

Q

AJ Number  $\rightarrow$  Maximum no. of AJ no. present in string  $\rightarrow$  return count

✓✓  
0, 1  $\rightarrow$  AJ  $\times$

[ No. ]  $\rightarrow$  AJ  $\checkmark$

- 1. 0 and 1 are not AJ numbers.
  - 2. 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 are AJ numbers.
  - Any number not divisible by the numbers in point 2 (Given above) are also AJ numbers.
- $x \% [No.] = 0$

let  
 $x \% [No.] = 0 \rightarrow$  AJ  $\checkmark$   
 Not divisible

given no.

In 4991, both 499 and 991 are AJ numbers but you can choose either 499 or 991, not both.

Common substring.

## Test Cases

for finding maximum

case-1

1 2 3 4 5

let suppose  
this is an  
AJ

Cnt = 1

case-2

let

1 2 3 4 5  
check 5 is an  
AJ no.

Cnt = 1

" AJ "

if i counted → then my task is to  
remove this no.

can i use this substring from

2 3 4

1 2 3 4 5

We are jus Assuming

NO

1 2 3 4

Cnt = 2

1 2 3

Cnt = 3

1 2

Not  
an AJ  
Cnt = 3

AJ ✓  
Cnt  
4

Cnt = 3

return 4

i.e. the maxi. no. of AJ no.

\* Actual example for understanding "AJ no."

Ex      81 615

1>  $0, 1 \rightarrow AJ \otimes =$       ~~Cnt++~~ ✓

2>  $[2, 3, 5, 7, 11, 13, 17, 19, 23, 29] \rightarrow AJ \checkmark$

3>  $(x \% [ ] \{ 1 = 0 \}) \rightarrow AJ \checkmark$       Cnt++ ✓

81      Not an AJ no.      AJ cnt = 2      (no change)  
in AJ Ent

$(8 \% 2 \{ = 0 \}) \rightarrow AJ \times$       false

5  $\rightarrow AJ \text{cnt} = \emptyset \ 1$

new no.      8161

remove 5       $x \% [\text{no.}]$       remainders  
6, 1  $\rightarrow$  are not AJ no.

remove 61 from 8161       $6 \% 2 \{ = 0 \} \rightarrow \text{Yes} \rightarrow AJ \checkmark$

$(81 \% 2 \{ = 0 \}) \rightarrow \text{True}$   
 $(81 \% 3 \{ = 0 \}) \rightarrow \text{false};$   
 $(81 \% 3 \{ = 0 \}) \rightarrow \text{break};$

Q. stronger string

$$n = 4$$

(a, b, c, d)

$$\text{ans} = \text{cabd}$$

$$n = 6$$

a, b, c, d, e, f

$$\text{ans} =$$

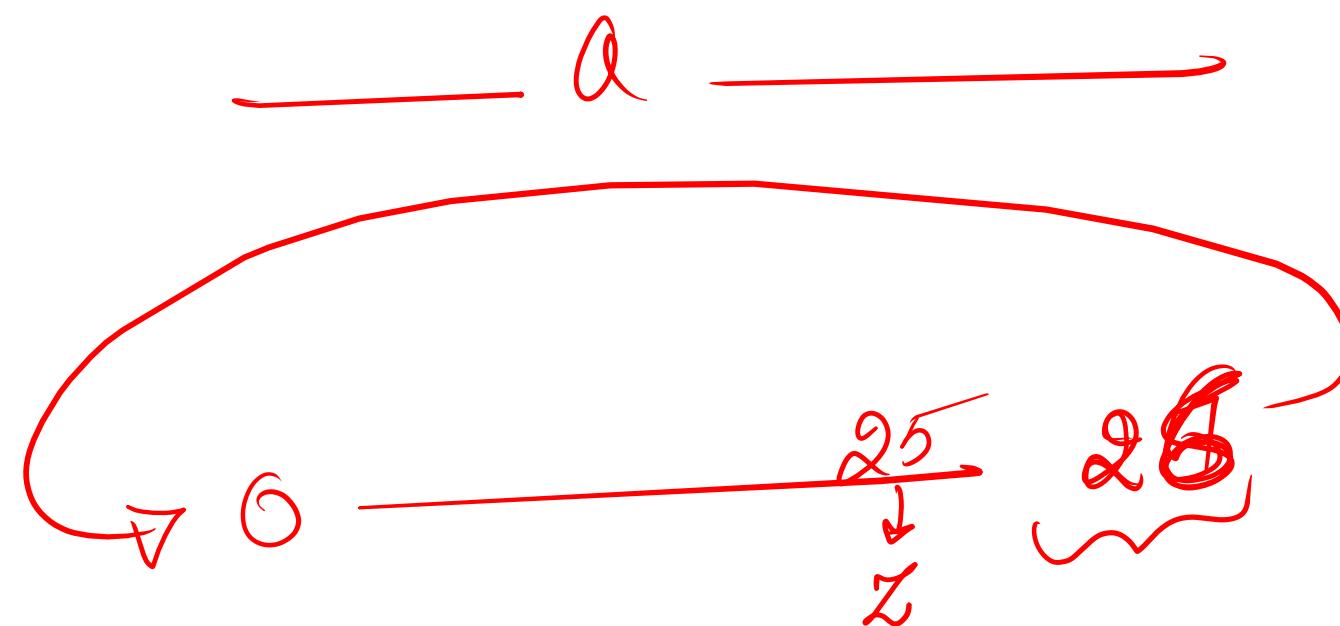
e c a b d f  
4 2 0 1 3 5  
  
even idx  
↓  
insert  
↓  
  
odd idx  
↓  
append the character

$n = 26$



return

$n = 27$



cyclic order  
use %

```

static String strangeString(int n) {
    //Write your code here
    StringBuilder ans = new StringBuilder("");
    ans.append('a');
    for(int i = 1 ; i<n ; i++){
        char temp = (char)('a' + (i%26));
        if(i % 2 == 0){ // even idx ---> add the char in the starting
            ans.insert(0,temp);
        }
        else{ // odd ---> add char at the ending
            ans.append(temp);
        }
    }
    return ans.toString();
}

```

$$\begin{array}{l} \text{n=6} \\ \text{n=0} \end{array}$$

$\underline{\text{ans}} = \underline{\text{a}}$

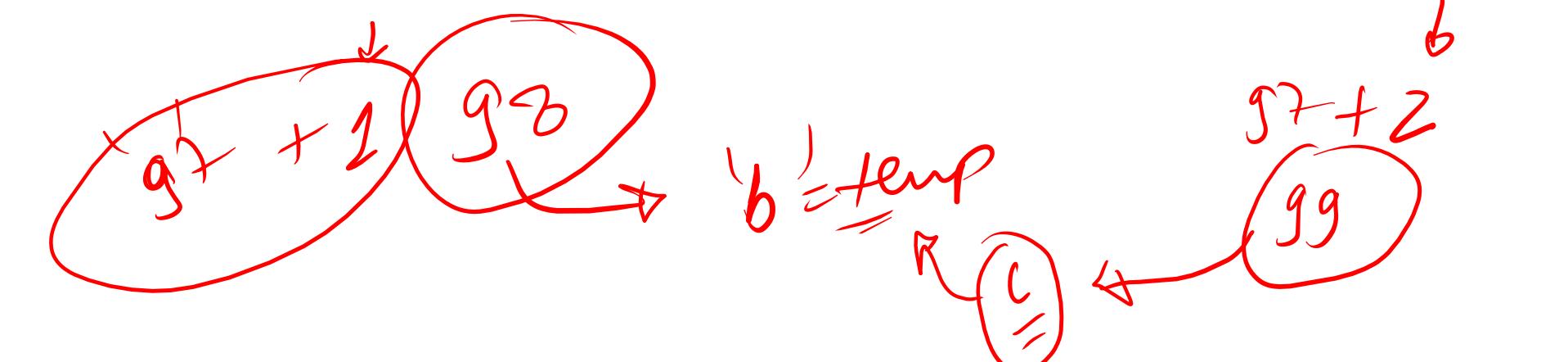
$\underline{\text{ans}} \geq \underline{\text{ab}}$

$$\text{temp} = (\text{char}) ('a' + (i \% 26))$$

1-1.26

$$\begin{array}{l} \text{n=1} \\ \downarrow \\ \text{odd} \end{array}$$

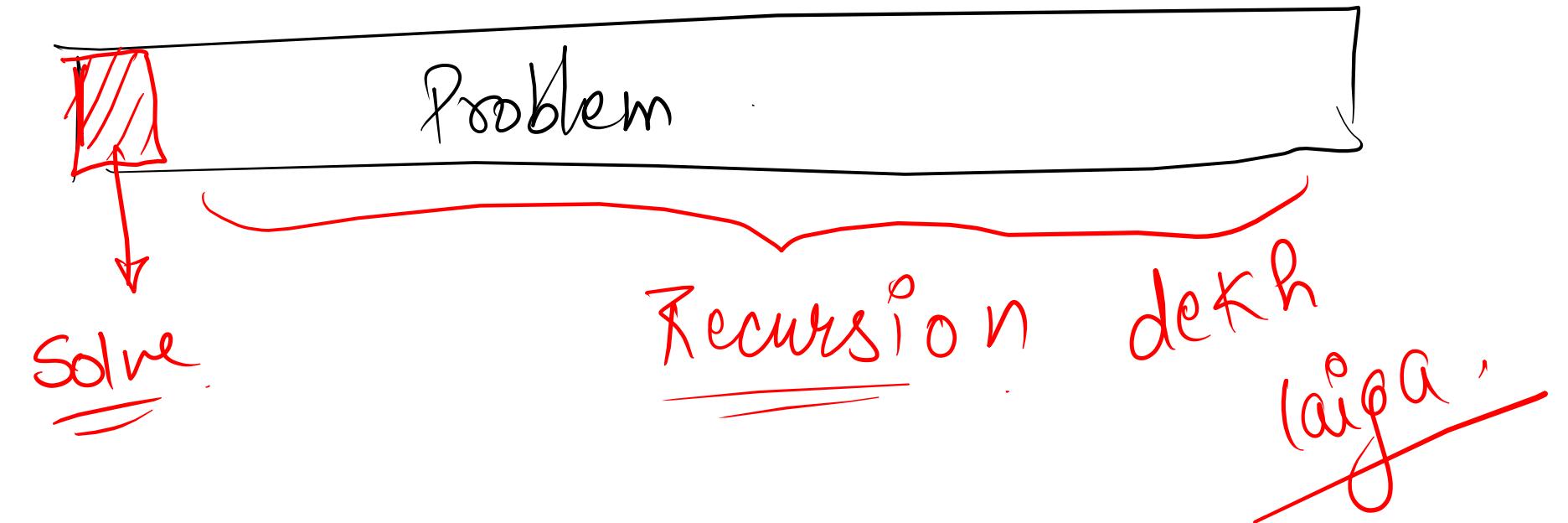
$\underline{\text{ans}} = \underline{\text{g}}$  c a b d f



## # Recursion

Defination: A function calling itself till the base case is not true :-

divide & conquer



\*

factorial

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

factorial (5)



5 \*

fact (4)

Recursion

(1)

factorial

$n=5$

~~f(2)  
f(1)  
f(0)~~

~~f(1)~~

~~f(2)~~

~~f(3)~~

~~f(4)~~

~~f(5)~~

~~call stack~~

$5 \times \underline{\text{fact}(4)}$

Recursion Tree

$4 \times \underline{\text{fact}(3)}$

$3 \times \underline{\text{fact}(2)}$

$2 \times \underline{\text{fact}(1)}$

$0 \times \underline{\text{fact}(-1)}$

$1 \times \underline{\text{fact}(-1)}$

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

if you didn't add  
base case.  
Then the  
function call  
itself at  $\infty$



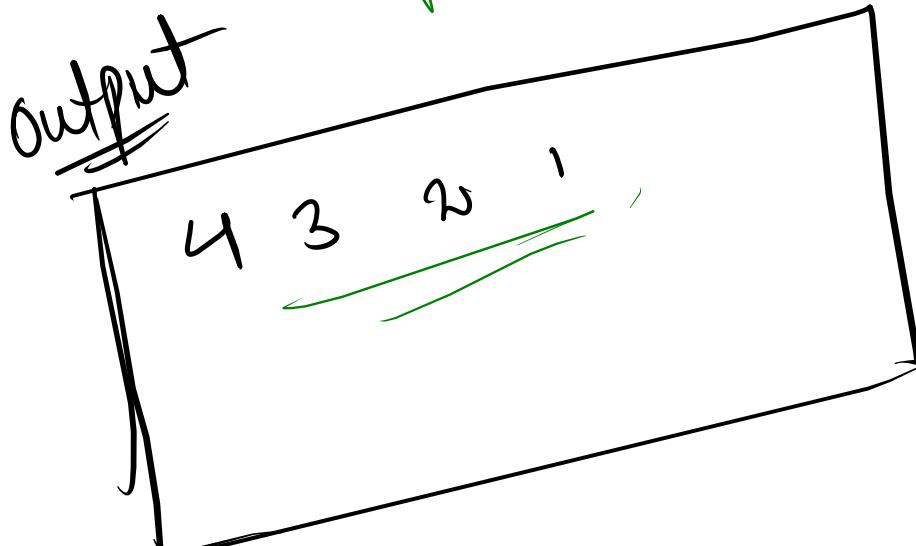
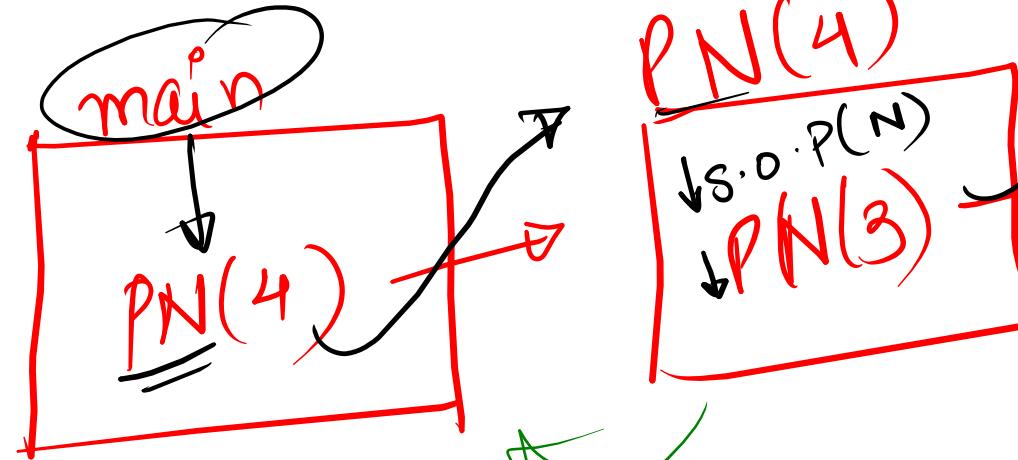
stack over flow

To avoid stack overflow  
i have to add base  
case.

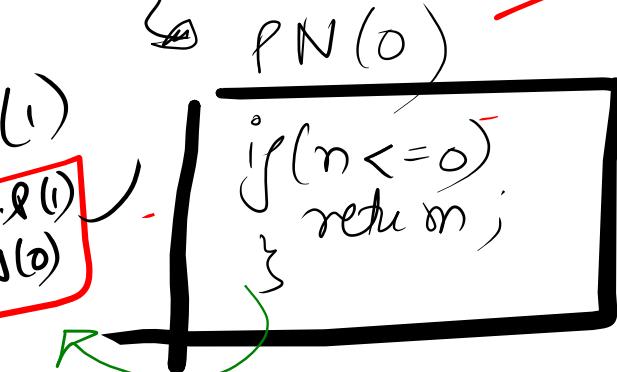
① Point no.  $[1, n]$

$$\underline{n = 4}$$

O/P  $\Rightarrow 4, 3, 2, 1$



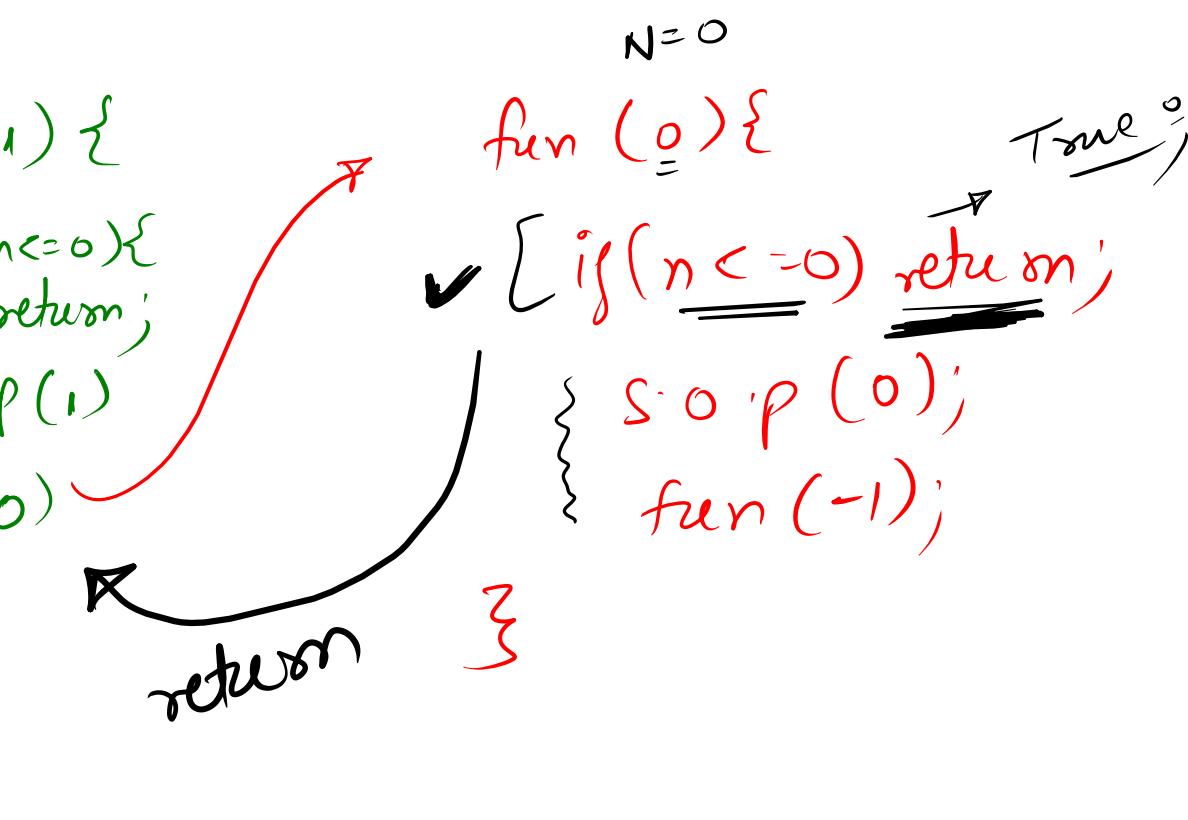
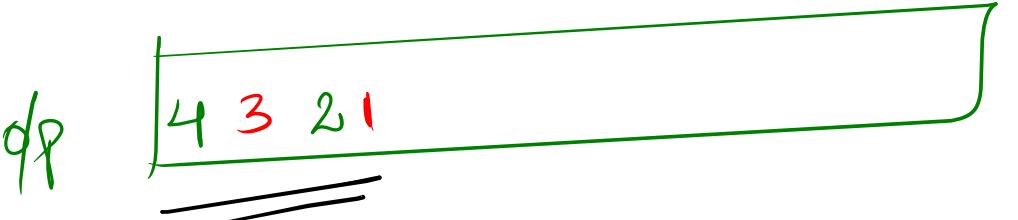
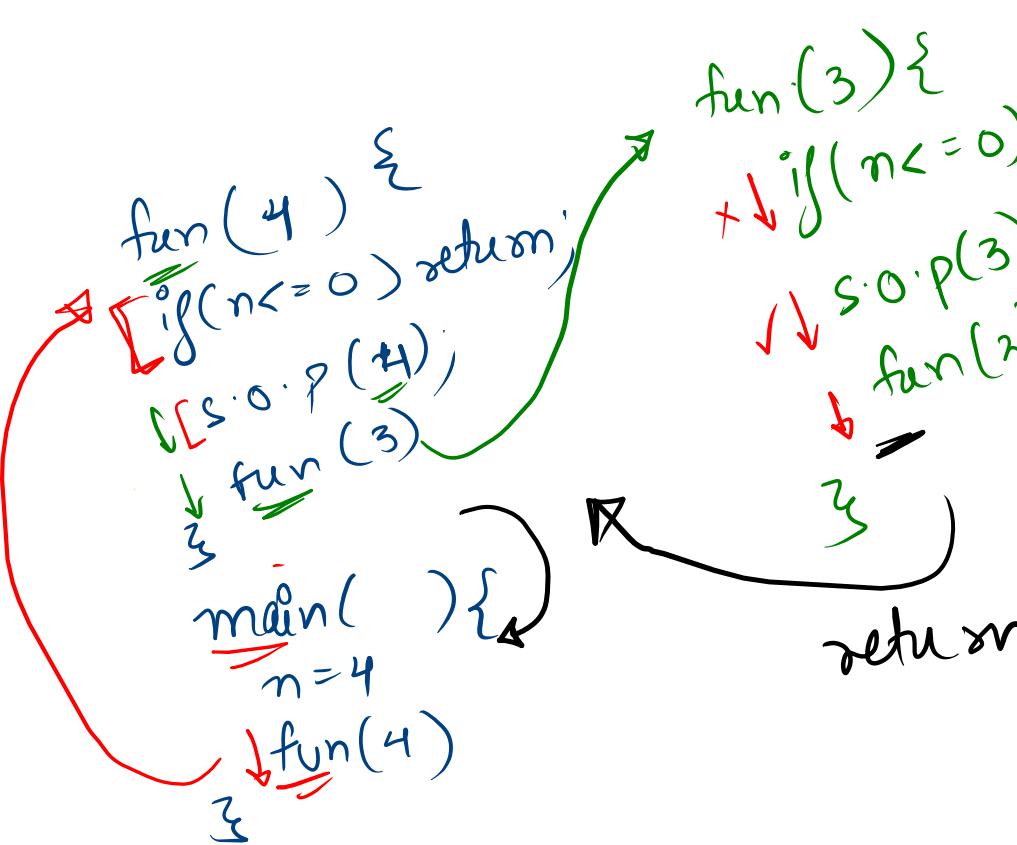
Base case

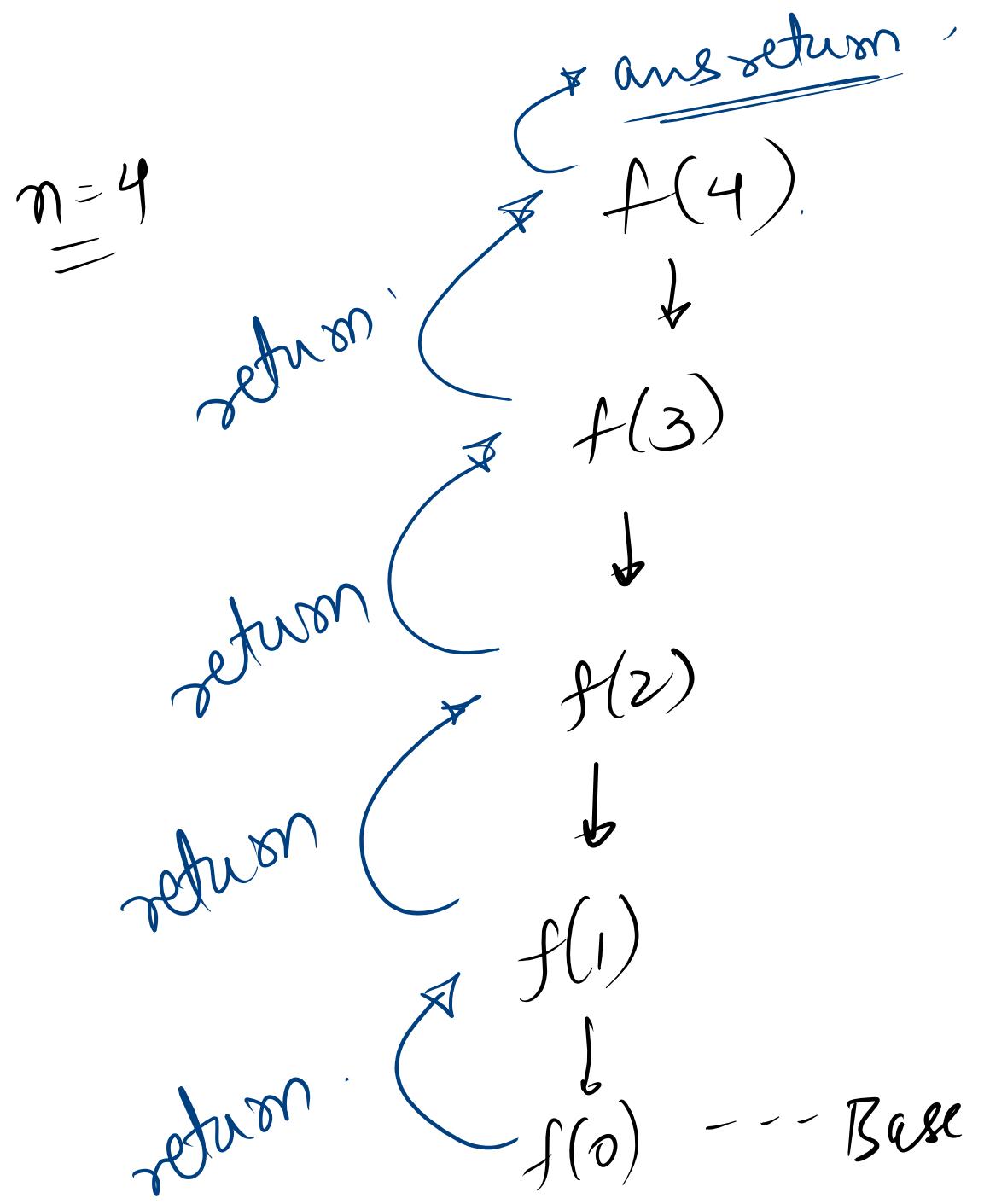


otherwise  
function  
calling

② Point No  $[1, N]$

$N = 4$





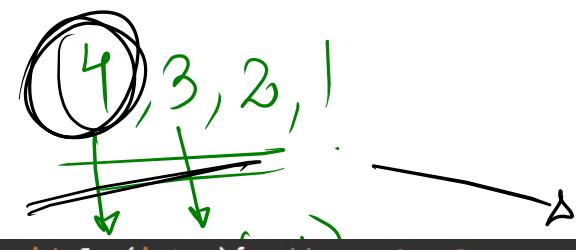
## Recursion Tree

function calling

$n \rightarrow \underline{\underline{(n+1)}}$

\* space complexity:  $O(N)$

The diagram illustrates a call stack with several function frames represented as horizontal black lines. A yellow diagonal line highlights the top frame, labeled `fun(0)`. Below it are `fun(1)`, `fun(2)`, `fun(3)`, and `fun(4)`, each with a green diagonal line. The bottom frame is `main()`. To the left, a green oval contains the text "function remove" with an arrow pointing to the `fun(0)` frame. To the right, a red arrow labeled "return" points upwards from the `fun(4)` frame towards the `fun(0)` frame. Green arrows on the right side indicate the flow of control between frames.

\* Point 

```

public static void fun(int n){ //n == 4 3
    if(n <= 0) return ;
    System.out.print(n+ " ");
    fun(n-1); // fun(3) fun(2)
}

```

$$\text{fun}(n) = n, \text{fun}(n-1)$$

Base case 

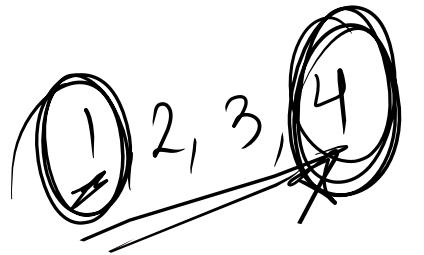
Point(4) ✓  
function call ✓  
(n-1) ↗

Point(3) ✓  
func.all ✓  
(n-1) ↗

Point(2) ✓  
fu. call(n-1) ↗

PN(1) ✓  
f call(n-1) ↗

P1

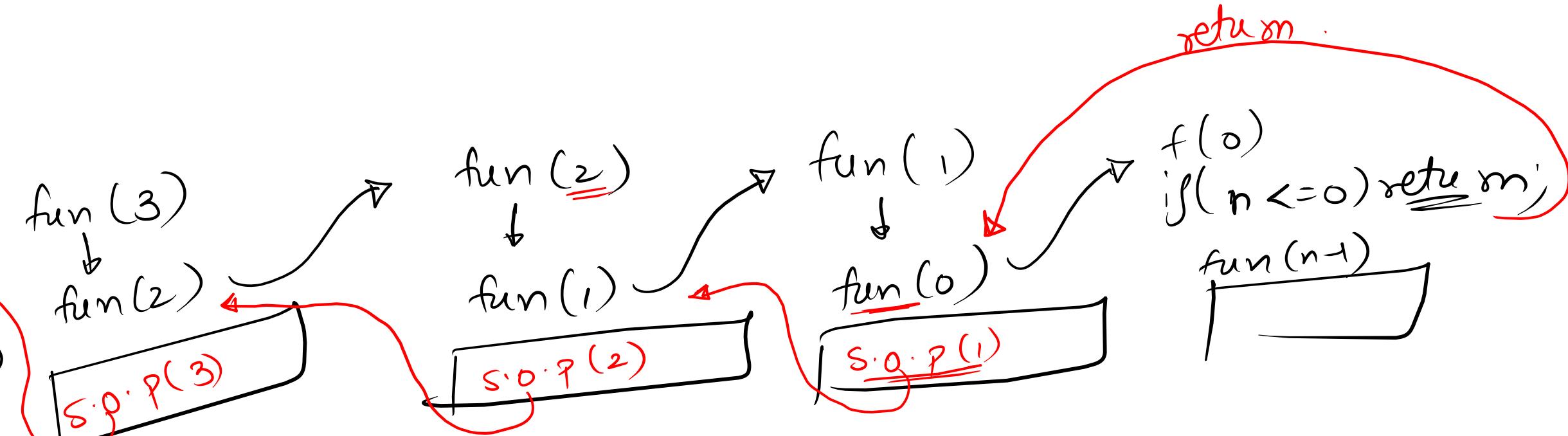
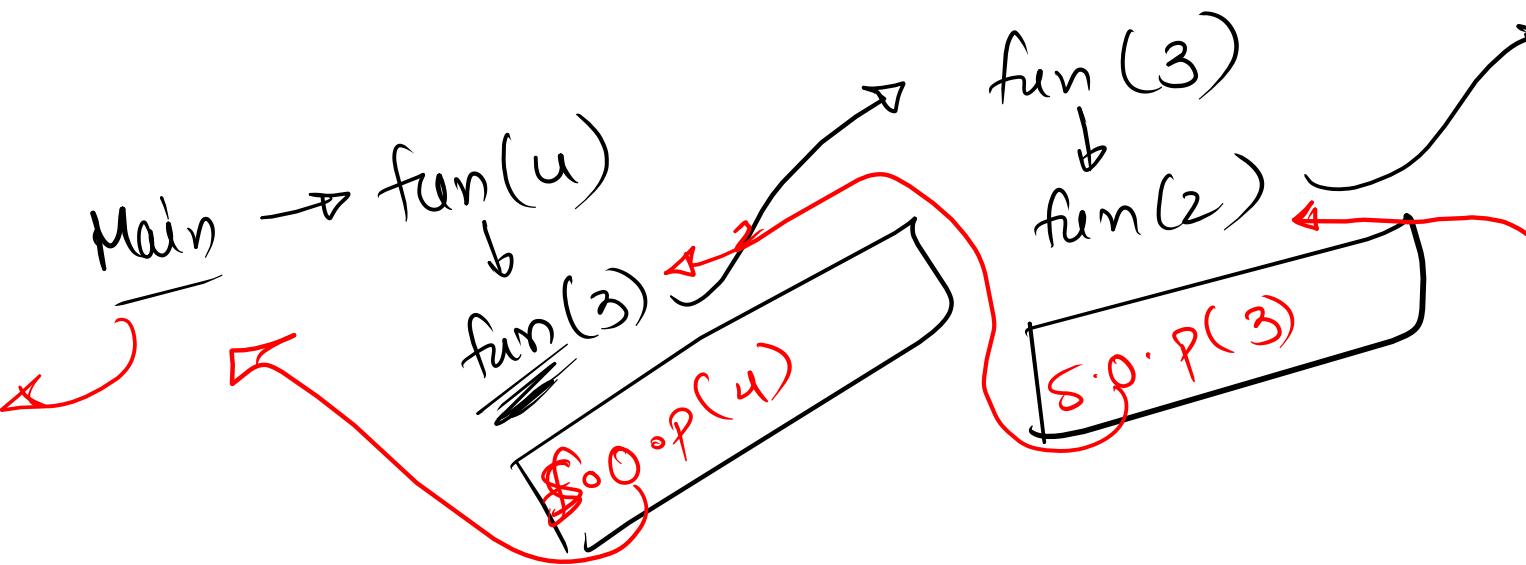
\* Point 

```

public static void fun2(int n){
    if(n <= 0) return ;
    fun2(n-1); //fun(3) fun(2) fun(1) fun(0)
    System.out.print(n+ " ");
}

```

\* PN  
= 1, 2, 3, 4

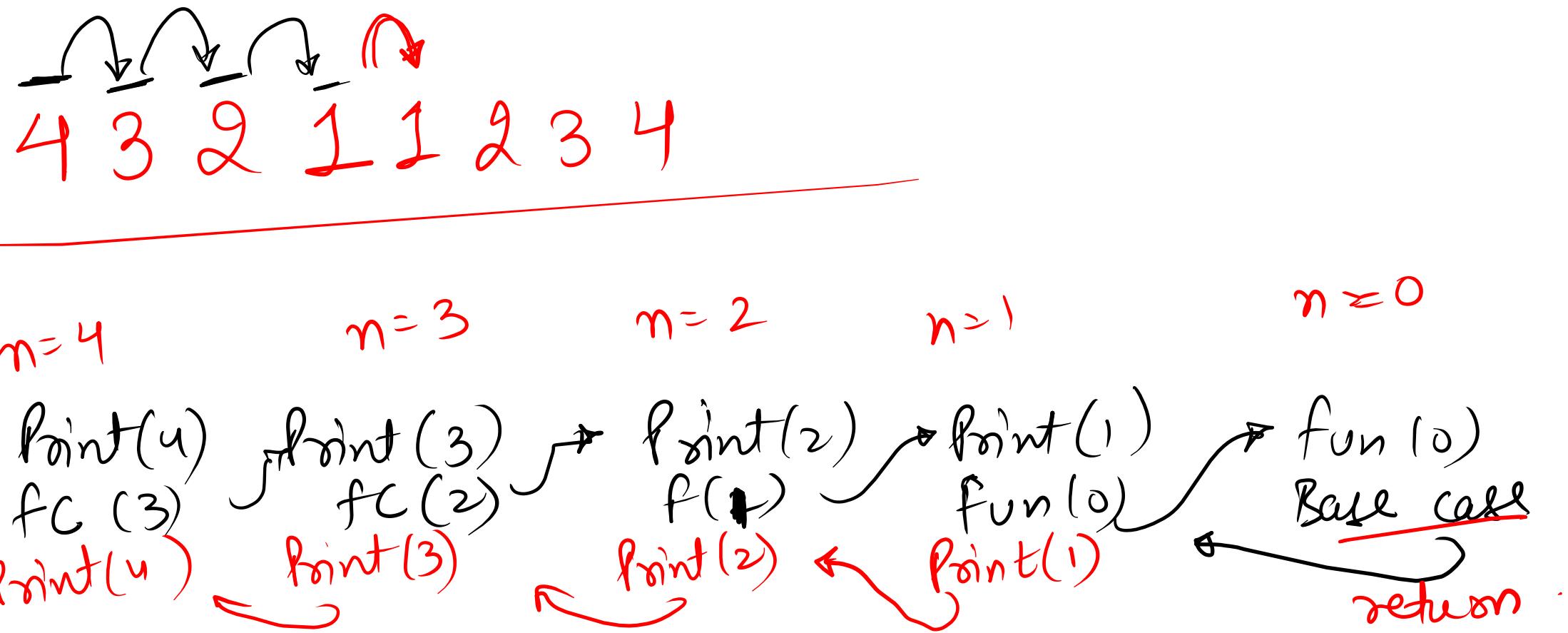


1, 2, 3, 4,

\* Point

n=4

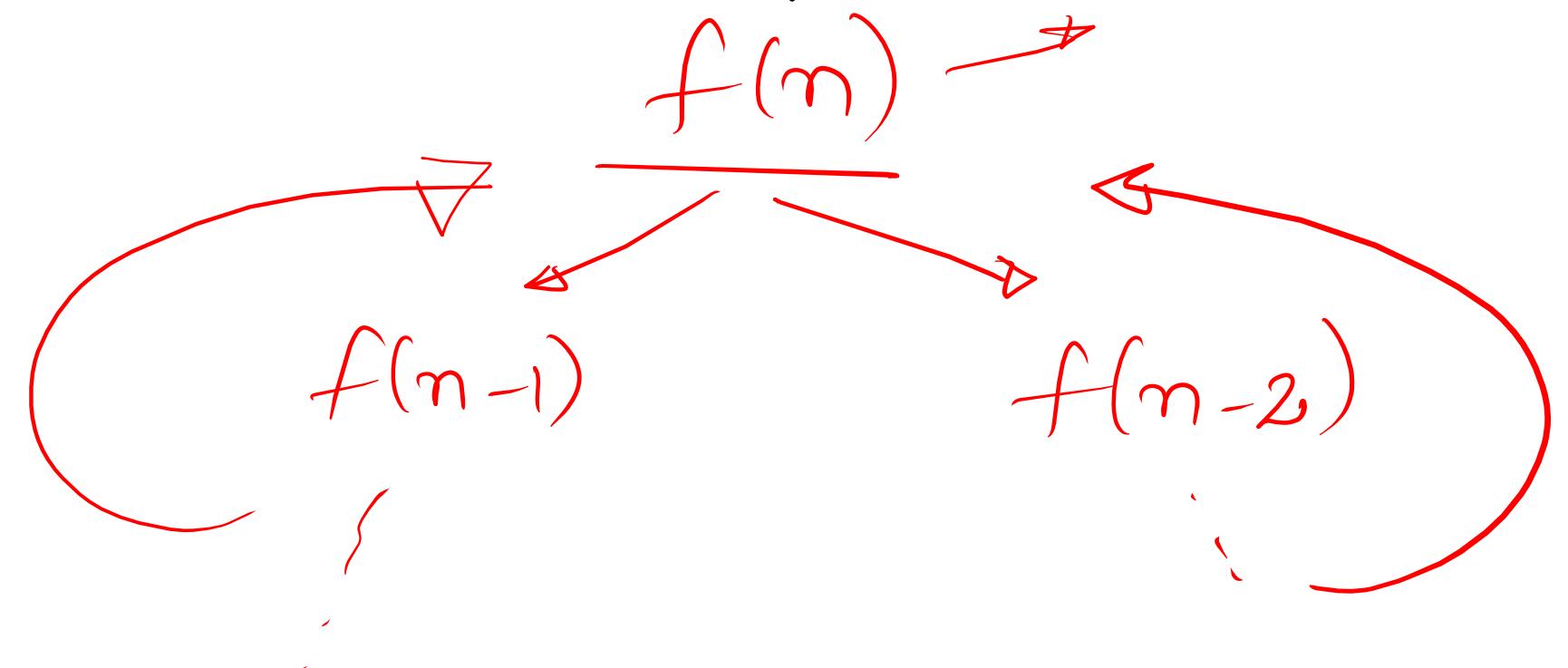
✓ ↓ Point()  
✓ ↓ func. call()  
✓ ↓ Point()



\* Fibonacci  $\Rightarrow$

1<sup>st</sup> term    2<sup>nd</sup> term    3<sup>rd</sup> term    4<sup>th</sup>  
0, 1, 1 = , 2  
 $t_{n-2} + t_{n-1} = t_n$

sum  
 $t_n = t_{n-1} + t_{n-2}$



//Base

if ( $n = 0$ ) return 0;

\* int  $a = \boxed{\text{fun}(n-1)}$

1<sup>st</sup> term

int  $b = \boxed{\text{fun}(n-2)}$

if ( $n == 1$ ) return 1;

2<sup>nd</sup> term

return  $a+b$ ;

1 0

