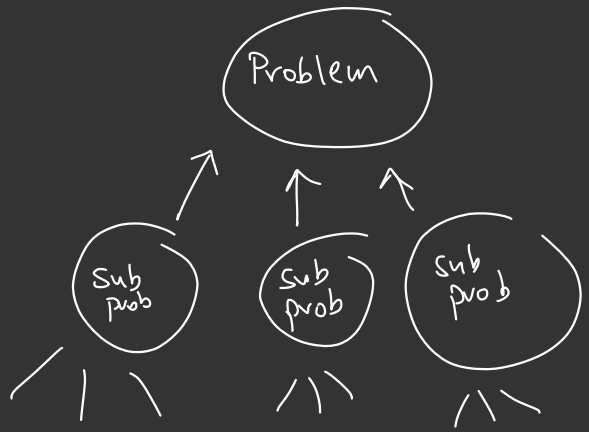
Recursion · 1

- useful & powerful technique in CS

[Contest 2/3
↓
New Date TBA
Slack/email]



Problems → Iterative (✓)
→ Recursive (✓)

Many other topics -

- Merge Sort / QuickSort
- Binary Trees
- DP
- Backtracking
- Graph
- Segment Trees

} Lot of Also are easy to implement using recursion

walk 1 step

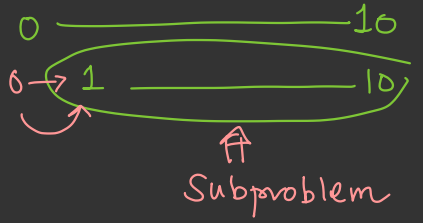

```
goToHome (int x, int Home) {
```

Iterative soln

```
    for (        , x < Home ; x++) {
        print(x)
    }
```

```
}
```

goToHome(0, 10)

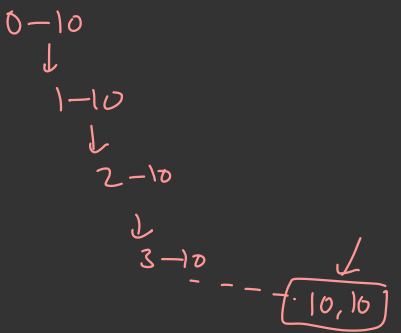



```
goToHome ( x , Home ){
    if(x == Home){
        return
    }
    x = x + 1
    goToHome (x , Home)
}
```

Base case

Rec case

- Take one step
- Rec call for rest of the work



3 Step Rule to write Rec Code:

PMI

Principle of Mathematical Induction

↓

School Days.

(1) Figure out smallest instance of that problem

Base case.

$f(0) = 1$

$0! = 1$

(2) Assume a subproblem can be solved

Factorial

$f(n)$ n

$f(K)$ ✓ is known for $K < n$

is True

(3) Using above assumption, write down $f(n)$ in terms of smaller subproblem(s)

Rec case

$$f(n) = f(n-1) * n$$

$n!$ $\quad$ $(n-1)! * n$

$n = 5$

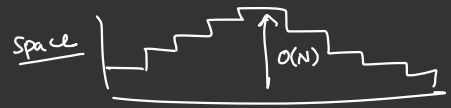
$f(5)$

$5! = 1 \times 2 \times 3 \times 4 \times 5$

$= 4! \, 5$

$= f(n-1) * n$

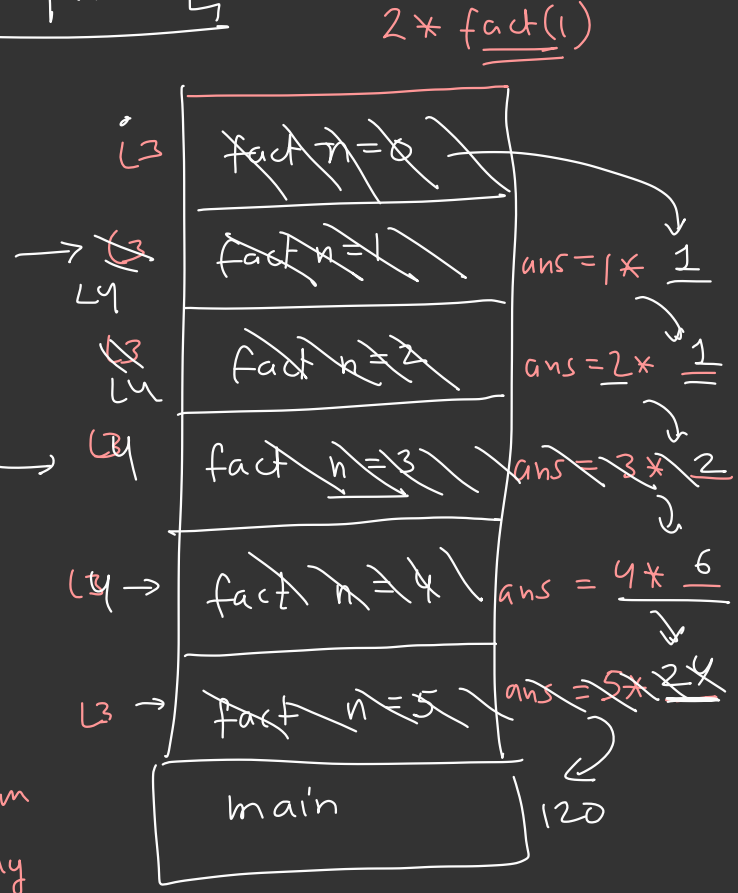
Prefer Iteration over Recursion for most problems

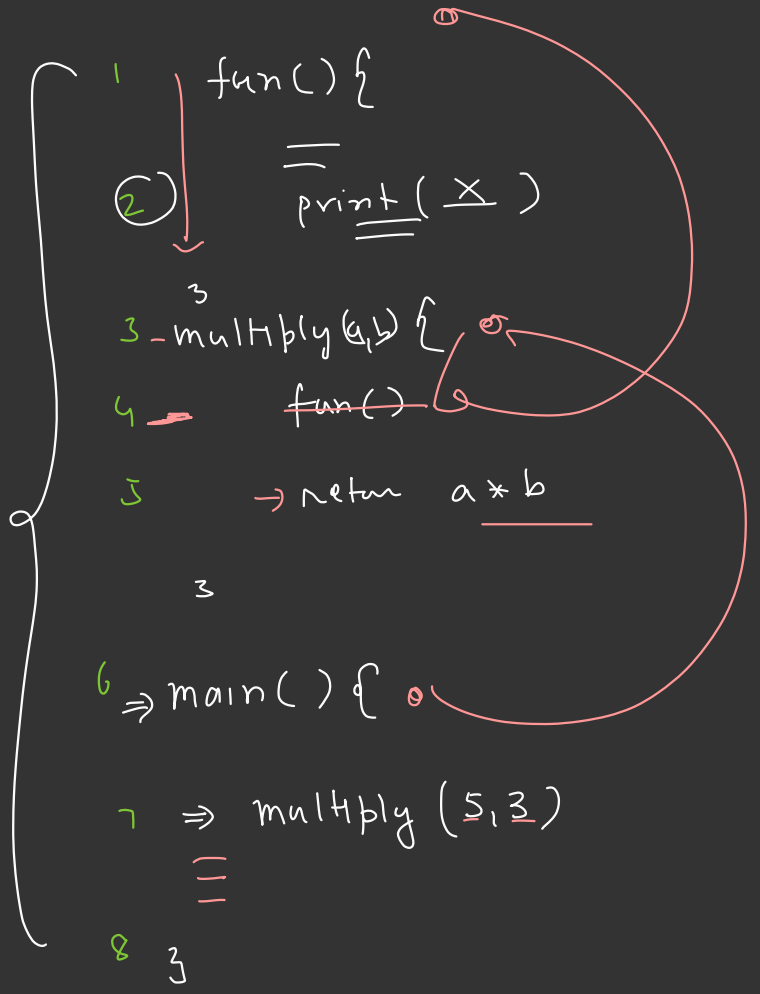



```
int fact(int n){
    // Base Case
L1  if(n==0){
L2      return 1
    }
    // Rec Case
Paused → L3 int ans = n * fact(n-1)
L4      return ans
}

main(){
    fact(5)
}
```

Solve a sub problem recursively



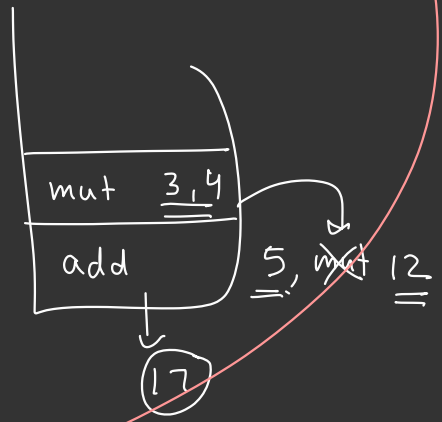

1 fun() {
2 print(x)
3
3 multiply(a,b) {
4 fun()
5 → return a*b
3
6 ⇒ main() {
7 ⇒ multiply(5,3)
8 }


Time = O(N)
Space = O(N)
LIFO
fun
mut
main
fill
clearing
Stack Memory

```
add(a, b)
    return a+b

mut(a, b)
    return a*b

add(5, mut(3, 4))
```



Q Finds sum of Numbers from 1 to N. (without formula)

```
int Sum (N) {
    if(N==1)
        return 1

    ans = N + Sum(N-1)
    return ans
}
```

Incorrect BC. → ↓ stack overflow error

$1+2+3+\_\_+N-1+N$

$=$

$Sum(N-1)$

N = 4

1 + 2 + 3 + 4
= 10

sum(4) = 4 + sum(3) 6 = 10

└ = 3 + sum(2) 3 = 6

└ 2 + sum(1) 1 = 3

└ 1

10.00 PM

Q

```
void inc (int n){
L0    if(n==0)
          return ← must
L1    inc(n-1)
L2    Print(n)
      return ← optional
}
```

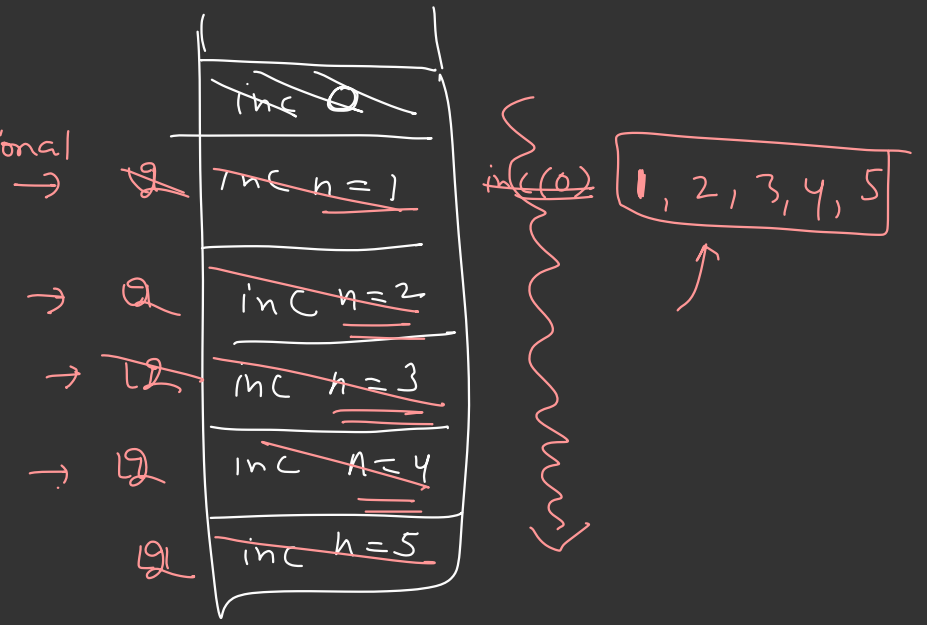
n = 5

1, 2, 3, 4, 5

n

inc 5 → 1, 2, 3, 4, 5

inc 4 → 1, 2, 3, 4



```
void dec (int n){
    if(n==0)
        return ← stop the rec
    print(n)
    dec(n-1)
}
```

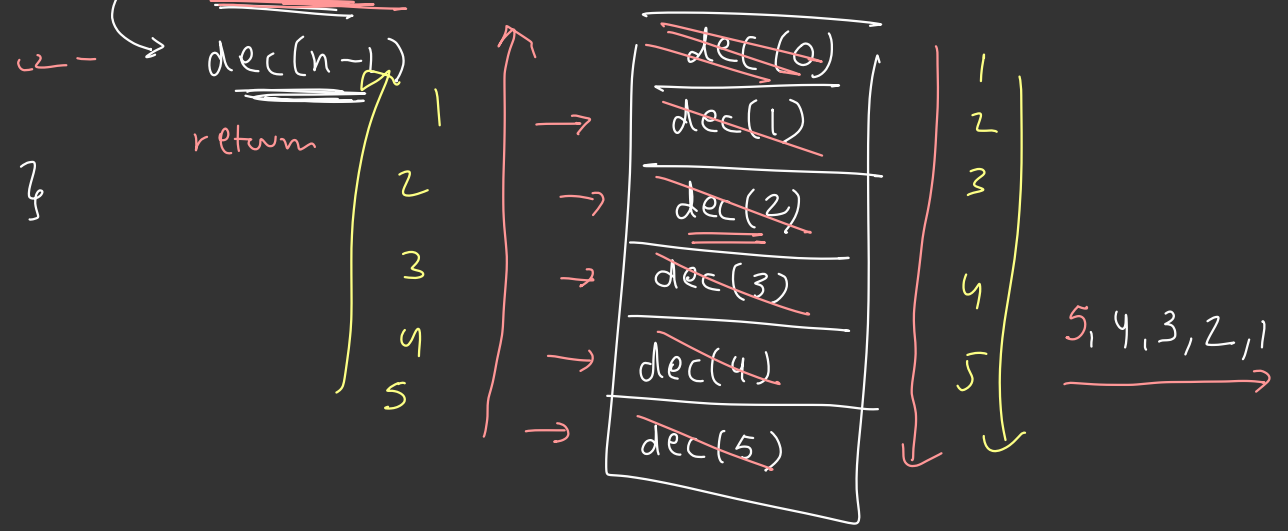
dec(4)

dec 5 → 5, 4, 3, 2, 1

dec 4 → 4, 3, 2, 1

dec(n) = n, dec(n-1)

Base Case



→ Code Before ← Towards Base Case

Rec Call

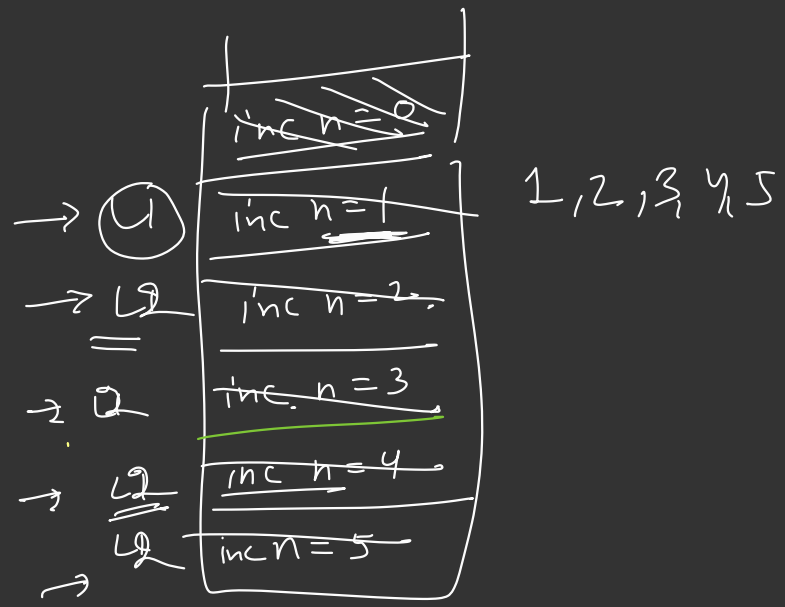
→ Code after → Coming after the hitting the base case

10:30
Break

$inc(n) = [1, 2, 3, \ldots, n-2, n-1], n$

$= inc(n-1), print(n)$

```
inc(n) {
    if(n==0)
        return
    L1 → inc(n-1)
    L2 → print(n).
    return.
}
```



Problem → 1 subproblem

$5! \rightarrow 5 * 4! = 120$

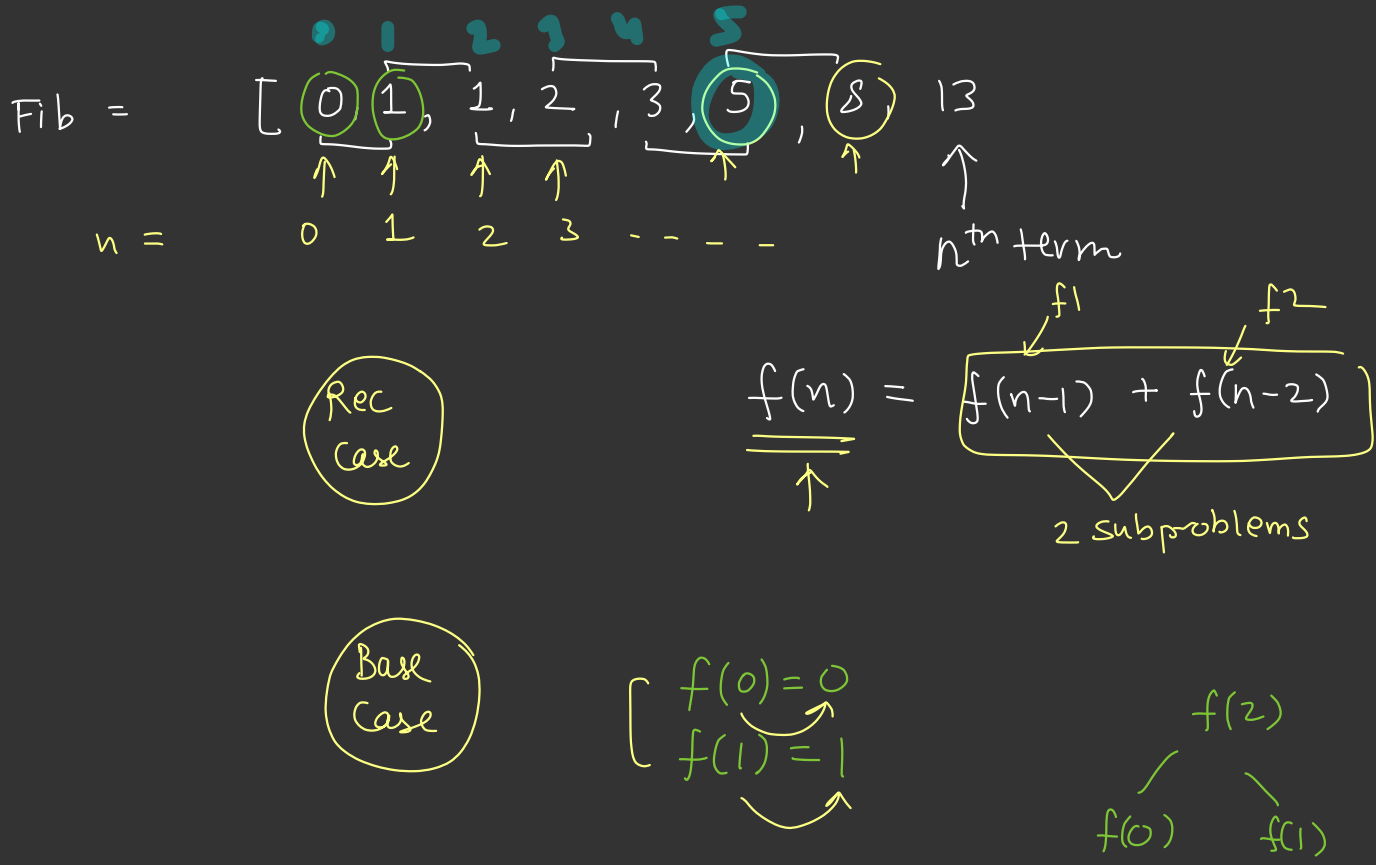
$4! = 4 * 3! = 24$

$3! = 3 * 2! = 6$

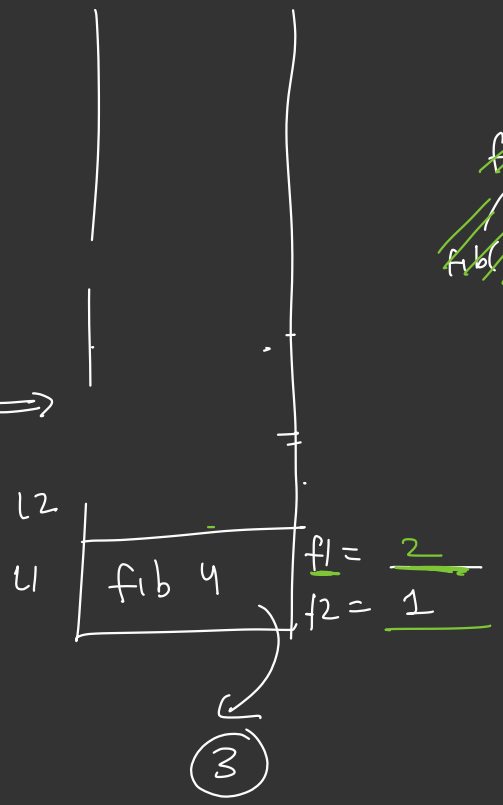
$2! = 2 * 1! = 2$

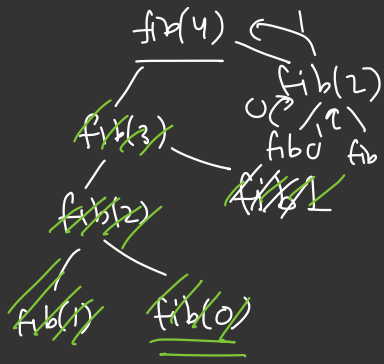
$1! = 1 * 0! = 1$

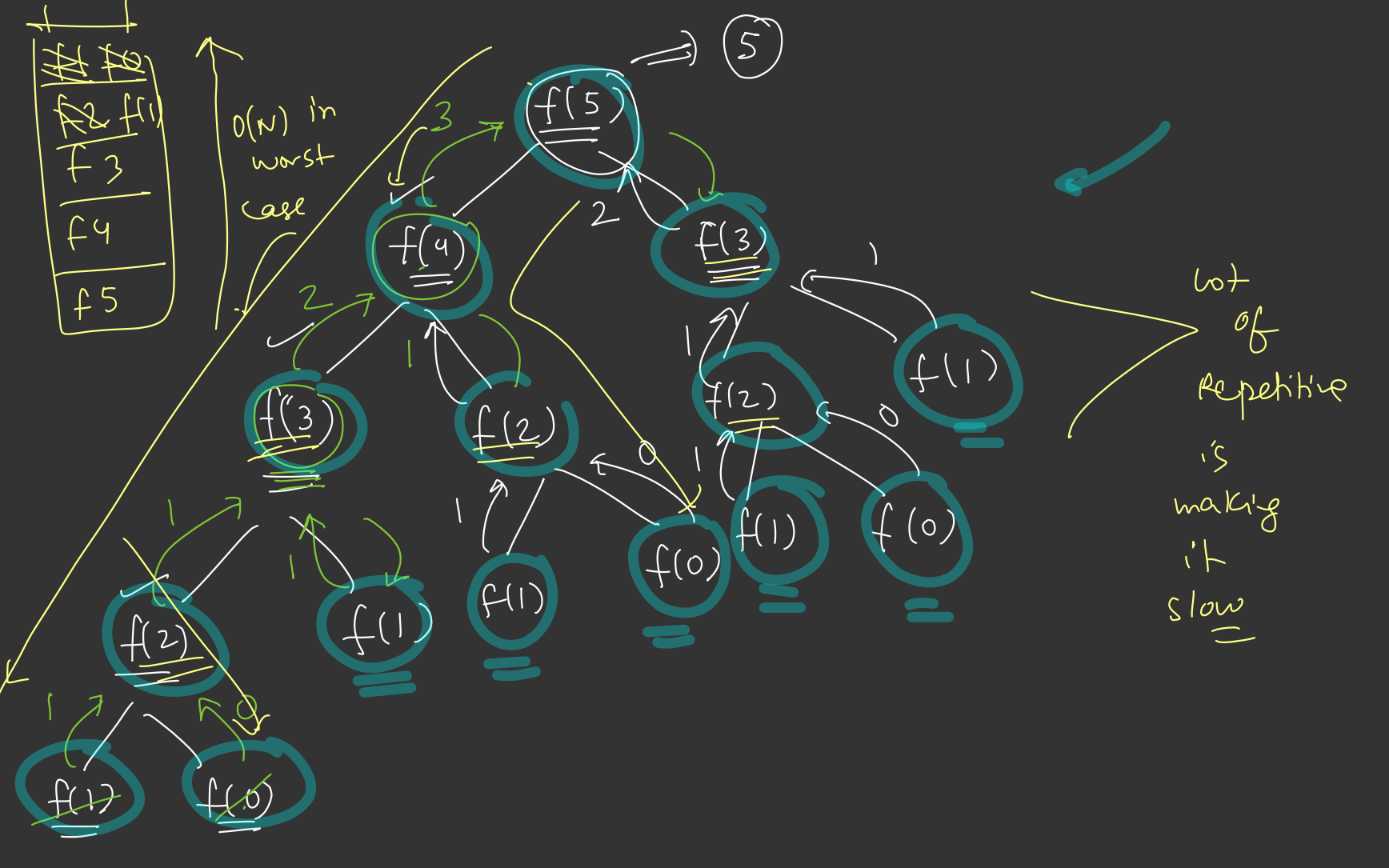
Problem → 2 or more subproblems

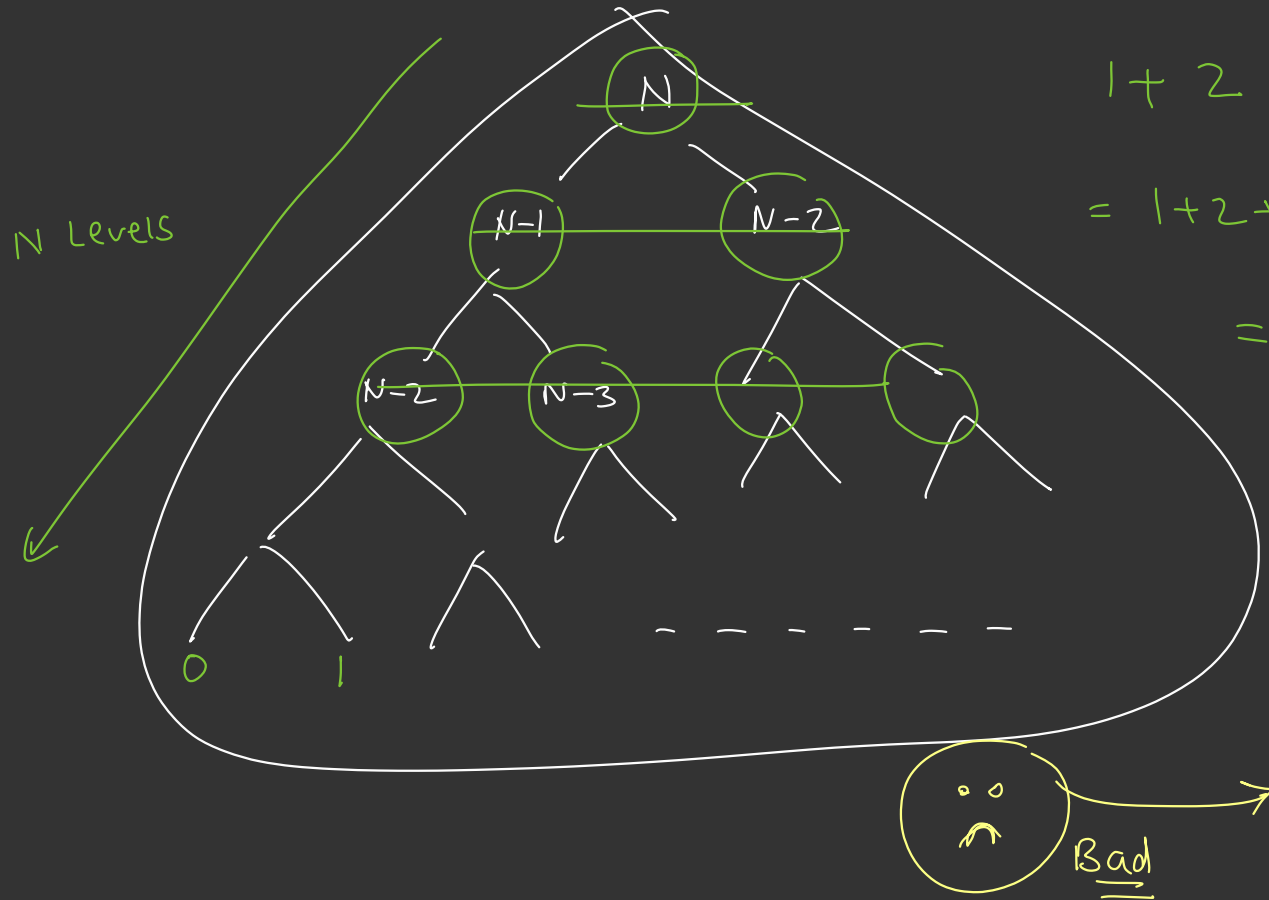

Fib =
[0, 1, 1, 2, 3, 5, 8, 13
0 1 2 3 4 5
n =
0 1 2 3 - - - -
nth term
Rec Case
f(n) = f(n-1) + f(n-2)
f1
f2
2 subproblems
Base Case
f(0) = 0
f(1) = 1
f(2)
f(0)
f(1)

```
int fib (int n){
    // Base Case
    if(n==0 || n==1){
        return n
    }
    // Rec Case
    L1 : f1 = fib(n-1)
    L2 : f2 = fib(n-2)
    L3 : return f1+f2
}
```






f5
f4
f3
O(N) in worst case
f(5)
5
f(4)
f(3)
f(3)
f(2)
f(2)
f(1)
f(1)
f(0)
f(1)
f(0)
f(2)
f(1)
f(1)
f(0)
lot of Repetitive is making it slow



$$1 + 2 + 4 + \dots + 2^{n-1}$$

$$= 1 + 2 + 2^2 + 2^3 + \dots + 2^{n-1}$$

$$= \frac{a(r^n - 1)}{r - 1}$$

$$= \frac{2^n - 1}{2 - 1}$$

$$= O(2^n)$$

Bad

0, 1, 1, 2, 3, 5, 8

for (i=2 → n) {

⇒ a[i] = a[i-1] + a[i-2]

}

O(N)

Time = Total Fn Calls * Time Spent in each call

Total Fn Calls: $O(2^n)$

Time Spent in each call: $O(1)$

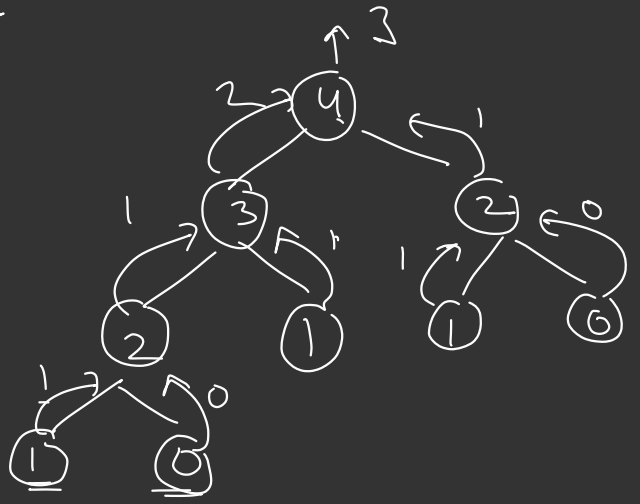
$= O(2^n)$

Space = Max Depth of the Tree * Space in each call

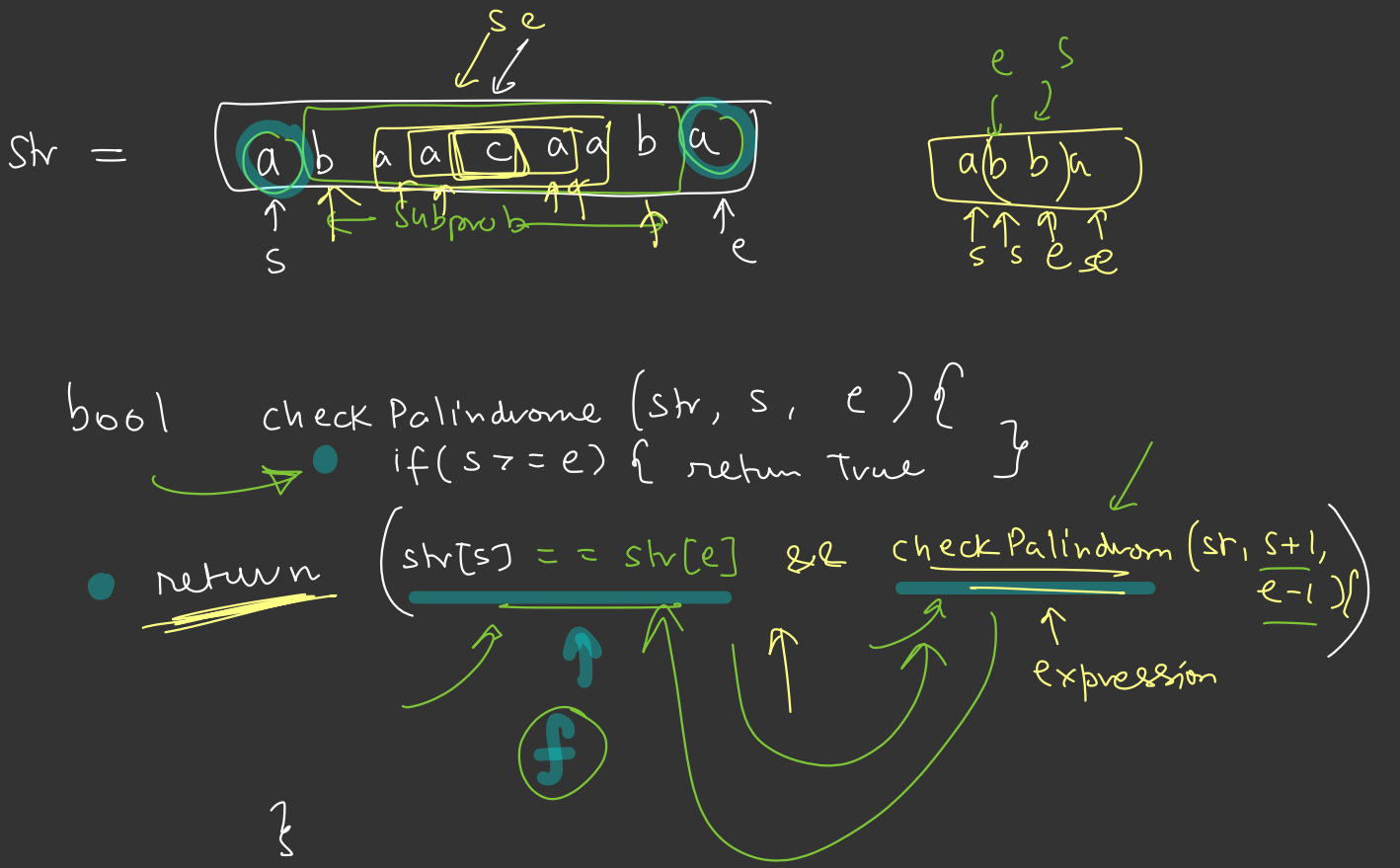
= O(N) O(1)

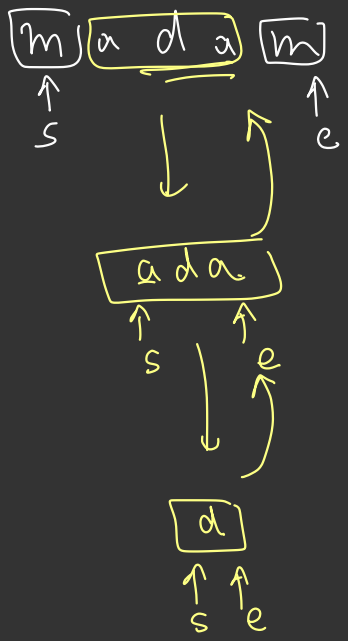
= O(N)

- Code using 3 Rule
- Tree Diagram (better than call stack)



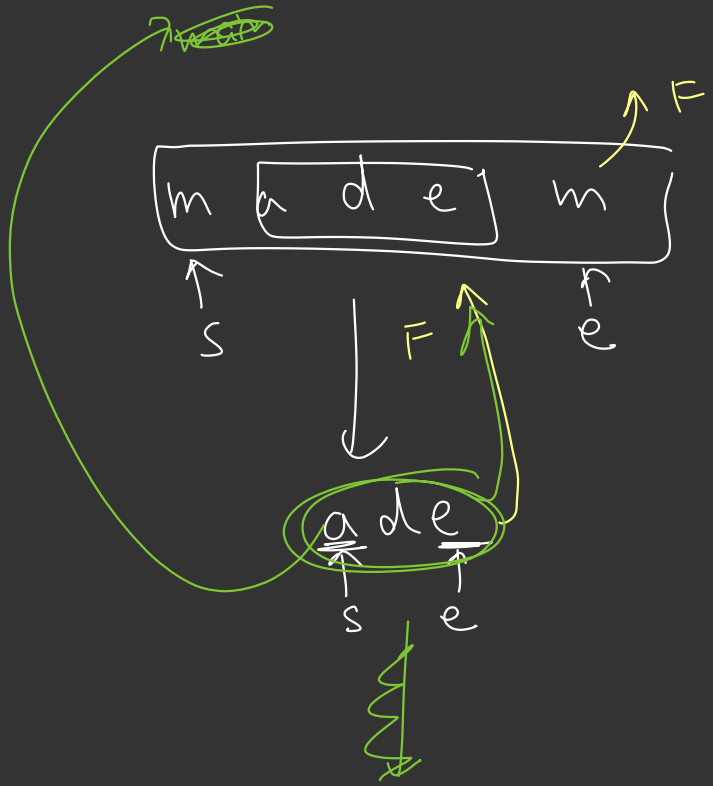
Every loop
↓
can be
written
Recursively


Str =
a b a a c a a b a
s e
Subproblem
e s
a b b a
s s e e
bool checkPalindrome (str, s, e) {
if(s >= e) { return True }
return (str[s] == str[e] && checkPalindrom (str, s+1, e-1))
expression
}



T && T = T

T && T = T



$T \propto e \quad \underline{F} = (F)$

$F \propto \underline{x}$