Recursion

Recursion is one of the most powerful tool in the programming language.

Recursion is defined as defining anything in terms of itself.

Recursion is a method of solving problems that involves breaking a problem down into smaller and smaller subproblems until you get a small enough problem that can be solved trivially.

Recursive Procedure:-

Suppose P is a procedure containing either a call statement to itself or a call statement to a second procedure that may eventually result in a call statement back to the original procedure P. Then P is called a recursive procedure.

Two basic properties of recursive procedure are as follows:-

⇒ Base Criteria

⇒ Recursive Procedure

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 or indirectly), it must be closer to the base criteria.

3) Recursion is an alternative to iteration in making a function execute repeatedly.

Types of Recursion

Recursion is of two types depending on whether a function calls itself from within itself or whether two functions call one another mutually.

The former is called direct recursion and the later is called indirect recursion.

Direct

```
int abc()
{
 abc()
}
```

Indirect Recursion

```
int abc()
{
 ---;
 ---;
 xyz();
}
int xyz()
{
 abc();
 ---;
}
```

Factorial Function

The product of positive integers from 1 to n is called "n factorial" and is denoted by n!.

$$n! = 1 \times 2 \times 3 \times 4 \ldots \times (n-2) \times (n-1) \times (n)$$

It is also convenient to define $0! = 1$, so that the function is defined for all non-negative integers.

Definition:- (Factorial Function) (recursive definition):

a) if $n=0$, then $n!=1$ // base case

b) if $n>0$, then $n! = n*(n-1)!$ // recursive function

n! is recursive since it refers to itself when it uses (n-1)!.

Recursive Procedure for finding the factorial of a number:

```
int factorial (int n)
{
if (n==1)          // if (n==0)
return 1;
else
return(n * factorial(n-1));
}
```

Sol:

Factorial (5) ⇒ 120

↓

(5 * factorial (4)) ⇒ 5 * 4 * 3 * 2 * 1

↓

(4 * factorial (3)) ⇒ 4 * 3 * 2 * 1

↓

(3 * factorial (2)) ⇒ 3 * 2 * 1

↓

(2 * factorial (1)) ⇒ 2 * 1.

↓

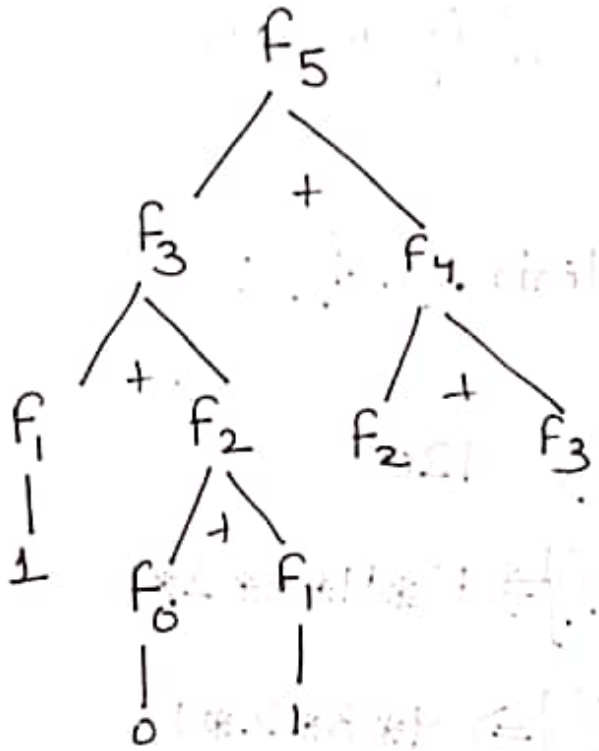
1

Fibonacci Series 0, 1, 1, 2, 3, 5, 8

1) If $n=0$ or $n=1$, then $f_n = n$ // Base Case

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

Eg., $n=5$



$\Rightarrow$ $f_0 = 0$

$f_1 = 1$

$f_2 = 1+0$
$= 1$

$f_3 = 1+1 = 2$

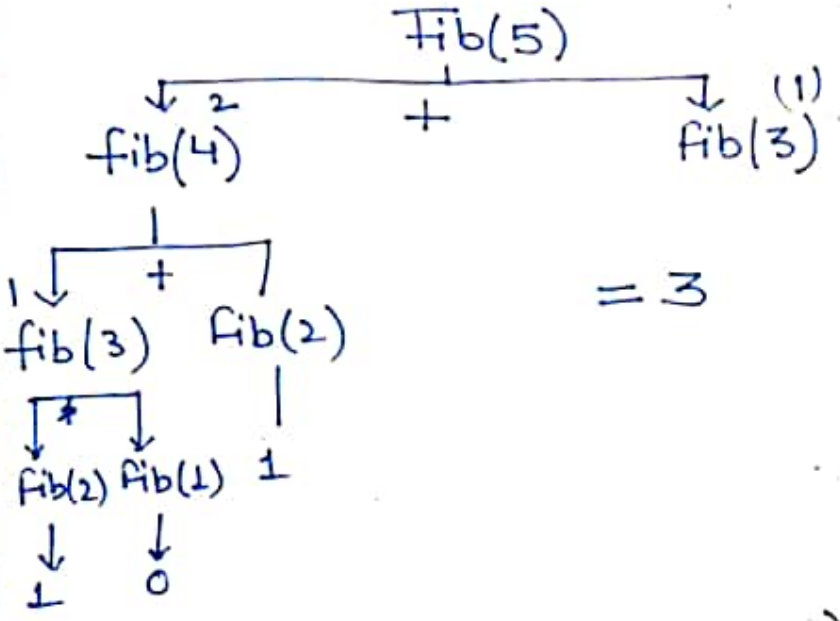
$f_4 = f_3 + f_2 = 3$

$f_5 = 2+3$
$= 5$

Recursive function to find n^{th} term of a fibonacci Series.

```
int fib(int n)
{
  if(n==1)
  return 0;
 if(n==2)
 return 1;
 else
 return fib(n-1)+fib(n-2);
 }
```

0, 1, 1, 2, 3, 5, 8.



$= 3$

Ques: $GCD(A,B) = \begin{cases} GCD(B,A), & \text{if } A<B \\ A, & \text{if } B=0 \text{ // base Case} \\ GCD(B, A\%B), & \text{if } A \geq B \end{cases}$

Find GCD(50,20)

$\Rightarrow$ GCD(20,10)

GCD(10,0) $\Rightarrow$ Since B=0

Ans = 10 i.e., A

Tower of Hanoi

→ In this recursion is used as a tool in developing an algo. to solve a particular problem.

→ Three pegs, A, B, C, on peg A there are placed n no. of disks.

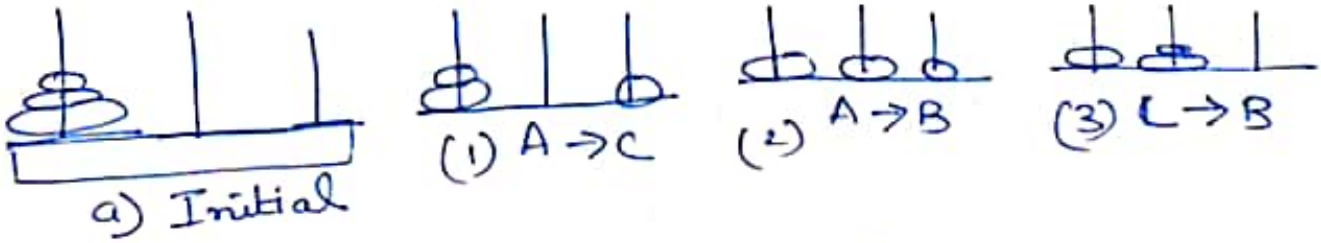
→ Object of the game is to move the disks from Peg A to Peg C using peg B as an auxiliary.

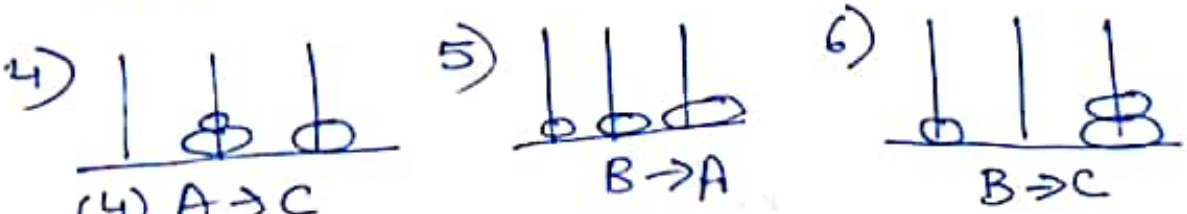
The rules of the game are as follows:-

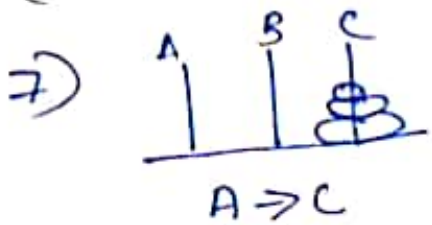
a) Only one disk may be moved at a time. Specifically, only the top disk on any peg may be moved to any other peg.

b) At no time can a larger disk be placed on a small disk.

$n = 3$







for $n=1$ $A \to C$

$n=2$ $A \to B$, $A \to C$, $B \to C$

Rather than finding a separate solution for each n, we use the technique of recursion to develop a general solution.

for $n>1$

1) Move the top n-1 disks from peg A to peg B.
2) Move the top disk from peg A to peg C:

$A \to C$

3) Move the top n-1 disks from peg B to peg C.

$n=3$

⇒ first move top 5 disks from A to B, then move large disk from A to C, then move top five disks from peg B to peg C.

for $n>1$

Tower (N, BEG, AUX, END)

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

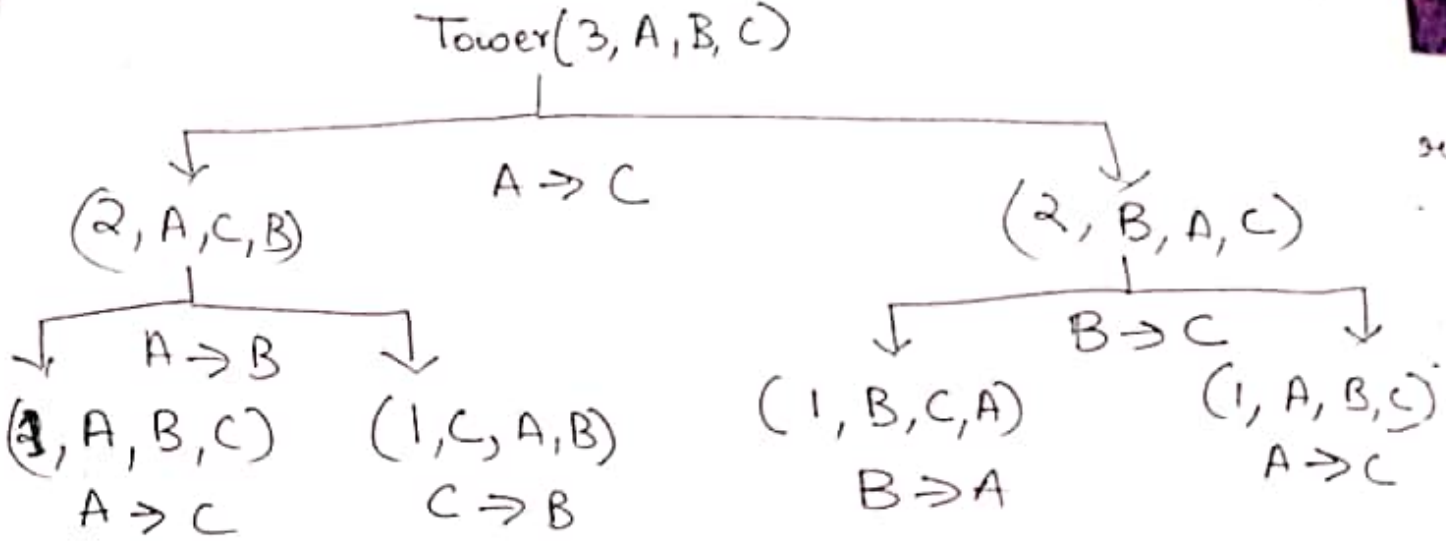
In general recursive sol^n requires $f(n) = 2^n - 1$ moves for n disks.

Algo.

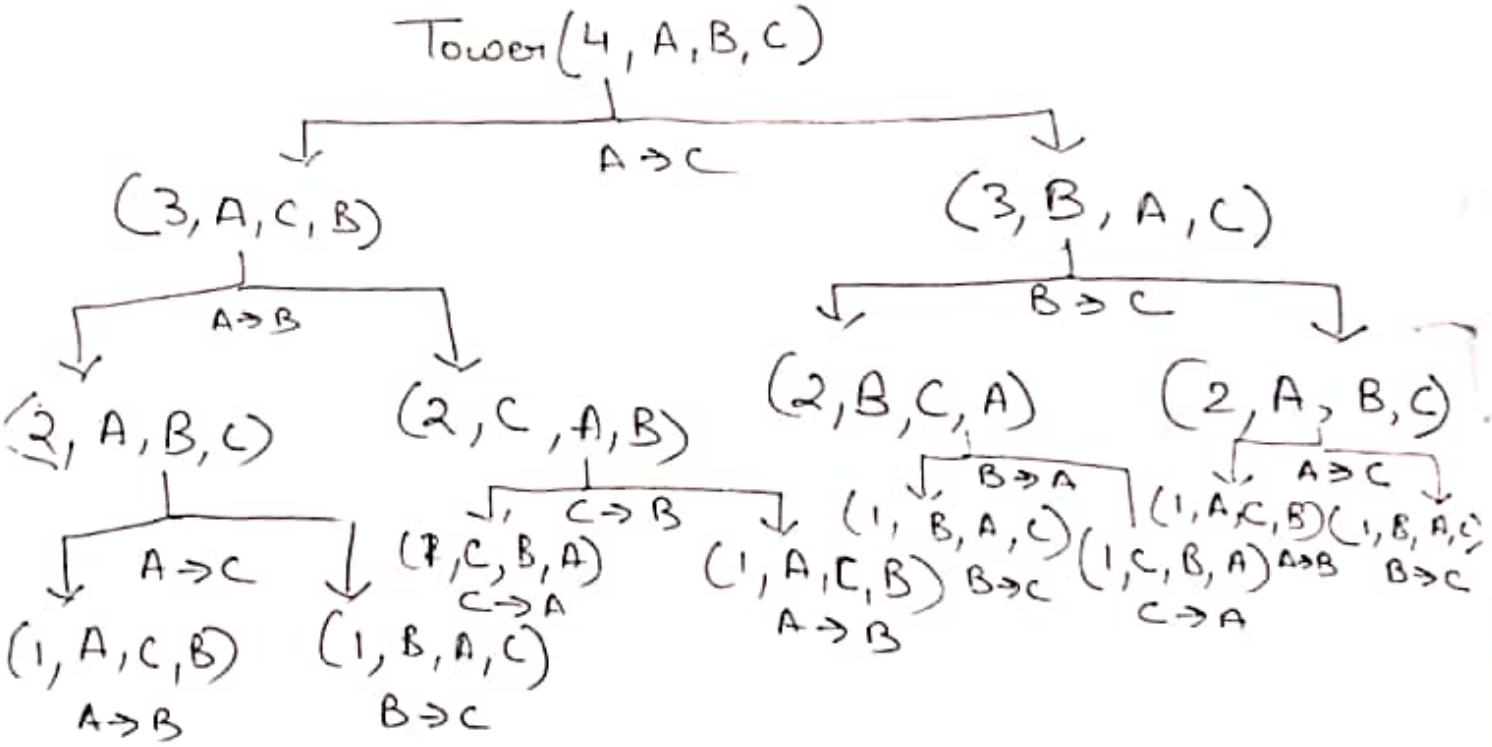
Tower(N, BEG, AUX, END)

This procedure gives a recursive solution to the Towers of Hanoi problem for N disks.

1. If N=1, then:
   a) Write BEG → END
   b) Return
   [End of if structure]
2) [Move N-1 disks from peg BEG to peg AUX]
   Call Tower (N-1, BEG, END, AUX).
3) Write: BEG → END
4) [Move N-1 disks from peg AUX to peg END].
   Call Tower (N-1, AUX, BEG, END).
5. Return.



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



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

C → A B → C A → B A → C B → C