The Divide and Conquer Paradigm.

↳ Divide into smaller sub problems.

↳ Conquer via recursive calls.

↳ Combine solutions of sub problems into one for the original problem.

Algo for counting Inversions I

The problem

input :- array A containing 1, 2, 3, ... n in some arbitrary order.

output :- no. of inversions = no. of pairs (i, j) of array indices with $i < j$ and $A[i] > A[j]$

Example :- (1, 3, 5, 2, 4, 6)

inversions :- (3, 2), (5, 2), (5, 4)

motivation :- numerical similarity measure between two ranked lists. (eg for collaborative filtering)

Counting inversions brute approach

```
for (i=0 to n) {
    for (j=i+1; j<n; j++) {
        if (arr[i] > arr[j])
            invcount++
    }
}
```

Counting inversions optimal approach using merge sort:

```
i = left_half_of_array (sorted)
j = right_half_of_array (sorted)
if (arr[i] <= arr[j])
    temp = arr[i]
else => arr[i] > arr[j]
    invcount += (mid - i + 1)
```

eg: arr[] = {1, 3, 5, 10, 2, 6, 8, 9}

1	3	5	10		2	6	8	9

↑ s=0 ↑ mid=3 j=2

1	2	3	5	6	8	9	10

inversion count = 3+1+1+1 = 6

Matrix multiplication

$X \cdot Y = Z$

(all $n \times n$ matrices)

where $z_{ij} =$ (ith row of X) $\cdot$ (jth column of Y)

$$= \sum_{k=1}^{n} x_{ik} \cdot y_{kj}$$

note: input size $= O(n^2)$

example $(n = 2)$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

iterative matrix mul runs in $O(n^3)$

Can we do better?

Applying divide and conquer

Recursive algo #1

$$X \cdot Y = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

step: 1 recursively compute 8 necessary products.

step: 2 - do the necessary additions $O(n^2)$ time.

fact: run time is $O(n^3)$ [follows from master method]

Strassen's algo. (1969)

step 1: recursively compute only (7) products.

step 2: - do the necessary (clever) additions + substractions (still $O(n^2)$ time)

fact: - better than cubic time!

The details.

The seven products: $P_1 = A(F-H)$, $P_2 = (A+B)H$, $P_3 = (C+D)E$, $P_4 = D(G-E)$, $P_5 = (A+D)(E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

Claim: $X \cdot Y = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$

step: 1 recursively compute 8 necessary products.

step: 2 - do the necessary additions $O(n^2)$ time.

fact: run time is $O(n^3)$ [follows from master method]

Strassen's algo. (1969)

step 1: recursively compute only (7) products.

step 2: - do the necessary (clever) additions + substractions (still $O(n^2)$ time)

fact: - better than cubic time!

The details.

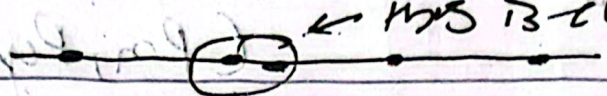
The seven products: $P_1 = A(F-H)$, $P_2 = (A+B)H$, $P_3 = (C+D)E$, $P_4 = D(G-E)$, $P_5 = (A+D)(E+H)$, $P_6 = (B-D)(G+H)$, $P_7 = (A-C)(E+F)$

Claim: $X \cdot Y = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$

Assumption :- (for convenience) all points have distinct x-coordinates, distinct y-coordinates.

Brute force search :- take $O(n^2)$ time.

1-D version of closest pair



1) sort points ($O(n \log n)$ time)
2) return closest pair of adjacent points ($O(n)$ time)

Goal :- $O(n \log n)$ time algo for 2D version.

High level Approach

1) make copies of points sorted by x-coordinate (P_x) and by y-coordinate (P_y). [$O(n \log n)$ time]
   and this is not enough!
2) use divide + conquer!

ClosestPair(P_x, P_y)

(1) Let Q = left half of P, R = right half of P. form Q_x, Q_y, R_x, R_y → (takes O(n) time)

2) (p_1, q_1) = closestPair(Q_x, Q_y)

3) (p_2, q_2) = closestPair(R_x, R_y)

4) (p_3, q_3) = ~~closestPair~~ splitPair(P_x, P_y, δ)

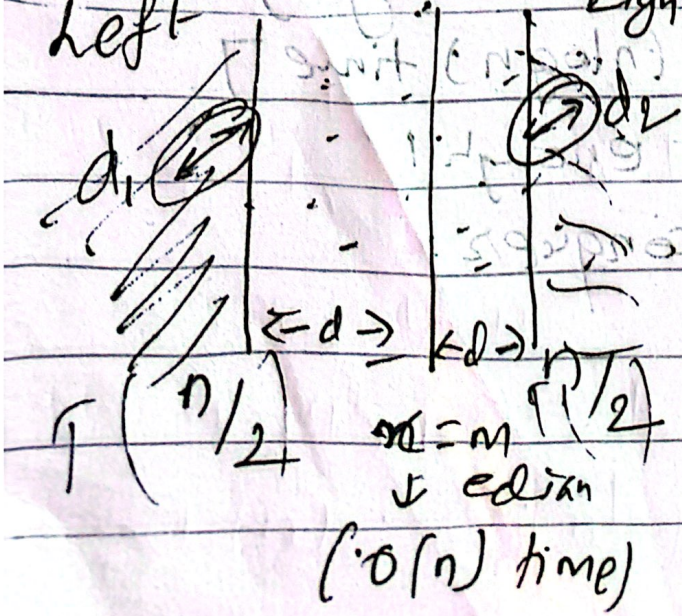
5) return best of (p_1, q_1), (p_2, q_2), (p_3, q_3)

↳ find out & which is minimum of two smallest distance.

Let us consider we have set of n points in 2D - $\{(x_1, y_1), \ldots (x_n, y_n)\}$

Output: find the closest pair.

$O(n^2) \rightarrow O(n \log n)$



Runtime complexity

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

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

$= O(n \log n)$

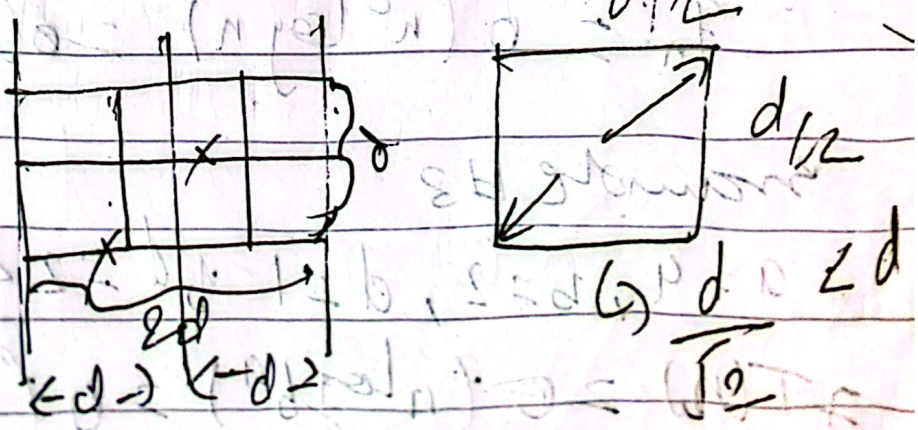
1) Find the closest pair in LEFT (d_1)

2) Find the closest pair in RIGHT (d_2)

3) ~~Find~~ check if there exists a pair such that one point is on the ~~left~~ LEFT region and other point is on the RIGHT region, and the distance b/w them is $< d = \min\{d_1, d_2\}$

4) Collect point in d region as in figure and sort them in the increasing order of y coordinates value and do distance computations - essentially, for each point consider the distance from itself to next 7 points. In total, 7 * no. of points

Correctness :-



~~Runtime comp~~

The Master Method

Recurrence $(T n)$

If $T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d)$

then,

$$T_n \leq \begin{cases} O(n^d \log n) & \text{if } a = b^d \text{ case 2} \\ O(n^d) & \text{if } a < b^d \text{ case 1} \\ O(n^{\log_b a}) & \text{if } a > b^d \text{ case 3} \end{cases}$$

where a = subproblems, b = size of subproblems.

- Example #1

Merge sort

$a = 2$

$b = 2$, $b^d = 2$

$d = 1$ $\Rightarrow$ it is case 2

$\Rightarrow T(n) \leq O(n^d \log n) = O(n \log n)$

for binary search. example #2

$T_n = O(n^0 \log n) = O(\log n)$

example #3

$\therefore a = 4, b = 2, d = 1$ $b^d = 2 < a$ case 3

$\Rightarrow T(n) = O(n^{\log_b a}) = O(n^{\log_2 4})$

$= O(n^2)$

example # 4

for Gauss' recursive integer multiplication

$a = 3$, $b^d = 2$ $\quad a > b^d$ (case 3)

$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$

example 3.

Strassen's matrix multiplication.

$a = 7$, $b = 2$, $d = 2$

$b^d = 4 < a$ (case 3)

$\Rightarrow T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

$\Rightarrow$ beats naive iterative algorithm

example # 6

fictitious recurrence.

$T(n) \leq 2T(n/2) + \Theta(n^2)$

$\Rightarrow a = 2$, $b = 2$, $d = 2$

$b^d = 4 > a$ (case 2)

$\Rightarrow T(n) = O(n^2)$

Solving recurrences using master method.

i) $T(n) = 4T(n/2) + n$

so, $a = 4$, $b = 2$, $n^d = n$ $\therefore d = 1$

here, $a > b^d$ so case 3 is applicable.

$T_n = O(n^{\log_b a})$

$= O(n^{\log_2 4}) = O(n^{\log_2 2^2})$

$= O(n^2)$

ii) $T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2$, $n^d = n^3$ $\therefore d = 3$

here $a < b^d$ so case 2 is applied.

so, $T(n) = O(n^d)$

$= O(n^3)$

iii) $T(n) = T(2n/3) + 1$

here, $a = 1$, $b = 3/2$, $n^d = 1$ $\therefore d = 0$

here, $a = b^d$ so case 2 applies

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

$= O(\log n)$