Data Structure and Algorithm [DSA]

By Er. Kushal Ghimire

Recursion

Recursion

- Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied.

- The process is used for repetitive computations in which each action is stated in terms of a previous result.

Principles of recursion:

Two important conditions must be satisfied by any recursive function:

1. Each time a function calls itself it must be closer, in some sense to solution.
2. There must be decision criterion for stopping the process or computation.

For designing good recursive program we must make certain assumptions such as:

1. ***Base case:*** It is the terminating condition for the problem while designing any recursive function. if condition in the recursive algorithm defines the terminating condition.
2. When a recursive program is subjected for execution, the recursive function calls will not be executed immediately.
3. The initial parameters input values are pushed onto the stack.
4. Each time a function is called a new set of local variable and formal parameters are again pushed onto the stack and execution starts from the beginning of the function using changed new value. This process is repeated till a base condition is reached.
5. Once the base condition or stopping condition is reached the recursive function calls popping elements from stack and return a result to the previous values of the function.
6. A sequence of returns ensures that the solution to the original problem is obtained.

Recursion vs Iteration

- Recursion is the technique of defining anything in terms of itself.
- There must be an exclusive if statement inside the recursive function, specifying stopping condition.
- Not all problems have recursive solution.
- Recursion is generally a worse option to go for simple problems, or problems not recursive in nature.

- Iteration is a process of executing a statement or a set of statements repeatedly, until some specified condition is specified.
- Iteration involves four clear-cut Steps like initialization, condition, execution, and updating
- Any recursive problem can be solved iteratively.
- Iterative counterpart of a problem is more efficient in terms of memory utilization and execution speed.

Recursion Example: Factorial

Factorial of number in a recursive manner is defined as:

Factorial(n) = 1 ,if n=0

= n*factorial(n-1), if n>0

/*Calculation of the factorial of an integer number using recursive function*/

```
#include<stdio.h>
#include<conio.h>
long int factorial(int n);
int main()
{
        int n;
        long int facto;
        printf("Enter value of n:");
        scanf("%d",&n);
        facto=factorial(n);
        printf("%d! = %ld",n,facto);
        return 0;
}
long int factorial(int n)
{
        if(n == 0)
                return 1;
        else
                return n * factorial(n-1);
}
```

Let's trace the evaluation of factorial(5):

Factorial(5)=

5*Factorial(4)=

5*(4*Factorial(3))=

5*(4*(3*Factorial(2)))=

5*(4*(3*(2*Factorial(1))))=

5*(4*(3*(2*(1*Factorial(0)))))=

5*(4*(3*(2*(1*1))))=

5*(4*(3*(2*1)))=

5*(4*(3*2))=

5*(4*6)=

5*24=

120

Recursion Example: Fibonacci sequence

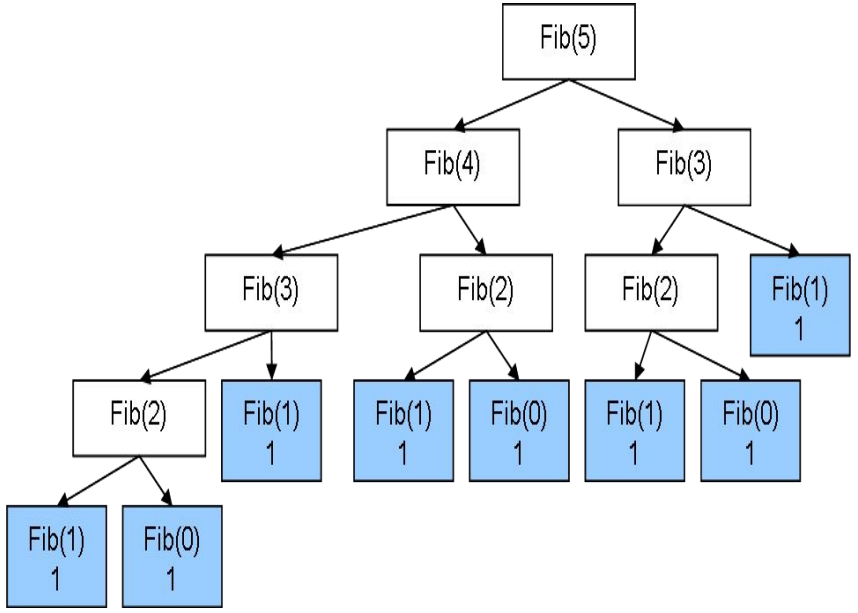
Fibonacci sequence is a series of positive integer in a manner that the next term of a series is the addition of previous two terms:
0,1,1,2,3,5,8,13,21,……
Fibonacci sequence in a recursive manner is defined as,

Fibseq(n) = 0, if n=0
= 1, if n=1
= Fibseq(n-1) +Fibseq(n-2), if n>1

/* Program to generate Fibonacci series up to n terms using recursive function*/

```
#include<stdio.h>
#include<conio.h>
int fibo(int);
int main()
{
        int n,i;
        printf("Enter n:");
        scanf("%d",&n);
        printf("Fibonacci numbers up to %d terms:\n",n);
        for(i=0;i<=n;i++)
        {
                printf("%d\n",fibo(i));
        }
        return 0;
}
int fibo(int n)
{
        if(n==0)
                return 0;
        else if(n==1)
                return 1;
        else
                return fibo(n-1)+fibo(n-2);
}
```

Draw a recursive tree diagram for the Fibonacci term fib(5)



Recursion Example: Tower of Hanoi (TOH) problem

Initial state:

- There are three poles named as origin, intermediate and 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, 4, ……,n.

Objective:

- Transfer all disks from origin pole to destination pole using intermediate pole for temporary storage.

Conditions:

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

Algorithm: - To move a tower of *n* disks from *source* to *dest* (where *n* is positive integer):

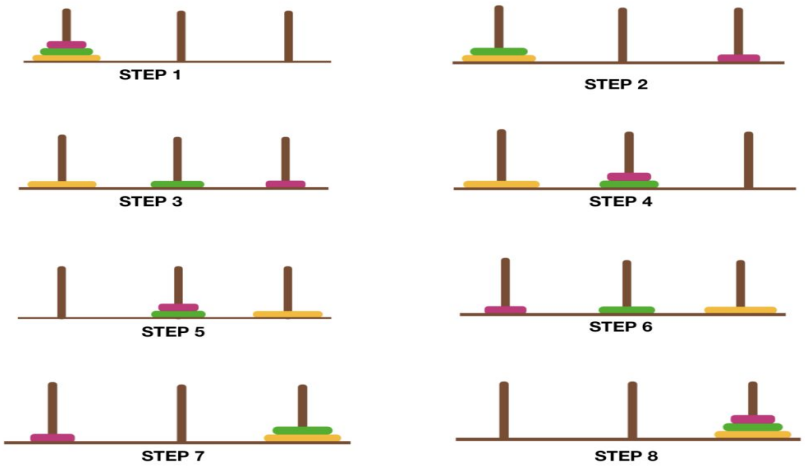
1. If *n* ==1:
    1.1. Move a single disk from *source* to *dest*.
2. If n > 1:
    2.1. Let *temp* be the remaining pole other than *source* and *dest*.
    2.2. Move a tower of (*n* – 1) disks form *source* to *temp*.
    2.3. Move a single disk from *source* to *dest*.
    2.4. Move a tower of (*n* – 1) disks form *temp* to *dest*.
3. Terminate.

Tower of Hanoi (TOH) problem

Contd...

Recursive solution of tower of Hanoi

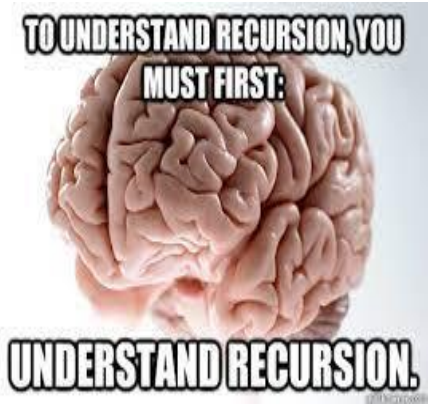
```
#include<stdio.h>
#include<conio.h>
void TOH(int, char, char, char); //Function prototype

void main()
{
        int n;
        printf("Enter number of disks");
        scanf("%d",&n);
        TOH(n,'O','D','I');
        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);
        }
}
```

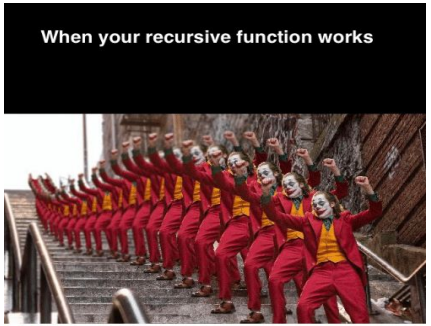



```
Enter number of disks3
Move disk 1 from O toD
Move disk 2 from O toI
Move disk 1 from D toI
Move disk 3 from O toD
Move disk 1 from I toO
Move disk 2 from I toD
Move disk 1 from O toD
```

Application of recursion

1. Recursion is applied for construction of binary search tree.
2. Recursion is applied for tree traversals such as in-order, pre-order and post-order traversal.
3. Recursion is applied to solve Tower of Hanoi (TOH) problem.
4. Recursion is applied for divide and conquer algorithm.
5. Recursion is applied for problem decomposition into smaller sub-problems.





All I have are recursive thoughts...