Q. AJ Number → Maximum no. of AJ no. present in string → return count

0, 1 → AJ ✗

[No.] → AJ ✓

1. 0 and 1 are not a AJ number.
2. 2,3,5,7,11,13,17,19,23,29 are AJ numbers.
3. Any ~~number not divisible by the numbers in point 2~~ (Given above) are also AJ numbers.

let

x % [No.] != 0 → AJ ✓

Not divisible

x % [No.] != 0

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 → "AJ"

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

for finding maximum

Case-1

1 2 3 4 5

Let suppose this is an AJ

Cnt = ∅ 1

Can i use this substring from 1 2 3 4 5

2 3 4

NO

we are Jus Assuming

Case-2

Let

1 2 3 4 5

check 5 is an AJ no.

Remove

Cnt = ∅ 1

1 2 3 4

Cnt = X 2

1 2 3

Cnt = ∅ 3

1 2

Not an AJ

cnt = 3

1 2

AJ ✗

cnt = 3

AJ ✓

cnt 4

return 4

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

* Actual example for understanding "AJ no."

Ex: 81615

1.) 0, 1 → AJ ⊗ (cnt++)

2.) [2,3,5,7,11,13,17,19,23,29] → AJ ✓ (cnt++ ✓)

3.) (x % [ ] != 0) → AJ ✓ (cnt++ ✓)

5 → AJcnt = 0 1

new no. 8161

remove 5

AJcnt = 1
AJ cnt = 1
AJcnt = 2

remove 61 from 8161

→ x % [no.] Remainders → Divisible.

[6 % 2 == 0
6, 1 → are not AJ no.

(61 % 2 != 0) → Yess → AJ ✓

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

81

81

(8 % 2 != 0) → false → AJ (X)

(81 % 2 != 0) → True

(81 % 3 != 0) → false; break;

Q. stranger string

$n = 4$ (a, b, c, d)

ans = cabd

$n = 6$

idx: 0 1 2 3 4 5

a, b, c, d, e, f

ans = e c a b d f

idx: 4 2 0 1 3 5

odd idx

even idx → insert.

odd idx → append the character

n = 26

0 1 ——— 25
↓ ↓ ↓
a b z

return

——— a ———

n = 27

0 ——— 25 26
↓
z

cyclic order

use %.

```java
//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();
```

n = 6

n = 0 → ans = a

n = 1 → odd → ans = ab

n = 2 → even → ans = cab

temp = (char)('a' + (i%26))

i = 1 → 1%26 → 'b' = temp

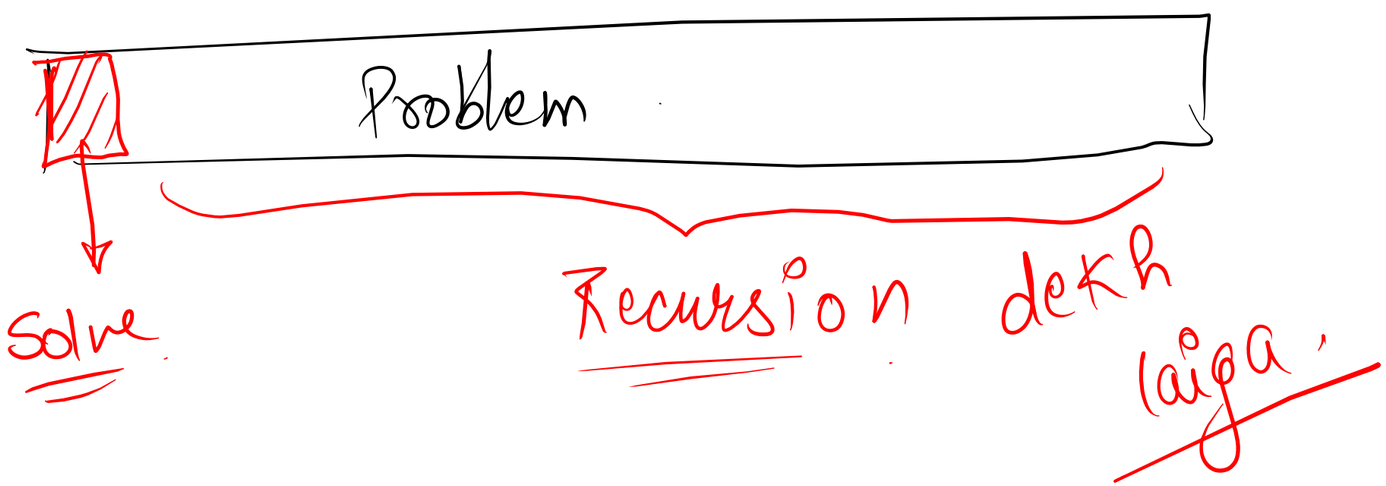
i = 2 → 2%26 → 'c'

Recursion

↓

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

divide & Conquer

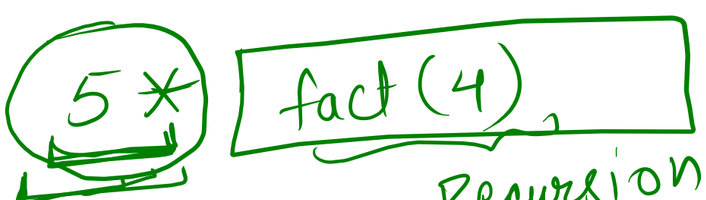


* factorial

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

$= 120$

factorial (5)



Recursion

factorial n = 5. 5×4×3×2×1

Recursion Tree

5 * fact(4)

4 * fact(3)

3 × fact(2)

2 × fact(1)

0 × fact(-1)

1 * fact(-1)

∞

f(-2)
f(-1)
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)

call stack

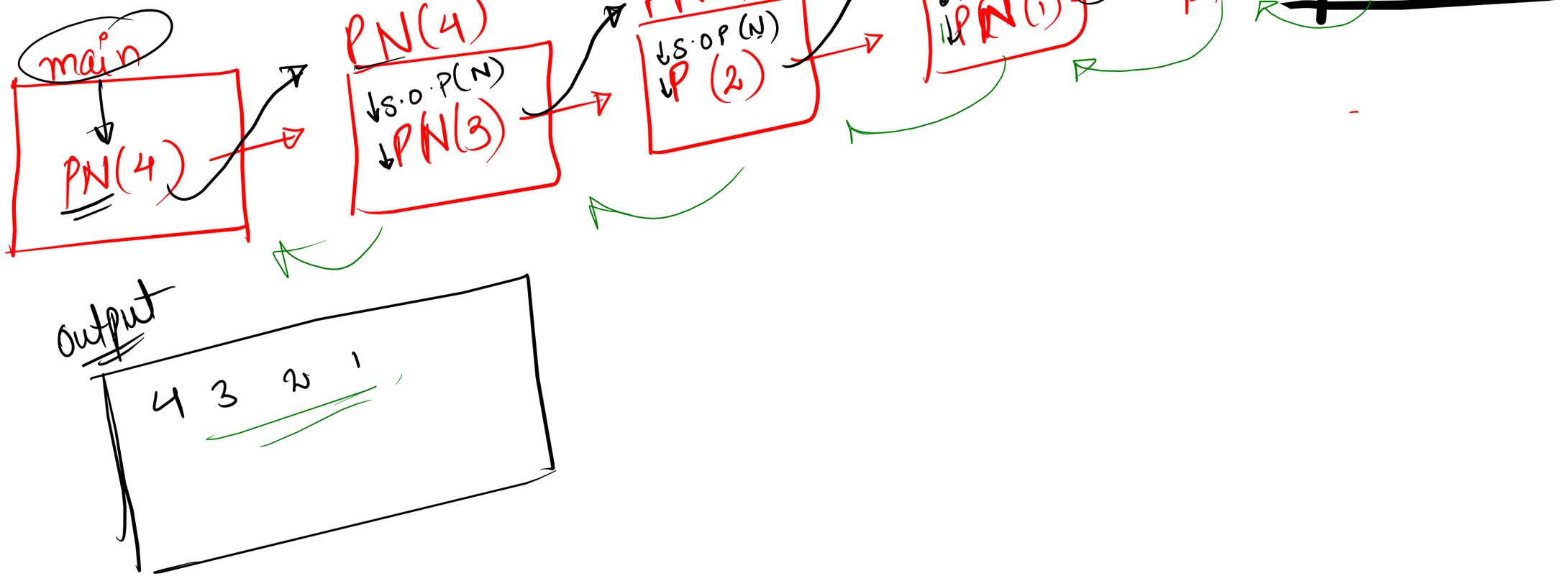
if you didn't add base case. Then the function call itself at ∞

stack overflow

So avoid stack overflow i have to add base case.

(1) Print no. [1, n]

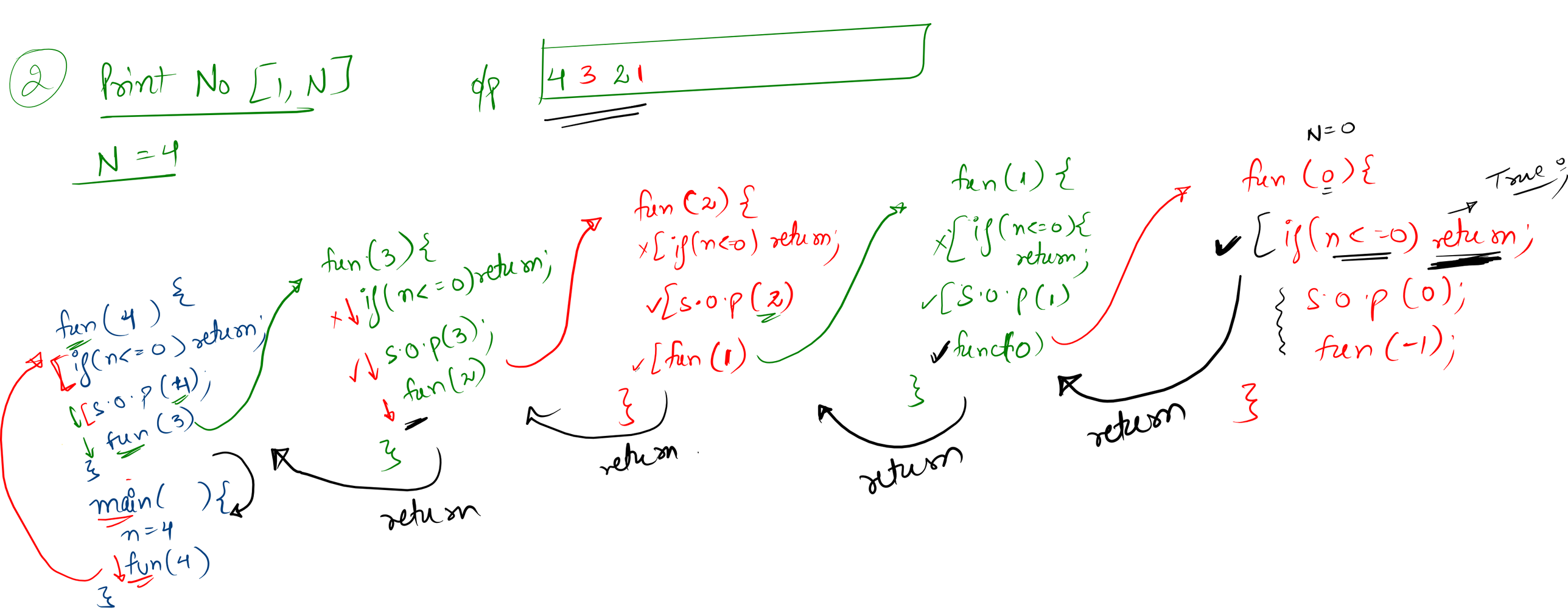
n = 4 o/p => 4, 3, 2, 1



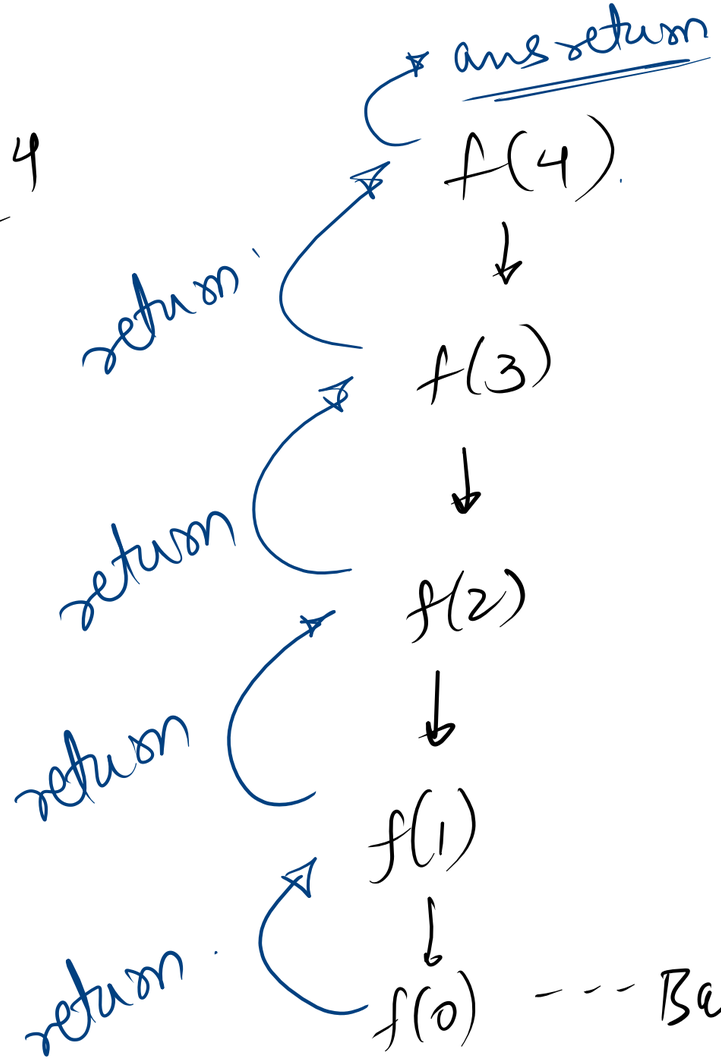
(2) Print No [1, N]

op 4 3 2 1

N = 4



n = 4

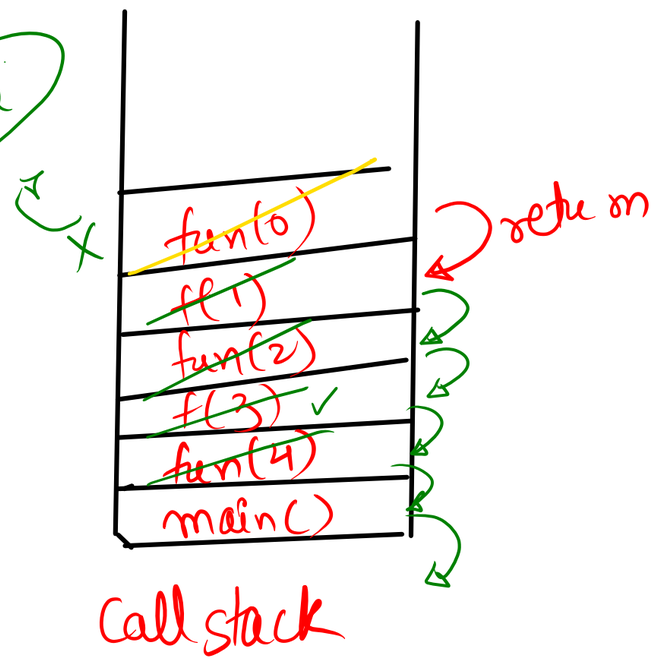


Recursion Tree

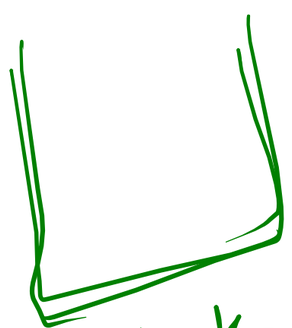
function calling
n → (n+1)

* Time complexity : O(N) ~ n

* space complexity : O(N)

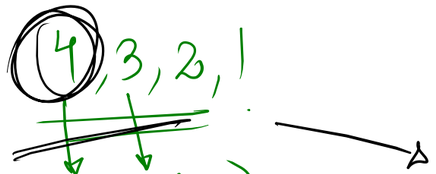


Call stack



stack empty

* Print 4, 3, 2, 1

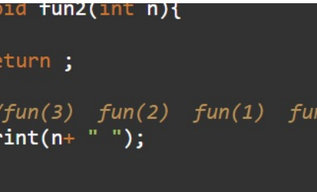


```java
public static void fun(int n){  //n == 4   3

   if(n <= 0) return ;

   System.out.print(n+ " ");    //4  3
   fun(n-1);  // fun(3)    fun(2)

}
```

fun(n) = n , fun(n-1)

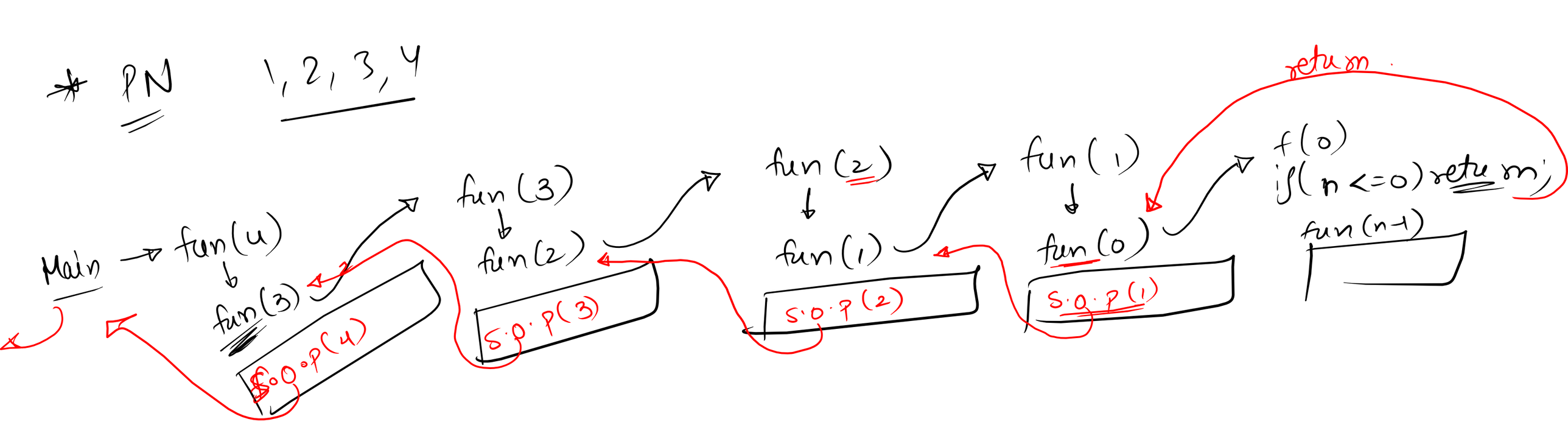
Print (4) ✓ → function call (n-1) ✓ → Print (3) ✓ → func call (n-1) ✓ → Print (2) ✓ → fu. call (n-1) → PN(1) ✓ → f call (n-1) → Base case ✓ → return → P1

* Print 1, 2, 3, 4



```java
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+ " ");

}
```

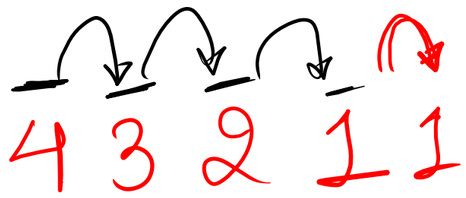
* PN 1, 2, 3, 4

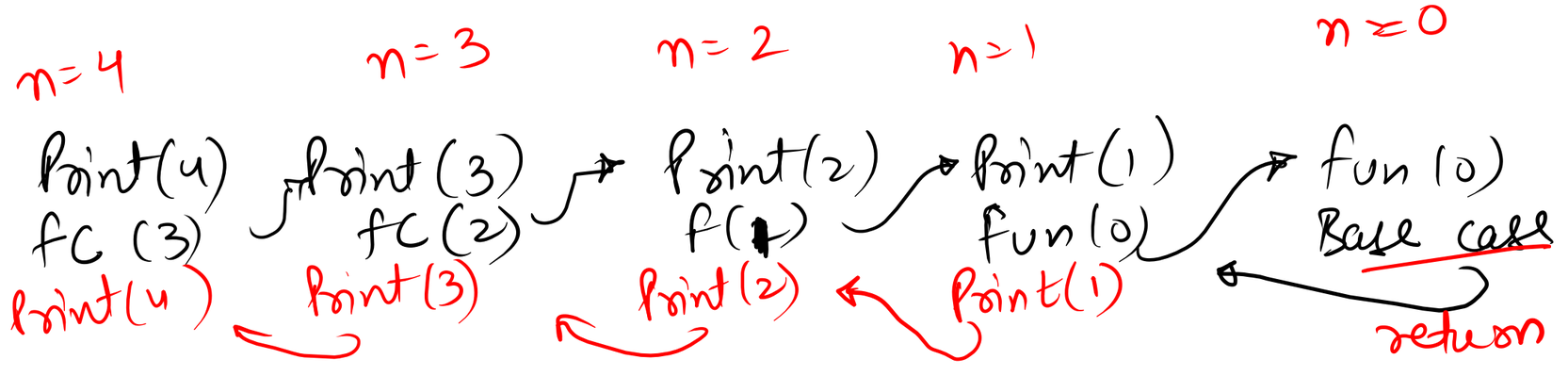

Main
fun(4)
fun(3)
S.O.p(4)
fun(3)
fun(2)
S.O.p(3)
fun(2)
fun(1)
S.O.p(2)
fun(1)
fun(0)
S.O.p(1)
return
f(0)
if(n<=0) return;
fun(n-1)

1, 2, 3, 4,

* Print

n = 4





✓ Print()
✓ Func. call()
✓ Print()

✱ fibonacci ⇒

1st term, 2nd term, 3rd term, 4th

0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , ...

5th, 6th

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

$t_n = t_{n-1} \oplus t_{n-2}$ ← sum

$f(n) \rightarrow$

$f(n-1)$

$f(n-2)$

//Base :- if (n == 0) return 0;

if (n == 1) return 1;

* int a = fun(n-1);

1st term

2nd term

int b = fun(n-2);

return a+b;