Typical Divide + Conquer Structure

```
MergeSort
Input  : Integer array A of length n
Output: Sorted array

  // Base Case
1 if n == 1 then
2 | return A;
3 end

  // Divide and Conquer
4 A_1 = MergeSort(A[1 : n/2]);
5 A_2 = MergeSort(A[n/2 + 1 : n]);

  // Combine
6 p_1 = p_2 = 1;
7 for i=1 to n do
8 | if A_1[p_1] < A_2[p_2] then
9 | | A[i] = A_1[p_1];
10 | | p_1++;
11 | else
12 | | A[i] = A_2[p_2];
13 | | p_2++;
14 | end
15 end
```

Base Case(s)

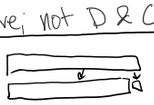
Preprocessing (none in this case)

Divide equally + solve recursively

Combine

Divide + Conquer:

Recursive, not D & C

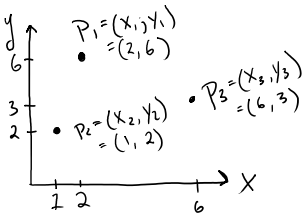


Divide + Conquer

- Description:
  - Base Case
  - Recursion } Divide + Conquer
  - Loops / Conditionals } Preprocess + Combine
- Correctness: (Strong) Inductive Proof (see review video)
- Time Complexity: Recurrence relation + master/tree method (see review video / PS 1)
- Ethics ...

Divide + Conquer Example:

Closest Points Problem



Input: array of pts

(2,6)
(1,2)
(6,3)

Output: closest $\boxed{\text{distance}}$ b/t any 2 points] in class
closest pair] programming

Distance: $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ but can use other distances

Assume unique x, y coordinate for each pt.

Applications

- Air traffic control
- Robotics
- Stereo → 3D

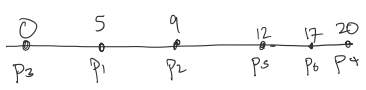
Ethics: (If you could fly anywhere, where would you go?)

- Who will benefit from this algorithm (applied to this domain)?
- Who will be harmed by this algorithm (applied to this domain)?
- Is the benefit or harm equally distributed, or are people with certain identities more likely to be impacted?
- Any other ethical concerns?
- Would you feel comfortable (from an ethical perspective) implementing this algorithm in this context?

Before starting, think about runtime we'd like to achieve

- Better than "Brute force" (Alg? Runtime?)
- Probably can't do better than 1-D ($O(n \log n)$ Alg)

1-D: | 5 | 9 | 0 | 20 | 12 | 17 | closest points



Brute Force: (check every pair)

```
min ← ∞   // min=∞   // min:=∞
for i = 1 to n-1:
    for j = i+1 to n:
        if dist(pi, pj) < min, then
            min ← dist(pi, pj)
Return min
```

" ← "

every pair

$O(1)$ $O(n)$ $O(n^2)$

On a Line (1-D)

Sort points using MergeSort $O(n \log n)$

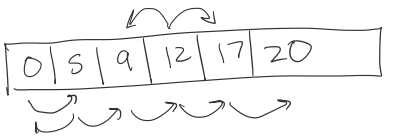
```
min ← ∞
for i = 1 to n-1:
    if dist(pi, pi+1) < min, then
        min ← dist(pi, pi+1)
Return min
```

+

$O(1)$ $O(n)$ $O(n \log n)$

Why correct?

After sorting $\Rightarrow$

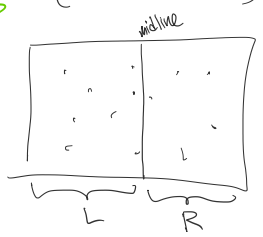


Goal: Better than $O(n^2)$. Best case, achieve $O(n \log n)$.

ClosPt (divide + conquer) — Closest Points' Distance

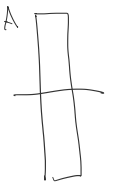
Base case (details later)

Divide:



P (details later)

Not:



Solve:

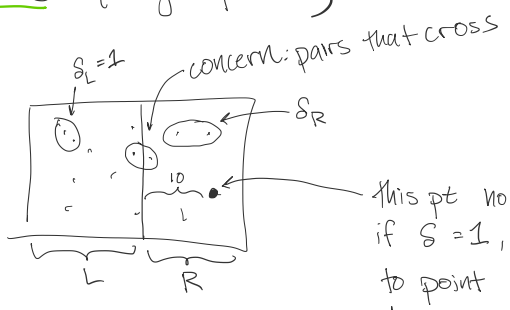
$\delta_L \leftarrow ClosPt(L)$

$\delta_R \leftarrow ClosPt(R)$

$\delta \leftarrow \min\{\delta_L, \delta_R\}$

δ = "delta"

Combine (thought process)



this pt not a concern if $\delta = 1$, because distance to point in L is too large

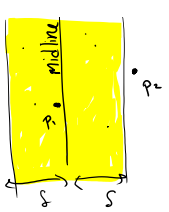
What points do we need to worry about? Those within _____.

A) $\delta/2$ of midline

B) $\sqrt{\delta}$ of midline

C) δ of midline

D) 2δ of midline?



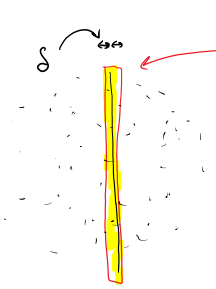
Lemma: If p_2 in R is more than δ from midline, its distance to any pt. p_1 in L is more than δ.

Pf Sketch: $d(p_1, p_2) = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$

But:
- $(x_1-x_2)^2 > \delta^2$
- $(y_1-y_2)^2 > 0$

So: $(x_1-x_2)^2 + (y_1-y_2)^2 > \delta^2$

Thus: $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} > \delta$



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

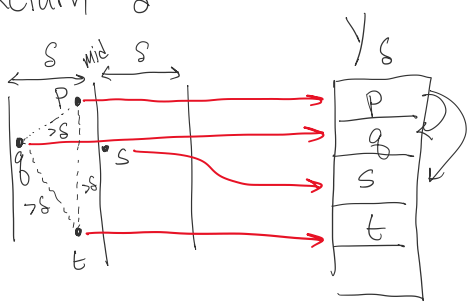
1. Sort by y-coordinate
2. For-loop to look at nearest neighbors

Combine

$Y_\delta \leftarrow$ y-sorted list of pts in P with x-coord within δ of midline (details later!)

For $p \in Y_\delta$:
| Check distance from p to next — pts in Y_δ
| If distance less than δ found, set δ to new dist.

Return δ



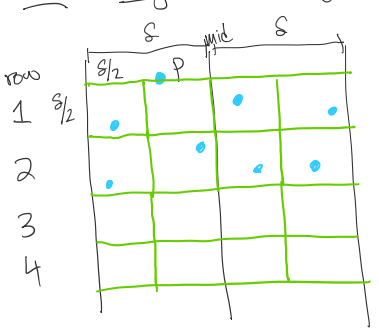
closest to p is s. Need to check at least 2

Group Questions

- Why only <u>next</u> pts + not previous? Already compared
- How many next pts?
- Where did uniqueness of x, y coordinates come into algorithm? Dividing midline is clear: each pt is in L or R. CAUTION! Programming Assignment!

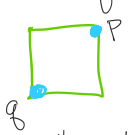
<u>Lemma</u>: Only need to look at next 7 pts.

<u>Pf</u>: <u>Imagine</u> dividing into $\delta/2 \times \delta/2$ squares, starting @ p



Each square can contain at most 1 pt.

(For contradiction, suppose 2 pts in a square:



$d(p,q) = \sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \frac{\delta}{\sqrt{2}} < \delta$

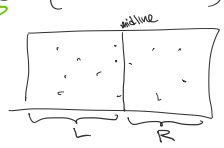
Not allowed b/c both points in L or R so must have distance at least δ.)

Points in row 3, 4, etc are $> \delta$ away from p.
Only 7 possible other pts in row 1, 2.
These will be the next 7 points in Y_δ, since Y-sorted

ClosPt (divide + conquer) — Closest Points' Distance

Base case (details later)

Divide:



} P (details later)

Solve:

$\delta_L \leftarrow ClosPt(L)$

$\delta_R \leftarrow ClosPt(R)$

$\delta \leftarrow \min\{\delta_L, \delta_R\}$

Combine

$Y_\delta \leftarrow$ y-sorted list of pts in P with x-coord within δ of midline (details later!)

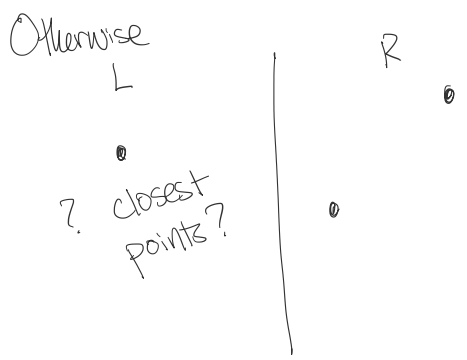
For $p \in Y_\delta$:

| Check distance from p to next — pts in Y_δ
| If distance less than δ found, set δ to new dist.

Return δ

Q: What size set of points should trigger base case?

A) 0 B) 1 C) ≤ 2 D) ≤ 3



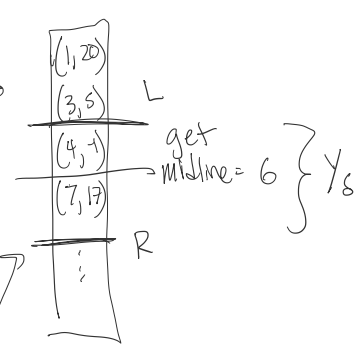
ClosPt(P) — n pts

$O(1)$ • If $n \leq 3$, brute force

$O(n \log n)$ • Sort P by x-coordinate →

$O(n)$ • Divide into L, R →

$2T(n/2) + O(1)$ • $\delta = \min\{ClosPt(L), ClosPt(R)\}$



$O(n)$ • Create Y_δ

$O(n \log n)$ • Sort Y_δ by y-coordinate

$O(n)$ • For $p \in Y_\delta$:

| For i = 1 to 7:
| | $\delta' = dist(p, p+i^{th})$
| | If $\delta' < \delta$, then $\delta \leftarrow \delta'$

$O(1)$ $O(n)$

$O(1)$ • Return δ

$T(n)=O(1)$ if $n \le 3$, $T(n) = 2T(n/2) + O(n\log n)$

comes from sorting

PreSortP(P)

X = P sorted by x
Y = P sorted by y

Runtime $\rightarrow O(n\log n)$

ClosPt(X, Y)

$O(1)$ • If $n \le 3$, brute force

~~$O(n\log n)$ • Sort P by x-coordinate~~

$O(n)$ • Divide into ~~L, R~~ X_L, X_R, Y_L, Y_R

$2T(n/2) + O(1)$ • $\delta = \min\{\text{ClosPt}(\overset{X_L, Y_L}{L}), \text{ClosPt}(\overset{X_R, Y_R}{R})\}$

$O(n)$ • Create Y_δ (from Y)

~~$O(n\log n)$ • Sort Y_δ by y-coordinate~~

$O(n)$ • For $p \in Y_\delta$:

For $i = 1$ to 7:

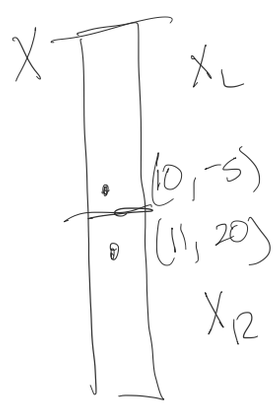
$\delta' = \text{dist}(p, p+i^{th})$

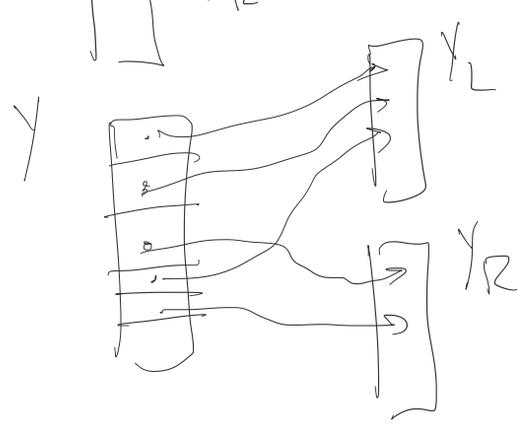
If $\delta' < \delta$, then $\delta \leftarrow \delta'$

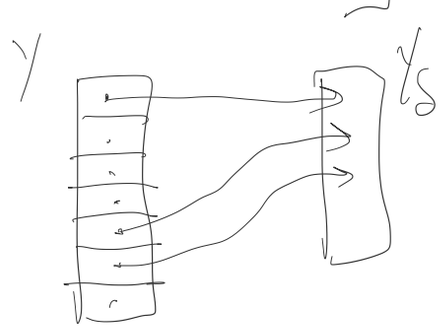
$O(1)$

$O(n)$

$O(1)$ • Return δ







$T(n)=O(1)$ if $n \le 3$, $T(n) = 2T(n/2) + O(n)$

AMAZING!

as good as 1-D

⋮

3D??

$T(n) = O(n\log n)$

WOW