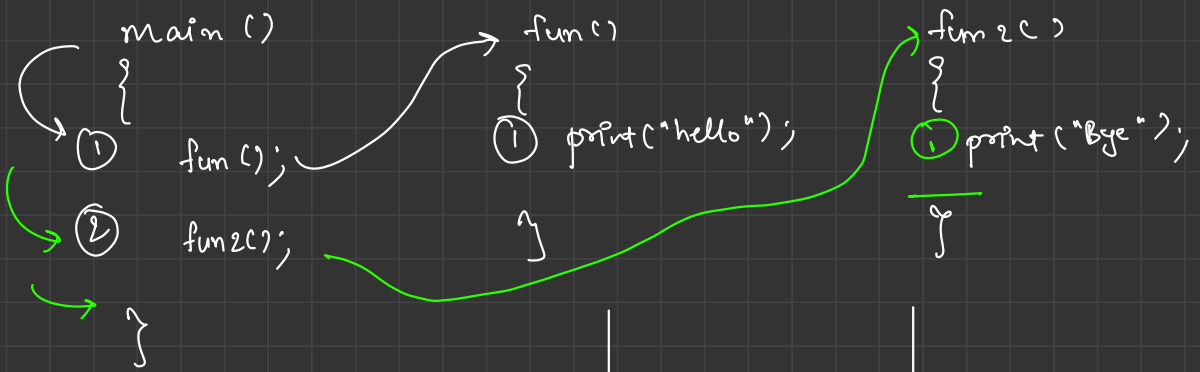
Recursion

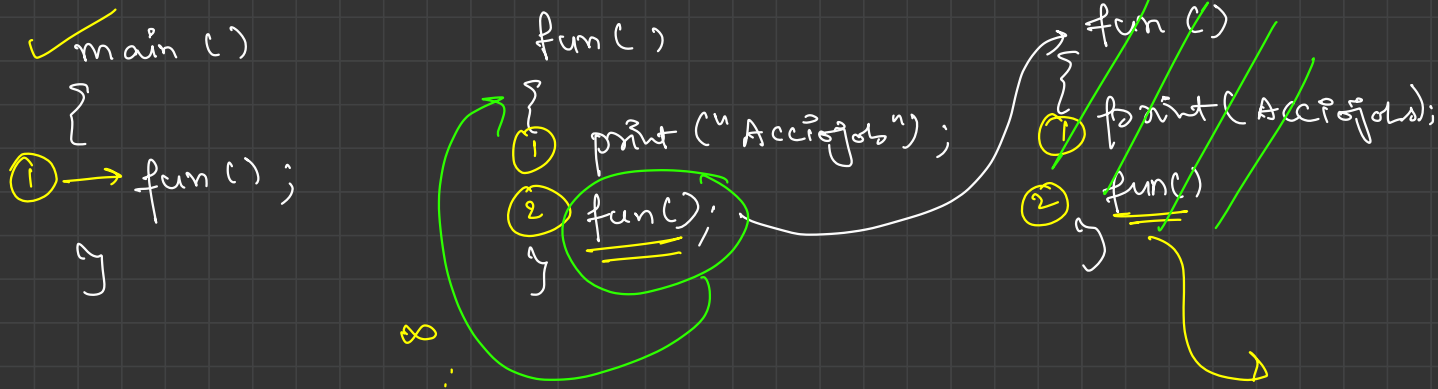


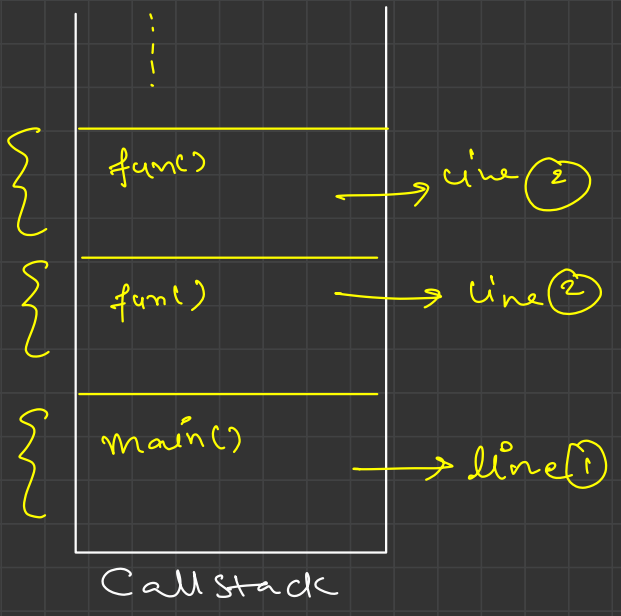
o/p

hello

Bye

Callstack





o/p

Acciojobs

Acciojobs

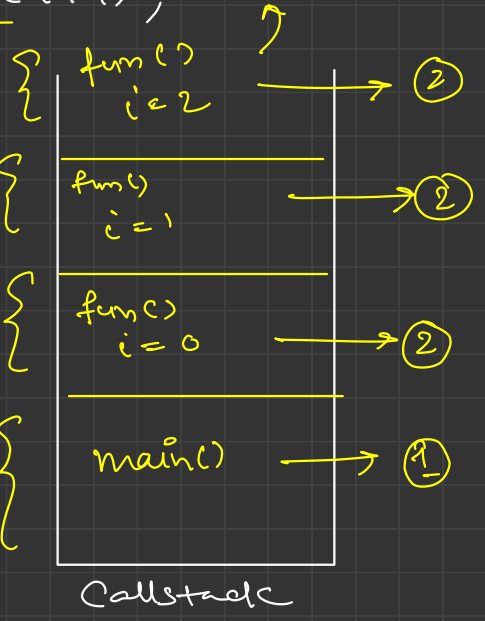
⋮

∞

When a function calls itself then it is known as recursion.

```
main()
{
    fun(0);
}
```

```
fun(int i)
{
    print("Hello" + i);
    fun(i+1);
}
```



O/P

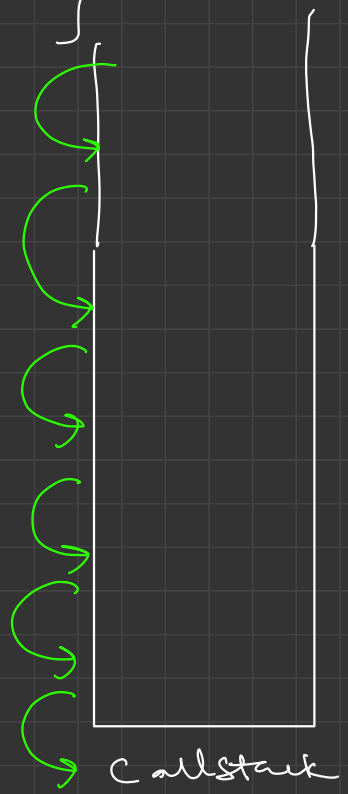
Hello 0
Hello 1
Hello 2

```
main()
{
    fun(0);
}
```

```
fun(int i)
{
    ① if (i == 4)
    ② { return; }
    ③ print("Hello" + i);
    ④ fun(i+1);
}
```

① terminated recursion → ① Base Case

② recursive call



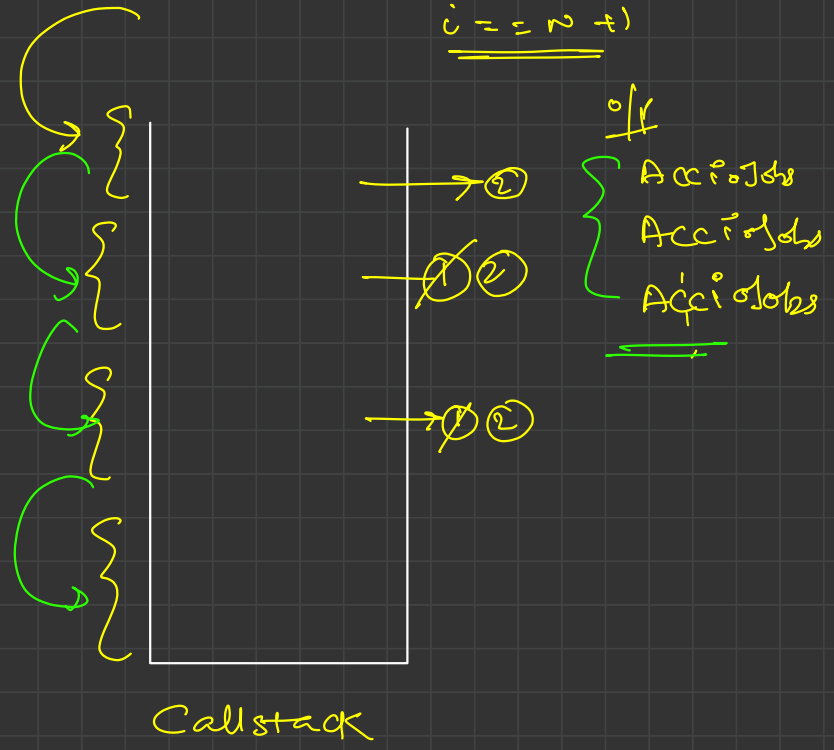
O/P

Hello 0
Hello 1
Hello 2
Hello 3

Q. given a num N;
print (Accio Jobs) N times using recursion!!

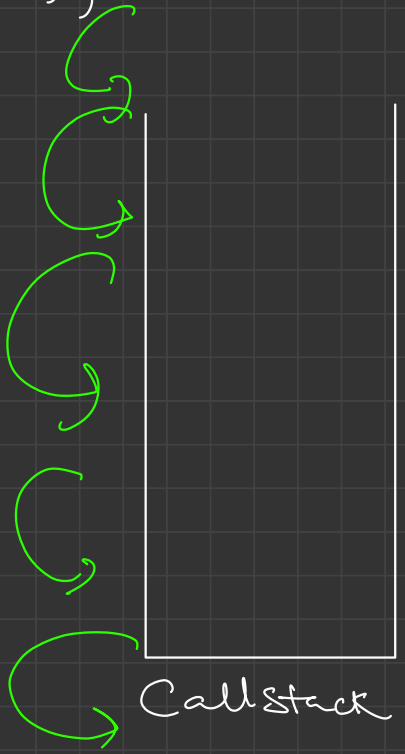
```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int N = sc.nextInt();

    printtillN(N, 1);

}

public static void printtillN(int N, int i) {
    if(i==N+1) return;
}
```

① print("Accio Jobs");
② printtillN(N, i+1);
}



```
fun ( int n)
{  if(n == 0) return;
① printf("Accio ");
② fun(n-1);
}
```

```
main()
{
  fun(3);
}
```

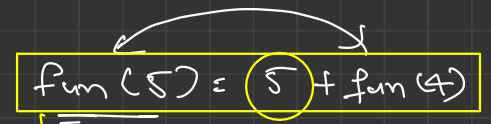


o/p

Accio Jobs
Accio Jobs
Accio Jobs

n Times

Q. n = 5,



fun(4) = 4 + fun(3)

fun(3) = 3 + fun(2)

recurrance relation

```
main()
{
    fun(5);
}
```

```
fun(int n)
{
    if(n==0)
        return;
    print(n);
    fun(n-1);
}
```

n, n-1, n-2 ..... 3, 2, 1

fun(2) = 2 + fun(1)

f(1) = 1 + fun(0)

no.end

Base Case

n given

o/p format (n , n-1 , n-2 , n-3 ------- , 3 , 2 , 1)

① fun(n) = n + fun(n-1) → { print(n) ; fun(n-1) }

② Base Case

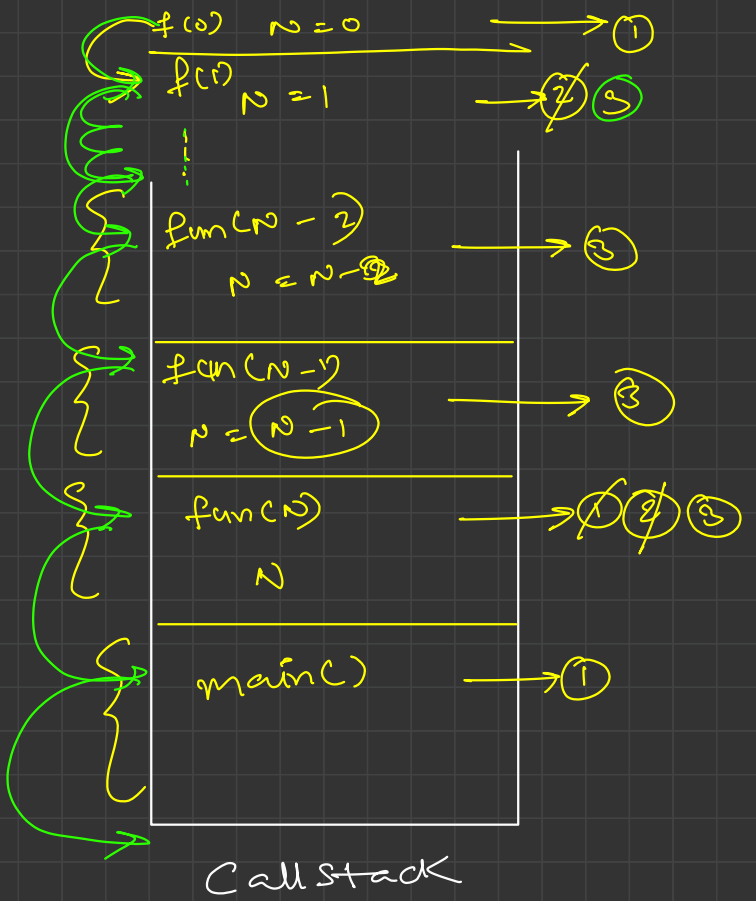
n == 0 → return

```
main()
{
  ① fun(n)
}
```

```
fun(n)
{
  ① if (n == 0)
        return;
  ② print(n);
  ③ fun(n-1);
}
```

o/p

n, n-1, n-2, ..., 1



Q

func(5) → o/p [5 4 3 2 1 1 2 3 4 5]

f(0)

$f(5) = \text{print}(5) + f(4) + \text{print}(5)$

$f(4) = \text{print}(4) + f(3) + \text{print}(4)$

$f(3) = \text{print}(3) + f(2) + \text{print}(3)$

$f(2) = \text{print}(2) + f(1) + \text{print}(2)$

$f(1) = \text{print}(1) + f(0) + \text{print}(1)$

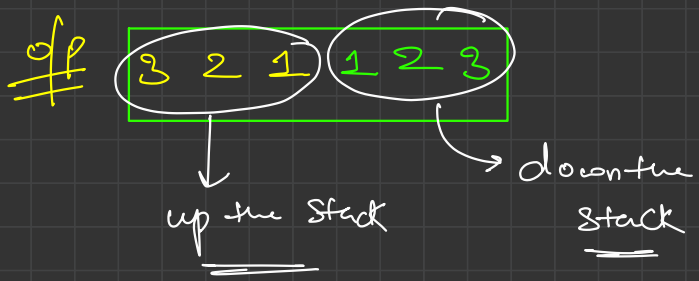
Base Case / Terminating Condition

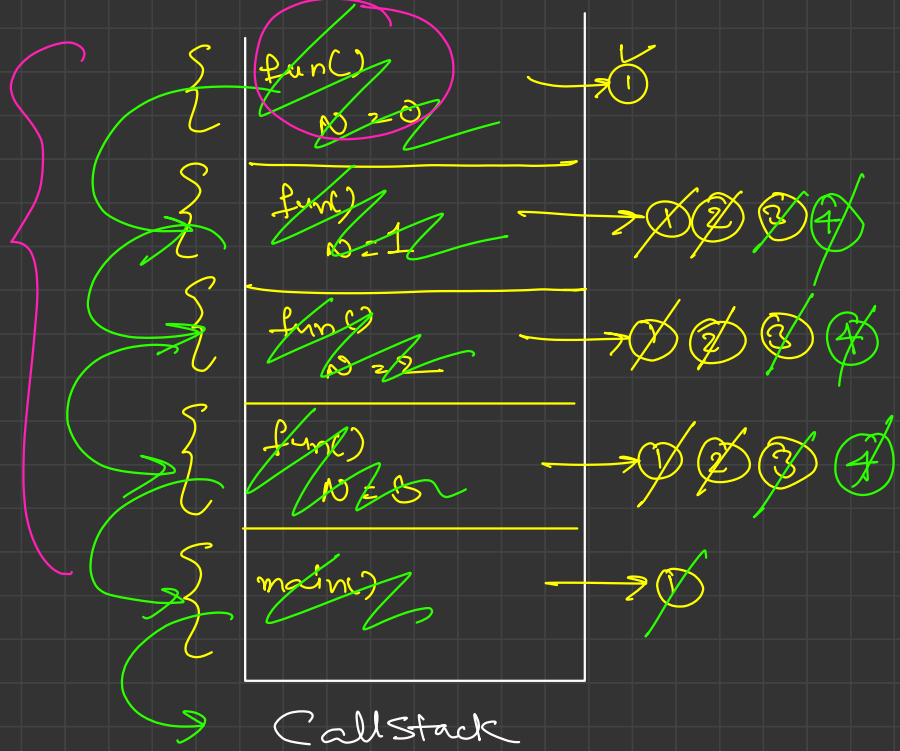
fun(N) = print(N) + fun(N-1) + print(N) → Recurrance

```
main()
{
  ① fun(N)   ③
}
```

TC: O(N)
SC: O(N)

```
fun(int N)
{
  ① if(N==0)
      return;
  ② print(N);
  ③ fun(N-1);
  ④ print(N);
}
```





N = 5 ⟶ 5 4 3 2 (1) 2 3 4 5

$f(5) = 5 + f(4) + 5$

$f(4) = 4 + f(3) + 4$

$f(3) = 3 + f(2) + 3$

$f(2) = 2 + f(1) + 2$

↓

Base Case

{ print()
return; }

factorial

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

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

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

$2! = 2 \times 1 = 2 \times 1!$

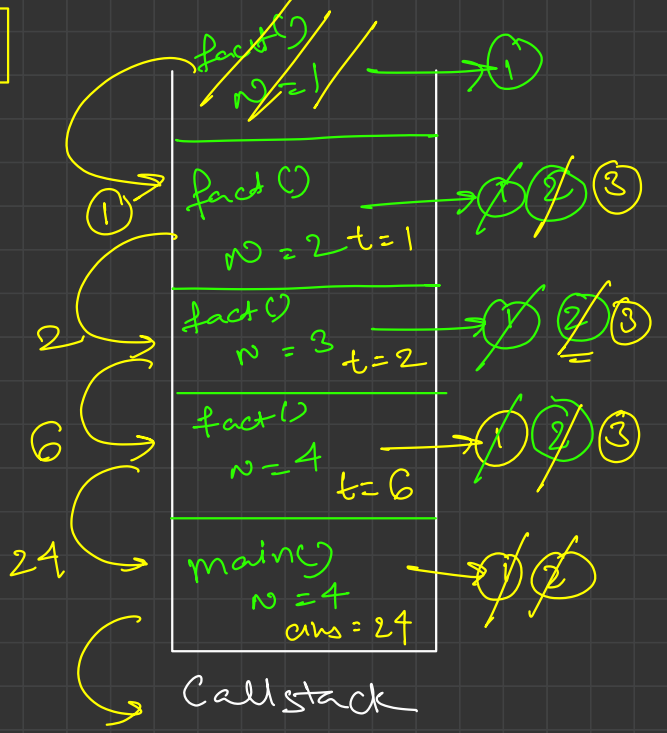
$1! = 1$ → Base case !

$f(n) \rightarrow n!$

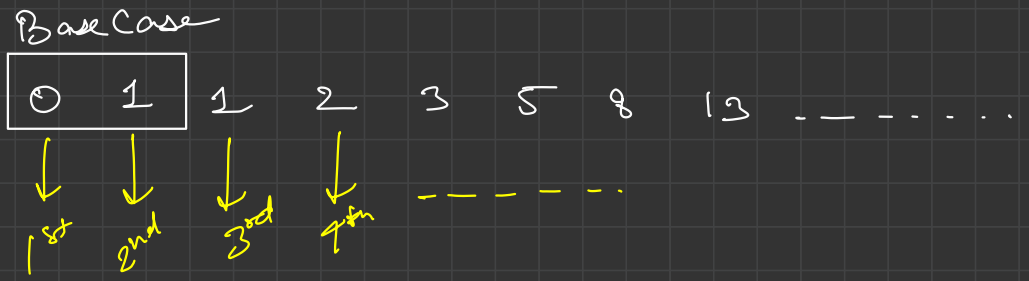
$f(n) = n \times f(n-1) = f(n-1) \times n$

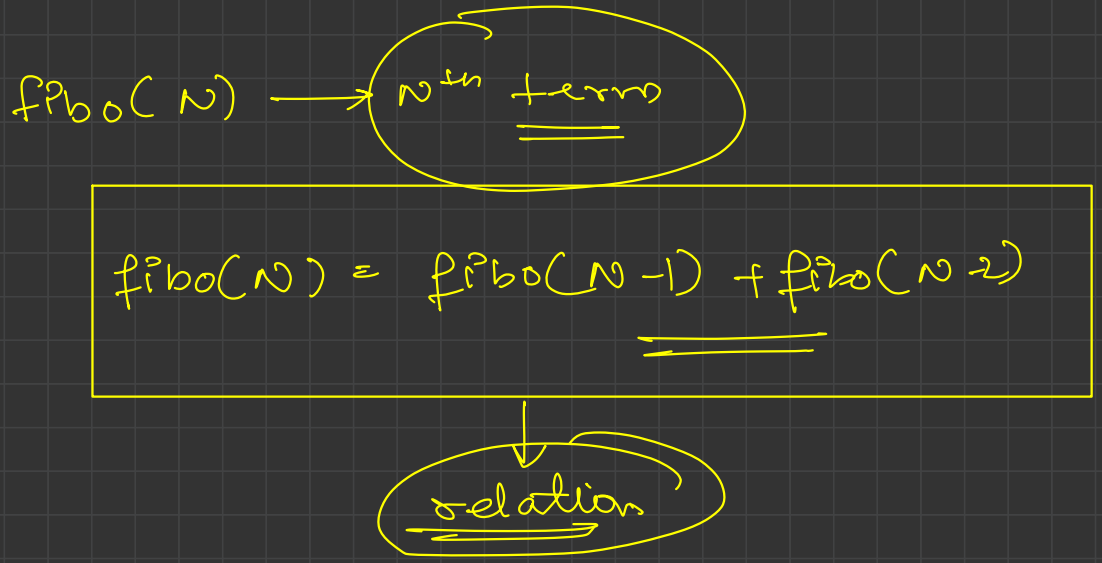
```
main()
{
    ans = fact(N);
    printf(ans);
}
```

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

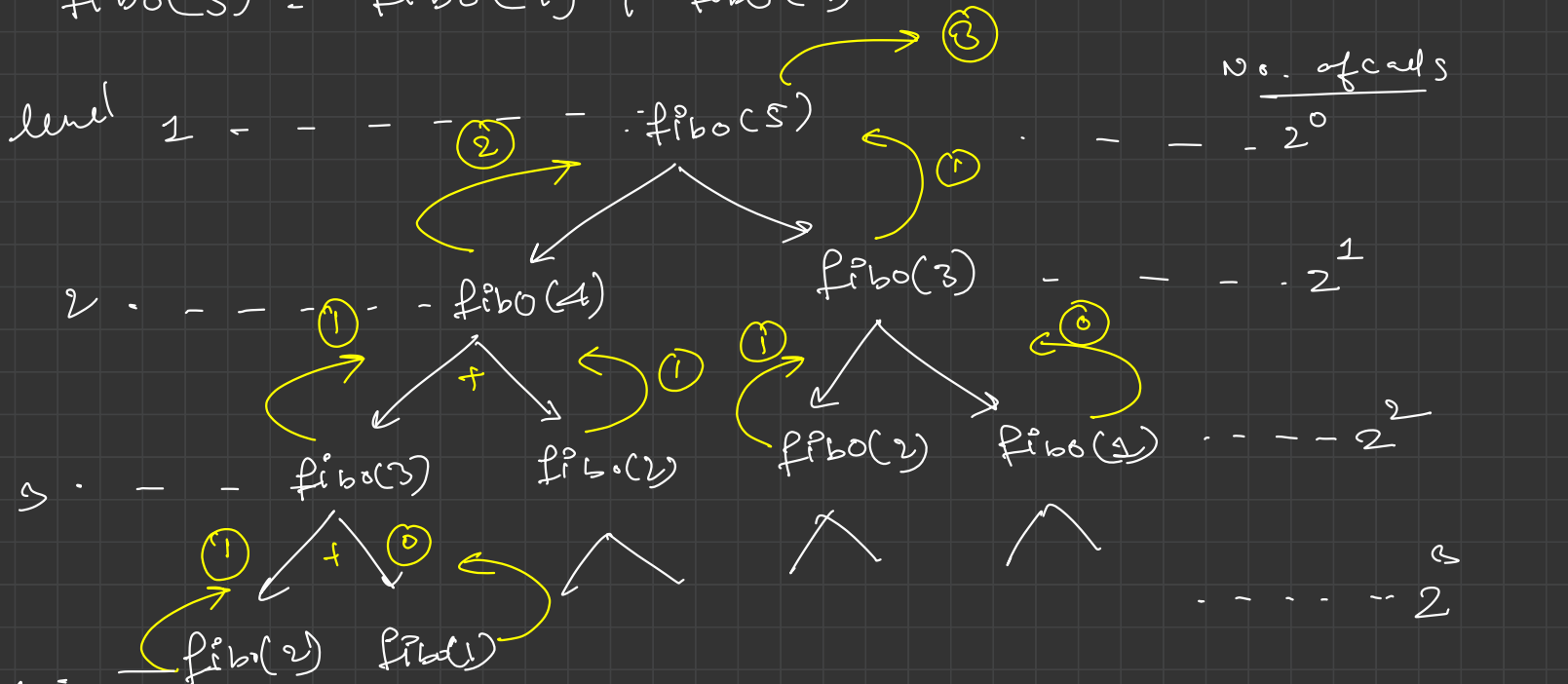


Get N^{th} Fibo Number





fibo(5) = fibo(4) + fibo(3)



SC: O(N)

TC: $O(2^n)$

$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{n-1}$

$S_n = \frac{1(2^n - 1)}{(2-1)} = (2^n - 1)$

```
main()
{
1 int ans = fibo(N);
2 print(ans)
}

int fibo(int N)
{
1 if(N==1) return 0;
2 if(N==2) return 1;
3 int prev1 = fibo(N-1);
4 int prev2 = fibo(N-2);
5 return prev1 + prev2;
}
```

O(1)

0/1/2 3

Q. $(p, q) \rightarrow p^q$

$\rightarrow$ func(p, q) $\rightarrow p^q$

$p^q = \overbrace{p \times p \times p \times \cdots \times p}^{q \text{ times}}$

$p^{q-1} = \overbrace{p \times p \times p \times p \times \cdots \times p}^{q-1 \text{ times}}$

$\vdots$

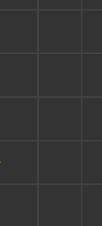
$p^1 = \overbrace{p}^{1 \text{ times}}$

$p^0 = 1$ $\rightarrow$ Base case

$$p^q = p \times p^{q-1}$$

$$f(p,q) = p \times f(p,q-1) = f(p,q-1) \times p$$

```
main()
{
int ans = fun(p, q);   ①
printf(ans);           ②
}

int fun(p, q)
{ if(q == 0) return 1;   ①
  int t = f(p, q-1);     ②
  return t * p;          ③
}
```



TC : O(q)

SC : O(q)

Space of stack

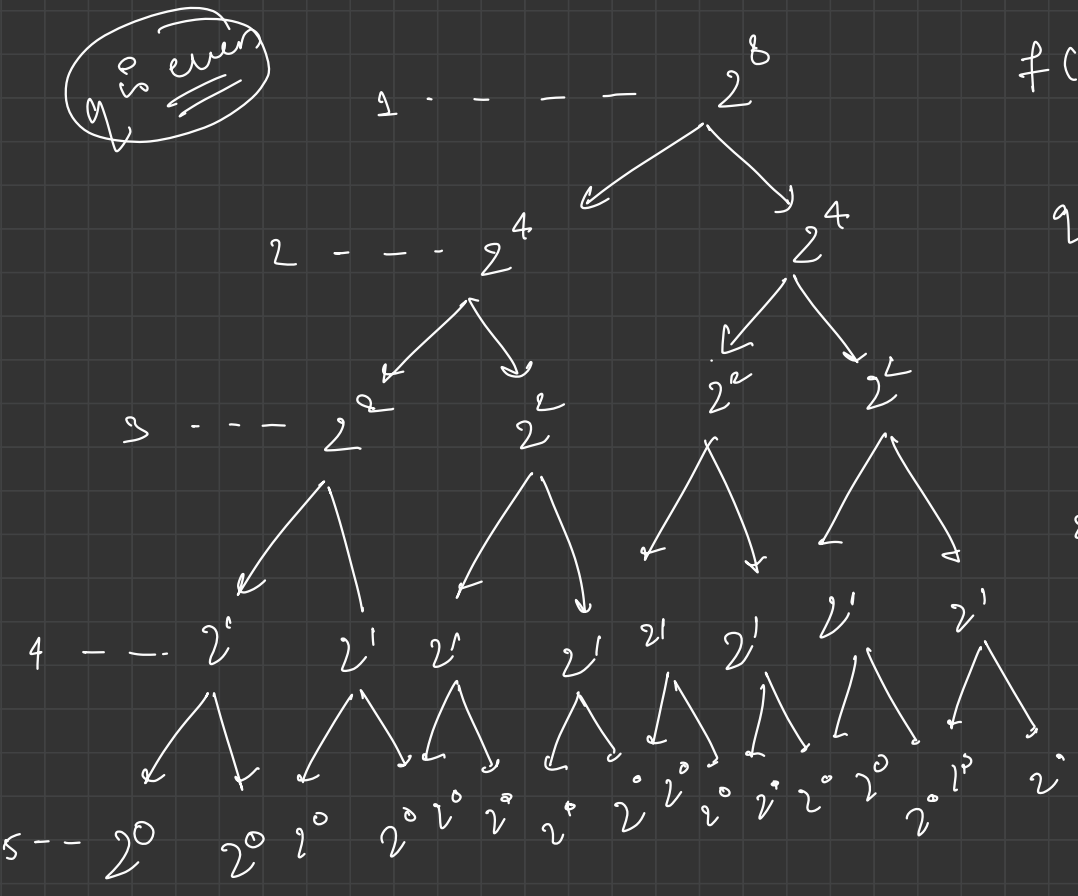
$$p^a = \underbrace{p \times p \times p \times \cdots \times p}_{a \text{ times}}$$

$$2^6 = \underbrace{2 \times 2 \times 2 \times 2}_{} \times \underbrace{2 \times 2 \times 2 \times 2}_{}$$

$$2^4 \times 2^4$$

$$2^2 \times 2^2 \qquad 2^2 \times 2^2$$

q is even



$f(p,q) = f(p,q/2) \times f(p,q/2)$

$q + \frac{q}{2} + \frac{q}{4} \cdots\cdots + 0$

$= \log(q)$

$2^0 + 2^1 + 2^2 \cdots + 2^{\log q}$

$= \frac{1(2^{\log_2 q} - 1)}{(2-1)}$

$= 2^{\log_2 q}$

$f(p,q) = (f(p,q/2))^2$

```
func(p,q)
{
  if(q==0) return 1;
  int t = f(p, q/2);
  return t*t;
}
```

f(2,8)
↓
f(2,4)
↓
f(2,2)
↓
f(2,1)
↓
f(2,0)

log(q) levels

TC: O(log q)

SC: O(log q)

$7/2 = 3$

$2^7 \longrightarrow (2^3)^2 \times 2$

$$p^q = \begin{cases} (p^{q/2})^2 & , q \text{ is even} \\ (p^{q/2})^2 \times p & , q \text{ is odd} \end{cases}$$

```
fun(p, q)
{ if(q==0) return 1;
  int t = f(p, q/2);
  if (q == odd) return t*t*p;
  else return t*t;
}
```

```
main()
{
1 int ans = f(2,8);
2 printf(ans);
}

int f(int p, int q)
{
1 if(q==0) return 1;
2 int t = f(p, q/2);
3 if(q%2==1)
    return t*t*p;
4 else
    return t*t;
}
```

ans
256

print pattern -

n = 5

f(5) = f(4) + print(5) times *

```
main()
{
  f(N);
}
f(int N)
{
  if(n==0) return;
  f(N-1);
  for(i=0 → n)
    print("* ");
}
```

$f(5) = f(4) + print(5*)$

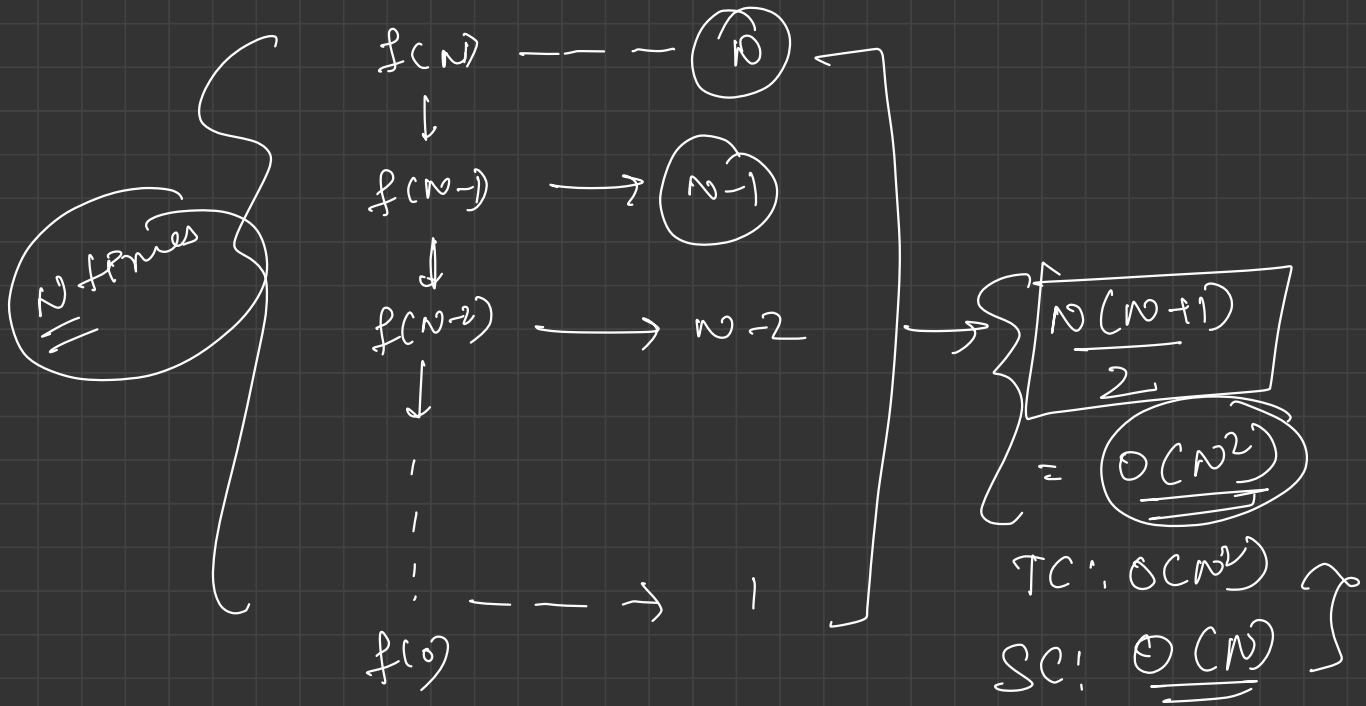
$f(4) = f(3) + print(4*)$

$f(3) = f(2) + print(3*)$

$f(2) = f(1) + print(2*)$

$f(1) = f(0) + print(1*)$

$f(0)$ → Base Case → return

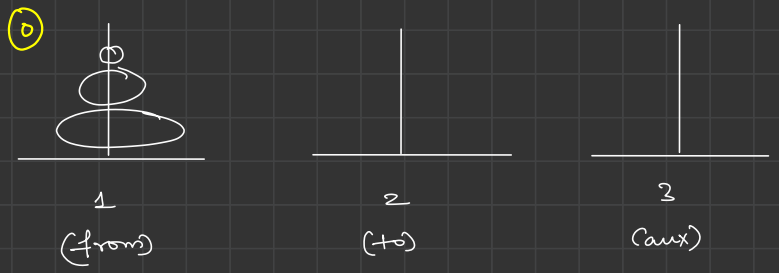

f(N)
N
f(N-1)
N-1
f(N-2)
N-2
f(0)
1
N times
N(N+1)/2
= O(N^2)
TC: O(N^2)
SC: O(N)

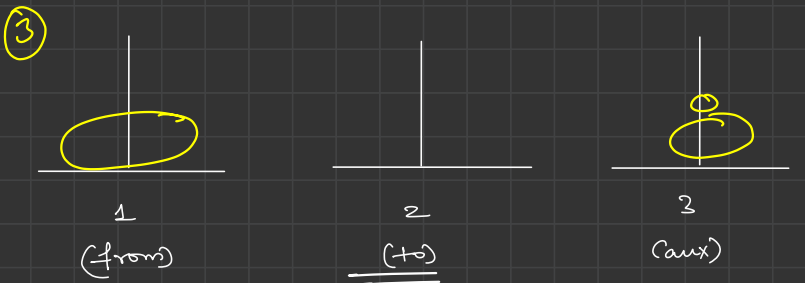
Tower of Hanoi

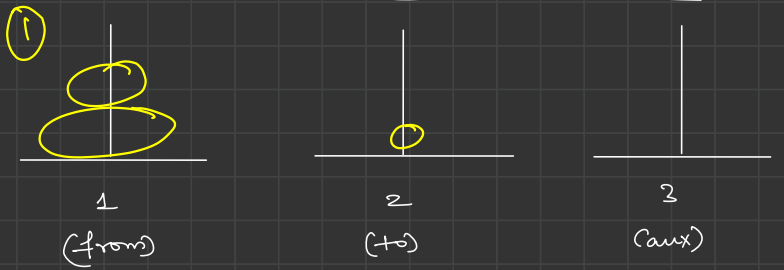
transfer all the disk from 1 to 2 in same order.
using aux rod (3)

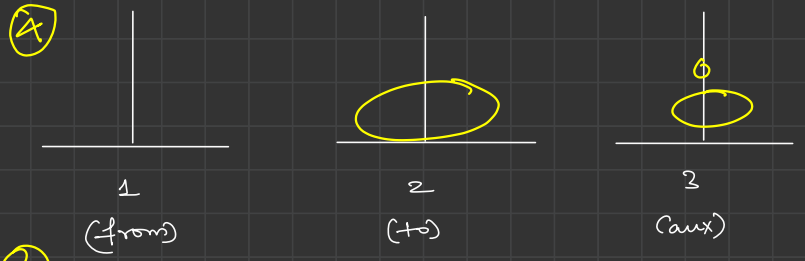
① you can only move one disk at a time

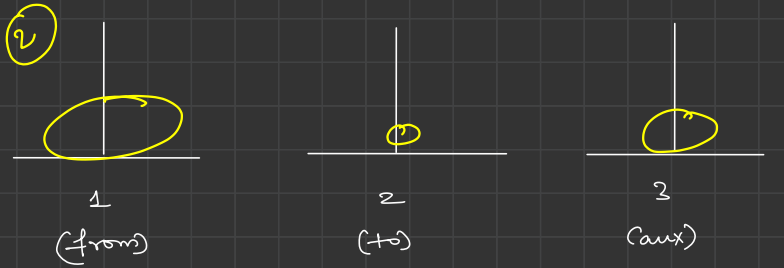
② you can't place a bigger disk on a smaller disk

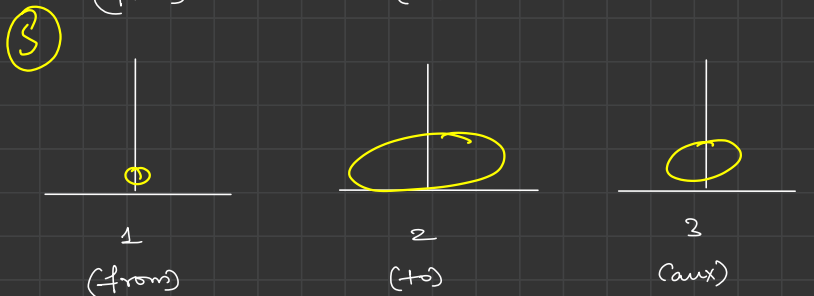

0
1
(from)
2
(to)
3
(aux)


3
1
(from)
2
(to)
3
(aux)

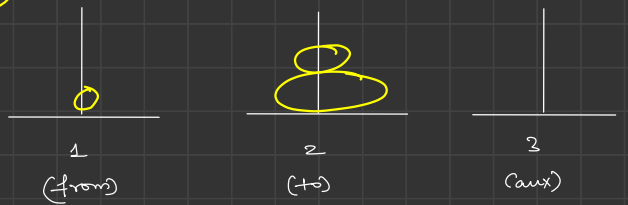

1
1
(from)
2
(to)
3
(aux)


4
1
(from)
2
(to)
3
(aux)

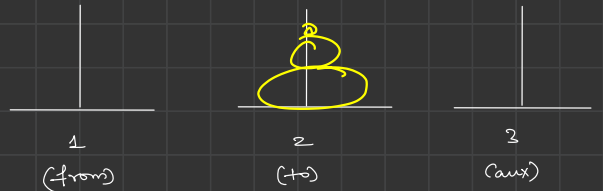

2
1
(from)
2
(to)
3
(aux)


5
1
(from)
2
(to)
3
(aux)

(6)



(7)



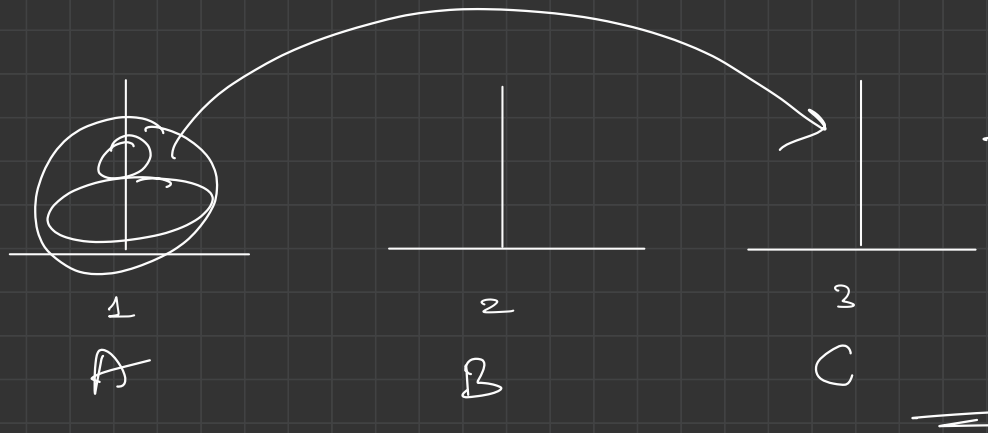
(7) disk , (5) disk , (6) disk

from to aux

toh (N , A , B , C)

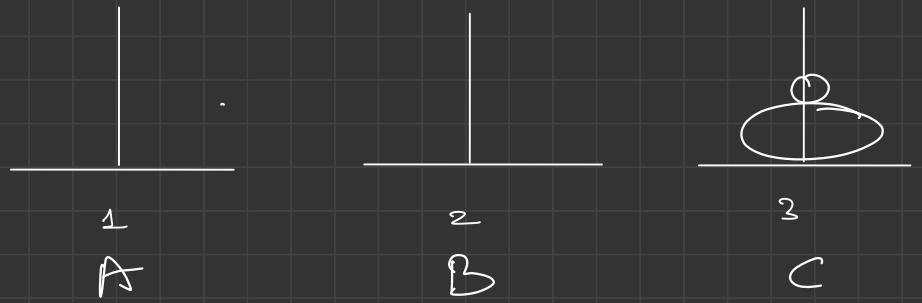
It will transfer N disk from 1 to 2 using 3 → C

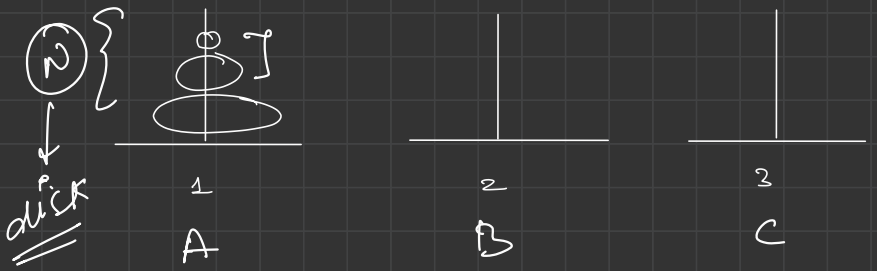
A → B



toh(2, A, C, B)

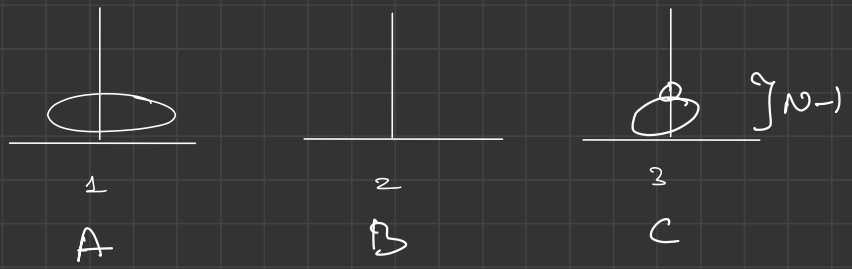
o/p



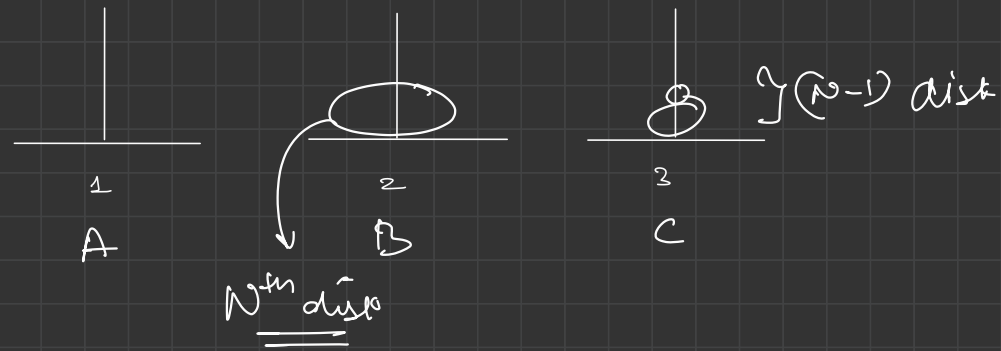


step1

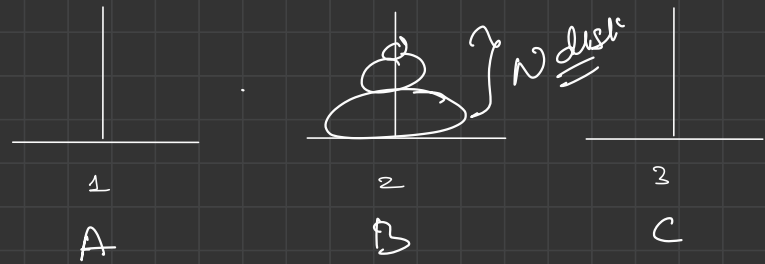
toh (N-1 , A , C , B)



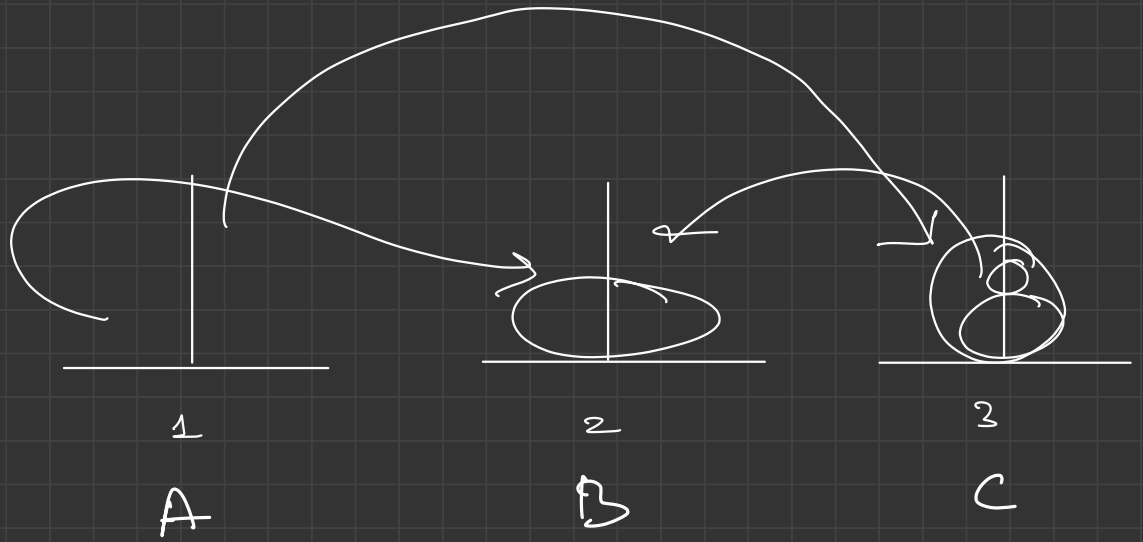
step2 move N^{th} disk from A to B



step3 toh(N-1, C, B, A)



```
toh(N, A, B, C)
{
    if(N==0)
        return;

    toh(N-1, A, C, B);
    print(move Nth disk from A to B);
    toh(N-1, C, B, A);
}
```

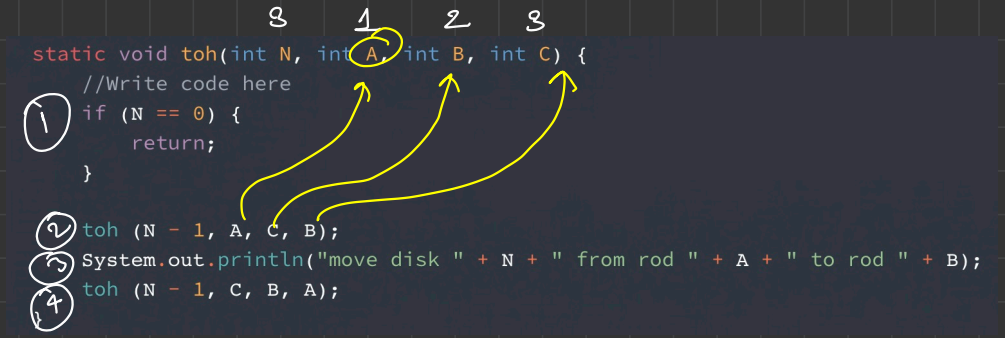


toh(N, A, B, C)

→ toh(2, A, C, B)

move 3 → A → B

→ toh(2, C, B, A) →



move disk 1 from rod 1 to rod 2
move disk 2 from rod 1 to rod 3
move disk 1 from rod 2 to rod 3
move disk 3 from rod 1 to rod 2
move disk 1 from rod 3 to rod 1
move disk 2 from rod 3 to rod 2
move disk 1 from rod 1 to rod 2

```
move disk 1 from rod 1 to rod 2
move disk 2 from rod 1 to rod 3
move disk 1 from rod 2 to rod 3
move disk 3 from rod 1 to rod 2
move disk 1 from rod 3 to rod 1
move disk 2 from rod 3 to rod 2
move disk 1 from rod 1 to rod 2
```

Callstack

Print Array

Q arr = [A, B, C, D, E] N = 5

```
void fun ( arr, N)
{  if (N==0) return;
   fun ( arr, N-1);
   print (arr[N-1])
}
```

fun(arr, N-1) ← Faith

print(arr[N-1]) → N^{th} Element

O/P : ABCDE

F(N) = ABCDE

F(N-1) = ABCD

F(N) : F(N-1) + print(E)

recurrance relation!

f(5) : f(4) + E

f(4) : f(3) + D

f(3) : f(2) + C

f(2) : f(1) + B

f(1) = f(0) + A

↓

Base Case

```
if (N == 0)
    return;
```

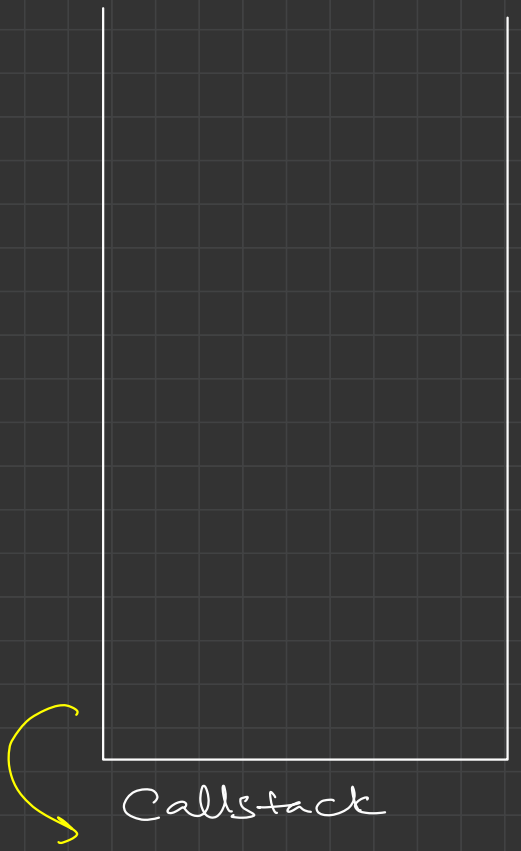
arr = [A, B, C, D, E]
0 1 2 3 4

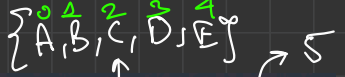
5

```java
public static void PrintArray(int[] arr, int n) {
    // Write your code here
    if (n == 0) {
        return;
    }

    PrintArray (arr, n - 1);

    System.out.print(arr[n - 1] + " ");
}
```

① ② ③

O/P A B C D E





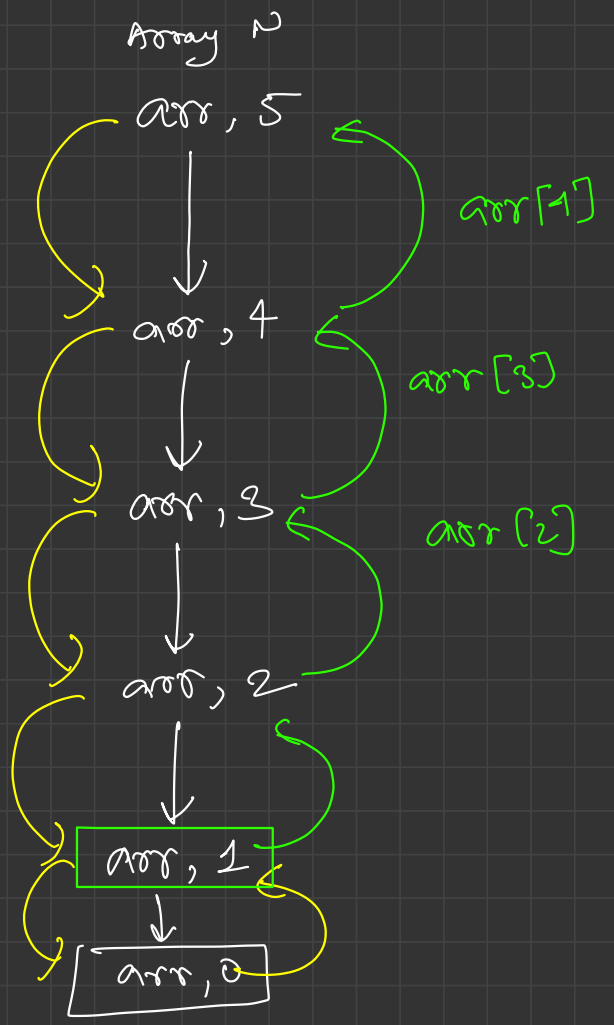

```java
public static void PrintArray(int[] arr, int n) {
    // Write your code here
    if (n == 0) {
        return;
    }

    PrintArray (arr, n - 1);

    System.out.print(arr[n - 1] + " ");
}
```

TC : O(N)
SC : O(N)

① while going down run everything before recursive call.

② while going up, run everthing after recursive call

0/1 A B C D E



Q

0	1	2	3	4
A	B	C	D	E

arr = [A, B, C, D, E] N = 5

fun(arr, N) o/p : E D C B A

f aidy :

f(5) = print(E) + f(4)

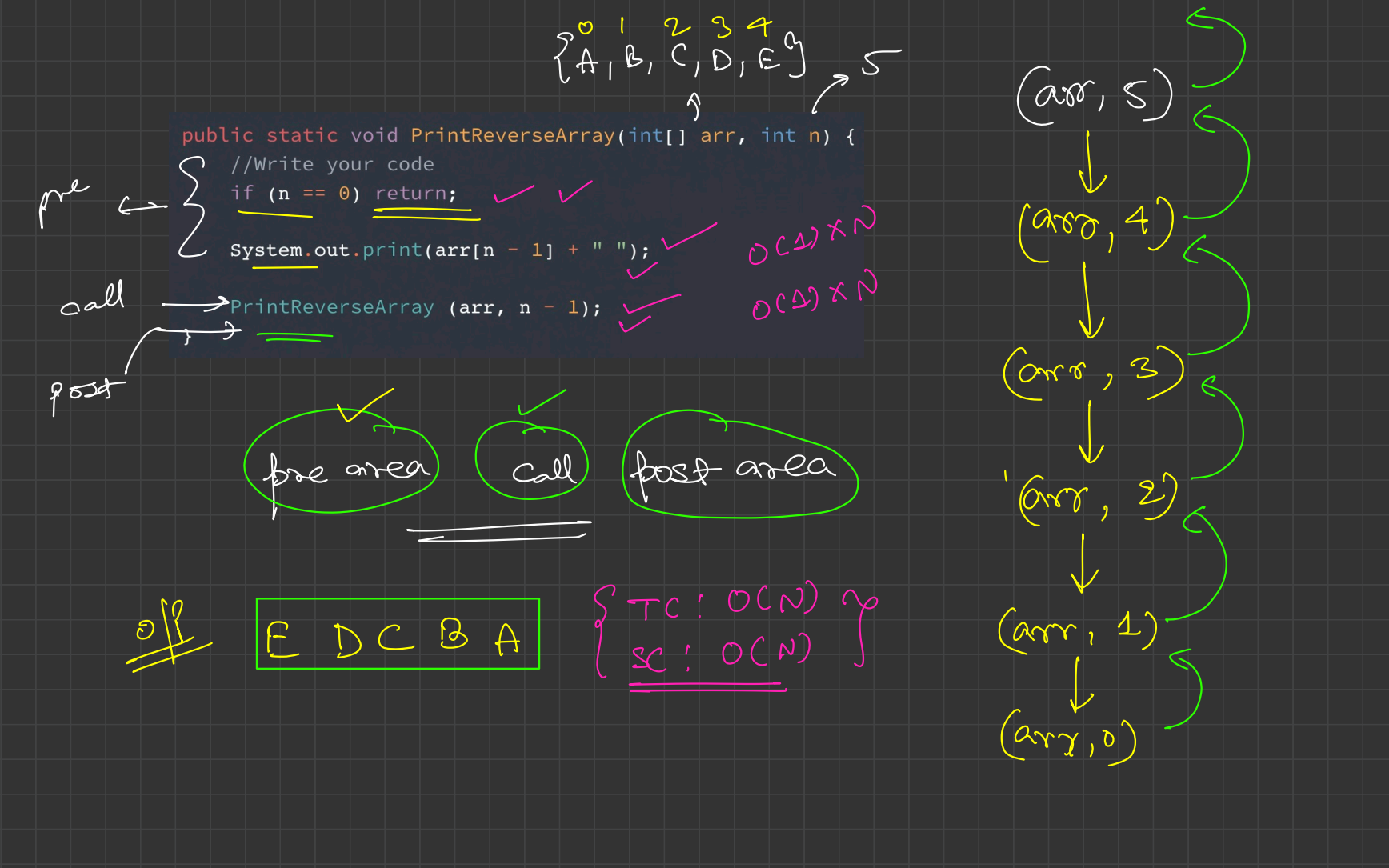
print arr of N size in reverse order.

f(5) : E + f(4)
f(4) : D + f(3)
f(3) : C + f(2)
f(2) : B + f(1)
f(1) : A + f(0)

If (N == 0) return;

Base Case

```
void fun ( arr, N)
{   if (N == =0) return;
    print (arr [N-1]);  ⟶ My work
    fun (arr, N-1);  ⟶ faith
}
```


0 1 2 3 4
{A, B, C, D, E}
5
public static void PrintReverseArray(int[] arr, int n) {
//Write your code
if (n == 0) return;
System.out.print(arr[n - 1] + " ");
PrintReverseArray (arr, n - 1);
pre
call
post
O(1) x N
O(1) x N
pre area
call
post area
O/P
E D C B A
{ TC : O(N)
SC : O(N) }
(arr, 5)
(arr, 4)
(arr, 3)
(arr, 2)
(arr, 1)
(arr, 0)

Q arr = [5, 4, 1, 3, 2] N = 5

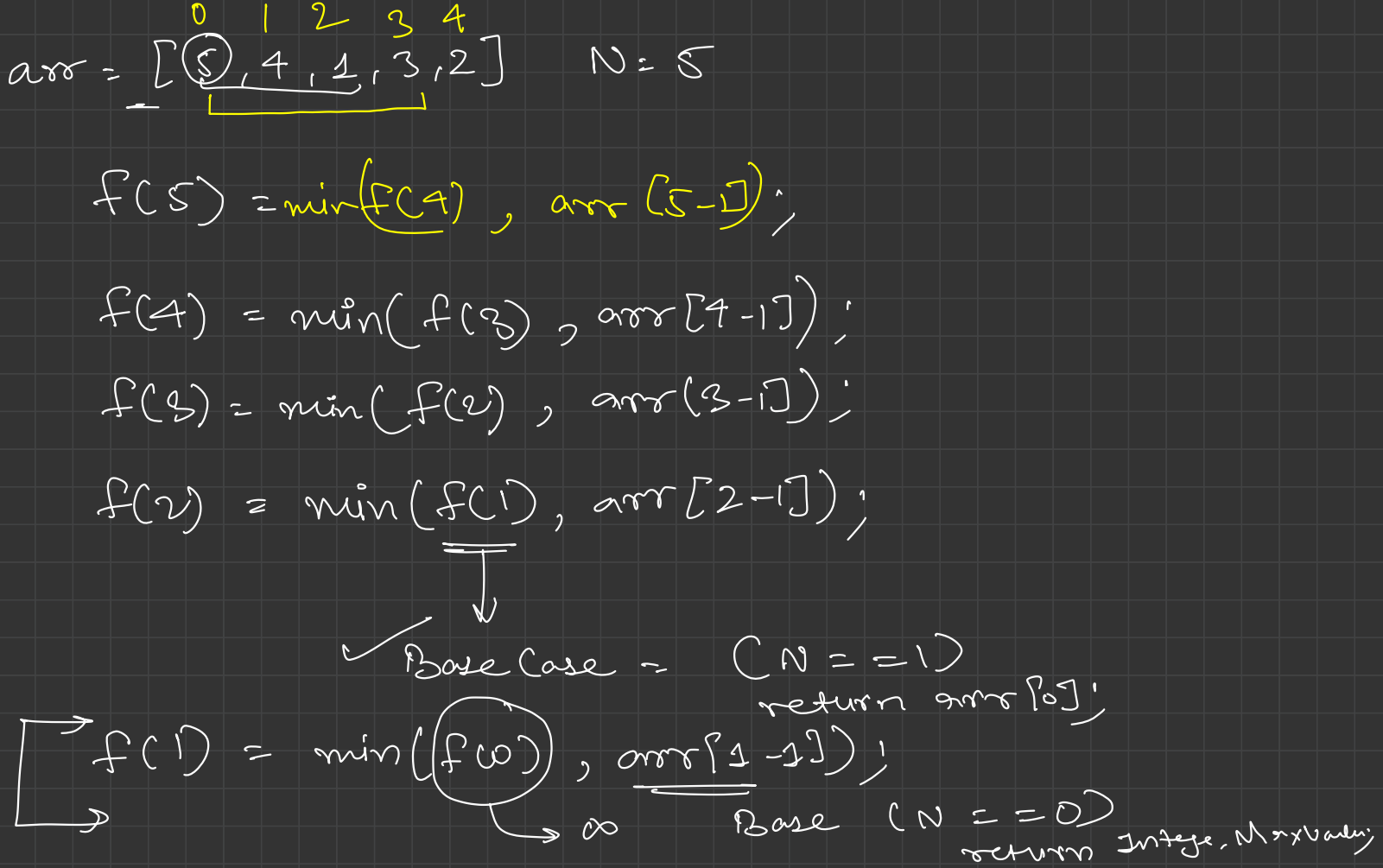
out fun(arr, N) N-1 of 1

→ Smallest Number present in the array.

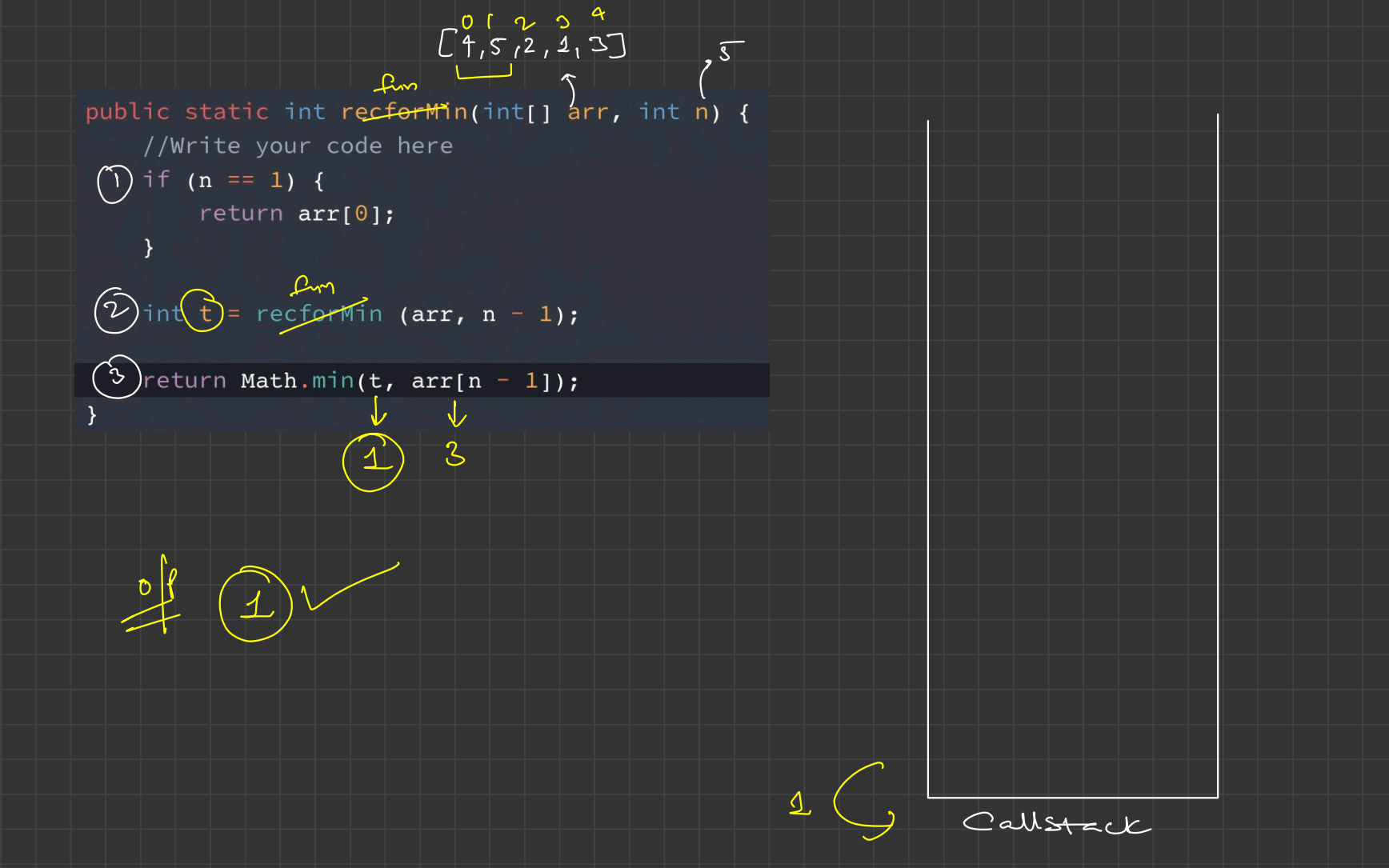
faith : fun()

→ give me smallest number present in arr of size N.

f(5) = min(f(4), arr[5-1])


0 1 2 3 4
arr = [5, 4, 1, 3, 2] N = 5
f(5) = min(f(4), arr[5-1]);
f(4) = min(f(3), arr[4-1]);
f(3) = min(f(2), arr[3-1]);
f(2) = min(f(1), arr[2-1]);
Base Case = (N == 1)
return arr[0];
f(1) = min(f(0), arr[1-1]);
∞
Base (N == 0)
return Integer.MaxValue;

```
int fun ( int [] arr , int N)
{
    if (N == 1)
        return arr[0];

    int t = fun(arr, N-1);

    return Math.min(t, arr[N-1]);

}
```


[4,5,2,1,3]
0 1 2 3 4
fun
public static int recforMin(int[] arr, int n) {
//Write your code here
if (n == 1) {
return arr[0];
}
int t = recforMin (arr, n - 1);
return Math.min(t, arr[n - 1]);
}
1
3
of
1
1
Callstack

[4, 5, 2, 1, 3] (0 1 2 3 4) 5

```java
public static int recforMin(int[] arr, int n) {
    //Write your code here
    if (n == 1) {
        return arr[0];
    }

    int t = recforMin (arr, n - 1);

    return Math.min(t, arr[n - 1]);
}
```

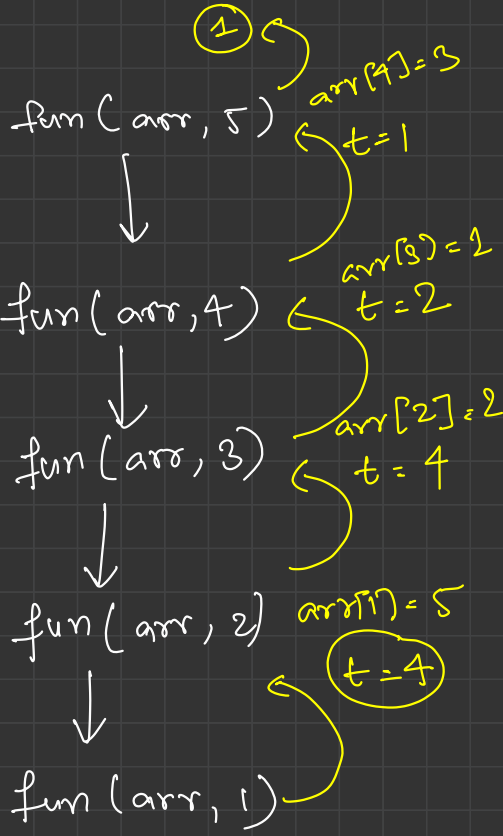
pre
call
post

pre , call , post

O/P

1

TC: O(N)
SC: O(N)



Q

arr = [1, 2, 3, 2, 1]
0 1 2 3 4

0 ↑ , n-1 ↑

0 ↑ start , n-1 ↑ end

boolean isPalindrome (arr , start , end)

Faith: it returns true when a array from given to given end index is a palindrome else it returns false.

f(arr, 0, 4) = check(arr[0] == arr[4]) +
f(arr, 1, 3)

5 = length

f(arr, 1, 3) = check(arr[1] == arr[3]) + f(arr, 0, 0)

Base Case
if(start == end)
return true

f(arr, 0, 5) = check(arr[0]==arr[5]) + f(arr, 1, 4)

↳ length: 6

f(arr, 1, 4) = check(arr[1]==arr[4]) + f(arr, 2, 3)

f(arr, 2, 3) = check(arr[2]==arr[3]) + f(arr, 3, 2)

↓

start > end

Base Case

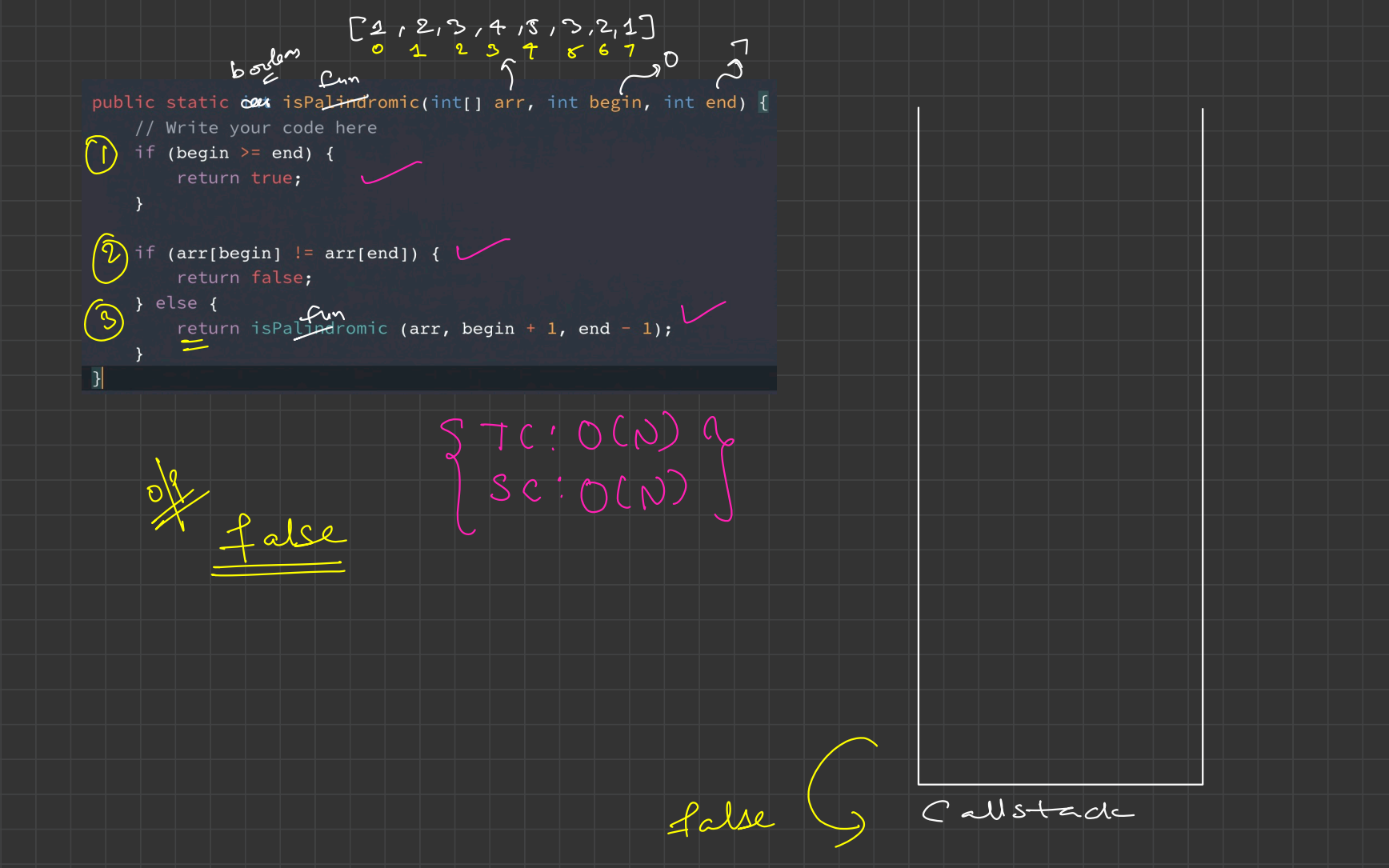
if (start > end)

return true

```
boolean fun(arr, start, end)
{
    if (start >= end)
        return true;

    if (arr[start] != arr[end])
        return false;

    else
        return fun(arr, start+1, end-1);
}
```

I'm not good enough

I'm good enough


[1, 2, 3, 4, 5, 3, 2, 1]
0 1 2 3 4 5 6 7
boolean
fun
public static boolean isPalindromic(int[] arr, int begin, int end) {
// Write your code here
if (begin >= end) {
return true;
}
if (arr[begin] != arr[end]) {
return false;
} else {
return isPalindromic (arr, begin + 1, end - 1);
}
}
1
2
3
fun
TC: O(N)
SC: O(N)
false
false
Callstack

Q arr[]: [5, 6, 4, 6, 4, 1, 2] N = 7 T = 4

0 1 2 3 4 5 6

fun(arr, T, startIndex)

```
printfun(arr, T, startIndex)
{
    if (startIndex == arr.length)
        return;
    print(arr[startIndex]);
    printfun(arr, T, startIndex + 1);
}
```

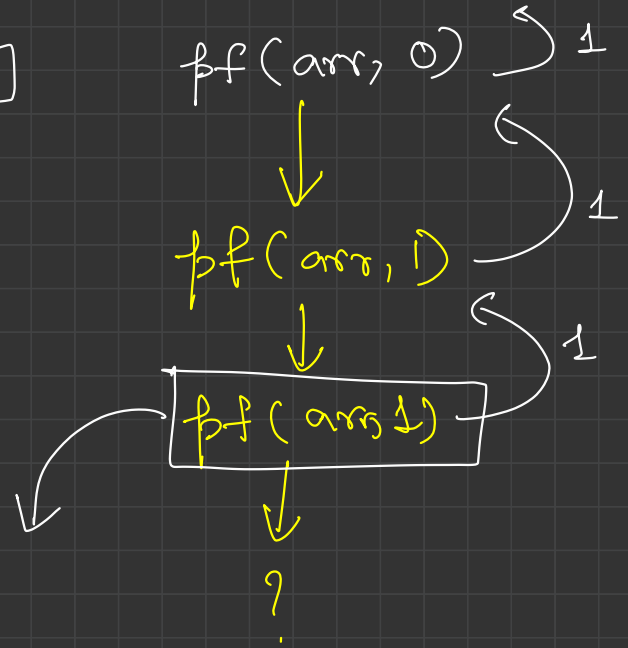
Faith

o/p

5
6
4
6
4
1
2

arr: [5, 3, 4, 4, 1]

T = 4



N = 5

0	5
1	3
2	4

Base case

pf(arr[si] == T)

return 1;

```
int firstOcc(arr, T, si)
{
    if (si == arr.length)
        return -1;
    if (arr[si] == T)
    {
        return si;
    }
    else
    {
        return firstOcc(arr, T, si+1);
    }
}
```

arr [1, 2, 3, 4, 4, 5, 5, 4]
0 1 2 3 4 5 6 7

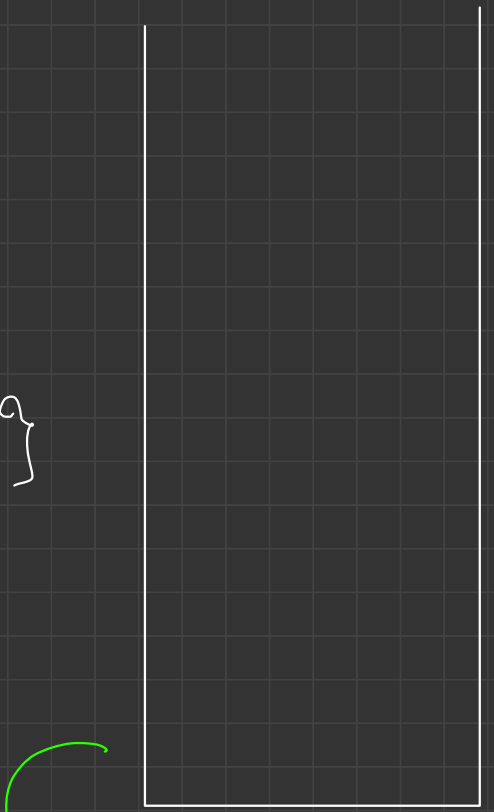
```java
static int firstIndex(int A[],int T,int startIndex)
{
    //Write your code here
    if (startIndex == A.length) {
        return -1;
    }

    if (A[startIndex] == T) {
        return startIndex;
    } else {
        return firstIndex (A, T, startIndex + 1);
    }
}
```

startIndex → 0

1
2
3

{ TC: O(N)
SC: O(N) }

o/p: 5th index



Q

```
      0  1  2  3  4  5  6  7  8
arr = [3, 3, 2, 3, 5, 5, 4, 1, 1]      T = 3      ei = 8
```

```
int lastIndex(int[] arr, int T, int ei)
{
    if (ei == -1)
        return -1;

    if (arr[ei] == T)
    {
        return ei;
    }
    else
    {
        lastIndex(arr, T, ei-1);
    }
}
```

Q

find first occurance?

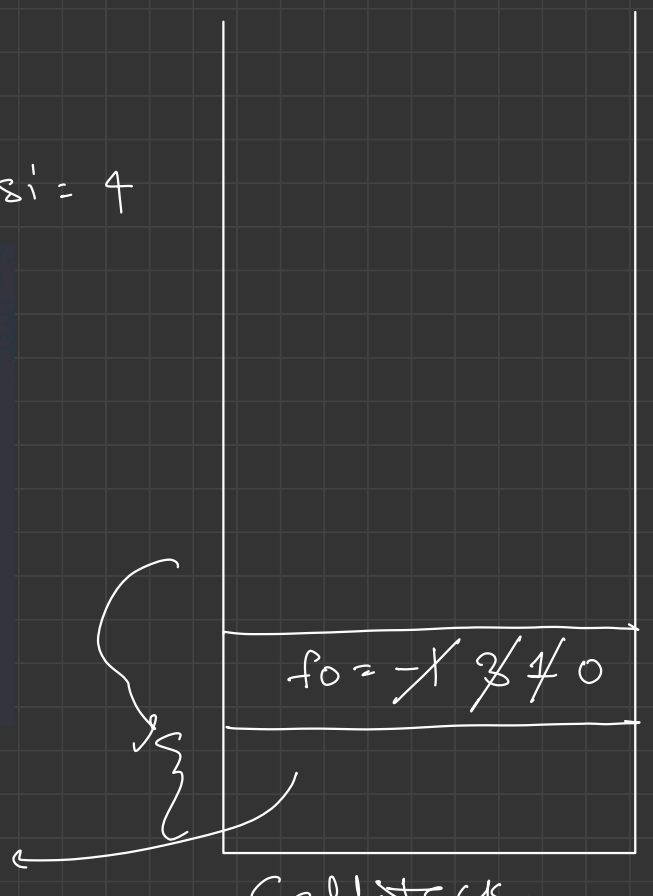
fun(arr, T, ei)

arr:-[3,3,2,3,1], T=3, si = 4

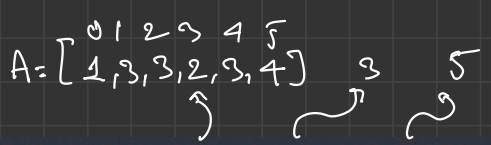
n-1

```
static int fo = -1;
static void firstIndex(int A[],int T,int startIndex)
{
    //Write your code here
    if (startIndex == -1) {
        return;
    }

    if (A[startIndex] == T) {
        fo = startIndex;
    }

    firstIndex(A, T, startIndex - 1);
}
```

① ② ③



fo = 0




```
static int firstIndex(int A[],int T,int startIndex)
{
    //Write your code here
    if (startIndex == -1) {
        return -1;
    }

    int ans = firstIndex(A, T, startIndex - 1);

    if (ans != -1) {
        return ans;
    } else {
        if (A[startIndex] == T) {
            return startIndex;
        } else {
            return -1;
        }
    }
}
```