Types of recursions –

① Linear

② Divide & conquer

→ **DIVIDE & CONQUER RECURRENCES** :

dividing in sub-problems

form : $\underbrace{T(x)}_{\text{whole problem}} = \underbrace{a_1 T(b_1 x + \varepsilon_1(x))}_{\text{conquer}} + a_2 T(b_2 x + \varepsilon_2(x)) + \dots + a_k T(b_k x + \varepsilon_k(x)) + \underbrace{g(x)}_{\text{combine}}$

form of program

time complexity required to do something with previous tasks/calls.

$x \geq x_0$ constant

Relating this form to binary search formula/relation,

$f(n) = f\left(\frac{n}{2}\right) + \underbrace{c}_{\text{constant}}$

Here, $a_1 = 1$, $b_1 = \frac{1}{2}$, $g(n) = c$

$\varepsilon_1(n) = 0$

→ D&C is a recursive strategy.

→ This technique divided into three parts –

① **Divide** – dividing problems into smaller sub-problems.

② **Conquer** – solve sub-problems by calling recursively until solved.

③ **Combining** – combine sub problems to get final soln of whole problem.

* Sub-problems should be same as main problem, for example if the main problem is of sorting, so the Sub-problems should also be sorting only which can be works recursively.
It should not be like you're doing another thing instead of sorting in sub-problems. That shouldn't be count in Divide & conquer technique.

GENERAL METHOD FOR DIVIDE & C.

```
D&C(P){
 if (small(P)){ //P is small
     Solve(P);}
else{
  divide P into P1, P2, P3,.....Pk
  Apply D&C(P1), D&C(P2)....
  combine(D&C(P1), D&C(P2),...)
}}
```

Recurrence Relation for Dividing fn

* $T(n) = 2T(n/2)+1 \rightarrow O(n^1 \log^0 n) \rightarrow O(n)$
* $T(n) = 4T(n/2)+n \rightarrow O(n^2)$
* $T(n) = 2T(n/2)+n \rightarrow O(n \log n)$
* $T(n) = 4T(n/2)+n^2 \rightarrow O(n^2 \log n)$
* $T(n) = 4T(n/2)+n^2 \log^2 n \rightarrow O(n^2 \log^3 n)$

Example of computing

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

Applying values of a & b in cases sol^n

$a = 2$, $b = 2$ $\quad f(n) = O(1)$
$= O(n^0 \log^0 n)$
$K = 0, \; P = 0$

from this,
$\log_2 2 = 1, K = 0$

case 1) $O(n^{\log_b a}) \Rightarrow O(n^{\log_2 2}) \Rightarrow O(n^1)$

MASTER THEORAM for dividing fns

General form $\Rightarrow T(n) = aT(n/b)+f(n)$

$a \geq 1$, $b > 1$, $f(n) = O(n^k \log^p n)$

from eq^n

Case 1 - if $\log_b a > K$
then $O(n^{\log_b a})$

Case 2 - if $\log_b a = K$
if $p > -1 \rightarrow O(n^k \log^{p+1} n)$
if $p = -1 \rightarrow O(n^k \log \log n)$
if $p < -1 \quad O(n^k)$

Case 3 - if $\log_b a < K$,
if $p \geq 0 \rightarrow O(n^k \log^p n)$
if $p < 0 \rightarrow O(n^k)$