Divide and Conquer Paradigm

The problem is divided into smaller sub-problems; we then recursively solve the small subproblems & then combine these to obtain the soln to the original problem.

Merge Sort

In merge sort we are given a seq^n of numbers $a_1, a_2, \ldots, a_n$

We first divide the seq^n into 2 subsequences of almost equal lengths.

We then recursively sort the two subsequence & finally "merge" them to obtain the sorted seq^n

For simplicity, we assume that
n is a power of two
Merge sort consists of two procedures.
The first one is MERGE(S, T)
Its inputs are two sorted seqs
S and T. The output is a sorted
seq consisting of the elements of S
and T in a sorted.

It works by repeatedly selecting the larger of two largest elements remaining in S & T & then deleting the selected element.

(Ex. Write down a pseudo-code for this procedure)

The other procedure is Sort(i,j)

that sorts the subset $x_i, x_{i+1}, \ldots, x_j$

SORT(i, j) (Here also we assume that the length of the seq is 2^k for some $k \geq 0$).

If $i = j$ Then return x_i

else

begin

$m \leftarrow \frac{i+j-1}{2}$

$S \leftarrow$ SORT(i, m)

$T \leftarrow$ SORT(m+1, j)

return MERGE(S, T).

[X Prove the correctness of MERGE SORT

Complexity Let $T(n)$ be the no. of comparisons required to sort a seq of n elements. Then $T(n)$ satisfies the following recurrence

$$T(n) = \begin{cases} 0 & \text{if } n = 1. \\ 2T(n/2) + (n-1) & \text{if } n > 1 \end{cases}$$

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

$$= 2T(2^{k-1}) + 2^k - (1 + 2 + 2^2 + \cdots + 2^{k-1})$$

$$= 2T(2^{k-2}) + (2^k - 2) + (2^k - 1)$$

$$= \{2T(2^{k-3}) + (2^{k-1} - 1)\} + (2^k - 2) + (2^k - 1)$$

$$T(n) = T(2^k) = 2T(2^{k-1}) + (2^k - 1)$$

$$n = 2^k$$

$$T(n) = \begin{cases} 0 & \text{if } n = 1. \\ T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + (n-1) \end{cases}$$

Show that $T(n) \leq n \lceil \log n \rceil$.

2. Counting Inversions.

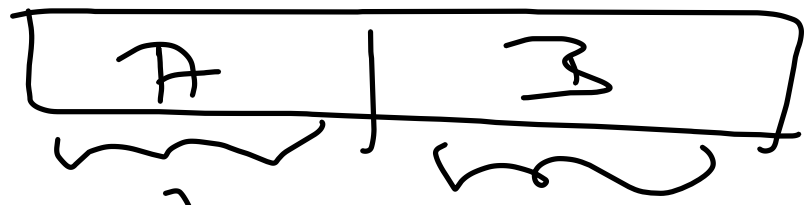
Defn. Let $A[1, n]$ be an array of distinct numbers.

If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion.

If $A[1, n]$ is sorted then the # of inversions is 0.

Algorithm SORT-AND-COUNT(L)

Input a list or array



Output The # of inversions in L
and L in sorted order.

If L has one element
Then return (L)

else divide L into two halves A and B
$(r_A, A) \leftarrow$ SORT-AND-COUNT(A)
$(r_B, B) \leftarrow$ SORT-AND-COUNT(B)

$b_j <$ $a_i < a_{i+1} < \cdots$ $(\gamma_{AB}, L) \leftarrow$ MERGE-AND-COUNT (A, B)

return $\gamma_A + \gamma_B + \gamma_{AB}$.

The following procedure counts the no. of inversions (a, b) with $a \in A$ & $b \in B$.

Procedure MERGE-AND-COUNT (A, B).

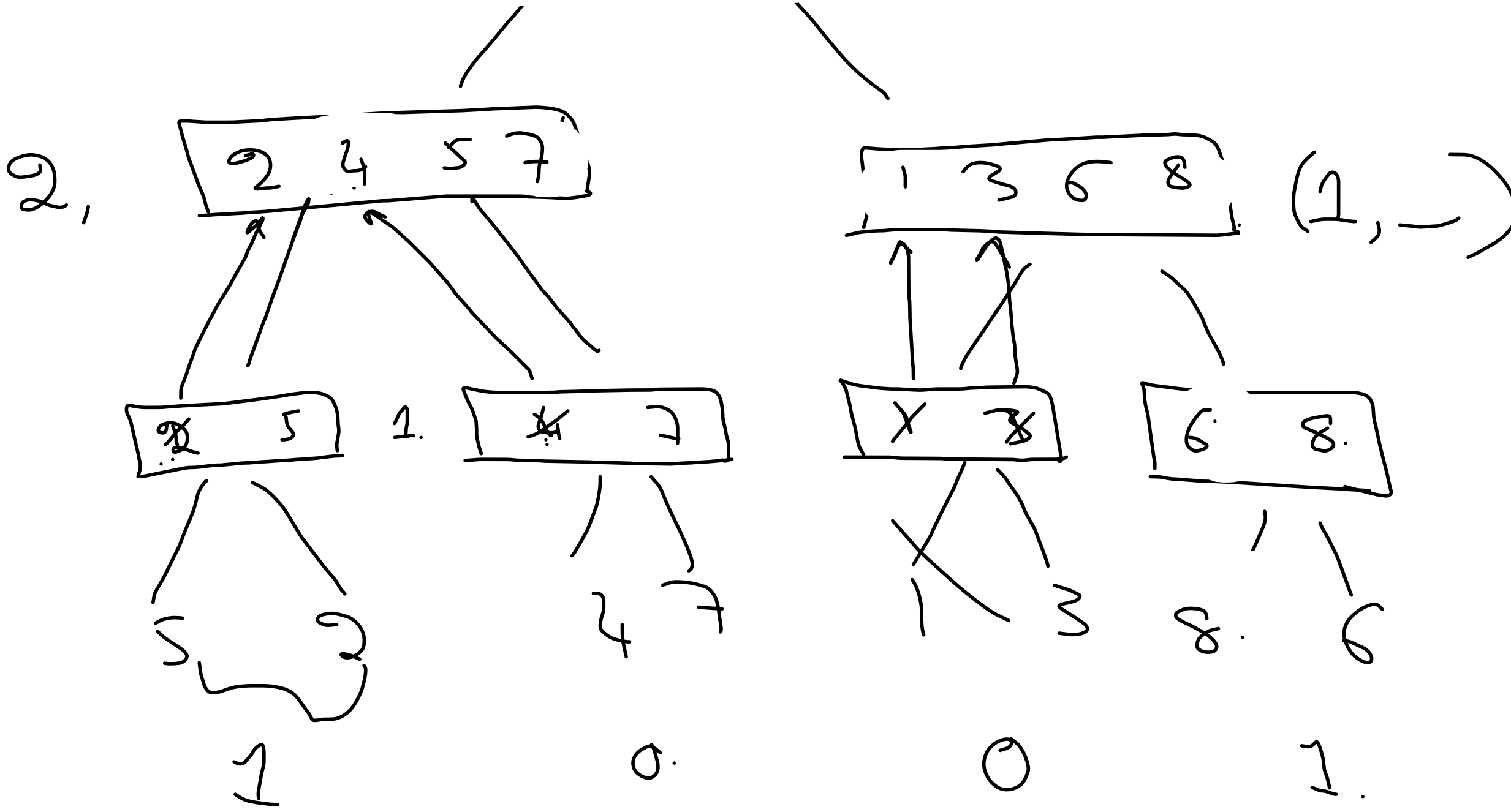
- Scan A and B from left to right
- Compare a_i and b_j
- if $a_i < b_j$ then a_i is not inverted with any element of B
- if $a_i > b_j$, then b_j is inverted with every other element in A
  Increase the count of inversions by $|A|$
- Append the smaller element to the sorted list (

Thm Algorithm SORT-AND-COUNT counts the no. of inversion in $O(n \log n)$ time.

Pf

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n) & \text{if } n > 1 \end{cases}$$

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


2,
2 4 5 7
1 3 6 8
(1,—)
2 5
1.
4 7
X 3
6 8
5 2
1
4 7
0.
1 3
0
8. 6
1.

3 Closest Pair of points.

Find the closest pair of points in a given set Q of n points in a plane. Brute-force method requires $O(n^2)$ time. We shall give a divide and conquer alg. that requires $O(n \log n)$ steps.

Divide-and-conquer algorithm

Each invocation of the recursive call takes as input a subset $P \subseteq Q$ and two arrays X and Y each containing all pts of P. The array X is sorted by monotonically increasing x-coordinate and Y is sorted by monotonically increasing y co-ordinate

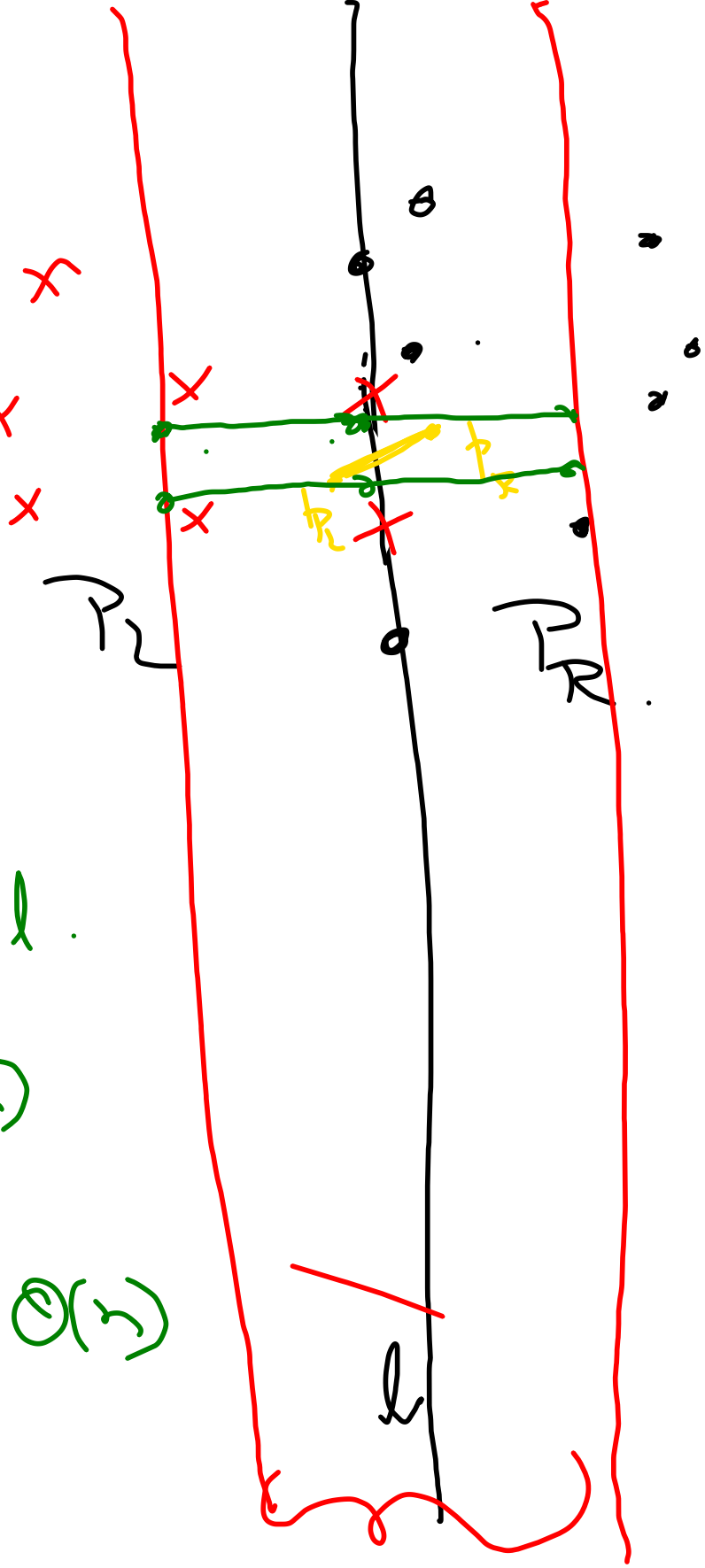
Each recursive call with inputs P, x, y first checks if $|P| \leq 3$. If so, then our alg. will output the closest by a brute-force method. Otherwise, we make the following divide-and-conquer algorithm.

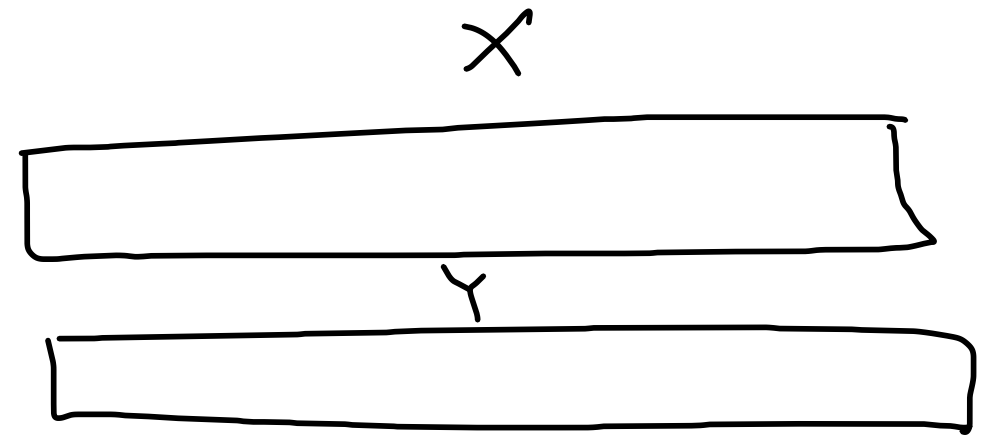
Divide Find a vertical line l dividing P into two sets P_L and P_R s.t. $|P_L| = \lceil \frac{|P|}{2} \rceil$, $|P_R| = \lfloor \frac{|P|}{2} \rfloor$. The points of P_L are on or to the left of l. While the points of P_R are on or to the right of l. The array X is divided into X_L and X_R containing the points of P_L and P_R respectively, each sorted acc. to the x-coordinate.

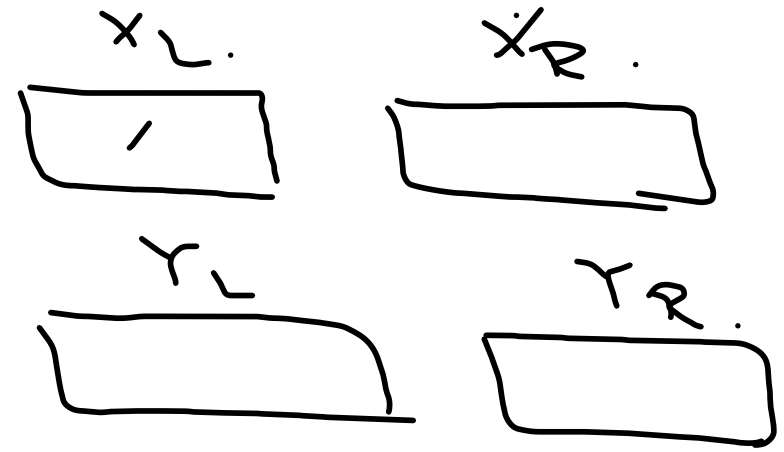
Both p_L & p_R lie in the $\delta \times 2\delta$ rectangles centred at l.

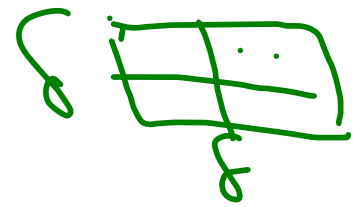
$\underline{T(n)} + O(n \log n)$

$T(n) = 2T(n/2) + O(n)$









Given a sorted array Y

P ⟶ P_L & P_R.

Y[i] ∈ P_L

⟶ Y_L

Similarly Y is divided into Y_L and Y_R.

Conquer Make two recursive calls, first to find the closest pair of points in P_L & the other to find the closest pair in P_R.

The inputs to the first recursive call are P_L, X_L and Y_L while the inputs to the 2nd call are P_R, X_R and Y_R. Let the closest dist^n for P_L and P_R be δ_1 & δ_2 respectively.

Let $\delta = \min\{\delta_1, \delta_2\}$.

Combine The closest pair is either one of
pairs returned by one of the recursive calls
or a pair of pt. with one point in P_L
and other pt. in P_R and the dist^n between
them is $< \delta$. Our alg. will find such a pair
with one pt. in P_1 & the other pt. in P_R with
dist^n $< \delta$. If such a pair exists, then each
of the each is within δ units of line l.
Hence both pts lie within the 2δ-vertical
strip centred at l. To find such a pair
we proceed as follows:

- from array Y' from Y by deleting all points not in the 2δ-strip.

  Sort Y' acc^n to Y-coordinate

- for each pt. p in Y', we look for pts in Y' which are within δ-dist^n. We shall show below that we need to consider only 7 pts. following p. Compute those 7 dist^n & upgrade the smallest dist^n δ' among all pairs in Y'
  - if δ' < δ, then we have found a smaller pair than those returned by the recursive calls.