Goals:
1. Understand Divide and Conquer Structure
2. Practice pre-design benchmarking
3. Figure out base case for closest points

Typical Divide + Conquer Structure

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

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

  // Divide and Conquer
  A1 = MergeSort(A[1 : n/2]);
  A2 = MergeSort(A[n/2 + 1 : n]);

  // Combine
  p1 = p2 = 1;
  for i=1 to n do
  |  if A1[p1] < A2[p2] then
  |  |  A[i] = A1[p1];
  |  |  p1++;
  |  else
  |  |  A[i] = A2[p2];
  |  |  p2++;
  |  end
  end
```

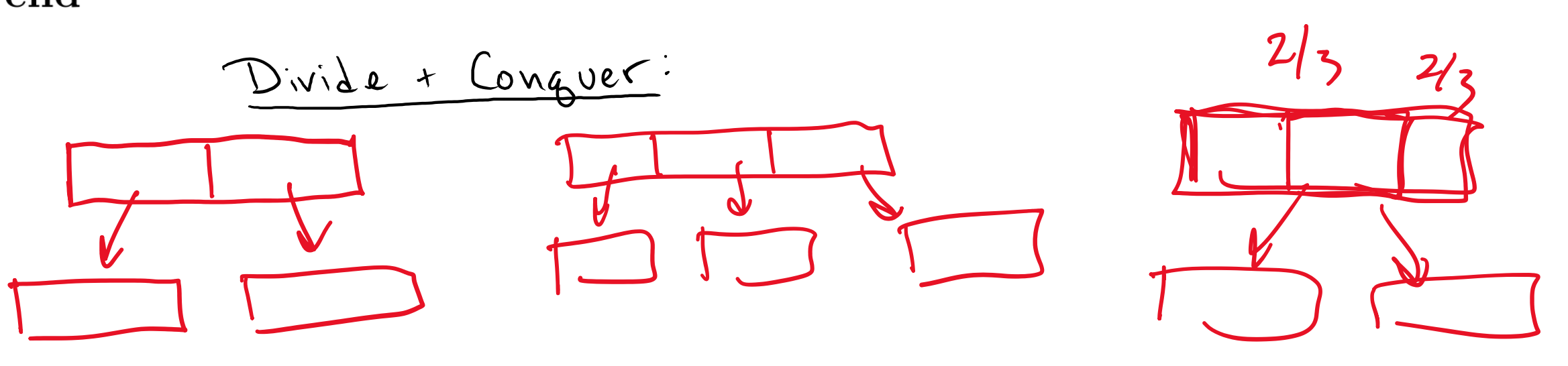
} Base Case(s)

← Preprocessing

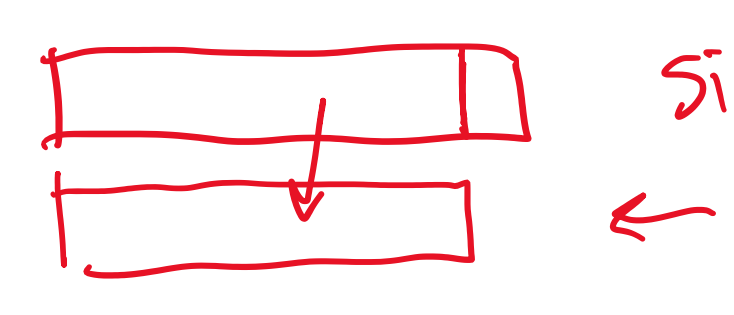
Divide Input + Call Recursively

} Combine / Postprocessing

Divide + Conquer:



Recursive, not D & C

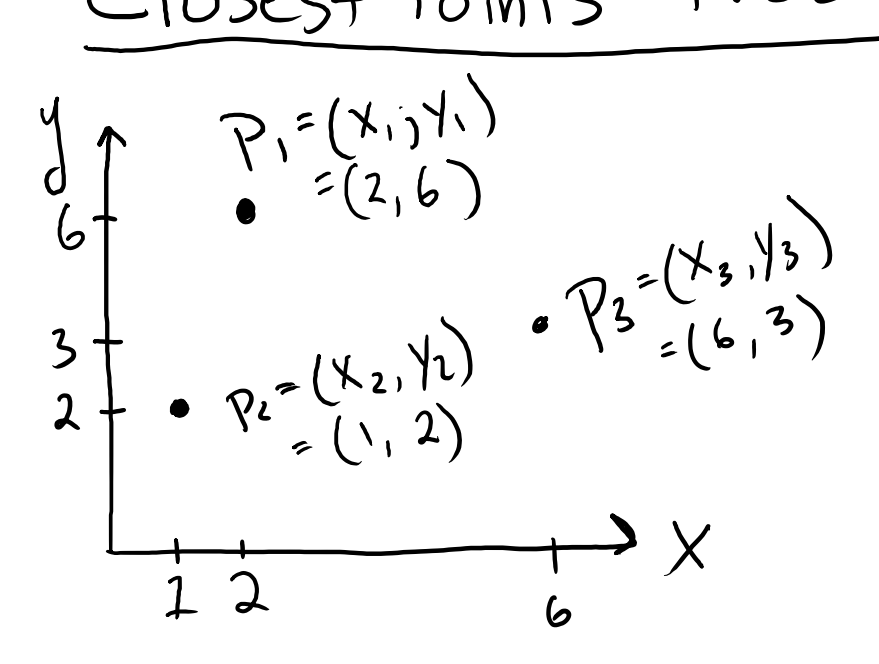


Divide + Conquer

- Description:
  - Base case
  - Recursions
  - Loops / Condition in pre/post processing
- Correctness: Strong Induction
- Time Complexity: Tree Formula (Master method) (a lot of the time)
- Ethics ... ?

Divide + Conquer Example:

Closest Points Problem



Input:

(2,6)
(1,2)
(6,3)
~~(2,8)~~

← Array

(2,8): not allowed b/c not unique x-coord

Output: Closest distance b/t any 2 pts ← class
closest points ← prog. assign

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

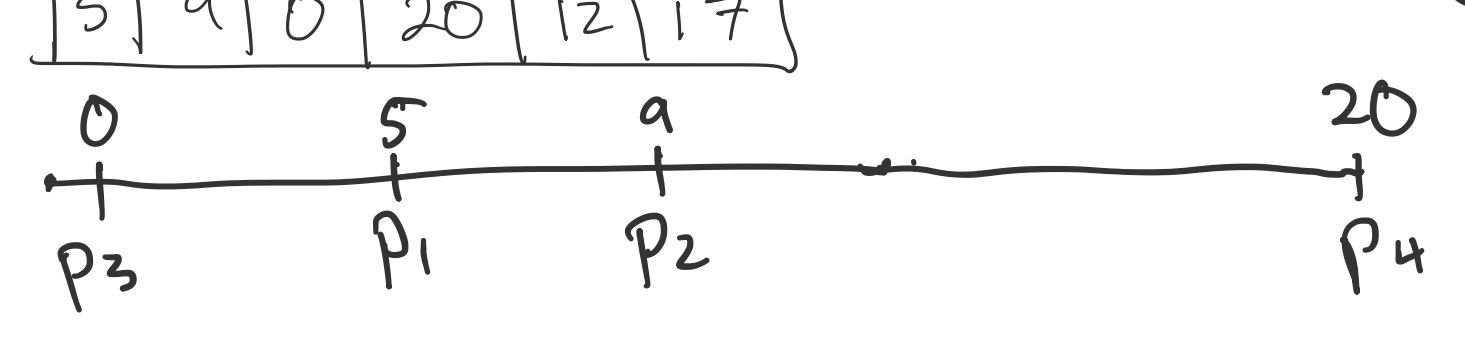
☆ Assume unique x, y coordinates

Applications
- Air traffic control
- Robotics
- Stereo → 3D

Ethics: (If you could fly anywhere, where would you go?)
1. Brainstorm all stake-holders.
2. Who might benefit from this algorithm (applied to this domain)?
   a. Passengers, airlines (reduced flight times), shareholders (more flights=more profit), Uber/Lyft (car traffic version), environmental benefit if shorter flights,
3. Who might be harmed by this algorithm (applied to this domain)?
   a. Environmental impact (more flight), folks living near airports (noise pollution),
4. Reinforce or counteract existing inequities?
   a. School districting (reinforces educational inequities), lower the cost of flying (make flying more accessible), bigger airlines -> bigger
5. Any other ethical concerns?
6. 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"
- Probably can't do better than 1-D $O(n \log n)$

Input: | 5 | 9 | 0 | 20 | 12 | 17 |



Brute Force: (Check every pair)

```
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
```

← $O(1)$ } ×~n $O(n)$ } ×~n $O(n^2)$

On a Line (1-D)

```
Sort points (Mergesort)
min ← ∞
for i = 1 to n-1
|  if dist(pi, pi+1) < min then min ← dist(pi, pi-1)
return min
```

Sort points ← $O(n \log n)$; min ← ∞ ← $O(1)$; loop body ← $O(1)$ } ×n $O(n)$

$O(n \log n)$

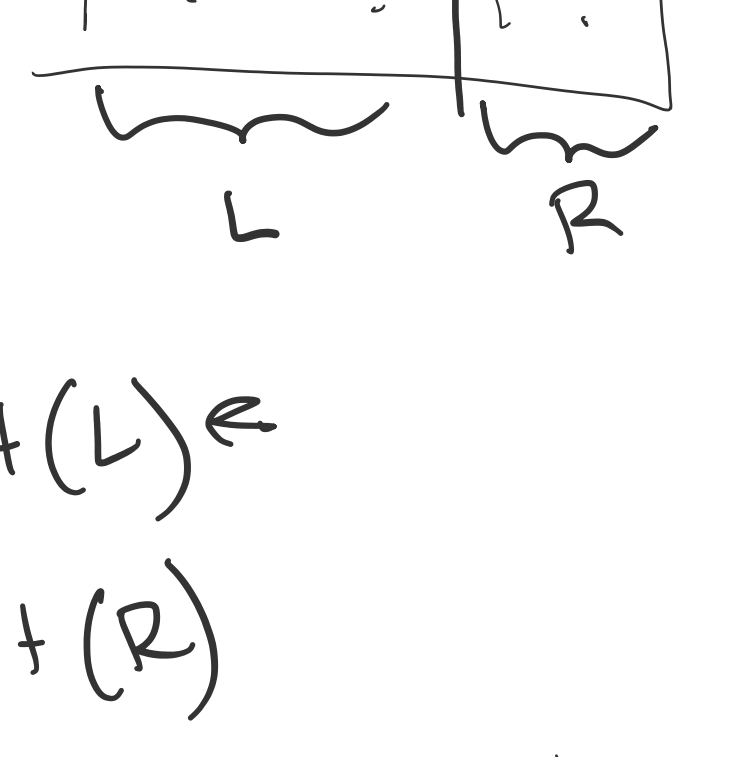
0	5	9	12	17	20

ClоPt (divide + conquer)

ClоPt = Closest Points' Distance

Base case ...

Divide:



P = set of all pts

Solve:

$\delta_L = ClоPt(L)$ ←

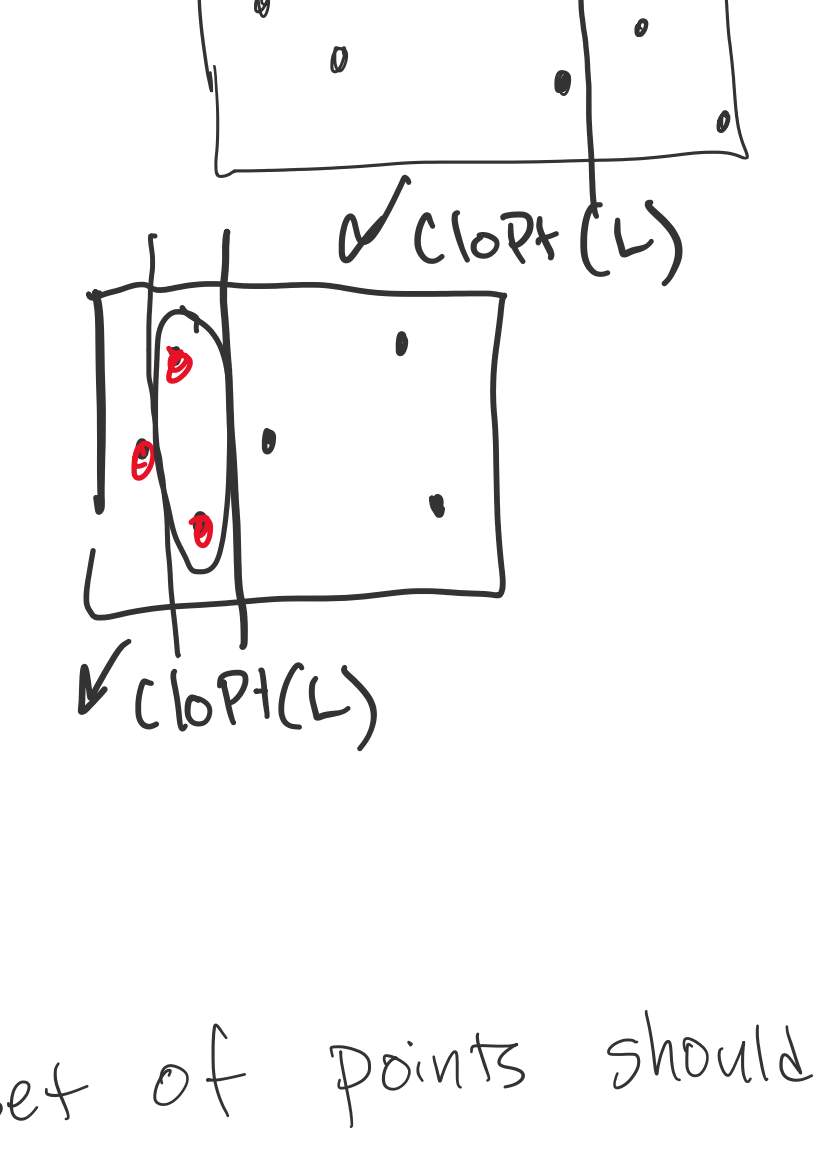
$\delta_R = ClоPt(R)$

δ = "delta"

Combine

...

Example:



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

~~A) 0~~ ~~B) ≤1~~ C) ≤ 2 (D) ≤ 3)

Base case: If $|P| \le 3$: do Brute Force