9.3-5) Suppose you have "black-box" worst case linear time median subroutine

Give a simple linear time algorithm that solves the selection problem for an arbitrary order statistic

To make the randomized algorithm QUICKSORT deterministic. We modify the partitioning procedure of it such that it is guaranteed to run in $O(n \log n)$

```
QUICKSORT (A, p, r):
    if p < r:
        q = PARTITION (A, p, r)
        QUICKSORT (A, p, q-1)
        QUICKSORT (A, q+1, r)
```

```
PARTITION (A, p, r):
    x = A[r], i = p-1        → Problem of not able to find good pivot
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1
```

SOLUTION : Find good 'x'

↳ Eliminate the worst case scenario.

ORDER STATISTICS which uses this PARTITION SUBROUTINE can be made $O(n)$ by finding good partition point. This can be achieved by using MEDIAN as the partitioning element.

Let the black box median finding algorithm be MEDIAN (A, p, r)

then linear time algorithm that solves SELECTION is.

Order statistic problem which is EXPECTED LINEAR TIME algorithm.

```
RANDOMIZED-SELECT (A,p,r,i):
    if p==r :
        return A[p]
    q = RANDOMIZED-PARTITION (A,p,r)
    k = q-p+1
    if i==k :
        return A[q]
    elseif i<k :
        return RANDOMIZED-SELECT (A,p,q-1,i)
    else :
        return RANDOMIZED-SELECT (A,q+1,r,i-k)
```

Worst case : $O(n^2)$

Can be replaced by black box

$O(n)$ MEDIAN FINDING ALGORITHM

RUNNING TIME :

$$n + n/2 + n/4 + \cdots 1$$

$$= n\left(1 + 1/2 + 1/4 + \cdots + \frac{1}{2^{\log n}}\right)$$

$$= n\left(\frac{1-(1/2)^{\log n}}{1/2}\right) = O(n)$$

```
SELECT (A,p,r,i) :
    if p==r :
        return A[p]
    m = MEDIAN (A,p,r)
    q = PARTITION (A,p,r,m)
    k = q-p+1
    if i==k :
        return A[q]
    elseif i<k :
        return SELECT (A,p,q-1,i)
    else :
        return SELECT (A,q+1,r,i-k)
```

Given $O(n)$

$O(n)$

```
PARTITION (A,p,r,m) :
    for i = p to r :
        if A[i] == m :
            break
    [PIVOT @r]
    exchange A[i] with A[r]
    x = A[r], i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1
```

2) For n distinct elements $x_1, x_2, x_3, x_4 \cdots x_n$ with +ve weights such that $\sum_1^n w_i = 1$ the weighted lower median is the element x_k satisfying

$$\sum_{x_i < x_k}^{1} w_i < 1/2 \qquad \sum_{x_i > x_k}^{1} w_i \geq 1/2$$

a) Argue that the median of $x_1, x_2 \cdots x_n$ is the weighted median of the x_i with weights $w_1 = w_2 \cdots = w_n = 1/n$

b) Show how to compute weighted median of n elements in $O(n \log n)$ worst case using sorting

c) Show how to compute weighted median in $\theta(n)$ worst case time using a linear time median algorithm such as SELECT

a) When weights of all elements are equal i.e $w_i = 1/n \; \forall i \in (1, n)$

$\Leftrightarrow$ Lower median being element (x_k) s.t $\left(\sum_{x_i < x_k}^{1} w_i < 1/2 \right)$

For lower median with n elements

if n is odd then

$x_i \text{ ----- } [x_k] \text{ ---- } x_n$

$\left(\frac{n-1}{2}\right) \qquad \left(\frac{n-1}{2}\right)$

[x_k is LOWER MEDIAN]

if n is even then

$x_1 \cdots [x_k] \; x_{k+1} \cdots x_n$

$\left(\frac{n-2}{2}\right) \qquad \left(\frac{n-2}{2}\right)+1$

$$\sum_{x_i < x_k}^{1} w_i = \left(\frac{n-1}{2}\right)\left(\frac{1}{n}\right) = \frac{1}{2} - \frac{1}{2n} < 1/2$$

$$\sum_{x_i > x_k}^{1} w_i = \frac{1}{2} - \frac{1}{2n} < 1/2$$

$$\sum_{x_i < x_k}^{1} w_i = \left(\frac{n-2}{2}\right)\left(\frac{1}{n}\right) = \frac{1}{2} - \frac{1}{n} < 1/2$$

$$\sum_{x_i > x_k}^{1} w_i = \left[\left(\frac{n-2}{2}\right)+1\right]\frac{1}{n} = \frac{1}{2} \leq 1/2$$

Thus when weights are equal MEDIAN == WEIGHTED MEDIAN

b) Computing weighted median of n elements in $O(n \log n)$ using sorting.

Let $x_1, x_2 \ldots x_n$ be n elements with weights $w_1, w_2 \ldots w_n$ corresponding

$$\Rightarrow \quad \begin{matrix} x_1 & x_2 & \ldots & x_n \\ w_1 & w_2 & \ldots & w_n \end{matrix} \xrightarrow{\text{SORT}} \begin{matrix} x_2 & x_4 & x_7 & \ldots & x_3 \\ w_2 & w_4 & w_7 & \ldots & w_3 \end{matrix}$$

Now use the following procedure : Replace each w_i with $\sum_{j=1}^{i} w_j$

So, our weight array of elements shall be as follows.

x_2	x_4	x_7	. . . . .	x_3	} $O(n)$
w_2	$w_2 + w_4$	$w_2 + w_4 + w_7$		1	

After computing this, scan the weight array for an element at index j such that $W_{k-1} < 1/2$ & $(1 - W_k) < 1/2$

x_2	x_4	x_7	. . . . .	x_k	. . . . .	x_3
w_2	$w_2 + w_4$	$w_2 + w_4 + w_7$		W_{k-1} W_k		1

$+ w_k$

Here W_k is the sum of weights of all elements less than (k)

ALGORITHM :

Array of elements

WEIGHTED-MEDIAN (A, w) : Corresponding weights

```
SORT (A, w)      // Procedure sorts A & arranges elements of w
for i = 1 to n :
    W[i] = w[i-1] + w[i]
    if W[i-1] < 0.5 and (1 - W[i]) < 0.5 :
        return i ;
```

c) Computing weighted median in $\Theta(n)$ using algorithm such as SELECT.

Let A be the array of elements, W be the array of corresponding wt's
{ Initially it is (0.5)

```
WEIGHTED-MEDIAN (A, W, p, r, threshold) :

if p==r :                                   Algorithm starts with
    return A[p]                             p=1, r=n & threshold = 1/2

m = MEDIAN (A, p, r)
q = PARTITION (A, p, r, m)
LeftSumOfWeights = Σ_{i=p}^{q-1} W[i]
if LeftSumOfWeights > threshold     // WEIGHTED MEDIAN is to the left of MEDIAN
    return WEIGHTED-MEDIAN (A, W, p, q-1, threshold)
elseif LeftSumOfWeights < threshold
    if LeftSumOfWeights + W[q] ≥ threshold   // WEIGHTED MEDIAN ( Σ_{x_i<x_k} w_i < 1/2 & Σ_{x_i>x_k} w_i ≤ 1/2 )
        return A[q]                         // As we are decreasing no. of elements
    else :                                  // Threshold should also be decreased.
        return WEIGHTED-MEDIAN (A, W, q+1, r, threshold - LeftSumOfWeights - W[q])
```

8) A group of n ghostbusters is battling n ghosts. Each ghostbuster carries a proton pack which shoots a stream at a ghost, eradicating it. Stream (St. line) terminates when it hits the ghost. (Ghostbuster - Ghost) pair q start to shoot.

Pairings stream should not cross.

ASSUMPTIONS: Position of each ghostbuster and each ghost is a fixed pt. in plane q no three pts are collinear.

a) Argue that there exists a line passing through one ghostbuster and one ghost each that no. of ghostbusters on one side of the line equals no. of ghosts on the same side. Describe how to find such a line in $O(n \log n)$ time.

b) Give an $O(n^2 \log n)$ algorithm to pair ghostbusters with ghosts in such a way that no streams cross.

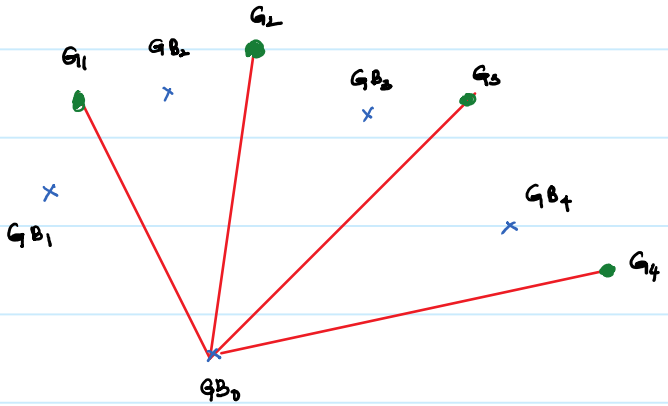
a) [Ghostbuster - Ghost] Pair $\rightarrow \exists$ line L (GB-G) s.t. $GB_L = G_L$ q $GB_R = G_R$

Lets prove this by CONTRADICTION i,e

There doesnot exist any line from ghostbuster to ghost such that No. of ghostbusters on one side of the line is same as No. of ghosts on the same side.

↓

For this to be true consider any RANDOM arrangements of point Without loss of generality assume GHOSTBUSTER be the lowest point in the arrangement.



[5GB, 4G]

Not possible with assumption of lines

(G_1) For the statement to be true, Line GB_0G_1 should have $\#GB_L \neq \#G_L$

So, $\#GB_L \geq 1$ → Consider least possible value i.e $\#GB_L = 1$, $\#G = 0$

(G_2) Lets look @ G_2GB_0 line

It should have $\#GB_L \neq \#G_L$ As $\#G_L = 1$ & $\#GB_L > 1$

$\#GB_L = 2$ (Least possible value)

(G_4) So considering line GB_0G_4

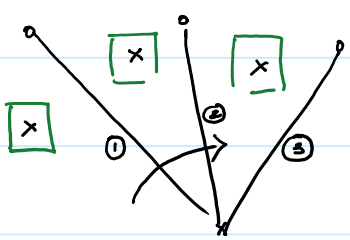
It should have $\#GB_L \neq \#G_L$, As $\#G_L = 3$, & $\#GB_L > 3$

$\#GB_L = 4$ (Least possible)

Thus for this arrangement there are 5 GB & 4 G [CONTRADICTION]

This approach of CONTRADICTION can be proved to n points as follows.

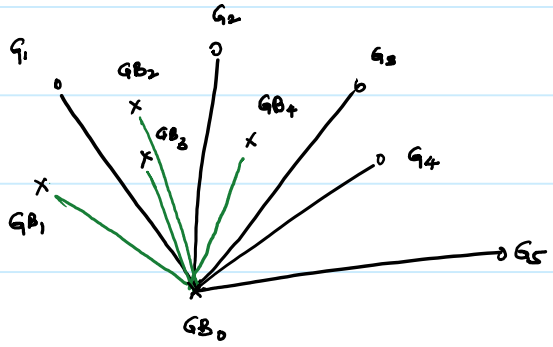
As we traverse from first line to next lines, between every line there has to be atleast one Ghostbuster



So, total Ghostbusters $\geq$ (# Ghosts) + 1

CONTRADICTION

Algorithm to find the line in $O(n \log n)$



① Consider slopes of lines from Ghostbuster to Ghost let them L_{GGB}

Sort the slopes of all lines $\rightarrow [L_{GGB}]$

Now to find a line with #GB = #G on one side of it

② Consider slopes of ALL lines from GB_0 to all GB_i's and G_j's

Let them be denoted by $[L_G]$

③

L_{GGB} : $[\ GB_0-G_1 \quad GB_0-G_2 \quad GB_0-G_3 \quad GB_0-G_4 \quad GB_0-G_5\ ]$

L_G : $[\ GB_0-GB_1,\ GB_0-G_1,\ GB_0-GB_2,\ GB_0-GB_3,\ GB_0-G_2,\ GB_0-GB_4,\ GB_0-G_3$

$GB_0-G_4,\ GB_0-G_5\ ]$

Maintain Array : 0 1 2 3 4

GB_q : [1 , 3 , 4 , 4 , 4]

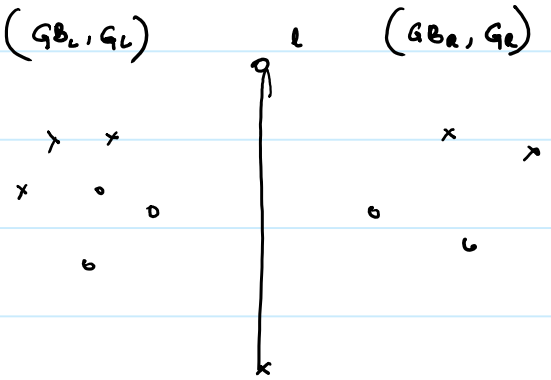
GB's before G

Find the index where $GB_q[i] = i$

As this procedure involves SORTING → its running time is $O(n \log n)$

To find all such lines we follow the following procedure :

Let GB_0, G_0 be the set of points.



Let l be the line that divides the set of points.

So, each line l divides set of points into non overlapping regions of points with no. of Ghostbusters on one side is same as no. of Ghosts.

Here the split into two sub regions may not be equal.

So,

$$T(n) = T(k) + T(n-k-1) + \Theta(n \log n)$$

Time to find (l) for each set of points.

Split into regions with (k) Ghostbusters & Ghosts

Split into regions with (n-k) Ghostbusters & Ghosts

Running time : $O(n^2 \log n)$ as $T(n) = T(0) + T(n-1) + \Theta(n \log n)$

$= T(n-1) + \Theta(n \log n)$

$\hookrightarrow O(n \log n \cdot n) = O(n^2 \log n)$

ALGORITHM

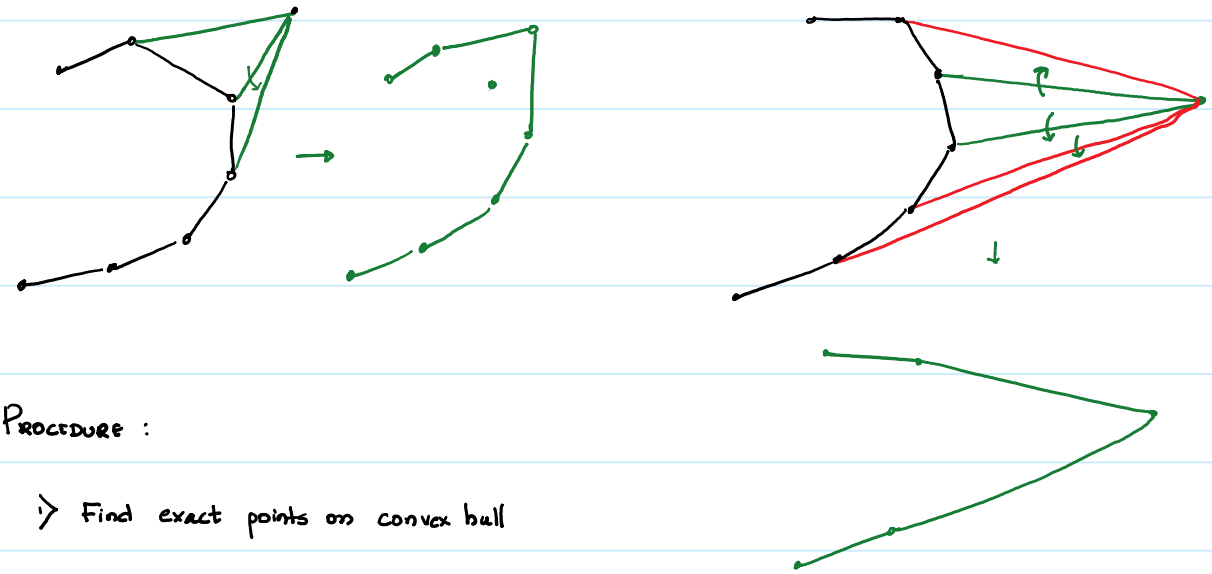
```
FIND-LINES (GB, G, ListOfLines)
    X ← Lowest point among all ⟨GB, G⟩
    L ← FIND-LINE (GB, G, X)
    ListOfLines.append (L)
    (GB_L, G_L) ← POINTS-TO-LEFT (L, GB, G)
    (GB_R, G_R) ← POINTS-TO-RIGHT (L, GB, G)
    FIND-LINES (GB_L, G_L, ListOfLines)
    FIND-LINES (GB_R, G_R, ListOfLines)
    return
```

(if X is assumed to be GB)

```
FIND-LINE (GB, G, X):
    L_GGB ← FIND-SORTED-SLOPES (X, G)
    L_G ← FIND-SORTED-SLOPES-ALL (X, G, GB)
    GB_G ← FIND-GB-BEFORE-G (L_GGB, L_G)
    for i in 1 to #GB_L:
        if GB_G[i] = i:
            return L_GGB[i]
```

33.3-5) In the online convex hull problem, we are given the set Q of n points one point at a time. After receiving each point, we compute the convex hull of the points seen so far. Obviously we can run Graham's scan once for each point with a total running time of $O(n^2 \lg n)$. Show how to solve this online convex hull problem in $O(n^2)$.

Logic : Points to be added to convex hull might resemble these figures



PROCEDURE :

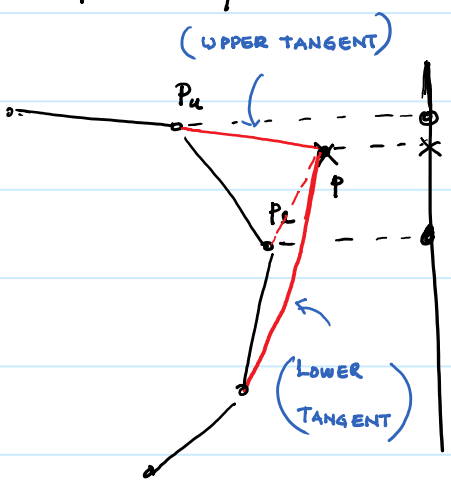
$O(\log n)$

1) Find exact points on convex hull where the new point lies between the projection of those points on y-axis. Let those points be P_u (UPPER), P_L (LOWER)

$O(n)$

2) Use P_u as reference start & find the UPPER TANGENT to convex hull from point (P)

3) Use P_L as reference start & find the LOWER TANGENT to convex hull from point (P)



RUNNING TIME : Worst-case all the elements on convex hull can be traversed for each point, during determination of UPPER & LOWER tangents $\Rightarrow O(n^2)$

C takes the old convex hull, new point (P)

ONLINE-CONVEX HULL (CH, P_0) : Cy returns updated convex hull.

(P_1, P_2) = BINARY-SEARCH (P_0) // As all points are arranged in increasing y-coo

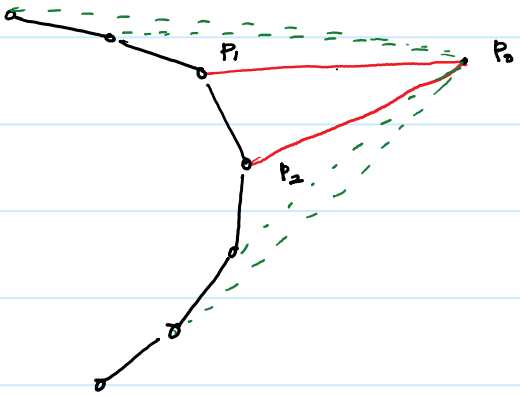
[Procedure to check P_0 is in next page]

if P_0 not inside CH :

CH = UPPER-TANGENT (CH, P_0, P_1)

CH = LOWER-TANGENT (CH, P_2, P_0)

return CH



UPPER-TANGENT (CH, P_0, P_1) :

$P_j = P_1$, // In computation of upper tangent we consider points as follows.

add P_0 to CH

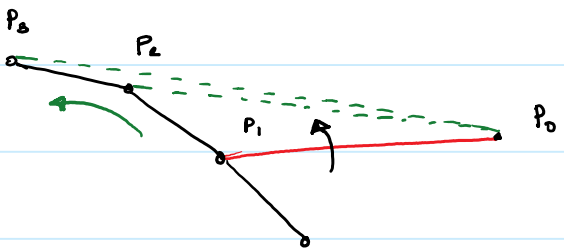
for pts P_i in CH after P_1 :

if $\langle P_i, P_1, P_0 \rangle$ make left turn

remove P_1 from CH

$P_j = P_i$

else : break.



LOWER-TANGENT (CH, P_0, P_2) :

$P_j = P_2$

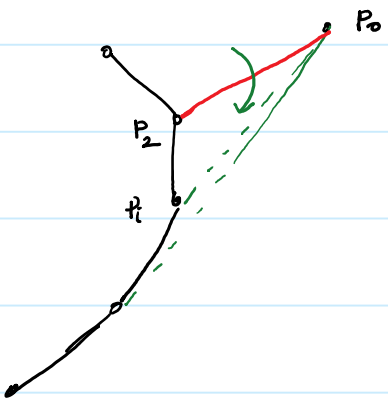
For pts P_i in CH after P_2 :

if $\langle P_i, P_2, P_0 \rangle$ make right turn :

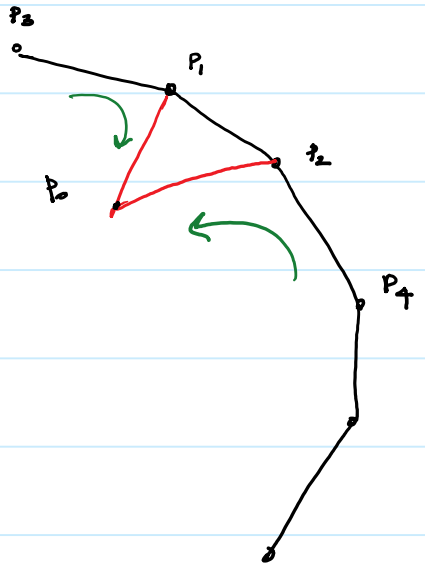
remove P_2 from CH

$P_j = P_i$

else : break



To check if pt is inside CH



$\langle P_3, P_1, P_0 \rangle$, $\langle P_4, P_2, P_0 \rangle$

both make left turns

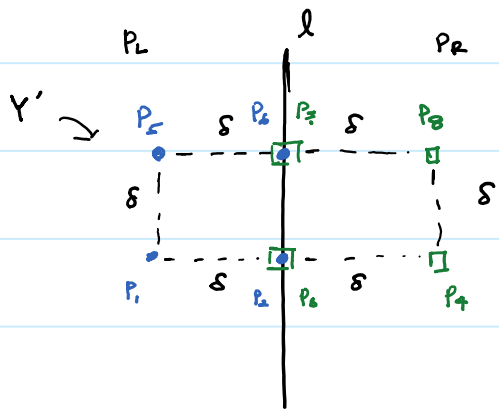
thus proving pt is inside convex hull

For these points we donot have to alter convex hull.

33.4-1 > Professor Williams comes up with a scheme that allows closest pair algorithm to check only 5 points following each point in array Y'. The idea is to place points on line l into set P_L. Then there cannot be pairs of coincident points on line L with one point in P_L and the other in P_R thus atmost 6 points can reside in $\delta \times 2\delta$ rectangle

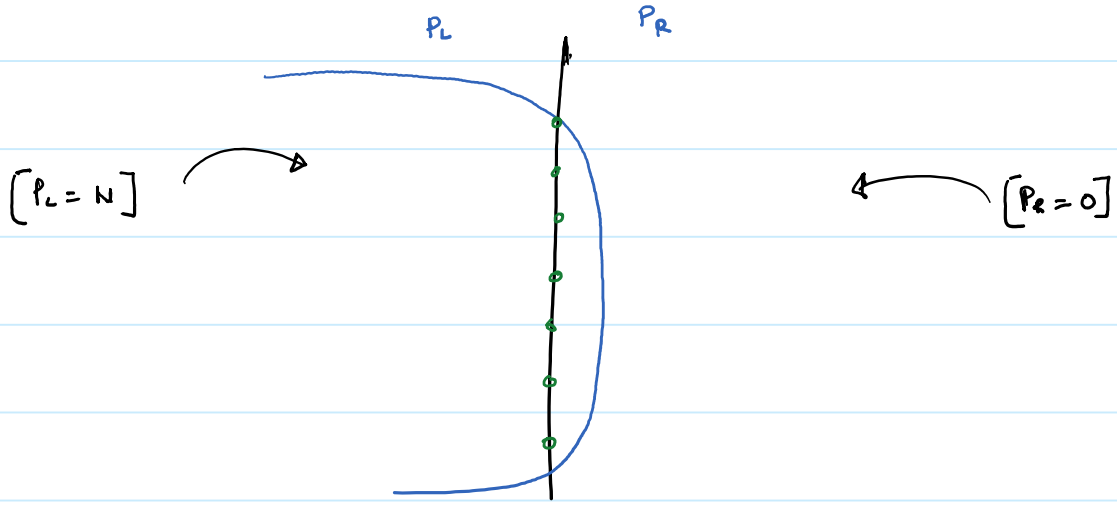
What is the flaw in the professor's scheme ?

Y' is the array of points sorted along y-coordinates that lie in the 2δ region



With distance between pts atleast (δ) there can be max (4) pts in each box

FLAW : There can be an arrangement as follows which proves the approach wrong



Including all the points into P_L makes P_R empty

Thus Divide and Conquer doesnot reduce the problem and approach fails.