Overview

\( d_{L} = \min(d_{L}, d_{R}, d_{C}) \)

\( \rightarrow \) returned to the call to Closest

on initial \( P_{L} \)

\( \downarrow \)

\( d_{C} \)

\( \Rightarrow \)

return \( \min(d_{L}, d_{R}, d_{C}) \)
Preprocess (P)
Sort P by x-coord.

1. Closest (P) \( \text{if } |P| = \frac{\sqrt{3}}{2} \) return \( \text{dist}(p_1, p_2) \) //using distance formula
   \( \leq \text{min distance of all pairs} \)

2. else

3. \( P_L, P_R = \text{Partition (P)} \) //partition P into left, right

4. \( d_L = \text{Closest (P_L)} \) //closest distance on left

5. \( d_R = \text{Closest (P_R)} \) //closest distance on right

6. \( d = \text{min} (d_L, d_R) \)

7. \( d_c = \text{closestCrossing (P_L, P_R, d)} \) //closest dist among crossing pairs

8. (What to eventually return?) //pairs.

9. return \( \text{min}(d_L, d_R, d_c) \)

<Overview of Algorithm>

Questions:
(1) How to partition?
(2) What if subspace contains only 1 point?
(3) How to find closest distance among crossing pairs in \( O(n^2) \) time?
(1) Partition: "How to partition into $P_L, P_R$?"

Partition ($P$)

- sort $P$ by $x$-coord
- $P_L = P[1 \ldots \lceil n/2 \rceil]$
- $P_R = P[\lceil n/2 + 1 \rceil \ldots n]$

return $P_L, P_R$.

Problem? Partition() called for every recursive call
So sort required each time.

Solution: Sort $P$ once at the beginning.

- Preprocess ($P$)
  - Sort $P$ by $x$-coord
  - add to Closest() code;
  - remove from Partition()"

(2) "What if subset contains only 1 point?"
Suppose subset has 5 points

1. 1st partition
2. Compute distance using dist. formula
3. 2nd partition
4. $\rho_0$
5. $\rho$

Can't compute distance for single point!

So set Base Case at $n=3$

Another Solution:
Have 2 Base Cases:
if $n=1$ return $\infty$
if $n=2$ return dist $(\rho_1, \rho_2)$

⇒ Change Code!

Question (3): Closest Crossing

How to find closest distance b/w crossing pairs?
Simplest approach? For every point in \( P_R \), compute its distance to every point in \( P_R \).

\[ \text{Max # computations} \approx \frac{n}{2} \times \frac{n}{2} = O(n^2) \]

"Want to do better than \( O(n^2) \), eventually \( n \), but even if we can ignore some points, that would help."

"Do we need to compute distance b/w \( p_i \) and every point in \( P_R \)?"

\[ \text{No! Again, probably not necessary to compute distance b/w these 2.} \]

"Which points can we ignore?"

Hint: Look back at Closest()"
"Hint: Look back at closest()"

We know closest distance on left = \(d_L\)

\[d_{L}^{10}\]  \[d_{R}^{20}\]

So we can ignore any points that are \(>d\) from partition

\[(\text{since distance } > 10, \text{can ignore this point})\]

Add to code:

1. \(d = \min(d_L, d_R)\) in closest()
2. Send \(d\) as parameter to closest()

Note: Could also ignore points that are \(>d\) away from \(p_i\) but this turns out to be less efficient (see why later)

Check only points that are \(<d\) from partition

How to find these points? <Group Work>

\(\Rightarrow\) Points are already sorted by x-coord, so just check if x-coord is \(<d\) from x-coord of partition.
In ClosestCrossing():

for each \( P_i \) in \( P_L \):

- compute distance b/w \( P_i \) and every point \( P_j \)

\(<\text{Leave space}>\)

Now, maximum number of comparisons:

\[
\frac{n}{2} \times \frac{n}{2} = O(n^2)
\]

\(<\text{ex}: \text{All points clustered around partition}>\)

So we need to also consider y-coords.

\(<d \text{ If vertical dist} > d, \text{ don't compare}>\)
< Update closest crossing >

for each \( p_i \):
- compute distance b/w \( p_i \) and every point \( p_j \) such that:
  - \( |x_j - x_{\text{part}}| < d \) and
  - \( |y_j - y_i| < d \)

Now, how many computations?

Turns out to be \( O(n) \).

For every point \( p_i \), will compute distance to only 15 other points!
ex' partition at \( 18 \) \( d = \min (d_L, d_R) \)

Need to consider only points within \( p \)'s box. How many such points?

Split \( p \)'s box into 16 regions:

Note \( p \) must be along vertical mid-line. At most how many other points in \( p \)'s box?
each region: \[ \begin{array}{|c|c|} \hline d/2 \\ \hline d/2 \\ \hline \end{array} \]

Max # of points per region?

Remember: all pairs left/right of partition are \( \geq d \) apart

Claim: At most 1 point per region.

Proof: By way of contradiction.
Suppose by contradiction, a region has 2 points.
Note they must be \( \geq d \) apart.

Furthest we can place 2 points in one region is at corners:

\[ \begin{array}{|c|c|} \hline \hline d/2 \\
\hline \hline \end{array} \]

Are they \( \geq d \) apart?

\[ \text{dist} = \sqrt{(d/2)^2 + (d/2)^2} = \sqrt{\frac{2d^2}{4}} = \sqrt{\frac{d^2}{2}} = \frac{\sqrt{2}}{2} \]

They are \( < d \) apart! Contradiction!

\[ \exists \text{ At most 1 point in each region} \]

\[ \text{15 points to compute distance to from p} \]
Recall closest crossing:

for each $p_i$:
\[
\begin{align*}
&\text{compute distance b/w } p_i \text{ and every} \\
&\quad \text{point } p_j \text{ such that} \\
&\quad \quad \cdot |x_j - x_{\text{part}}| < d \quad \text{and} \\
&\quad \quad \cdot |y_j - y_i| < d
\end{align*}
\]

$O(15)$

Total time for $n$ points: $O(15n)$. 
Question: How to find these 15 points in constant time? (HW question)

Runtime of Algorithm?

\[ T(n) = 2T(n/2) + n \]

(Master's Merge-Sort: \( T(n) = \Theta(n \log n) \))

Note: If we looked \( d \) from \( p_i \) (instead of \( d \) from partition)

Each region surrounding partition could have at most 2 points (not at most 1)