Recursion :

- It is a programming paradigm which will solve a bigger problem by solving smaller instance of the exact same problem

To solve any recursive problem,

(1) Assumption ↓ In this order
(2) main logic
(3) Base condition

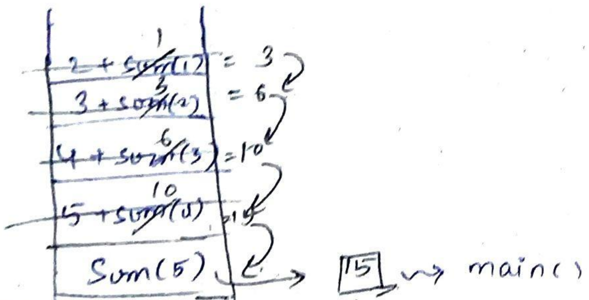
Ex : Sum of 5 natural numbers,

sum(5) = 5 + sum(4)
= 5 + 4 + sum(3) } Breaking the bigger problem to smaller problem

```
int sum(int N)      → Sum of N natural no.s   (1) Assumption
{
    if(N==0)                                  (3) Base condition
        return 0;

    return N + sum(N-1);                      (2) main logic
}
```

Call stack



=> (N!)

```
int factorial(int N)     (1) Assuming to find factorial
{
    if(N==0)
        return 1;        (3) Base condition

    return N * factorial(N-1);
                         (2) main logic
}
```

=> 1 1 2 3 5 8 13 ---- (fibonacci)
1 2 3 4 5 6 7 ......)

return N^{th} fibonacci number,

```
int fibonacci(int N)     (1) Assuming to return Nth fibonacci
{
    if(N==1 || N==2)     (3) Base condition
        return 1;

    return fibonacci(N-1) + fibonacci(N-2);
                         (2) main logic
}
```

=> a + (a + d) + (a + 2d) + ... + (a + (n-1)d)
Add all terms.

```
int AP_sum(int a, int d, int n)
{
    if(n==1)
        return a;

    return a + (n-1)d + apsum(a, d, n-1);
        → a + apsum(a+d, d, n-1);
}
```

For iterative codes, we look at total no. of "Iterations"

For recursive codes, we do recurrence relations

---

(1) assume it takes T(N)

```
int sum(int N){
    if(N==0)
        return 0;
    return N + sum(N-1);
}
```

if (N == 0) return 0; $\Rightarrow$ $\boxed{T(0) = 1}$ (constant time for sum(0))

return $\underbrace{N}_{1} + \underbrace{sum(N-1)}_{T(N-1)}$;

$\Rightarrow \boxed{T(N) = 1 + T(N-1)}$

$T(N) = \underline{T(N-1)} + 1$

$T(N-1) = [T(N-2)+1] + 1$

$= T(N-2) + 2$

$\Rightarrow [T(N-3) + 1] + 2$

$= T(N-3) + 3$

⋮ Some pattern :)!

$= T(N-k) + k$

Imagine,

$$\boxed{\begin{aligned} T(N-k) &= T(0) \\ N-k &= 0 \\ N &= k \end{aligned}}$$

&

$T(N-N) + N$

$T(0) + N$

$1 + N \cong \underline{O(N)}$

② factorial code,

$$T(N) = 1 + T(N-1)$$

$\vdots$

$O(N)$

③ fibonacci code.

$$T(N) = 1 + T(N-1) + T(N-2)$$

$$T(N) = T(N-1) + T(N-2) + 1$$

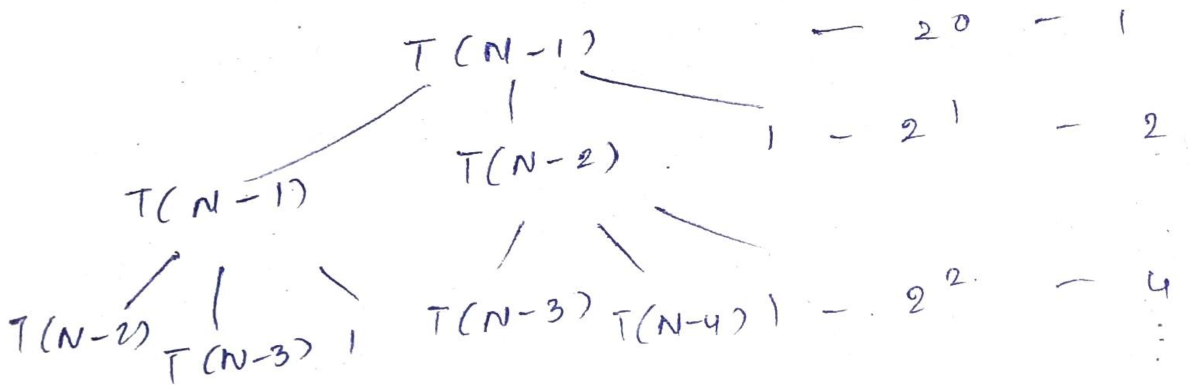
$$= [T(N-2) + T(N-3) + 1] + [T(N-3) + T(N-4) + 1] + 1$$

$$= T(N-2) + 2T(N-3) + T(N-4) + 3$$

$\vdots$ keeps on increasing continuously.

★★ so, Recurrence Relation ✗

Then what to do, go with with " Recurrence Tree ".



$$2^0 + 2^1 + 2^2 + \dots + 2^{N-1} = (2^N - 1) \Rightarrow O(2^N)$$

4 AP Sum series,

Recurrence Relations : $\boxed{T(1) = 1}$

1) $T(N) = 2T\left(\frac{N}{2}\right) + N \rightarrow N \log N$

2) $T(N) = 2T\left(\frac{N}{2}\right) + 1 \rightarrow N$

3) $T(N) = T\left(\frac{N}{2}\right) + N \rightarrow N$

4) $T(N) = T\left(\frac{N}{2}\right) + 1 \rightarrow N$

5) $T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{4}\right) + N^2 \rightarrow N^2$

(1) $T(N) = 2T\left(\frac{N}{2}\right) + N \rightarrow 1$

$= 2\left[2T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N$

$= 4T\left(\frac{N}{4}\right) + 2N \rightarrow 2$

$= 8T\left(\frac{N}{8}\right) + 3N \rightarrow 3$

$\vdots$

$\Rightarrow 2^K T(N/2^K) + KN$

$\Rightarrow N * 1 + (\log_2^N) N$

$= N \log_2^N + N$

$= N \log_2^N$

$T(1) = 1$

$T\left(\frac{N}{2^K}\right) = T(1)$

$N = 2^K$

$K = \log_2^N$

2)

$T(N) = 2T\left(\frac{N}{2}\right) + 1 \rightarrow (1)$

$= 2\left[2T\left(\frac{N}{4}\right) + 1\right] + 1$

$= 4T\left(\frac{N}{4}\right) + 3 \rightarrow (2)$

$= 4\left[2T\left(\frac{N}{8}\right) + 1\right] + 3$

$= 8T\left(\frac{N}{8}\right) + 7 \rightarrow (3)$

$\vdots$

$2^K T\left(N/2^K\right) + (2^K - 1)$

$N * 1 + 2^{\log_2 N} - 1 \Rightarrow O(N)$

$N + N - 1 = (2N - 1)$

$\frac{N}{2^K} = 1$

$K = \log_2 N$

3)

$T(N) = T\left(\frac{N}{2}\right) + N \rightarrow (1)$

$= \left(T\left(\frac{N}{4}\right) + \frac{N}{2}\right) + N$

$T\left(\frac{N}{4}\right) + \frac{3N}{2} \rightarrow (2)$

$T\left(\frac{N}{8}\right) + \frac{N}{4} + \frac{3N}{2}$

$T\left(\frac{N}{8}\right) + \frac{7N}{4} \rightarrow (3)$

$\vdots$

$N = 2^K$

$K = \log_2 N$

$2^{K-1} = \frac{N}{2}$

$T\left(\frac{N}{2^K}\right) + \frac{(2^K - 1)}{2^{K-1}} \cdot N$

$1 + \frac{N-1}{(N/2)} \cdot N = 2N \simeq O(N)$

(4) $T(N) = T\left(\frac{N}{2}\right) + 1 \rightarrow ①$

$= \left(T\left(\frac{N}{4}\right) + 1\right) + 1 = T\left(\frac{N}{4}\right) + 2 \rightarrow ②$

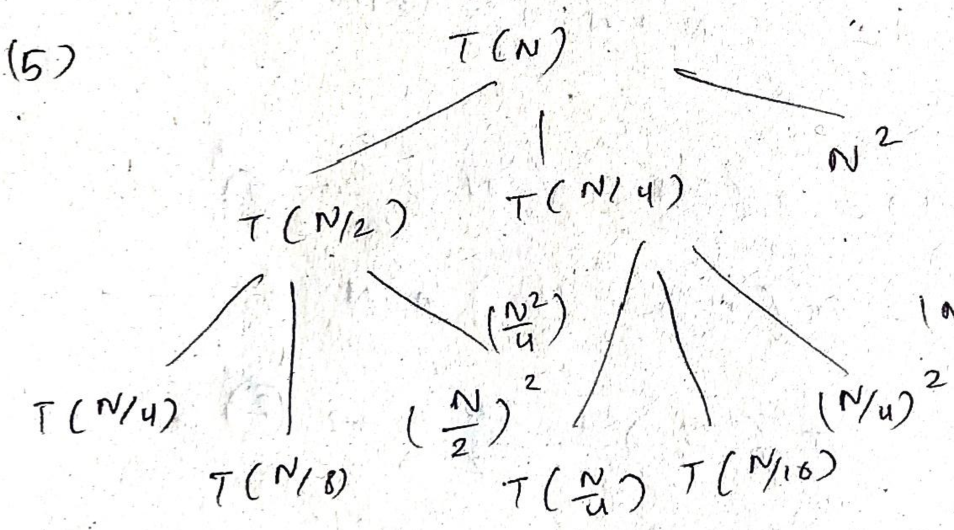
$= T\left(\frac{N}{8}\right) + 3 \rightarrow ③$

$\vdots$

$T\left(\frac{N}{2^K}\right) + K = 1 + N \cong \underline{O(N)}$

$N = 2^K$

(5)



$N^2 \rightarrow N^2 = \left(\frac{5}{16}\right)^0 N^2$

$(N^2/16) \rightarrow \frac{N^2}{4} + \frac{N^2}{16} = \frac{5N^2}{16} = \left(\frac{5}{16}\right)^1 N^2$

$\rightarrow \frac{N^2}{16} + \frac{N^2}{64} + \frac{N^2}{16} + \frac{N^2}{256} = \frac{25N^2}{256}$

$\left(\frac{5}{16}\right)^2 N^2$

$\vdots$

$\left(\frac{5}{16}\right)^K N^2$

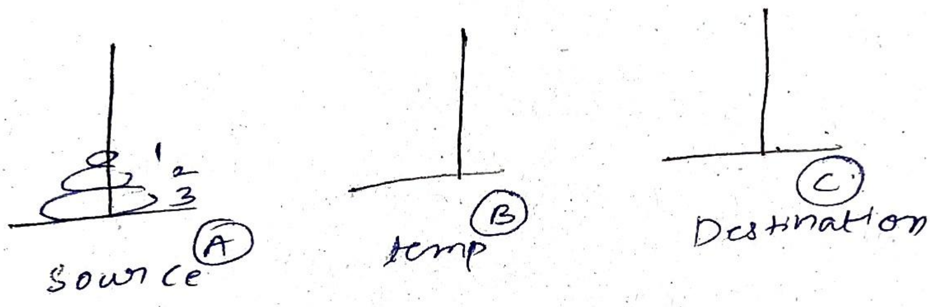
$= N^2\left[\left(\frac{5}{16}\right)^0 + \left(\frac{5}{16}\right)^1 + \left(\frac{5}{16}\right)^2 + \dots + \left(\frac{5}{16}\right)^K\right]$

$$= N^2 \left[\frac{1 - \left(\frac{5}{16}\right)^k}{1 - \frac{5}{16}} \right] = N^2 \left[\frac{16}{11} \cdot \left[1 - \left(\frac{5}{16}\right) \right.\right.$$

$$= O(N^2)$$

Towers of hanoi

N=3 disks



Constraints :-

(1) move one Disk at a time

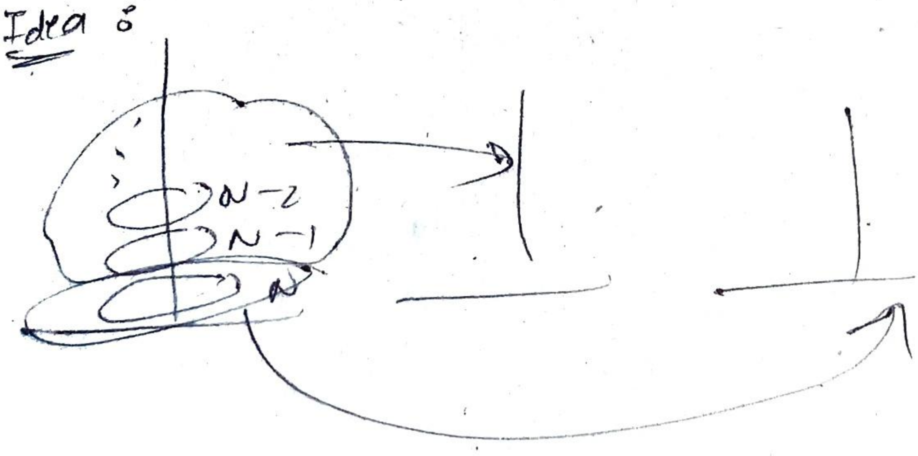
(2) Never place Bigger disk on a smaller disk.

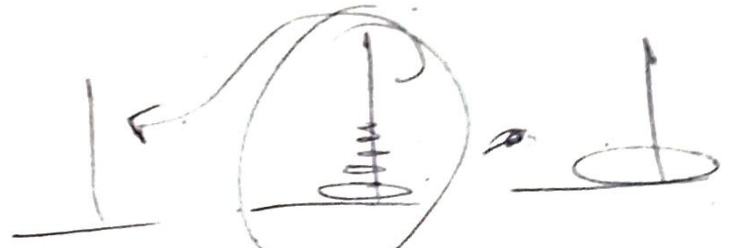
$\text{ill}^{\text{r}}(A \to C)$

$(N-1) \to$ src, temp	① src $\to$ temp via dest	1 : A $\to$ C 2 : A $\to$ B 1 : C $\to$ B
N^{th} disk	② src $\to$ dest	$\boxed{3 : A \to C}$
$(N-1)$ temp, dest	③ temp $\to$ dest via Source	1 : B $\to$ A 2 : B $\to$ C 1 : A $\to$ C

$\underline{\underline{7}}$ steps

Idea :





src acts a temp

temp acts as source

$$= 3 \quad A \quad C \quad B.$$

```
void TOH ( int N, char src, char dest, char temp )
{ (1) if (N == 0) return;
  (2) TOH (N-1, src, temp, dest);
  (3) print (N: src -> dest);
  (4) TOH (N-1, temp, dest, src);
}
```

N-1

N

$$T(N) = 2T(N-1) + 1 \rightarrow (1)$$

$$= 2(2T(N-2) + 1) + 1$$

$$= 4T(N-2) + 3 \rightarrow (2)$$

$$= 8T(N-3) + 7 \rightarrow (3)$$

$$2^k T(N-k) + (2^k - 1)$$

$$T(0) = 1$$

$$T(N-k) = T(0)$$

$$N - k = 0$$

$$\underline{N = k}$$

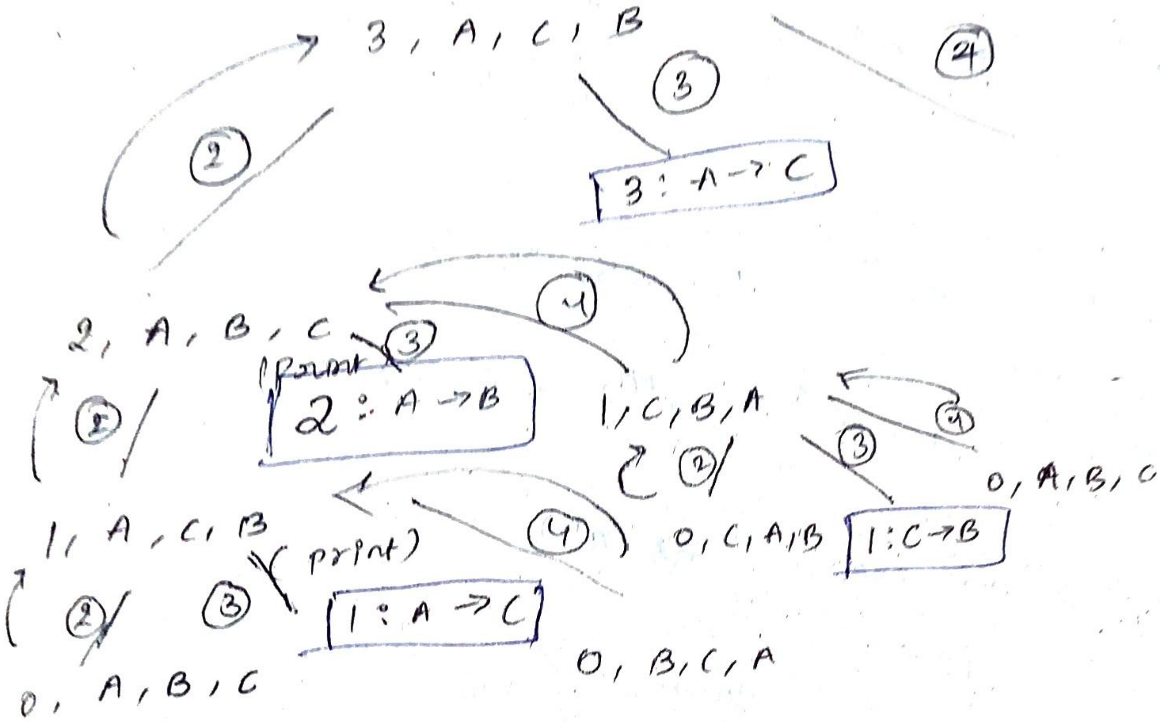
$$2^N(1) + 2^N - 1$$

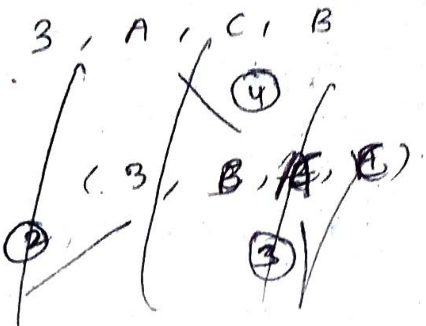
$$= 2^{N+1} - 1 \simeq O(2^N)$$

Tells no. of function calls

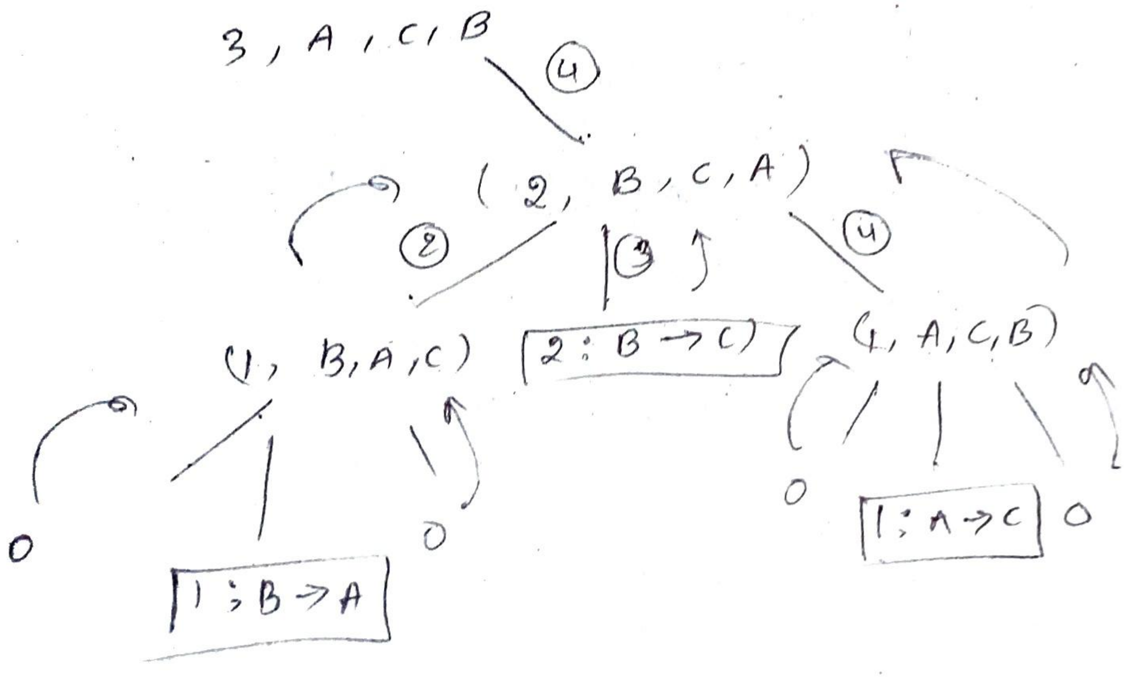
Trace 3, A, C, B

1 2 3





TOH(2, C, A, B) 3:



$a^N \rightarrow$ recursion

Bit manipulation $\rightarrow \log_2^N$

$a^N = a \cdot a^{N-1}$ $\quad T(N) = 1 + T(N-1)$

$O(N)$

$\rightarrow a^N = a^{N/2} \cdot a^{N/2} \rightarrow$ if (Even)

$= a \cdot a^{N/2} \cdot a^{N/2} \rightarrow$ if (odd)

```
int pow( int a, int N)
{
        if( N = = 0)
            return 1;          x = pow (a, N/2)
                               multiple times
        if( N % 2 = = 0)       calls are repeated
            return  pow(a, N/2) * pow(a, N/2)
                       x      *     x
        else
            return  a * pow(a, N/2) * pow(a, N/2)
                            x             x
}
```

$T(N) = 1 + 2T\left(\frac{N}{2}\right) \simeq \underline{O(N)}$

$\Downarrow$

$T(N) = T(N/2) + 1 = \log_2^N$