Recursion-1

Recursion
- Linked list
- Binary Trees / BST
- Heaps.
- Tries / Graph.
- segment Tree's
- Dynamic Prog.
- Backtracking

Observation :-
- Size keeps on decreasing.
- All of the dolls were similar except the size.
- After last / smallest doll, we can't break further.

Recursion :- function calling itself.

↳ It is technique of solving a problem using smaller instance of same problem.

subproblem.

Ex $Sum(N) = 1 + 2 + 3 + 4 + \cdots + (N-1) + N.$

$Sum(N-1)$

Sub-problem.

Steps :-

1) Assume that your recursive fun^ will work.
2) Main logic :
   Solve the bigger problem using subproblem.
3) Base / Exit condition.
   ↳ Decide when recursion should stop.

⇒ Sum(N) = 1 + 2 + 3 + - - - (N-1) + N.

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

Base Cond^n

(I) Assumption :
Sum(N) will return sum of 1st N natural numbers.

(II) Main logic
Sum(N) = Sum(N-1) + N.

(III) Base condition :-
N==1 : sum = 1.

Ex Factorial of N.

$N! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot (N-1) \cdot N.$

$(N-1)!$

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

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

① Assumption :-
fact(N) will return N!

② Main logic
fact(N) = fact(N-1) * N.

③ base condition.

$1! = 1$ ✓

* $0! = 1$ → ✓

fact(4) = fact(0) * 1

1

Eg :- Fibonacci Numbers.

N :	0	1	2	3	4	5	6	7	8
	1	1	2	3	5	8	13	21	34

* Golden Ratio

5th fibonacci : 4th fib. + 3rd fib.

⇓

* fib(N) = fib(N-1) + fib(N-2) ✓

```
int fib(N) {
if(N==0 || N==1)
    return 1;

// Main logic
return fib(N-1) + fib(N-2);
}
```

(I) Assume that fib(N) will return Nth fib NO.

(II) Main logic.

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

(III) Base cond^n.

fib(2) = fib(1) + fib(0) ✓ ✓

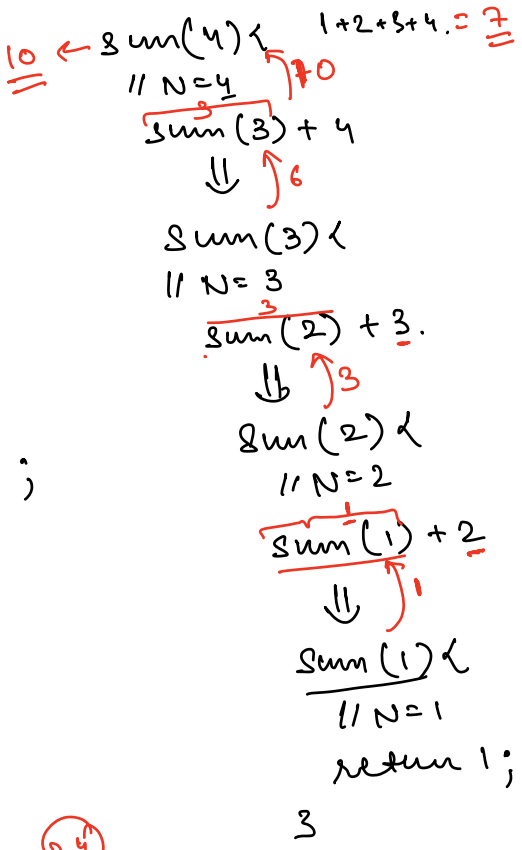
fib(1) = fib(0) + fib(-1)

fib(0) = fib(-1) + fib(-2)

⇒ Working of Recursion:

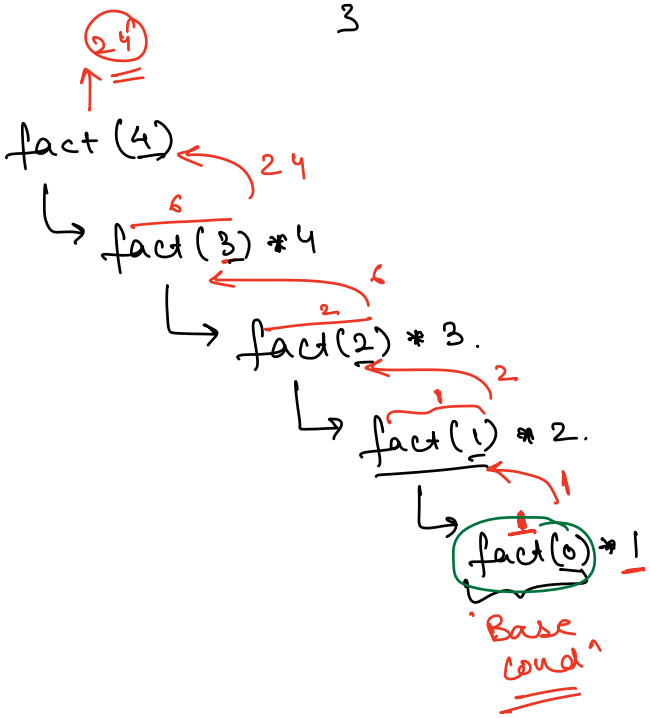
1)

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

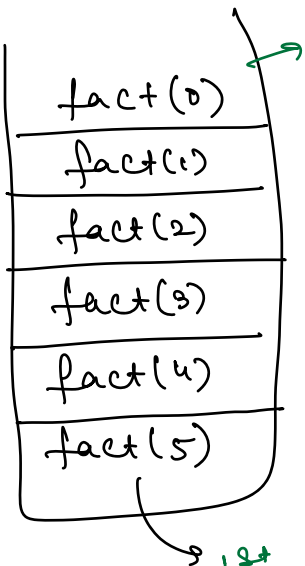


2)

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



⇒ fact(5) → fact(4) → fact(3) → fact(2) → fact(1) → fact(0)



Call Stack.

LIFO

↳ Last In First Out.

⇒ Stack.

⇒ StackOverflowException.

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

output :- 6

Call / Function Stack.

HW :- Draw fun stack for fibonacci fun.

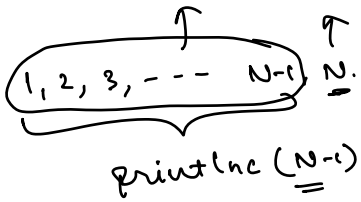
Q. Should we consider fun stack memory in our S.C. ?

⇒ YES.

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

```
void printInc(int N){
    if(N==1){
        print(1); return;
    }
    // Main logic
    • printInc(N-1); ⇒ print 1 to N-1.
    • print(N)
}
```

Base condition



Output :- 1, 2, 3.

```
printInc(3){
    // N=3
    printInc(2)
    print(3)
}

printInc(2){
    // N=2
    printInc(1)
    print(2)
}

printInc(1){
    // N=1
    print(1)
}
```

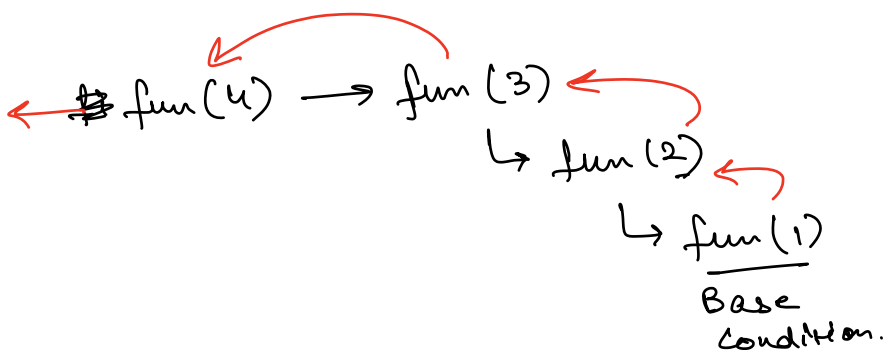
Print in Descending order.

Print In Descending order.

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

1, 2, 3 - - - N-1, N.

N, N-1, N-2 - - -



4, 3, 2, 1

Q. Given a String, write a recursive fun to check if the given string Palindrome.

→ madam

→ malayalam

→ Check for the first and last character, if they are same then check for the string.

```
boolean isPalindrome(str, s, e) {
    if(s >= e) {
        return true;
    }
    // Main logic.
    if(str[s] == str[e]) {
        → return isPalindrome(str, s+1, e-1);
    }
    return false;
}
```

Assumption :- isPalindrom(str,s,e) will return true if str is a palindrome

→ 'm a d a m'
  0 1 2 3 4
  s,e

s=1 e=3

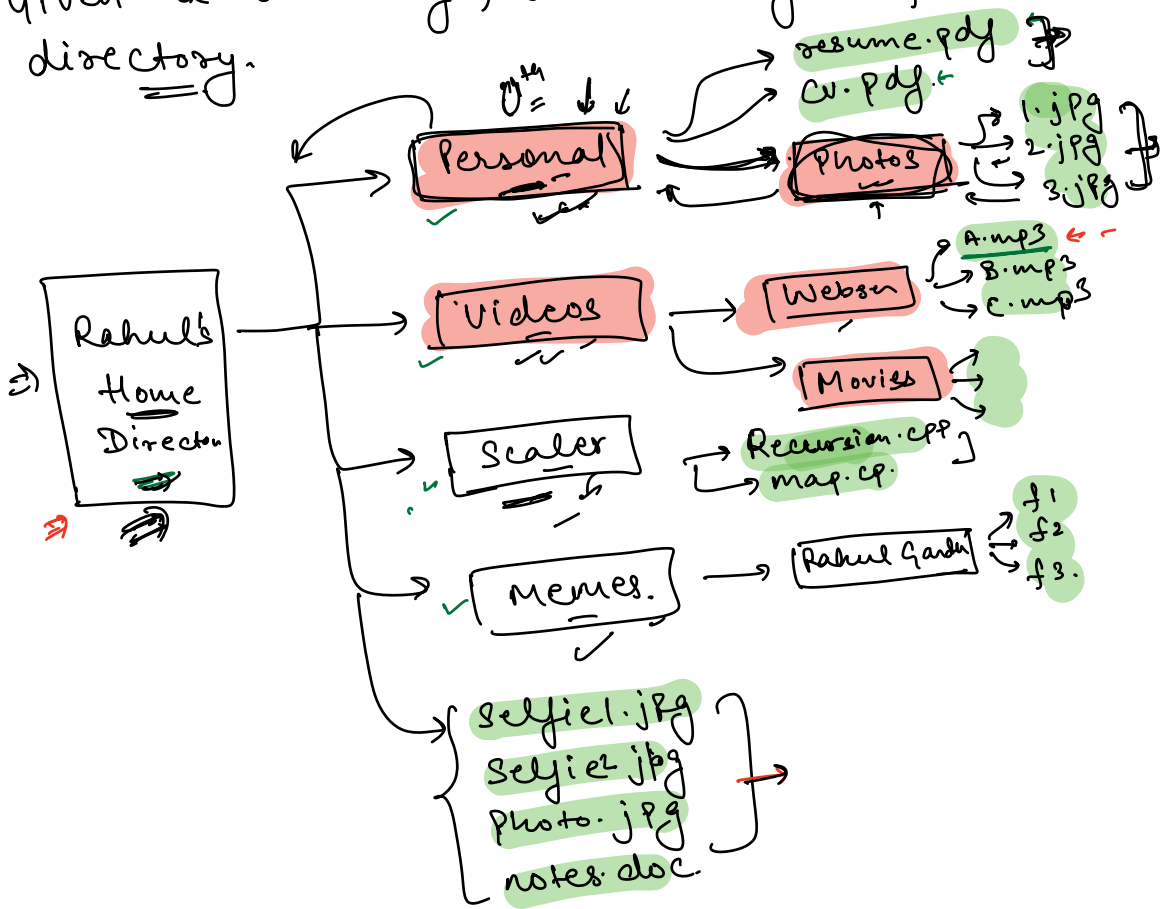
isPal(str, 0, 4) → T
isPal(str, 1, 3) → T
isPal(str, 2, 2) → T
isPal(str, 3, 1)
s e.

True

Q

Facebook/ Meta. Google.

Given a directory, search a given file in the directory.



⇒ Given 2 functions :-

List<String> getAllDirectories (directoryName)

List<String> getAllFiles (directoryName)

Approach :-

1) Check all the files in the Home directory first, if found return true.

2) Check in all the sub directories one by one.

```
boolean searchFile(dir, fileName) {
```

- Assumption: searchFile(D, F) will return TRUE if 'F' is present in 'D'.
- Main logic

```
List<String> files = getAllFiles(dir);
for(i=0; i<files.size(); i++){
    if(files[i] == fileName){
        return true;
    }
}
List<String> directories = getAllDirectories(dir);
for(i=0; i<directories.size(); i++){
    if(searchFile(directories[i], fileName))
        return true;
}
return false;
}
```

Base Condition