Recursion → Function calling itself.

What is recursion? ✓
How to write recursive code. ✓
How to devise easy recursive codes. ✓

TC & SC → next class

Used → 1) Merge Sort / Quick Sort
2) Binary Tree / BST
3) Backtracking
4) Dynamic Programming
5) Graphs

Use → solving answer for current problem using subproblems.

smaller instance of same problem.

Q → Given a positive integer N, find sum of first N natural numbers. $N > 0$

N = 4 → 1 + 2 + 3 + 4 = 10

$$\Sigma N = \frac{N*(N+1)}{2}$$

N = 5 → 1 + 2 + 3 + 4 + 5

sum(5) = sum(4) + 5 → sum(N) = sum(N-1) + N

sum(5): current problem; sum(4): subproblem

Steps to write recursive code →

1) Define what the function do. → int sum(N) {... } sum of first N natural numbers

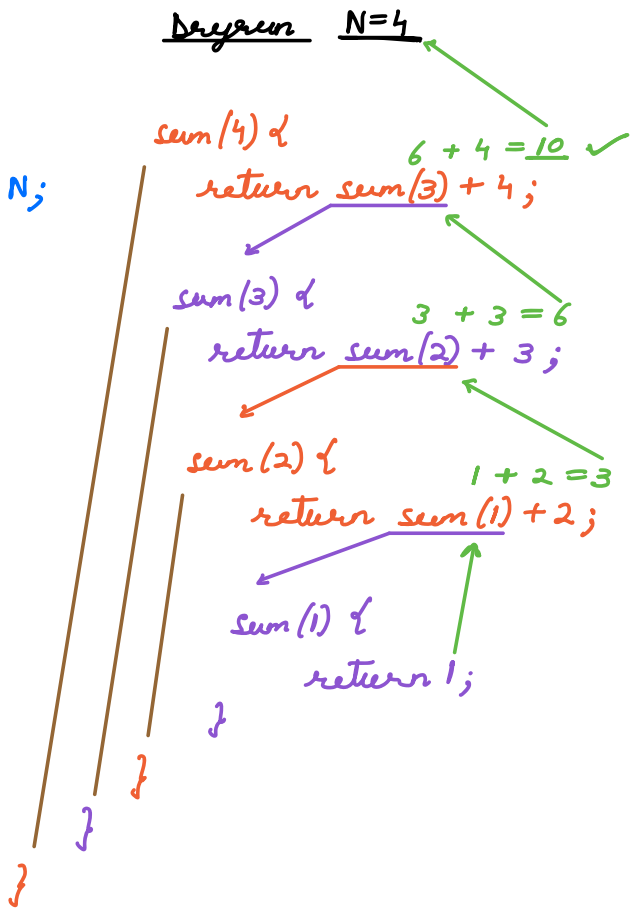
2) Build logic on how to use subproblems to solve the current problem. → sum(N) = sum(N-1) + N

3) Define base case.
↓
smallest subproblem for which we already know the answer.

sum(1) = 1

↓
sum(N-2) + (N-1)
↓
sum(N-3) + (N-2)
⋮

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



Q → Write recursive code to find factorial of N. N>0

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

N = 5 ⟶ 1*2*3*4*5

fact(N) = fact(N-1) * N

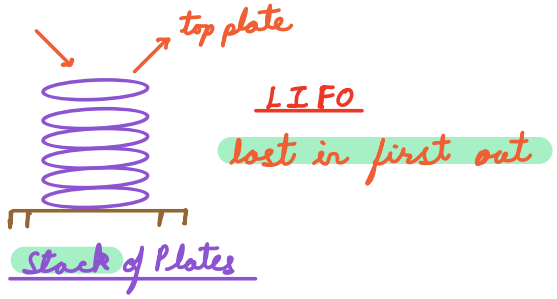
fact(1) = 1

```
long fact(N) {
    if (N==1)
        return 1;
```

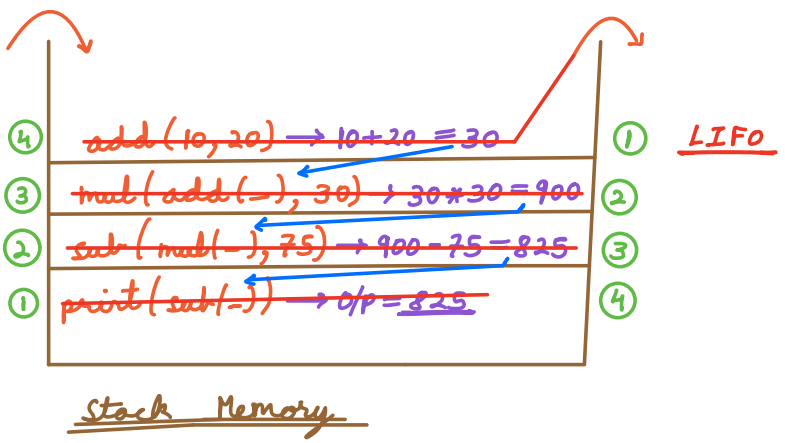
```
    return fact(N-1) * N;
}
```

Function call using recursion stack

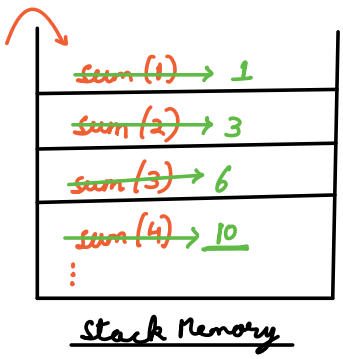
```
int add (x, y) {
    return x+y ;
}
int mul(x, y) {
    return x*y ;
}
int sub(x, y) {
    return x-y ;
}
```

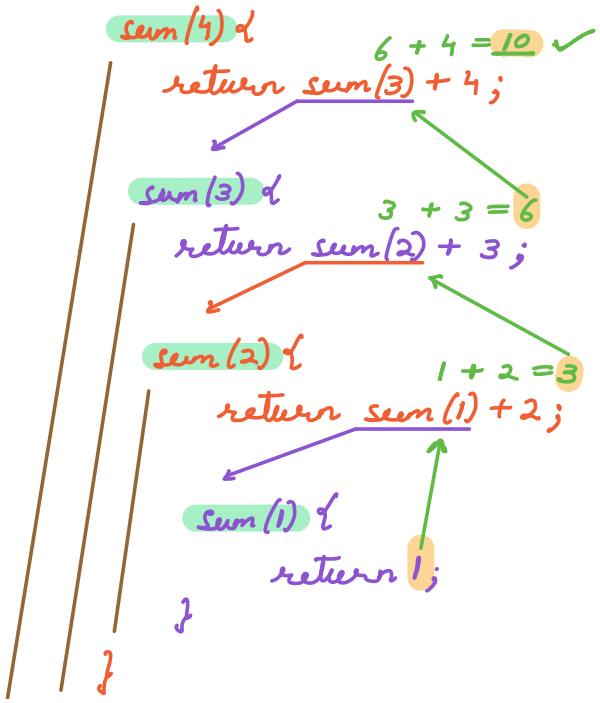


① ② ③ ④
print (sub (mul (add (10, 20), 30), 75))



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



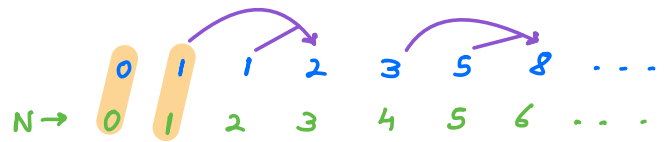


```
    }
}
```

10:25 PM

Q → **Fibonacci Numbers**

(sum of previous two numbers is the current number)



Find N^{th} fibonacci number using recursion.

$$fib(N) = fib(N-1) + fib(N-2)$$

$$fib(0) = 0 \qquad fib(1) = 1$$

```
int fib(N) {
    if (N==0 || N==1)
        return N;
    return fib(N-1) + fib(N-2);
}
```

① ② → 3

```
fib(3) {
    return fib(2) + fib(1);

    fib(2) {
①       return fib(1) + fib(0);

        fib(1) {
①           return 1;
        }
        fib(0) {
②           return 0;
        }
    3   return 1+0;
    }
    fib(1) {
②       return 1;
    }
```

~~fib(1)~~ → 1 ~~fib(0)~~ → 0
~~fib(2)~~ → 1 ~~fib(1)~~ → 1
~~fib(3)~~ → 2

Stack Memory

```
int fib(N) {
    if (N==0 || N==1)
        return N;
    return fib(N-1) + fib(N-2);
}
```

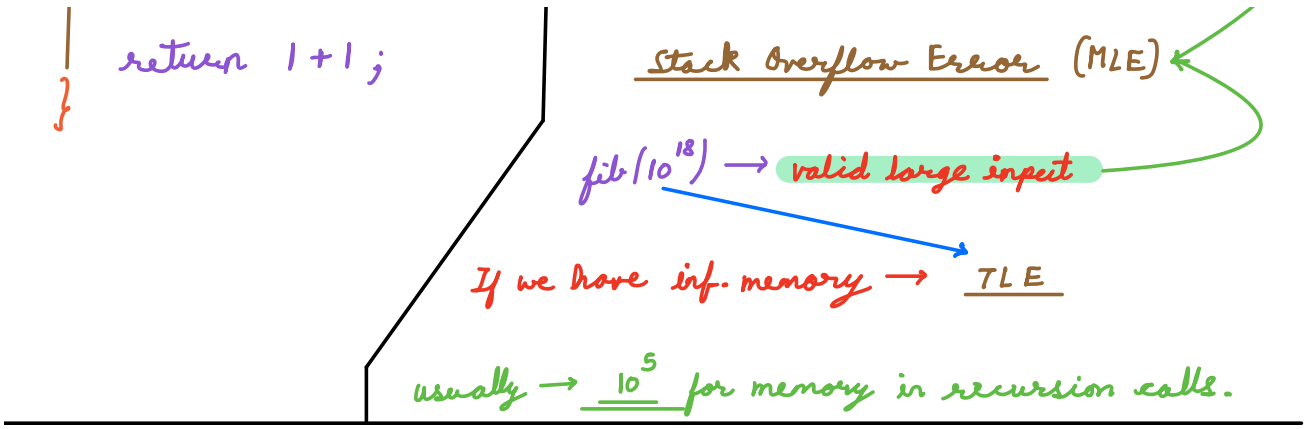
① ② → 3

fib(-8) → invalid input

fib(-8) → fib(-9) + fib(-10) . . .

↓

fib(-10) + fib(-11) . . .



Q → Given a positive integer N,
print first N natural numbers from 1 to N. (using recursion)

N=5 → o/p = 1 2 3 4 5 (N=4) ✓ N=4 → o/p = 1 2 3 4

1) void prt(N) {...}
2) prt(N) → prt(N-1)
   print(N)
3) N==1 → print(1)

```
void prt(N) {
    if (N==1) {
        print(1);
        return;
    }
① prt(N-1); → recursion call
② print(N); → o/p → N
}
```

N=4

```
prt(4) {
    prt(3) {
        prt(2) {
            prt(1) {
                print(1); → 1
            }
            print(2); → 2
        }
        print(3); → 3
    }
    print(4); → 4
}
```

o/p → 1 2 3 4

H.W → What is o/p of → N=4
(Easy)

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

```
    }
}   ② pet(N-1);
```

Q → Give a string, check if it is a palindrome using recursion.

s = reverse(s)

0 ↑ ↑ N-1

1) boolean isPalindrome(s, l, r) {...} Eg → madam, nanan, abccba, malayalam, etc

2) malayalam
0 1 2 3 4 5 6 7 8

isPalindrome(s, 0, N-1) = (s[0] == s[N-1]) &&
isPalindrome(s, 0+1, (N-1)-1)

isPalindrome(s, l, r) = (s[l] == s[r]) &&
isPalindrome(s, l+1, r-1)

3)

malayalam	l	r
malayalam	0	8
alayala	1	7
layal	2	6
aya	3	5
y → true	4	4

l == r

abcbbcba	l	r
abcbbcba	0	7
bcbbcb	1	6
cbbc	2	5
bb	3	4
empty string → true	4	3

l > r

if (l >= r) → return true;

H.W → code + Dryrun
