Recursion
Chapter 5

Presented by: Er. Aruna Chhatkuli

Nepal College of Information Technology,
Balkumari, Lalitpur

Recursion

Recursion:

- The recursion is **a process by which a function calls itself**.
- We use recursion to solve bigger problem into smaller sub-problems.
- One thing we have to keep in mind, that if each sub-problem is following same kind of patterns, then only we can use the recursive approach.
- A recursive function has two different parts.

Recursion

A recursive function has two different parts.

Base criteria:

- There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.

- The base case and the recursive case. The base case is used to terminate the task of recurring. If base case is not defined, then the function will recur infinite number of times (Theoretically).

Recursive:

- The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

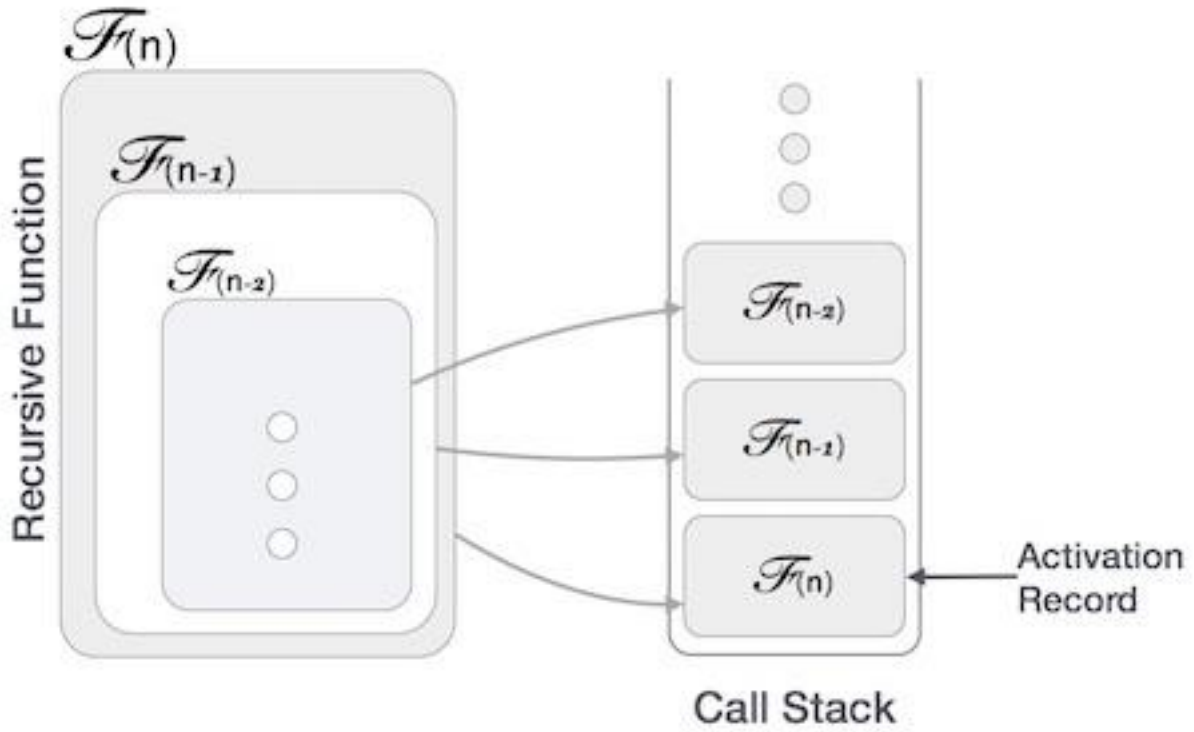
Recursion

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.
- Using a recursive algorithm, certain problems can be solved quite easily.
- Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.
- A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems.

Recursion

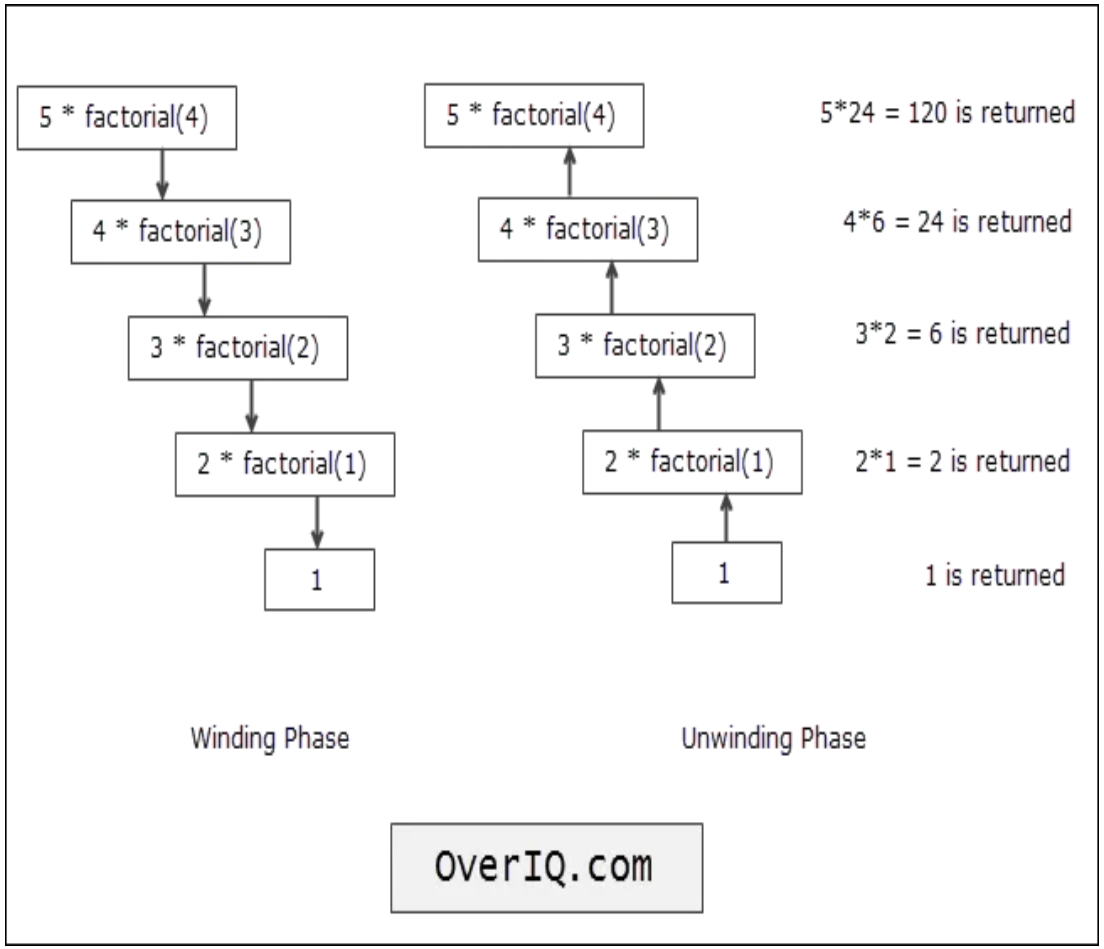
- Many more recursive calls can be generated as and when required.
- It is essential to know that we should provide a certain case in order to terminate this recursion process.
- So we can say that every time the function calls itself with a simpler version of the original problem.

Recursion


F(n)
F(n-1)
F(n-2)
Recursive Function
F(n-2)
F(n-1)
F(n)
Activation Record
Call Stack

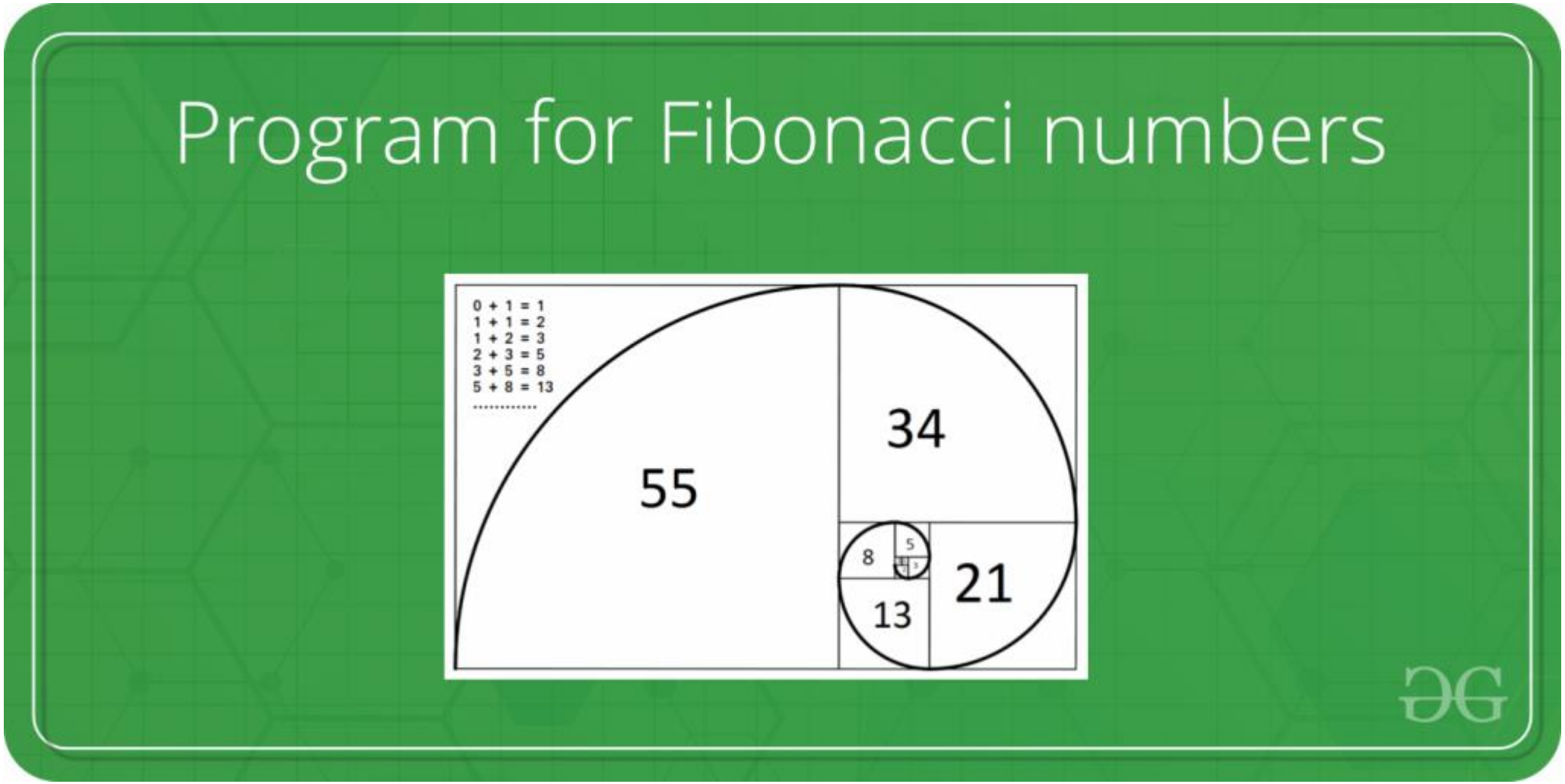
Factorial using recursion

```
#include <stdio.h>
int fact(int);
int main() {
   int n;
   printf("enter the value for n: ");
   scanf("%d",&n);
   int factorial=fact(n);
   printf("factorial of the n is:%d",factorial);
    return 0;        }
int fact(int n){
    if(n<=0)
       return 1;
    else   {
       return n*fact(n-1);   }
```



Fibonacci series using recursion

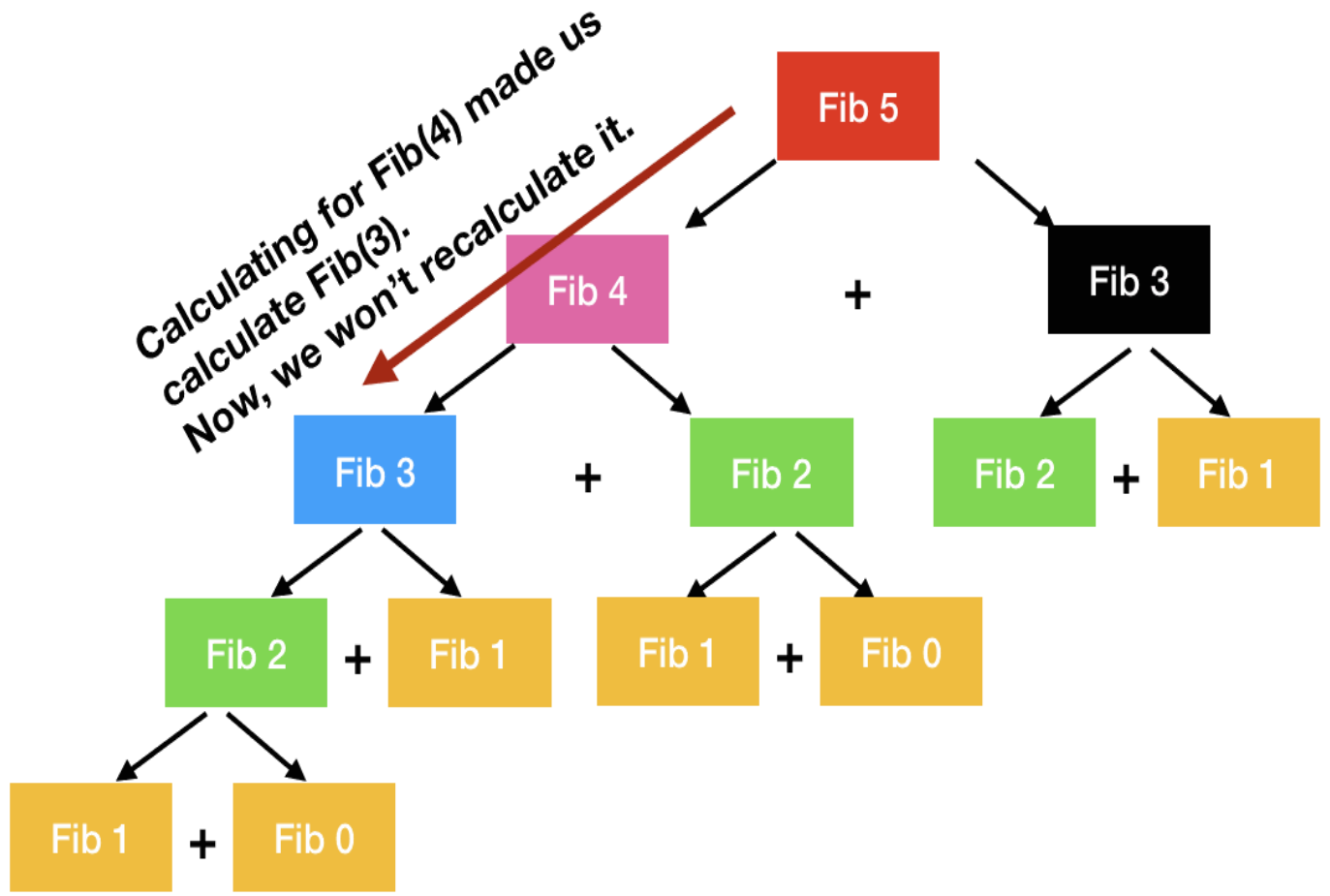
Fibonacci:



Fibonacci series using recursion

Fibonacci:

```
#include <stdio.h>
int fib(int n){
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
int main(){
    int n =19;
    printf("%d", fib(n));
    getchar();
    return 0;
}
```



Recursion

Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).

Advantages of Recursion

i. The main benefit of a recursive approach to algorithm design is that it allows programmers to take advantage of the repetitive structure present in many problems.

ii. Complex case analysis and nested loops can be avoided.

iii. Recursion can lead to more readable and efficient algorithm descriptions.

i. Recursion is also a useful way for defining objects that have a repeated similar structural form.

Disadvantages

i. Slowing down execution time and storing on the run-time stack more things than required in a non recursive approach are major limitations of recursion.

ii. If recursion is too deep, then there is a danger of running out of space on the stack and ultimately program crashes.

iii. Even if some recursive function repeats the computations for some parameters, the run time can be prohibitively long even for very simple cases.

Some examples of recursion

- Factorial of a number
- Fibonacci series
- Tower of Hanoi

Tower of Hanoi

- It is one of the main application of recursion whose objective is to transfer all disks from origin pole to destination pole using intermediary pole as a temporary storage

Condition:

1. Move one disk at a time
2. Each disk must be placed around the pole
3. Never place larger disk on top of smaller disk

Algorithm for TOH

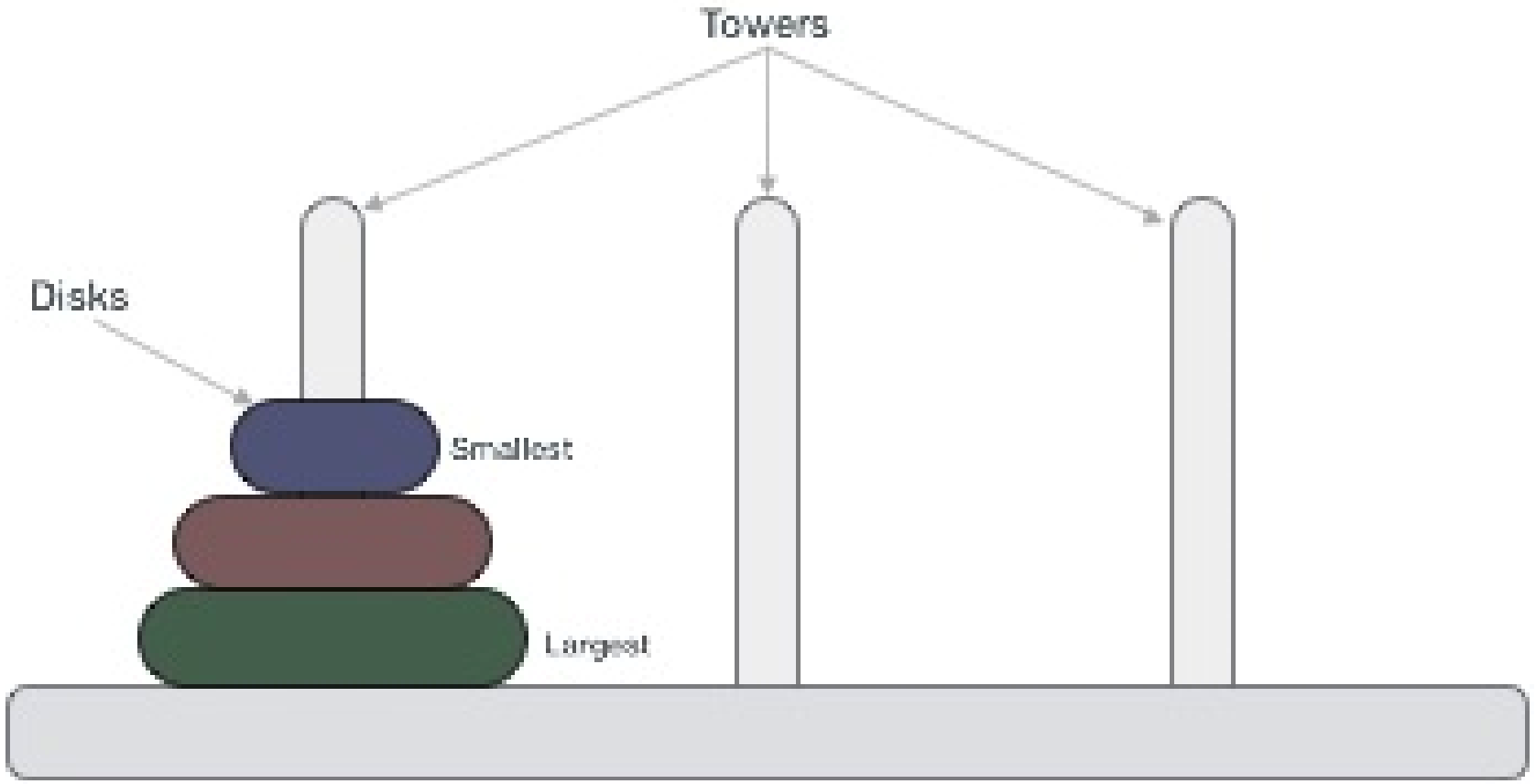
Let's consider move 'n' disks from source peg (A) to destination peg (C), using intermediate peg (B) as auxiliary.

1. Assign three pegs A, B & C

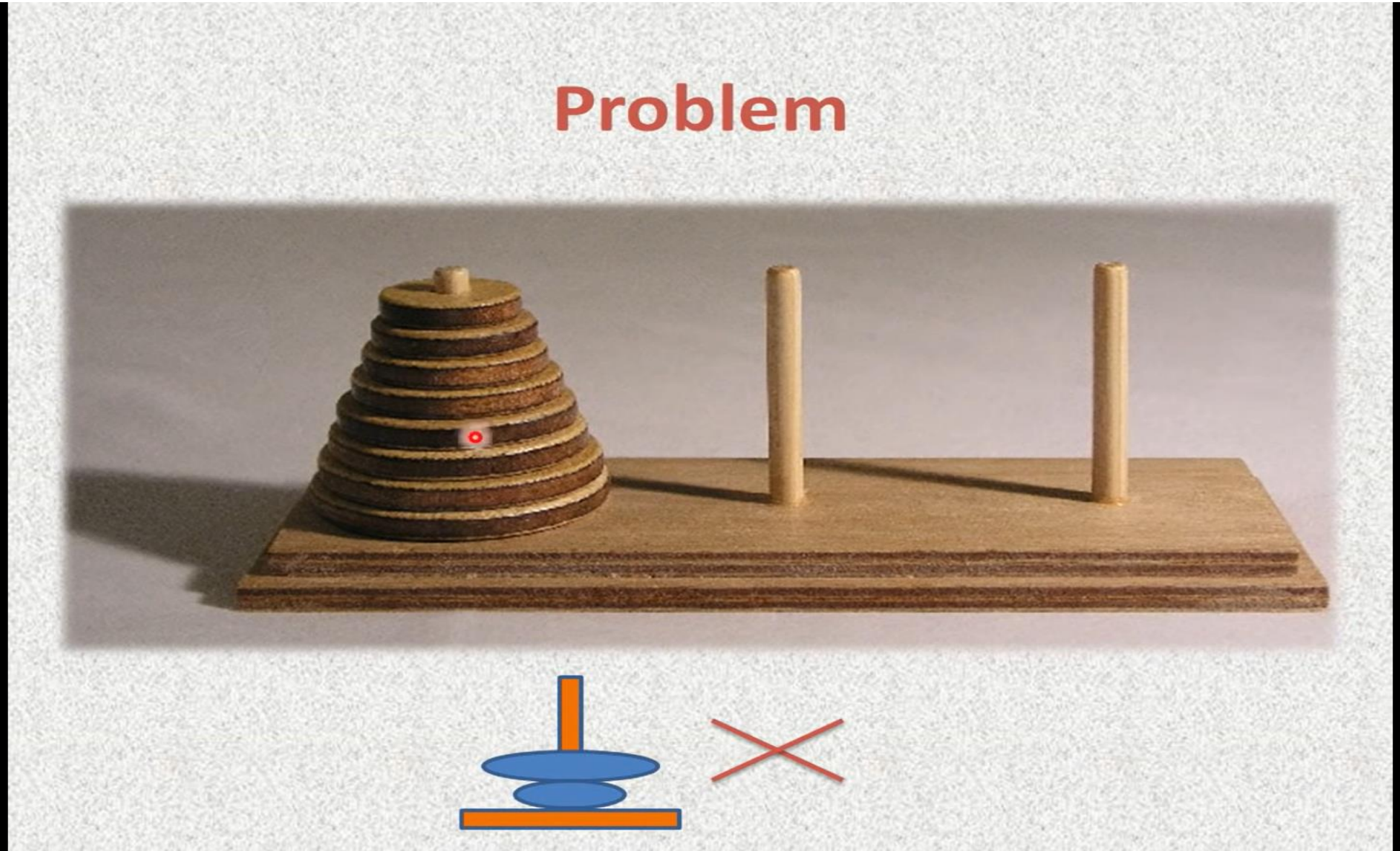
2. If n==1, Move the single disk from A to C and stop.

3. If n>1
   i. *Shift 'N-1' disks from 'A' to 'B', using C.*
   ii. *Shift last disk from 'A' to 'C'.*
   iii. *Shift 'N-1' disks from 'B' to 'C', using A.*
   iv. Terminate

Problem is to transfer the disks from left most tower to right most tower with the help of middle tower(intermediary tower)



Problem is to transfer the disks from left most tower to right most tower with the help of middle tower(intermediary tower)



Following are the steps to solve the TOH problem for 3 disks

Move disk 1 from tower A to tower C

Move disk 2 from tower A to tower B

Move disk 1 from tower C to tower B

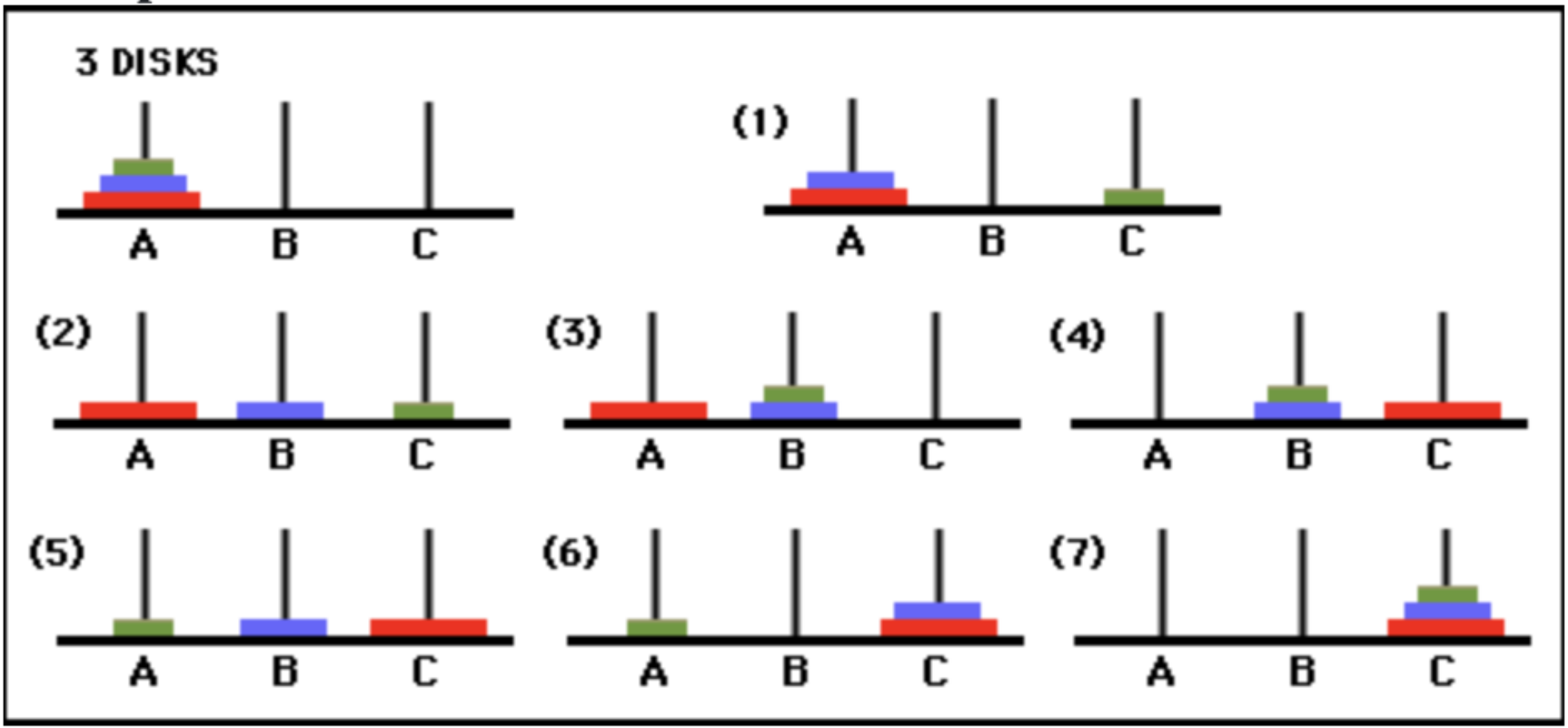
Move disk 3 from tower A to tower C

Move disk 1 from tower B to tower A

Move disk 2 from tower B to tower C

Move disk 1 from tower A to tower C

Example for 3 disks: 7 moves

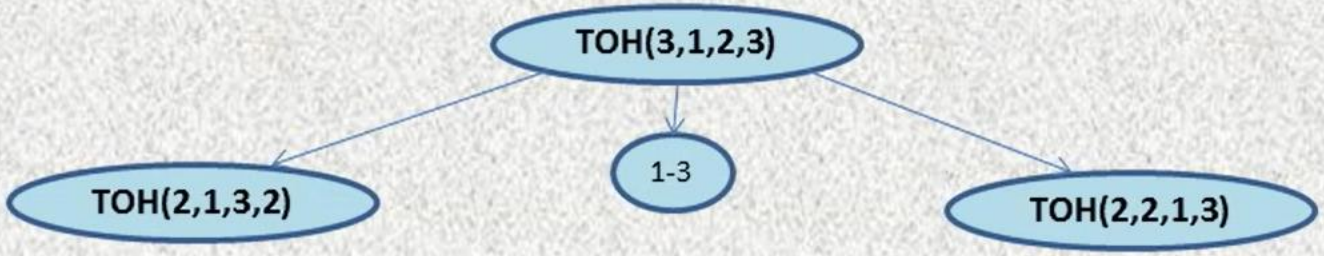


Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```

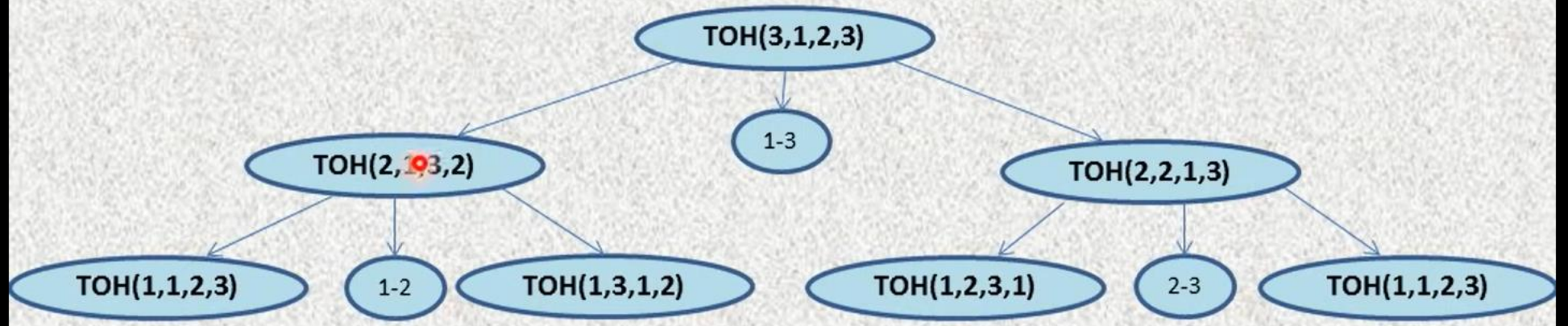
Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



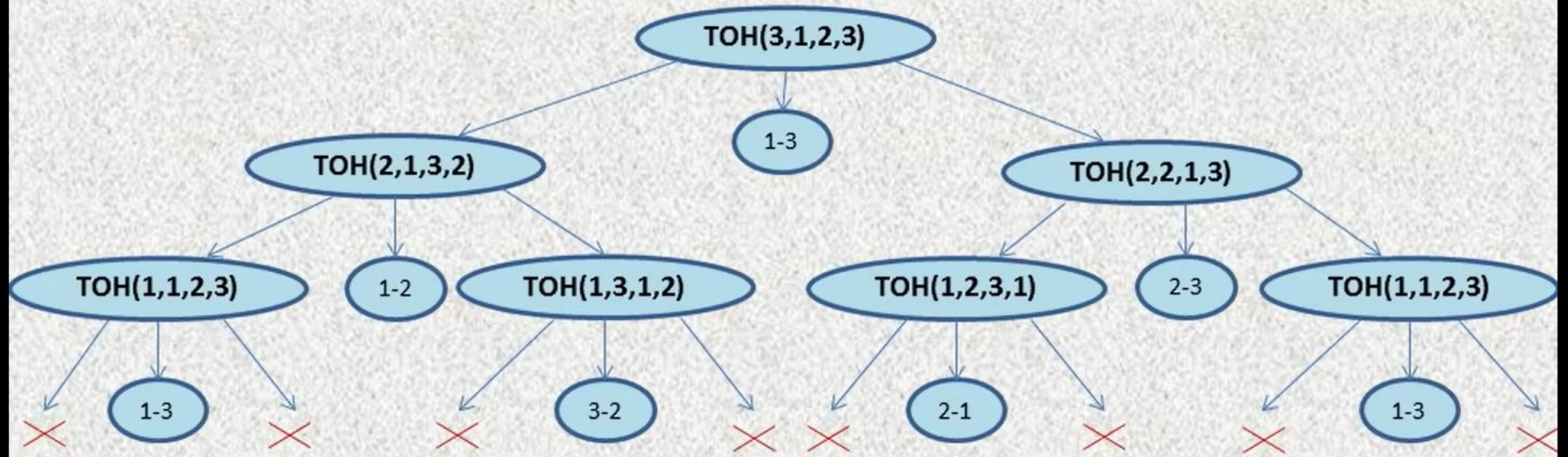
Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



Tracing for 3 Discs

```
void TOH(int n,  int A,  int B,  int C)
{
        if(n>0)
        {
                TOH(n-1, A , C , B);
                printf( "Move a Disc from %d to %d", A , C);
                TOH(n-1, B , A , C);
        }
}
```



(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

(1-3)

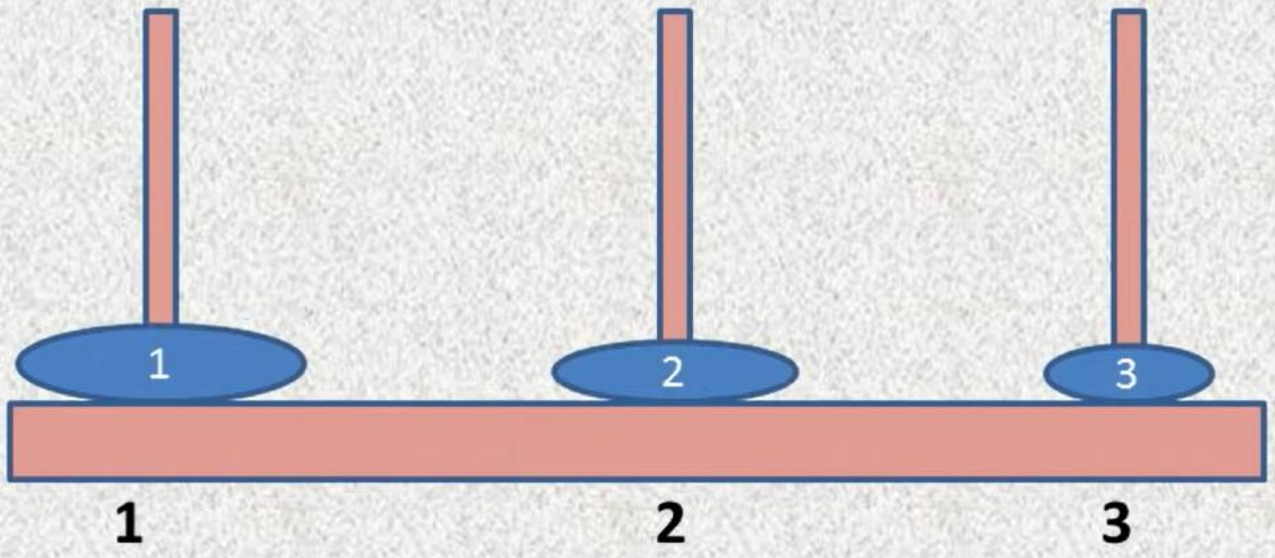
Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

(1-2)

Tracing for 3 Discs

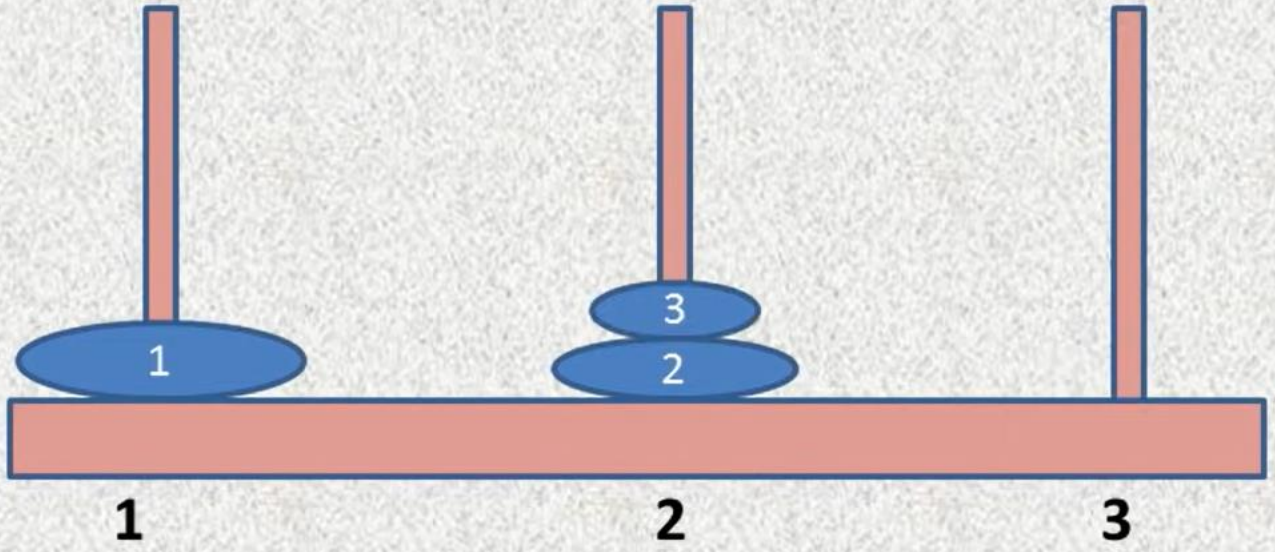
(1-3) (1-2) (3-2) **(1-3)** **(2-1)** **(2-3)** **(1-3)**



(3-2)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) **(2-1)** **(2-3)** **(1-3)**



(1-3)

Tracing for 3 Discs

(1-3) (1-2) (3-2) (1-3) (2-1) (2-3) (1-3)

Done

Pseudocode for TOH

```
void towers(int n,char fromtower,char totower,char auxtower) { /* If only 1 disk, make the move and return */

if(n==1) {

printf("\nMove disk 1 from tower %c to tower %c",fromtower,totower);

return;

}

/* Move top n-1 disks from A to B, using C as auxiliary

towers(n-1,frompeg,auxtower,totower);

/* Move remaining disks from A to C */

printf("\nMove disk %d from tower %c to tower %c",n,fromtower,totower);

/* Move n-1 disks from B to C using A as auxiliary */

towers(n-1,auxtower,totower,fromtower);

}
```

Thank you!!!!!