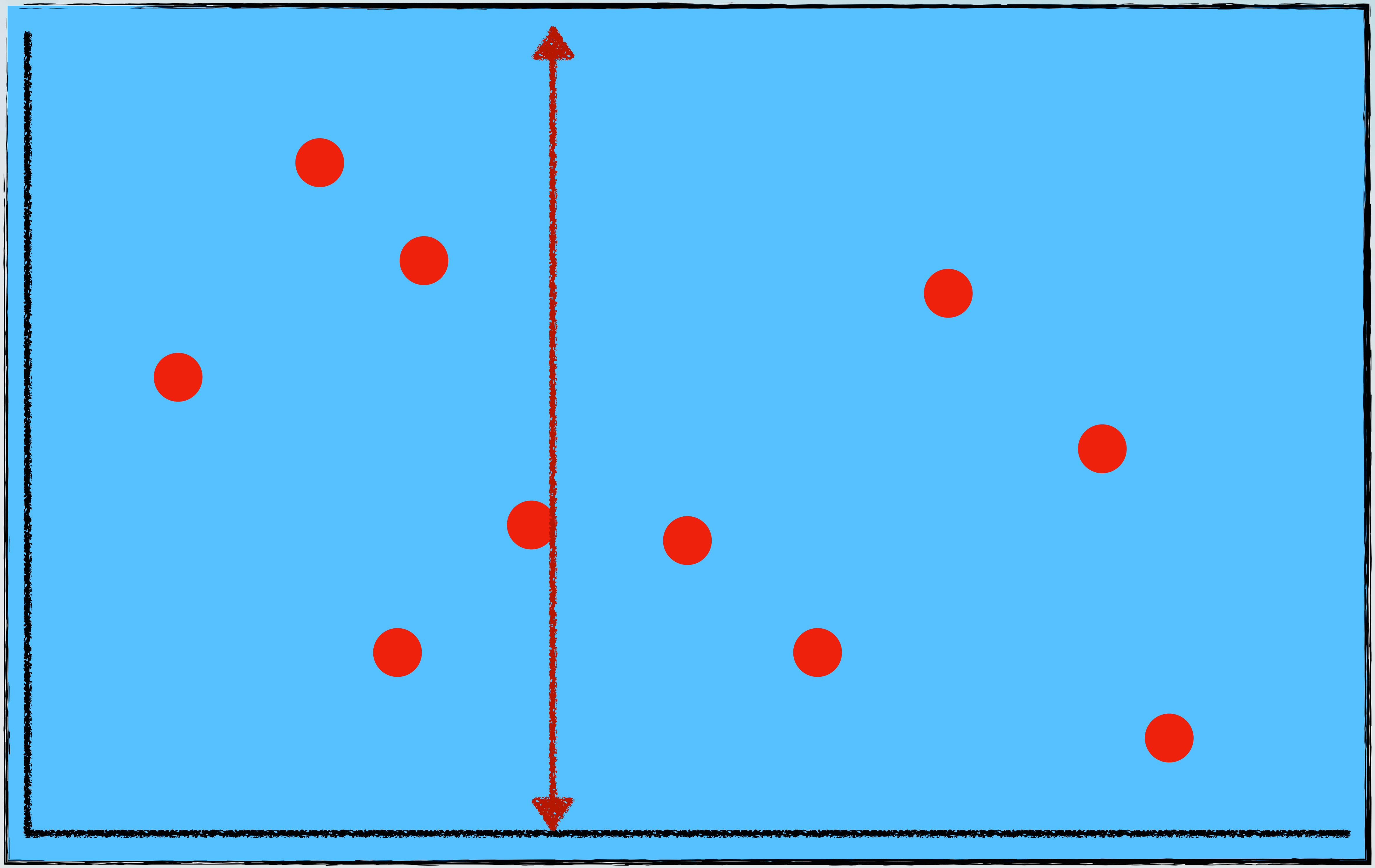
Array **V**

1	2	4	6	10	14	17	19	21	24

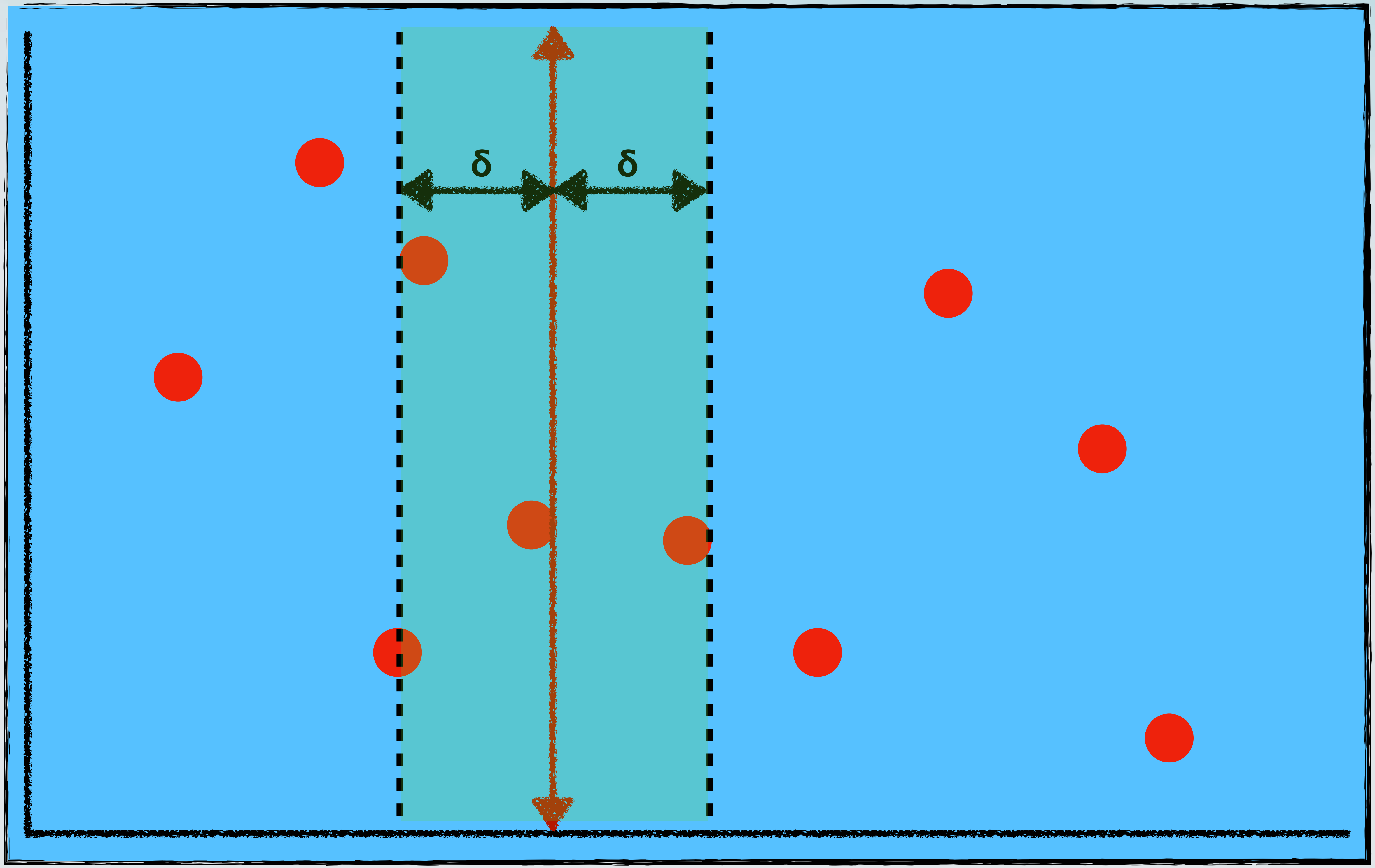
Do we get more than a set?

We actually get a **sorted list!**
(sorted in the y-coordinate)
Call this S_v.

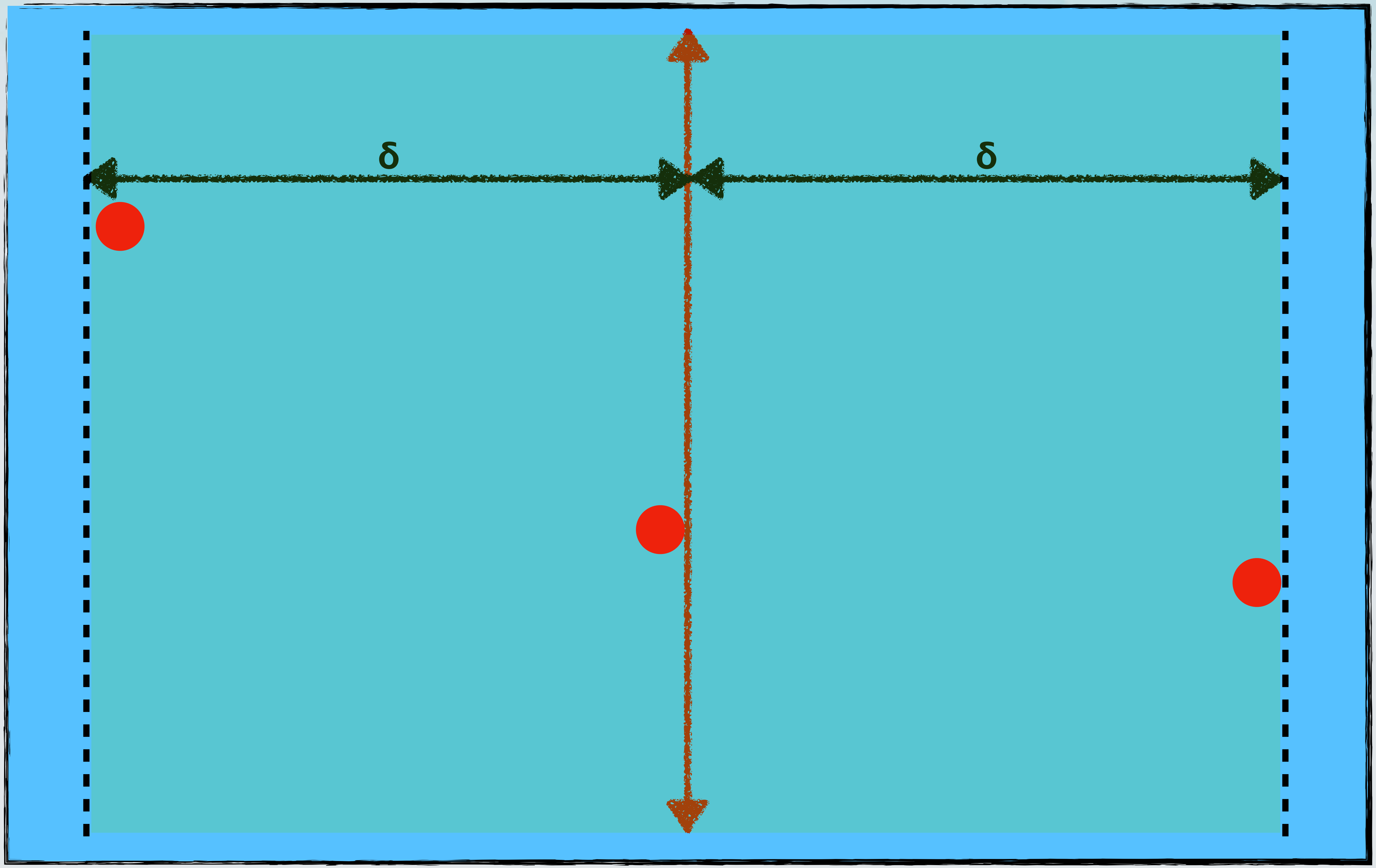
Finding the closest pair of points



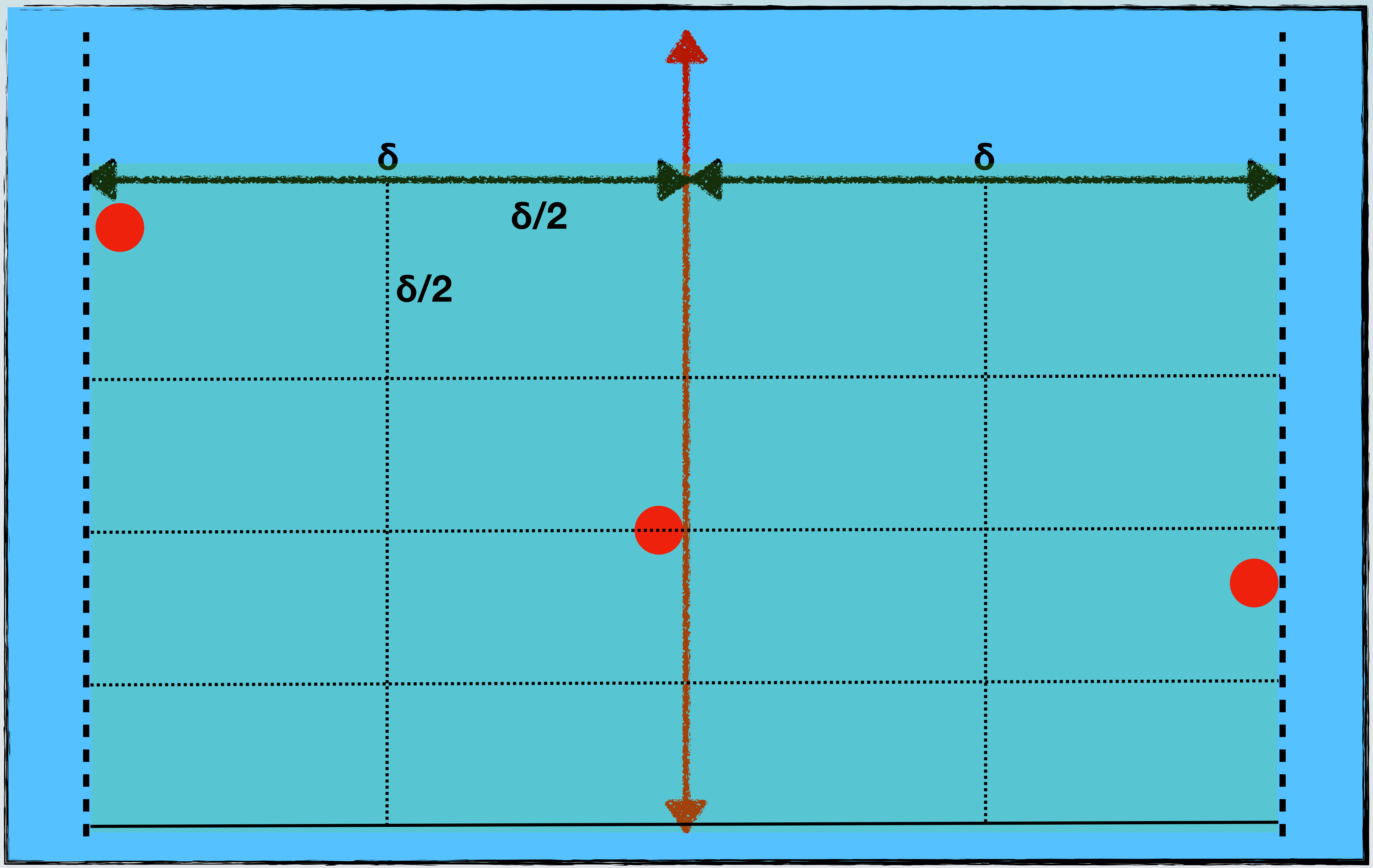
Finding the closest pair of points



Zooming in



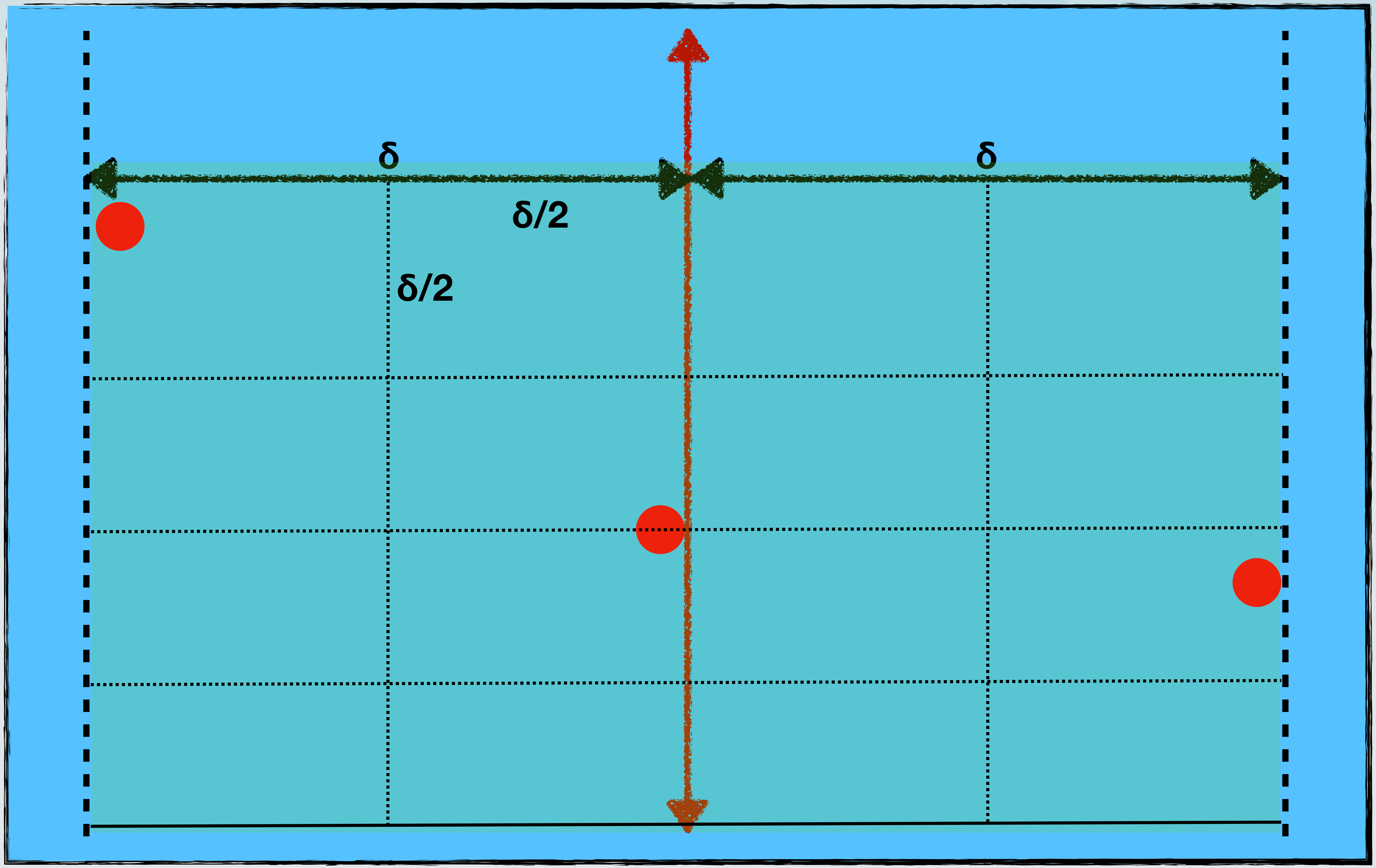
Partitioning the square



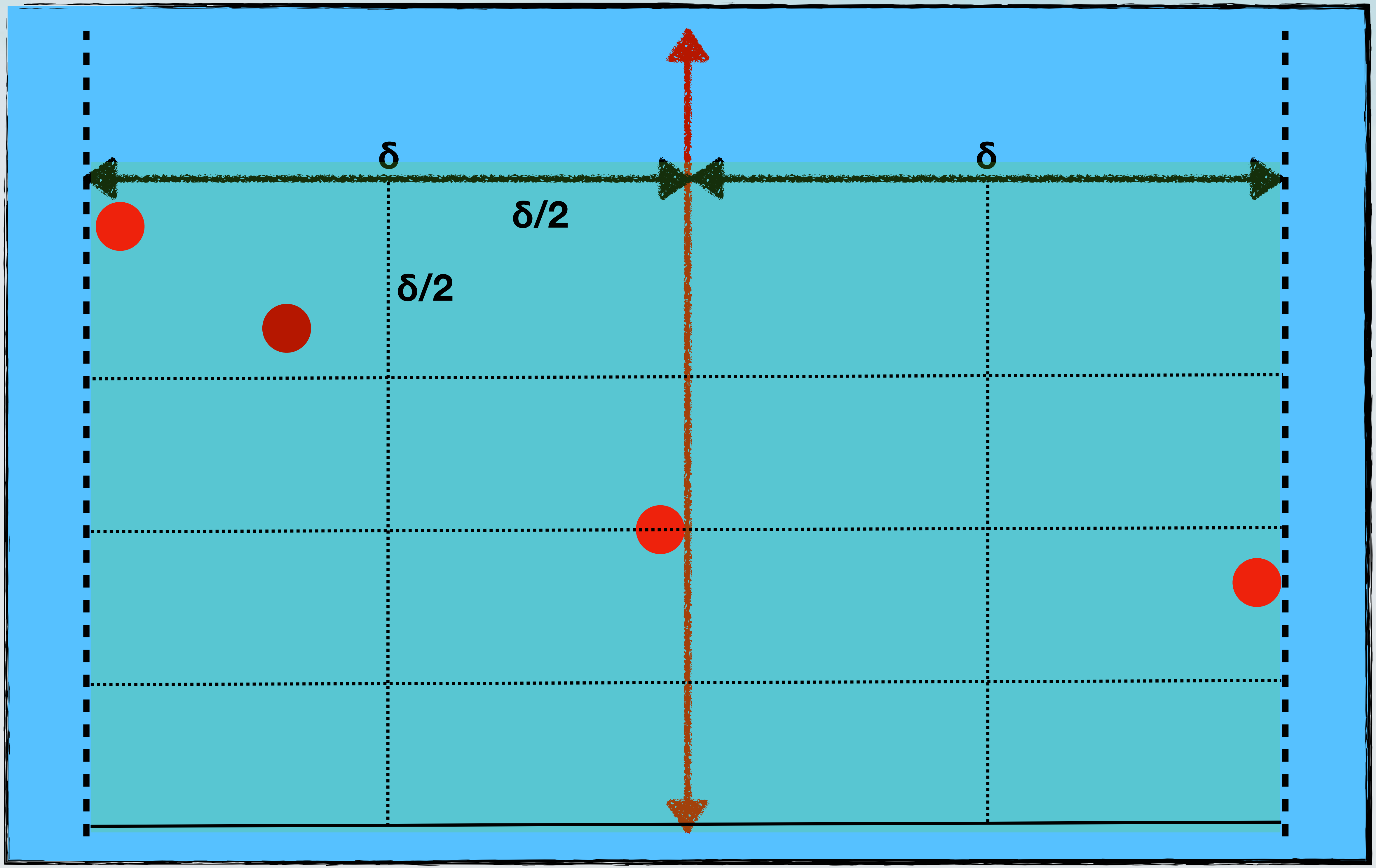
Claims

- **Claim 1:** In each box, there can only be a single point.

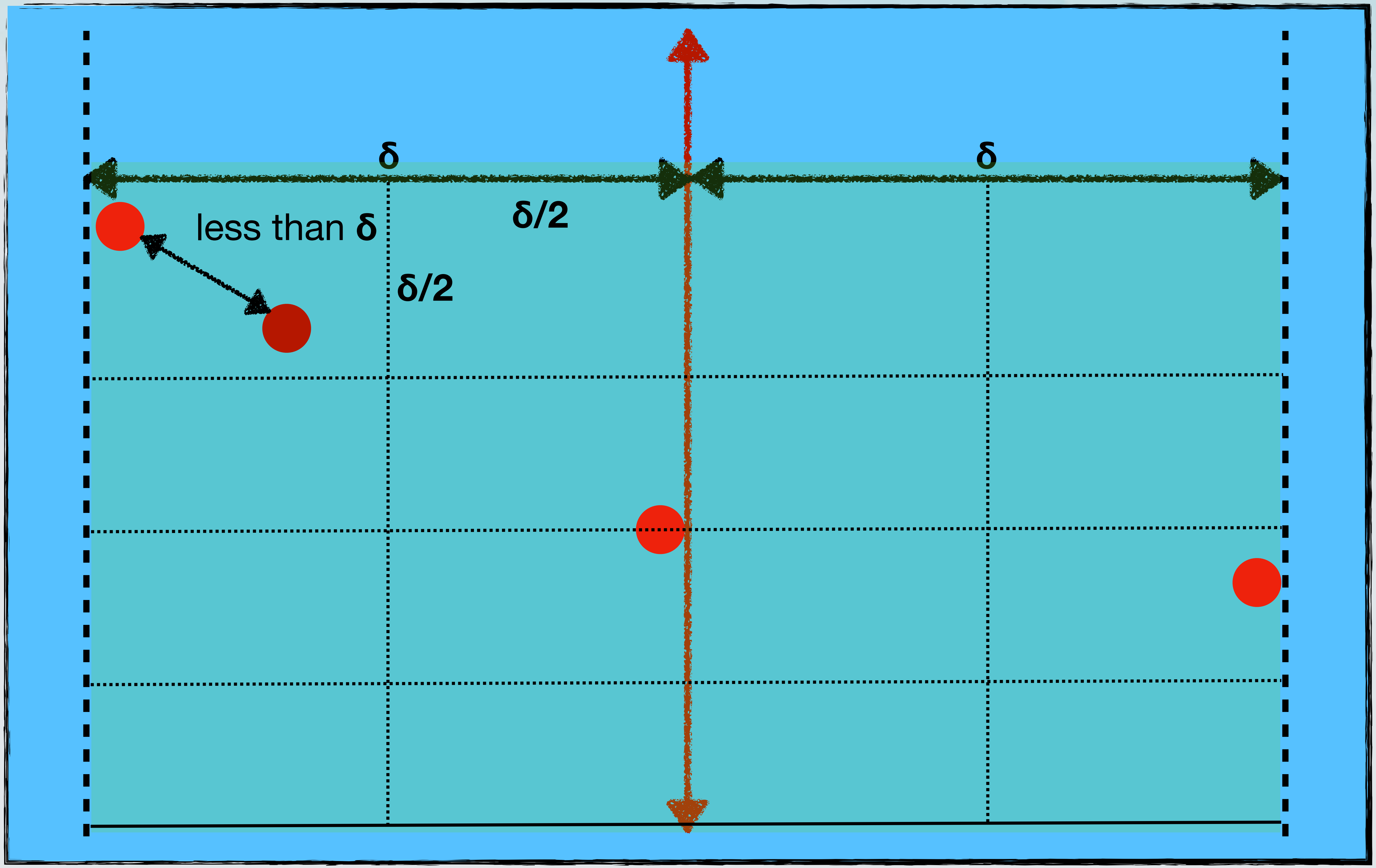
Partitioning the square



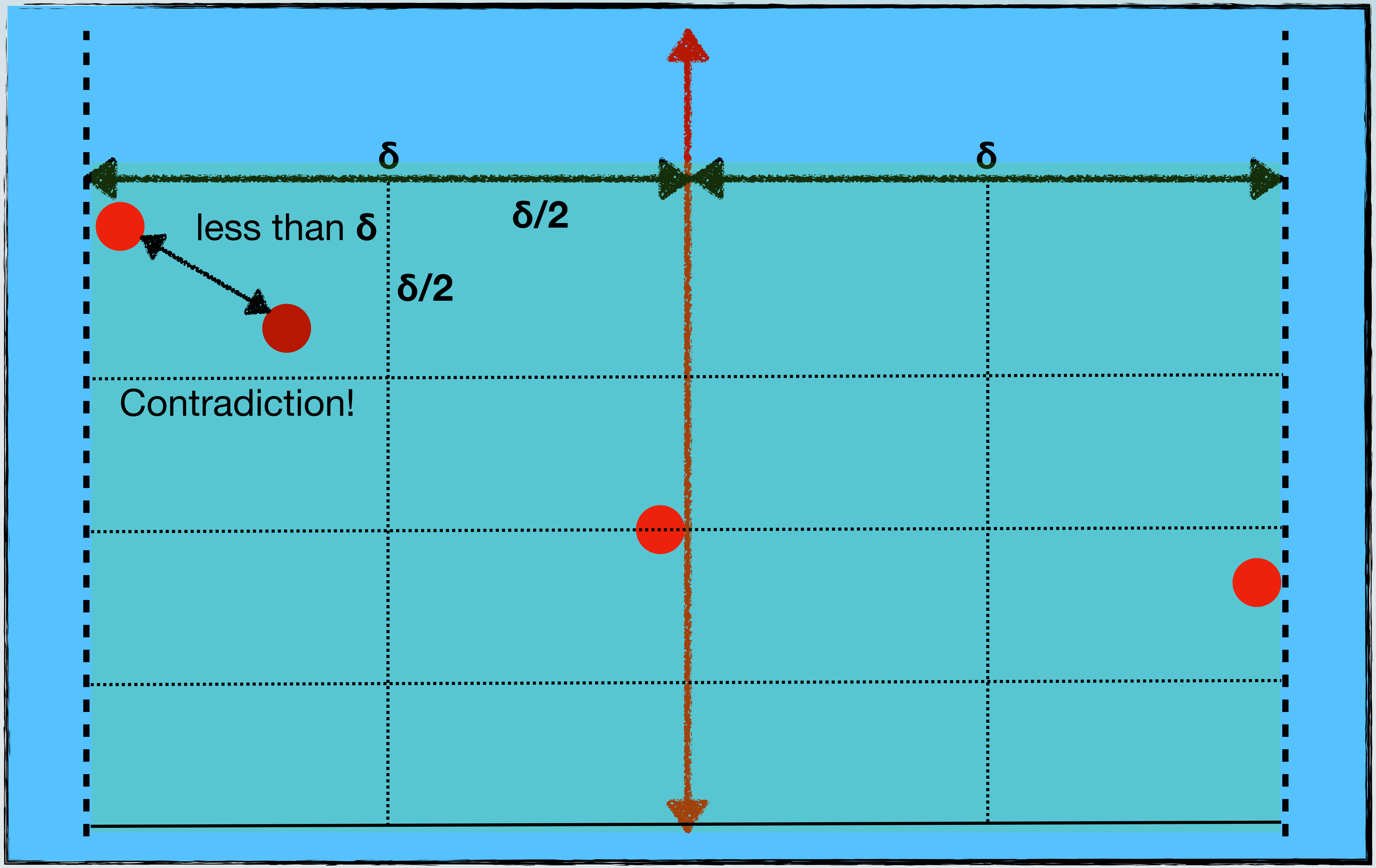
Partitioning the square



Partitioning the square



Partitioning the square



Claims

- **Claim 1:** In each box, there can only be a single point.
- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$
  - In other words, the two points are within 15 positions of each other in the sorted array $\mathbf{S_v}$.

Claims

Claims

- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$

Claims

- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$
- Assume by contradiction that this is not the case, and p_1 and p_2 are at least 16 positions apart in $\mathbf{S_v}$.

Claims

- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$

- Assume by contradiction that this is not the case, and p_1 and p_2 are at least 16 positions apart in $\mathbf{S_v}$.

- By **Claim 1**, there can be at most one point in each box.

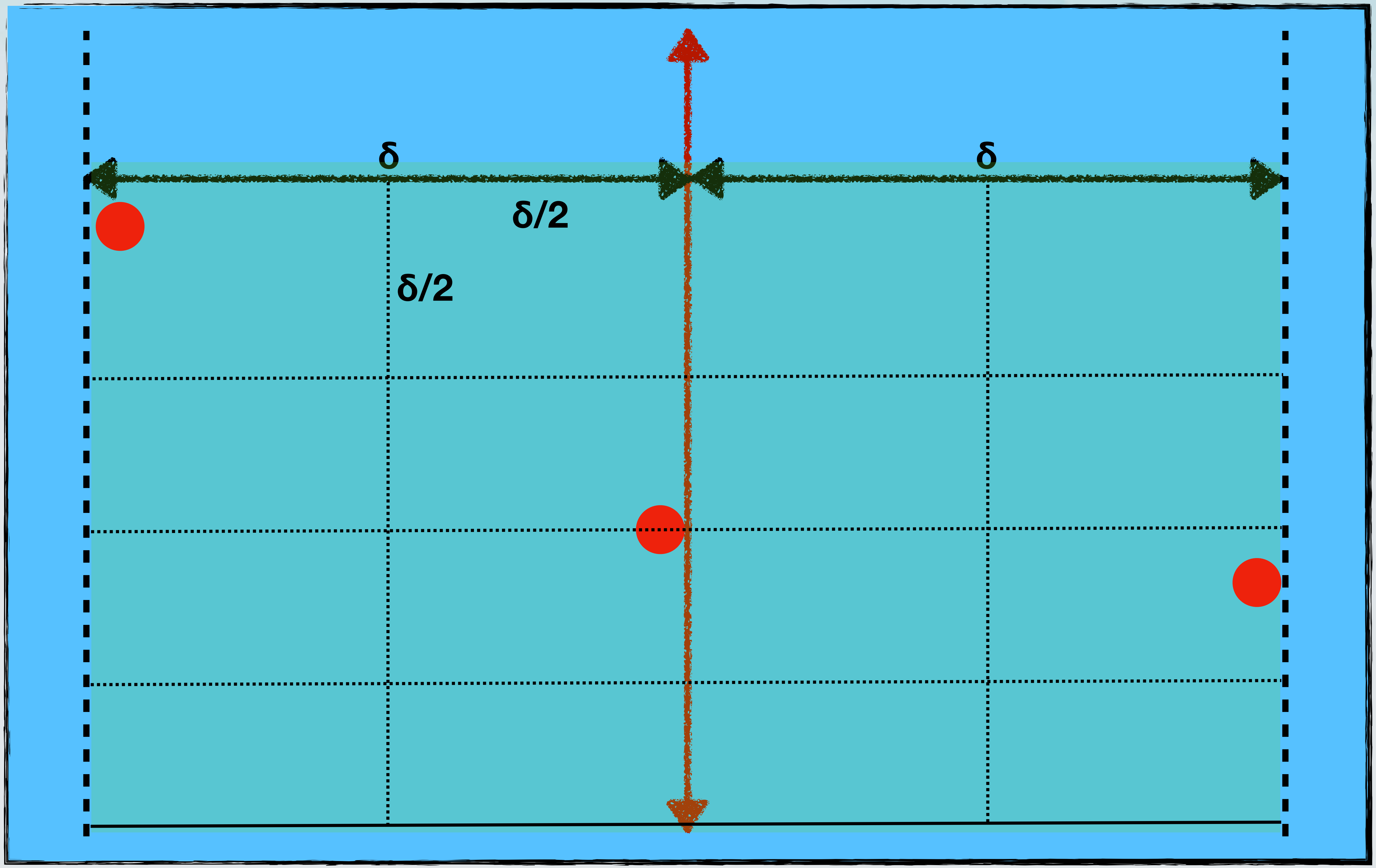
Claims

- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$
- Assume by contradiction that this is not the case, and p_1 and p_2 are at least 16 positions apart in $\mathbf{S_v}$.
- By **Claim 1**, there can be at most one point in each box.
    - To be 16 positions apart, there must be at least **3 rows** of boxes separating the points.

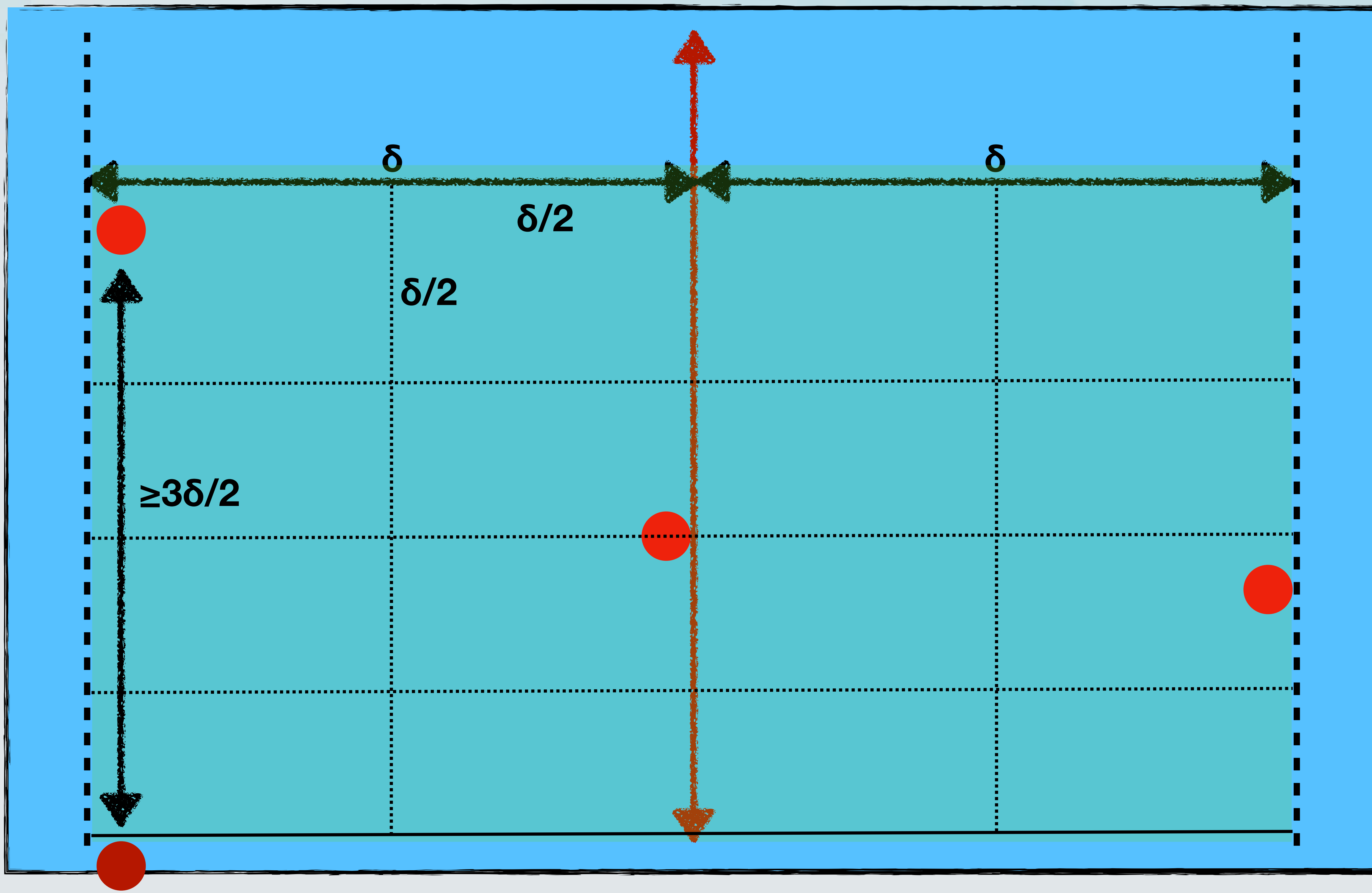
Claims

- **Claim 2:** If two points p_1 and p_2 are such that $d(p_1,p_2) < \delta$, then $\mathbf{S_v}[p_{1y}] - \mathbf{S_v}[p_{2y}] \leq 15$
- Assume by contradiction that this is not the case, and p_1 and p_2 are at least 16 positions apart in $\mathbf{S_v}$.
- By **Claim 1**, there can be at most one point in each box.
    - To be 16 positions apart, there must be at least **3 rows** of boxes separating the points.
    - But then the distance is at least $3\delta/2$, a contradiction.

Partitioning the square



Partitioning the square



Concluding the algorithm

Concluding the algorithm

- Make pass through S_v, and for each element, find the distance to the next **15** elements in the array.

Concluding the algorithm

- Make pass through $\mathbf{S_v}$, and for each element, find the distance to the next **15** elements in the array.
- Find the pair of points p_1 and p_2 for which the minimum of all these distances is achieved.

Concluding the algorithm

- Make pass through $\mathbf{S_v}$, and for each element, find the distance to the next **15** elements in the array.
- Find the pair of points p_1 and p_2 for which the minimum of all these distances is achieved.
- Compare this to δ and output the pair with the minimum distance.

ClosestPair Pseudocode

Algorithm ClosestPair($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

Procedure ClosestPairRec(**H**,**V**)

If |**H**| = |**V**| ≤ 3

Check all pairwise distances

Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = ClosestPairRec($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = ClosestPairRec($\mathbf{H_2}$,$\mathbf{V_2}$)

$\delta = \min(d(l_1,l_2), d(r_1,r_2))$

$x^* = \max_i x_i$ for $i=1,\ldots,n$

$\mathbf{L} = \{(x,y) : x=x^*\}$

S = set of points within distance δ of **L.**

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s

and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2)

that minimises this distance.

If $d(s_1,s_2) \leq \delta$

Return (s_1,s_2)

Else if $d(l_1,l_2) < d(r_1,r_2)$

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm ClosestPair($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

Procedure ClosestPairRec(**H**,**V**)

If |**H**| = |**V**| ≤ 3

Check all pairwise distances

Construct **H₁**, **H₂**, **V₁**, **V₂**

(l_1,l_2) = ClosestPairRec($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = ClosestPairRec($\mathbf{H_2}$,$\mathbf{V_2}$)

$\delta = \min(d(l_1,l_2), d(r_1,r_2))$

$x^* = \max_i x_i$ for $i=1,\ldots,n$

L = $\{(x,y) : x=x^*\}$

S = set of points within distance δ of **L.**

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s

and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2)

that minimises this distance.

If $d(s_1,s_2) \leq \delta$

Return (s_1,s_2)

Else if $d(l_1,l_2) < d(r_1,r_2)$

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm **ClosestPair**($p_1,\dots,p_n$)

    Construct arrays **H** and **V**

    (p_1,p_2) = **ClosestPairRec**(**H**,**V**)

Procedure **ClosestPairRec**(**H**,**V**)

    If $|\mathbf{H}| = |\mathbf{V}| \leq 3$

        Check all pairwise distances

    Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

    (l_1,l_2) = **ClosestPairRec**($\mathbf{H_1}$,$\mathbf{V_1}$)

    (r_1,r_2) = **ClosestPairRec**($\mathbf{H_2}$,$\mathbf{V_2}$)

    $\delta = \min(d(l_1,l_2), d(r_1,r_2))$

    $x^* = \max_i x_i$ for $i=1,\dots,n$

    $\mathbf{L} = \{(x,y) : x=x^*\}$

    S = set of points within distance δ of **L**.

?

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

    Compute distance between s

    and the next 15 points of $\mathbf{S_v}$.

    Return the pair of points (s_1,s_2)

    that minimises this distance.

If $d(s_1,s_2) \leq \delta$

    Return (s_1,s_2)

Else if $d(l_1,l_2) < d(r_1,r_2)$

    Return (l_1,l_2)

Else return (r_1,r_2)

Finding the closest pair of points

Finding the closest pair of points

Preprocessing:

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.
- To populate **H** and **V**, simply run through the given points $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, … , $p_n=(x_n,y_n)$ and put $x_1, x_2, \dots, x_n$ into **H** and $y_1, y_2, \dots, y_n$ into **V**.

Finding the closest pair of points

Preprocessing:

- Maintain two arrays **H** and **V** for the horizontal and vertical coordinates of the points respectively.
- To populate **H** and **V**, simply run through the given points $p_1=(x_1,y_1)$, $p_2=(x_2,y_2)$, … , $p_n=(x_n,y_n)$ and put $x_1, x_2, \ldots, x_n$ into **H** and $y_1, y_2, \ldots, y_n$ into **V**.
- Sort the points in **H** and in **V**, using some sorting algorithm.

Running Time

Algorithm ClosestPair($p_1,\ldots,p_n$)

?

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

Procedure ClosestPairRec(**H**,**V**)

If |**H**| = |**V**| ≤ 3

Check all pairwise distances

Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = ClosestPairRec($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = ClosestPairRec($\mathbf{H_2}$,$\mathbf{V_2}$)

δ = min(d(l_1,l_2), d(r_1,r_2))

x^* = $\max_i x_i$ for i=$1,\ldots,n$

L = {(x,y) : x=x^*}

S = set of points within distance δ of **L.**

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2) that minimises this distance.

If d(s_1,s_2) ≤ δ

Return (s_1,s_2)

Else if d(l_1,l_2) < d(r_1,r_2)

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm **ClosestPair**($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = **ClosestPairRec**(**H**,**V**)

O(n log n)

Procedure **ClosestPairRec**(**H**,**V**)

If |**H**| = |**V**| ≤ 3

Check all pairwise distances

Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = **ClosestPairRec**($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = **ClosestPairRec**($\mathbf{H_2}$,$\mathbf{V_2}$)

$\delta = \min(d(l_1,l_2), d(r_1,r_2))$

$x^* = \max_i x_i$ for $i=1,\ldots,n$

$\mathbf{L} = \{(x,y) : x=x^*\}$

S = set of points within distance δ of **L**.

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s

and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2)

that minimises this distance.

If $d(s_1,s_2) \le \delta$

Return (s_1,s_2)

Else if $d(l_1,l_2) < d(r_1,r_2)$

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm ClosestPair($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

O(n log n)

Procedure ClosestPairRec(**H**,**V**)

If |**H**| = |**V**| ≤ 3

Check all pairwise distances

Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = ClosestPairRec($\mathbf{H_1}$,$\mathbf{V_1}$)

?

(r_1,r_2) = ClosestPairRec($\mathbf{H_2}$,$\mathbf{V_2}$)

δ = min(d(l_1,l_2), d(r_1,r_2))

$x^* = \max_i x_i$ for i=$1,\ldots,n$

L = {(x,y) : x=x^*}

S = set of points within distance δ of **L.**

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s

and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2)

that minimises this distance.

If d(s_1,s_2) ≤ δ

Return (s_1,s_2)

Else if d(l_1,l_2) < d(r_1,r_2)

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm ClosestPair($p_1,\ldots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = ClosestPairRec(**H**,**V**)

O(n log n)

Procedure ClosestPairRec(**H**,**V**)

If $|\mathbf{H}| = |\mathbf{V}| \leq 3$

Check all pairwise distances

Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = ClosestPairRec($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = ClosestPairRec($\mathbf{H_2}$,$\mathbf{V_2}$)

O(n log n)
known recurrence

$\delta = \min(d(l_1,l_2), d(r_1,r_2))$

$x^* = \max_i x_i$ for $i=1,\ldots,n$

$\mathbf{L} = \{(x,y) : x=x^*\}$

S = set of points within distance δ of **L**.

Construct $\mathbf{S_v}$

For each point s in $\mathbf{S_v}$

Compute distance between s and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2) that minimises this distance.

If $d(s_1,s_2) \leq \delta$

Return (s_1,s_2)

Else if $d(l_1,l_2) < d(r_1,r_2)$

Return (l_1,l_2)

Else return (r_1,r_2)

Running Time

Algorithm **ClosestPair**($p_1,\dots,p_n$)

Construct arrays **H** and **V**

(p_1,p_2) = **ClosestPairRec**(**H**,**V**)

O(n log n)

Procedure **ClosestPairRec**(**H**,**V**)

If $|\mathbf{H}| = |\mathbf{V}| \leq 3$

Check all pairwise distances

O(n) Construct $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{V_1}$, $\mathbf{V_2}$

(l_1,l_2) = **ClosestPairRec**($\mathbf{H_1}$,$\mathbf{V_1}$)

(r_1,r_2) = **ClosestPairRec**($\mathbf{H_2}$,$\mathbf{V_2}$)

O(n log n)
known recurrence

$\delta = \min(d(l_1,l_2), d(r_1,r_2))$

$x^* = \max_i x_i$ for $i=1,\dots,n$

O(n) **L** = $\{(x,y) : x=x^*\}$

S = set of points within distance δ of **L**.

Construct $\mathbf{S_v}$ O(n)

For each point s in $\mathbf{S_v}$ O(n)

Compute distance between s

and the next 15 points of $\mathbf{S_v}$.

Return the pair of points (s_1,s_2)

that minimises this distance.

If $d(s_1,s_2) \leq \delta$

Return (s_1,s_2) O(1)

Else if $d(l_1,l_2) < d(r_1,r_2)$

Return (l_1,l_2)

Else return (r_1,r_2)

Integer Multiplication

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

```
    1100
x   1101
--------
```

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

$$\begin{array}{r} 1100 \\ \times\ 1101 \\ \hline 1100 \end{array}$$

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

```
   1100
x  1101
-------
   1100
  0000
```

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

```
      1100
  x   1101
----------
      1100
     0000
    1100
```

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

```
    1100
x   1101
--------
    1100
   0000
  1100
 1100
```

Integer Multiplication

```
     1100
 x   1101
---------
     1100
    0000
   1100
  1100
---------
```

Input: 2 integer numbers, in *binary*

n is the number of *bits* of each number

Integer Multiplication

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

```
      1100
   x  1101
  ---------
      1100
     0000
    1100
   1100
  ---------
  10011100
```

Integer Multiplication

```
      1100
   x  1101
  ---------
      1100
     0000
    1100
   1100
  ---------
  10011100
```

Input: 2 integer numbers, in *binary*
n is the number of *bits* of each number

What is the running time of this algorithm?

Integer Multiplication

```
     1100
 x   1101
---------
     1100
    0000
   1100
  1100
---------
 10011100
```

Input: 2 integer numbers, in *binary*

n is the number of *bits* of each number

What is the running time of this algorithm?

O(n) time to compute each partial product

n partial products

Time $O(n^2)$

Can we do it faster?

- We will use the Divide-and-Conquer approach.
- We will reduce the problem to solving some instances with $n/2$ bits.
- Then we will use this approach recursively to get a solution for the original problem.

Faster multiplication

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$$

$$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T}(n) \leq \mathbf{4T}(n/2) + \mathbf{c}n$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

T(*n*) ≤ 4T(*n/2*) + c*n*

Solving the recurrence gets us to **T(n) = O($n^{\log 4}$) = O(*n*^2)**

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T(\mathit{n}) \leq 4T(\mathit{n/2}) + c\mathit{n}}$

Solving the recurrence gets us to $\mathbf{T(n) = O(\mathit{n}^{\log 4}) = O(\mathit{n}\hat{\ }2)}$

Generally, solving the recurrence $\mathbf{T(\mathit{n}) \leq \mathit{q}T(\mathit{n/2}) + c\mathit{n}}$ gets us to $\mathbf{T(n) = O(\mathit{n}^{\log \mathit{q}})}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T(}n\mathbf{) \leq 4T(}n/2\mathbf{) + c}n$

Solving the recurrence gets us to $\mathbf{T(n) = O(}n^{\log 4}\mathbf{) = O(}n^{\wedge 2}\mathbf{)}$

Generally, solving the recurrence $\mathbf{T(}n\mathbf{)} \leq q\mathbf{T(}n/2\mathbf{) + c}n$ gets us to $\mathbf{T(n) = O(}n^{\log q}\mathbf{)}$

$(\mathbf{x_1} + \mathbf{x_0})(\mathbf{y_1} + \mathbf{y_0}) = \mathbf{x_1} \cdot \mathbf{y_1} + \mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1} + \mathbf{x_0} \cdot \mathbf{y_0}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T(\mathit{n}) \leq 4T(\mathit{n/2}) + c\mathit{n}}$

Solving the recurrence gets us to $\mathbf{T(n) = O(\mathit{n}^{\log 4}) = O(\mathit{n}^{\wedge 2})}$

Generally, solving the recurrence $\mathbf{T(\mathit{n}) \leq \mathit{q}T(\mathit{n/2}) + c\mathit{n}}$ gets us to $\mathbf{T(n) = O(\mathit{n}^{\log \mathit{q}})}$

$(\mathbf{x_1} + \mathbf{x_0})\,(\mathbf{y_1} + \mathbf{y_0}) = \mathbf{x_1} \cdot \mathbf{y_1} + \mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1} + \mathbf{x_0} \cdot \mathbf{y_0}$

$= \mathbf{p}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{p} - \mathbf{x_1} \cdot \mathbf{y_1} - \mathbf{x_0} \cdot \mathbf{y_0}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T(}n\mathbf{)} \leq \mathbf{4T(}n/2\mathbf{)} + \mathbf{c}n$

Solving the recurrence gets us to $\mathbf{T(n) = O(}n^{\log 4}\mathbf{) = O(}n\text{^}2\mathbf{)}$

Generally, solving the recurrence $\mathbf{T(}n\mathbf{)} \leq q\mathbf{T(}n/2\mathbf{)} + \mathbf{c}n$ gets us to $\mathbf{T(n) = O(}n^{\log q}\mathbf{)}$

$(\mathbf{x_1} + \mathbf{x_0})\,(\mathbf{y_1} + \mathbf{y_0}) = \mathbf{x_1} \cdot \mathbf{y_1} + \mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1} + \mathbf{x_0} \cdot \mathbf{y_0}$

$= \mathbf{p}$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{p} - \mathbf{x_1} \cdot \mathbf{y_1} - \mathbf{x_0} \cdot \mathbf{y_0}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T}(n) \leq \mathbf{4T}(n/2) + \mathbf{c}n$

Solving the recurrence gets us to $\mathbf{T}(n) = \mathbf{O}(n^{\log 4}) = \mathbf{O}(n\text{^}2)$

Generally, solving the recurrence $\mathbf{T}(n) \leq q\mathbf{T}(n/2) + \mathbf{c}n$ gets us to $\mathbf{T}(n) = \mathbf{O}(n^{\log q})$

$(\mathbf{x_1} + \mathbf{x_0})(\mathbf{y_1} + \mathbf{y_0}) = \mathbf{x_1} \cdot \mathbf{y_1} + \mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1} + \mathbf{x_0} \cdot \mathbf{y_0}$

$= \mathbf{p}$

We get the recurrence $\mathbf{T}(n) \leq \mathbf{3T}(n/2) + \mathbf{c}n$

Faster multiplication

$\mathbf{x} = \mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}$

$\mathbf{y} = \mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0}$

$\mathbf{x} \cdot \mathbf{y} = (\mathbf{x_1} \cdot 2^{n/2} + \mathbf{x_0}) \cdot (\mathbf{y_1} \cdot 2^{n/2} + \mathbf{y_0})$

$= \mathbf{x_1} \cdot \mathbf{y_1} \cdot 2^{n/2} + (\mathbf{p} - \mathbf{x_1} \cdot \mathbf{y_1} - \mathbf{x_0} \cdot \mathbf{y_0}) \cdot 2^{n/2} + \mathbf{x_0} \cdot \mathbf{y_0}$

4 multiplications of **n/2** bit numbers

$\mathbf{T}(n) \leq \mathbf{4T}(n/2) + \mathbf{c}n$

Solving the recurrence gets us to $\mathbf{T(n) = O(}n^{\log 4}\mathbf{) = O(}n\text{\^{}}2\mathbf{)}$

Generally, solving the recurrence $\mathbf{T}(n) \leq q\mathbf{T}(n/2) + \mathbf{c}n$ gets us to $\mathbf{T(n) = O(}n^{\log q}\mathbf{)}$

$(\mathbf{x_1} + \mathbf{x_0})(\mathbf{y_1} + \mathbf{y_0}) = \mathbf{x_1} \cdot \mathbf{y_1} + \mathbf{x_1} \cdot \mathbf{y_0} + \mathbf{x_0} \cdot \mathbf{y_1} + \mathbf{x_0} \cdot \mathbf{y_0}$

$= \mathbf{p}$

We get the recurrence $\mathbf{T}(n) \leq \mathbf{3T}(n/2) + \mathbf{c}n$

Faster multiplication

$\mathbf{x} = x_1 \cdot 2^{n/2} + x_0$

$\mathbf{y} = y_1 \cdot 2^{n/2} + y_0$

$\mathbf{x} \cdot \mathbf{y} = (x_1 \cdot 2^{n/2} + x_0) \cdot (y_1 \cdot 2^{n/2} + y_0)$

$= x_1 \cdot y_1 \cdot 2^{n/2} + (p - x_1 \cdot y_1 - x_0 \cdot y_0) \cdot 2^{n/2} + x_0 \cdot y_0$

4 multiplications of **n/2** bit numbers

$\mathbf{T(n) \leq 4T(n/2) + cn}$

Solving the recurrence gets us to $\mathbf{T(n) = O(n^{\log 4}) = O(n\text{^}2)}$

Generally, solving the recurrence $\mathbf{T(n) \leq qT(n/2) + cn}$ gets us to $\mathbf{T(n) = O(n^{\log q})}$

$\mathbf{\log_2 3 = 1.59}$

$(x_1 + x_0)(y_1 + y_0) = x_1 \cdot y_1 + x_1 \cdot y_0 + x_0 \cdot y_1 + x_0 \cdot y_0$

$= p$

We get the recurrence $\mathbf{T(n) \leq 3T(n/2) + cn}$

Faster multiplication

$\mathbf{x} = x_1 \cdot 2^{n/2} + x_0$

$\mathbf{y} = y_1 \cdot 2^{n/2} + y_0$

$\mathbf{x} \cdot \mathbf{y} = (x_1 \cdot 2^{n/2} + x_0) \cdot (y_1 \cdot 2^{n/2} + y_0)$

$= x_1 \cdot y_1 \cdot 2^{n/2} + (p - x_1 \cdot y_1 - x_0 \cdot y_0) \cdot 2^{n/2} + x_0 \cdot y_0$

4 multiplications of **n/2** bit numbers

$\mathbf{T(n) \leq 4T(n/2) + cn}$

Solving the recurrence gets us to $\mathbf{T(n) = O(n^{\log 4}) = O(n\textasciicircum 2)}$

Generally, solving the recurrence $\mathbf{T(n) \leq qT(n/2) + cn}$ gets us to $\mathbf{T(n) = O(n^{\log q})}$

$\mathbf{\log_2 3 = 1.59}$

$\mathbf{T(n) = O(n^{1.59})}$

$(x_1 + x_0)(y_1 + y_0) = x_1 \cdot y_1 + x_1 \cdot y_0 + x_0 \cdot y_1 + x_0 \cdot y_0$

$= p$

We get the recurrence $\mathbf{T(n) \leq 3T(n/2) + cn}$