Divide and Conquer

Closest Pair

Closest Pair

- **Input:** A set n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
- **Output:** A pair of distinct points whose distance is smallest.

Straight-forward Idea

- **Input:** A set n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
- **Output:** A pair of distinct points whose distance is smallest.
- Plan 1: Brute-force
  - Compute all $\frac{n(n-1)}{2}$ pairs.
  - Output the smallest one.

Straight-forward Idea

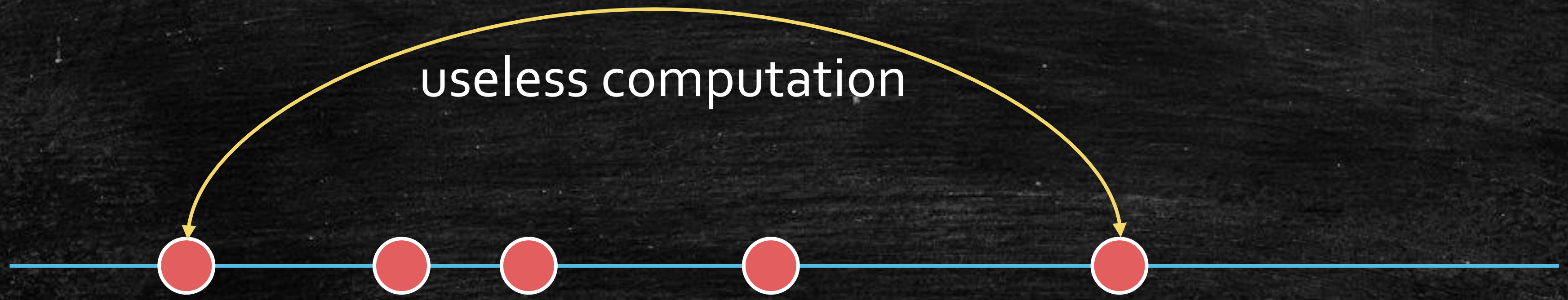
- **Input:** A set n points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$.
- **Output:** A pair of distinct points whose distance is smallest.
- Plan 1: Brute-force
  - Compute all $\frac{n(n-1)}{2}$ pairs.
  - Output the smallest one.
  - $\boldsymbol{O(n^2)}$

Can we do better?

- Improve it by **sorting**
- Avoid some useless computation

Can we do better?

- Improve it by **sorting**
- Avoid some useless computation
- Special case: all points are on the **same line**.

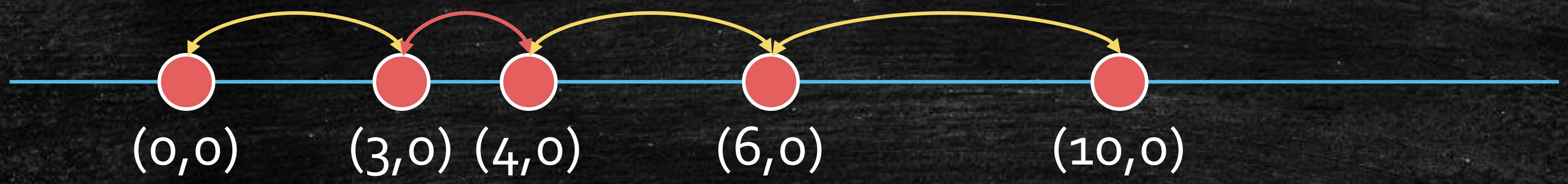


Can we do better?

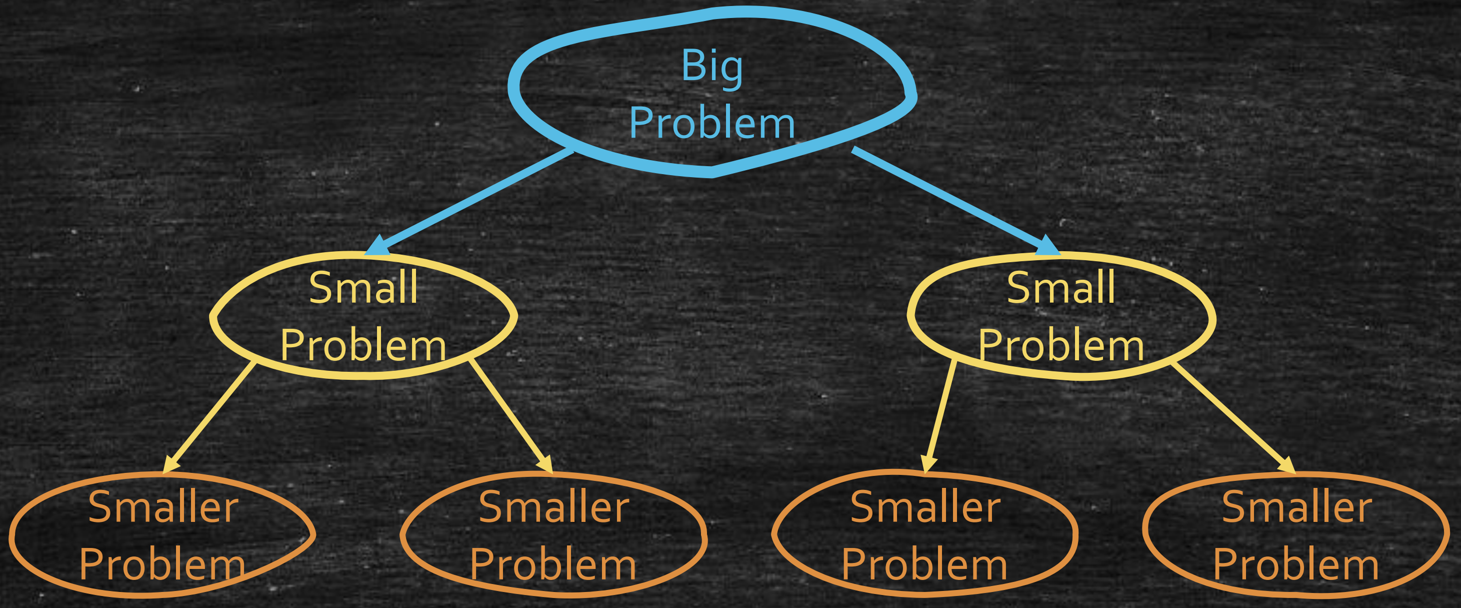
- Special case: all points are on the **same line**.
- Plan 2: Sorting
- Sort the points (by the x-coordinate)
  - (6,0)
  - (3,0)
  - (0,0)
  - (10,0)
  - (4,0)

Can we do better?

- Special case: all points are on the same line.
- Plan 2: Sorting
- Sort the points (by the x-coordinate) $\boxed{O(n \log n)}$
  - Only compute the distance of adjacent point pair. $\boxed{O(n)}$
  - Output the closest pair.



How to extend this Idea to general case?



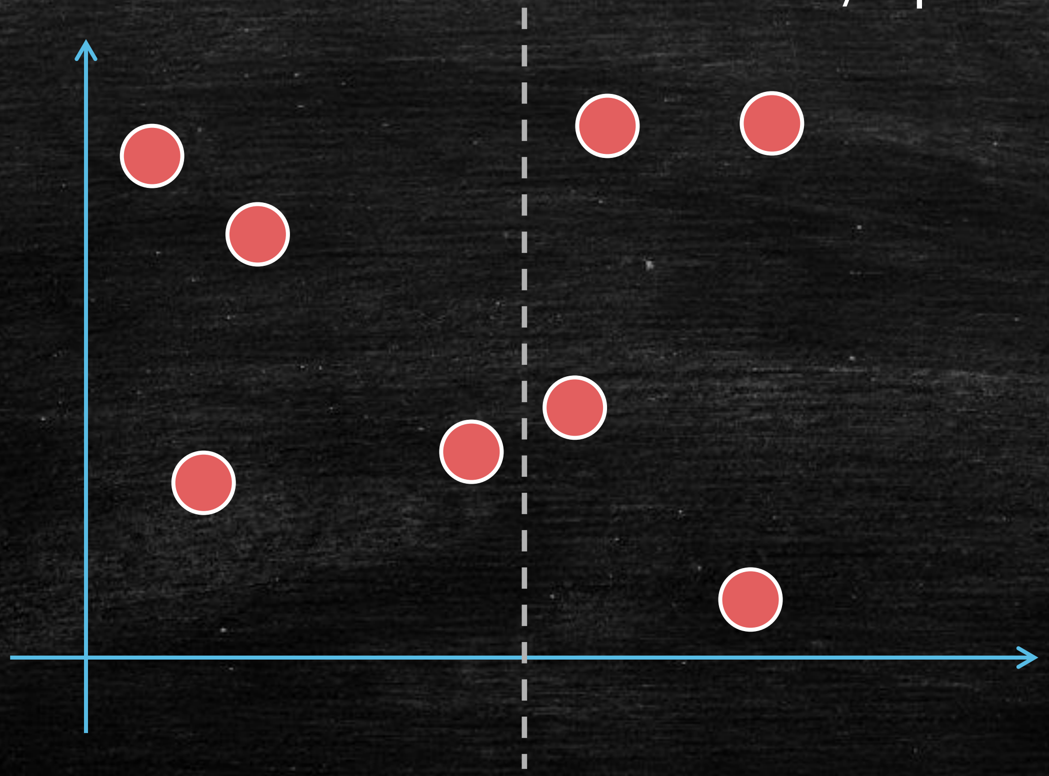
Ok! Let’s move to divide and conquer!

Divide and Conquer

- **Input:** A set n points $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$.
- **Output:** A pair of distinct points whose distance is smallest.
- Plan 3: Divide and Conquer
  - Divide:
    - Sort the points (by the x-coordinate)
      - Assume all x-coordinate are different.
    - Points are sorted by the x-coordinate.
    - By a vertical line so that each side has $n/2$ points

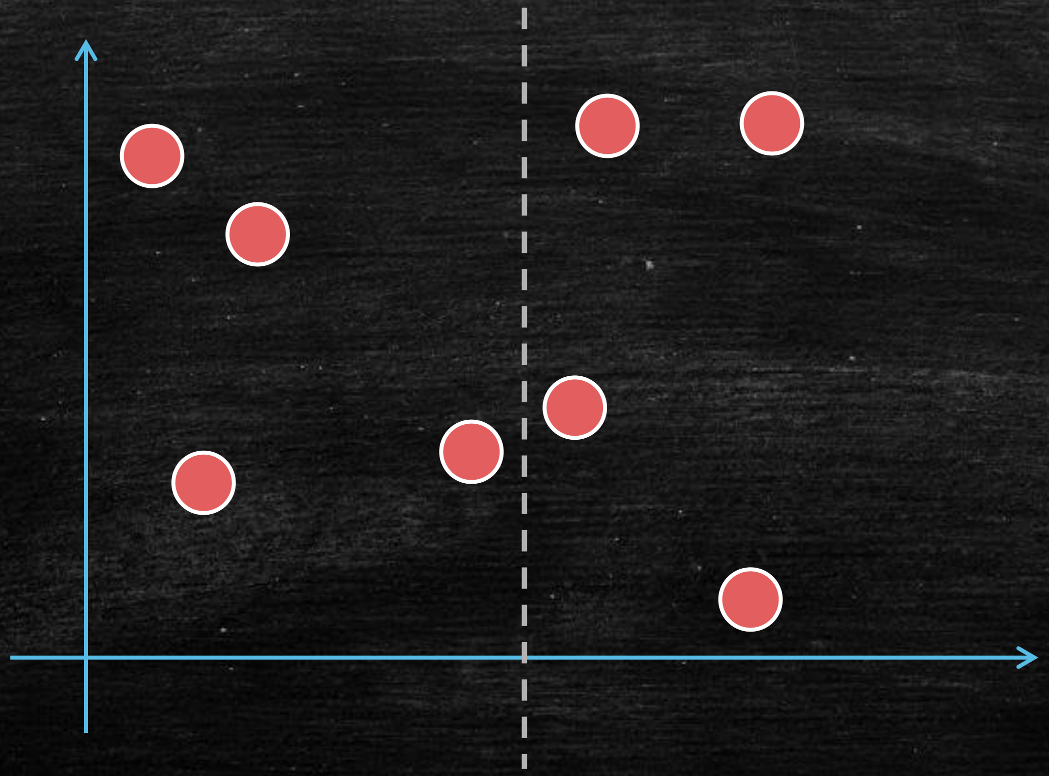
Divide

- Divide:
  - Sort the points (by the x-coordinate)
  - Draw a vertical line so that each side has $n/2$ points.



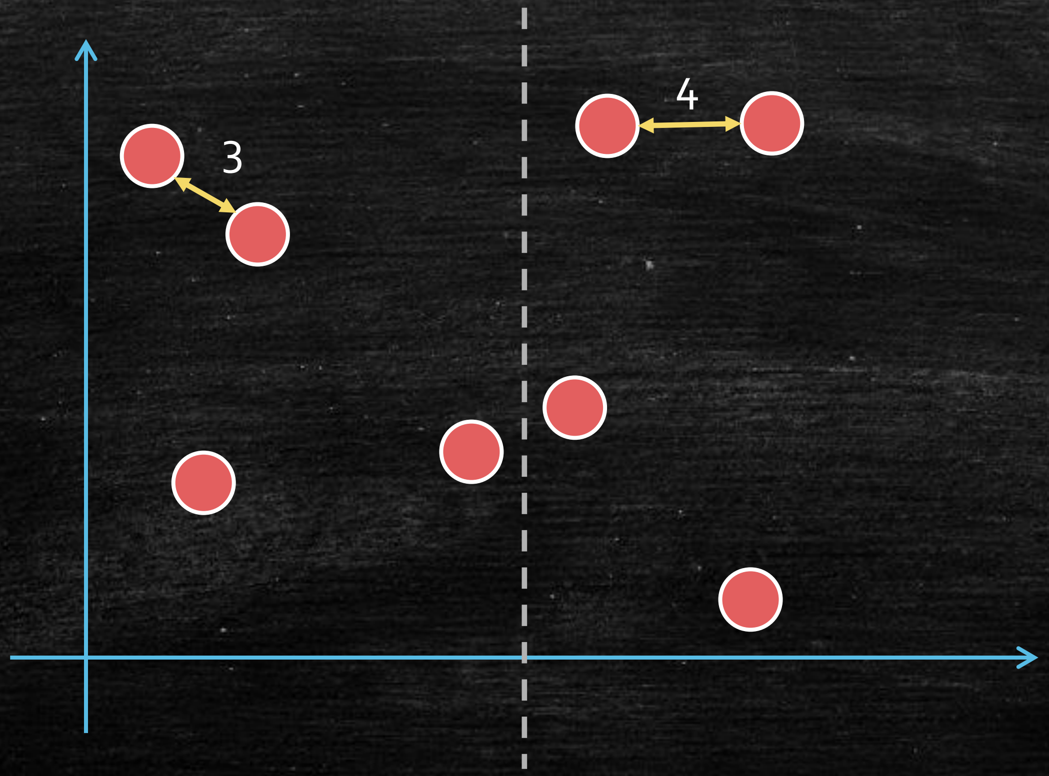
Recurse

- Recurse
  - Find the closest pair in each side.



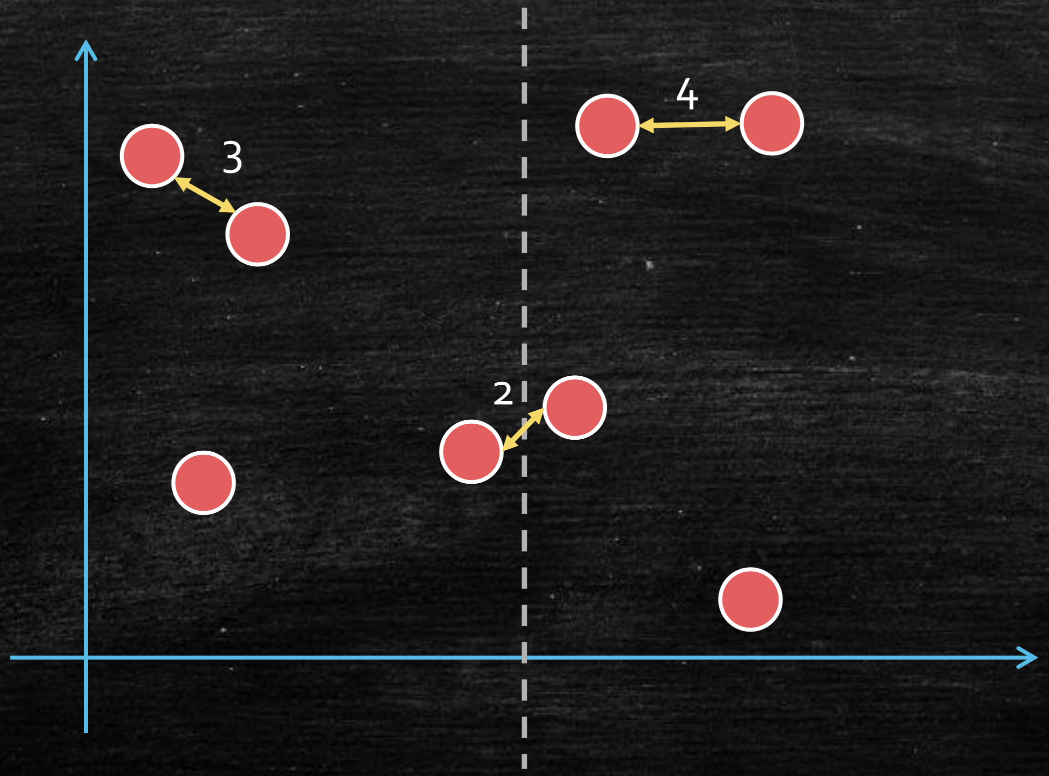
Recurse

- Recurse
  - Find the closest pair in each side.



Recurse

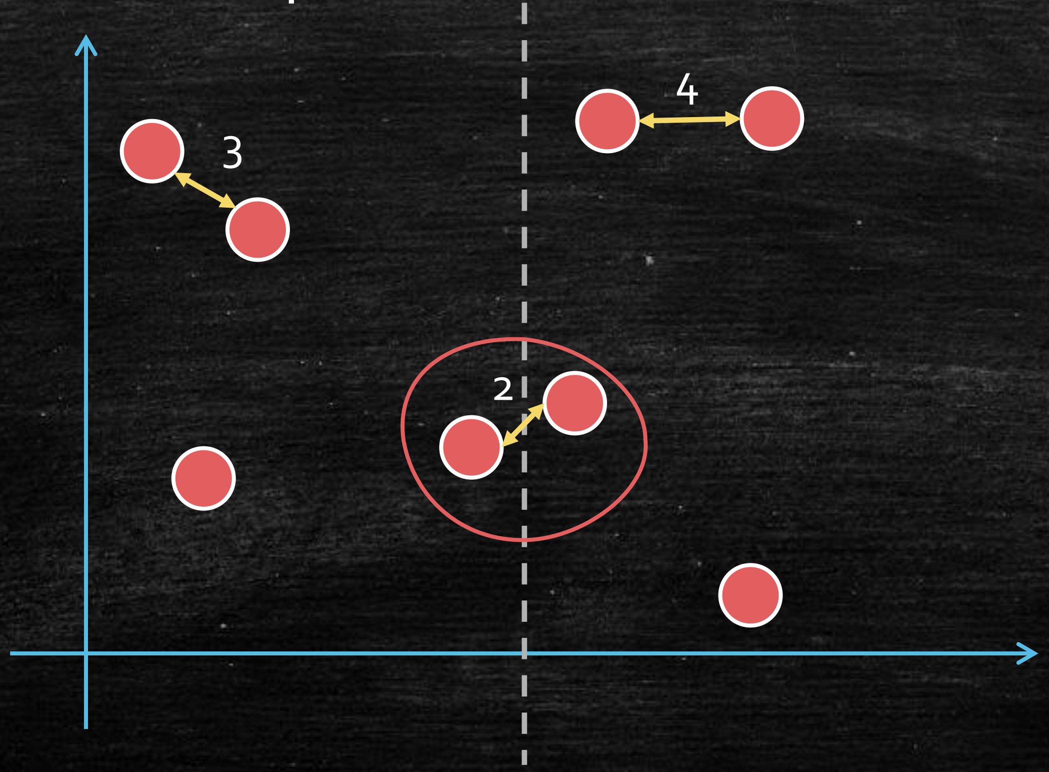
- Combine
  - Find the closet pair between two sides.



Recurse

- Combine
  - Find the closet pair between two sides.
  - Output the min of 3 pairs.

How long it takes?

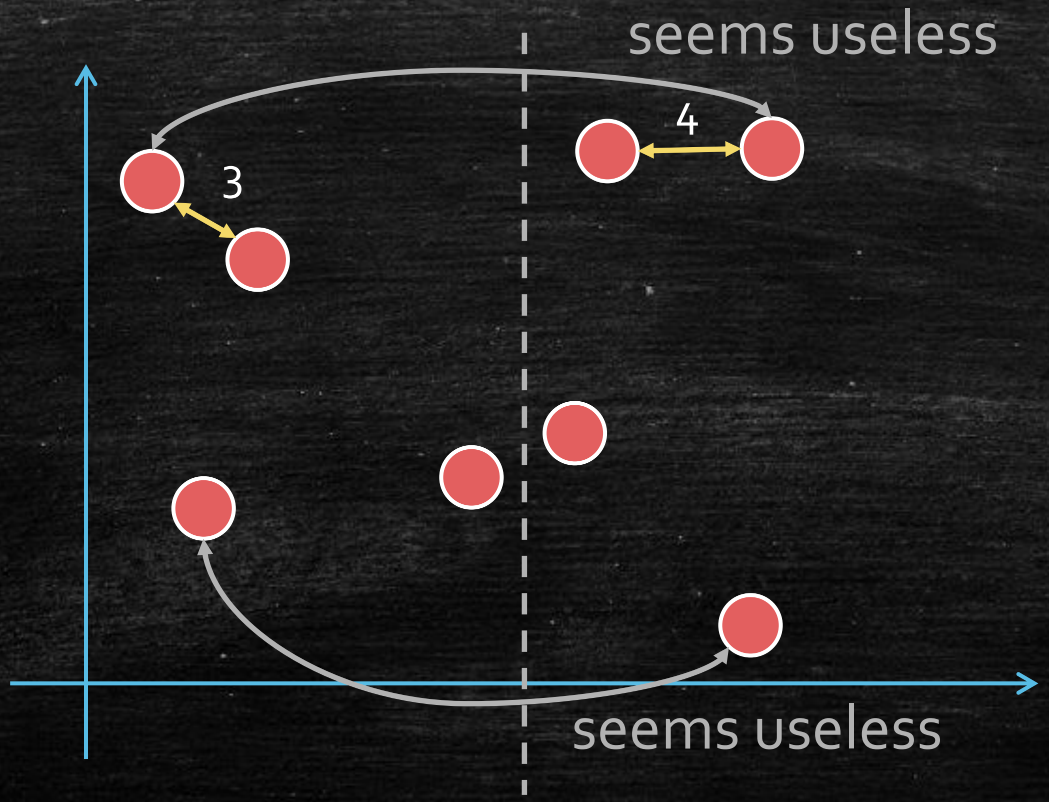


Closet pair between two sides

- Straight-forward?
  - Compute all $\left(\frac{n}{2}\right)^2$ pairs, with one point on each side.
  - Return the closest one.
- What about the running time?
  - Divide: $O(n \log n)$
    - Points are sorted by the x-coordinate.
    - By a vertical line so that each side has $n/2$ points
  - Recurse: $2\mathrm{T}(\frac{n}{2})$
    - Find the closest pair in each side.
  - Combine: $O(n^2)$
  - Overall: $\mathbf{T(n)} = \boldsymbol{O}(\boldsymbol{n}^2) + \boldsymbol{2T}\left(\frac{\boldsymbol{n}}{\boldsymbol{2}}\right) = \boldsymbol{O}(\boldsymbol{n}^2)$ Master Theorem

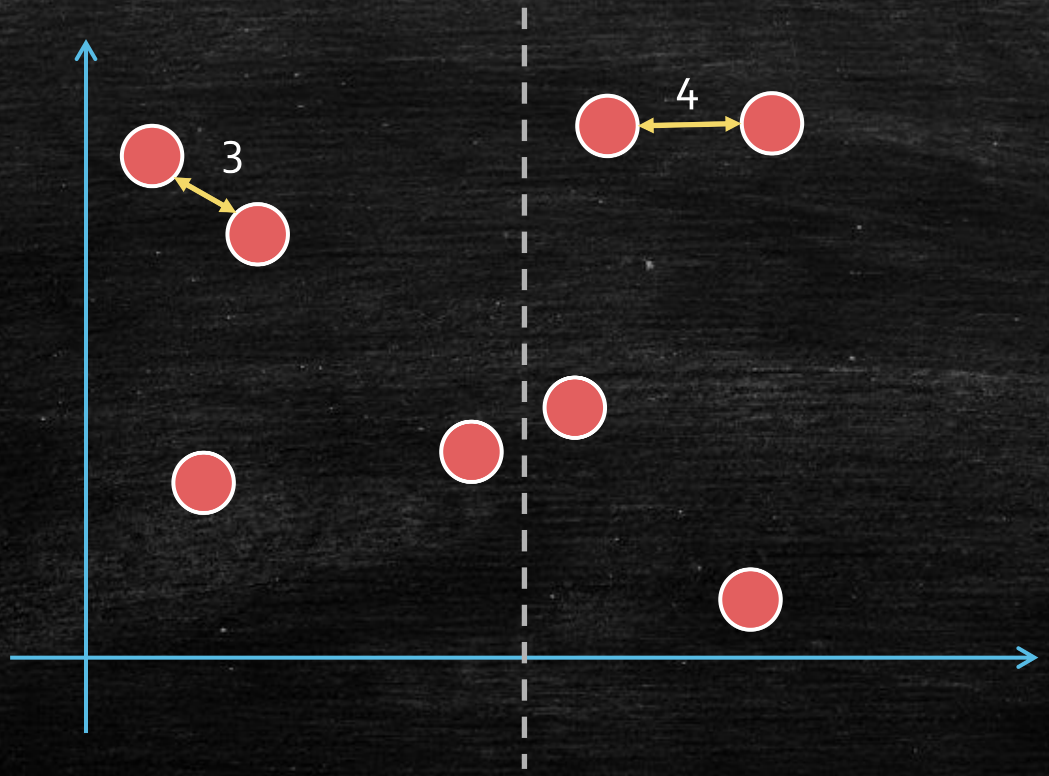
Closet pair between two sides

- Key idea
  - We need not compute all pairs



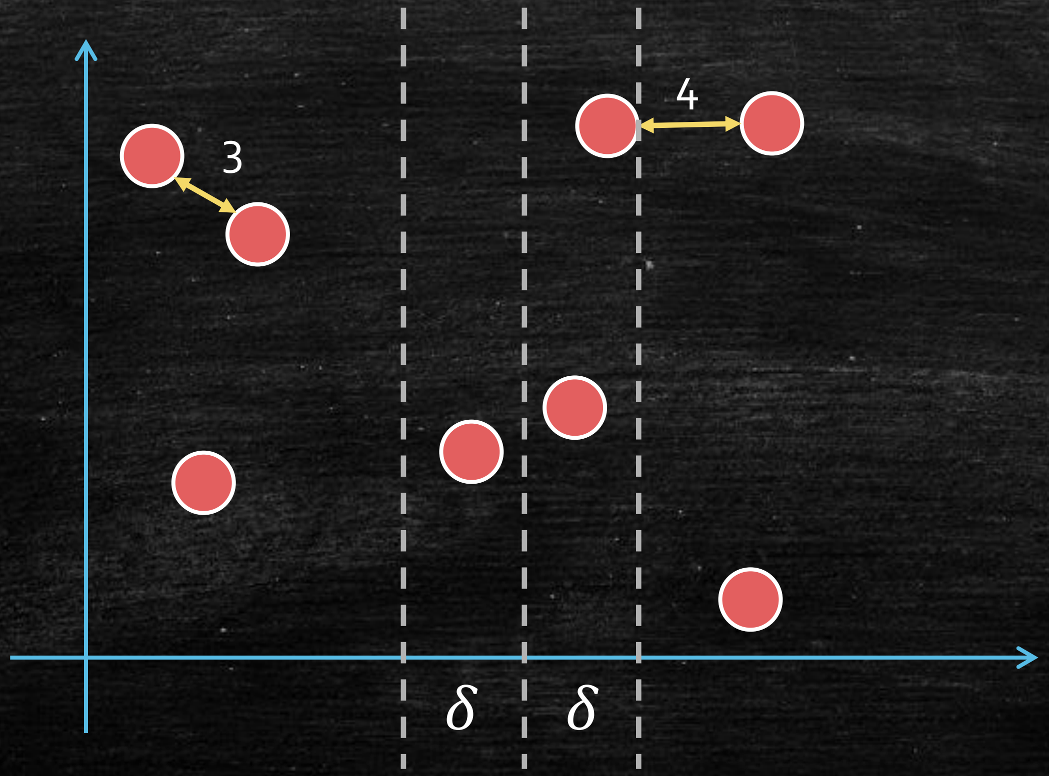
Closet pair between two sides

- δ_L, δ_R: smallest distance on left and right
- δ: $\min\{\delta_L, \delta_R\}$ (e.g., $\delta = 3, \delta_L = 3, \delta_R = 4$)



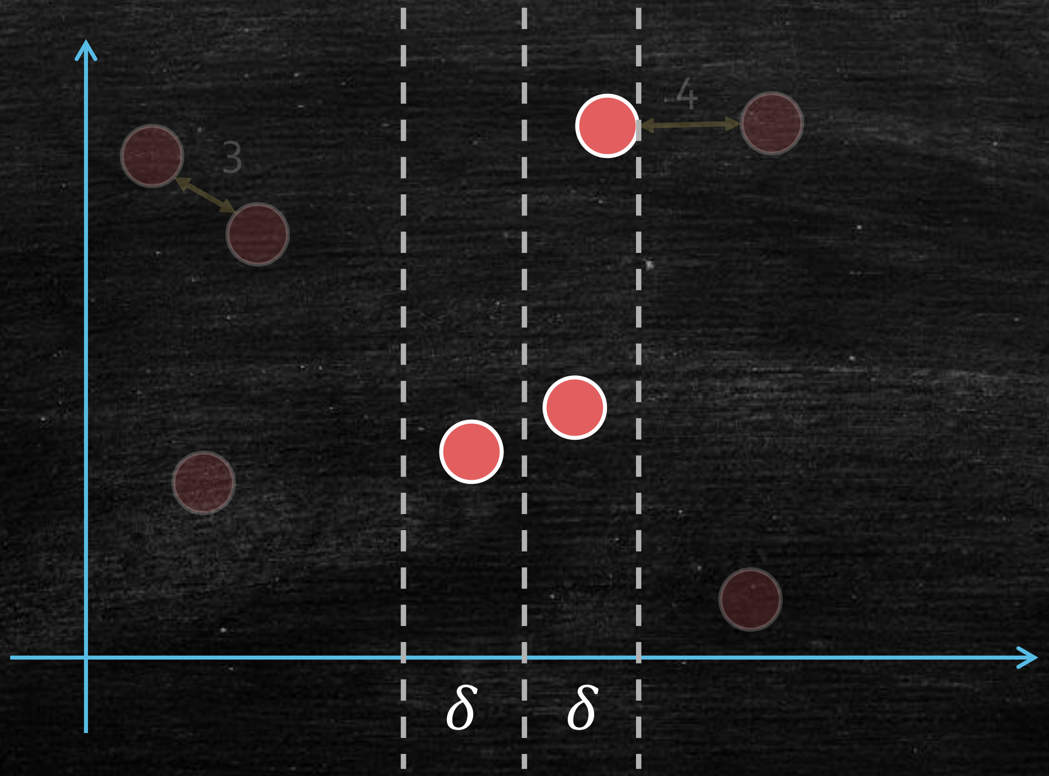
Closet pair between two sides

- Draw two lines, with δ apart from the middle line.



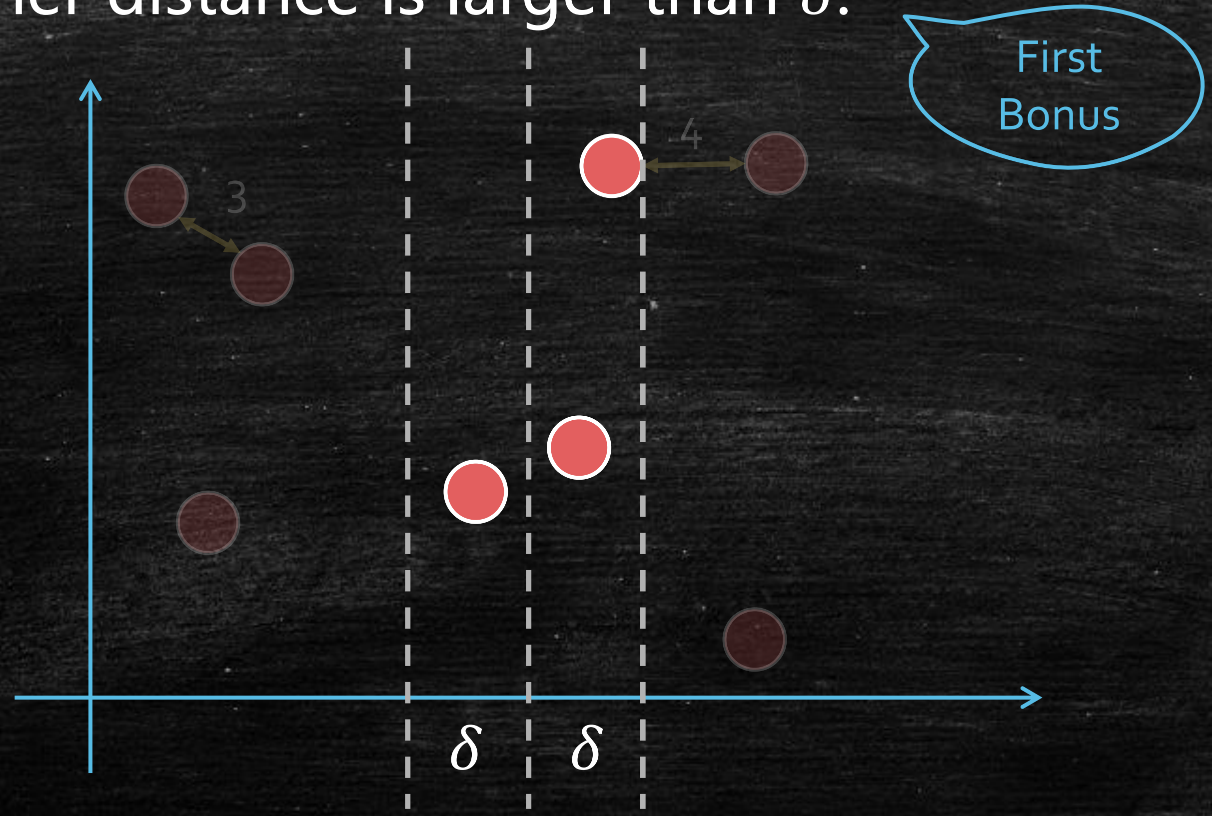
Closet pair between two sides

- Draw two lines, with δ apart from the middle line.
- Only focus on the points **inside** the two lines.



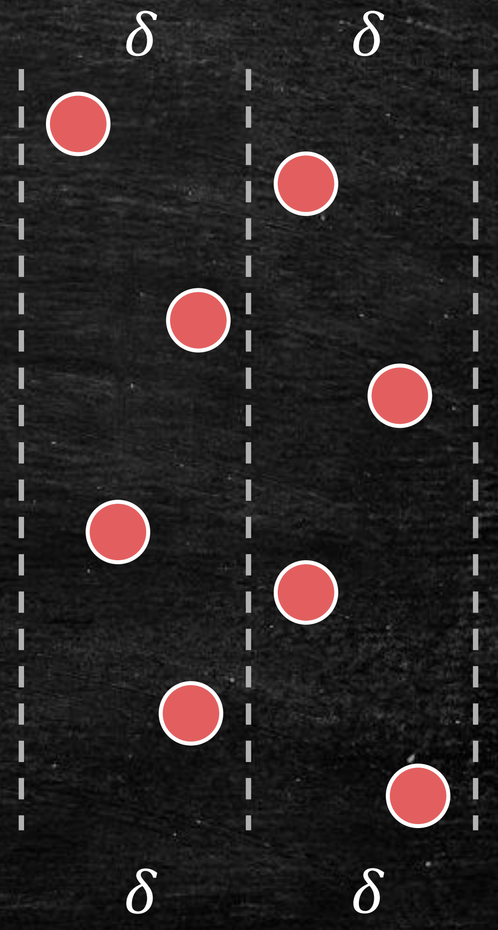
Closet pair between two sides

- Only focus on the points **inside** the two lines.
- All the other distance is larger than δ.



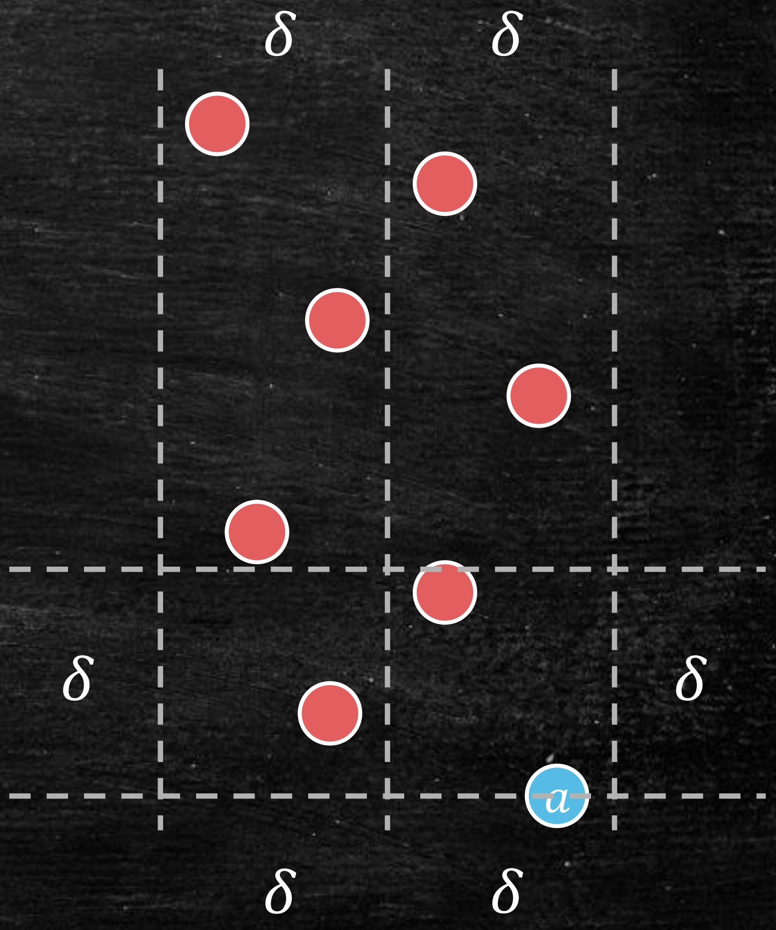
Closest pair in the 2δ-strip

- Brute-force
  - Compute all pairs inside the 2δ-strip.
  - $O(m^2)$: number of points inside
  - Can we bound m?
  - **No:** m can be equal to n!



How to improve?

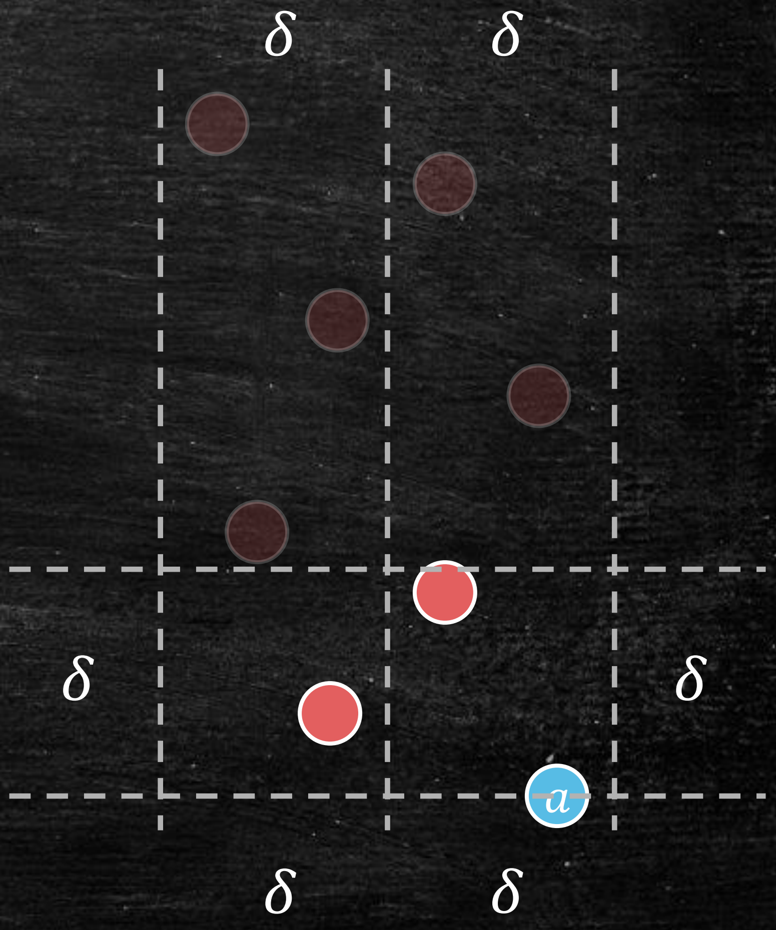
- Fix a point a
- Focus on pair (a, b)
  - b is above a.
- What kind of pairs is **impossible** to be the closest one?



How to improve?

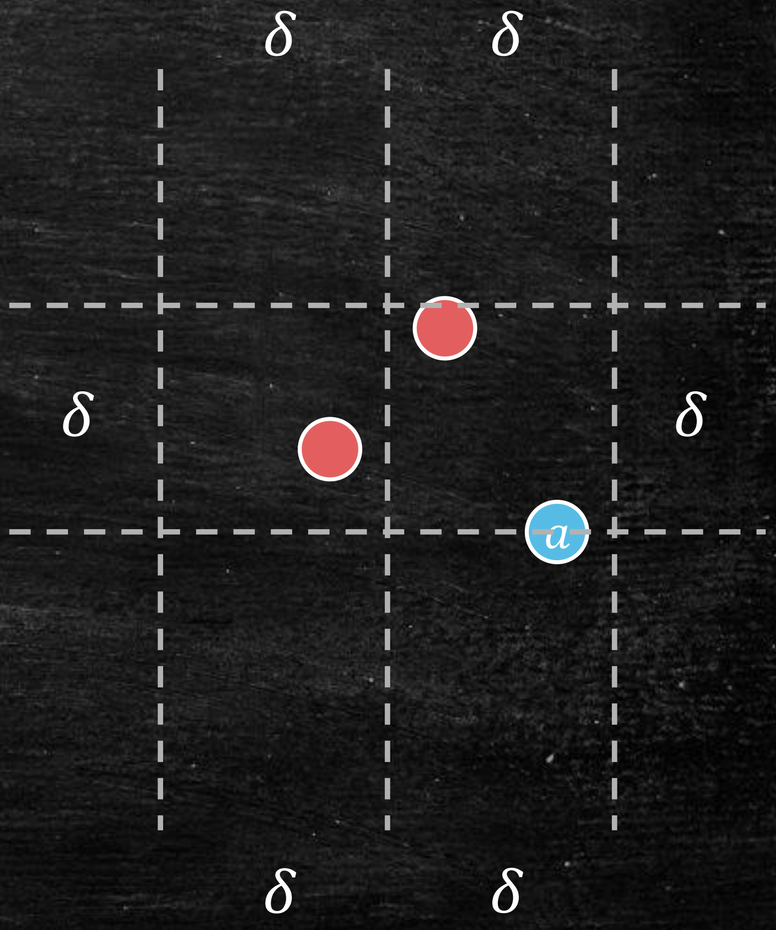
- Fix a point a
- Focus on pair (a, b)
  - b is above a.
- What kind of pairs is **impossible** to be the closest one?
  - b is outside the $\mathbf{2\delta \times \delta}$**-rectangle**.
- Focus on the $\mathbf{2\delta \times \delta}$**-rectangle**

Second Bonus



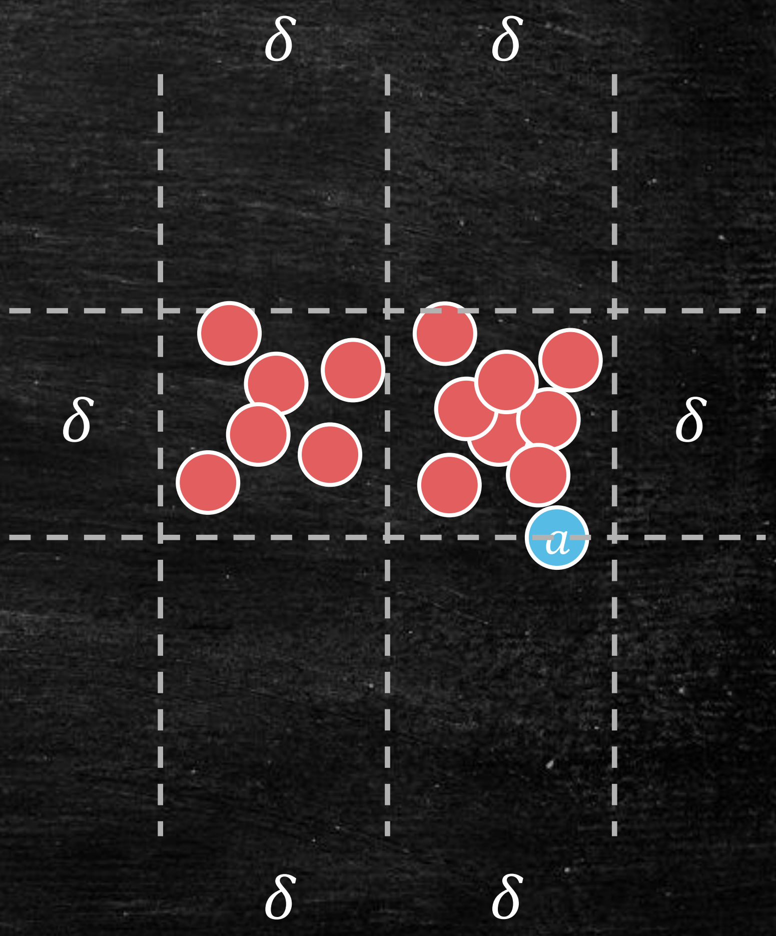
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the **$2\delta \times \delta$-rectangle**



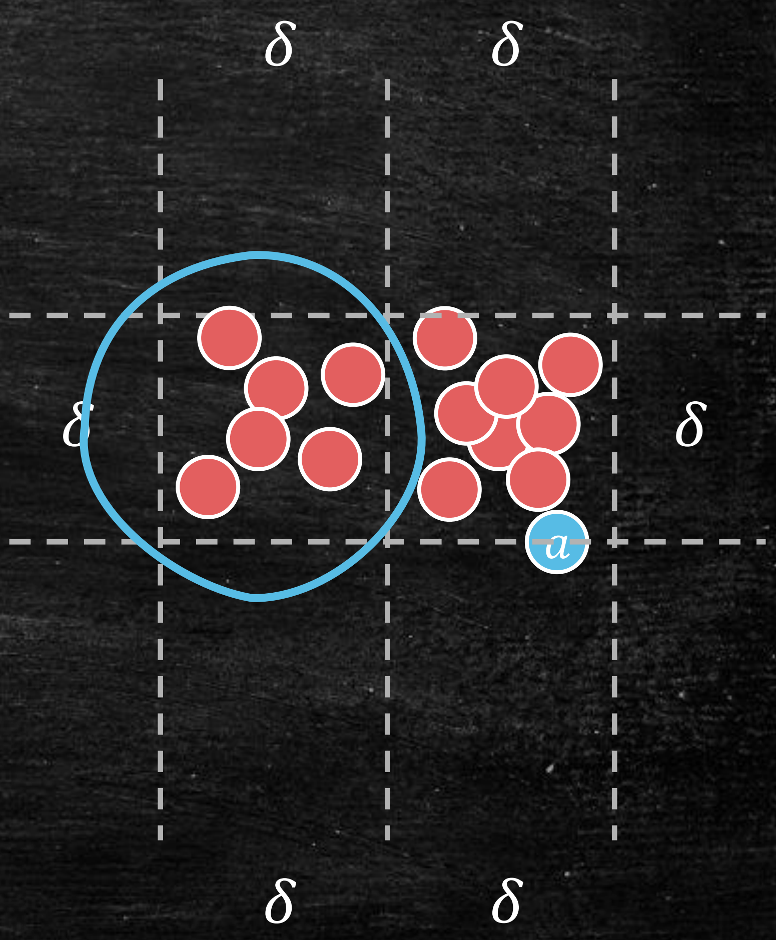
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the **$2\delta \times \delta$-rectangle**



Is that enough now?

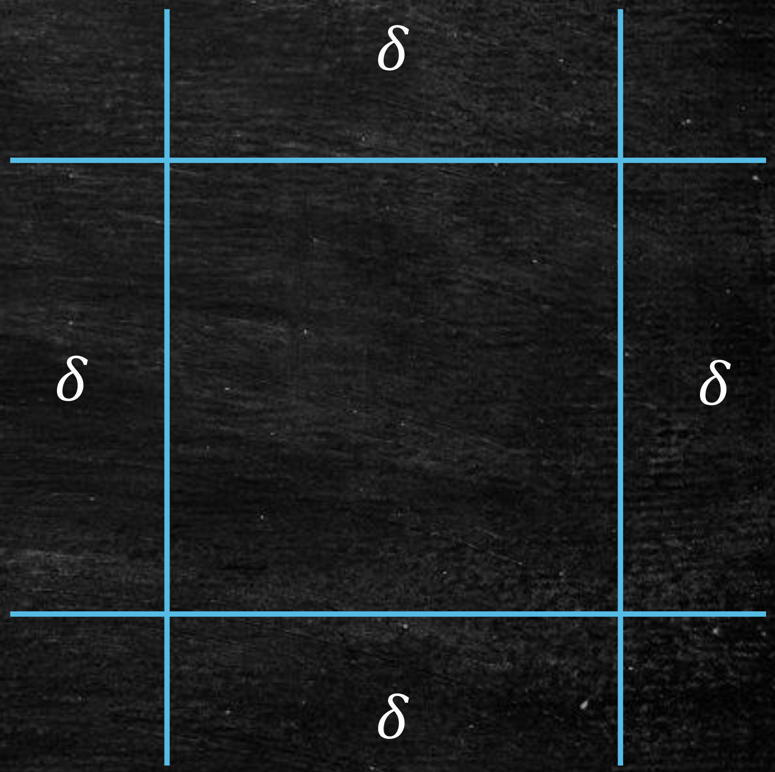
- Why the first bonus is not enough?
  – We can not **bound** the number of points!
- Can we **bound** it now?
  – inside the **$2\delta \times \delta$-rectangle**
- Focus on a $\delta \times \delta$-square



Points inside a $\delta \times \delta$-square

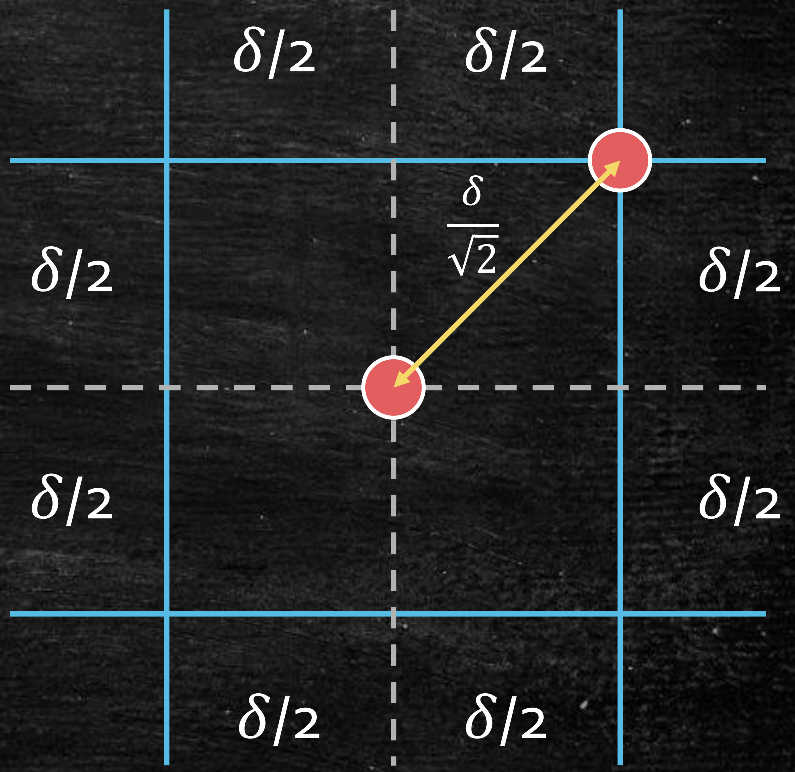
- How many points can **at most** appear in the square?
- Tips: distance at least δ
  - $\delta = \min(\delta_L, \delta_R)$

Discussion



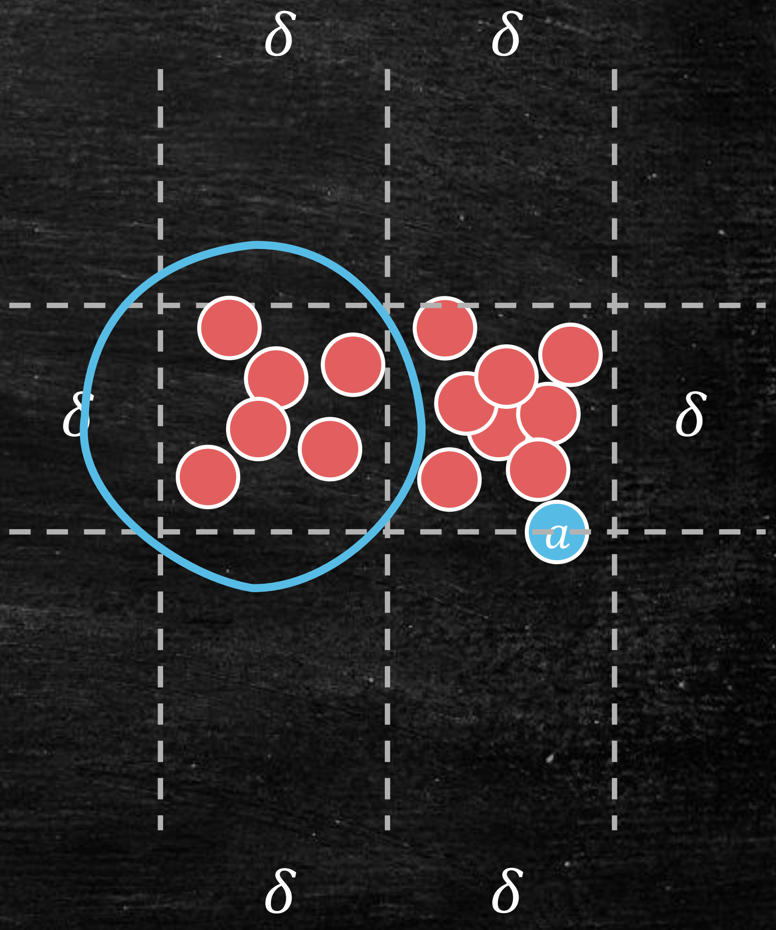
Points inside a $\delta \times \delta$-square

- How many points can **at most** appear in the square?
- Tips: distance at least δ
  - $\delta = \min(\delta_L, \delta_R)$
- Divide into four **sub-square**
  - How many point can **at most** appear in the **sub-square**?
    - Two points are at most $\frac{\delta}{\sqrt{2}} < \delta$ apart.
    - At most **one** point!
- At most **Four** points in the square!



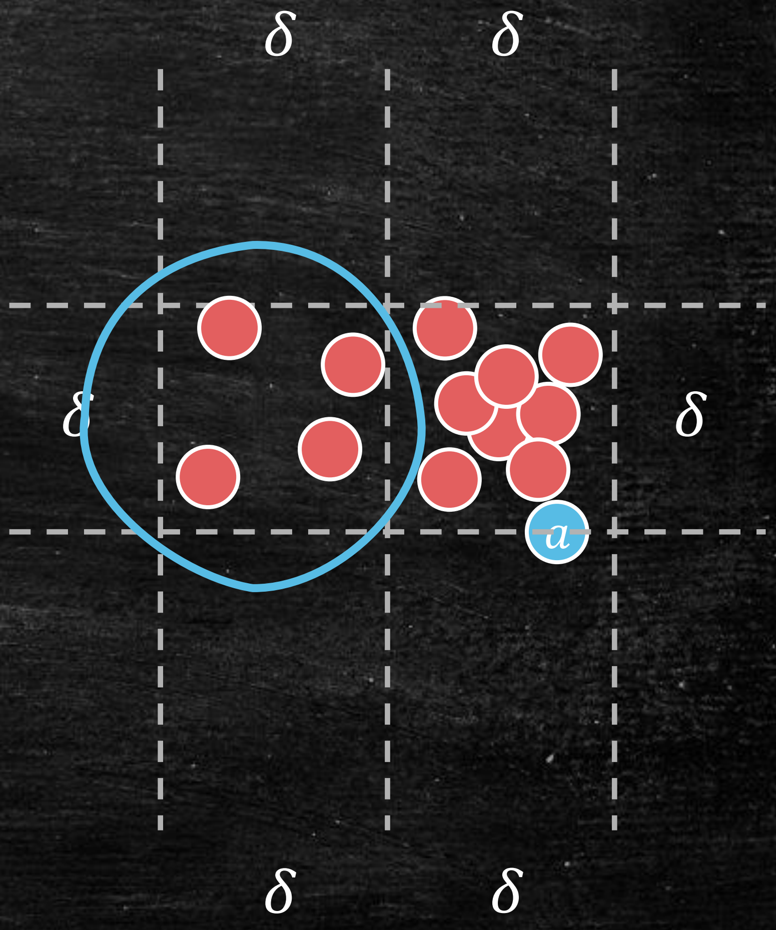
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the **$2\delta \times \delta$-rectangle**
- Focus on a $\delta \times \delta$-square



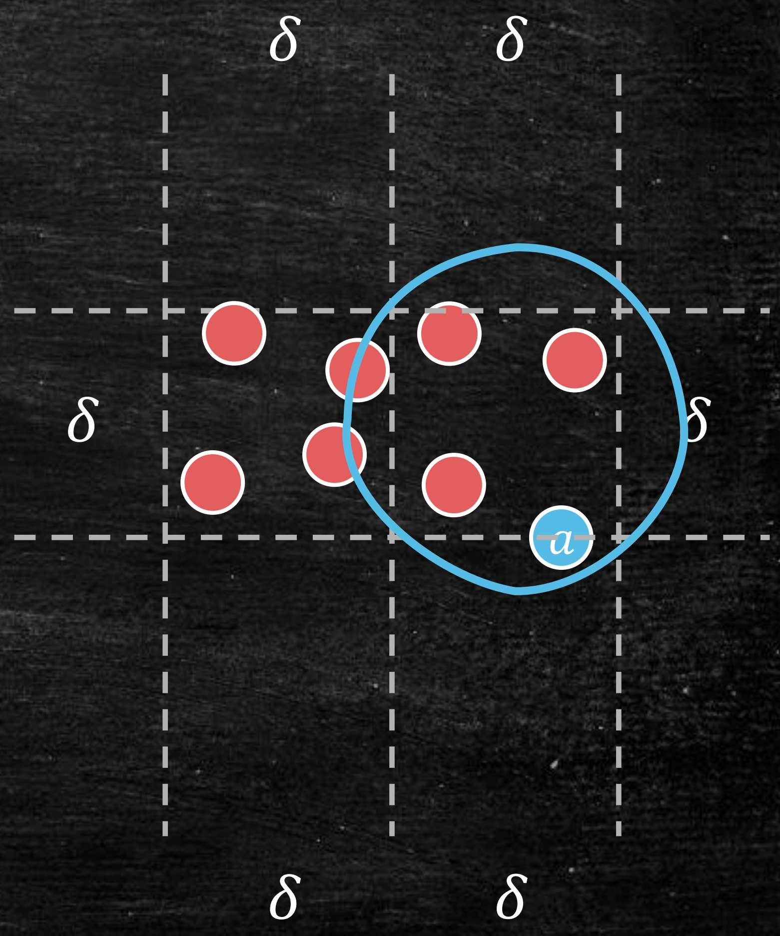
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the **$2\delta \times \delta$-rectangle**
- Focus on a $\delta \times \delta$-square
  - 4 points on the left



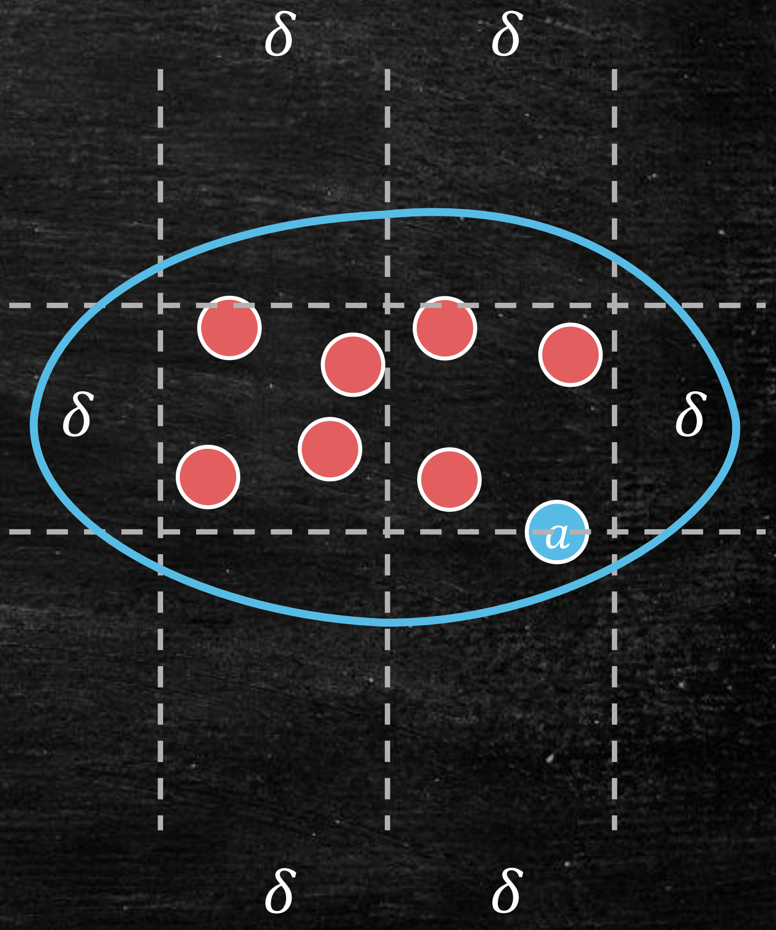
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the $2\delta \times \delta$**-rectangle**
- Focus on a $\delta \times \delta$-square
  - 4 points on the left
  - 4 points on the right (including a)



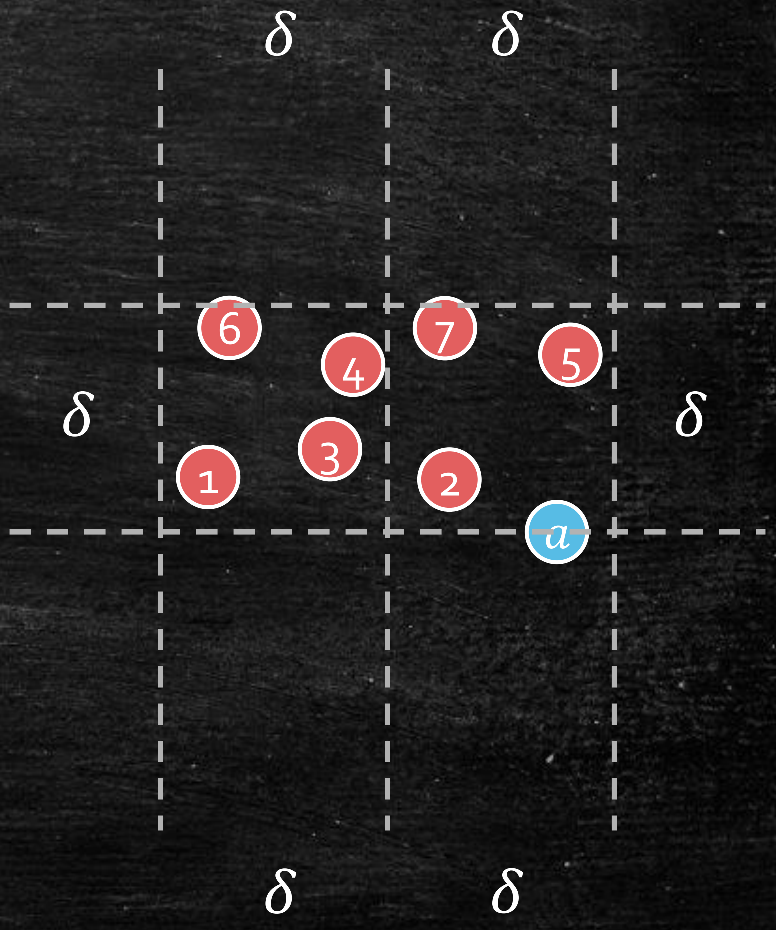
Is that enough now?

- Why the first bonus is not enough?
  - We can not **bound** the number of points!
- Can we **bound** it now?
  - inside the **$2\delta \times \delta$-rectangle**
- Focus on a $\delta \times \delta$-square
  - 4 points on the left
  - 4 points on the right (including a)
  - 8 points totally (including a)



Closest pair in the 2δ-strip

- Brute-force
  - Compute all pairs inside the 2δ-strip.
  - $O(m^2)$: number of points inside
  - Can we bound m?
  - **No:** m can be equal to n!
- Improved way
  - Focus on point a
  - Focus on pair (a, b)
    - b is above a.
  - We only need to compute **Seven** b above a.



Divide and Conquer Algorithm

Function ClosestPair(S)

- Divide:
  1. Sort the points (by the x-coordinate).
  2. Draw such a **vertical line** ℓ that each side has $n/2$ points.
- Recurse
  3. Find the closest pair in each side, let δ_L, δ_R be the distance.
- Combine
  4. Let $\delta = \min\{\delta_L, \delta_R\}$ and S' be the set of points at most δ from ℓ.
  5. Sort S' by the y-coordinate.
  6. For each $a \in S'$, check 7 b above a inside S', find the closest pair.
  7. Return the closest pair among step 3 and 6.

Running time

Function ClosestPair(S)

- Divide:
  1. Sort the points (by the x-coordinate).
  2. Draw such a **vertical line** ℓ that each side has $n/2$ points.
- Recurse
  3. Find the closest pair in each side, let δ_L, δ_R be the distance.
- Combine
  4. Let $\delta = \min\{\delta_L, \delta_R\}$ and S' be the set of points at most δ from ℓ.
  5. Sort S' by the y-coordinate.
  6. For each $a \in S'$, check 7 b above a inside S', find the closest pair.
  7. Return the closest pair among step 3 and 6.

Divide: $\boldsymbol{O(n \log n)}$

Recurse: $\boldsymbol{2T(\frac{n}{2})}$

Recurse: $\boldsymbol{O(n \log n)}$

Analysis

- $T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$
- Recall Master Theorem
  - $T(n) = O(n \log n)$ if $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
- Claim: $T(n) = O(n \log^2 n)$
  - We **can not** directly apply Master Theorem.
  - Prove it by induction!
  - Prove it by keep expending $T(n)$!

Improve more

- Can we improve divide and combine to $O(n)$?
  - If we success, then $T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log n)$
- Tips
  - Do we really need sorting every time?
  - What happens if do sorting before divide and conquer?
- Even more
  - A randomized algorithm achieves $O(n)$.
    - **Samir Khuller and Yossi Matias** (1995).
    - A simple randomized sieve algorithm for the closest-pair problem.

Today's goal

- Learn the closest pair algorithm
- Learn why we have the magical number **7** **analytically**
  – 7 is not important, can you tell why it can be bounded?
- Learn to analyze the running time without Master Theorem