Recursion Part-I

Wednesday, July 28, 2021 3:17 PM

EXAMPLE :
A child couldn't sleep, so her mother told her a story about a little frog who couldn't sleep, so the frog's mother told her a story about a little bear, who couldn't sleep, so the bear's mother told her a story about a little who fell asleep.
... and the little beat fell asleep;
... and the frog fell asleep;
... and the child fell asleep

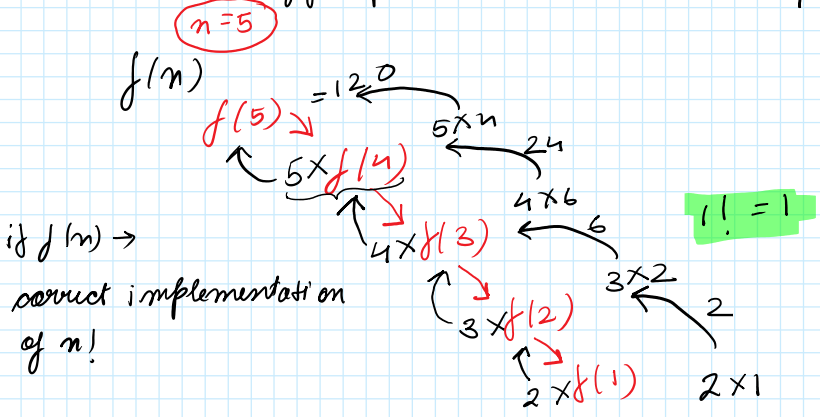
Q. What is recursion?
Ans : Mathematical definition : It is a logical procedure which is specified by a sub procedure that yields values or instances of a function repeatedly by applying given routine operation.
Fundamental definition : Recursion is defined when a function calls itself by applying some sub-routine on the parameters by keeping an extra space overhead.

Factorial :

$f(n) = n \times f(n-1)$ $\quad 5! = 5 \times 4!$

f^n that returns n!

breakdown bigger problem → smaller sub-problem



if f(n) → correct implementation of n!

Principle of Mathematical Induction → Recursion

1 + 2 + 3 + ... n (sum of n natural numbers) = n(n+1)/2

3 steps

1 → what is the minimal value for which we know the ans

2 → Assume the formula works for n = k

↳ $\frac{k(k+1)}{2}$

3 → Prove the formula works for k+1 $= \frac{(k+1)(k+2)}{2}$

Recursion :

Recursion:

1. Find out the smallest subproblem for which we know the ans
2. Assume that for the given problem, recursion will correctly calculate a subproblem
3. Selfwork

$f(n) = n \times f(n-1)$

1.) $n = 1$, $f(1) = 1$ (base case)

2.) Calculate $f(n-)$ → recursive assumption

3.) selfwork → return $n \times f(n-1)$

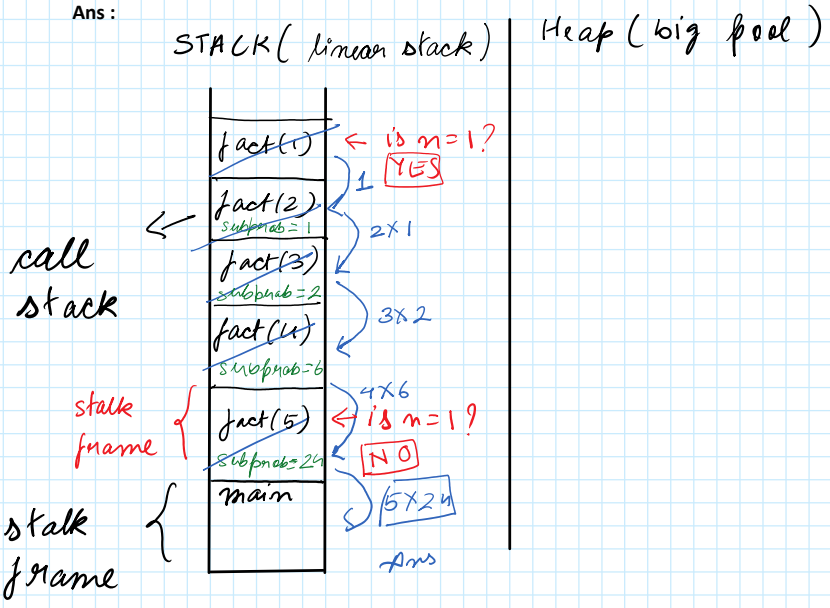
```
Int factorial(int n)
{
        //base case
        If(n  ==1)
                Return 1;

        //recursive assumption
        Int subprob = factorial(n-1);

        //self work
        Return n*subprob;
}
```

Q. How memory is distributed for a process?

Ans :



return function removes the function from the stack

$f(n)$ → factorial

TC → $O(n)$

SC → defined as the max. space allocated at any point of time during execution of the process.

$O(n)$

of the process.

$O(n)$

Why do we need recursion?
Code becomes really short.
Not unlimited space in stack. Chances of STACKOVERFLOW.

Q. Fibonacci series. To find n^{th} fibonacci.

0, 1, 1, 2, 3, 5, 8, 13, ...

→ i) Base case:

0^{th} fib = 0

1^{st} fib = 1

i^{th} term is sum of previous 2 terms

ii) Recursion intution:

calculate fib(n-1) and fib(n-2)

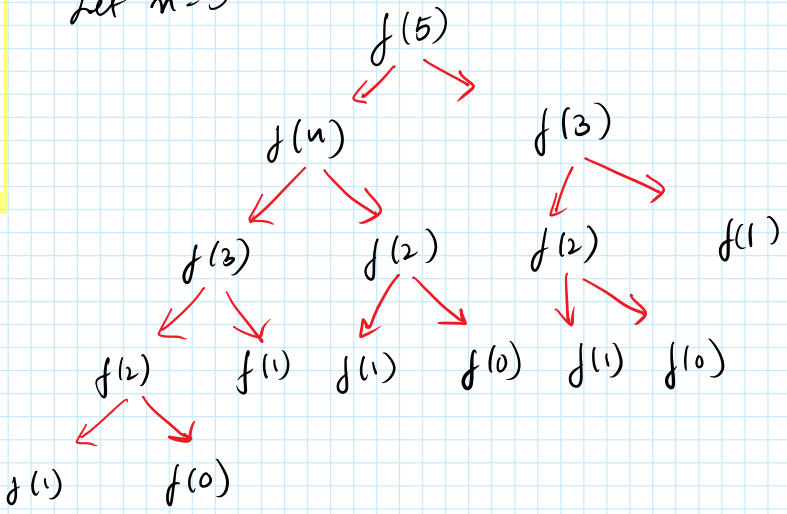
iii) Self work:

return fib(n-1)+fib(n-2)

```
Int fib(int n)
{
        If(n == 0 || n==1)
                Return n;

        Return fib(n-1)+fib(n-2);
}
```

Let n = 5



Recurrence Relation : eg are fibonacci series and factorial

Q. Print first N natural numbers recursively.

N = 5

1
2
3
4
5

Ans:-

```
Void printnatural(int n)
{
        If(n == 0)
                Return;

        Printnatural(n-1);

        Cout<<n<<endl;
}
```

1
2
3
4
5

```
Void printnatural(int n)
{
        If(n == 0)
                Return;

        Cout<<n<<endl;

        Printnatural(n-1);

}
```

5
4
3
2
1

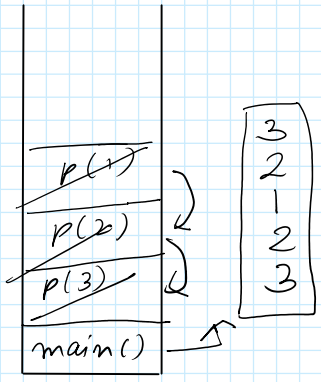
Q. Print

5
4
3
2
1
2
3
4
5

Ans:-

```
Void pattern(int n)
{
        If(n==1)
        {
                Cout<<1<<endl;
                Return;
        }
        cout<<n<<endl;
        pattern(n-1);
        cout<<n<<endl;
}
```

Let (n = 3)



Q. Given a value 'n', how many binary strings of length n are there with no consecutive ones.

Ans:-

n = 3

0 0 0
0 0 1
0 1 0
1 0 0
1 0 1
1 1 0 X
1 1 1 X

5

n = 4

0 0 0 0
0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 0
1 0 1 0
0 1 0 1
1 0 0 1

8

n = 1 0 1 2

BC →

n = 1 | 0
1 | 2 | BC →
if (n = 1)
return 2;

n = 2 | 0 0
1 0
0 1 | 3 | if (n = 2)
return 3;

n = 3 | | 5 | (3+2)

n = n | | 8 | (5+3)

⋮

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