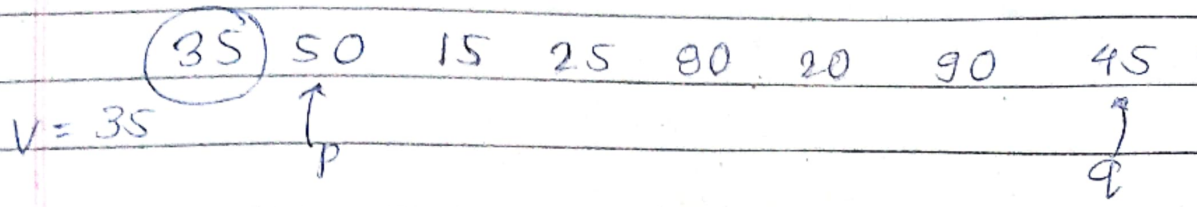
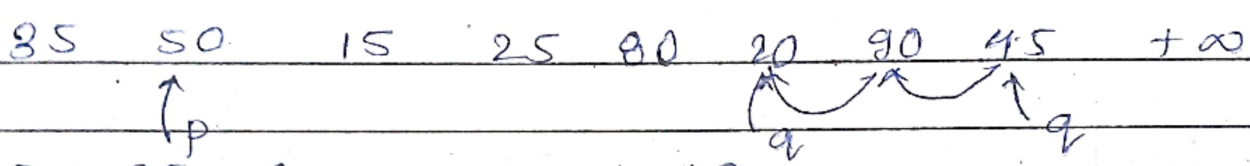
Divide & Conquer

```
DAC(P). DACCP) {
    if (small(P)){
        S(P)
    }
    else{
divede P into P1, P2, P3 --- Pk
Apply DAC(P1), DAC(P2) ---- DAC(Pk)
Combine (DAC(P1), DAC(P2) --- DAC(Pk))
    }
}
```

QuickSort (DAC)

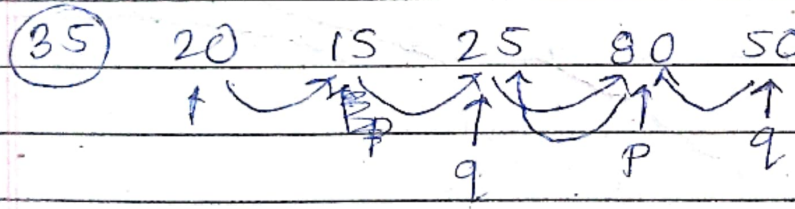


p goes towards the right & q goes to left,
p stops when it finds element greater than ~~else~~ pivot & q stops when it finds element less than (equal to) pivot



50 > 35, ∴ p stops at 50
45 > 35, ∴ q goes left
90 > 35, — " —
20 < 35, ∴ q stops

check whether p & q have crossed each other or not, & swap elements at p & q
∴ Swap 50 & 20

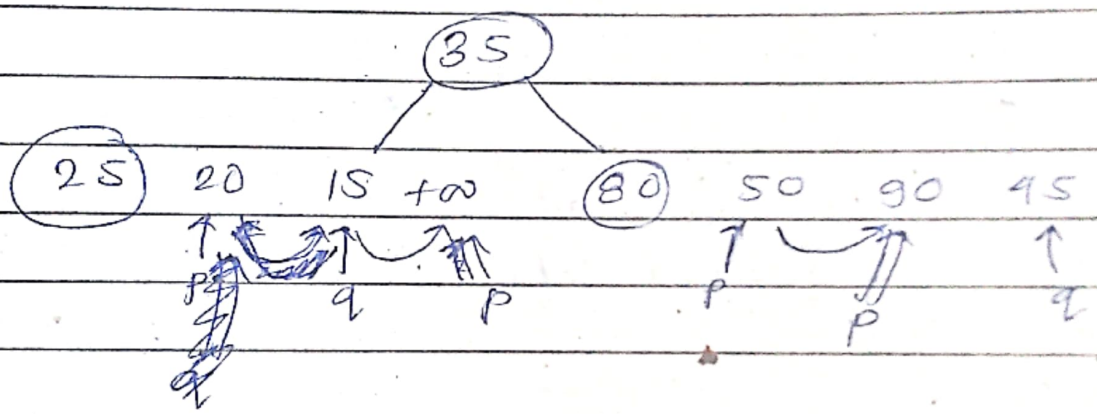


20 < 35, ∴ p goes right
15 < 35, — " —

Now, whenever p & q cross each other replace pivot element with element at position q

25 20 15 (35) 80 50 90 45

Now we have completed the 1st pass.
Now all the elements less than pivot are to the left of it & those greater than pivot are to the right of it.
Now, the problem is divided into 2 parts now apply quicksort to the left & right parts ~~rep~~ resp.

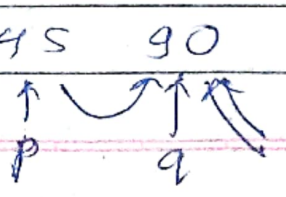


20 < 25, p goes right 15 < 25, — 11 — +∞ > 25, p stops	q stops at 15

Now as p & q have crossed each other
Swap p & q

50 < 80, p goes right 90 > 80, ∴ p stops	45 < 80, q stops

Now swap 90 & 45 as p & q haven't crossed each other



90 > 80, p stops 45 < 80, q stops
q

80 50 45 90 +∞
   ↑↑ ↑p
   q

As p & q have crossed each other swap 45 & 80

∴ Array now

15 20 25 35 45 50 80 90

~~Time complexity~~:

Recurrence rel^n:
- $T(n) = 2T(n/2) + n$ - Best case
- Avg. case: $T(n) = T(N/9) + T(9N/10) + \theta(n)$
- $T(n) = T(n-1) + \theta(n)$ - Worst case

```
Partition (l, h){
  pivot = A[l];
  i = l, j = h;
while( i < j){
  do{
    i++;
  } while (A[i] < pivot)

  do{
    j--;
  } while (A[j] > pivot)

  if (i < j)
      swap (A[i], A[j])
}
```

```
swap(A[i], A[j]);
return j;
}

Quicksort(l, h){
    if(l<h){
        j = partition(l, h);
        Quicksort(l, j);
        Quicksort(j+1, h);
    }
}
```

$\boxed{T(n) = O(n \log n)}$ (Best case)

Always the partitioning should be in the middle

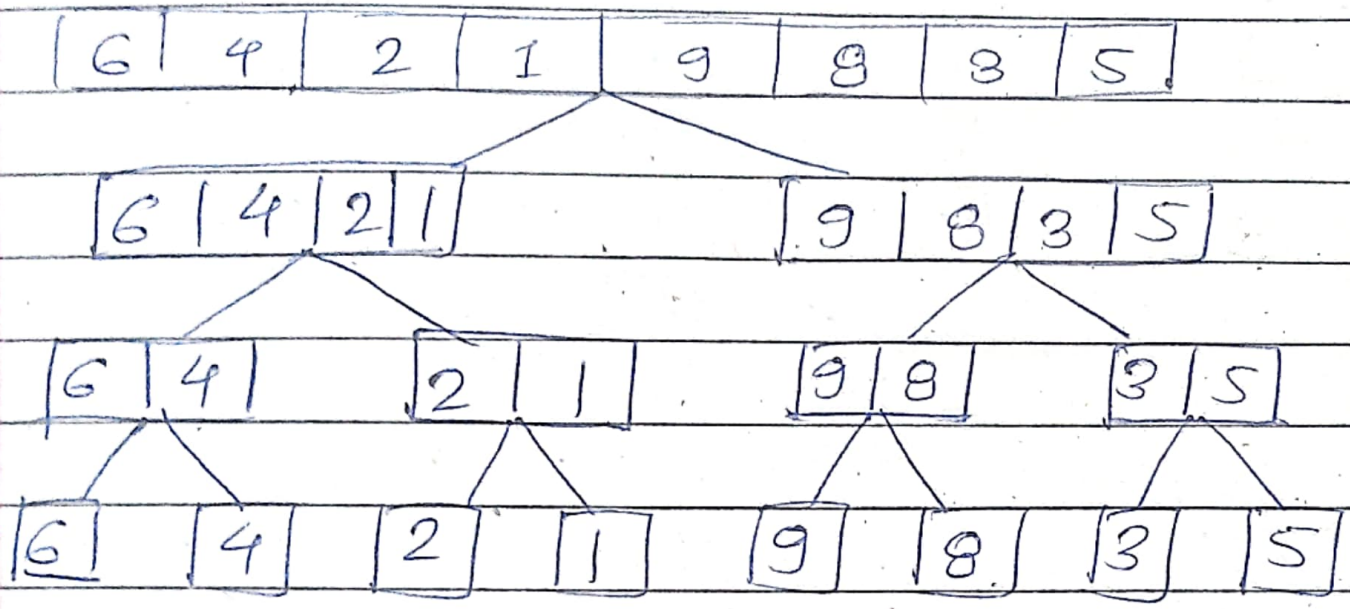
Worst Case $\boxed{T(n) = O(n^2)}$

Test is ~~abc~~ already sorted

Removing Worst case

→ Select middle element as pivot
→ Select random element as pivot.

Merge sort:



Keep on dividing the array until there is only one element left.

Now start merging

Page No:
Date:

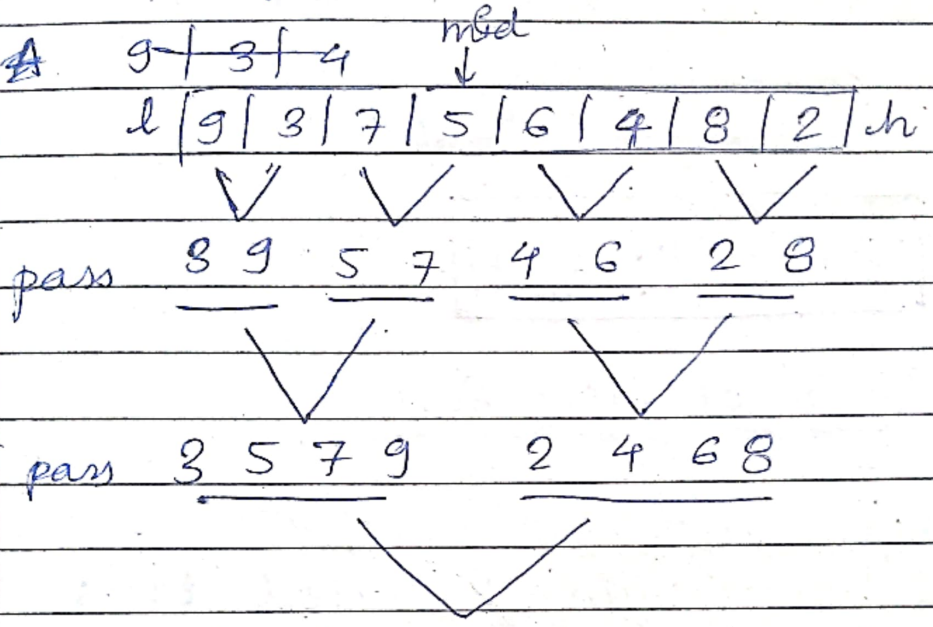
Merging

A	B	C
2	5	2
8	9	5
15	12	8
18	17	9
		12
		15
		17
		18

```
Merge (A[], B[], m, n){

    i=1, j=1, k=1

while(i <=m && j<=n){

    if(A[i] < B[j])
        C[k++] = A[i++];

    else
        C[k++] = B[j++];

        }
}
// If there are any remaining elements
// in any of the lists

for(;i<=m;i++){
        C[k++] = A[i]
```

```
for (; j<=n; j++)
    C[k++] = B[j];
}
```

2-way Merge Sort



No. of passes req^d for n elements = $\log_2 n$

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

```
MergeSort(l, h) {                    — T(n)
    if (l < h) {
        mid = (l+h)/2;               — 1
        MergeSort(l, mid);           — T(n/2)
        MergeSort(mid+1, h);         — T(n/2)
        Merge(l, mid, h);            — n
    }
}
```

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

$a = 2, \quad b = 2$

$f(n) = n \qquad \log_b a = \log_2 2 = 1$

$n^k = n^1$

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

Pros & Cons of MergeSort

Pros:

→ Large size list
→ Preferable for merging 2 sorted ~~ll's~~ linked lists
→ External sorting

RAM

f1	f2
10	5

RAM
4GB

f1	f2	f3
2	3	2
10	5	3
20	15	:

5GB 10GB
HD

4) Stable → Relative ordering of nos/data is maintained

Page No.:
Date: / /

Cons:

1) Extra-space (not in place sort)
2) No small problem
3) Recursive Algorithm so req. stack. so req. $\log_2 n$ space in stack (which is the ht. of binary tree)

$\therefore$ Space Complexity = $O(n + \log_2 n)$

Strassen's Matrix Mul.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

2×2 $\quad$ 2×2

$$c_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

```
for(i=0; i<n; i++){
    for(j=0; j<n; j++){
        c[i,j]=0
        for(k=0; k<n; k++){
            c[i,j]+=A[i,k]*B[k,j];
        }
    }
}
```

@ $c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$

$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$

$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$

$c_{22} = a_{21} * b_{21} + a_{22} * b_{22}$

$A =$

a_{11}	a_{12}	a_{13}	a_{14}
A_{11}		A_{12}	
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
A_{21}		A_{22}	
a_{41}	a_{42}	a_{43}	a_{44}

$B =$

b_{11}	b_{12}	b_{13}	b_{14}
B_{11}		B_{12}	
b_{21}	b_{22}	b_{23}	b_{24}
b_{31}	b_{32}	b_{33}	b_{34}
B_{21}		B_{22}	
b_{41}	b_{42}	b_{43}	b_{44}

Algorithm . MM (A, B, n)

```
if (n <= 2) {
    c = 4 formulas
    }
else {
    mid ---- n/2
MM(A11, B11, n/2) + MM(A12, B21, n/2)
MM(A11, B12, n/2) + MM(A12, B22, n/2)
MM(A21, B11, n/2) + MM(A22, B21, n/2)
MM(A21, B12, n/2) + MM(A22, B22, n/2) } }
```

$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$
$C_{12} = A_{12} * B_{12} + A_{12} * B_{22}$
$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$
$C_{22} = A_{21} * B_{21} + A_{22} * B_{22}$

$$T(n) = \begin{cases} 1 & n < 2 \\ \boxed{8T(n/2) + n^2} & n > 2 \end{cases}$$

$a = 8 \quad b = 2 \quad \log_b a = \log_2 8 = 3$

$f(n) = n^2 \quad n^k = n^2 \quad k = 2$

$\boxed{T(n) = \theta(n^3)}$

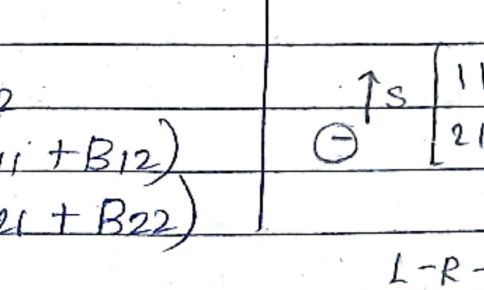
Strassen's Matrix Multiplication

We reqd 8 multiplications in normal Matrix Multiplication but in Strassen's Matrix Multiplication we require 7

$P = (A_{11} + A_{22}) * (B_{11} + B_{22})$
$Q = (A_{21} + A_{22}) * B_{11}$
$R = A_{11} * (B_{12} - B_{22})$
$S = A_{22} * (B_{21} - B_{11})$
$T = (A_{11} + A_{12}) * B_{22}$
$U = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$V = (A_{12} - A_{22}) * (B_{21} + B_{22})$

Inside A, Bracket B
vice-versa

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + U$



$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

$\log_2 7 = 2.81 \qquad K = 2$

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

Dynamic Programming: It divides the problems into series of overlapping sub-problems

Two features:

1) Optimal substructure
2) Overlapping sub-problems

Pros	Cons
→ Faster: TC of $O(n^{\log 2(7)})$ over $O(n^3)$ for normal	→
→ Uses less memory	
→ reduces no. of intermediate matrices	
→ Easily parallelizable allowing multiple processors to work on different parts of multiplications	
→ Useful in image process. & signal process.	
→ Useful in design of FFT	

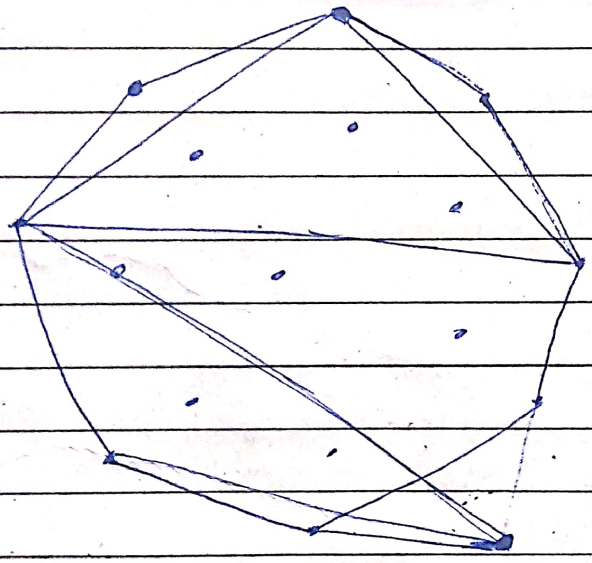
Divide & Conquer

Convex Hull

Convex hull is the smallest convex polygon that contains all the points in the plane.

→ Given a set X of points $P_1(x_1, y_1) \cdots P(x_n, y_n)$ in plane we want to find the convex hull.

→ Divide & conquer algorithm takes $O(n \log n)$ time to compute convex hull in clockwise order.



Algorithm:

→ If the number of pts. is 1 or 2 return the set of points as convex hull.

→ Divide the points into 2 subsets of roughly equal size by splitting the points along the median x-coordinate

→ If X_1 is empty then upper hull is simply line with endpoints P_1 & P_n

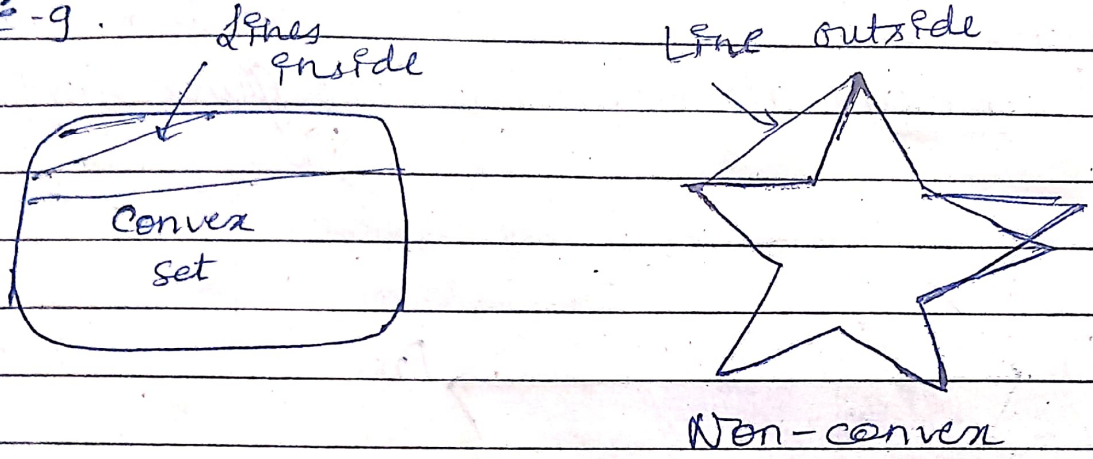
→ If X_1 is not empty then find P_{max} in X_1 which is farthest from line P_1P_n.

4) If lies a point outside P_1P_n then the pt. of maximum angle $\angle P_{max}P_1P_n$ is selected

5) Now Algo identifies all ~~pt~~ points of x, that are a list of line $\overline{P_1P_{max}}$ goto step 1.

Convex: A set of points (finite or ∞) in the plane is called convex if for any 2 pts. P & Q in the set, the entire line segment with the endpoints at P & Q belongs to the set.

E-g.



Convex Hull: The convex hull of a set S is the smallest convex set containing S.

Convex Polygen: Convex Hull of any set of $n > 2$ points (not all on the same line) is a convex polygen with vertices at some of the points of S. (If all points lie on the same line, the polygen degenerates to a line segment)

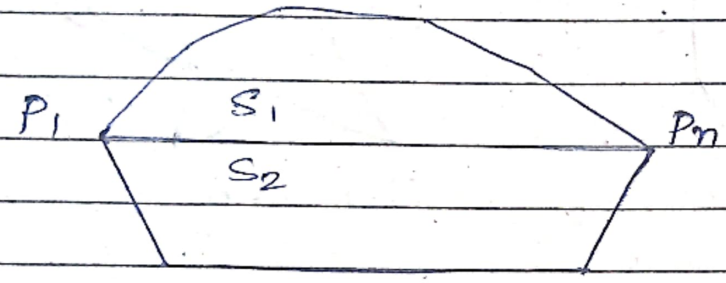
Convex hull problem: Problem of constructing convex hull for a given set S of n points. To solve it points that will serve vertices of polygen are found.
Vertices of such points are called "Extreme Points"

→ Extreme pts: point of a convex set is a point of this set that is not a middle pt. of any line segement with endpoints in the set.

Uses: Geographic info. systems
Image processing
Pattern recognition
Game Theory

Quick hull

Quick hull algo. (Similar to Quick Sort)



→ Let $P_1 = (x_1, y_1) \cdots P_n = (x_n, y_n)$ be set S of $n > 1$ points

→ Assume pts. to be sorted in increasing order of their x-co-ordinates, ties resolved by inc. order of y-co-ordinates of the pts. involved.

→ P_1 & P_n are 2 distinct extreme points of sets convex hull

→ They define seperates points of S into 2 sets S_1 (pts. to left), S_2 (pts. to right)

→ The points of S on line $P_1 P_n$ other than P_1 & P_n cannot be extreme pts. & thus excluded from further consideration

→ The boundary of convex hull of S is made of 2 polygon chains called Upper & Lower Hull.

→ Upper Hull → sequence of line segments with vertices at P_1, pts. in S_1 & P_n

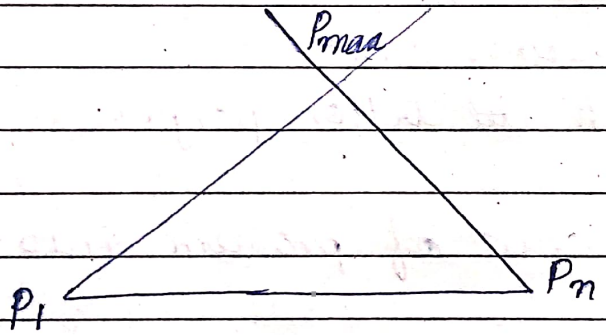
→ Lower Hull: sequence of line segments with vertices at P_1, points on S_2 & P_n

→ Upper & lower hull are constructed independently by Quick Hull Algorithm.

Constructing Upper Hull ~~using~~ (Quick Hull)

→ Point P_{max} vertex is identified in S_1 (of S_1 not empty) - farthest from line P_1P_n. If there is a tie, the point maximises the $\angle P_{max}P_1P_n$.

→ Algorithm identifies all pts. of set S_2, that are to the left of line P_1P_{max} make up set $S_{1,1}$.



→ ~~Alt~~ Algorithm identifies all points of set S_1 that are to the left of line $P_{max}P_n$ along with P_{max}, P_n make up the set $S_{1,2}$.

→ Algorithm can continue constructing Upper Hulls of $P_1 \cup S_{1,1} \cup P_{max}$ then join them & $P_{max} \cup S_{1,2} \cup P_n$ ~~respectively~~ recursively & to get upper hull of set $P_1 \cup S_1 \cup P_n$.

Checking whether the point lies to the left of line or not

Area of $\Delta P_1P_2P_3$ is 1/2 the mod mag. of determinant

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

→ Sign of this expression is +ve iff pt. $P_3 = (x_3, y_3)$ is to left of line P_1P_2

→

Time Complexity:

$T(n) = O(n^2)$ → worst case

Avg. case - much better perf.

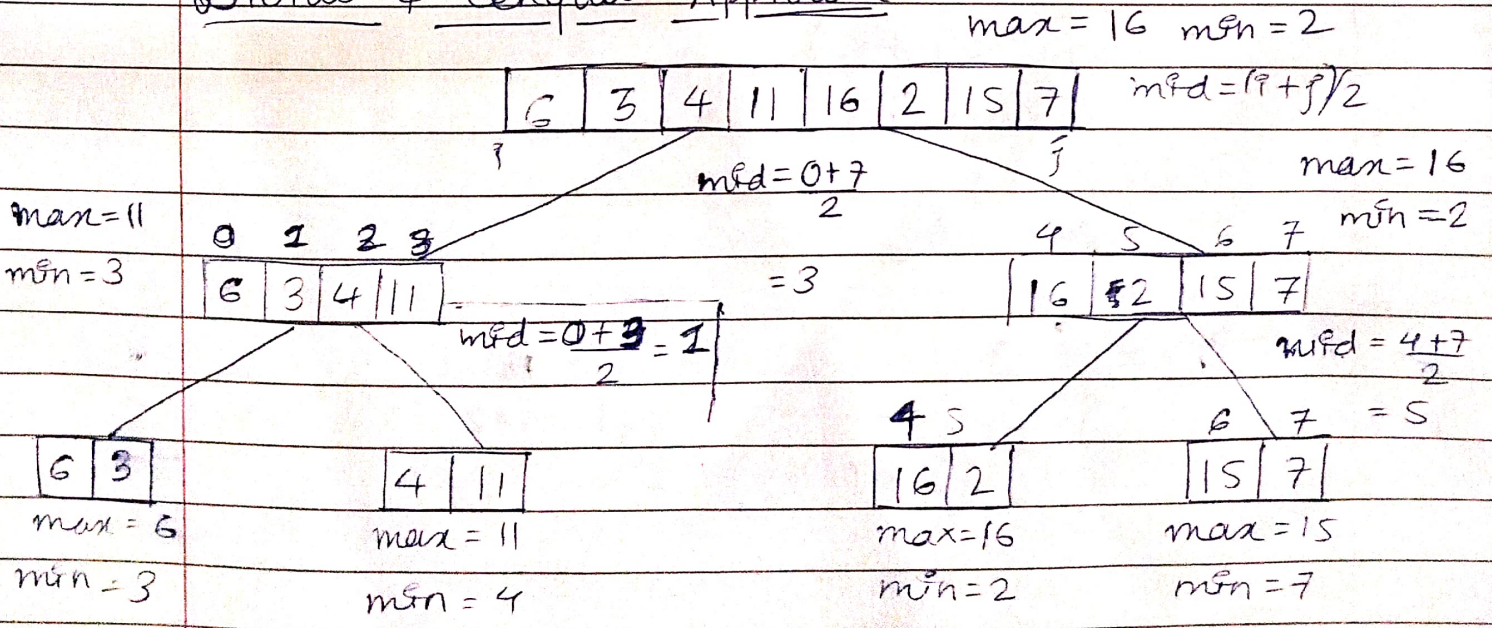
1) Avg. balanced split of problem into smaller sub-problem
2) Seq. fraction of pts. ~~inside~~ inside $\Delta P_1P_{max}P_r$ are eliminated

Finding Maximum & Minimum using Divide & Conquer.

Naive approach:

```
Find Max_Min (A, n) {
    max = min = A[0];

    for (i = 1; i < n; i++) {
        if (max < A[i])
            max = A[i]

        else if (min > A[i])
            min = A[i]

    }

    return (max, min);
}
```

Divide & Conquer Approach:



```
DAC Max_min (A, i, j, max, min)
{
    if (i == j)
        max = min = A[i]

    else if (i == j-1) {
        if (A[i] < A[j])
            max = A[j], min = A[i]
        else {
            max = A[i], min = A[j]
        }
    }
    mid = (i + j)/2

    DAC & Max_Min(A, i, mid, max, min)
    DAC  Max_Min(A, mid+1, max, min)

    if(max1 < max2)
        max = max2
    else
        max = max1

    if(min1 < min2)
        min = min1
    else
        min = min2
```

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

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

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

$\therefore T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + 2\right] + 2 = 2^2 T(n/2^2) + 2^2 + 2$

$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 2\right] + 2^2 + 2$

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

$\vdots$

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

At $n = 2$, $T(n) = 1$

$\therefore \frac{n}{2^k} = 2$ (∵ value becomes 2 at $n=1$)

$\therefore n = 2^{k+1}$ $\quad \therefore \frac{n}{2} = 2^k$

$= 2^k \times 1 + 2^k + 2^{k-1} + \cdots + 2^2 + 2$

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

$= 2^k + \frac{2(2^k - 1)}{2-1}$ $\quad \cdots \quad \frac{a(r^n - 1)}{r-1}$

$= 2^k + 2^{k+1} - 2$

$= \frac{3n}{2} + n - 2 = \frac{3n}{2} - 2$

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