Recursion
- Merge sort & Quick sort
- Trees (Binary Tree | BST)
- Heap
- Segment Tree
- Dynamic Programming
- Backtracking

Observations :

1) Size keeps decreasing
2) All the dolls are same, except size.
3) After last doll, we can't break further.

Recursion

⇒ Function calling itself.

⇒ Recursion is a way of solving a problem using smaller sub-problem.

Same problem of small size.

Ex

$$Sum(N) = \underbrace{1 + 2 + 3 + 4 + 5 + \cdots + (N-1)}_{Sum(N-1)} + N.$$

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

Steps to write the recursive code:

1) **Assume | Trust** your recursive code will work

2) **Main logic**: Solve the bigger problem using smaller subproblem.

3) **Base | Exit Condition**
   ↳ Decide when recursion should stop.

```
int Sum(int N) {
    if(N==1)
        return 1;
    // Main logic
    return Sum(N-1) + N;
}
```

1) assumption:
Sum(N) ⇒ returns sum of first N natural no's.

2) Main logic:
Sum(N) = Sum(N-1) + N.

3) Exit Cond^
Sum(1) → 1

Ex Factorial of a number.

$$N! = \underbrace{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdots (N-1)}_{(N-1)!} \cdot N$$

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

```
int fact(int N) {
    if(N==0)
        return 1;
    return fact(N-1) * N;
}
```

1) Trust :

fact(N) ⇒ N!

2) Main Logic

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

3) Base Cond^ :-

1! = 1

0! = 1

Ex: Fibonacci Series.

Indexes	0	1	2	3	4	5	6	7	8	9	10
Series:	1	1	2	3	5	8	13	21	34	55	89 ...

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

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

1. Assumption

fib(N) → N^th fibonacci no

2. Main Logic

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

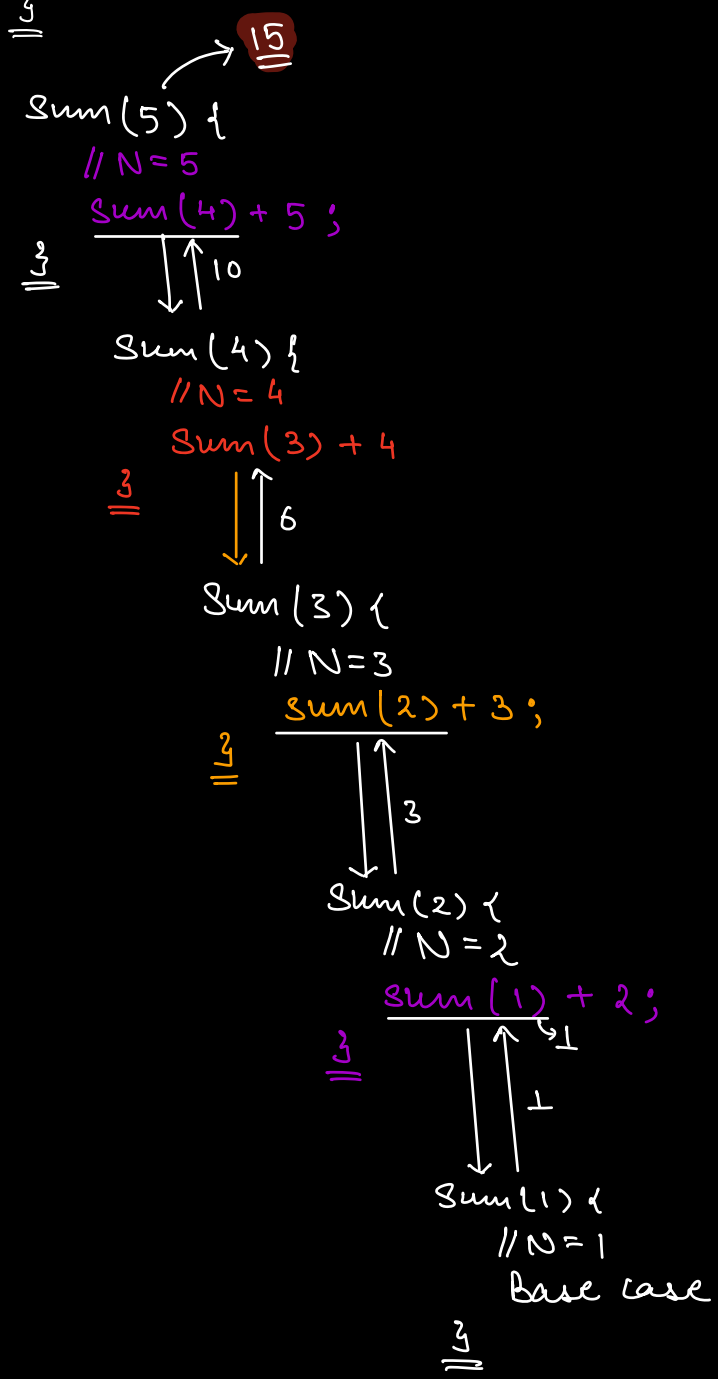
3. Exit cond^

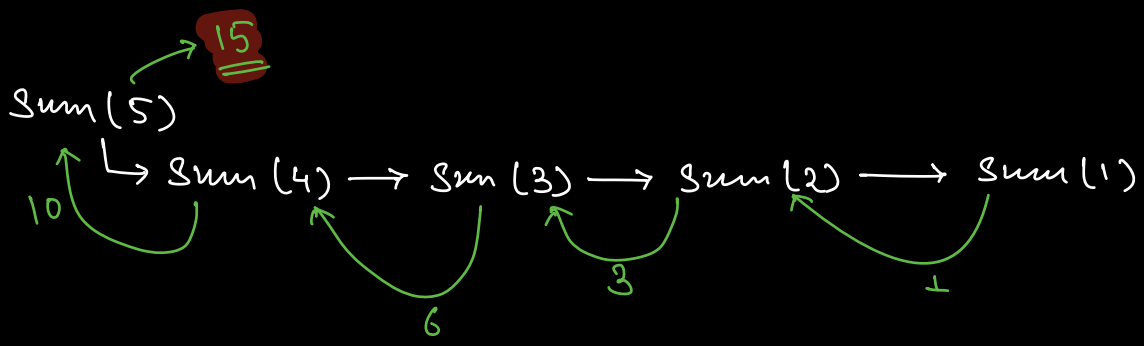
× fib(1) = fib(0) + fib(-1)

× fib(0) = fib(-1) + fib(-2)

✓ fib(2) = fib(1) + fib(0)

```
int Sum(int N) {
    if(N==1)
        return 1;
    // Main logic
    return Sum(N-1) + N;
}
```



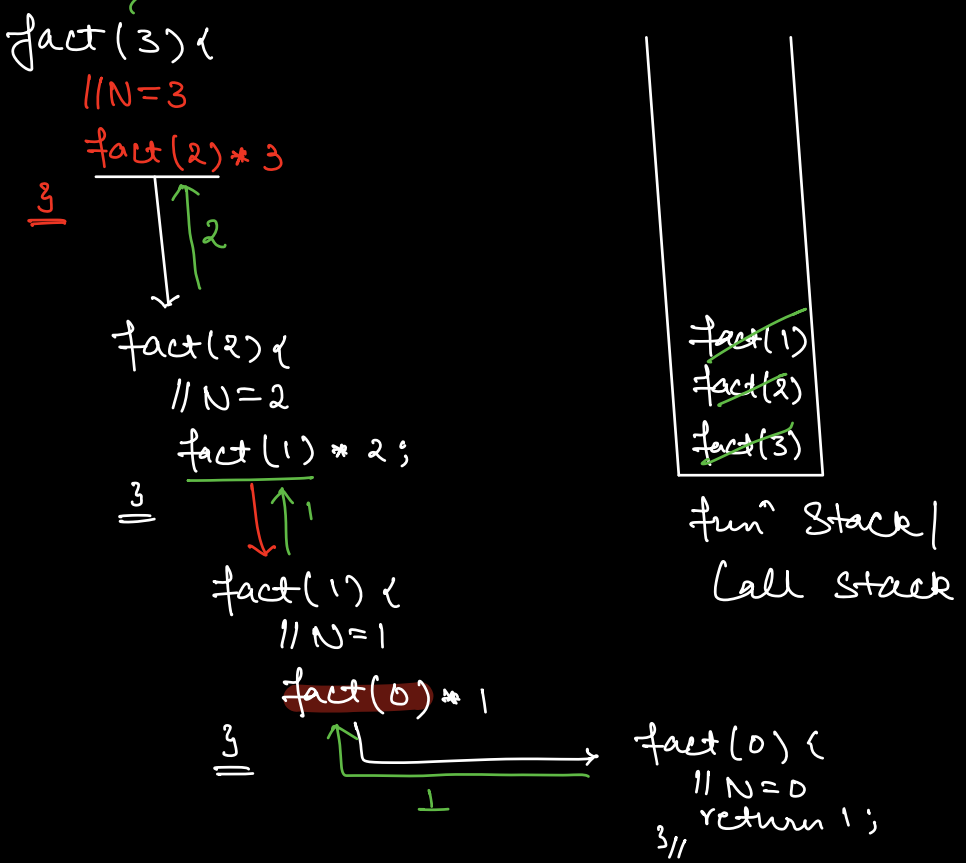


⇒ LIFO (Last In, First Out)

↳ Stack.

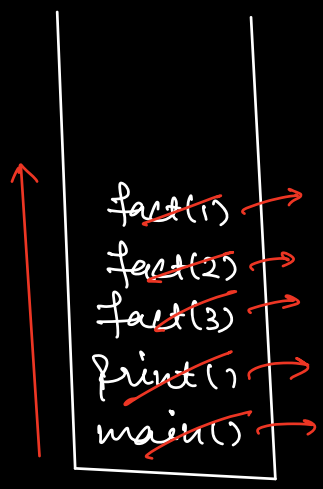
⇒

```
int fact(int N){
    if(N==0)
        return 1;
    return fact(N-1) * N;
}
```



⇒

```
main() {
    print(fact(3))
}
```



Call stack | Function stack

Q. Should we count the fun^ calls getting stored in the call stack in Space Complexity ?

⇒ YES.

When this call stack overflows ⇒ StackOverflow exception.

Q: Given a number N, print all the numbers from 1 to N in increasing using recursion.

N=5 ⇒ 1, 2, 3, 4, 5

```
void printInc (int N) {
    if (N == 0) {
        return;
    }
    printInc (N-1);
    print (N);
}
```

1. Assumption

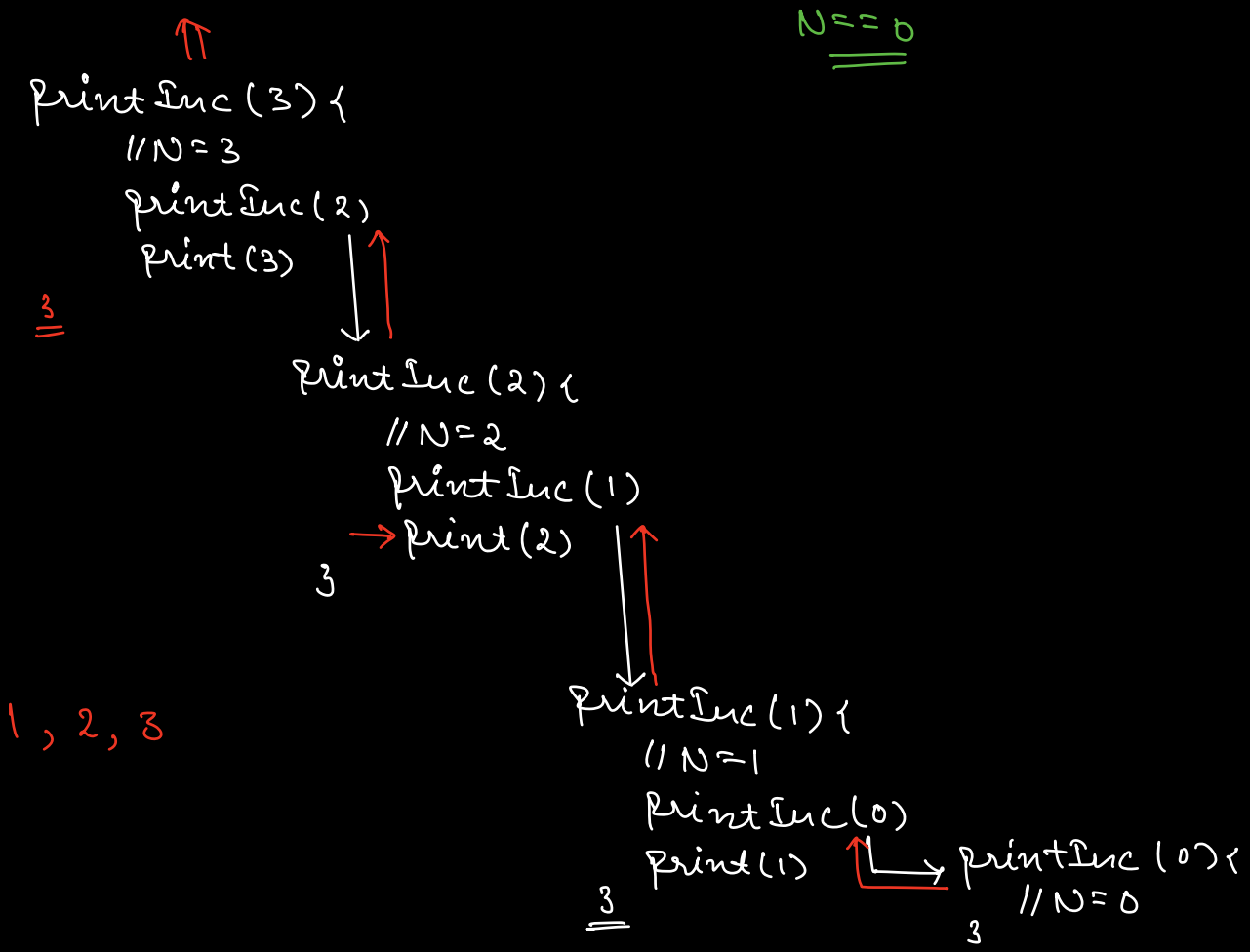
printInc(N) → prints all the no's from 1 to N in increasing order.

2. Main logic

printInc(N-1)
print(N)

3. Base Condition

N == 0



1, 2, 3

Q: Given a number N, print all the numbers from 1 to N in decreasing using recursion.

N ⇒ N, N-1, ---- 3, 2, 1

recursion.

```
void printDec(int N) {
    if(N==0) return;
    print(N)
    printDec(N-1)
}
```