* Recursion

Suppose 'P' is a procedure that contains a call statement to itself or a call statement to another procedure that may eventually result in a call statement back to original procedure P. Then 'P' is called a recursive procedure.

Recursive fuction must have two properties-

(1) There must be certain criteria, called base criteria, for which the procedure does not call itself.

(2) Each time the procedure does call itself (directly / indirectly) it must be closer to base criteria.

Procedure with above properties is said to be well defined.

⇒ Factorial fuction

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-2)(n-1)(n).$$

Also,

$$n! = n \cdot (n-1)!$$

Factorial fuction can be defined as:

a) If $n=0$ then $n! = 1$

b) If $n>0$ then $n! = n(n-1)!$

This definition is recursive in nature.

Example

Lets calculate 4! using this definition.

1) $4! = 4 \cdot 3!$
2) $3! = 3 \cdot 2!$
3) $2! = 2 \cdot 1!$
4) $1! = 1 \cdot 0!$
5) $0! = 1 \rightarrow$ Base co
6) $1! = 1 \cdot 1 = 1$
7) $2! = 2 \cdot 1 = 2$
8) $3! = 3 \cdot 2 = 6$
9) $4! = 4 \cdot 6 = 24$

FACTORIAL (FACT, N)

1. If N=0 the set FACT = 1 and Return.
2. Set FACT = 1
3. Repeat for K=1 to N
   Set FACT = FACT * K.
   [End of loop]
4. Return.

FACTORIAL (FACT, N)

1. If N=0 then set FACT=1 and Return.
2. Call FACTORIAL (FACT, N-1)
3. Set FACT = N * FACT.
4. Return.

Complexity = O(n). $t(n) = t(n-1) + 1$

$t(n) = t(n-2) + 1 + 1$

* FIBONACCI Sequence

The fibonacci sequence (usually denoted by $f_0, f_1, f_2, \ldots$) is as follows -

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$f_0 = 0$ & $f_1 = 1$ and each succeeding term is the sum of previous two terms.

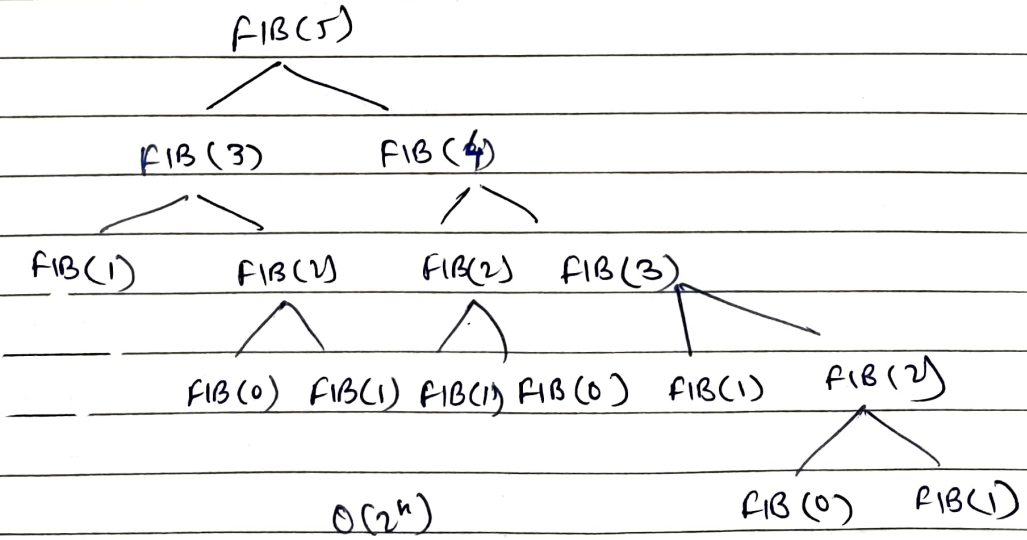
Definition

a) If $n=0$ or $n=1$ $f_n = n$

b) If $n>1$ then $f_n = f_{n-2} + f_{n-1}$

FIBONACCI (FIB, N)

1. If N=0 or N=1 then set FIB := N & Return
2. Call FIBONACCI (FIBA, N-2)
3. Call FIBONACCI (FIBB, N-1)
4. Set FIB = FIBA + FIBB
5. Return.

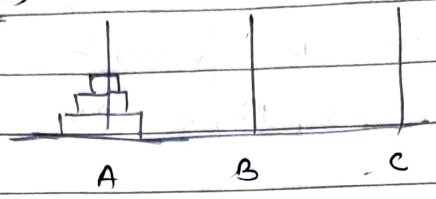


$O(n)$ ⇓ for non recursive call

$O(2^n)$ ⇓ for Recursive call.

TOWER of HANOI

for N=3



A→C , A→B , C→B , A→C , B→A , B→C , A→C

TOH can be reduced to the following subproblems

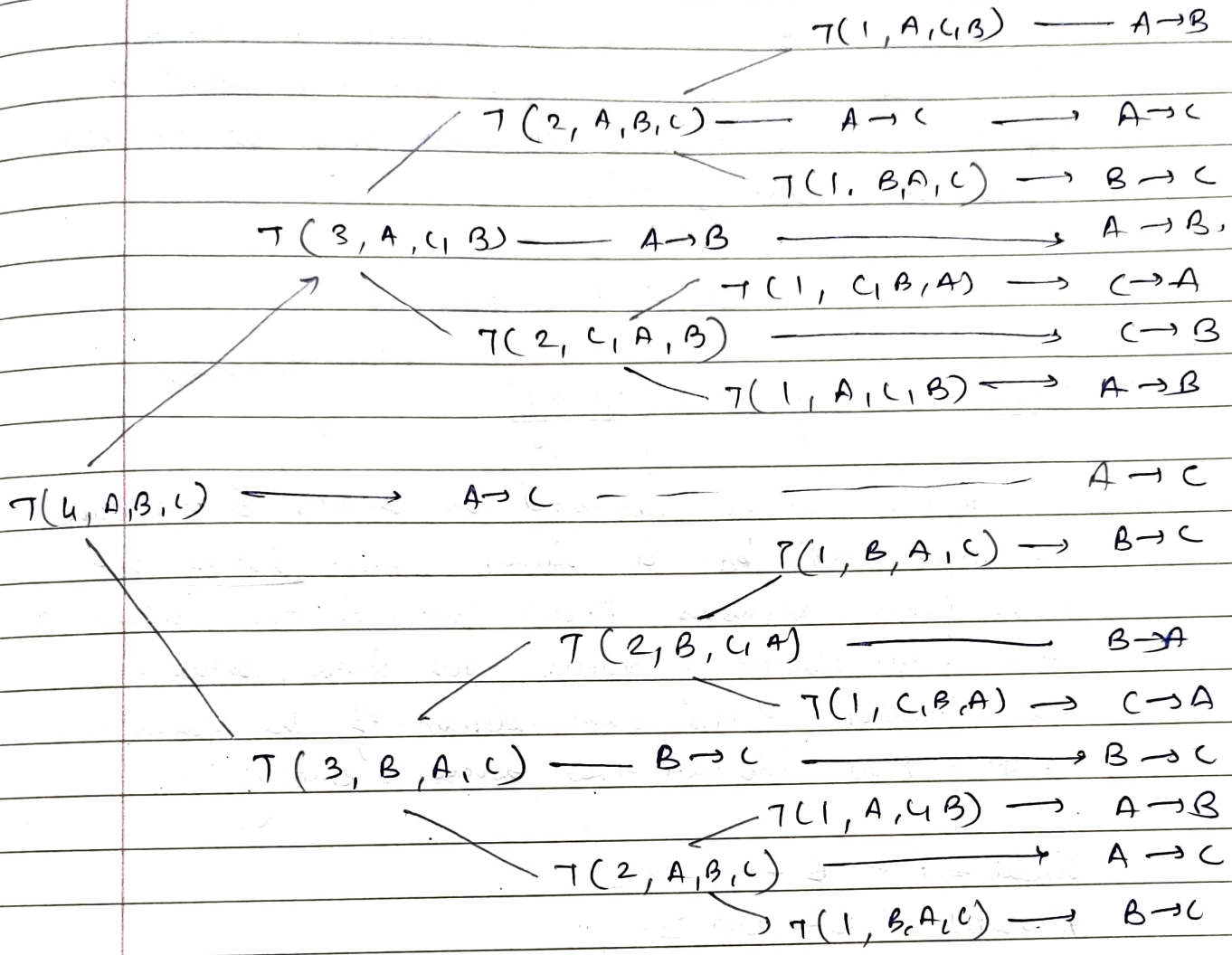
1. Move top (n-1) disk from A → B
2. Move top disk from A → C
3. Move the top (n-1) disk from B → C

when N=1 , TOWER (1, BEG, AUX, END)

BEG → END.

when N>1

1. TOWER (N-1, BEG, END, AUX)
2. TOWER (1, BEG, AUX, END) or BEG → END
3. TOWER (N-1, AUX, BEG, END)



Recursive solution to TOH for n=4.