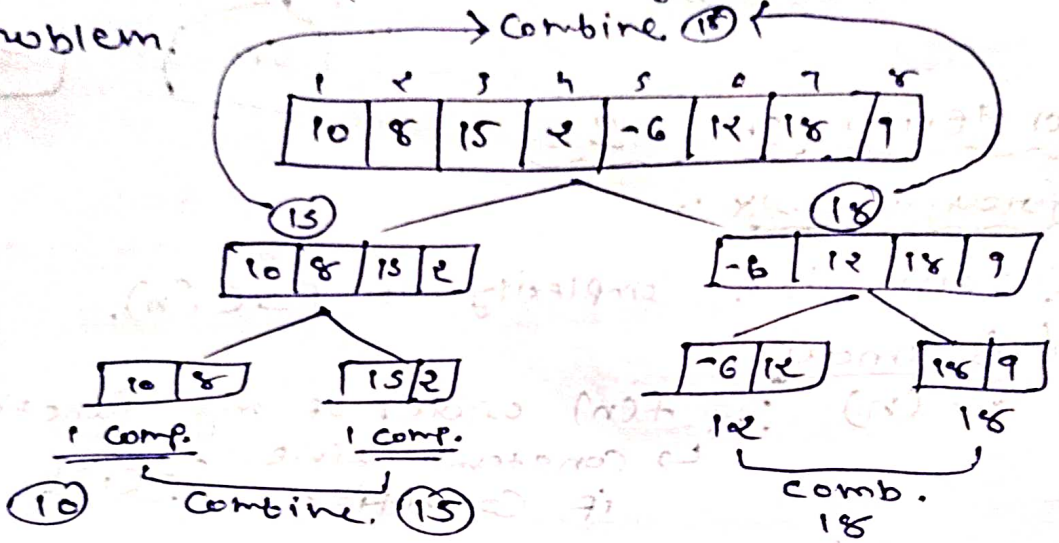
# Design Strategies.

## ⇒ Divide & Conquer : (D&C)

⇒ When Problem become large / Complex. then divide Problems into sub problem. until sub Problem become small.

* then the solve (Solver) smaller problem combine the Result Require. to get Solution of original Problem.




⇒ Control Abstraction :.

```
Algorith DAndC (P)
{ if (small(P)) then return S(P);
  else
  { divide P into smaller instances P1, P2 ... Pk K≥1
    Apply DAndC to each of these subproblems.
    Return Combine( DAndC(P1), DAndC(P2), ....
                    DAndC(Pk));
  }
}
```

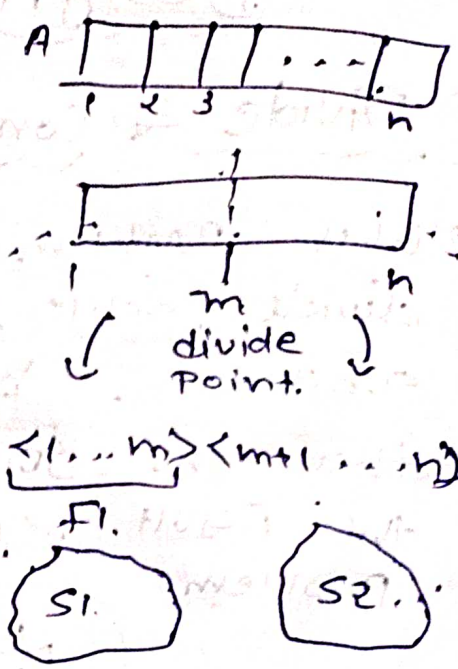
Divide problem in K parts.

Recursively.

*) Processor of flow is clear.
*) Details Number of steps is Abstract.

} Control Abstraction.

```
Algorithm D&C (A, l, n)
{
  if (small (l, n))
      return (s(A, l, n));
  else
  {
    m = Divide (l, n)
    S1 = D&C (A, l, m); → Fl.
    S2 = D&C (A, m+1, n);
    Combine (S1, S2);
  }
```




## Time complexity of D&C framework :-

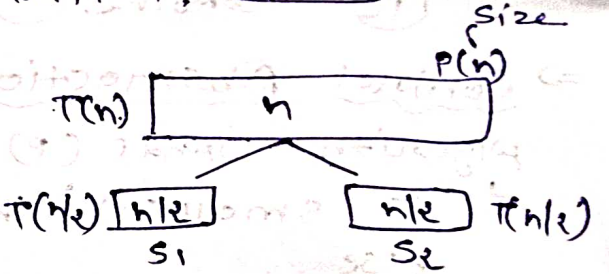
⇒ let T(n) repr. Time Complexity of D&C (n).

*) if 'n' is small
T(n) = f(n) ∴ f(n) order of my function S.
↳ Constant time.
if Condition, O(1).

*) else 'n' is large

↳ T(n/2) + T(n/2)
S1 S2

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




$T(n) = 2T(n/2) + g(n)$ ← time (Divide + Combine + small)

$T(n) = a \cdot T(n/2) + g(n)$

a: No of subproblem. n/2: size of each problem. g(n): Time Divide & Combine.

Generalized $T(n) = a \cdot T(n/b) + g(n)$

a: No of sub problem. n/b: size of each problem.

$a \geq 1$
$b > 1$
$g(n)$ +ve.

$T(n)$

| 1/3 | 2/3 |
|---|---|

$^{n}$ size diff.

$T(n) = T(n/3) + T(2n/3) + g(n)$ // here $n \geq 1$.

General $T(\alpha n) + T((1-\alpha)n) + g(n)$ $\quad 0 < \alpha < 1$

*) D&C :=> Divide is mandatory.
But Combine is optional.

*) Ex :. Binary search

---

① MaxMin :. Procedure to find max & min. (simultaneously).

```
max = min = A[1]
for i = 2 to n ---> n-1
{ if (A[i] > max) (1)
      max = A[i]
  if (A[i] < min) (1)
      min = A[i]
}
```

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 8 | 16 | 0 | -5 | 18 | 2 | 9 | 60 | 30 | 25 |

=> The total # of element comparisions made in none-D&C : total :. $2(n-1)$
$= 2n-2$
All cases

=> Every itteration में 2 comparision.
Change it into if else :.

```
max = min = A[1]
for i = 2 to n ---> n-1
{ if (A[i] > max) (1)
      max = A[i];
  else
  if (A[i] < min) (1)
      min = A[i];
}
```

(i) Best case :.
element in increse order. $O(n-1)$

(ii) worst case :.
element in decreje order. $2(n-1)$.

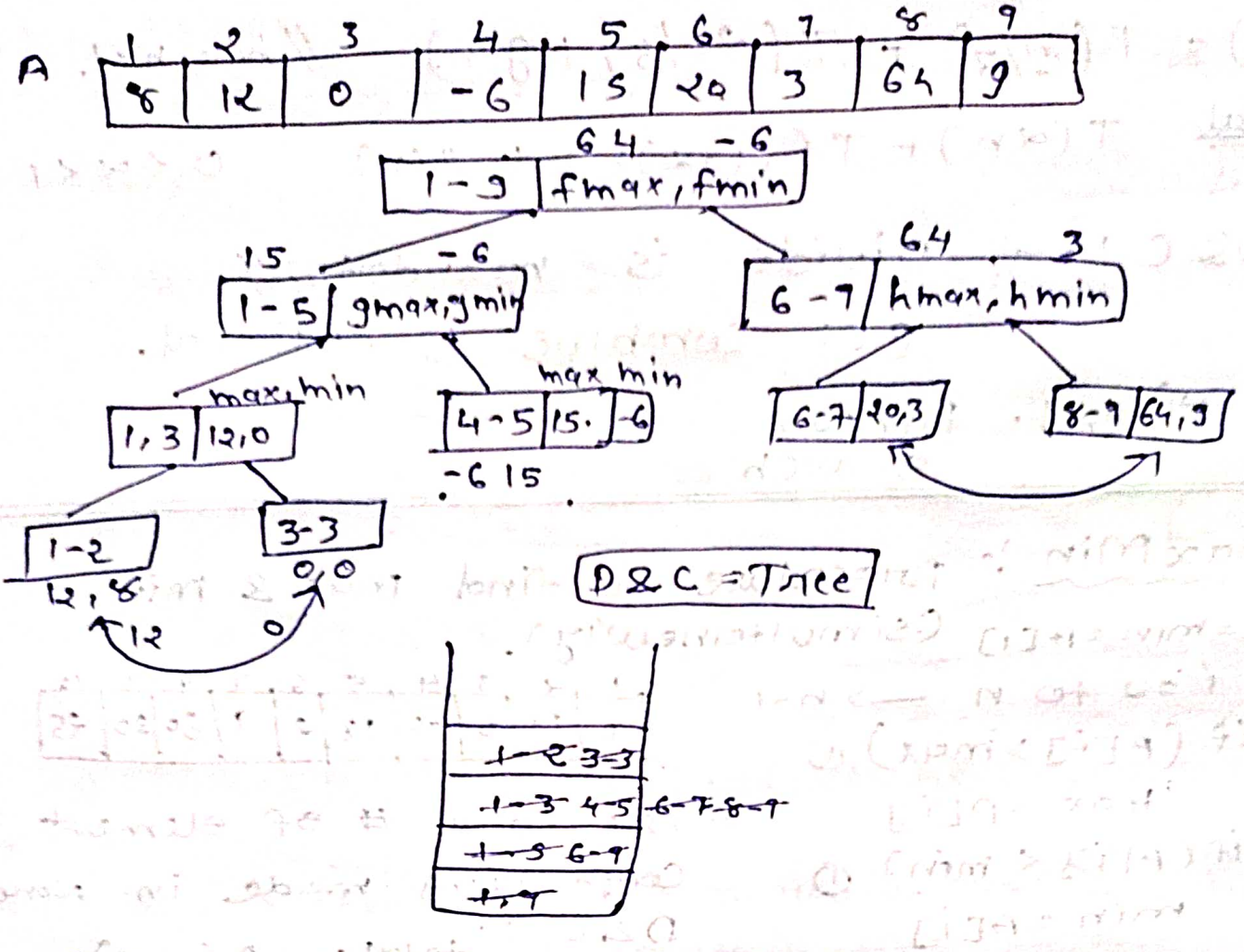
Note :. in Best case 1st if always be true. so inc. order.

| | 1st Comp | 2nd comp. | total. |
|---|---|---|---|
| ① Best Case (increasing) | $(n-1)$ | 0 | $(n-1)$ |
| ② worst case decreje | $(n-1)$ | $(n-1)$ | $2(n-1)$ |
| ③ Avg Case. | $(n-1)$ | $n/2$ | $(\frac{3n}{2}-1)$ |

## II) Divide & Conquer :-

A

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 8 | 12 | 0 | -6 | 15 | 20 | 3 | 64 | 9 |




D&C = Tree




⇒ Performance of Divide & Conquer.

i) No of element comp's :-

T(n) Repr No of element 'n' element.

$T(n) = 0, \ n = 1$
$= 1 \ \ n = 2$ } small.
$= 2T(n/2) + 2$

$T(n) = 2T(n/2) + 2$ ——— (1)

$T(n/2) = 2T(n/4) + 2$ ——— (2)

$T(n) = 2[2T(n/4) + 2] + 2$

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

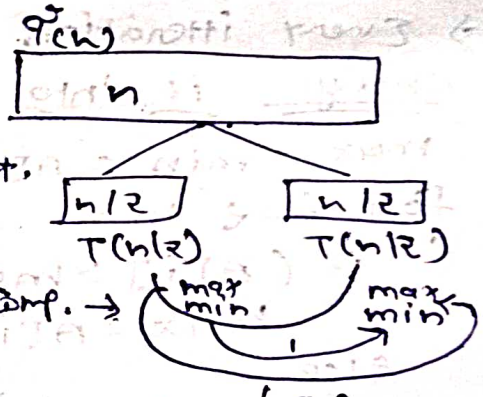
$= 2^2 T(n/2^2) + 2^2 + 2^1$

$\vdots$

$2^K + T(n/2^K) + \sum_{i=1}^{K} 2^i$

$= \frac{a(r^n - 1)}{r - 1} = \frac{2(2^K - 1)}{2 - 1} = \boxed{2^{K+1} - 2}$

Number of comparison




$\frac{n}{2^K} = 2 \quad \frac{n}{2} = 2^K$

$= \frac{n}{2} \cdot T(2) + 2^{K+1} - 2$

$= \frac{n}{2} + 2 \cdot 2^K - 2$

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