S.KIMMEL

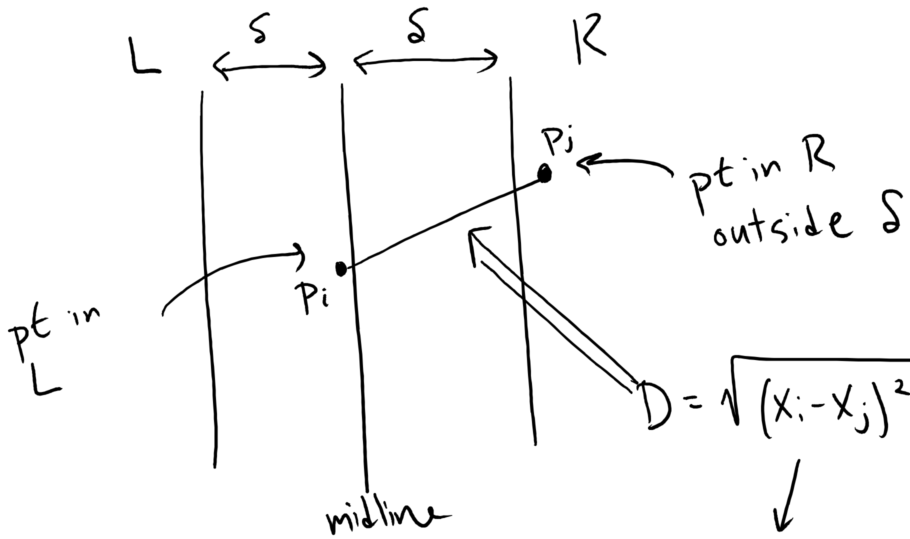
Goals

- Describe and analyze Closest Points Alg.

- Proof writing resources
- δ of midline
- PS feedback

Algorithm Sketch for Closest Points

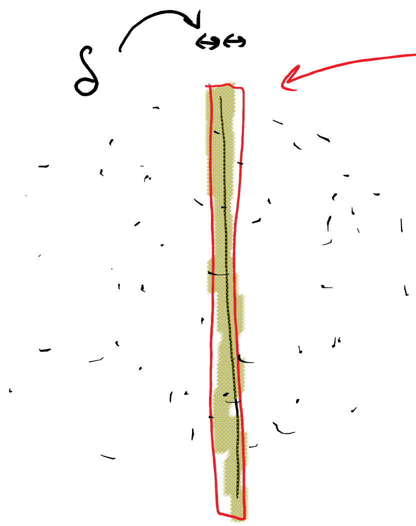
1. Base Case: 2 or 3 pts, do brute force
2. Recursive Step: Recurse on L, R halves, let δ be smallest distance in either half



$$D = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$$(x_i - x_j)^2 \geq \delta^2 \qquad (y_i - y_j)^2 \geq 0$$

so $D \geq \sqrt{\delta^2} = \delta$



If squint, looks like points on a line! For line:

1. Sort by y-coordinate
2. For-loop to look at nearest neighbors

3. Create sorted list of points within δ of midline (Y_δ). Loop through Y_δ, checking distance between each point and next __ pts. Let δ' be smallest distance found in whole loop.
4. Return $\min\{\delta, \delta'\}$

S. KIMMEL

Algorithm Sketch Summary for Closest Points

1. Base Case: 2 or 3 pts, do brute force
2. Recursive Step: Recurse on L, R halves, let δ be smallest distance in either half
3. Create sorted list of points within δ of midline (Y_s). Loop through Y_s , checking distance between each point and next __ pts. Let δ' be smallest distance found in whole loop.
4. Return $\min\{\delta, \delta'\}$

Q:

A) Why only need to check next and not previous?

B) Next ? points...

(Hint... no two points in L or R are closer than δ)

C) Why did unique x,y coordinates make our lives easier?

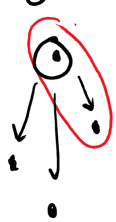
S. KIMMEL

Algorithm Sketch Summary

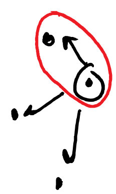
1. Base Case: 2 or 3 pts, do brute force
2. Recursive Step: Recurse on L, R halves, let δ be smallest distance in either half
3. Create sorted list of points within δ of midline (Y_δ). Loop through Y_δ, checking distance between each point and next __ pts. Let δ' be smallest distance found in whole loop.
4. Return $\min\{\delta, \delta'\}$

Q:

A) Why only need to check next and not previous?



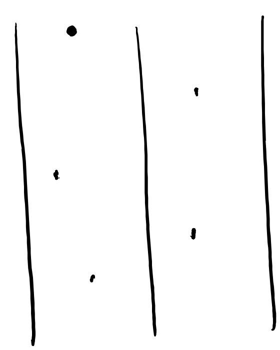
Compare to next



Don't need to compare to previous because already checked that distance

B) Next __?__ points...

(Hint... no two points in L or R are closer than δ



C) Why did unique x, y coordinates our lives easier?

Every point in L or R. Otherwise could have a cluster all on midline

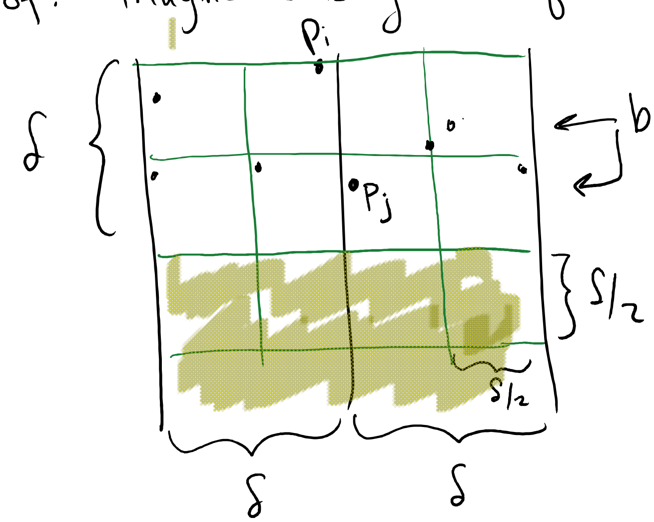
S. KIMMEL

Let

- Y_δ be array of points, within δ of midline line, sorted by y-coordinate
- p_i be i^{th} element of Y_δ

Claim ♡: If $d(p_i, p_j) < \delta$, then $|i-j| \leq 7$

Proof: Imagine dividing into squares of $\frac{\delta}{2} \times \frac{\delta}{2}$, starting at p_i

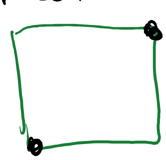


boxes where p_j might be (8 possible)

Too far away

NOTE: there is ≤ 1 pt in each square

For contradiction, suppose 2 pts in square:



Largest distance at corners

Distance: $\delta/\sqrt{2}$

Each square in L or R, so points must have distance at least δ by inductive assumption. Contradiction!

8 squares possible → 8 pts possible → check next 7 pts

(Can do better analysis, but more work for little improvement)

S.KIMMEL

Time analysis:

Q: For each step, what is big-O run time?
Let $T(n)$ = runtime on n points, $|P|=n$

ClosestPair (P)

1. If $|P| \le 3$, brute force

2. Sort by x-coordinate into L, R

3. $\delta = \min\{\text{ClosestPair}(L), \text{ClosestPair}(R)\}$

4. Create Y_δ, an array of pts within δ of midline, sorted by y-coordinate

5. Loop through Y_δ, calculate distance from each pt to next 7 pts, keep track of smallest distance δ'

6. return $\min\{\delta', \delta\}$

S. KIMMEL

Time analysis:

Q: For each step, what is big-O run time?

ClosestPair (P)

1. If $|P| \leq 3$, brute force $O(1)$

2. Sort by x-coordinate into L, R $O(n \log n)$

3. $\delta = \min\{\text{ClosestPair}(L), \text{ClosestPair}(R)\}$ $2T(\frac{n}{2})$

4. Create Y_δ, an array of pts within δ of midline, sorted by y-coordinate $O(n \log n)$

5. Loop through Y_δ, calculate distance from each pt to next 7 pts, keep track of smallest distance δ' $O(n)$

6. return $\min\{\delta', \delta\}$ $O(1)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

S.KIMMEL

Q: Now what is runtime of each step

Preprocess: Sort P into X, Y

arrays of all points sorted by x, y-coordinate

ClosestPair (X, Y)

1. If $|P| \leq 3$, brute force

2. Create X_L, Y_L X_R, Y_R for left/right halves

3. $\delta = \min\{\text{ClosestPair}(X_L, Y_L), \text{ClosestPair}(X_R, Y_R)\}$

4. Create Y_δ, an array of pts within δ of midline, sorted by y-coordinate

5. Loop through Y_δ, calculate distance from each pt to next 7 pts, keep track of smallest distance δ'

6. return $\min\{\delta', \delta\}$

S. Kimmel

Better Runtime:

0. Preprocess: Sort P into X, Y $O(n\log n)$

arrays of all points sorted by x, y-coordinate

<u>ClosestPair</u> (X, Y)

1. If $|P| \leq 3$, brute force $O(1)$

2. Create X_L, Y_L X_R, Y_R for left/right halves $O(n)$

3. $\delta = \min\{\text{ClosestPair}(X_L, Y_L), \text{ClosestPair}(X_R, Y_R)\}$ $2T(\frac{n}{2})$

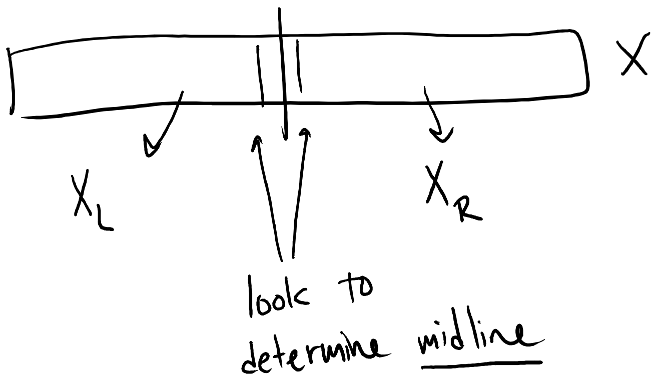
4. Create Y_δ, an array of pts within δ of midline, sorted by y-coordinate $O(n)$

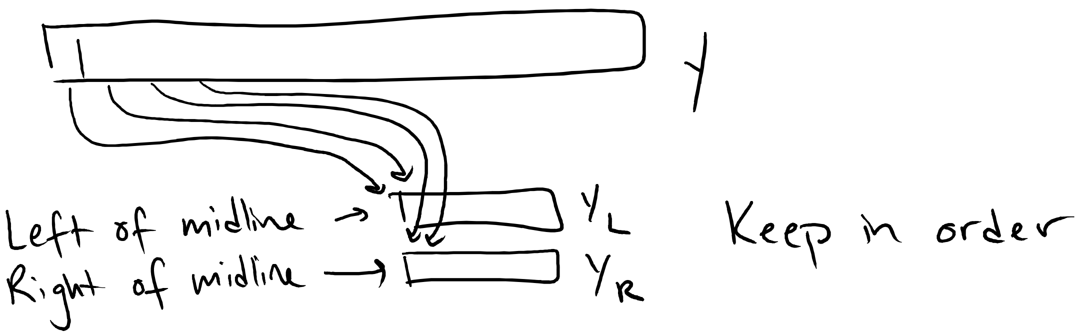
5. Loop through Y_δ, calculate distance from each pt to next 7 pts, keep track of smallest distance δ' $O(n)$

6. return $\min\{\delta', \delta\}$ $O(1)$

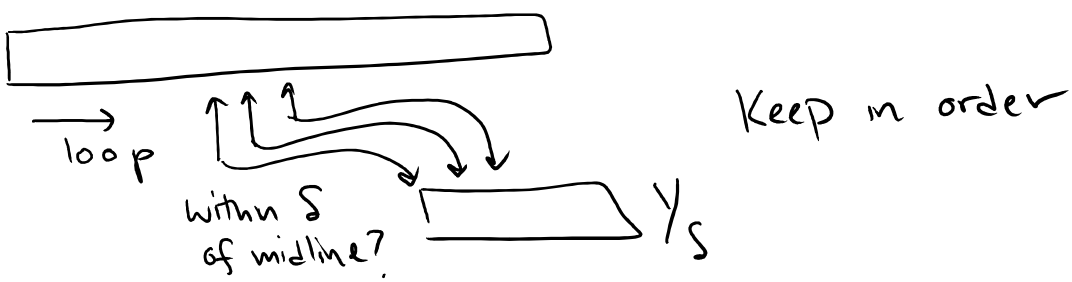
$T(n) = 2T(n/2) + O(n)$ (preprocess $O(n\log n)$)

Step 2:


X
X_L
X_R
look to
determine midline


Y
Left of midline
Right of midline
Y_L
Y_R
Keep in order

Step 4


loop
within δ
of midline?
Y_S
Keep in order