Unit-4
Recursion

⊛. **Definition:-** The process in which a function calls itself directly or indirectly is called recursion. It is a powerful technique of writing a complicated algorithm in an easy way. According to this technique a problem is defined in terms of itself. The problem is solved by dividing it into smaller problems, which are similar in nature to the original problem. These smaller problems are solved and their solutions are applied to get the final solution of our original problem.

A function will be recursive, if it contains following features;

i) Function should call itself.

ii) Function should have a stopping condition (base criteria).

Using 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.

⊛. **Recursive vs. Iterative algorithm:-**

Property	Recursion	Iteration
i) Definition	Function calls itself	A set of instructions repeatedly executed.
ii) Application	It is used for functions	It is used for loops.
iii) Termination	It will terminate at base case, where there will be no function call.	It will terminate when the condition for iteration is satisfied.
iv) Usage	It is used when code size needs to be small, and time complexity is not an issuse.	It is used when time complexity needs to be balanced against an expanded code size.
v) Code Size	It has smaller code size	vi) It has larger code size.
vi) Time Complexity	It has very high (generally exponential) time complexity.	It has relatively lower time complexity (generally polynomial or logarithmic).

⊛ Tail recursion (Definition & Example):-

A recursive function is tail recursive when recursive call is the last thing executed by the function. So, when nothing is left to do after comming back from the recurive call.

Example (Using C++):

```
#include<iostream>
using namespace std;
void printN(int n){ if(n<0){
                        return;
                    }
                    cout<<n<<" ";
                    printN(n-1);
                   }
int main(){ printN(10);
          }
```

Output:

10 9 8 7 6 5 4 3 2 1 0

The tail recursion is better than non-tail recursion. As there is no task left after the recursive call, it will be eaiser for the compiler to ~~organize~~ optimize the code.

A non-tail recursive function can be written as tail-recursive to optimize it. Consider the following function to calculate factorial of n.

```
long fact(int n){ if(n<=1)
                      return 1;
                  n*fact(n-1);
                }
```

This function is a non-tail recursive function because the value returned by fact(n-1) is used in fact(n), so the call to fact(n-1) is not the last thing done by fact n. The above function can be written as tail recursive function by adding some other parameters as follows:-

```
long fact(long n, long a){ if(n==0)
                               return a;
                           return fact(n-1, a*n);
                         }
```

⊛ Examples of Recursive Algorithms:

1) Factorial: Factorial of any number n is denoted as n! and is equal to; $n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$.

for e.g; Factorial of 5 is;

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$= 120$

Recursive Algorithm for calculation of factorial of given integer;

Step1: Start

Step2: Read number n

Step3: call factorial(n).

Step4: Print factorial f

Step5: Stop.

function to call

int factorial(n)

Step1: if n==1 then return 1.

Step2: else,

f = n*factorial(n-1)

Step3: return f;

Now we can easily write C or C++ program on the basis of this algorithm.

2) Fibonacci Sequence: Fibonacci series are the numbers in the sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

By the definition, the first two numbers are 0 and 1. & each subsequent numbers in the series is equal to the sum of the previous two numbers.

Recursive Algorithm for Fibonacci Sequence:

Step 1: Start

Step2: Read number n.

Step3: Call fib(n).

step4: Print fib(n)

step5: Stop.

int fib(int n)

Step1: if (n=1 || n==2)
return 1;

Step2: else, fib(n)= fib(n-1)+Fib(n-2)

Step3: return fib(n);

3) Greatest Common Divisor (GCD): [Imp]

GCD or Greatest Common Divisor of two or more integers is the largest positive integer that can divide both the number without leaving any remainder.

Example:- GCD of 20 and 8 is 4.

Recursive Algorithm for GCD:

Step1: Start

Step2: Read two numbers a and b.

Step3: Call gcd(a,b)

Step4: Print gcd.

Step5: Stop.

int gcd (int a, int b)

Step1: if (b==0)
return a;

Step2: else, gcd=gcd(b, a%b);

Step3: return gcd;

⊛ Tower of Hanoi:- [Imp]

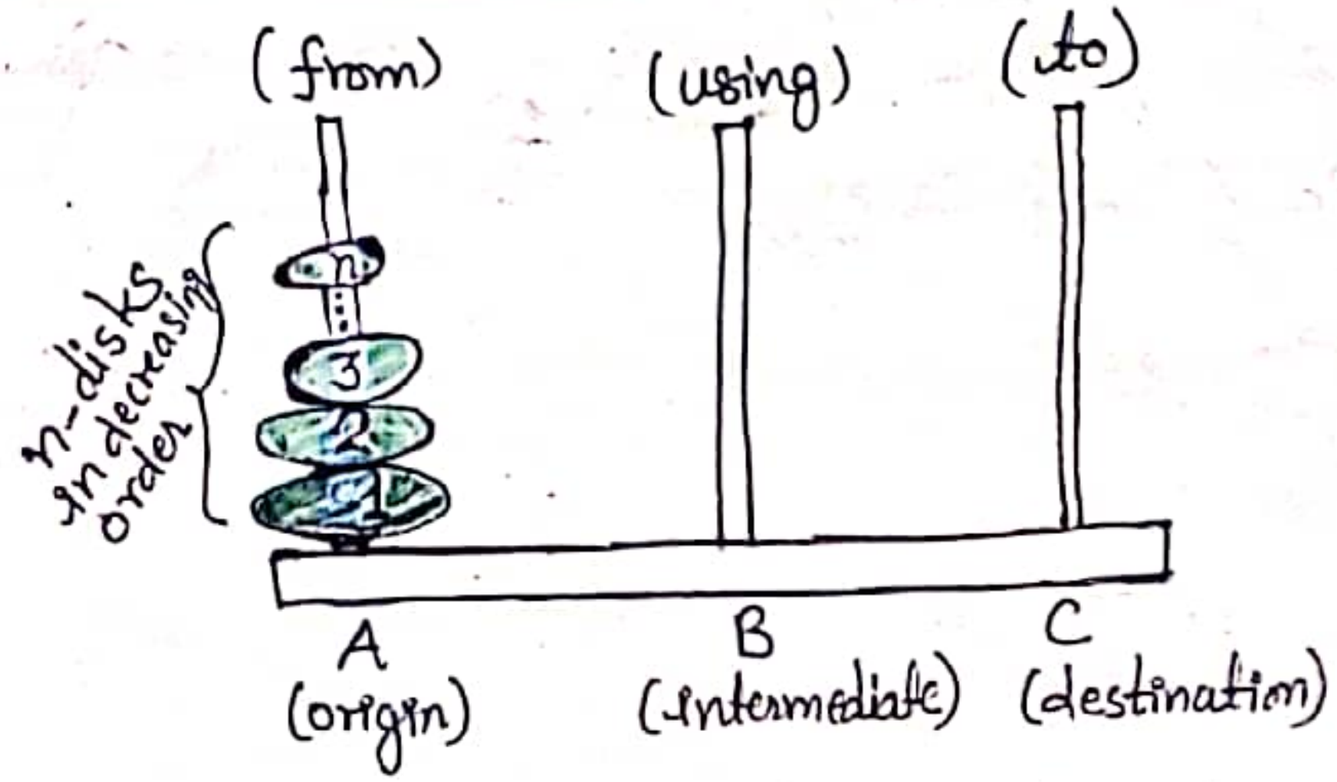


fig. Representation of tower of Hanoi (Initial state)

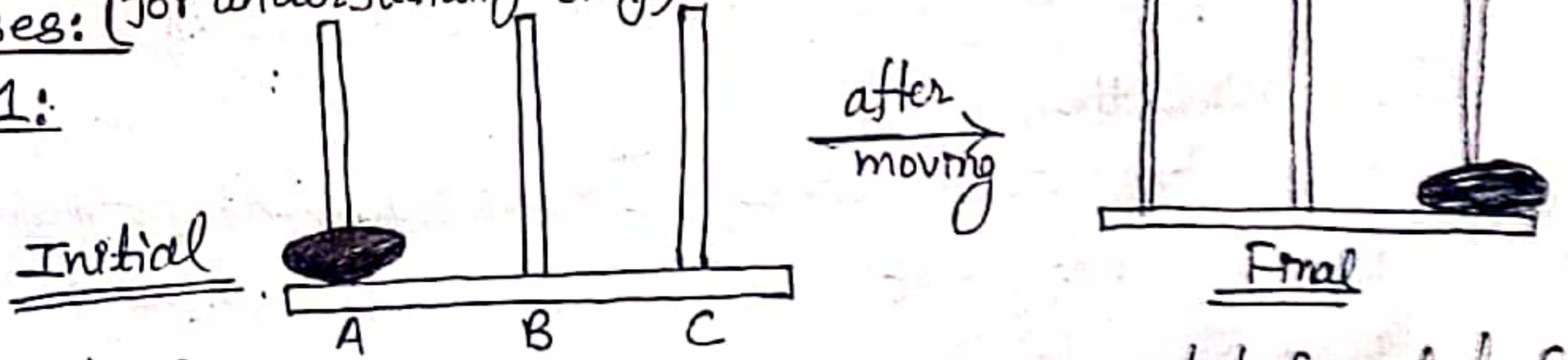
→ There are three poles named as A (origin), B(intermediate) and C (destination).
→ n number of different sized disks having hole at the center is stacked around the origin pole in decreasing order..
→ The disks are numbered as 1,2,3,...,n.

Objective: The objective of tower of Hanoi is to transfer all disks from origin pole to destination pole using intermediate pole for temporary storage.

Not imp

Cases: (for understanding only)

Case 1:



If only one disc is given then move a disk from A to C directly.

Case 2: If two disks are given then:
→ Move a disk from A to B
→ Then move a disk from A to C
→ Finally move a disk from B to C. (Same as single case since single remaining after moving one)

Case3: If 3 disks are given;

→ Move 2 disks from A to B ←
→ Move a disk from A to C (as case 1)
→ Move 2 disks from B to C. ← (as case 2)

∴ For n-disks

→ Move n-1 disks from A to B. using C.
→ Move a disk from A to C
→ Move n-1 disks from B to C using A

Conditions:

i) Move only one disk at a time.
ii) Each disk must always be placed around one of the pole.
iii) Never place larger disk on top of smaller disk.

[Imp] Algorithm:

Algorithm to move a tower of n disks from source to destination. (where n is positive integer).

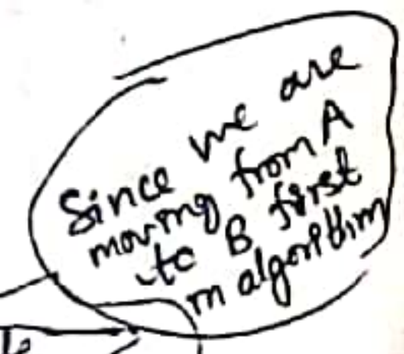
Step1: If n==1;
move a single disk from source to destination.

Step2: If n>1;
i) Let temp be the other pole than source and destination.
ii) Move a tower of (n-1) disks from source to temp.
iii) Move a single disk from source to destination.
iv) Move a tower of (n-1) disks from temp to destination.

Step3: Terminate.

⊛ Implementation of tower of Hanoi using C;

```
#include<stdio.h>
#include<conio.h>
void TOH(int, char, char, char); //function prototype.
void main(){
        int n;
        printf("Enter number of disks");
        scanf("%d", &n);
        TOH(n,'O','D','I'); //function call
        getch();
}
void TOH(int n, char A, char B, char C){
        if(n>0){
                TOH(n-1, A, C, B);
                printf("Move disk %d from %c to %c \n",
                        n, A, B);
                TOH(n-1, C, B, A);
        }
}
```



⊛ Applications of Recursion:

i) The most important data structure 'Tree' doesnot exist without recursion. We can solve that in iterative way also but that will be a very tough task.

ii) All puzzle games like Chess, Candy Crush etc broadly uses recursion.

iii) Recursion is the backbone of AI.

iv) Many of the well known sorting algorithms (like Quick, Merge etc.) uses recursion.

v) It is the backbone for searching.

⊛. Advantages of Recursion:

i) The code becomes much easier to write.
ii) It helps to solve some problems which are naturally recursive such as tower of Hanoi.
iii) Reduces unnecessary calling of functions.

⊛. Disadvantages of Recursion:

i) Recursive functions are generally slower than non-recursive functions.
ii) It may require a lot of memory to hold intermediate results on the system stack.
iii) ~~It is~~ It is difficult to think recursively so one must be very careful when writing recursive functions.