12. Convex Hull and Closest Pair

CPSC 535 ~ Fall 2019

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY
FULLERTON

December 2, 2020

Closest Pair Problem

closest pair problem
input: set of $n \geq 2$ points Q
output: two points $p, q \in Q$ minimizing $d(p, q)$

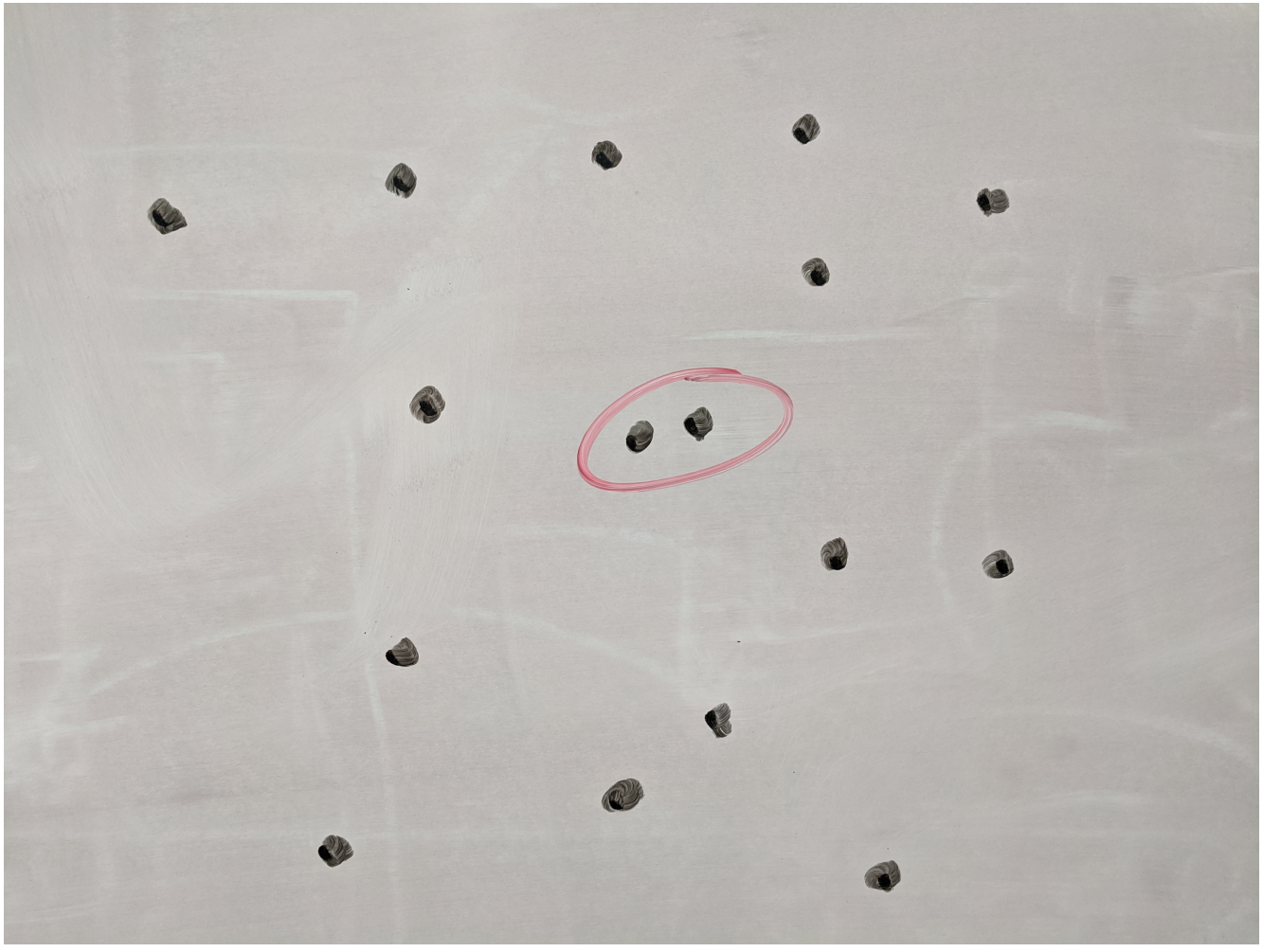
$d(p, q)$ is standard Euclidean distance

$$d((x_p, y_p), (x_q, y_q)) \equiv \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

Applications

- find two objects at greatest risk of collision
- determine numerical precision needed for points
- match predicted user preference to products
- match players for fair contest

Example



Baseline Algorithm

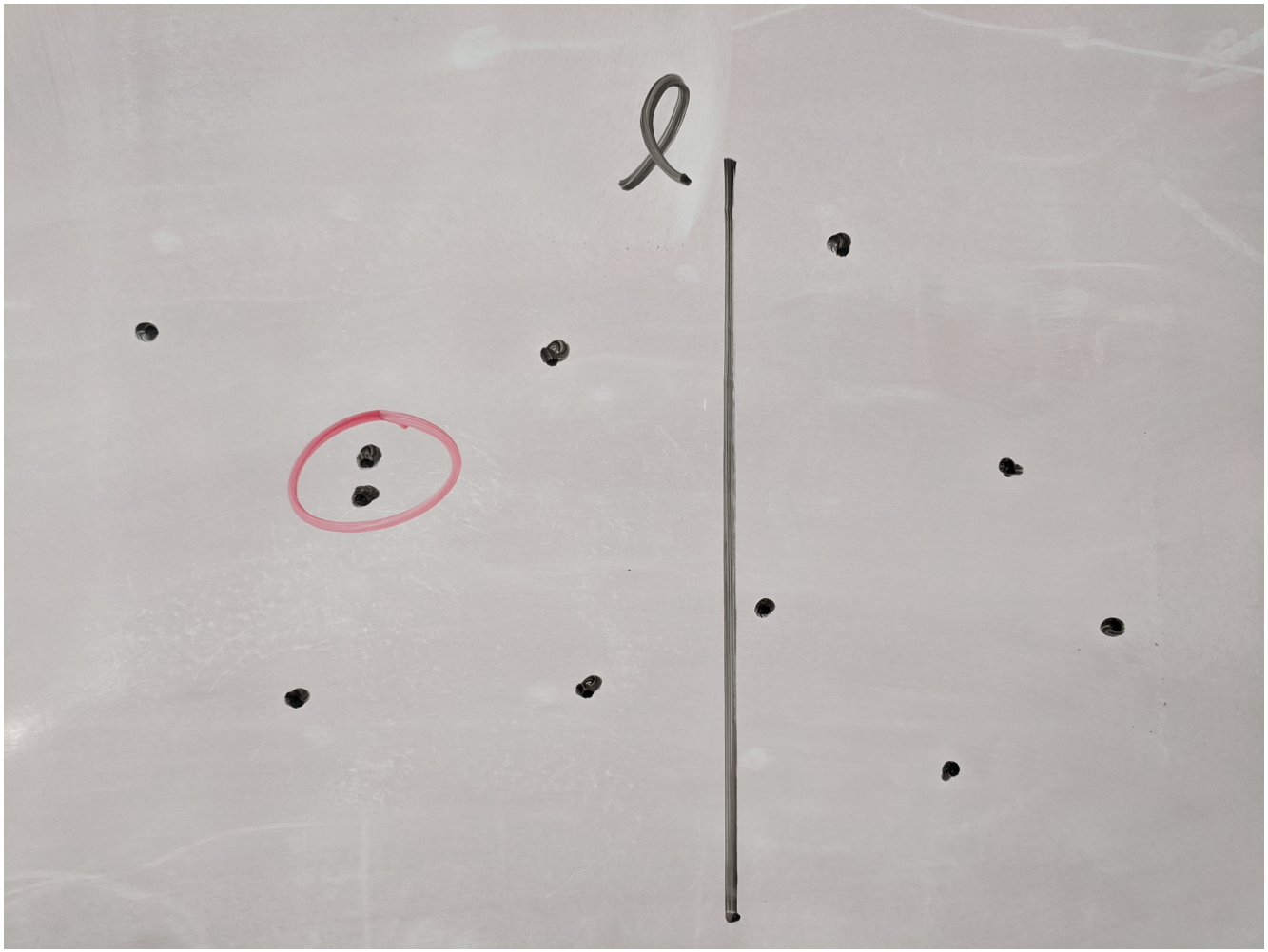
```
1:  function CLOSEST-PAIR-NAIVE(Q)          ▷ guaranteed |Q| ≥ 2
2:      p = q = NIL
3:      δ = ∞
4:      for distinct a, b ∈ Q do
5:          δ_ab = d(a, b)
6:          if δ_ab < δ then
7:              p = a, q = b, δ = δ_ab
8:          end if
9:      end for
10:     return p, q
11: end function
```

Analysis: $\Theta(n^2)$

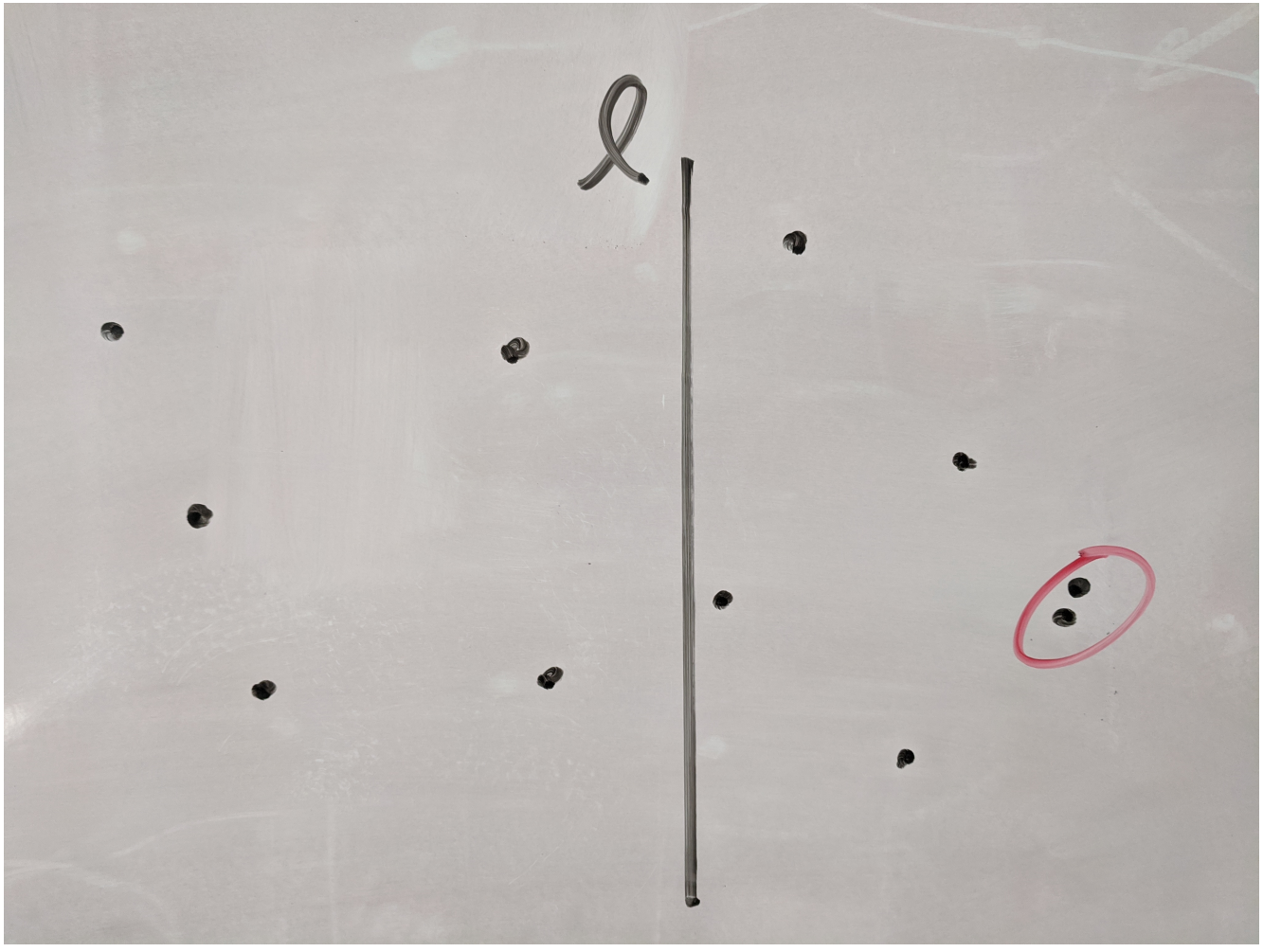
Divide-and-Conquer First Draft

- base case: $n \leq 3$, use baseline algorithm
- else draw vertical line ℓ dividing Q into halves L, R
- recursively find closest pairs p_L, q_L and p_R, q_R
- solution is one of
  - (from the left) p_L, q_L
  - (from the right) p_R, q_R
  - (straddling the boundary) some $p \in L$ and $q \in R$ even closer than $d(p_L, q_L)$ and $d(p_R, q_R)$
- naïve search for straddling case is $\Theta(n^2) \implies$ need to be more clever to speed up
- clever = use geometry

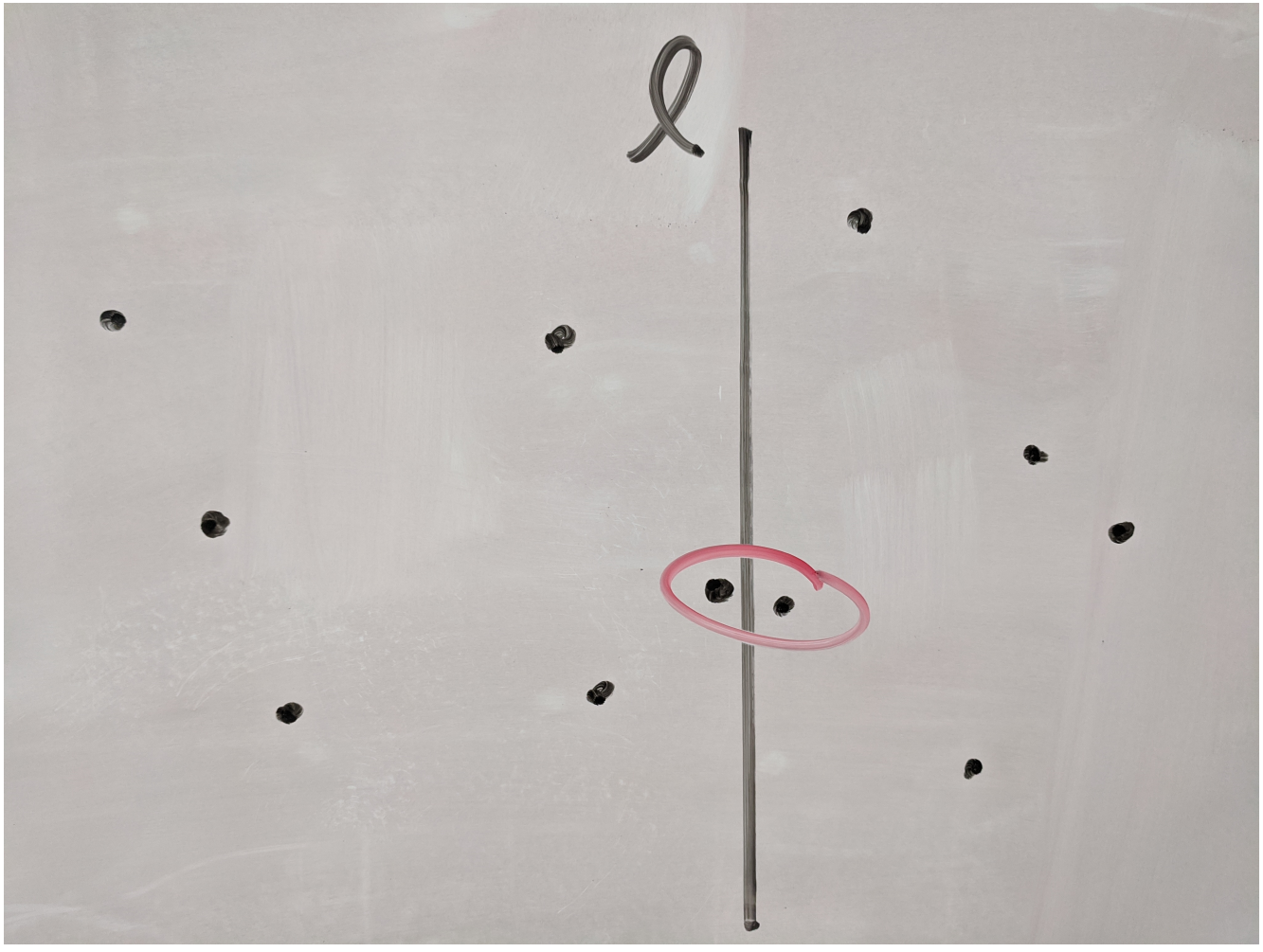
Divide-and-Conquer: Closest Pair on the Left



Divide-and-Conquer: Closest Pair on the Right



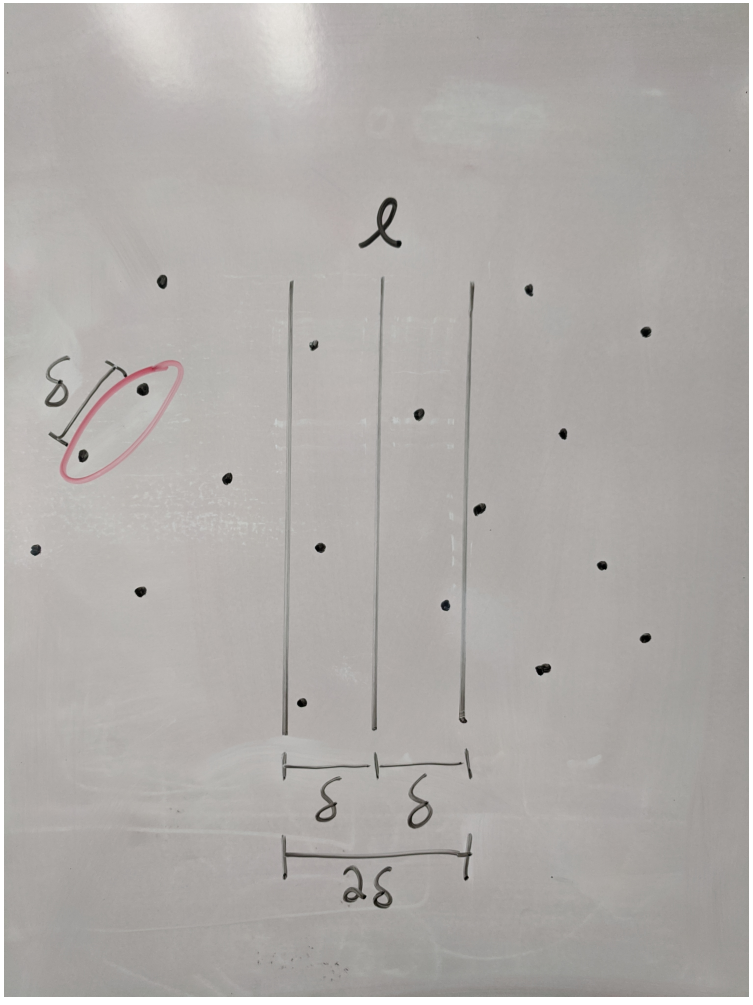
Divide-and-Conquer: Closest Pair on the Boundary



Narrowing Search at Boundary

- **Claim**: only need to check $O(n)$ pairs of straddling points, not $\Theta(n^2)$
- let $\delta = \min(d(p_L, q_L), d(p_R, q_R))$ = distance between closest pair entirely in L or entirely in R
- suppose $\exists p_S$ left of ℓ, q_S right of ℓ, with p_S, q_S closer than δ
- such p_S, q_S must reside in a $2\delta \times \delta$ rectangle centered on ℓ

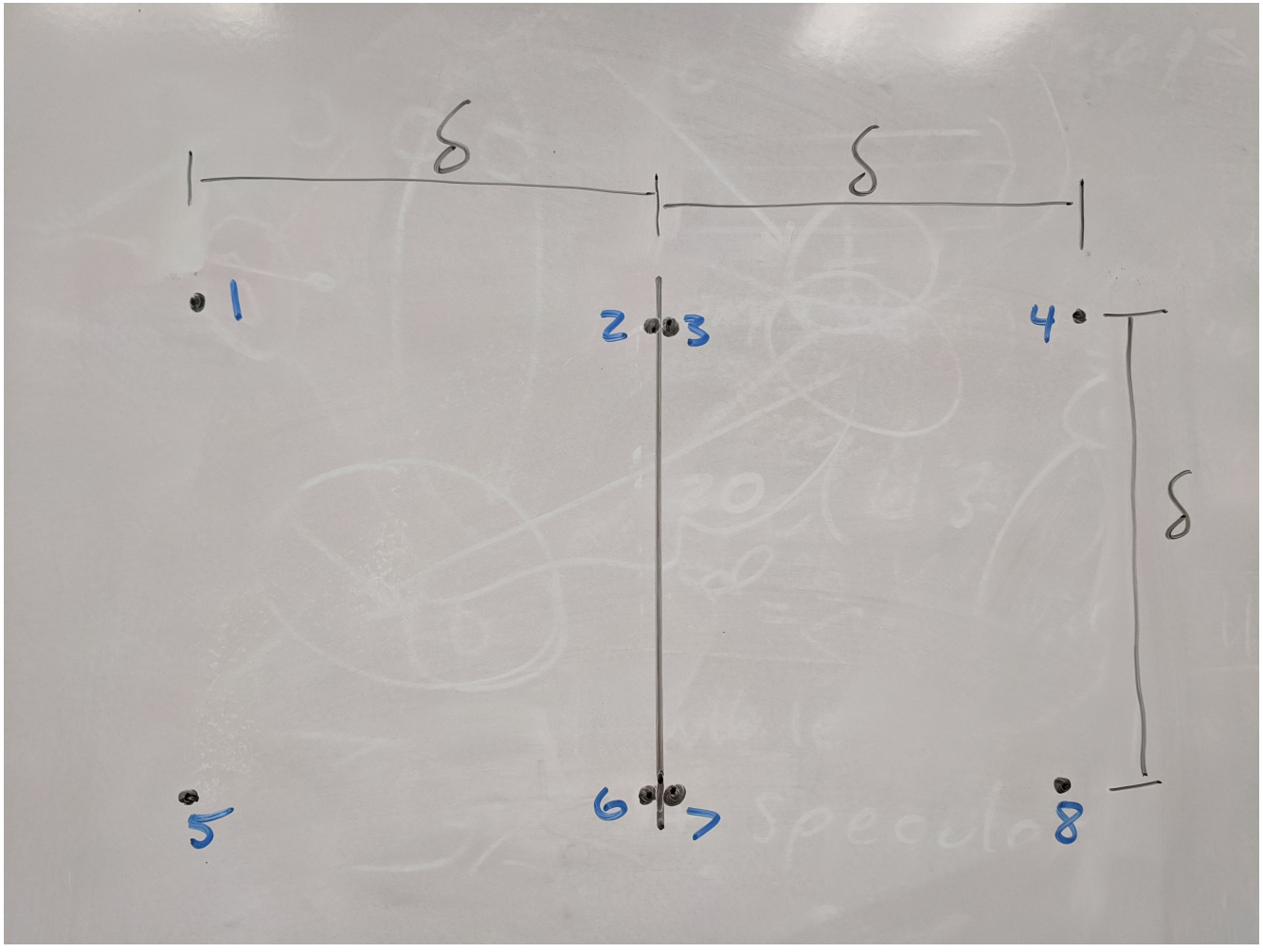
Region that Could Contain a Closest Straddling Pair



Narrowing Search at Boundary

- *packing argument:* since non-straddling point pairs are separated by $\geq \delta$, there are at most 8 non-straddling points in this rectangle (4 per corner of each square)
- $\therefore$ for each point p within δ of ℓ, test p against the 7 points nearest p in y-direction
- $\leq n$ points within δ of ℓ so $\leq 7n$ pairs of points $\in O(n)$

Densest Packing of 8 Points all δ Apart



Divide-and-Conquer Second Draft

```
1:  function CLOSEST-PAIR-DC(Q)
2:      if n ≤ 3 then
3:          return CLOSEST-PAIR-NAIVE(Q)
4:      else
5:          X = sort Q by x-coordinate
6:          Y = sort Q by y-coordinate
7:          ℓ = vertical line through median x-coordinate
8:          L = {p ∈ Q : p left of ℓ}, R = Q − L
9:          p_L, q_L = CLOSEST-PAIR-DC(L)
10:         p_R, q_R = CLOSEST-PAIR-DC(R)
11:         p, q = closer of p_L, q_L versus p_R, q_R; δ = d(p, q)
12:         for a ∈ Q and within δ of ℓ do
13:             for 7 points b preceding a in Y do
14:                 if d(a, b) < δ then
15:                     p = a, q = b, δ = d(a, b)
16:                 end if
17:             end for
18:         end for
19:         return p, q
```

Second Draft Analysis

- base case is $\Theta(1)$
- each sort is $\Theta(n \log n)$
- compute ℓ is $\Theta(1)$ (given sorted X)
- build L, R is $\Theta(n)$
- straddling **for** loop is $\Theta(7n) = \Theta(n)$
- $T(n) = 2T(n/2) + \Theta(n \log n)$
- by master theorem, $\Theta(n \log^2 n)$
- **bottleneck** is sorting X, Y; can do this once before recursion

Third Draft – Outer Algorithm

```
1: function CLOSEST-PAIR(Q)
2:     X = sort Q by x-coordinate
3:     Y = sort Q by y-coordinate
4:     Return CLOSEST-PAIR-HELPER(X, X, Y)
5: end function
```

Third Draft – Recursive Helper

```
1:  function CLOSEST-PAIR-HELPER(P, X, Y)
2:      if n ≤ 3 then
3:          return CLOSEST-PAIR-NAIVE(P)
4:      else
5:          x_m = median x-coordinate in P
6:          ℓ = vertical line through x_m
7:          L = {p ∈ P : p left of ℓ}, R = P − L
8:          p_L, q_L = CLOSEST-PAIR-HELPER(L, X, Y)
9:          p_R, q_R = CLOSEST-PAIR-HELPER(R, X, Y)
10:         p, q = closer of p_L, q_L versus p_R, q_R; δ = d(p, q)
11:         for a ∈ P and within δ of ℓ do
12:             for 7 points b preceding a in Y do
13:                 if d(a, b) < δ then
14:                     p = a, q = b, δ = d(a, b)
15:                 end if
16:             end for
17:         end for
18:         return p, q
19:     end if
```

Third Draft Analysis

- helper:
  - find median x is $\Theta(n)$
  - (use general median-finding algorithm; or count $k = |P \cap X|$ then iterate past $k/2$ elements of X)
  - compute ℓ is $\Theta(1)$ (given median)
  - form L, R is $\Theta(n)$
  - straddling **for** loop is $\Theta(7n) = \Theta(n)$
  - $T(n) = 2T(n/2) + n \in \Theta(n \log n)$ by master theorem
- outer algorithm:
  - each sort is $\Theta(n \log n)$
  - helper is $\Theta(n \log n)$
- total $\Theta(n \log n)$

Closest Pair Summary

Divide-and-conquer algorithm takes $\Theta(n \log n)$ time.

Builds on

- geometric packing argument: checking only $7n$ pairs of straddling points suffices
- sort in $\Theta(n \log n)$
- median in $\Theta(n)$
- master theorem