Agenda : Recursion ?

How to write recursive code / tracing

TC / SC of recursive code → wednesday's session

Why recursion ?

1. Binary tree / BST / Segment Trees / Tries
2. Dynamic Prog
3. Back
4. Graphs

Recursion : Function calling itself is c/a Recursion.

$\Rightarrow$ Solving problem, using smaller instance of same problem. c/a subproblem.

eg: $sum(N) = 1 + 2 + 3 + \cdots + N-1 + N$

$sum(N) = \underline{sum(N-1)} + N$

↳ smaller instance of same problem } subproblem.

② How to write recursive code?

1) Assumption $\Rightarrow$ decide what your function does

2) Main Logic $\Rightarrow$ solve assumption using subproblems.

3) Base cond^n $\Rightarrow$ Input for which we need to stop

and RETURN

?

Q1 int sum(N) [Ass: Given N, calculate sum of N natural numbers.

```
int sum(N)
{
    if(n==1) return 1;

    return sum(N-1) + N;
}
```

$\underbrace{\text{sum(N-1)}}$ ↳ sum of 1st (N-1) natural numbers.

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

Q2 int fact(N) [Ass: calc. & return N!

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

Function call Tracing

```
int add(int x, int y)
{
    return x+y;
}

int mul(x,y)
{
    return x*y
}

int sub(x,y)
{
    return x-y;
}

main()
{
    x=10   y=20
    print(sub(mul(add(x,y),30),75);
}
```

$$\text{sub}(\overbrace{\text{mul}(\text{add}(x,y)\ ,30)}^{900}, 75) \Downarrow 500-75 = 825$$

↳ $\text{mul}(\overbrace{\text{add}(x,y)}^{30}, 30) \Rightarrow 900$

↳ $\text{add}(x,y) \Rightarrow 30$

add($\overset{10}{x}$, $\overset{20}{y}$) return 30 // once returned, it will come out.

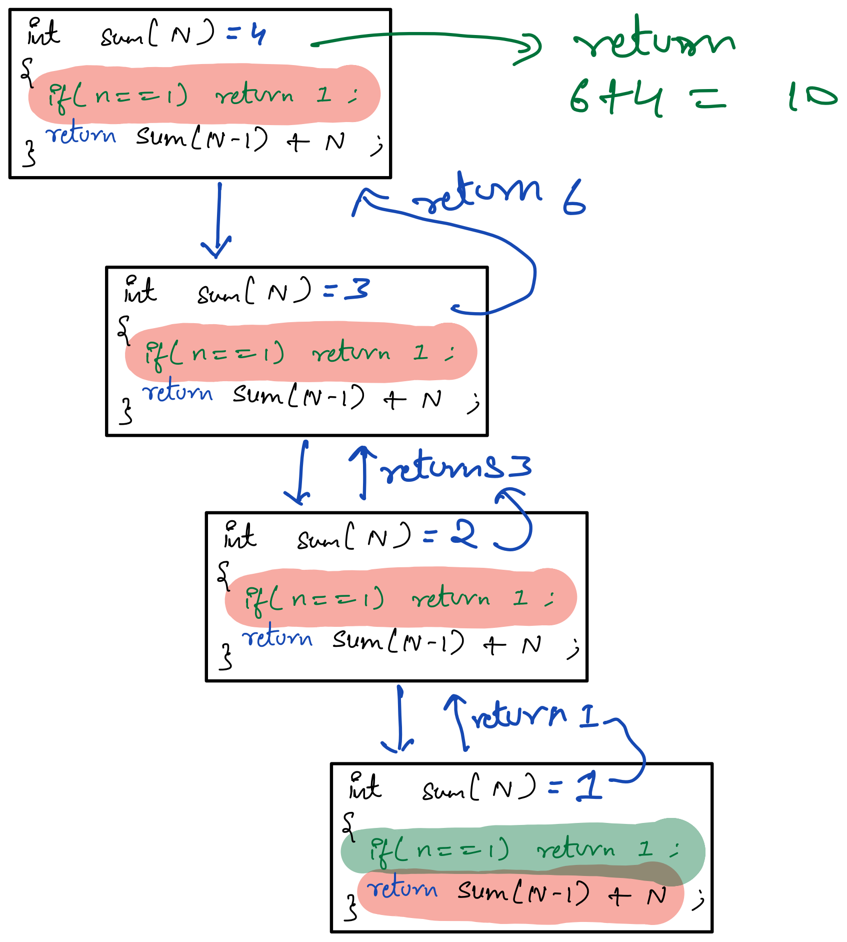
mul(add(x,y), 30) retun 900 // once returned, it will come out.

sub(mul(add(x,y), 30), 75) return 825

Obs 1 : → Whenever function calls, insert it a top

obs 2 : → whenever function returns, come out of stack.

obs 3 ⇒ Insert at top, delete from top. c/a Stack.

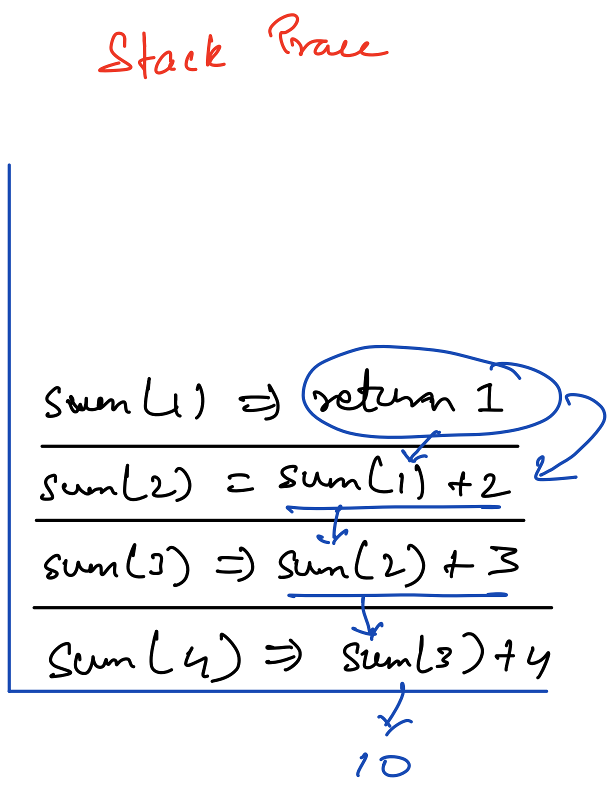


```
int sum( N ) = 4
{
  if( n==1) return 1 ;
  return sum(N-1) + N ;
}

int sum( N ) = 3
{
  if( n==1) return 1 ;
  return sum(N-1) + N ;
}

int sum( N ) = 2
{
  if( n==1) return 1 ;
  return sum(N-1) + N ;
}

int sum( N ) = 1
{
  if( n==1) return 1 ;
  return sum(N-1) + N ;
}
```



Without Base condition, func^n will not stop.

Infinite loop

TLE

insert func^n at top of stack

⇓

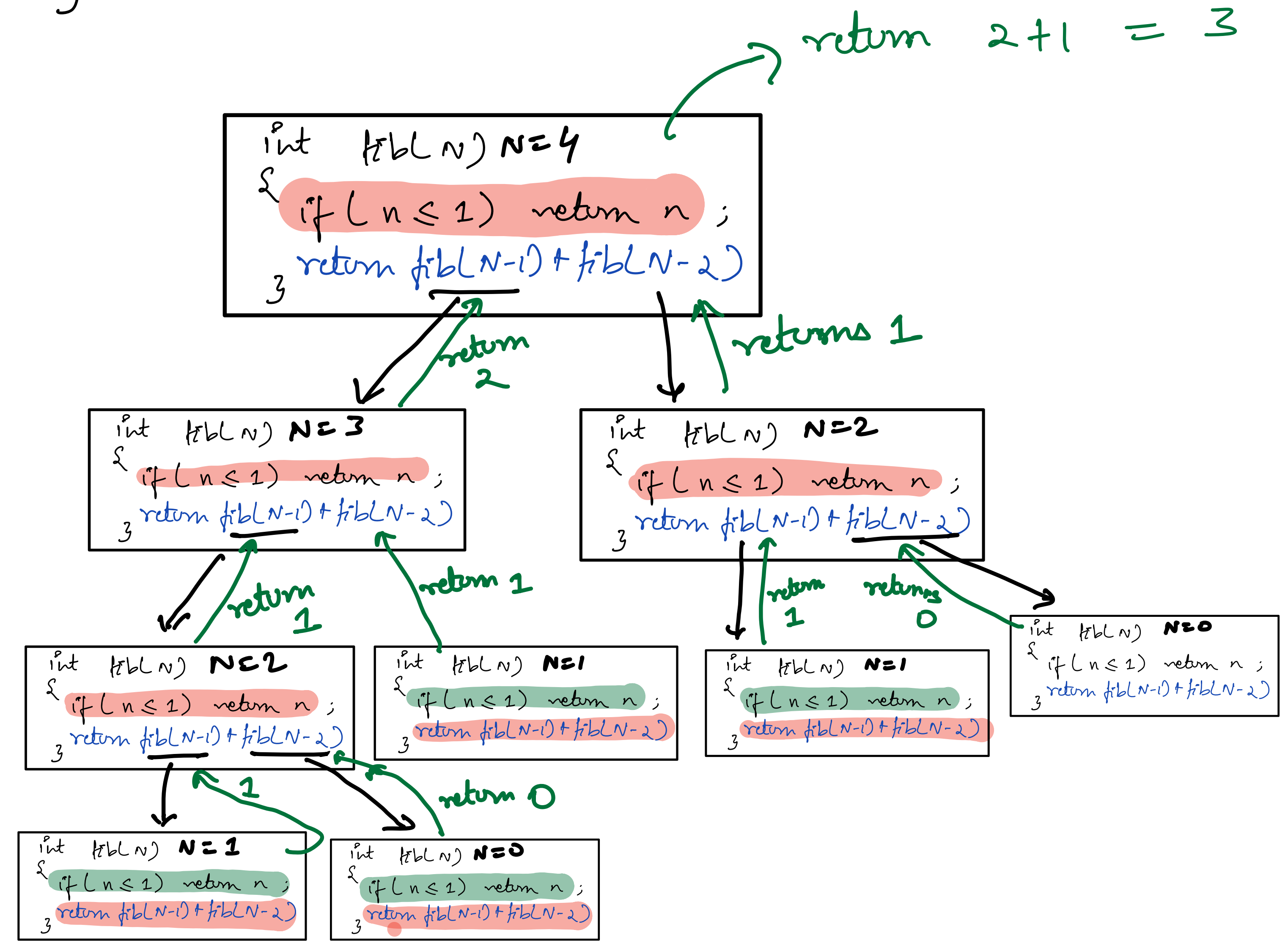
Stack overflow

Note: In recursion, if your code gives memory limit exceeded, that means code is not properly stopped. Verify base cond^n.

Q3 Fibonacci series

i/p	0	1	2	3	4	5	6
Fib( )	0	1	1	2	3	5	8

```
int fib(N)  Ass: find & return Nth fibonacci number.
{
    if (n ≤ 1) return n ;
   return fib(N-1) + fib(N-2)
}
```



return 2+1 = 3
int fib(N) N=4 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
return 2
returns 1
int fib(N) N=3 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
int fib(N) N=2 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
return 1
return 1
return 1
returns 0
int fib(N) N=2 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
int fib(N) N=1 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
int fib(N) N=1 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
int fib(N) N=0 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
1
return 0
int fib(N) N=1 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }
int fib(N) N=0 { if (n ≤ 1) return n ; return fib(N-1) + fib(N-2) }

Stack Trace → TODO

Q Given N, print all numbers from 1 → N in increasing order.

Inc(3) ⇒ 1 2 3

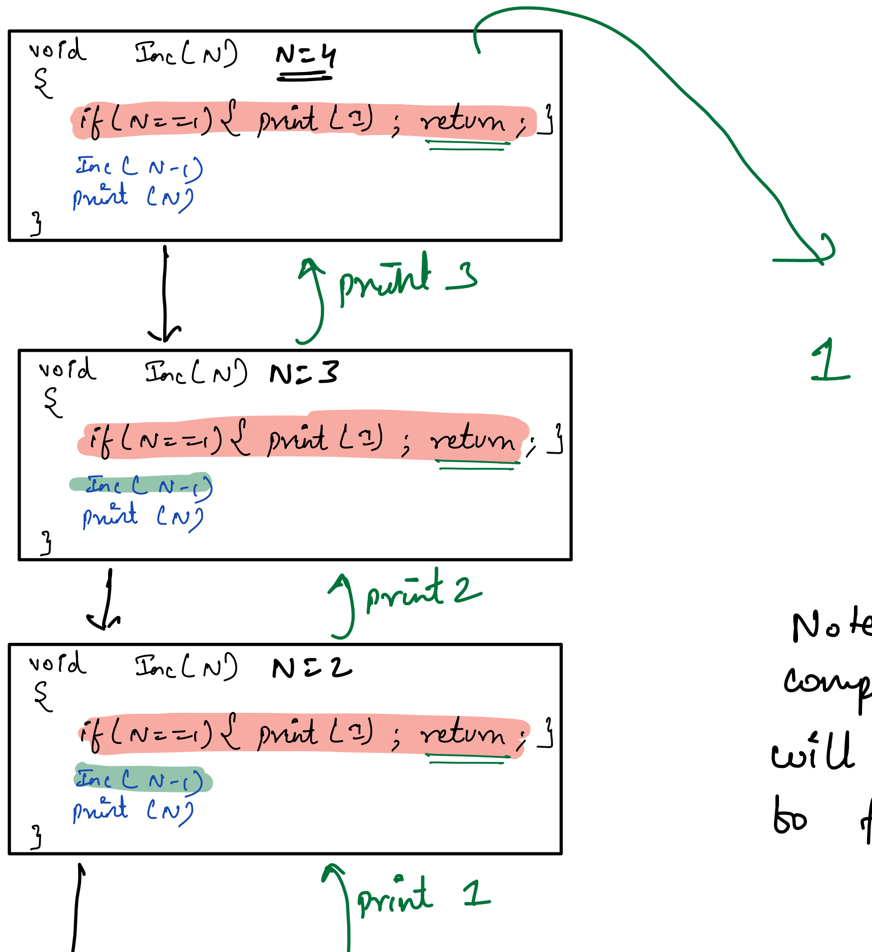
Inc(4) ⇒ 1 2 3 4

Inc(N) ⇒ 1 2 3 . - - . - - N-1 N

~~print (N)~~ ~~Inc (N-1)~~ ⇓ will print in descending order.

Inc (N-1) ✓ print (N)

```
void  Inc(N)   Ass: Given N, print all numbers from 1→N
{
    if(N==1){ print(1) ; return ; }
    Inc ( N-1)
    print (N)
}
```



Note: When func^n is completely executed, it will automatically return to func^n which call.

```
void   Inc(N)   N = 1
{
   if(N==1){ print(1) ; return ; }
   Inc(N-1)
   print(N)
}
```

Q5 Given a substring, check if it is a palindrome or not.

eg: good**dad** ✓ s=4 s=6 return true

eg. go**oddad** s=2 s=6 return false.

eg: s s+1 - - - - . e-1 e

To check ch[s, e]

Check ch[s] == ch[e]
&&
isPal(ch[], s+1, e-1) should be palindrome

Ass: return if given substring [s, e] is palindrome or not.

```
bool   isPal( char ch[], int s, int e)
{
    if( s > e) { return true; }

    if( ch[s] == ch[e]
           && isPal( ch[], s+1, e-1))
    {
        return true;
    }
    return false.
}
```

i/p = $\overset{0\,1\,2\,3\,4\,5}{\text{maddam}}$ S = 0   e = 5

```
                                        0         5
bool  isPal ( char ch[] , int  s ,  int  e)
{
 if( s>e) { return true; }
 if( ch[s] == ch[e]
                   &&
                   isPal( ch[], s+1, e-1))
 {
     return  true ;
 }
 return  false.
}
```

→ return true.

↓ ↑ true

```
                                        1         4
bool  isPal ( char ch[] , int  s ,  int  e)
{
 if( s>e) { return true; }
 if( ch[s] == ch[e]
                   &&
                   isPal( ch[], s+1, e-1))
 {
     return  true ;
 }
 return  false.
}
```

↓ ↑ true.

```
                                        2         3
bool  isPal ( char ch[] , int  s ,  int  e)
{
 if( s>e) { return true; }
 if( ch[s] == ch[e]
                   &&
                   isPal( ch[], s+1, e-1))
 {
     return  true ;
 }
 return  false.
}
```

↓ ↑ true

```
                                        3         2
bool  isPal ( char ch[] , int  s ,  int  e)
{
 if( s>e) { return true; }
 if( ch[s] == ch[e]
                   &&
                   isPal( ch[], s+1, e-1))
 {
     return  true ;
 }
 return  false.
}
```