**Goals**
- Describe and analyze Closest Points Alg.
- Proof writing resources
- PS feedback
- S of midline

**Algorithm Sketch for Closest Points**

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

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

\[
(x_i - x_j)^2 \geq S^2 \quad (y_i - y_j)^2 \geq 0
\]

so \( D \geq \sqrt{S^2} = S \)
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 $S$ of midline ($Y_s$). Loop through $Y_s$, checking distance between each point and next points. Let $S'$ be smallest distance found in whole loop.

4. Return $\min\{S, S'\}$
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 S be smallest distance in either half

3. Create sorted list of points within \( S \) of midline \( (Ys) \). Loop through \( Ys \), checking distance between each point and next — pts. Let \( S' \) be smallest distance found in whole loop.

4. Return \( \min \{ S, S' \} \)

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 \( S \))

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

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

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 S

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

Every point in L or R. Otherwise could have a cluster all on midline
Let
\( Y_5 \) be array of points, within \( S \) of midline line, sorted by \( y \)-coordinate
\( p_i \) be \( i \)-th element of \( Y_5 \)

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

Proof: Imagine dividing into squares of \( \frac{\frac{S}{2}}{2} \times \frac{\frac{S}{2}}{2} \), starting at \( p_i \)

\[
\begin{array}{c}
\{ & \\
& \\
& \leftarrow \text{boxes where } p_j \text{ might be} \\
& \left\{ 8 \text{ possible} \right\} \\
& \leftarrow \text{Too far away} \\
\end{array}
\]

\( S \)
\( S \)

NOTE: there is \( \leq 1 \) pt in each square

For contradiction, suppose 2 pts in square:

Largest distance at corners
Distance: \( \frac{\sqrt{2}S}{2} \)

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

8 squares possible \( \rightarrow \) 8 pts possible \( \rightarrow \) check next 7 pts

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

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

Let $T(n)$ = runtime on $n$ points, $|P|=n$

**ClosestPair** ($P$)

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

2. Sort by x-coordinate into $L$, $R$

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

4. Create $Y_S$, an array of pts within $S$ of midline, sorted by y-coordinate

5. Loop through $Y_S$, calculate distance from each pt to next 7 pts, keep track of smallest distance $S'$

6. return $\min \{ S', S \}$
Time analysis:

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

ClosestPair (p)
1. If |P| ≤ 3, brute force \( O(1) \)

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

3. \( S = \min \{ \text{ClosestPair} (L), \text{ClosestPair} (R) \} \) \( 2T\left(\frac{n}{2}\right) \)

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

5. Loop through \( Y_s \), calculate distance from each pt to next 7 pts, keep track of smallest distance \( S' \) \( O(n) \)

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

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \]
Now what is runtime of each step

Preprocess: Sort $P$ into $X,Y$ arrays of all points sorted by $x,y$-coordinate

$\text{ClosestPair}(X,Y)$

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

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

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

4. Create $Y_S$, an array of pts within $S$ of midline, sorted by $y$-coordinate

5. Loop through $Y_S$, calculate distance from each pt to next 7 pts, keep track of smallest distance $S'$

6. return $\min \{ S', S \}$
Better Runtime:

0. Preprocess: Sort P into \(X, Y\) arrays of all points sorted by \(x, y\)-coordinate \(O(n \log n)\)

\[\text{ClosestPair}(X, Y)\]

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

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

3. \(S = \min \{\text{ClosestPair}(X_L, Y_L), \text{ClosestPair}(X_R, Y_R)\}\) \(2T(n/2)\)

4. Create \(Y_S\), an array of pts within \(S\) of midline, sorted by \(y\)-coordinate \(O(n)\)

5. Loop through \(Y_S\), calculate distance from each pt to next 7 pts, keep track of smallest distance \(S'\) \(O(n)\)

6. Return \(\min \{S', S\}\) \(O(1)\)

\[T(n) = 2T(n/2) + O(n)\] (preprocess \(O(n \log n)\)
Step 2: 

Look to determine midline.

Left of midline → $Y_L$
Right of midline → $Y_R$

Keep in order.

Step 4

Loop

Within $S$ of midline?

Keep in order.