Recursion → Function calling itself.

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

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

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

Steps for Recursion →

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))

30

~~add~~ (10, 20)

900

mul (~~add (__)~~, 30)

825

sub (~~mul (__)~~, 75)

print (sub (_____)) → o/p = <u>825</u>

...

Stack (Last In First Out)

Q → Find factorial of N using recursion.

N! = 1 * 2 * 3 * . . . * N

5! = $\boxed{1 * 2 * 3 * 4}$ * 5 = <u>120</u>

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
}
```

24

```
fact (4) {
    return fact (3) * 4
```

6 * 4

```
fact (3) {
    return fact (2) * 3
```

2 * 3

fact (2) {

return fact (1) * 2   (1*2)

fact (1) { return 1 }

}

}

}

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

N → 0   1   2   3   4 . . .

0   1   1   2   3 . . .

fib (5) = 5   fib (6) = 8   fib (7) = 13

1) int fib (N) { . . . }
2) fib (N) = fib (N-1) + fib (N-2)   2 subproblems req ⇒ 2 base case
3) fib (0) = 0
   fib (1) = 1

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

① fib (N-1) ③ + ② fib (N-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) {
    if (N <= 1) return 1
    return fact (N-1) * N
}
```

TC = $O(1 * N) = O(N)$

SC = $O(N)$

fact(1)
fact(2)
fact(3)

fact(3)

fact(2)

fact(1)

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

fib(N) 1

fib(N-1) fib(N-2) 2

fib(N-2) fib(N-3) fib(N-3) fib(N-4) 4

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

function calls $= 1 + 2 + 4 + 8 + \ldots + 2^{N-1}$

$= \underline{2^N - 1}$

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

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

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

N = 3 o/p → $\boxed{1 \quad 2}$ 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 = O(N)

SC = 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 $\boxed{4 \quad 3 \quad 2 \quad 1 \quad 0}$

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

```
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 }
                }
            }
```

o/p → 5 4 3 2 1 0

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

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