

Recursion  $\rightarrow$  Function calling itself.

Sum of first  $N$  natural numbers  $\rightarrow \frac{N * (N+1)}{2}$

$$\text{sum}(5) = \boxed{1 + 2 + 3 + 4} + 5$$

$\text{sum}(4)$

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

Steps for Recursion  $\rightarrow$

- 1) Decide what the function exactly do.
- 2) Divide the problem into smaller subproblems & use subproblems to get the answer.
- 3) Define base case (smallest subproblem).

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

Function call tracing

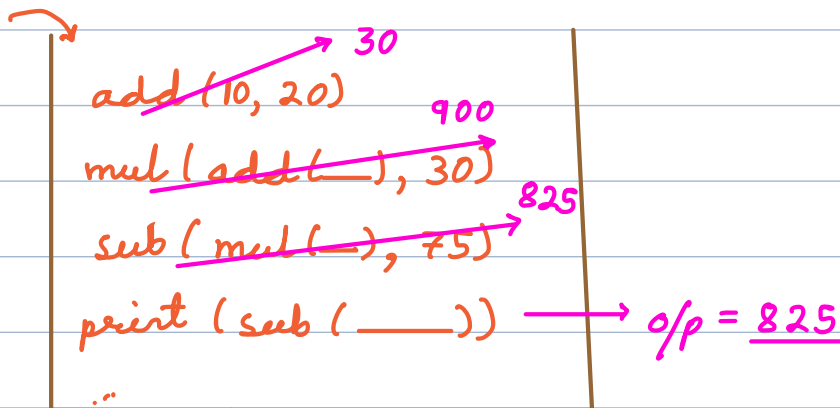
```
int add(x, y) {  
    return x + y;  
}
```

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

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

$x = 10$     $y = 20$

```
print(sub(mul(add(x, y), 30), 75))
```



Stack (Last In First Out)

Q → Find factorial of  $N$  using recursion.

$$N! = 1 * 2 * 3 * \dots * N$$

$$5! = \boxed{1 * 2 * 3 * 4} * 5 = \underline{120}$$

$4!$

- 1) long fact( $N$ ) {...}
- 2) fact( $N$ ) = fact( $N-1$ ) \*  $N$
- 3) fact(1) = 1 / fact(0) = 1

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

```
fact(4) {
    return fact(3) * 4
}
fact(3) {
    return fact(2) * 3
}
fact(2) {
    return fact(1) * 2
}
fact(1) {
    return 1
}
```

Diagram showing the recursive calls for fact(4):

- fact(4) calls fact(3) \* 4 (result 24)
- fact(3) calls fact(2) \* 3 (result 6)
- fact(2) calls fact(1) \* 2 (result 2)
- fact(1) returns 1

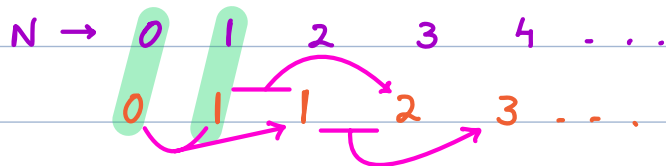
```

fact(2) {
    return fact(1) * 2
}
fact(1) { return 1 }
}
}

```

$1 * 2$

Q → Find  $N^{\text{th}}$  fibonacci number using recursion.



$\text{fib}(5) = \underline{5}$

$\text{fib}(6) = \underline{8}$

$\text{fib}(7) = \underline{13}$

- 1) int fib(N) { ... }
- 2)  $\text{fib}(N) = \text{fib}(N-1) + \text{fib}(N-2)$
- 3)  $\text{fib}(0) = 0$   
 $\text{fib}(1) = 1$ 

2 subproblems req  $\Rightarrow$  2 base case

```

int fib(N) {
    if (N <= 1) return N
    return fib(N-1) + fib(N-2)
}

```

① ③ ②

```

fib(4) {
    fib(3) {
        fib(2) {

```

```

    fib(1) { return 1 }
    fib(0) { return 0 }
    return 1 + 0 = 1
}
fib(1) { return 1 }
return 1 + 1 = 2
}
fib(2) {
    fib(1) { return 1 }
    fib(0) { return 0 }
    return 1 + 0 = 1
}
return 2 + 1 = 3
}

```

---

### Time Complexity

→ (Time taken by 1 function call) \* (# function calls)

### Space Complexity

→ Max size of stack at any point.

→ Height of recursive tree.

↘ tree generated while tracing recursive code.

long fact(N) {

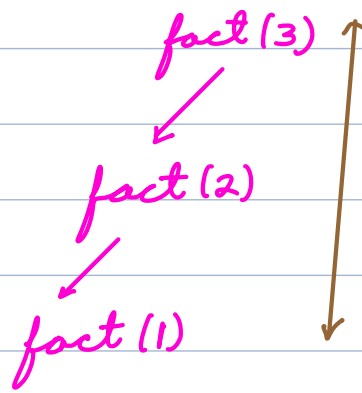
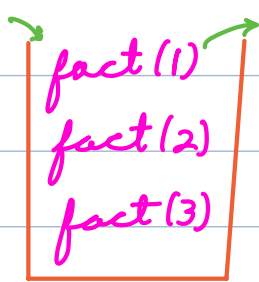
$$TC = O(1 * N) = \underline{O(N)}$$

if (N <= 1) return 1

$$SC = \underline{O(N)}$$

return fact(N-1) \* N

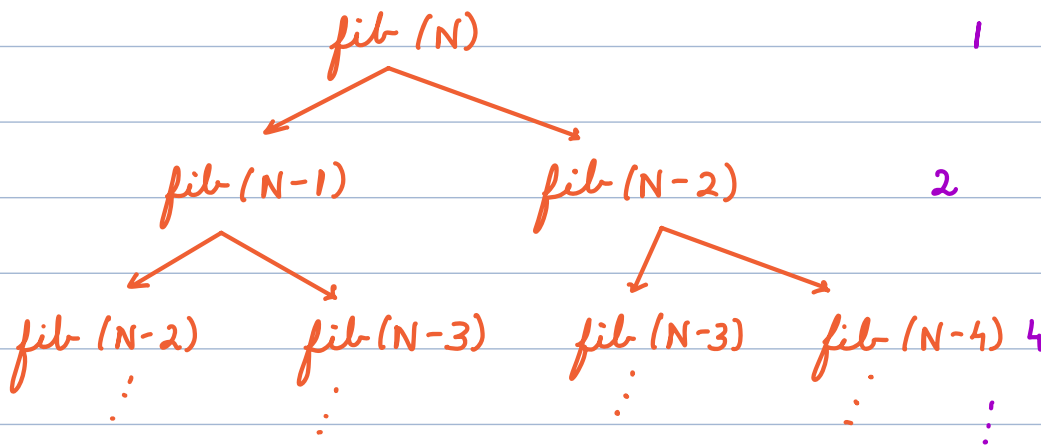
}



```

int fib(N) {
    if (N <= 1) return N
    return fib(N-1) + fib(N-2)
}

```



fib(1) / fib(0)  $\sim 2^{N-1}$

$$\begin{aligned} \# \text{ function calls} &= 1 + 2 + 4 + 8 + \dots + 2^{N-1} \\ &= \underline{2^N - 1} \end{aligned}$$

$$TC = O(1 \times 2^N) = \underline{O(2^N)}$$

$$SC = \underline{O(N)}$$


---

Q → Given an integer  $N$  ( $N > 0$ ), print all numbers from 1 to  $N$  using recursion.

$N = 3$

o/p → 1 2 3  
          print(2)

- 1) void inc(N) { ... }
- 2) inc(N-1)  
   print(N)
- 3) if (N==0) return / if (N==1) { print(1)  
  return }

```
void inc(N) {  
    if (N==0) return  
    inc(N-1)  
    print(N)  
}
```

o/p → 1 2 3

$TC = \underline{O(N)}$

$SC = \underline{O(N)}$

```
inc(3) {  
    inc(2) {  
        inc(1) {  
            inc(0) { return }  
            print(1)  
        }  
        print(2)  
    }  
    print(3)  
}
```

Q → Whirlpool wants to design a timer for the washing machines, which is a simple countdown timer. When user sets a time the washing machine needs to show each minute passing till it reaches 0. (Recursively)

$N = 5$       o/p → 5 4 3 2 1 0

```
void dec(N) {  
    if (N == 0) { print(0)  
                  return }  
    print(N)  
    dec(N-1)  
}
```

o/p → 5 4 3 2 1 0

$TC = O(N)$

$SC = O(N)$

```
dec(5) {  
    print(5) ✓  
    dec(4) {  
        print(4) ✓  
        dec(3) {  
            print(3) ✓  
            dec(2) {  
                print(2) ✓  
                dec(1) {  
                    print(1) ✓  
                    dec(0) { print(0) ✓  
                        return }  
                    }  
                }  
            }  
        }  
    }  
}
```

$\{$

$\}$

$\}$

---