Recursion Doubts with Lakshay [Extra Class]

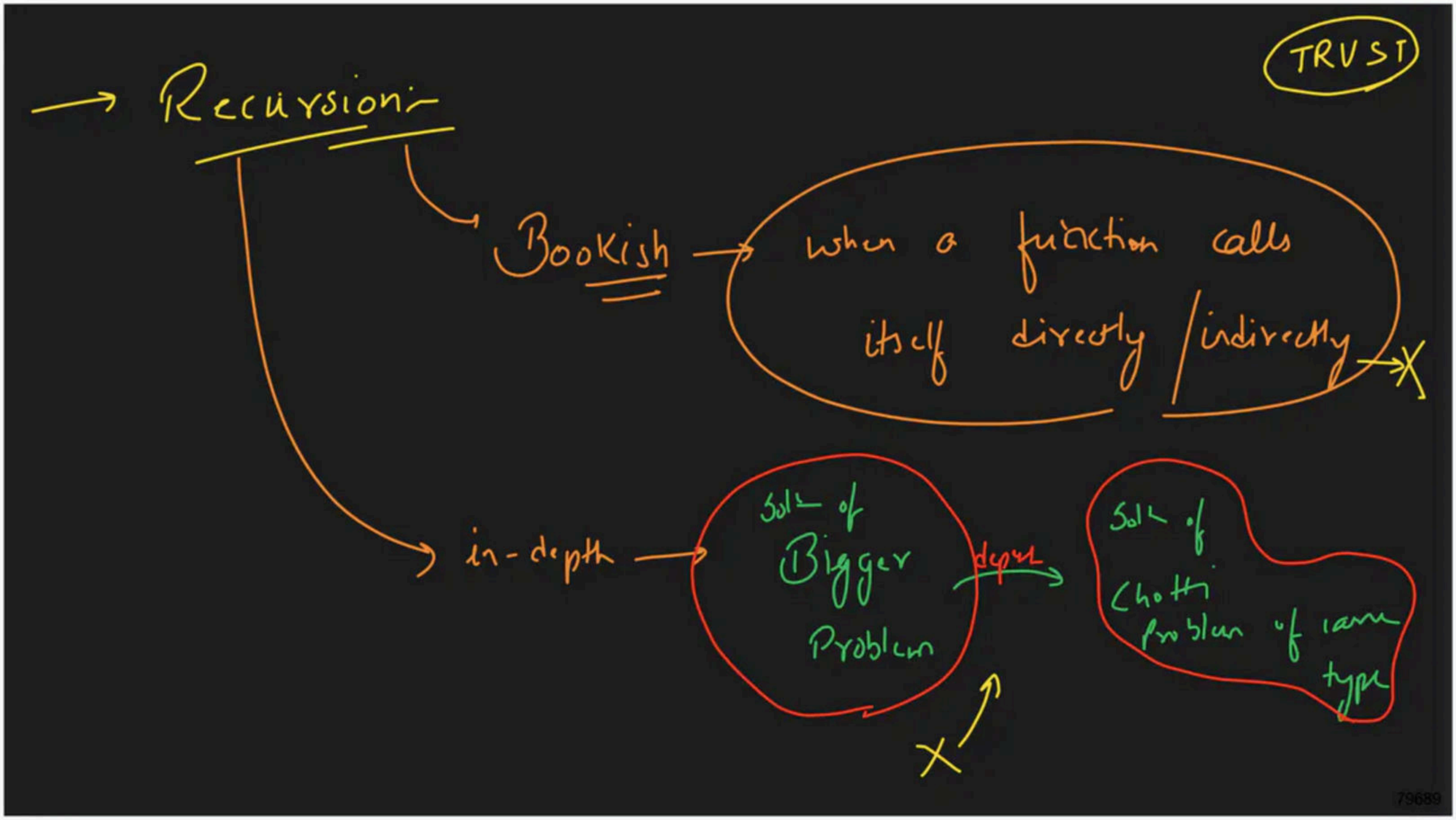
Special class

Love Babbar • Oct 19, 2023

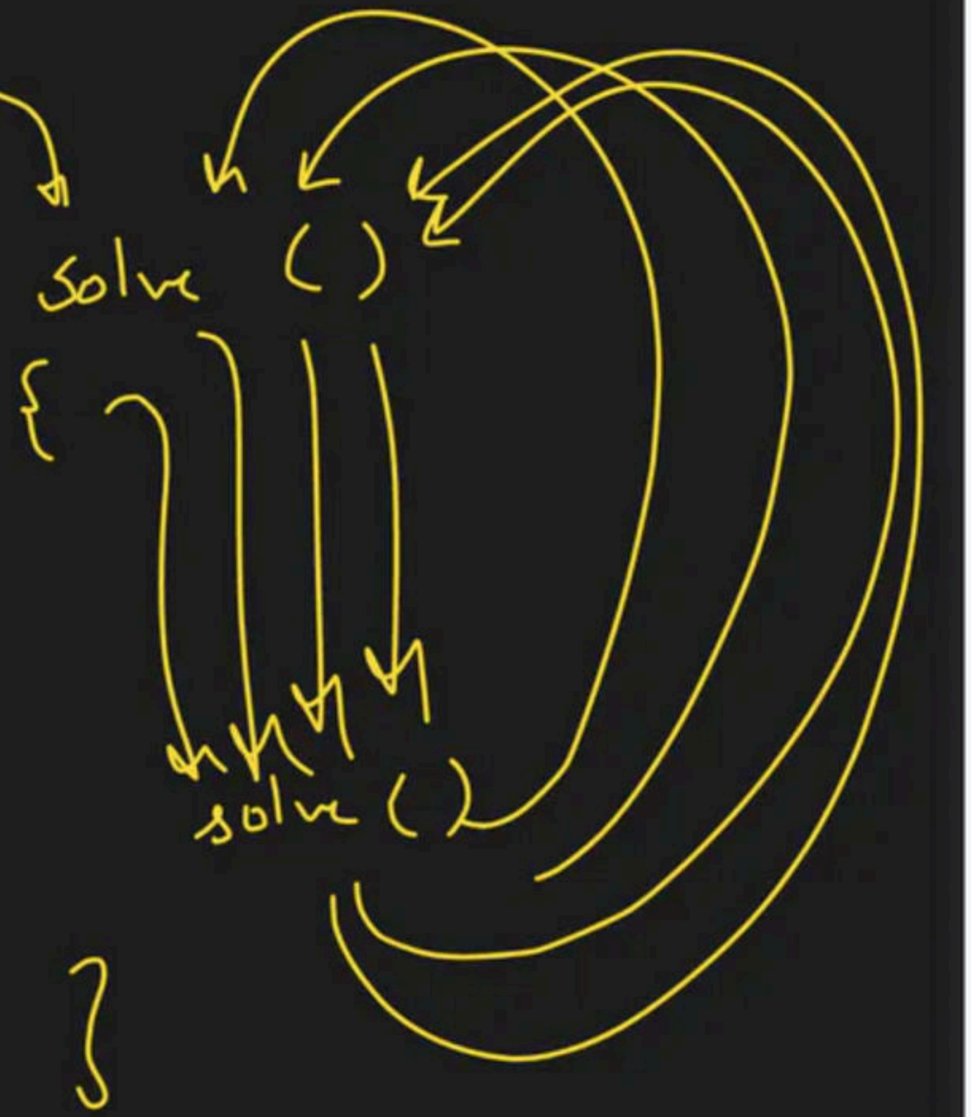
Recursion - Class 1

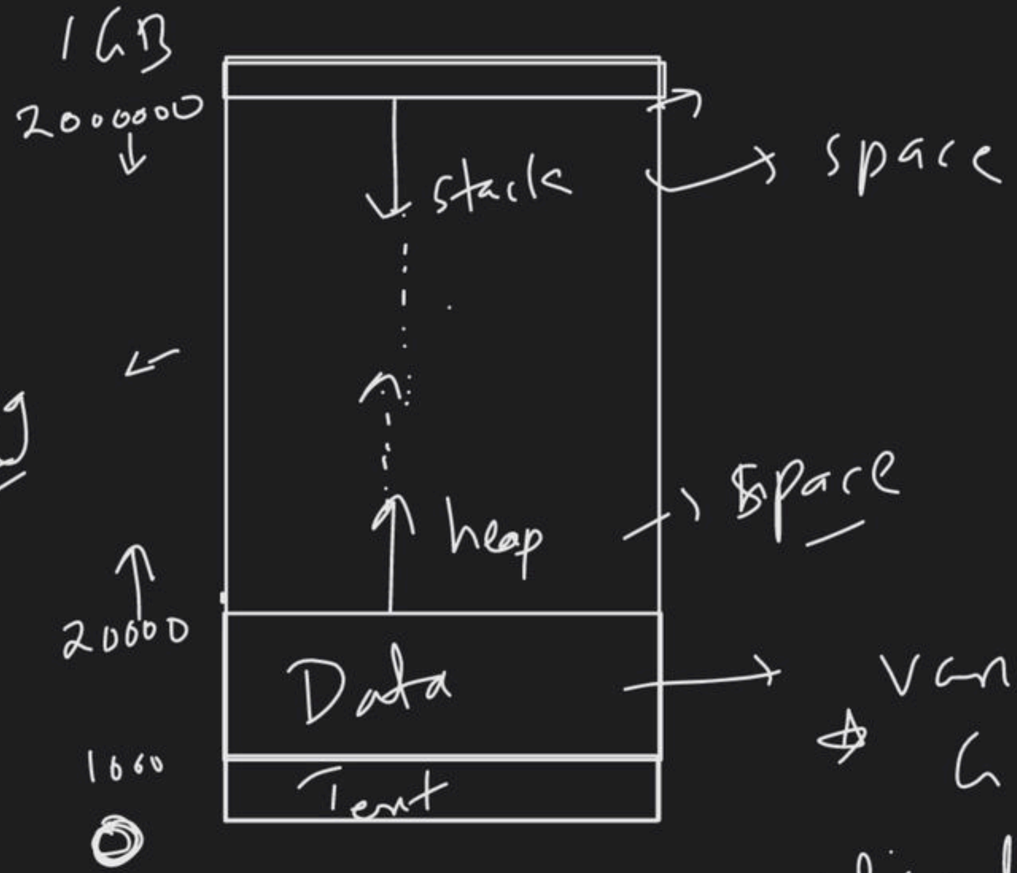
Special class

Love Babbar • Oct 9, 2023


TRUST
Recursion
Bookish
when a function calls itself directly / indirectly
X
in-depth
Sol^n of Bigger Problem
depend
Sol^n of Chotti problem of same type
X

Recursion :-





pubg

variable
Global
static => local

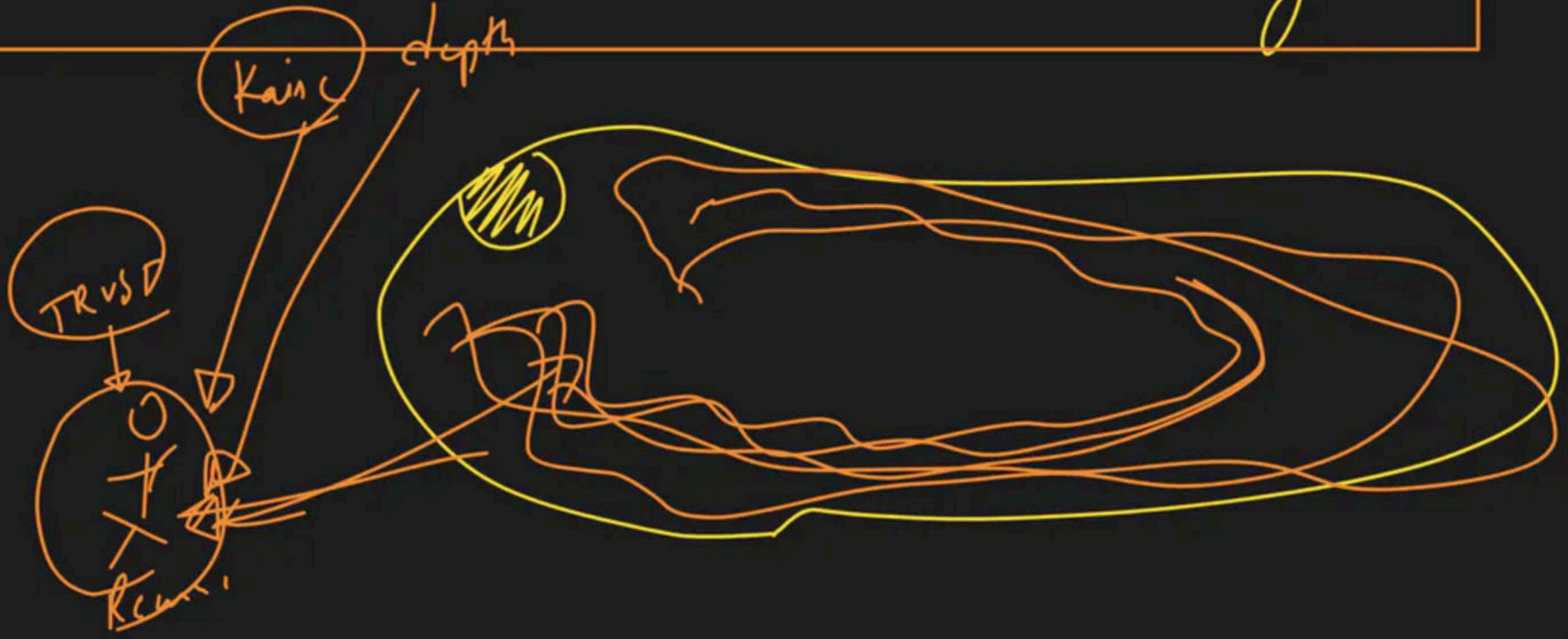
OS → Process

By default

8KB

8MB

→ 1 case hum solve karo, baaki recursion sambhal lega



```
① fun ( int a )
{ if( a >1000) return;
  vector <int>V= {1,2,3,4,5};
  fun (++a);
}
```

5000

5000 → 50001

Stack

⑤	24 B =	
④	fun( )	24 B
③	fun( )	24 B
②	fun( )	→ 24 byte
①	fun( )	→ 24 byt

5×4

(a)

main allowed → [24 × 4] ★★

stack size By OS

9 GB ★△

★ seg. fault

stack overflow

T.n§ Complexity

① O(n)

② $O(2^n)$ → exponential.

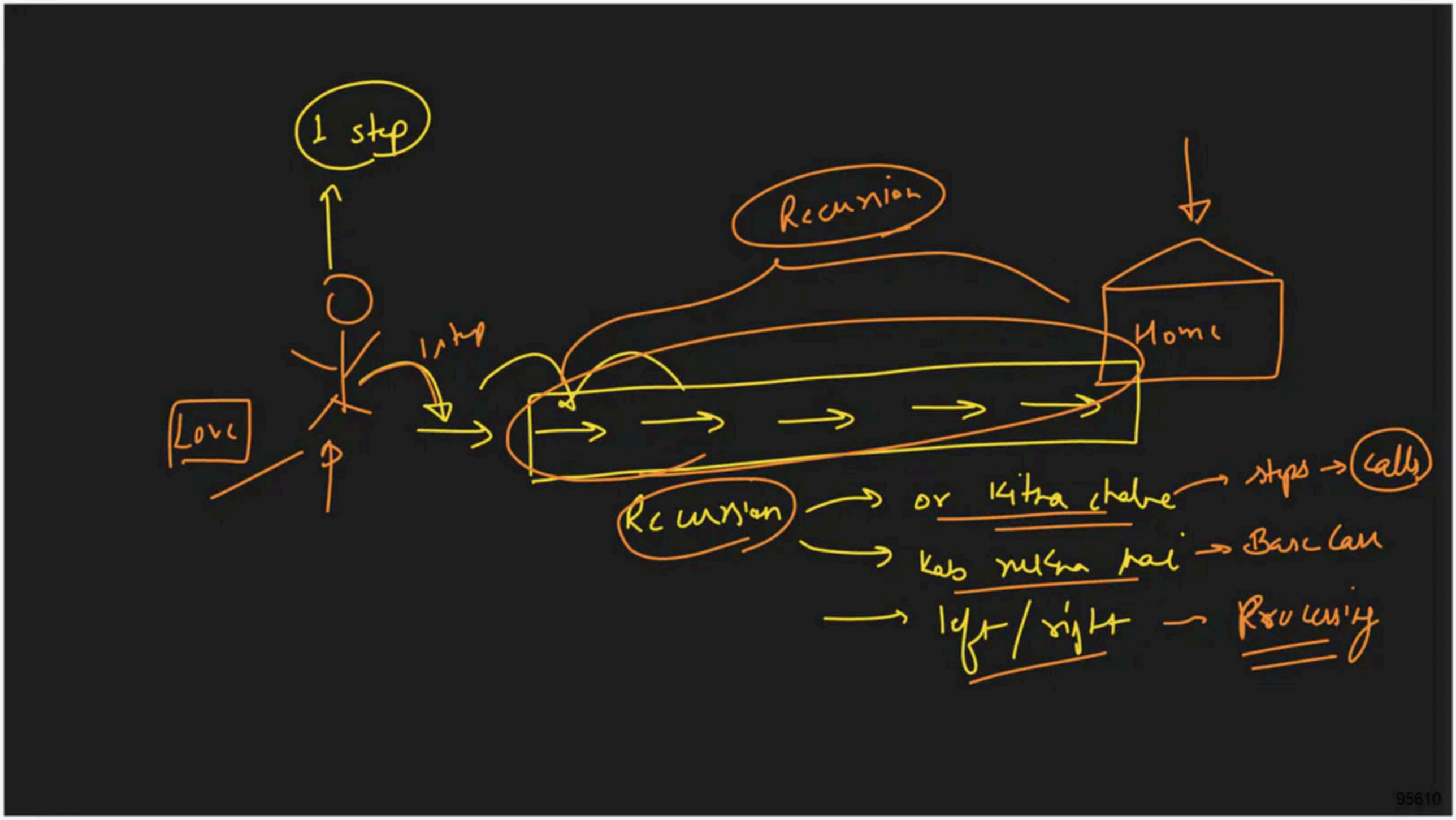
↳ Bakwas ☆△

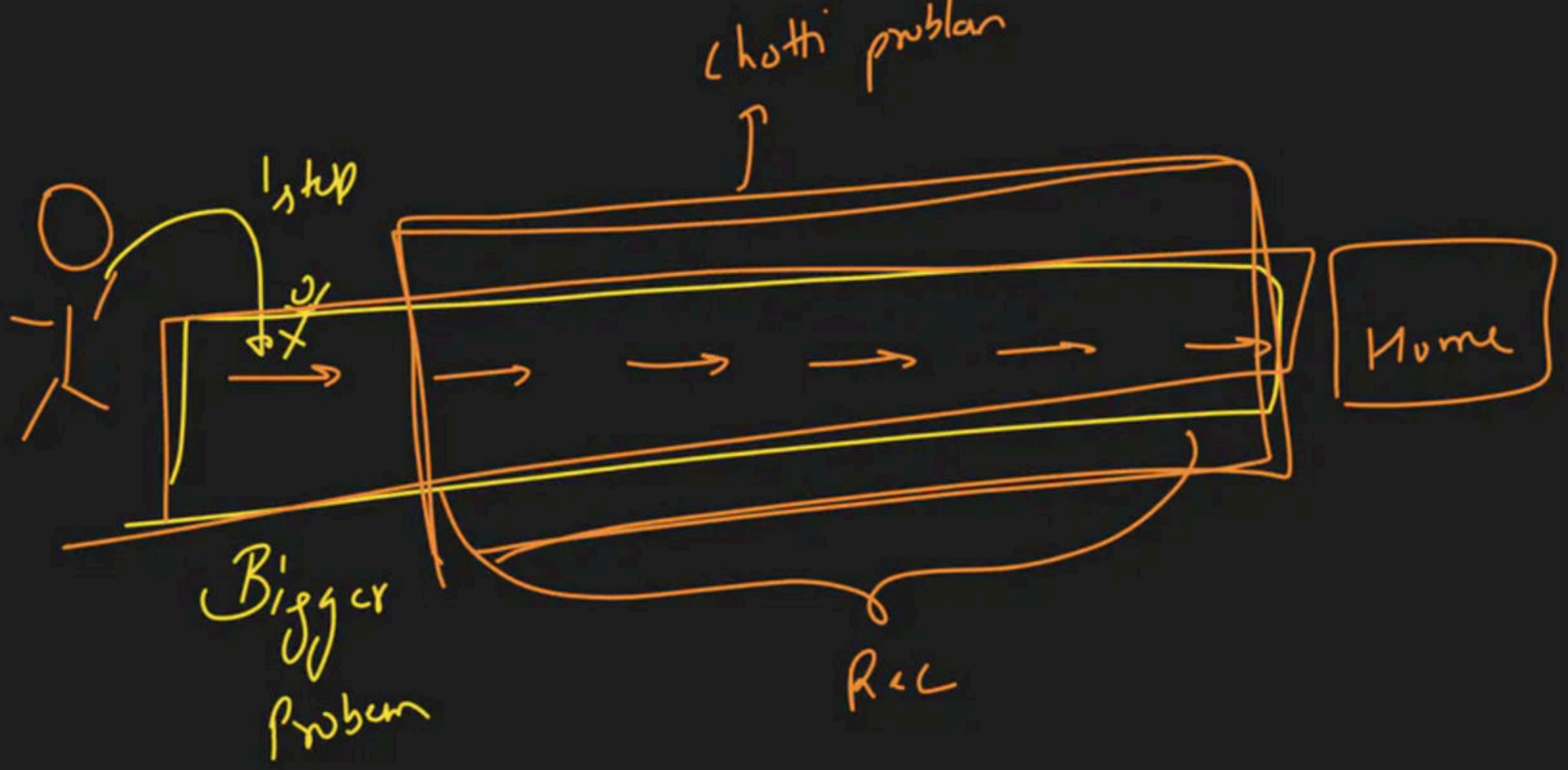
③ O(logn)

```
fun()
{
  a = fun()
  b = fun()
} c = fun()
  d = fun()
  return a+b+c+d;
}
```

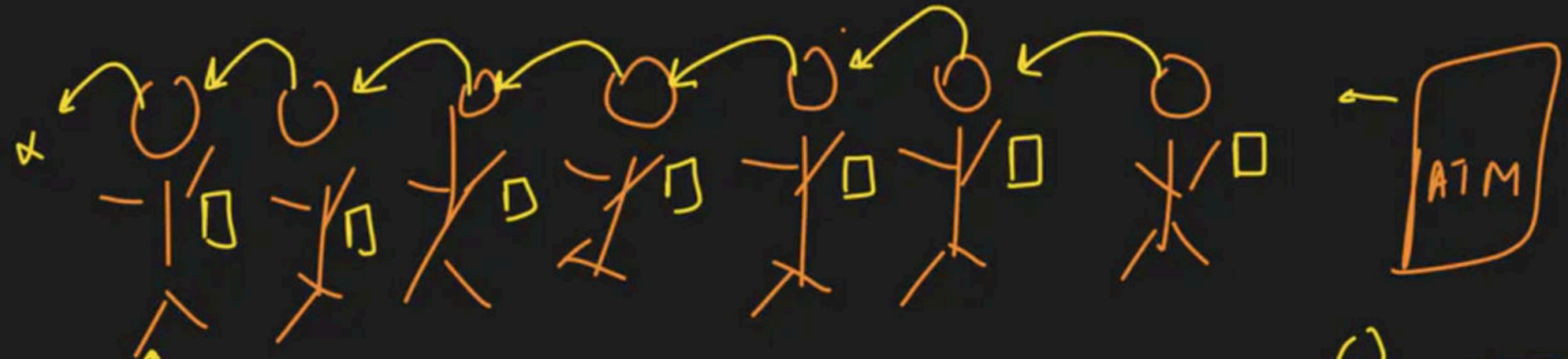
RE → Space Comp. ≡ no. of stack Blocks Taken

CdUs


1 step
Recursion
Home
Love
1 step
Recursion
or kitna chalne → steps → calls
kab rukna hai → Base Case
left / right → Recursion


Chotti problem
1 step
Home
Bigger Problem
R<L

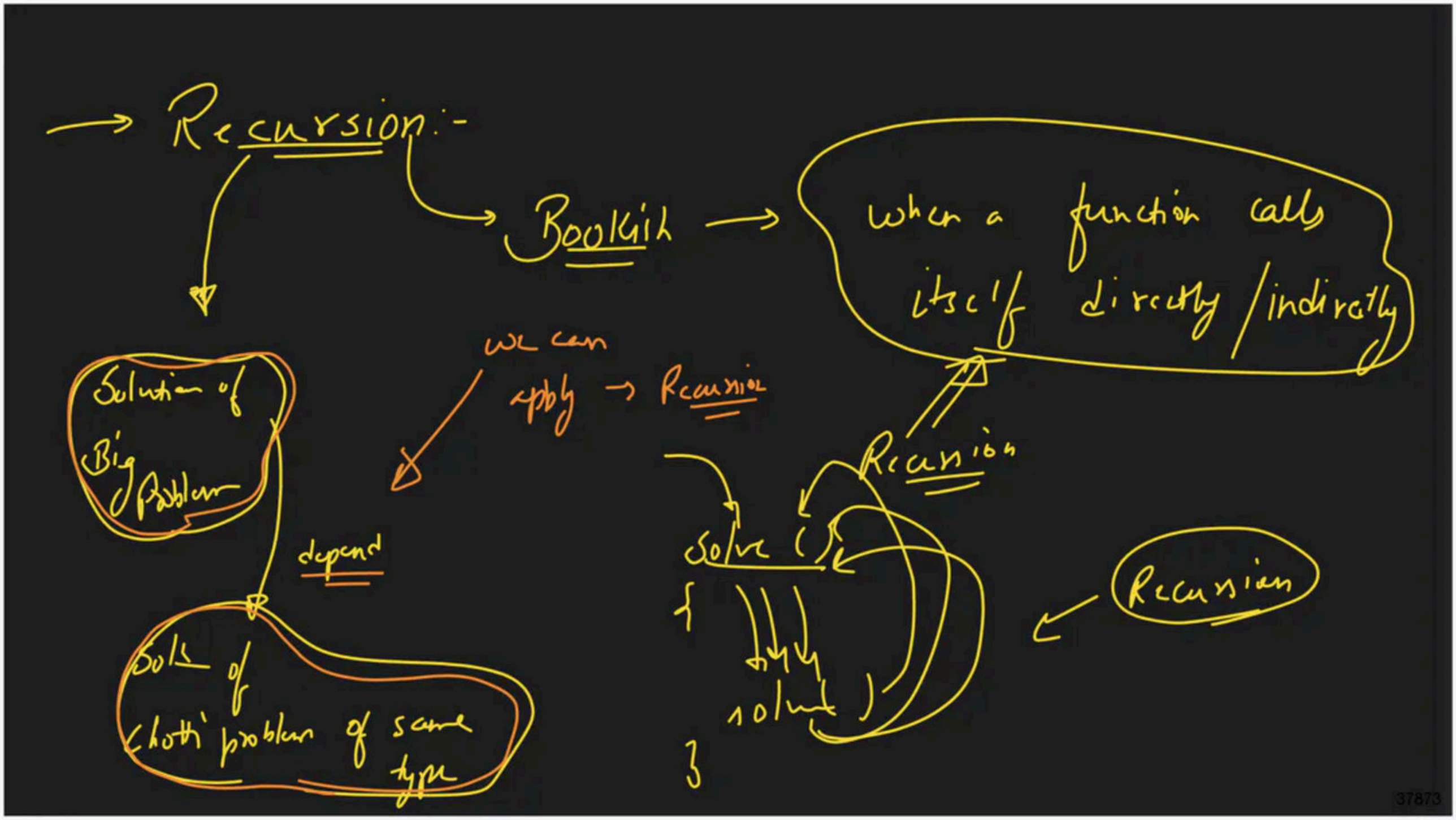
→ To understand Recursion, you need to first understand Recursion

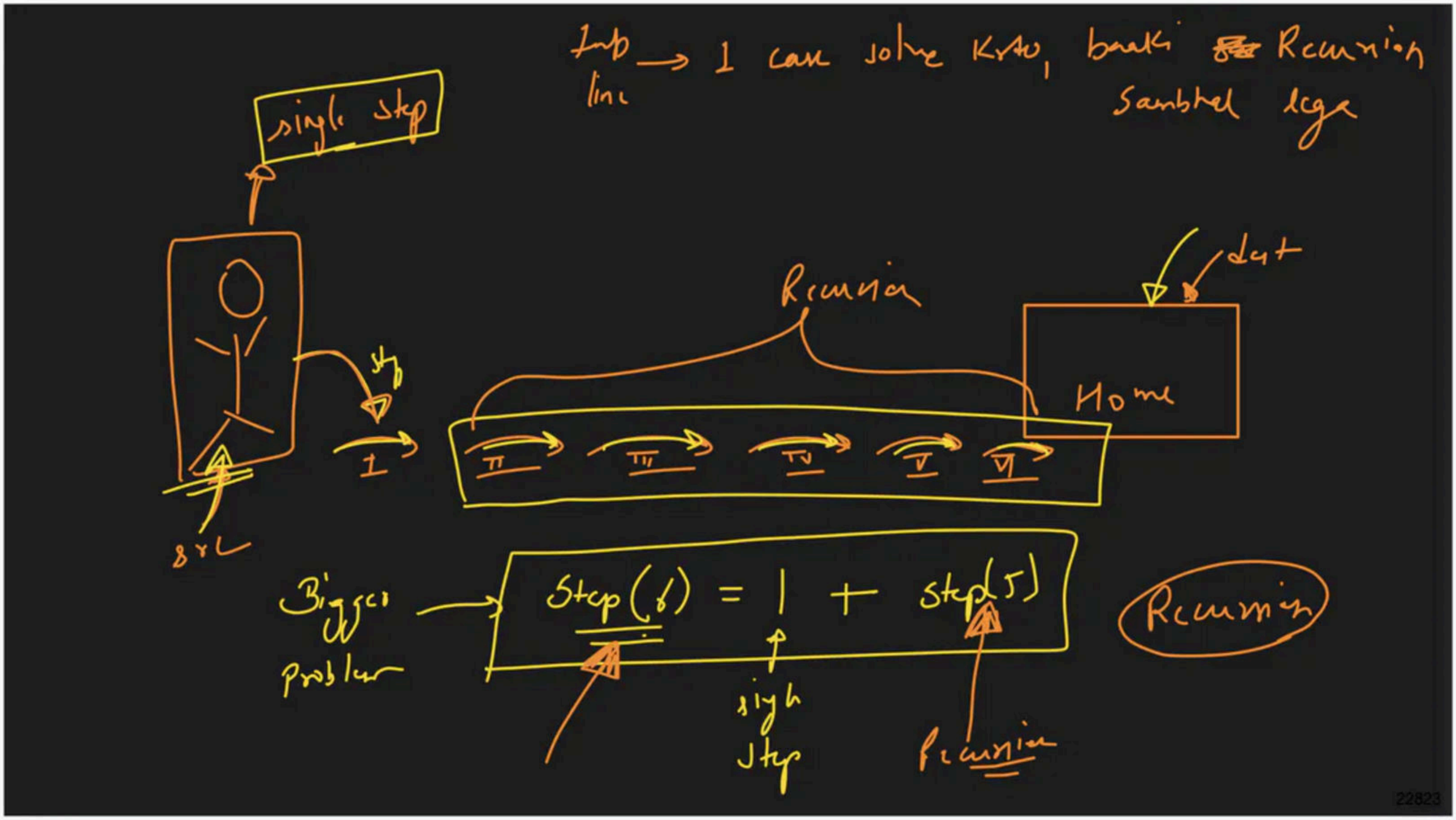


nuk gyz

Ruk → (A) pichi koi vyakti pass paper

(B) agar kui nahi khada pichi, nuk gyo


Recursion :-
Bookish
when a function calls itself directly / indirectly
we can apply → Recursion
Solution of Big Problem
depend
Sols of thoda problem of same type
Recursion
solve ()
{
solve ()
}
Recursion


single step
Recursion
Home
Step(6) = 1 + step[5]
Recursion

~~f(n)~~

solve(n) ⟶ 2^n

solve(n-1) = 2^{n-1}

Bigger Problem → 2^n

solve(n) = 2^n

solve(n) = $2 \times 2^{n-1}$

solve(n) = 2 × solve(n-1)

Big

Chotti

Solu(n) ⟶ (n ⟶ 1) counting print

solu(n-1) ⟶ (n-1 ⟶ 1) counting

solu(n) ⇒ n, n-1, n-2, ----, 1

solu(n) = n, (n-1, n-2, ---- 1)

Solu(n) = n, solu(n-1)

Factorial

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

soln(5) = 5!

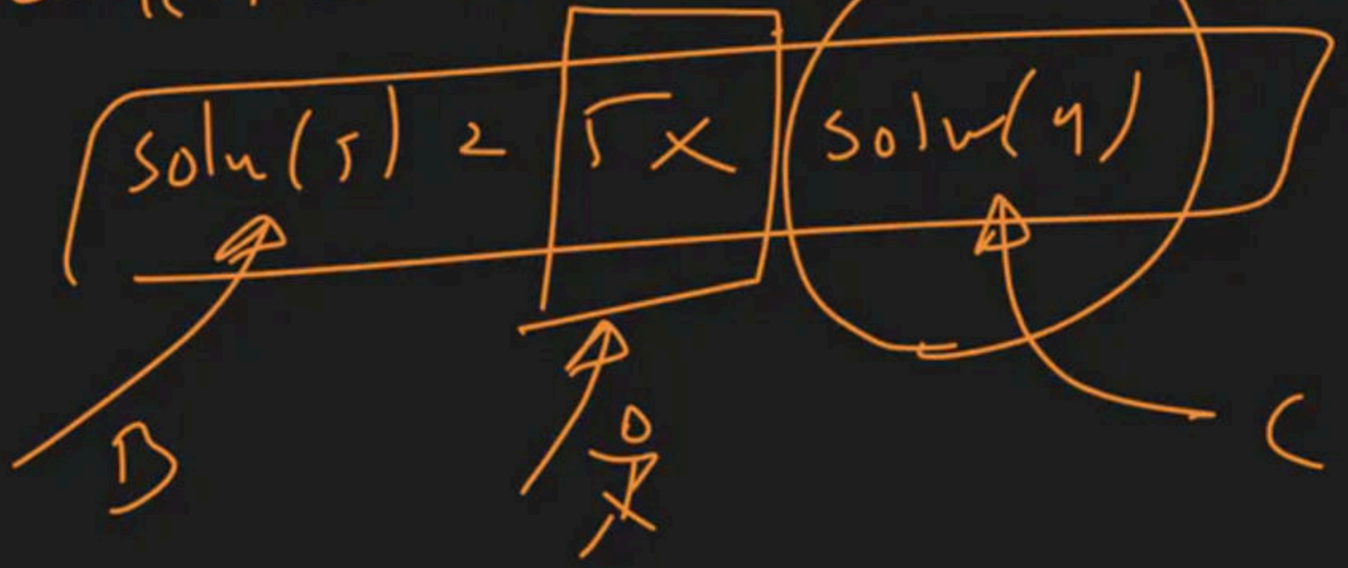
soln(5) = 5 × 4 × 3 × 2 × 1

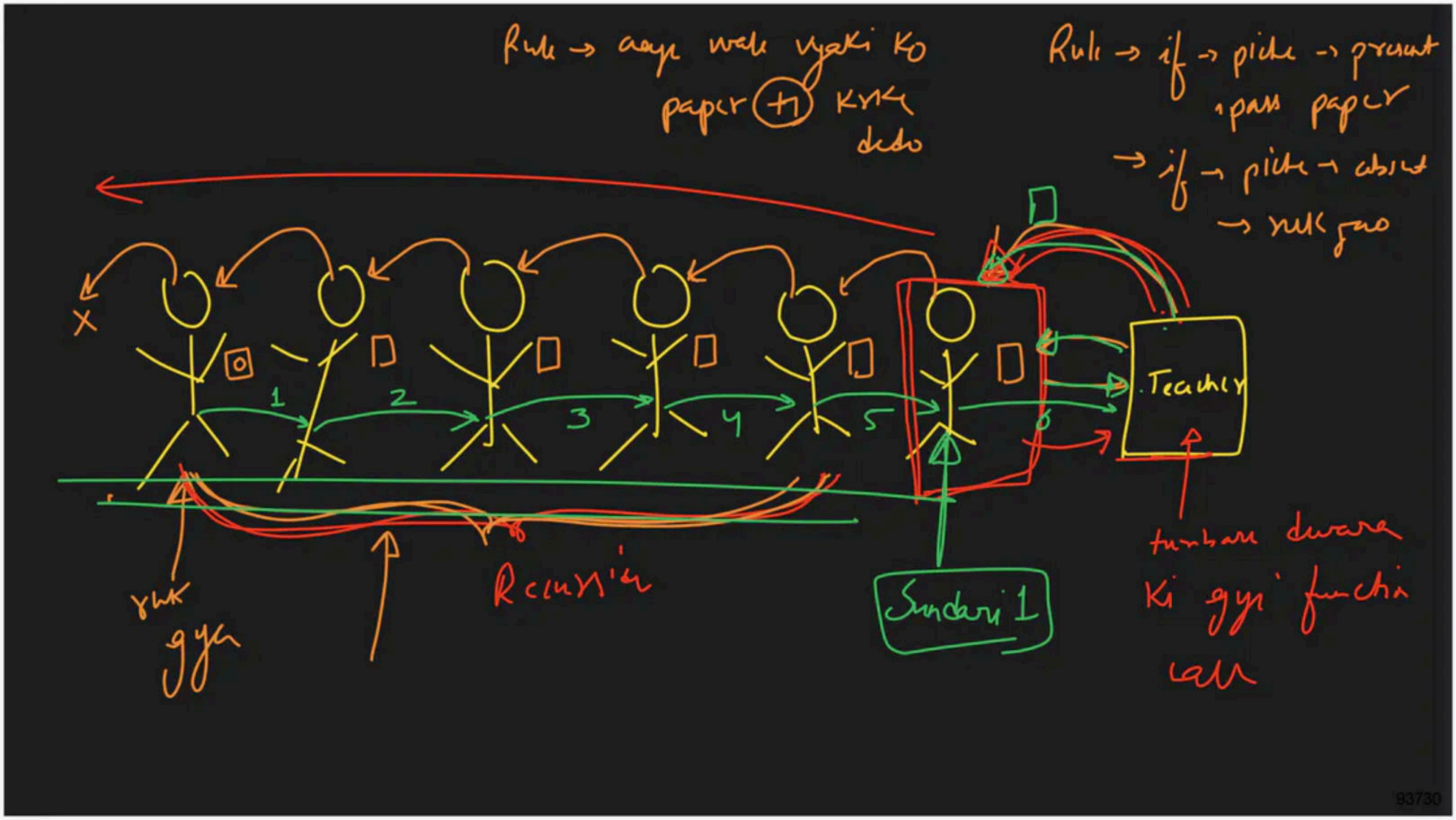
soln(4) = 4!

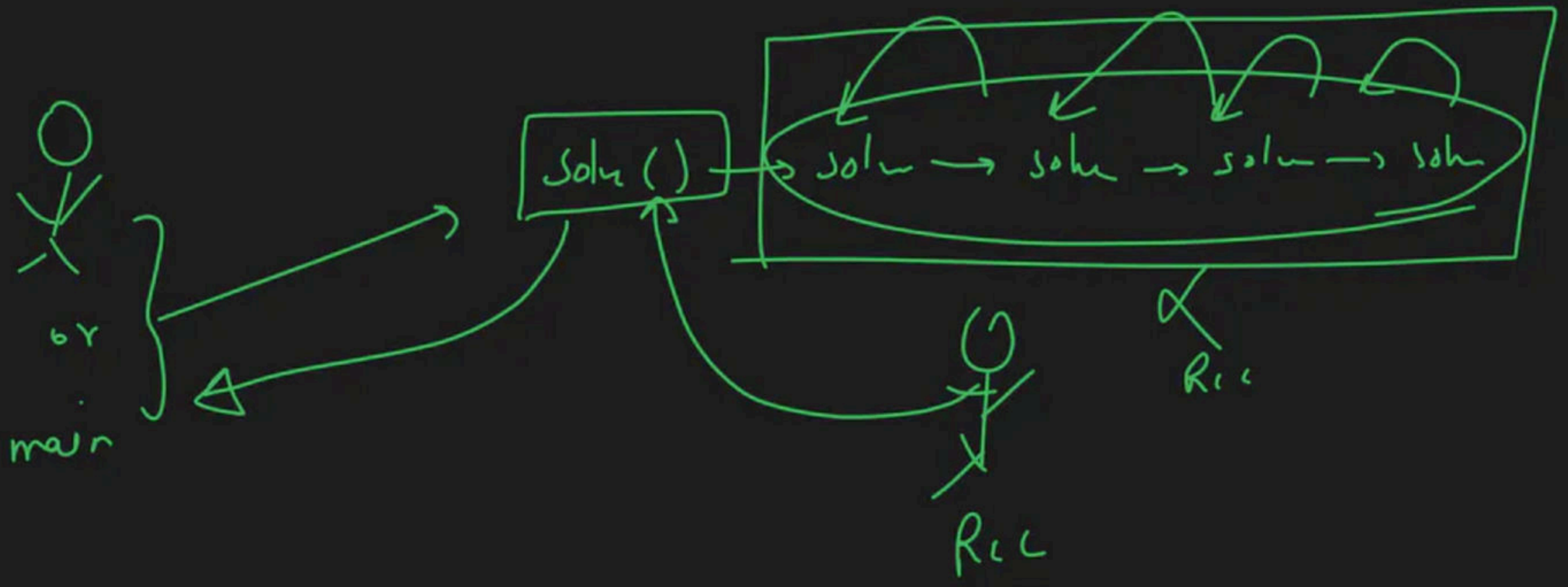
= 4 × 3 × 2 × 1

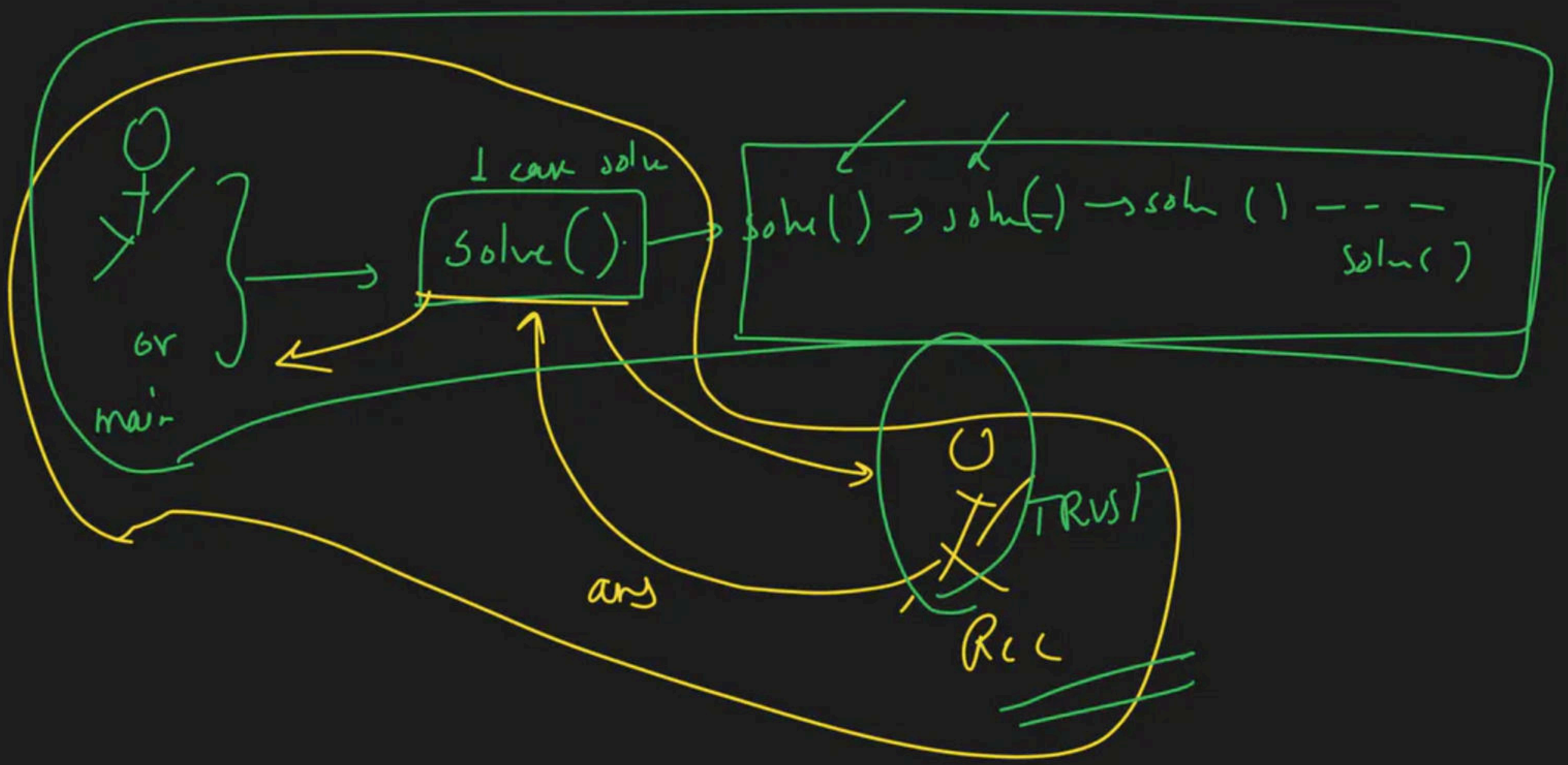
soln(5) = 5 × soln(4)

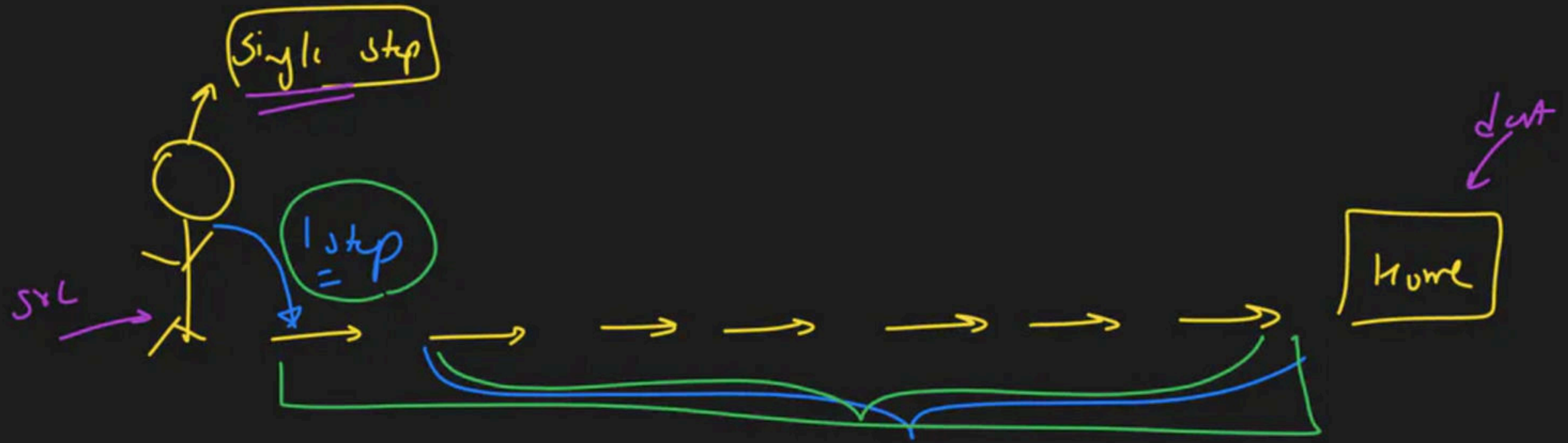
→ Recursion




Rule → aage wale vyakti ko paper (+1) krke dedo
Rule → if → piche → present → pass paper
→ if → piche → absent → null pass
X
1
2
3
4
5
6
Teacher
Recursion
Sundari 1
gya
tumhare dwara ki gyi function call


Solu ( )
Solu → Solu → Solu → Solu
or
main
Rec
Rec

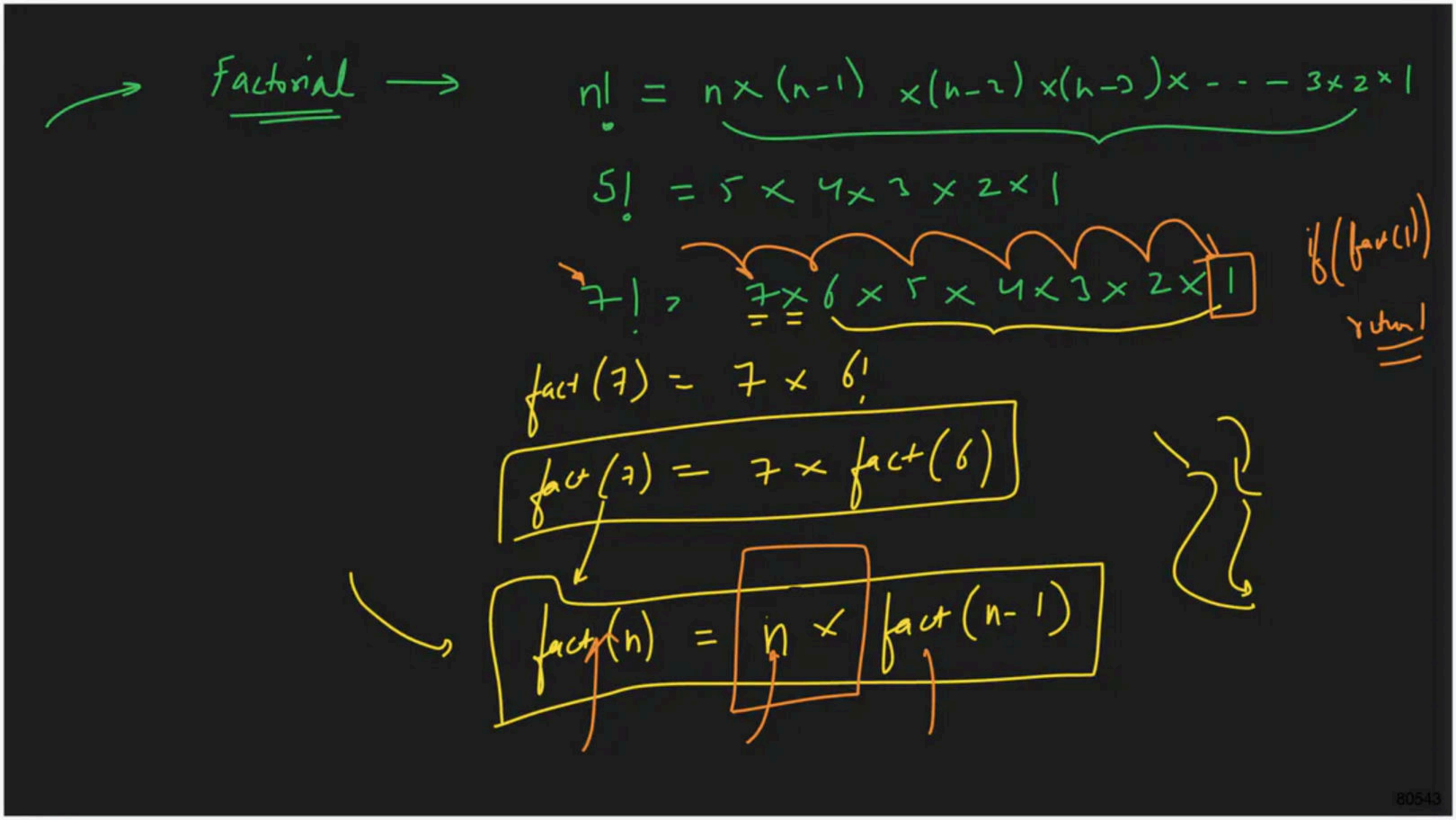

I can solve
Solve()
solve() → solve() → solve() - - -
solve()
or
main
ans
TRUST
Rec

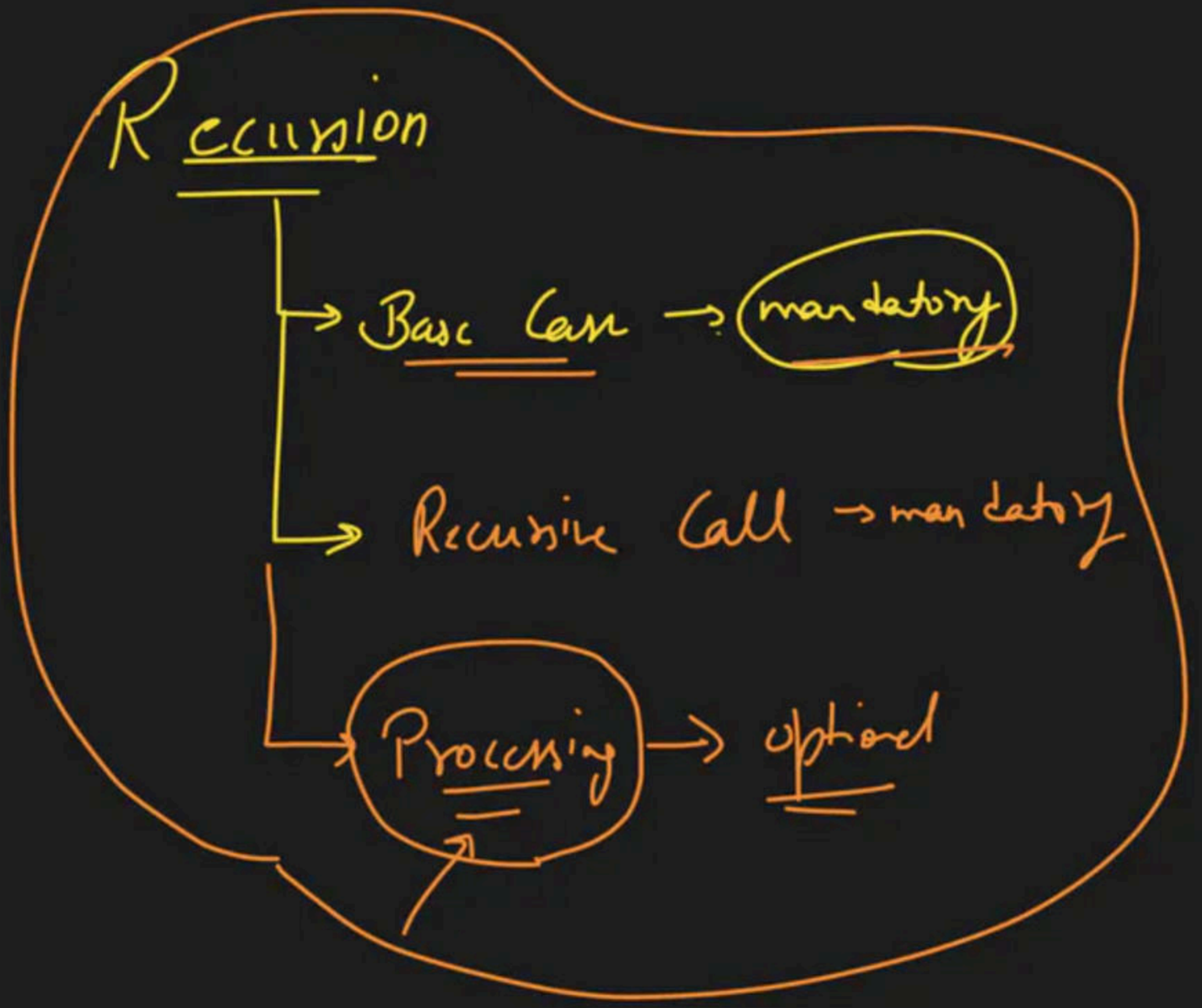


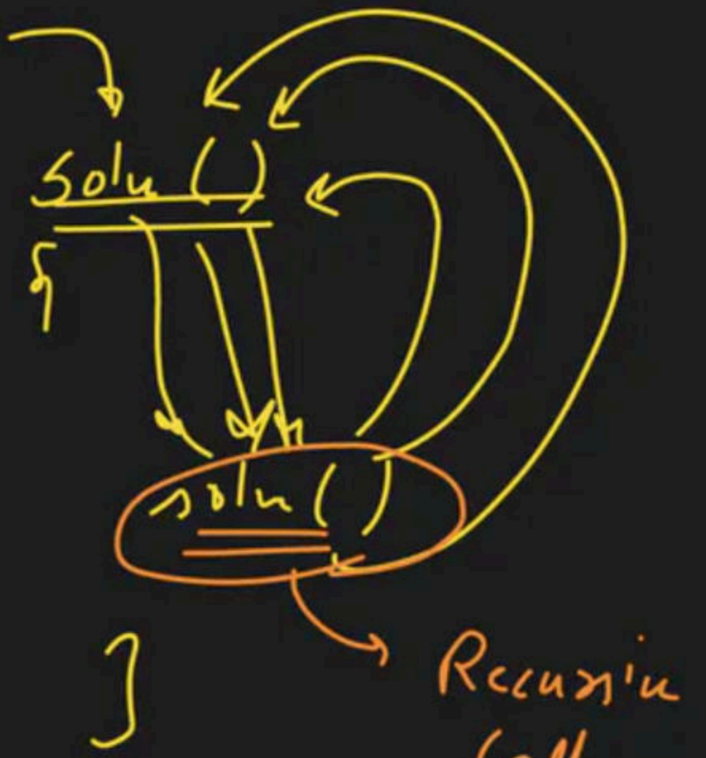
Recursion

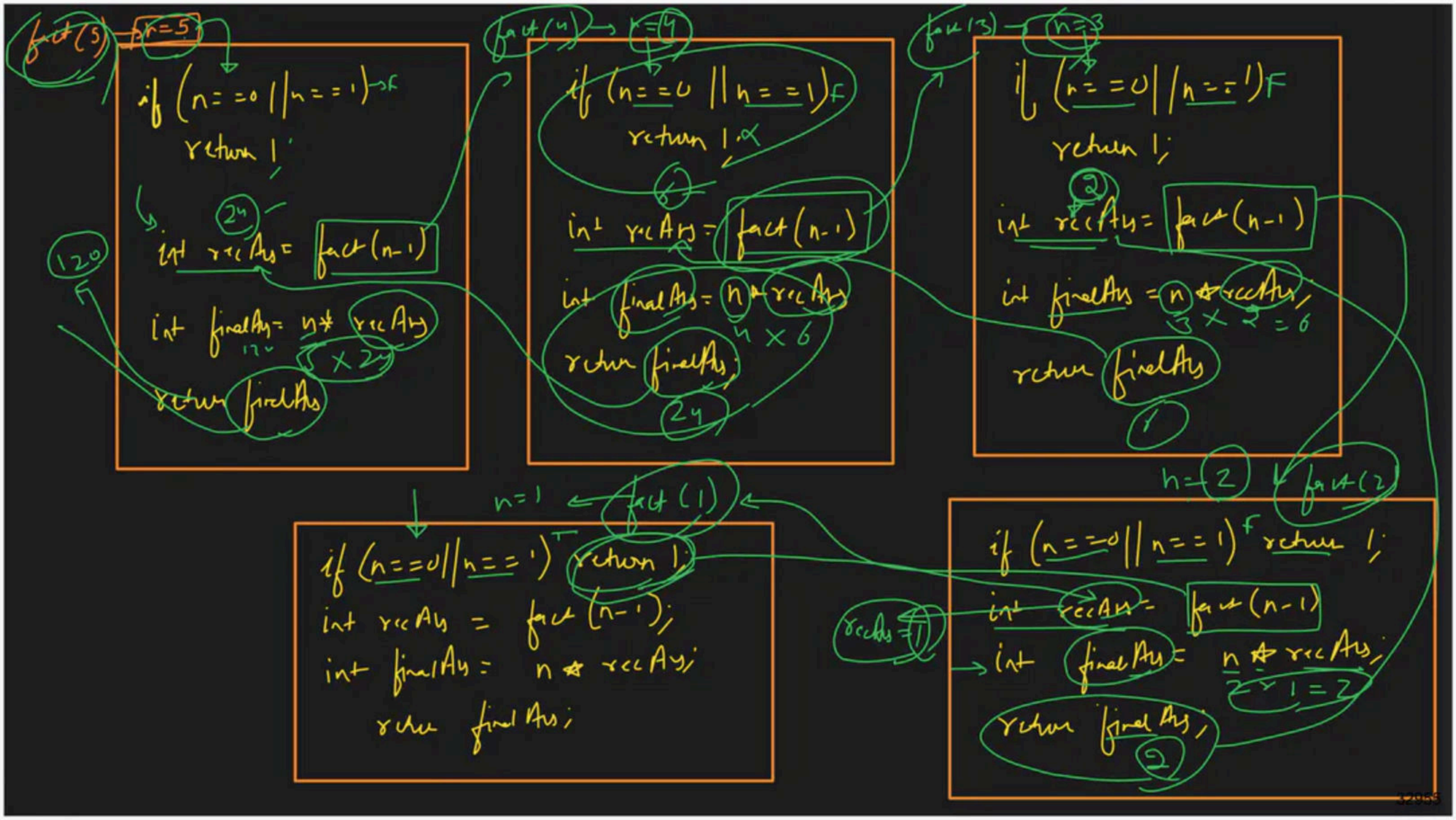
Overall = 1 + Recursion
Ans Ans

Big 2 1 + Cloth


Factorial
n! = n x (n-1) x (n-2) x (n-3) x - - - - 3 x 2 x 1
5! = 5 x 4 x 3 x 2 x 1
7! = 7 x 6 x 5 x 4 x 3 x 2 x 1
if (fact(1)) return 1
fact(7) = 7 x 6!
fact(7) = 7 x fact(6)
fact(n) = n x fact(n-1)


Recursion
Base Case → mandatory
Recursive Call → mandatory
Processing → optional


solu()
{
solu()
}
Recursion
Call
relati


fact(5)
n=5
if (n==0 || n==1) return 1;
int recAns = fact(n-1)
int finalAns = n * recAns
return finalAns
24
120
fact(4)
n=4
if (n==0 || n==1) return 1;
int recAns = fact(n-1)
int finalAns = n * recAns
4 x 6
return finalAns
24
fact(3)
n=3
if (n==0 || n==1) return 1;
int recAns = fact(n-1)
int finalAns = n * recAns
3 x 2 = 6
return finalAns
2
1
n=2
fact(2)
if (n==0 || n==1) return 1;
int recAns = fact(n-1)
int finalAns = n * recAns
2 x 1 = 2
return finalAns;
2
recAns = 1
n=1
fact(1)
if (n==0 || n==1) return 1;
int recAns = fact(n-1);
int finalAns = n * recAns;
return finalAns;

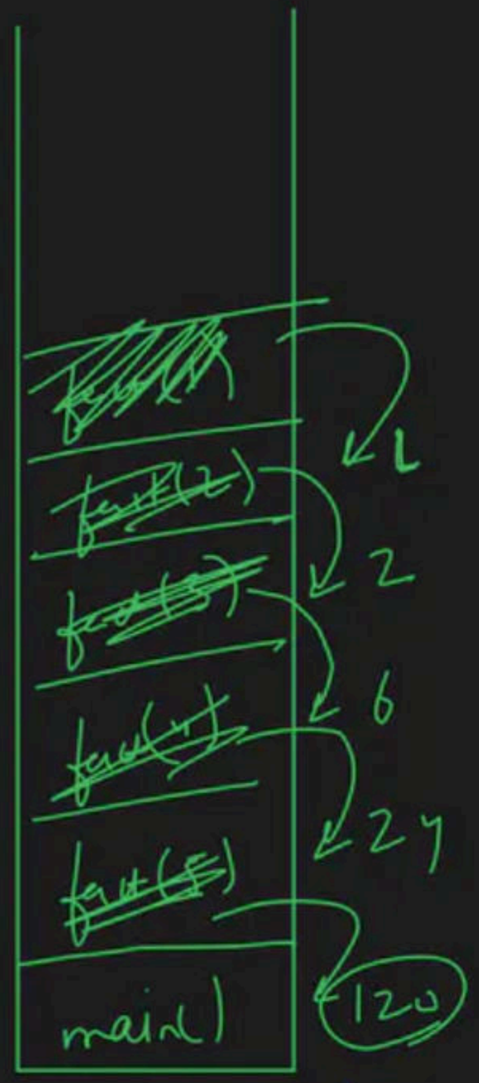
Base Case ?

f(-9)
f(-1)
f(-2

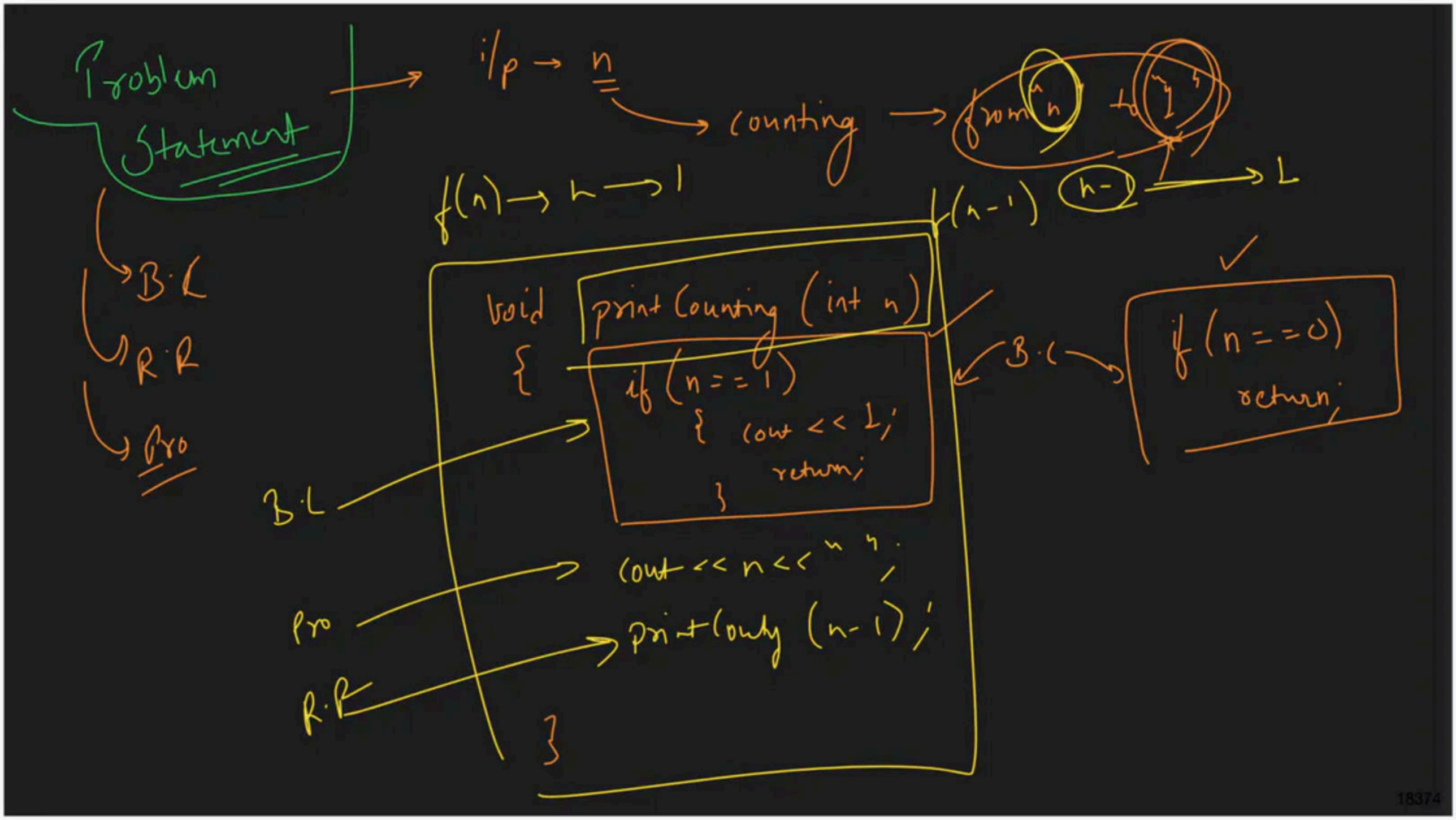
f(-1)
Hoy
f(1)
f(2)
f(3)
f(4)
f(5)
main

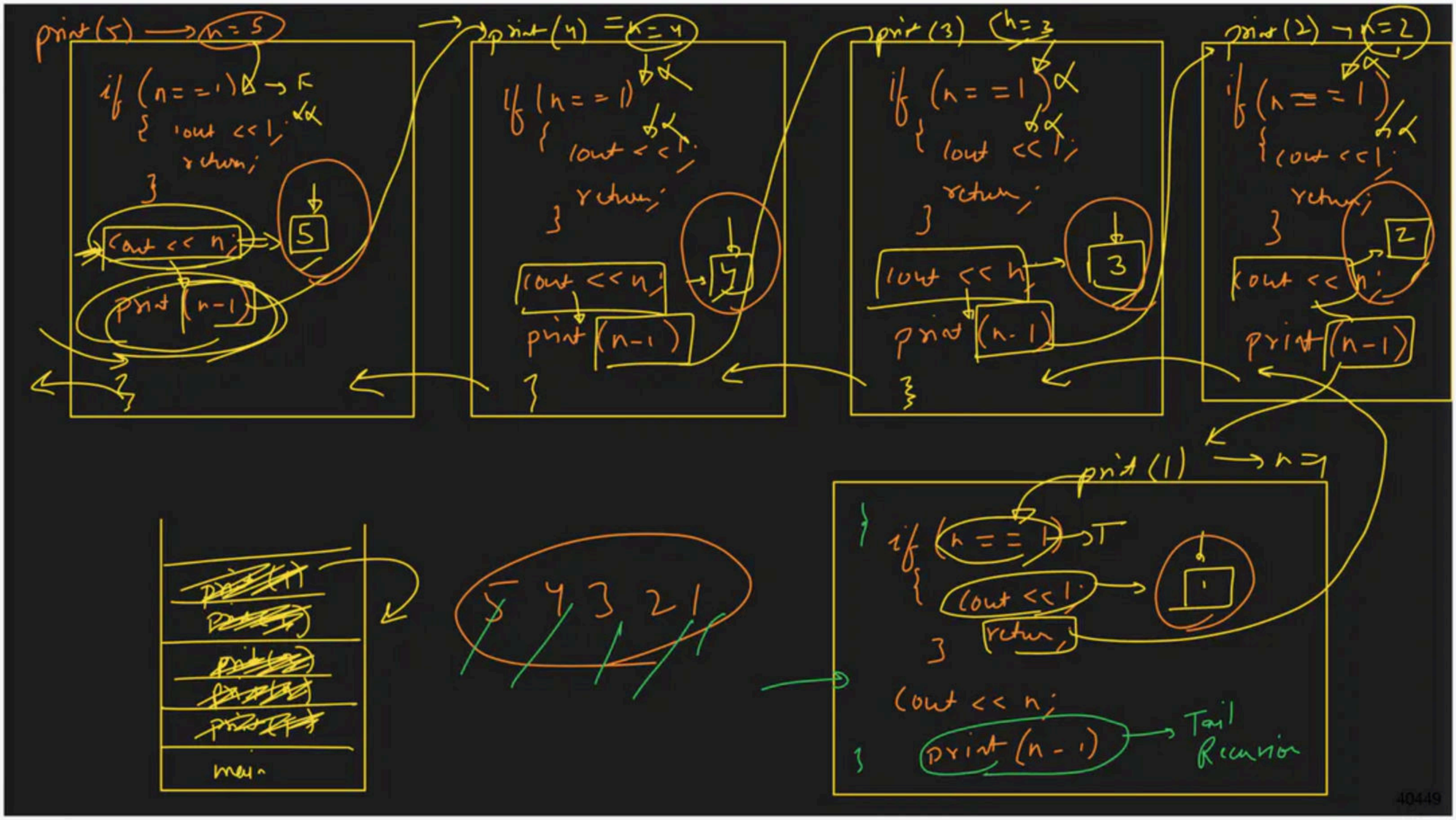
Stack Overflow

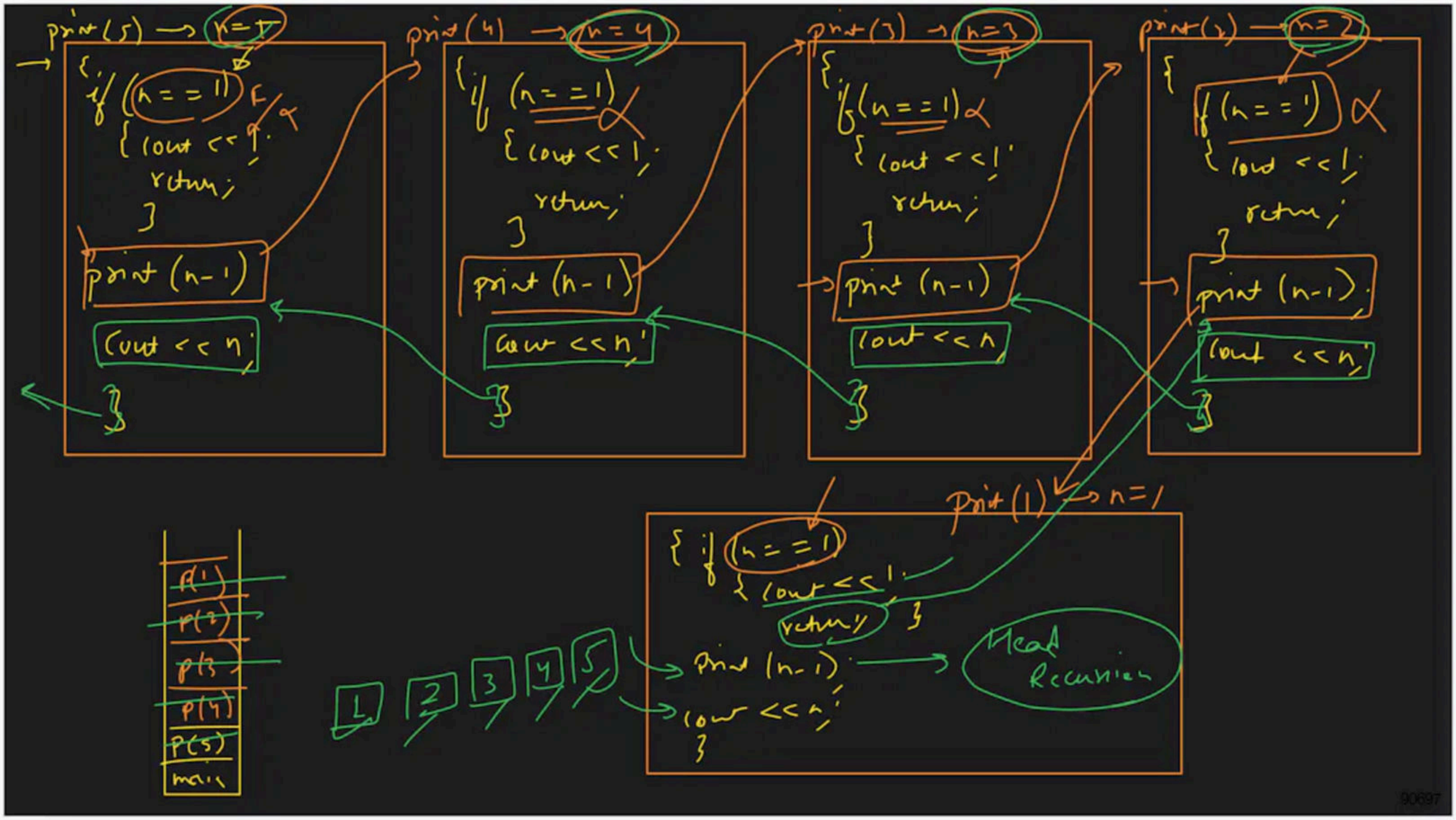
H/W



call Stack


Problem Statement
i/p → n
counting
from n to 1
f(n) → n → 1
(n-1)
n-1
L
B.C
R.R
Pro
void printCounting (int n)
{
if (n == 1)
{ cout << 1;
return;
}
B.C
if (n == 0)
return;
B.L
cout << n << " ";
Pro
printCounty (n-1);
R.R
}


print(5) → n=5
if (n==1) → F
{ cout << 1;
return;
}
cout << n;
print (n-1)
5
print(4) → n=4
if (n==1)
{ cout << 1;
return;
}
cout << n;
print (n-1)
4
print(3) → n=3
if (n==1)
{ cout << 1;
return;
}
cout << n;
print (n-1)
3
print(2) → n=2
if (n==1)
{ cout << 1;
return;
}
cout << n;
print (n-1)
2
print (1) → n=1
if (n==1) → T
{ cout << 1;
return;
}
cout << n;
print (n-1)
1
Tail Recursion
main
5 4 3 2 1


print(5) → n=5
print(4) → n=4
print(3) → n=3
print(2) → n=2
print(1) → n=1
if (n==1)
cout << 1;
return;
print(n-1);
cout << n;
P(1)
P(2)
P(3)
P(4)
P(5)
main
1 2 3 4 5
Head Recursion
90697

P.S $\longrightarrow$ 2^n

pow(n-1) = 2^{n-1}

pow(n) $\longrightarrow$ 2^n

pow(n) = $2 \times 2^{n-1}$

2^5

2×2^4

2×2^3

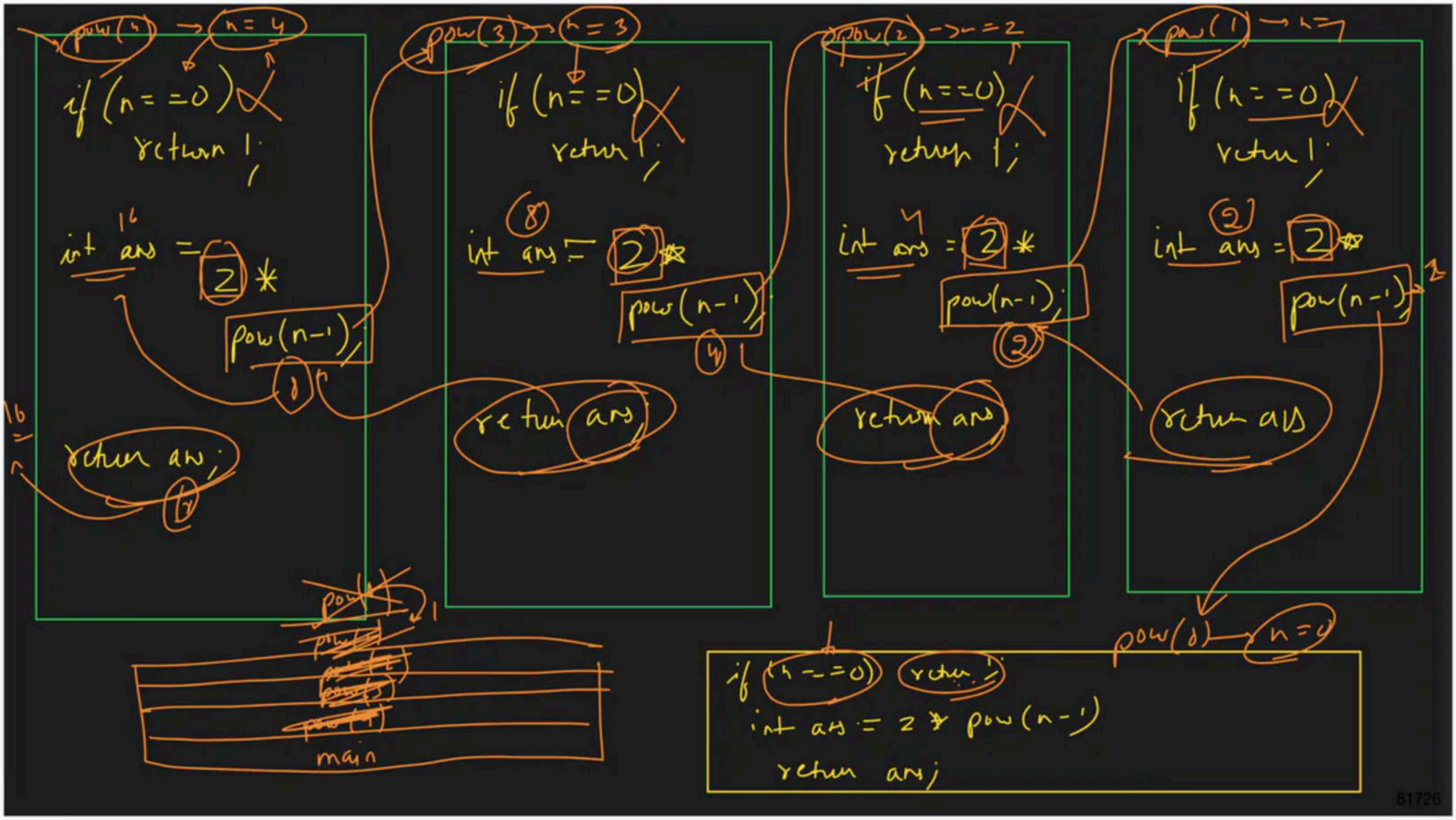
2×2^2

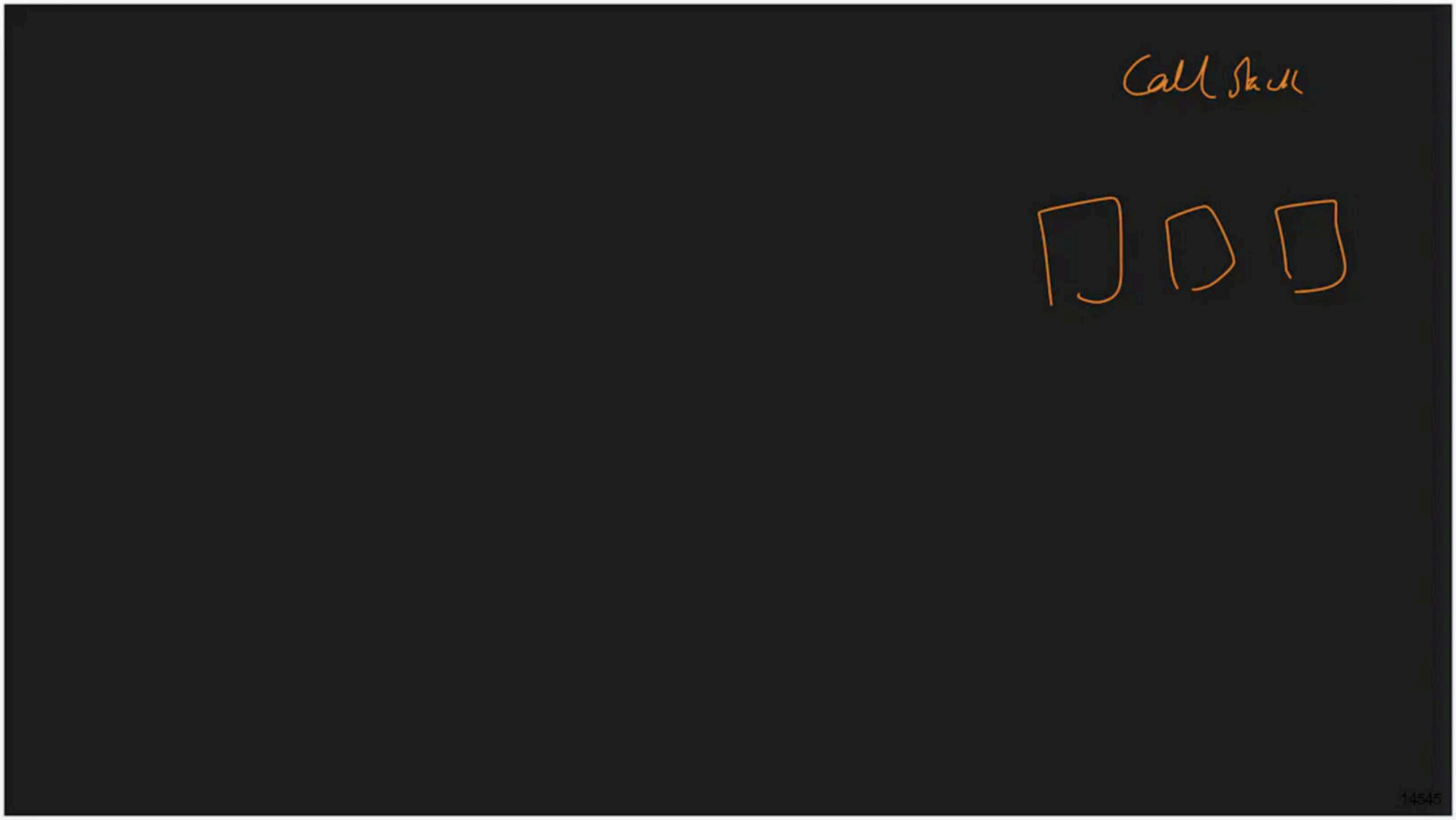
2×2^1

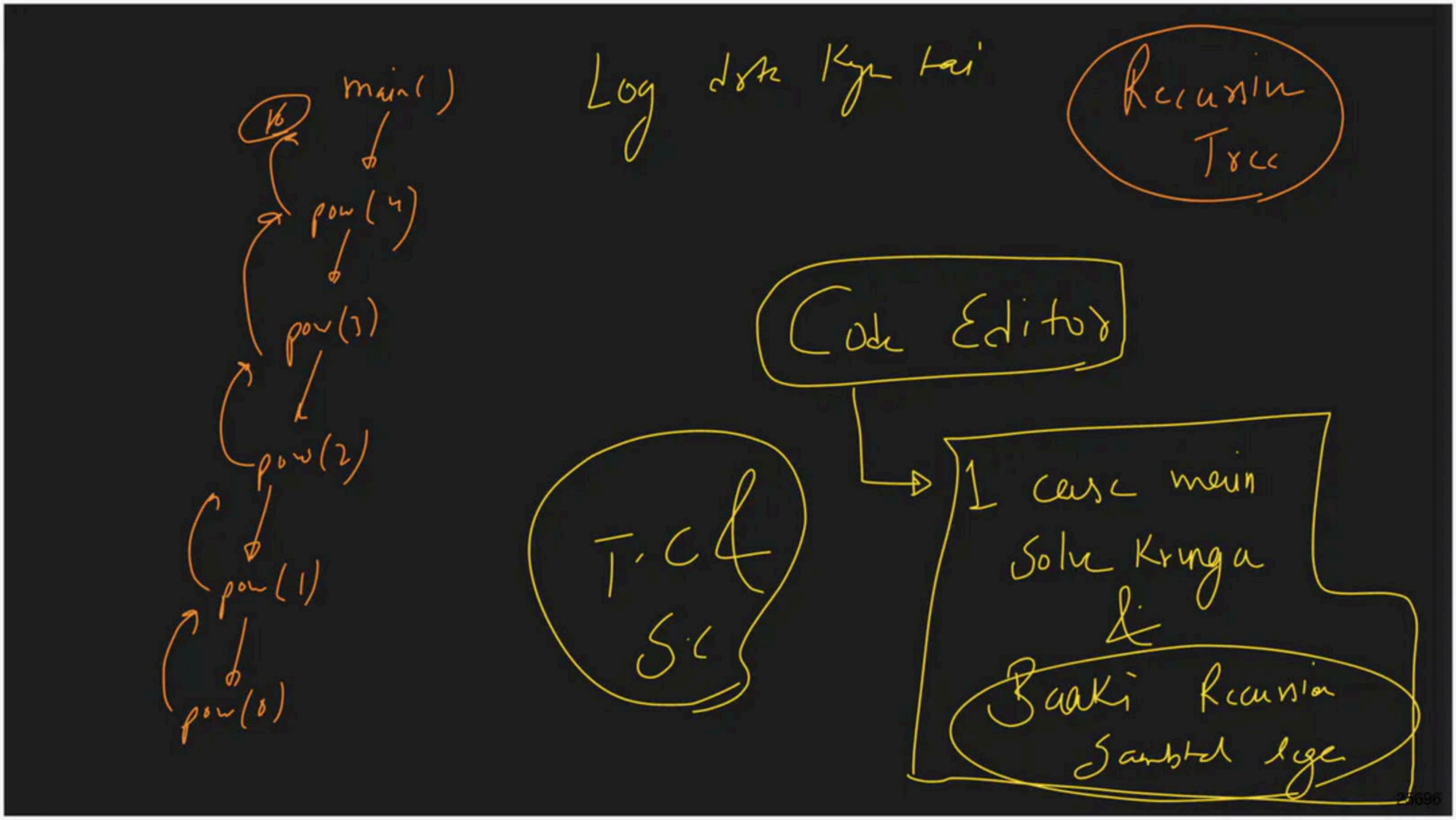
2×2^0 → =1 → if(n==0) return 1

pow(n) = 2 × pow(n-1)

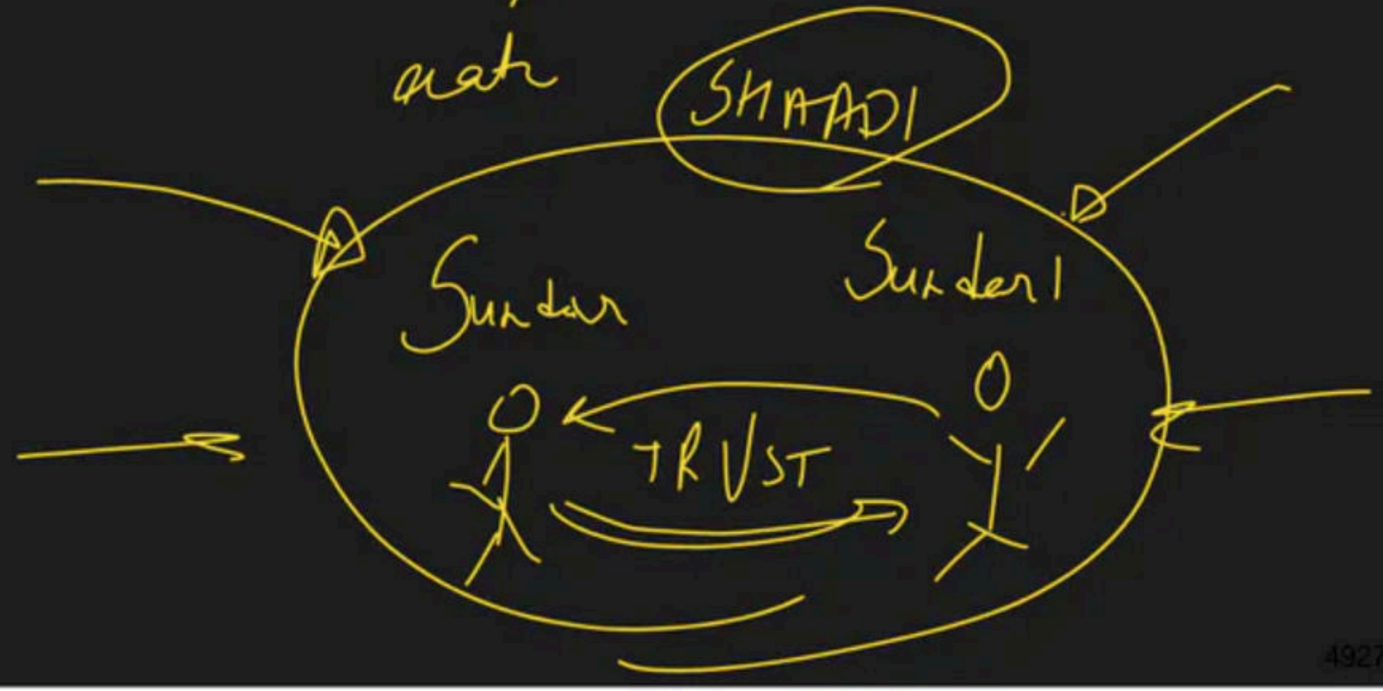
Recursive relation


pow(4)
n = 4
if (n == 0)
return 1;
int ans = 2 *
pow(n-1);
return ans;
pow(3)
n = 3
if (n == 0)
return 1;
int ans = 2 *
pow(n-1);
return ans;
pow(2)
n = 2
if (n == 0)
return 1;
int ans = 2 *
pow(n-1);
return ans;
pow(1)
n = 1
if (n == 0)
return 1;
int ans = 2 *
pow(n-1);
return ans;
pow(0)
n = 0
if (n == 0) return 1;
int ans = 2 * pow(n-1)
return ans;
main

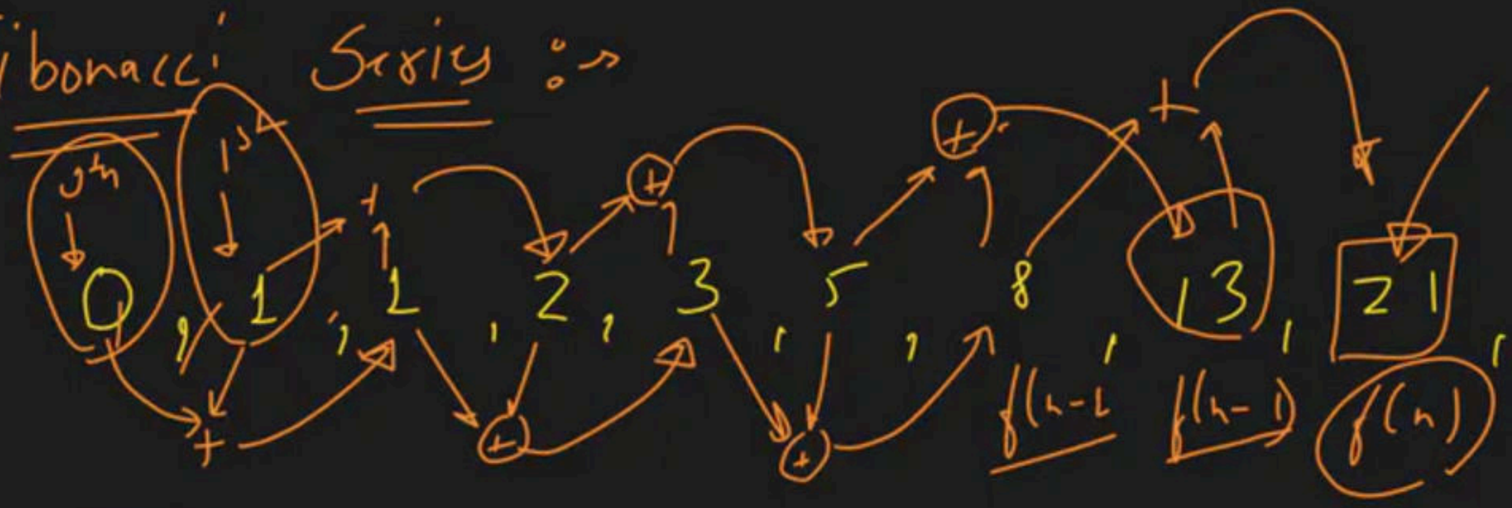

Call Skull


main()
pow(4)
pow(3)
pow(2)
pow(1)
pow(0)
Log dekh kya rahi
Recursion Tree
Code Editor
T.C & S.C
1 case main solve krunga &
Baaki Recursion sambhal lega

To understand Recursion, you need to first understand Recursion



→ Fibonacci Series :-



find n^{th} term

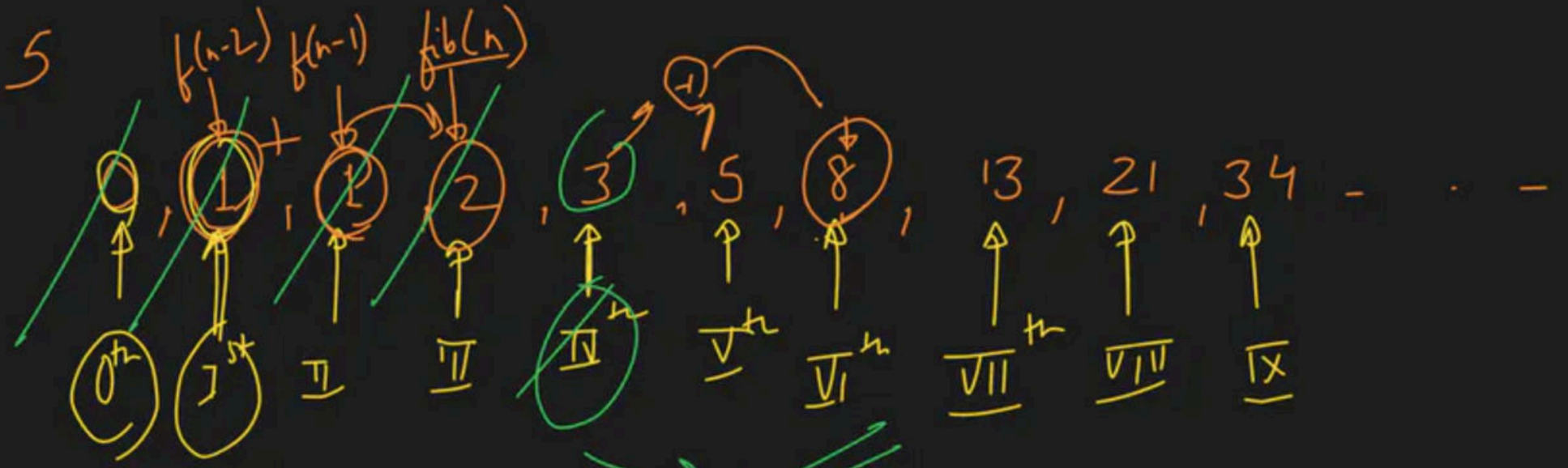
$$f(n) = f(n-1) + f(n-2)$$

Rec. relation

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

```
if(n==0)
    return 0;
if(n==1)
    return 1;
```

Ex 5



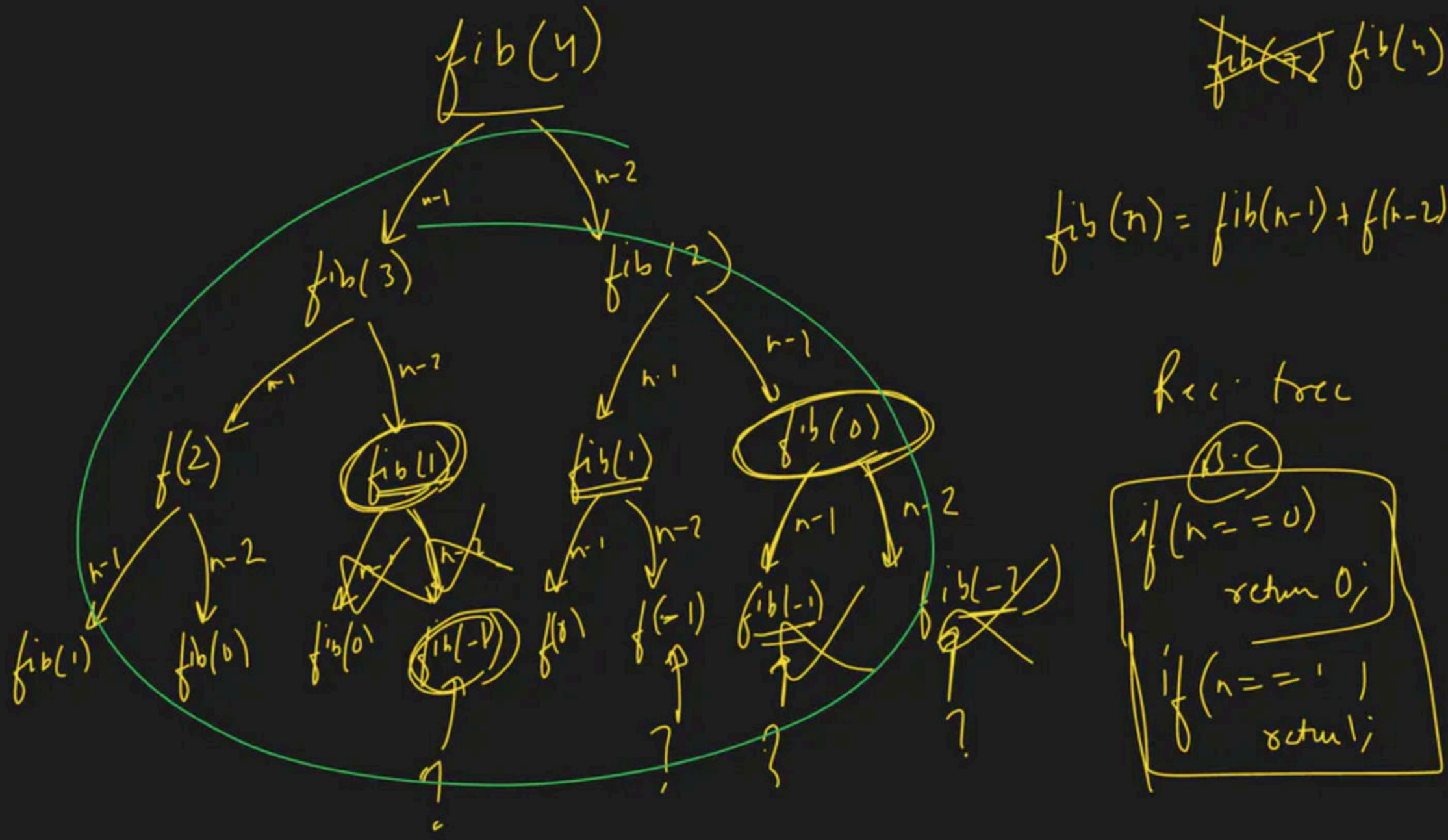
find n^{th} term → $f(n)$

B.C

$$fib(n) = fib(n-1) + fib(n-2)$$

fib(7)

f

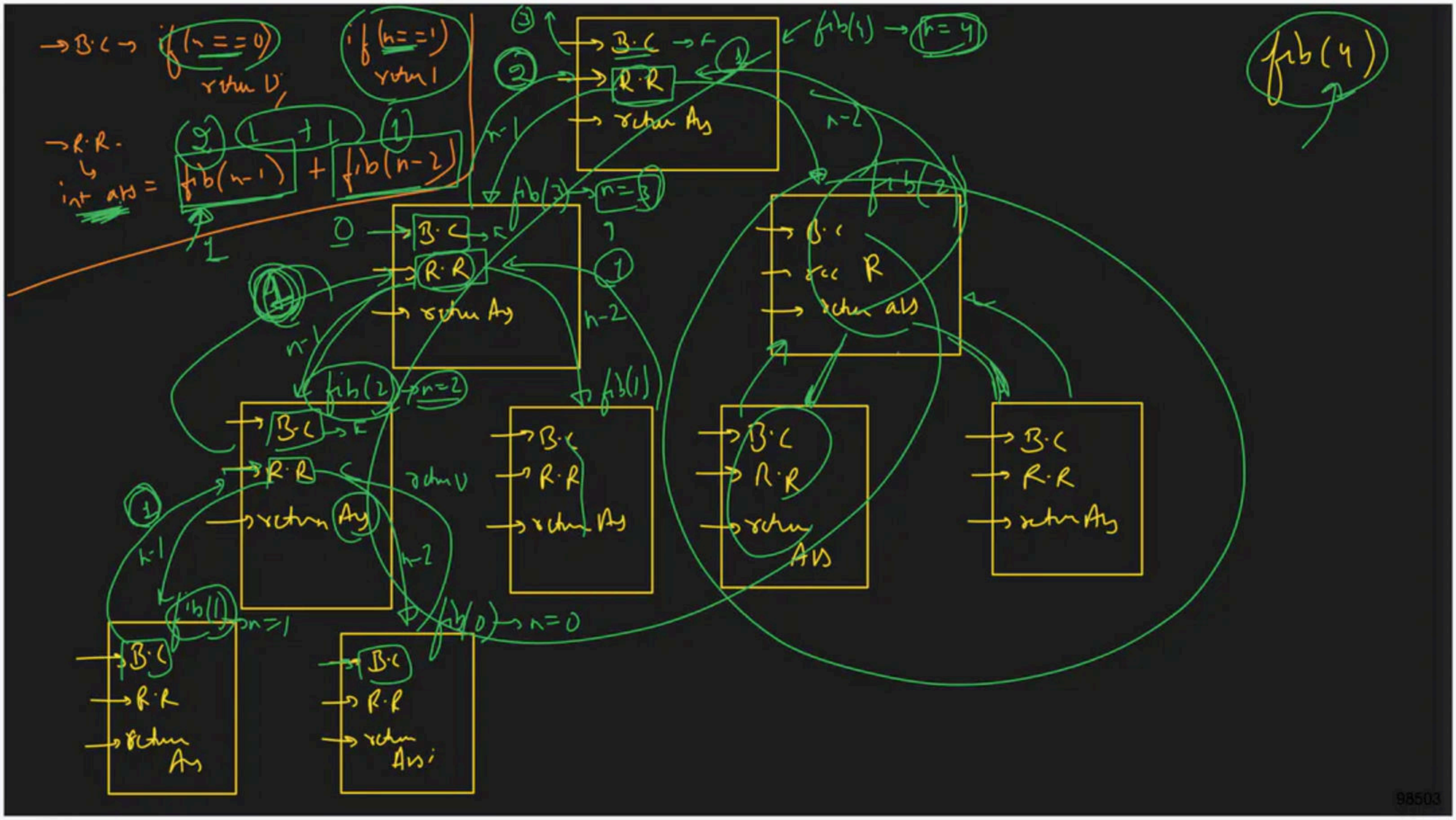


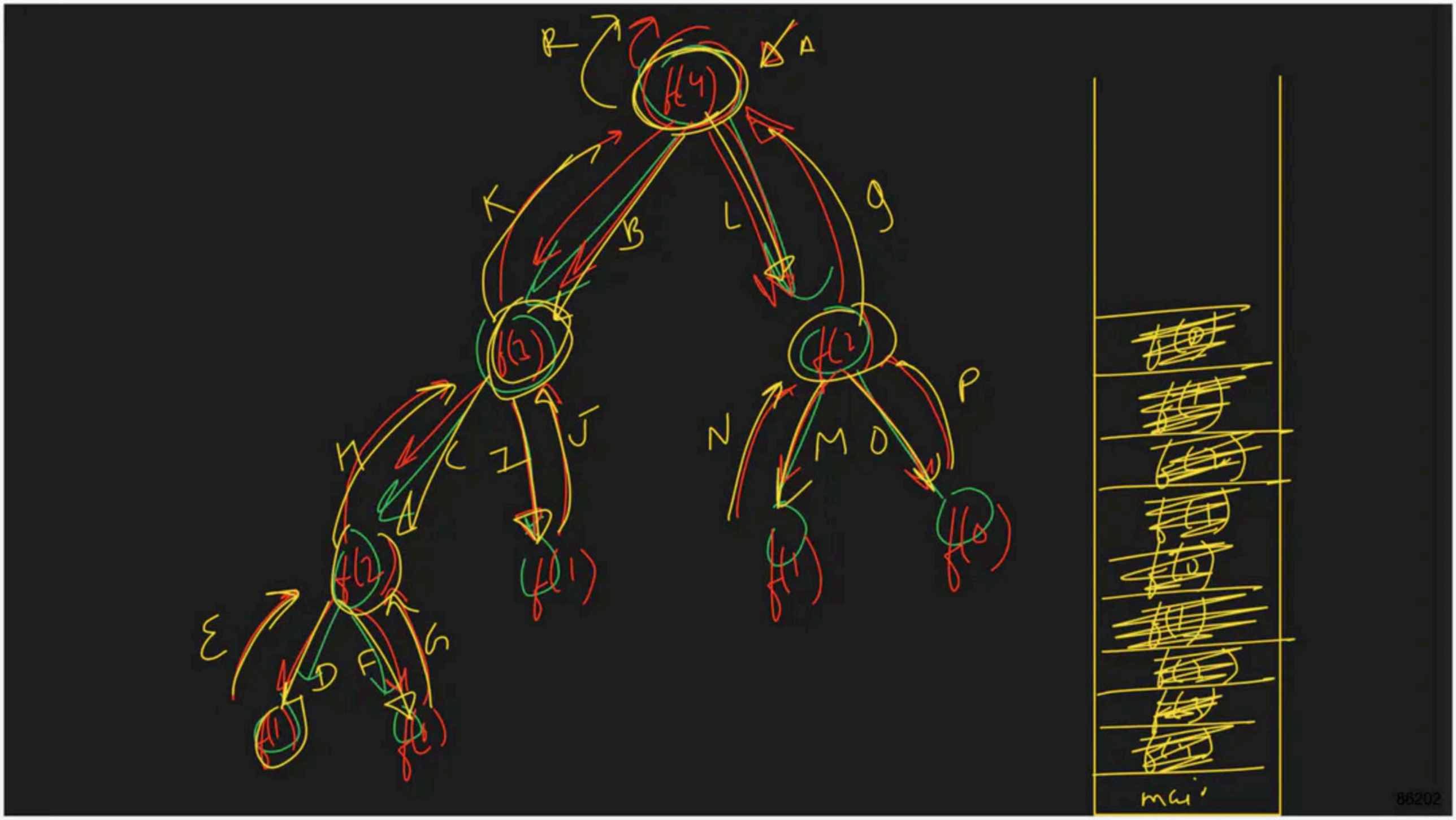
~~fib(-1)~~ fib(4)

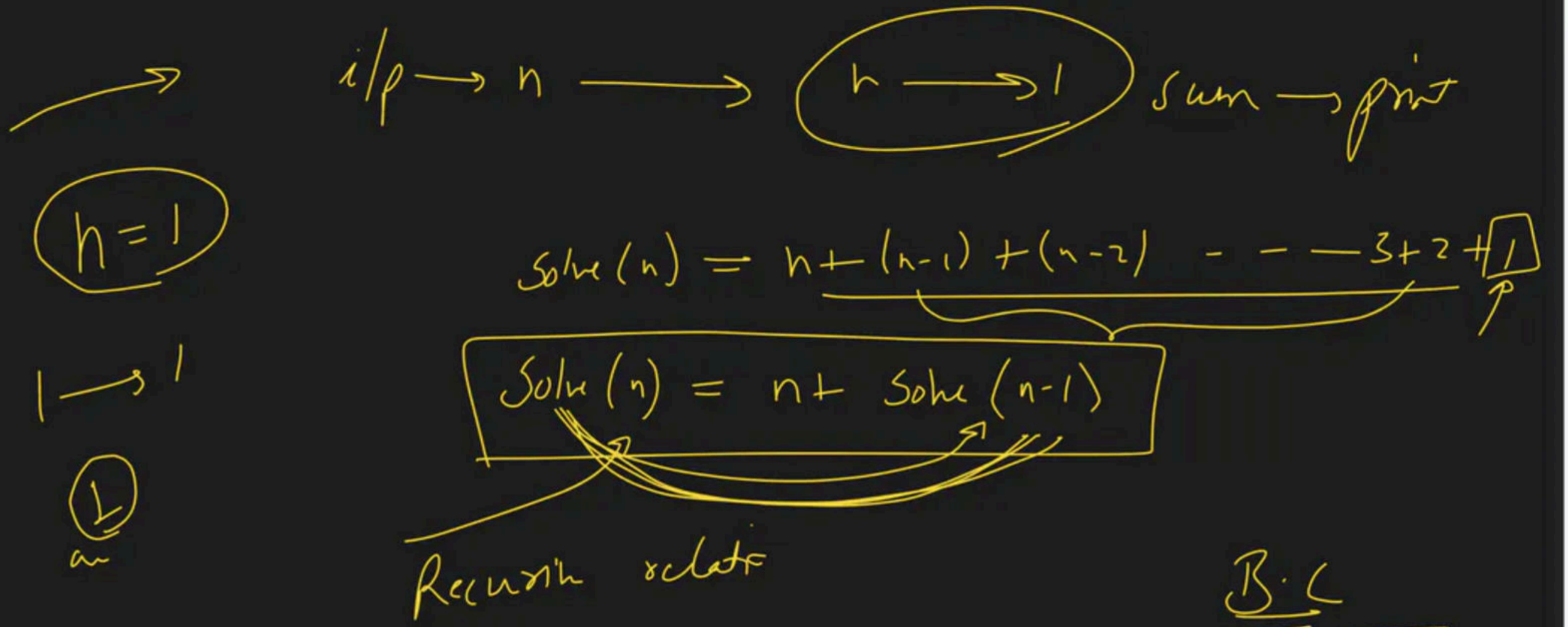
$$fib(n) = fib(n-1) + f(n-2)$$

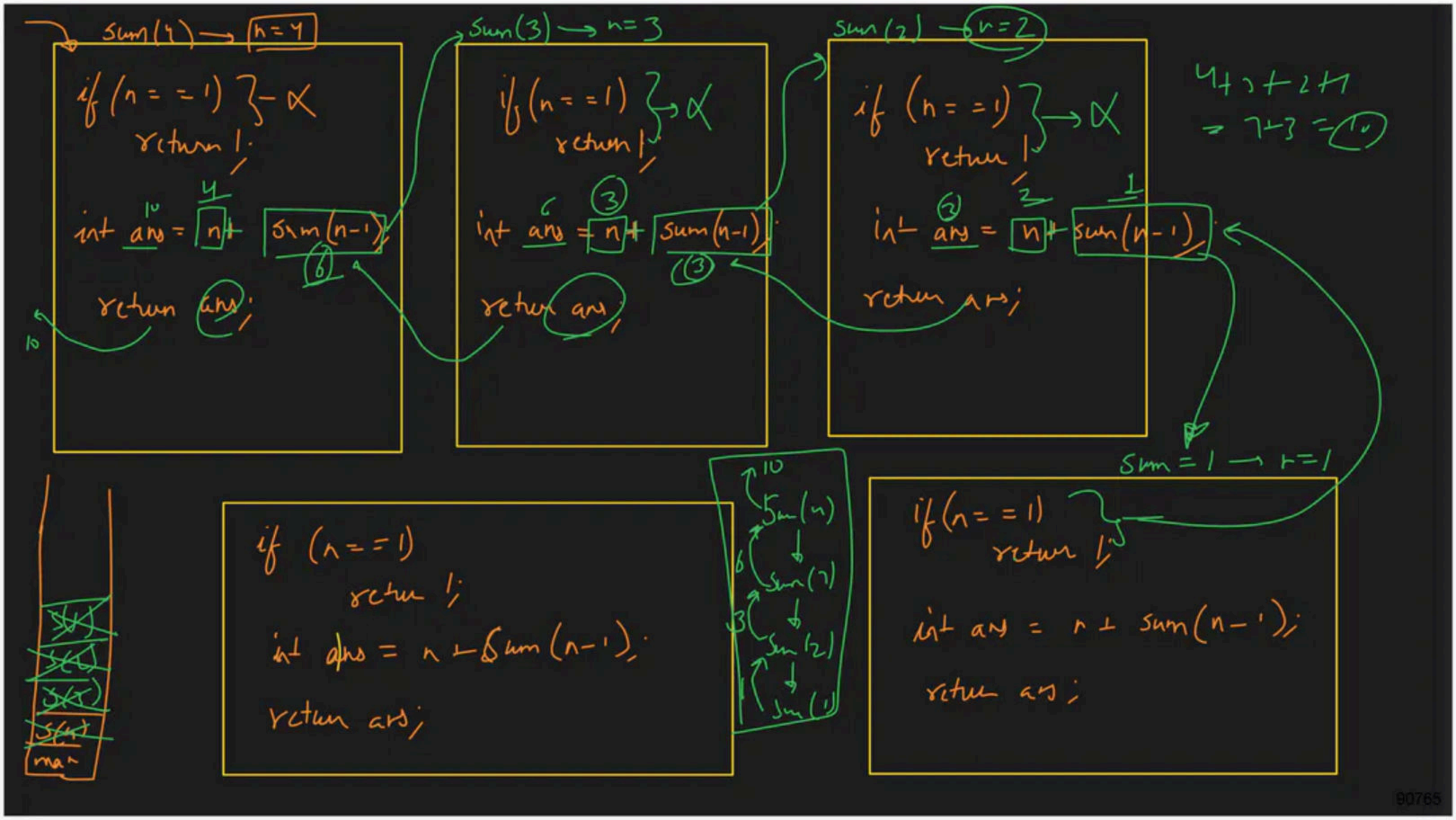
Ricorsione

```
0-C
if(n==0)
    return 0;
if(n==1)
    return 1;
```


B.C
R.R
return Ans
fib(n-1) + fib(n-2)
int ans =
if(n==0) return 0
if(n==1) return 1
fib(4)
fib(3)
fib(2)
fib(1)
fib(0)
n=4
n=3
n=2
n=1
n=0
n-1
n-2
return 0
return 1
return ans
98503


R
A
K
B
L
g
H
C
I
J
N
M
O
P
E
D
F
S
main


i/p → n
n → 1
sum → print
n=1
1 → 1
1
Solve(n) = n + (n-1) + (n-2) - - - 3+2+1
Solve(n) = n + Solve(n-1)
Recursive relate
B.C
if (n==1)
return 1

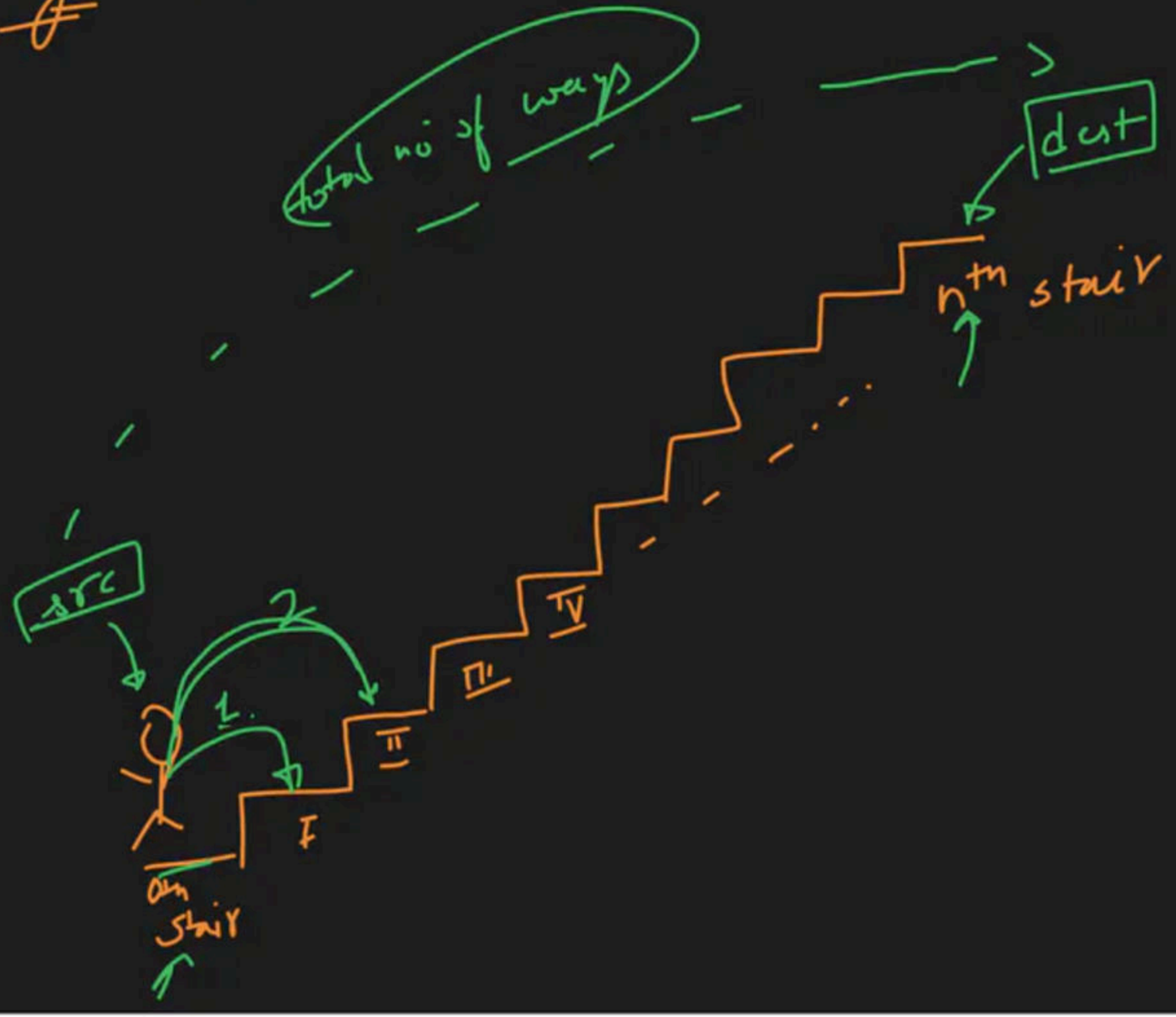

sum(4) → n=4
if (n==1) return 1;
int ans = n + sum(n-1);
return ans;
sum(3) → n=3
if (n==1) return 1;
int ans = n + sum(n-1);
return ans;
sum(2) → n=2
if (n==1) return 1;
int ans = n + sum(n-1);
return ans;
4+3+2+1
= 7+3 = 10
sum=1 → n=1
if (n==1) return 1;
int ans = n + sum(n-1);
return ans;
if (n==1) return 1;
int ans = n + sum(n-1);
return ans;
sum(4)
sum(3)
sum(2)
sum(1)
10
6
3
1
main

Recursion - Class 2

Special class

Love Babbar • Oct 10, 2023

Climbing Stairs



⇒ Climbing stair

→

N = 3

=

☆☆☆

All possibilities

=

RE

Constraints

↳ 1 step

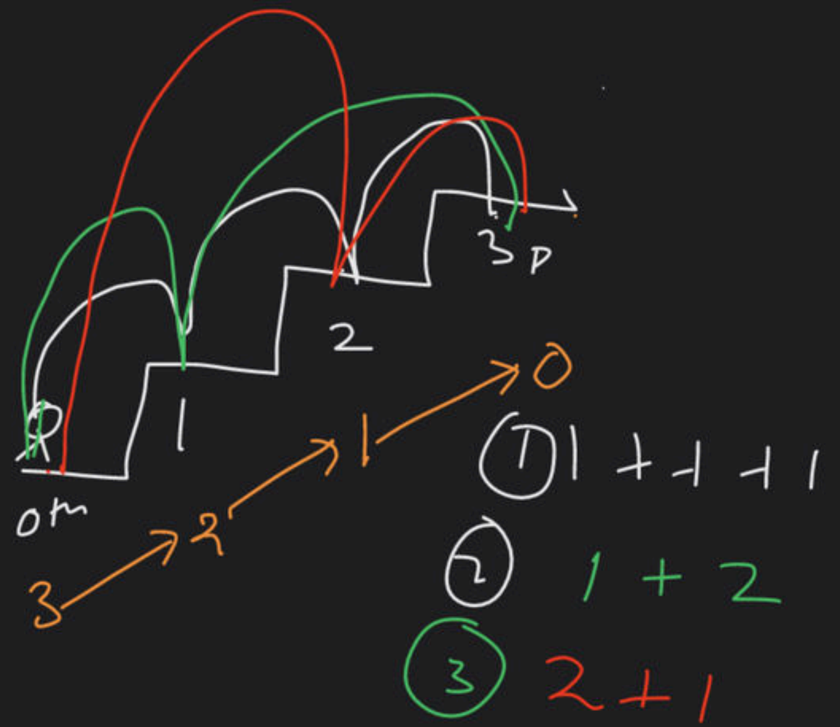
OR

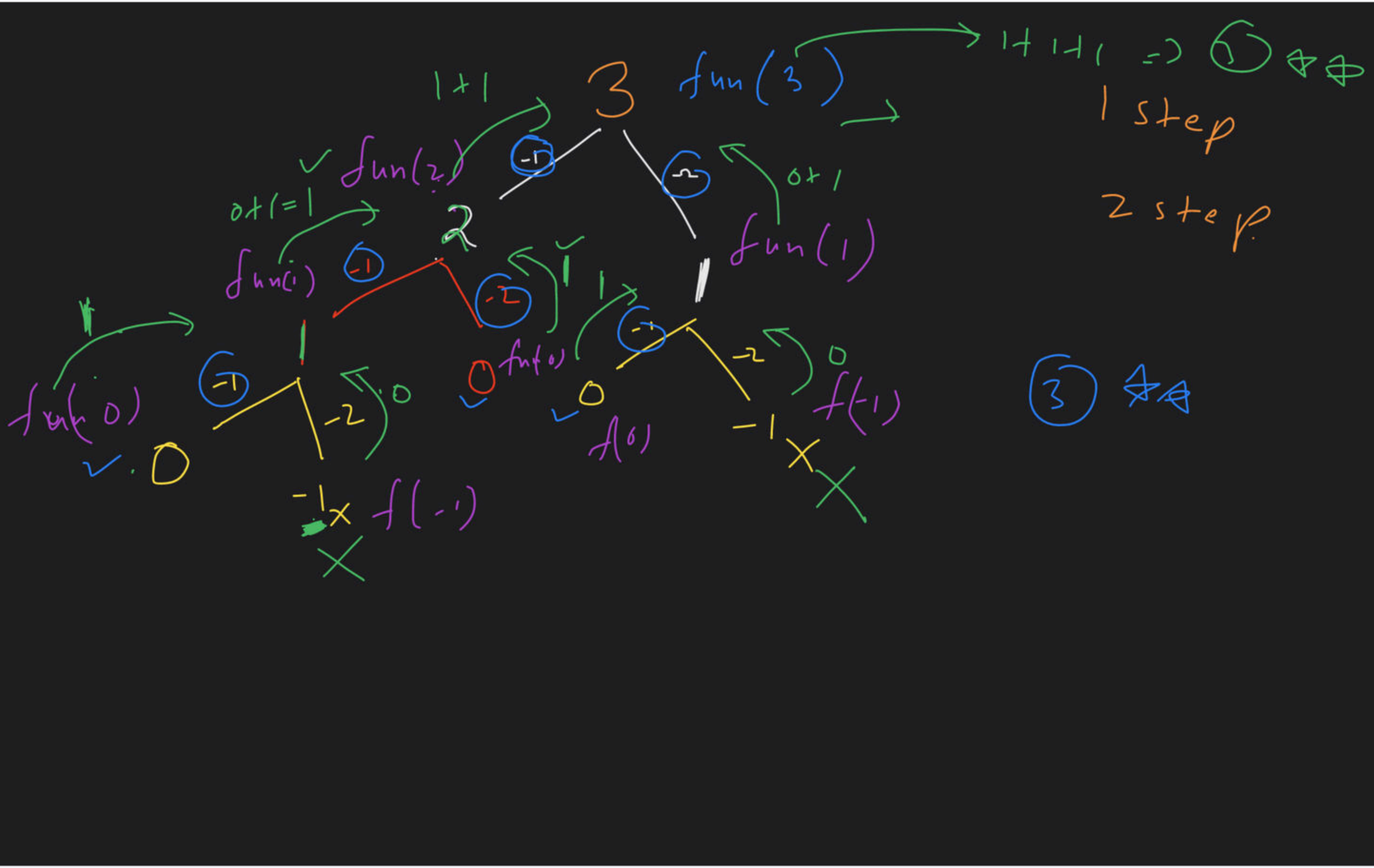
2 step

=

k ; the ways

=




1+1
3
fun(3)
1+1+1 => 5
1 step
fun(2)
0+1=1
2
0+1
2 step
fun(1)
fun(1)
-1
-2
1
-1
fun(0)
0
-1
-2
0
0
fun(0)
-2
0
f(-1)
f(0)
-1
-1
f(-1)
3

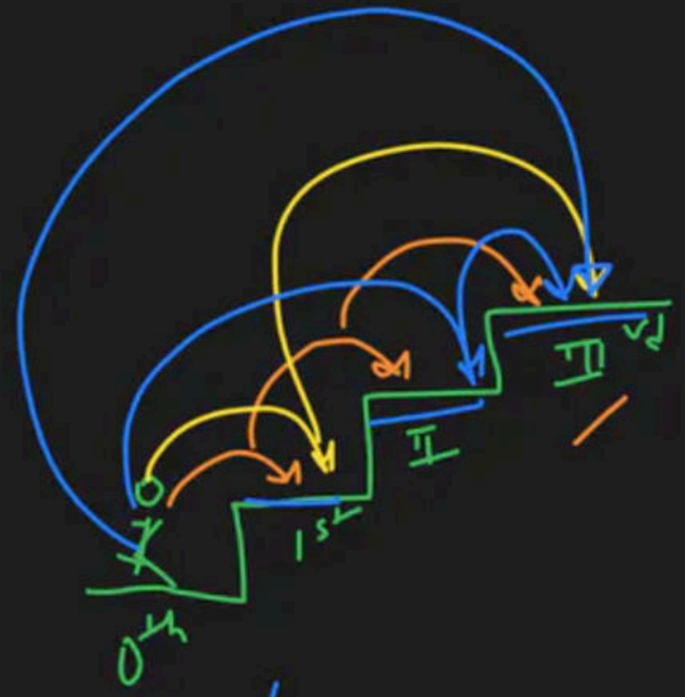
```
int fun(int n)
{   Base can.    if(n==0)  return 1;
                 if(n<0)   return 0;

    int take1step = fun(n-1);
    int take2step = fun(n-2);
    return take1step + take2step;
}
```

✓

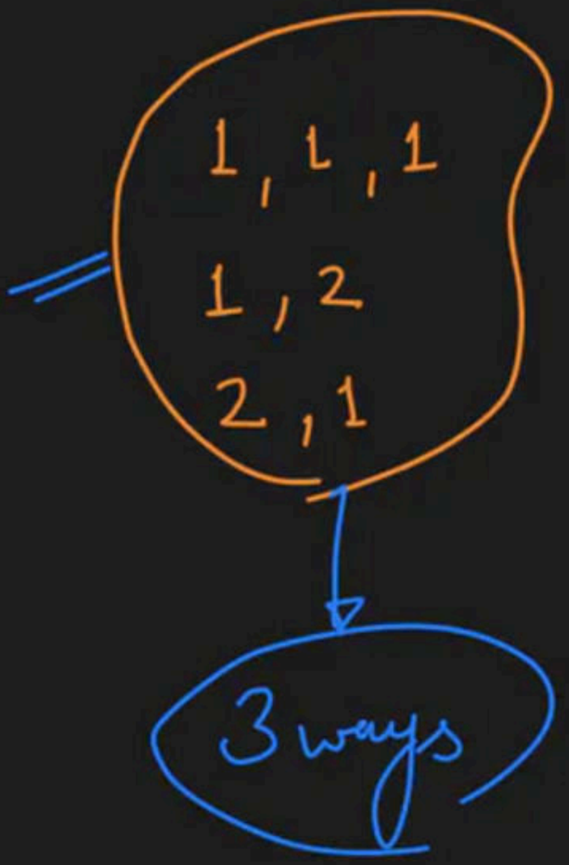
} B. Case ✗

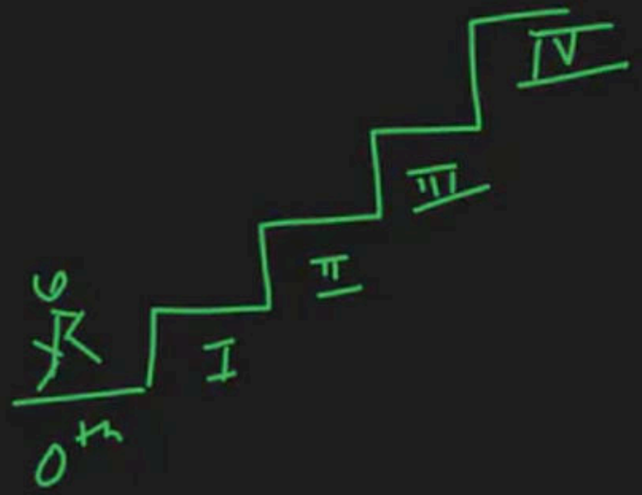
} RE

} 1 case solved


0th
1st
2nd
3rd


1
2
1


1, 1, 1
1, 2
2, 1
3 ways



1, 1, 1, 1

1, 1, 2

1, 2, 1

2, 1, 1

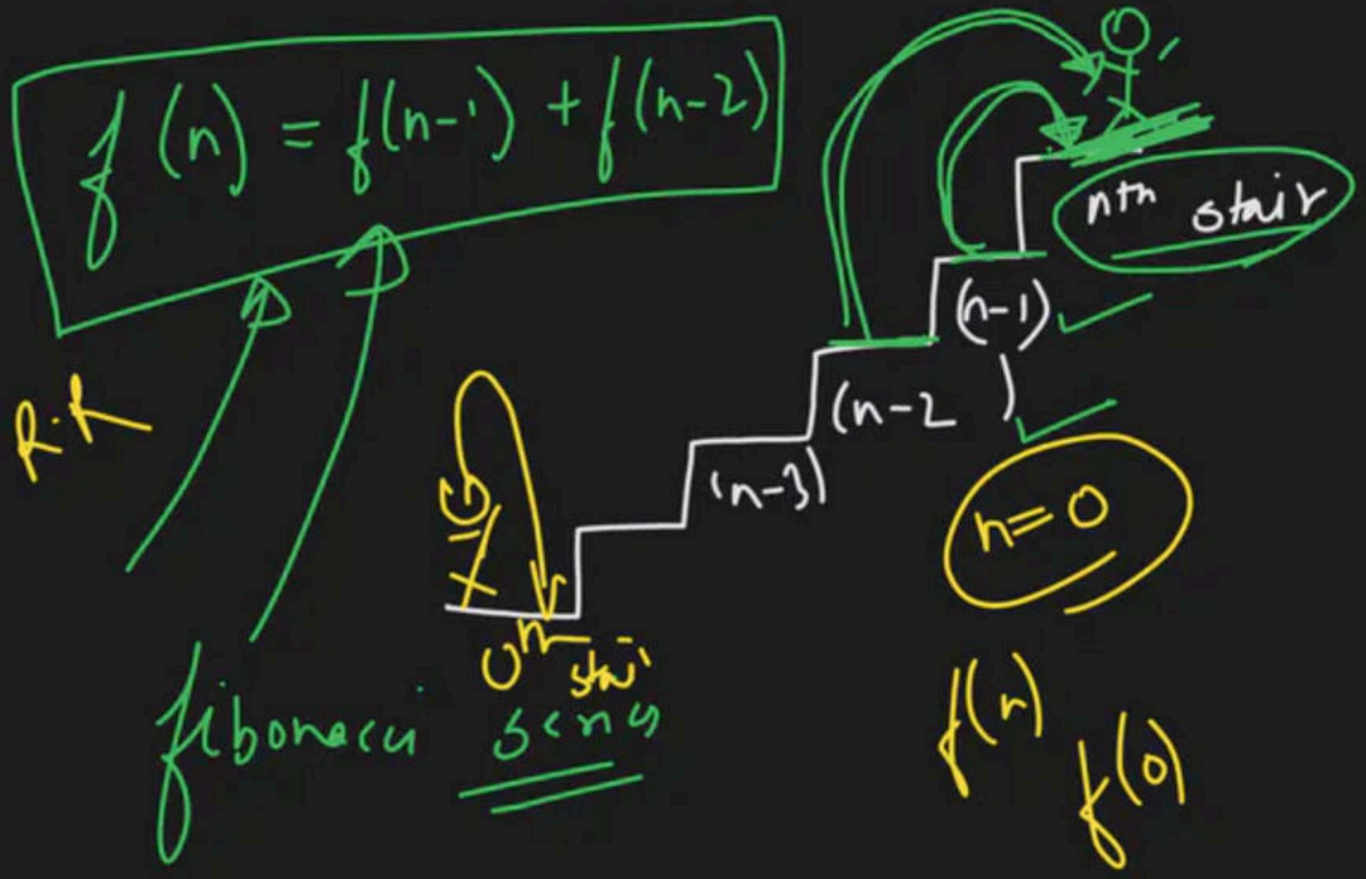
2, 2

→ 5 ways

n^{th} Stair → total no of ways

$f(n)$ → no of ways to reach n^{th} stair

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



1, 1, 1, 1, 1

1, 1, 1, 2

1, 1, 2, 1

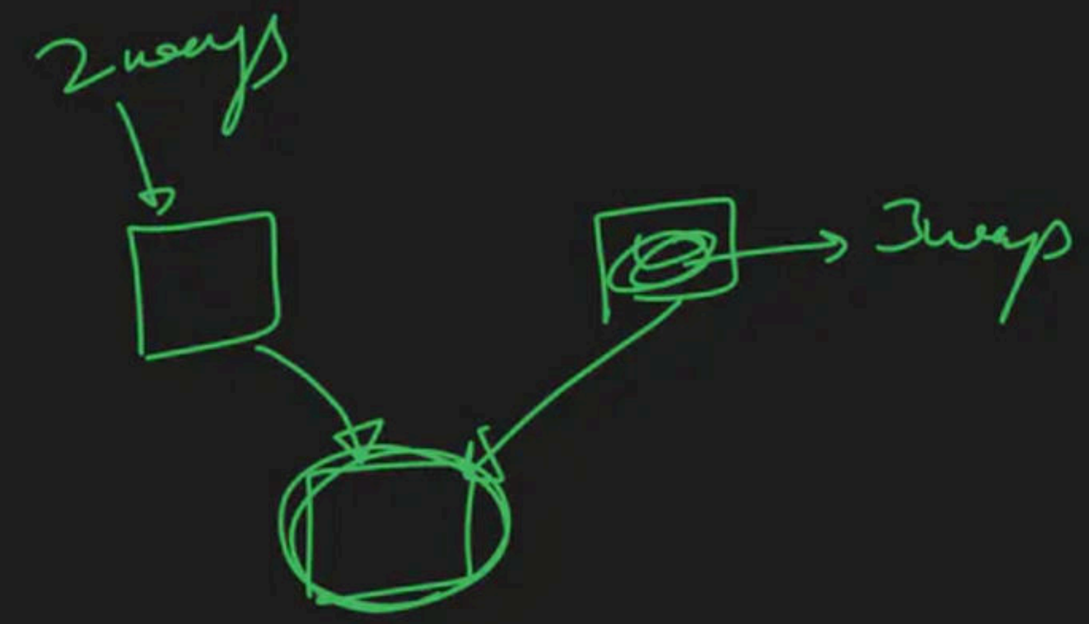
1, 2, 1, 1

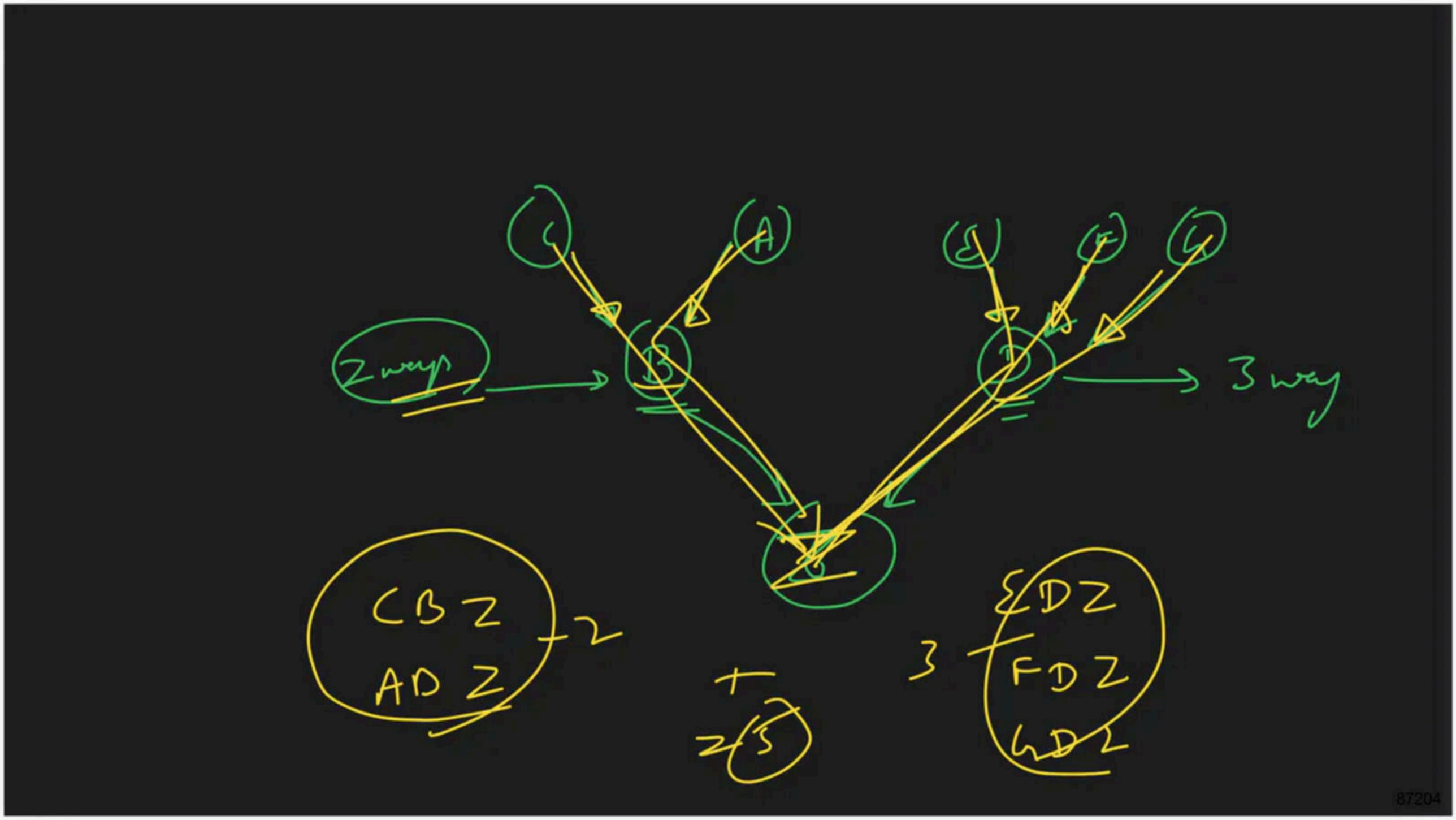
2, 1, 1, 2

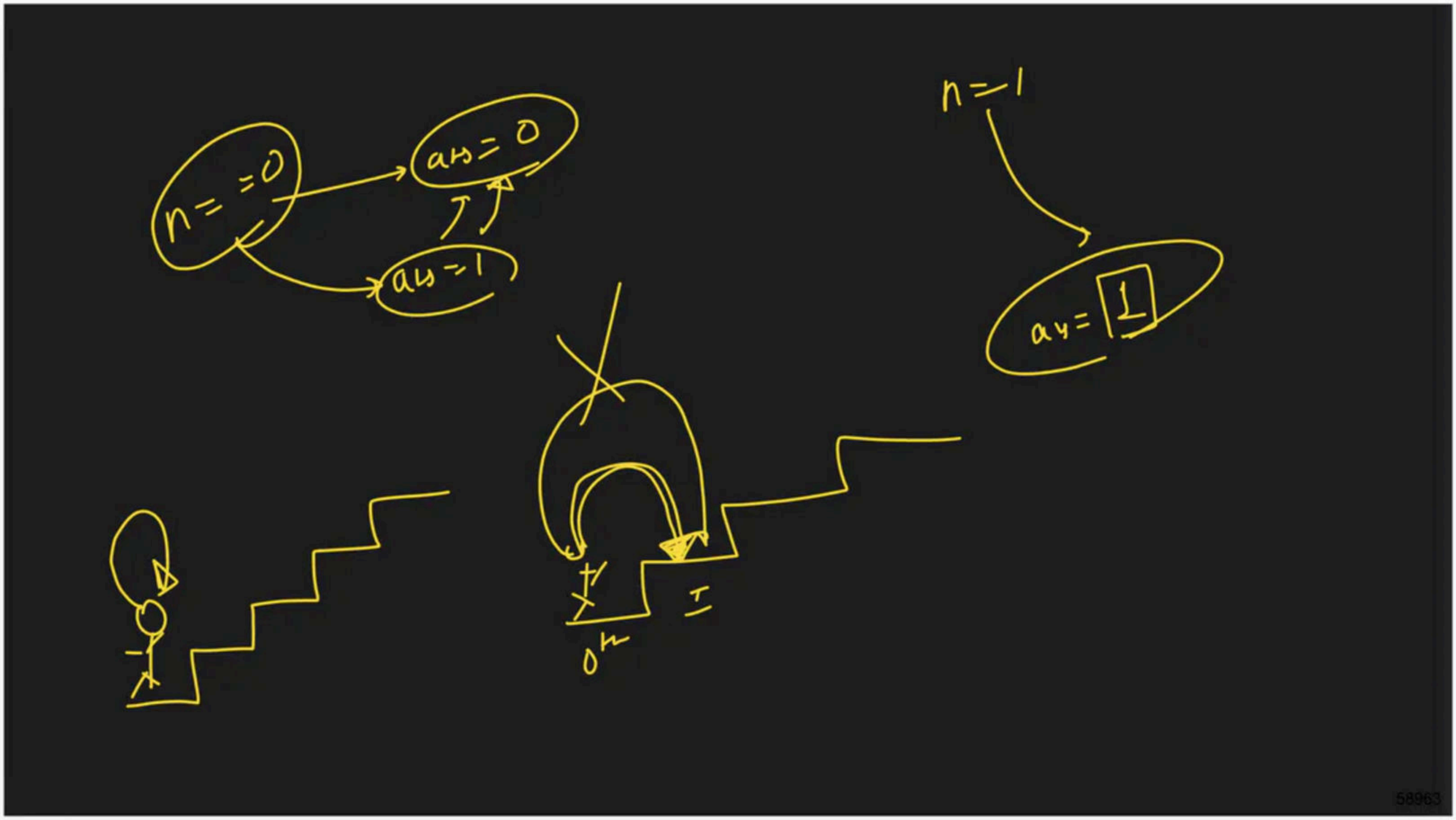
1, 2, 2

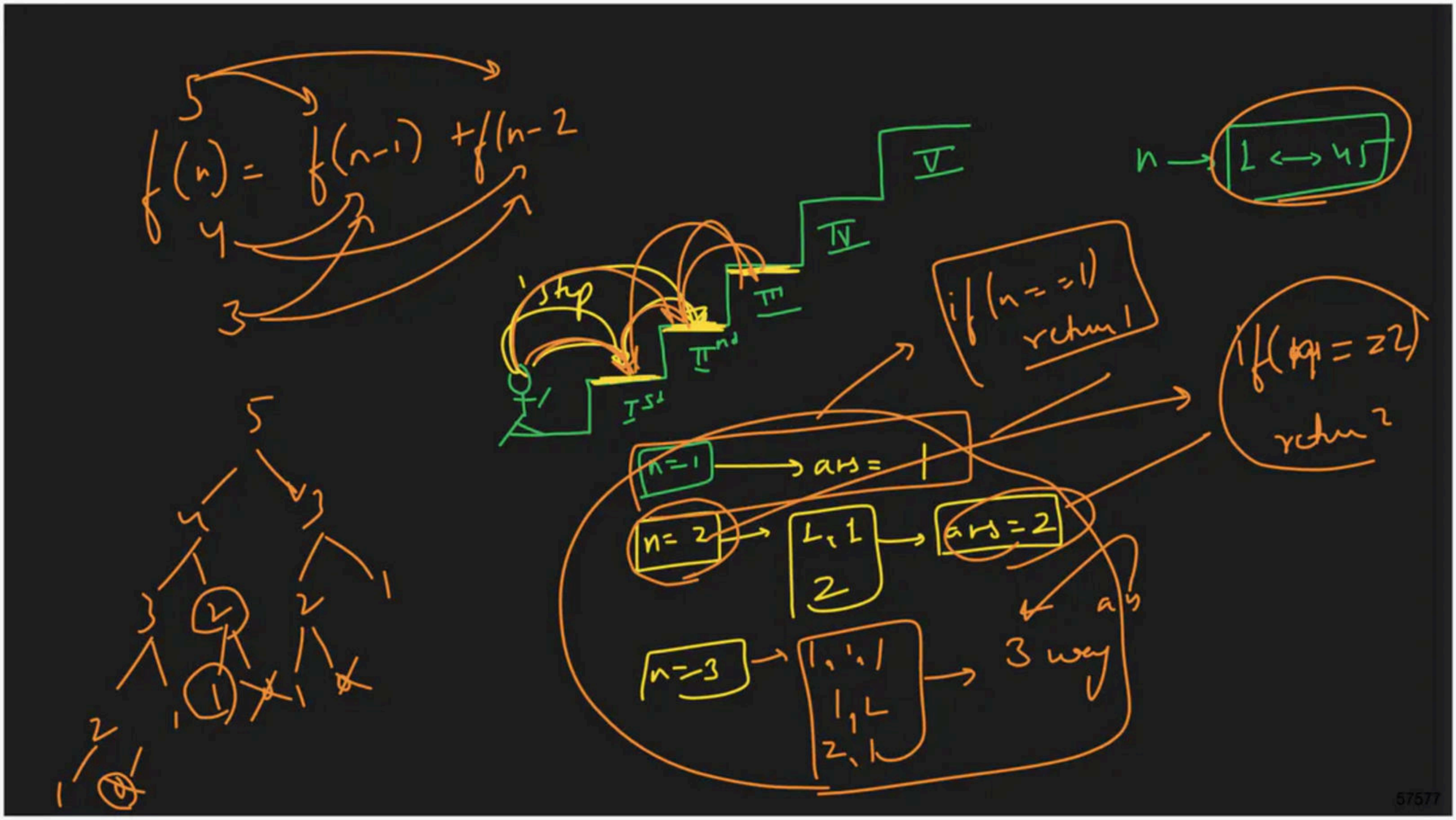
2, 1, 2

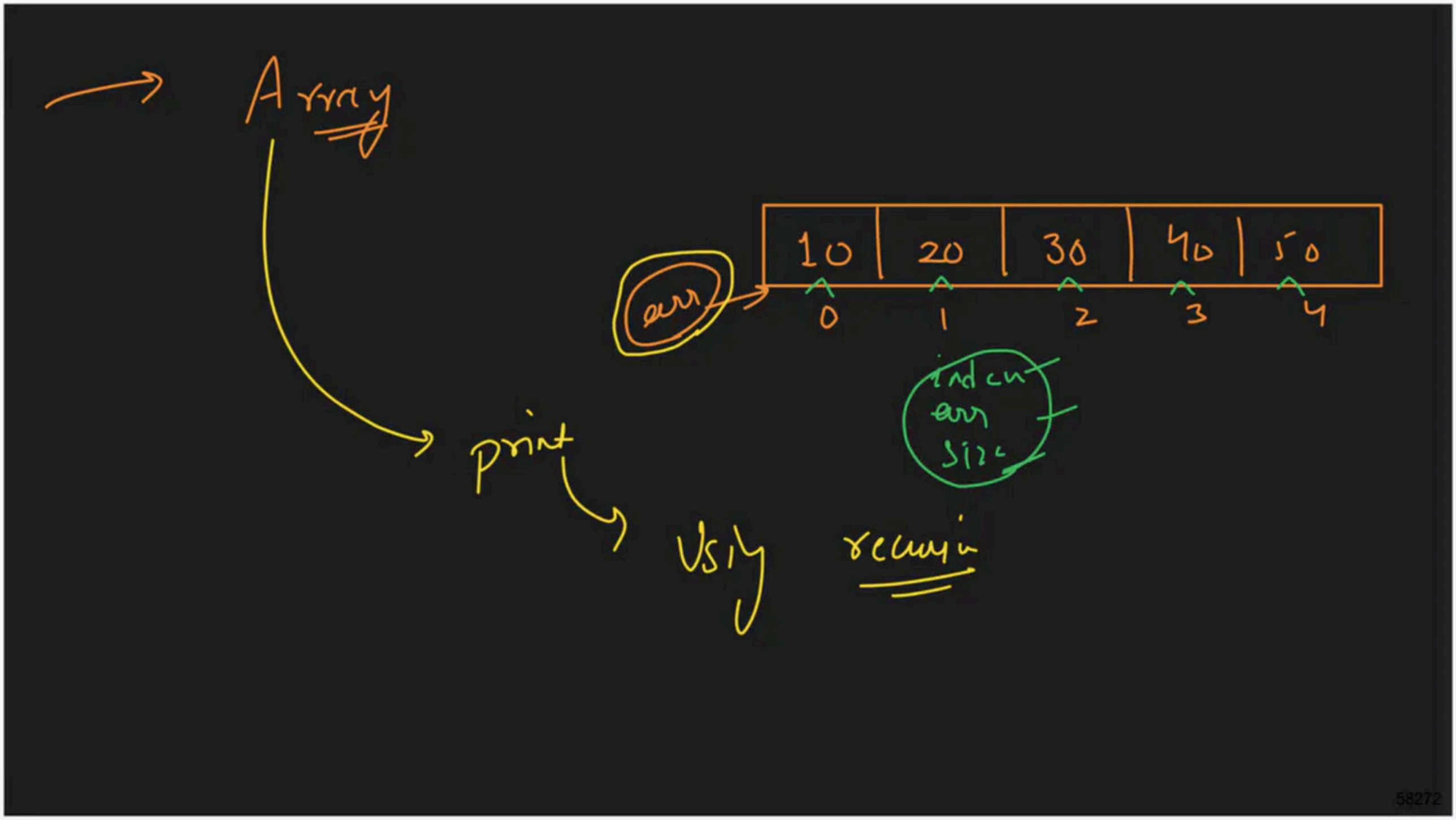
2, 2, 1

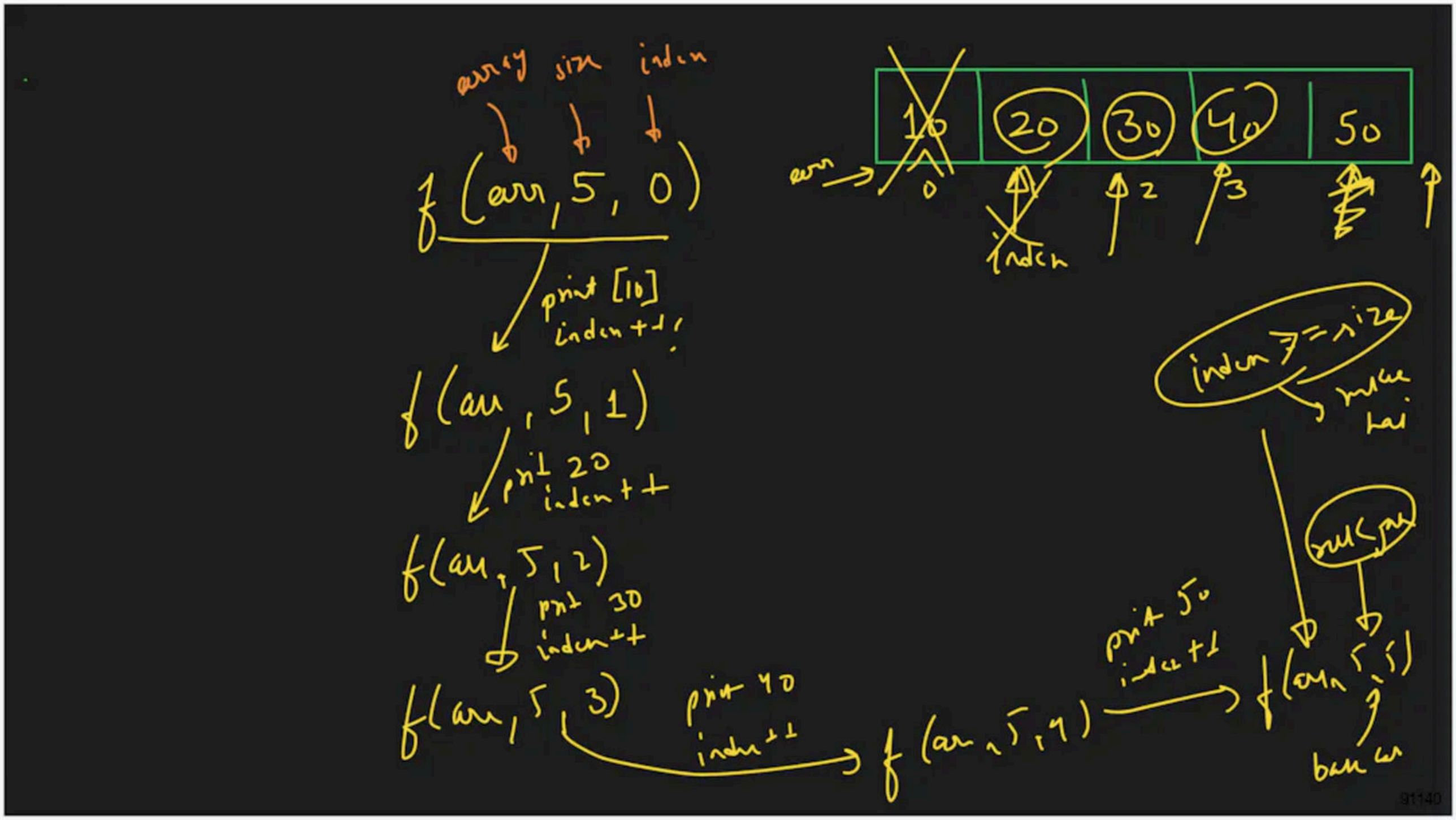

2 ways
3 ways


C
A
B
2 ways
D
E
F
3 way
C B Z
A D Z
+2
+
2 5
E D Z
3
F D Z
G D Z


n==0
ans=0
ans=1
n==1
ay=1

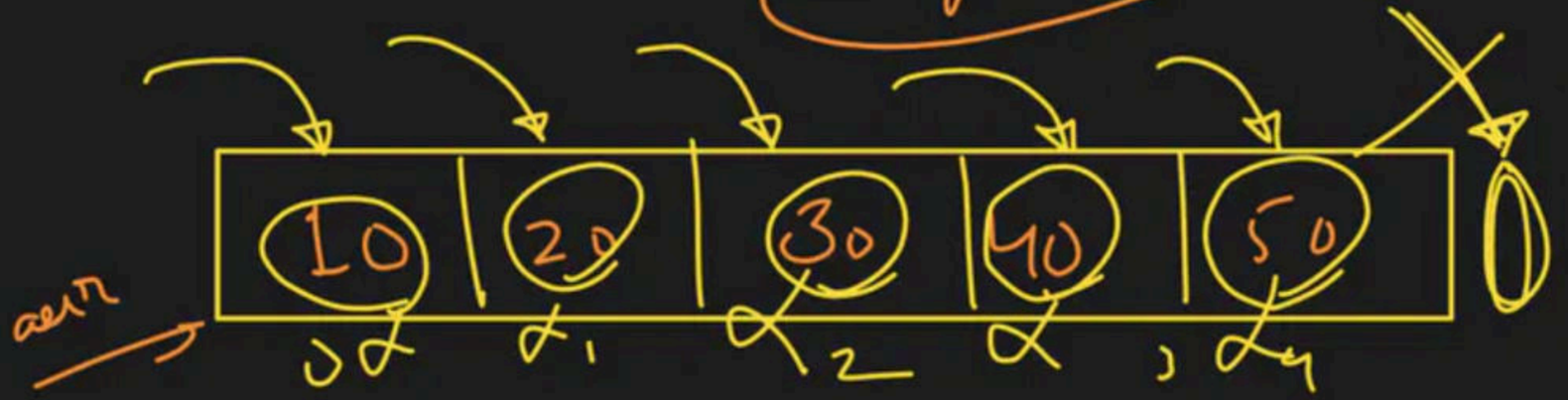

f(n) = f(n-1) + f(n-2)
5
4
3
1 step
n → 1 ↔ 45
if (n == 1) return 1
if (n == 2) return 2
n = 1 → ans = 1
n = 2 → 1, 1 / 2 → ans = 2
n = 3 → 1,1,1 / 1,2 / 2,1 → 3 ways
ans
57577


Array
10
20
30
40
50
arr
0
1
2
3
4
index
arr
size
print
visit
recursion


arr
size
index
f(arr, 5, 0)
print [10]
index++
f(arr, 5, 1)
print 20
index++
f(arr, 5, 2)
print 30
index++
f(arr, 5, 3)
print 40
index++
f(arr, 5, 4)
print 50
index++
f(arr, 5, 5)
10
20
30
40
50
0
1
2
3
4
index >= size
return
null
base case

Search in Array

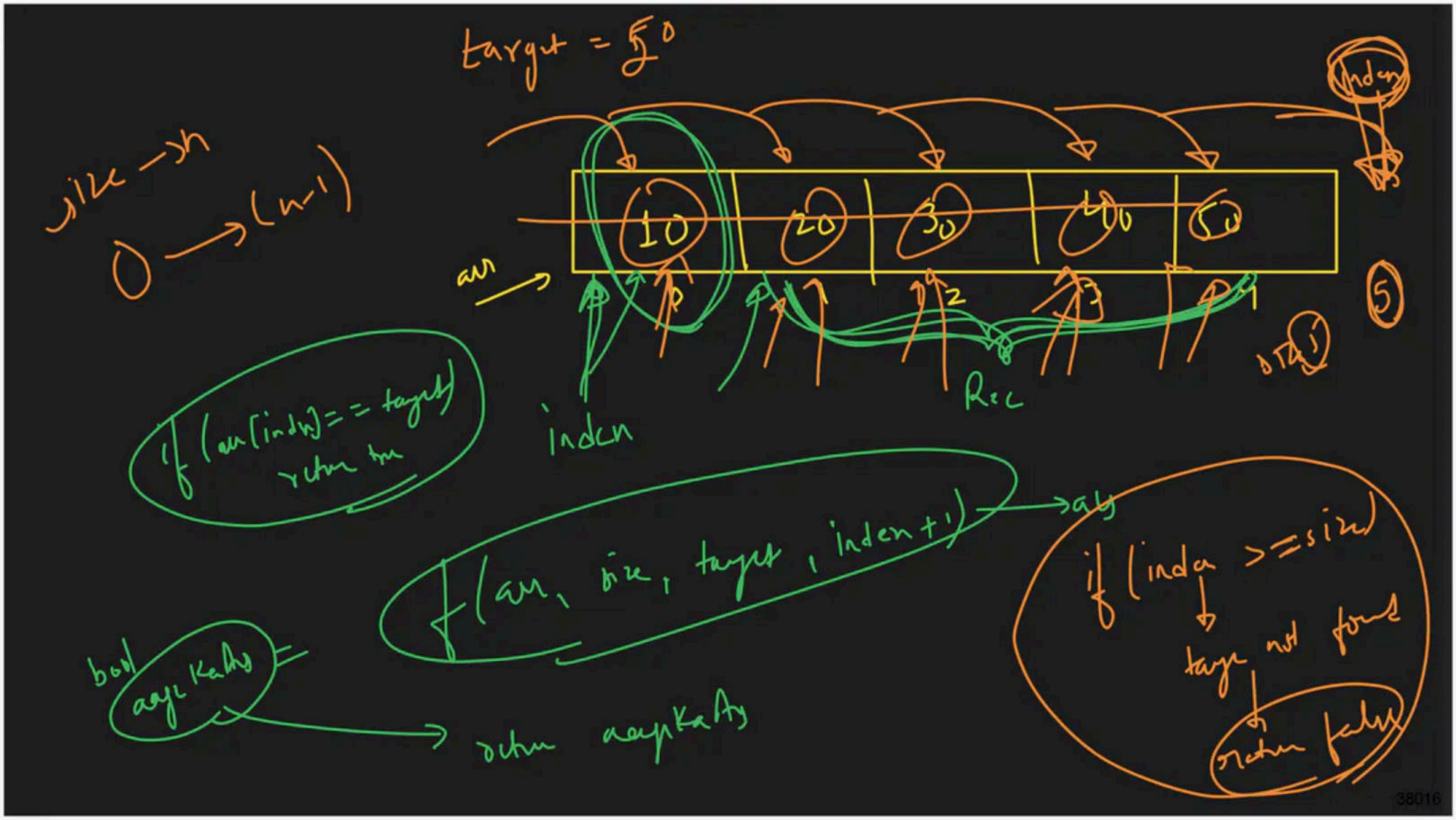
target = 50



T/F

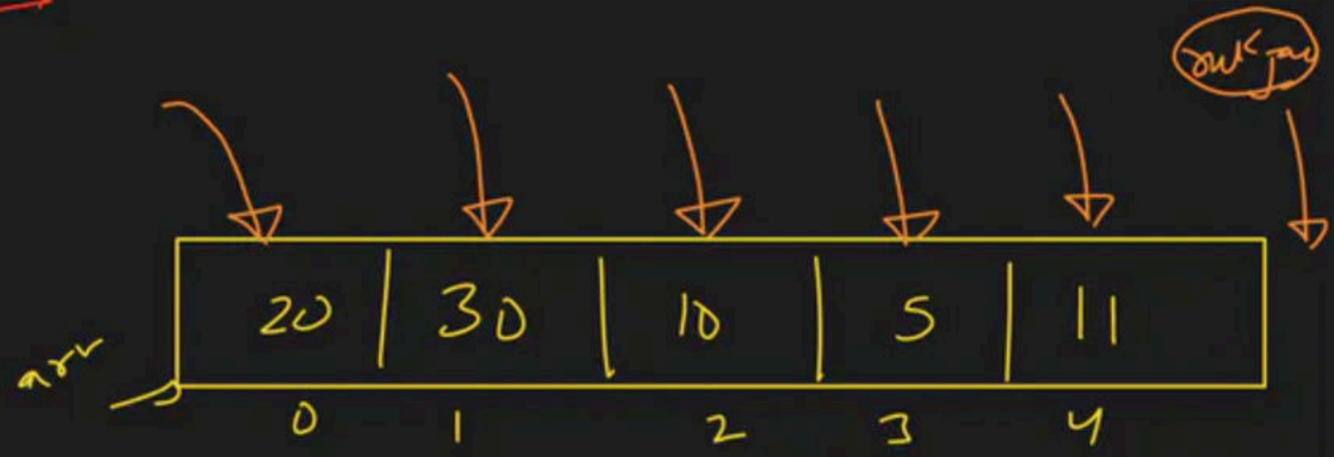
milne pr
↳ return
true

entire array
traverse hogya → B.C
↓
return false


target = 50
size -> n
0 -> (n-1)
arr
10
20
30
40
50
index
index
Rec
5
if (arr[index] == target)
return true
f(arr, size, target, index+1)
bool
arrKaAy
return arrKaAy
if (index >= size)
target not found
return false


if (index >= size)
return false
if (arr[index] == target)
return true
int ans = search(arr, size, target, index+1);
return ans;
search(arr, 3, 30, 1)
search(arr, 3, 30, 2)
search(arr, 3, 30, 3)
main → search(arr, 3, 30, 0)
array
size
target
index
arr
10
20
30
0
1
2
target = 30

min no in Array



B.C

```
if (index >= size)
{
    return;
}
```

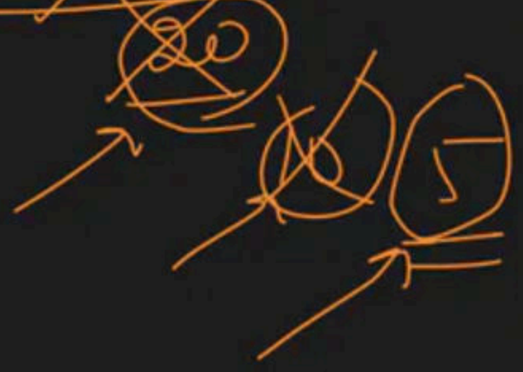
Recursion

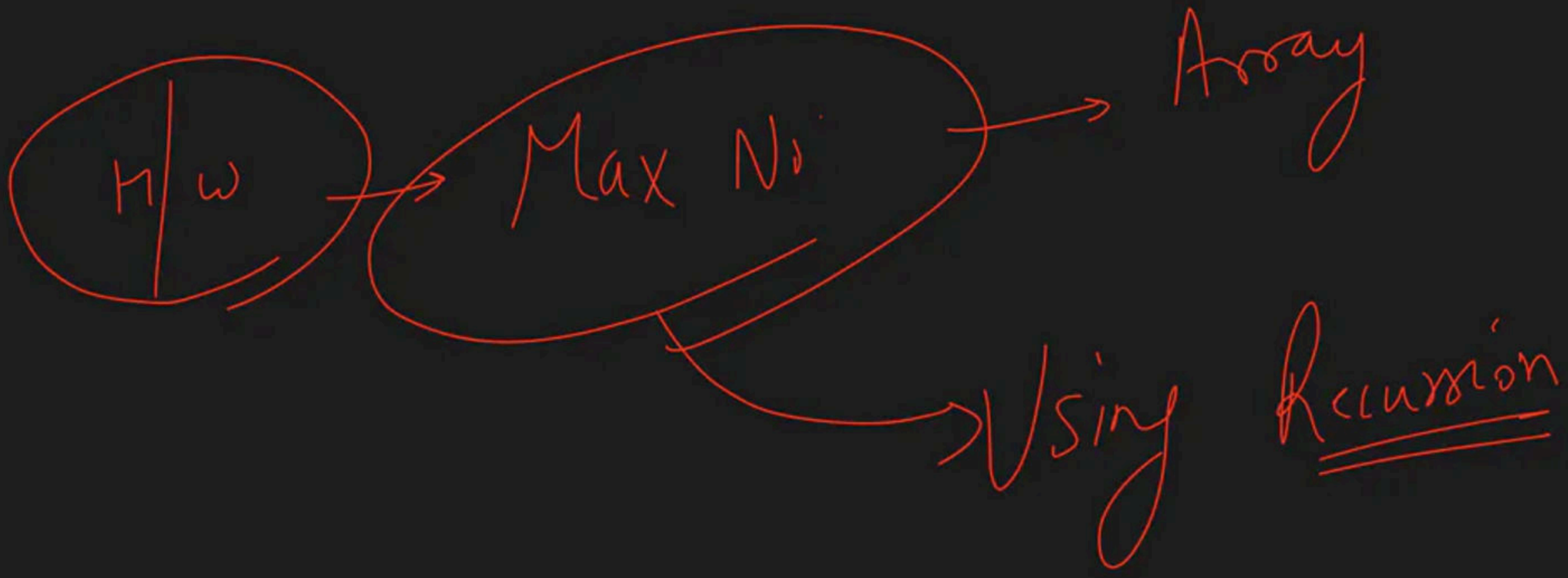
mini = min (mini, arr[index])

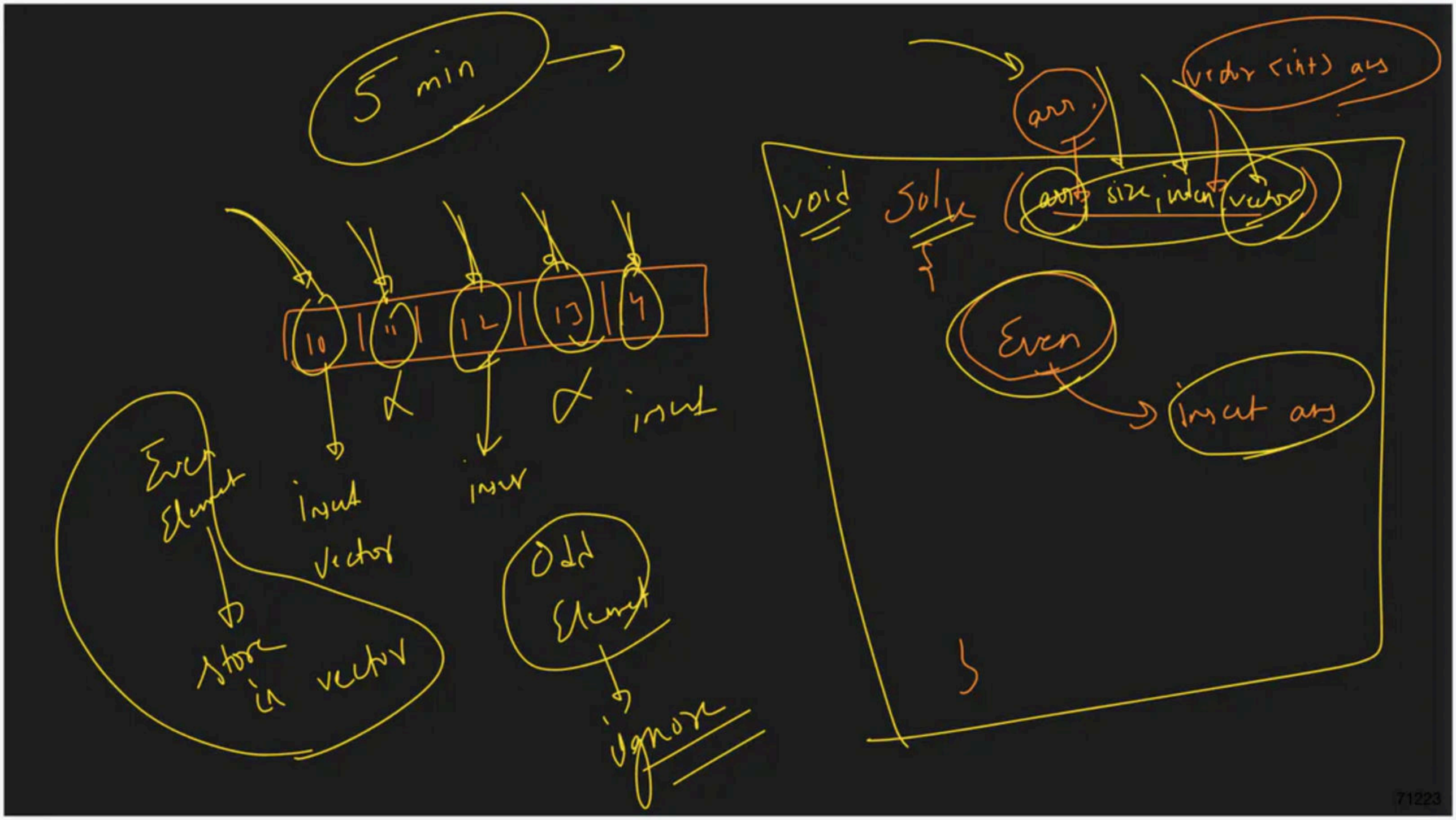
R.R

f (arr, size, index+1, mini)

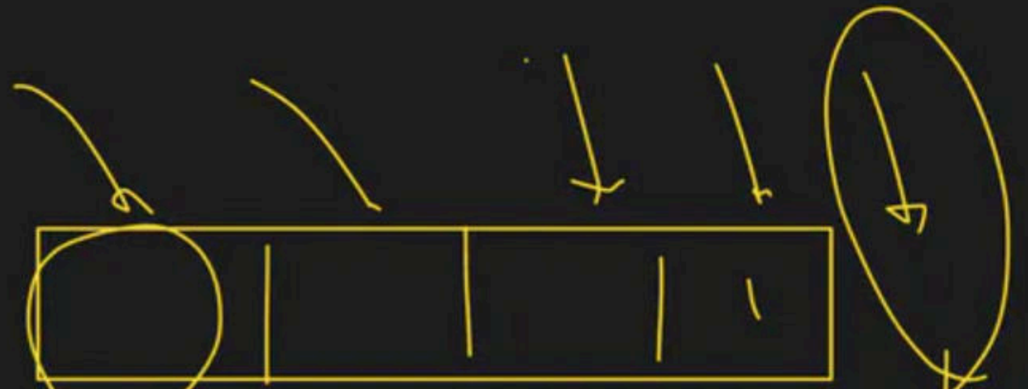
int mini = INT_MAX




H/W
Max No
Array
Using Recursion

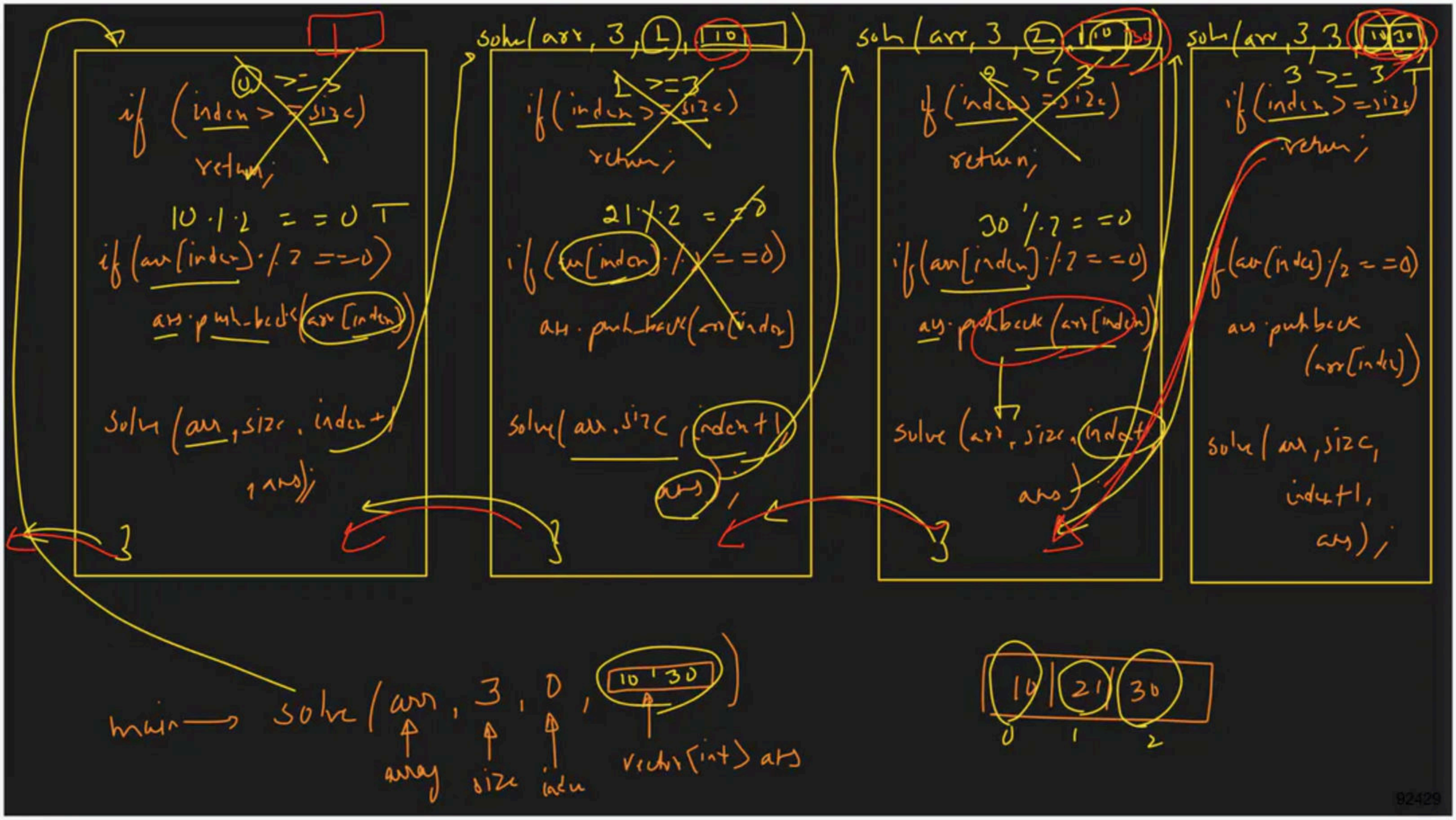

5 min
10
11
12
13
4
input
input
input
Even Element
input Vector
store in vector
Odd Element
ignore
arr
vector<int> arr
void
Sol
(arr, size, index, vector)
{
Even
input arr
}

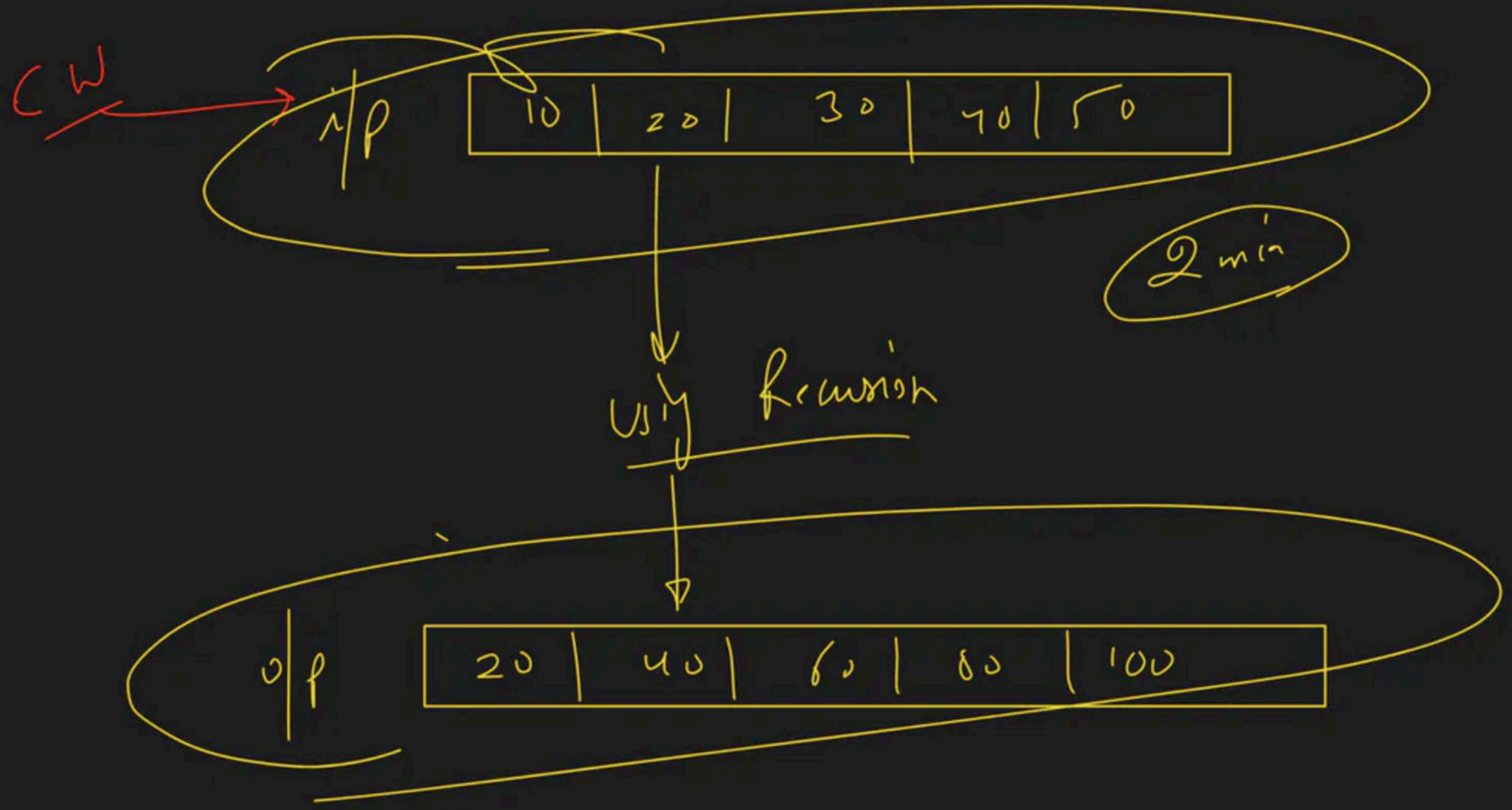
```
//R.R
sol(arr, size, index, ans);

//proxy
if(arr[index]%2==0)
{
  ans.push_back(arr[index])
}
```



```
push_back
```

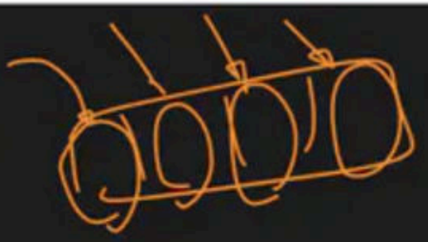
```
B.C
if(index >= size)
{
  return
}
```


if (index >= size)
return;
10 % 2 == 0 T
if (arr[index] % 2 == 0)
ans.push_back(arr[index])
solve(arr, size, index+1, ans);
solve(arr, 3, 1, 10)
if (index >= size)
return;
21 % 2 == 0
if (arr[index] % 2 == 0)
ans.push_back(arr[index])
solve(arr, size, index+1, ans);
solve(arr, 3, 2, 10 30)
if (index >= size)
return;
30 % 2 == 0
if (arr[index] % 2 == 0)
ans.push_back(arr[index])
solve(arr, size, index+1, ans);
solve(arr, 3, 3, 10 30)
if (index >= size)
return;
if (arr[index] % 2 == 0)
ans.push_back(arr[index])
solve(arr, size, index+1, ans);
main → solve(arr, 3, 0, 10 30)
array
size
index
vector<int> ans
10 21 30
0 1 2


CW
i/p
10 | 20 | 30 | 40 | 50
2 min
Using Recursion
o/p
20 | 40 | 60 | 80 | 100

```
void doubleArr(arr, size, index)
{
    // Base Case
    if (index >= size)
        return;

    // process
    arr[index] = 2 * arr[index];

    // RR
    doubleArr(arr, size, index+1);
}
```

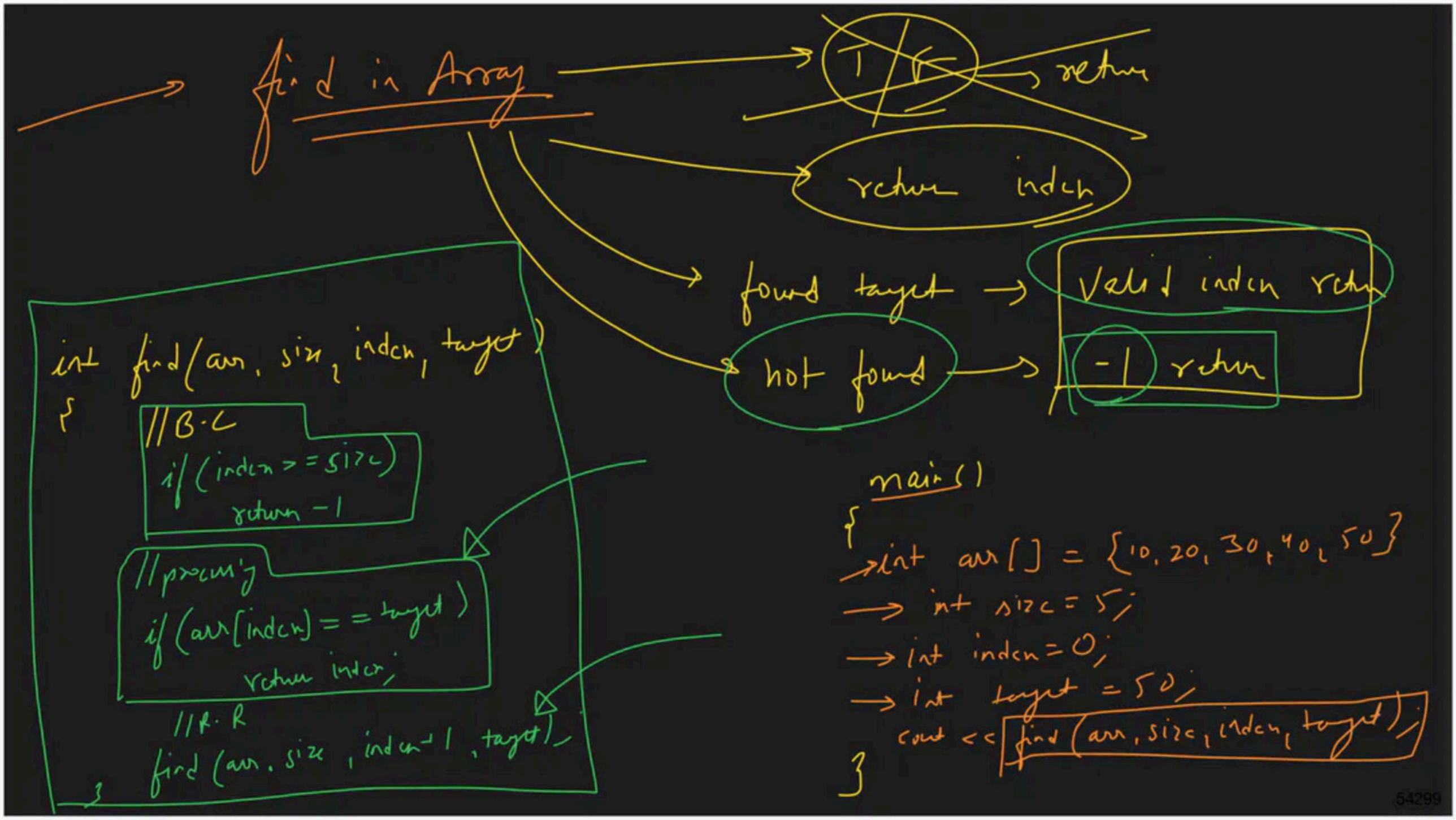
Max no in Array → 2 min



```
main()
{
  arr[] = {          }
  size =
  index = 0
  int maxi = INT_MIN;
  findMax( arr, size, index, maxi);
}
```

by ref

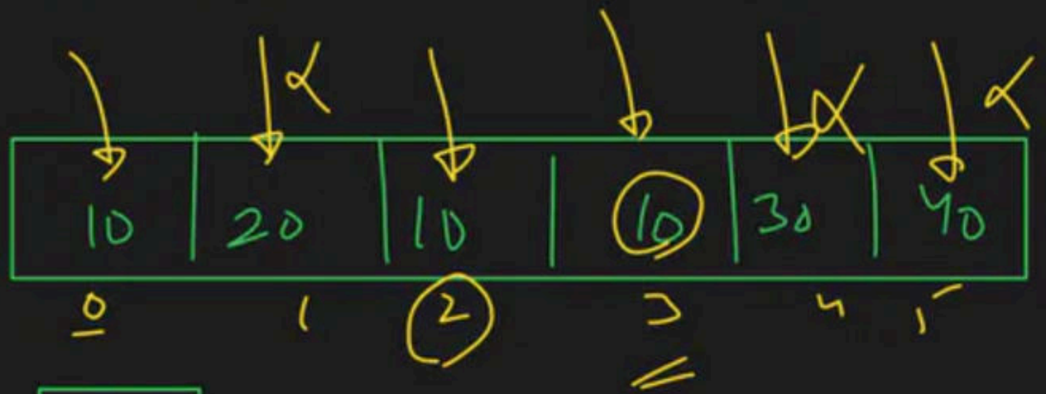
```
void findMax (arr, size, index, maxi)
{
  // base Case
  if (index >= size)
    return;

  // proving
  maxi = max(maxi, arr[index])

  // R.R
  findMax (arr, size, index+1, maxi)
}
```


find in Array
T
F
return
return index
found target
Valid index return
not found
-1 return
int find(arr, size, index, target)
{
//B.C
if(index >= size)
return -1
//processing
if(arr[index] == target)
return index;
//R.R
find(arr, size, index+1, target);
}
main()
{
int arr[] = {10, 20, 30, 40, 50}
int size = 5;
int index = 0;
int target = 50;
cout << find(arr, size, index, target);
}

Print index of all Occurrence of target in Array

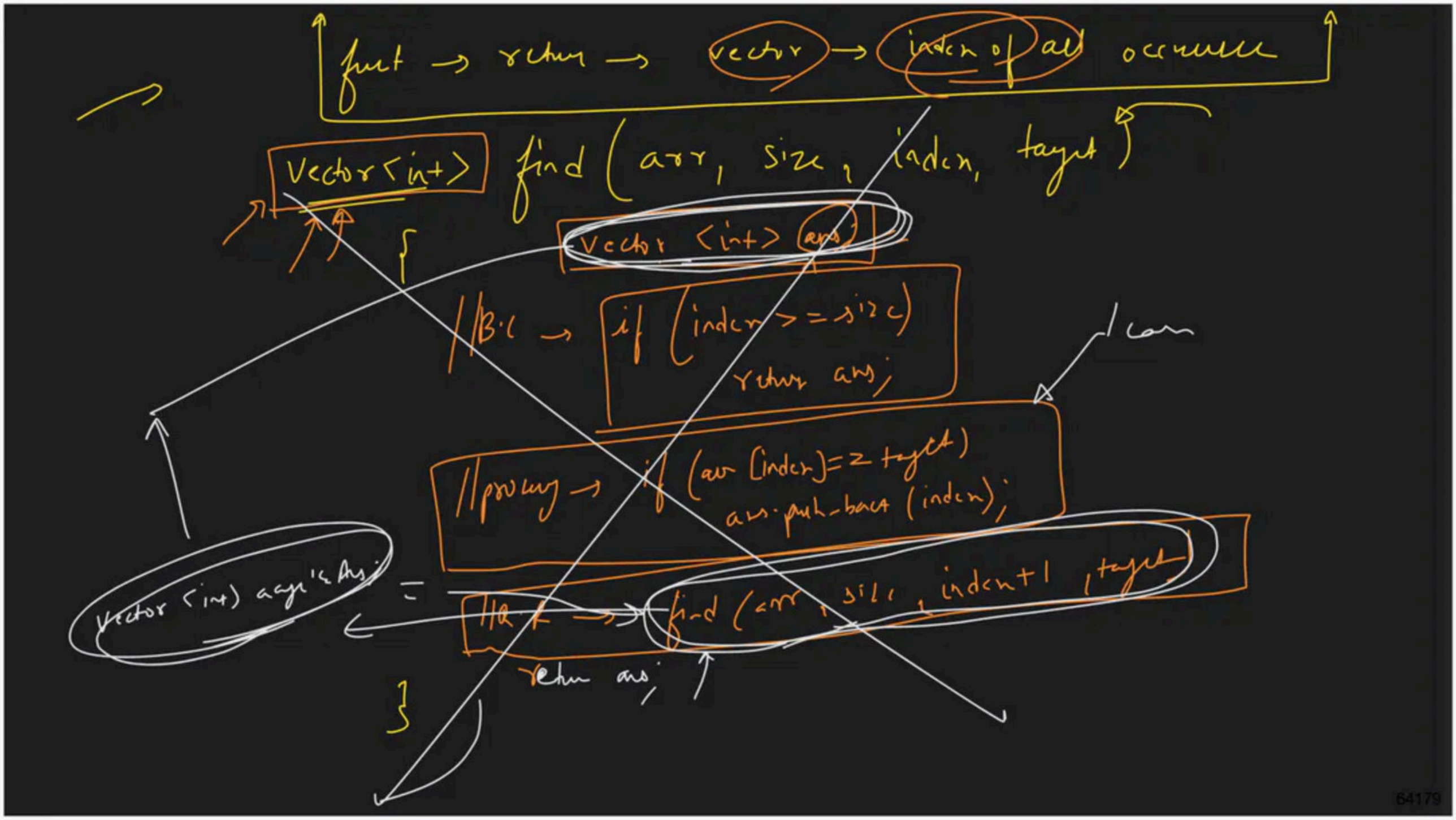
2 min

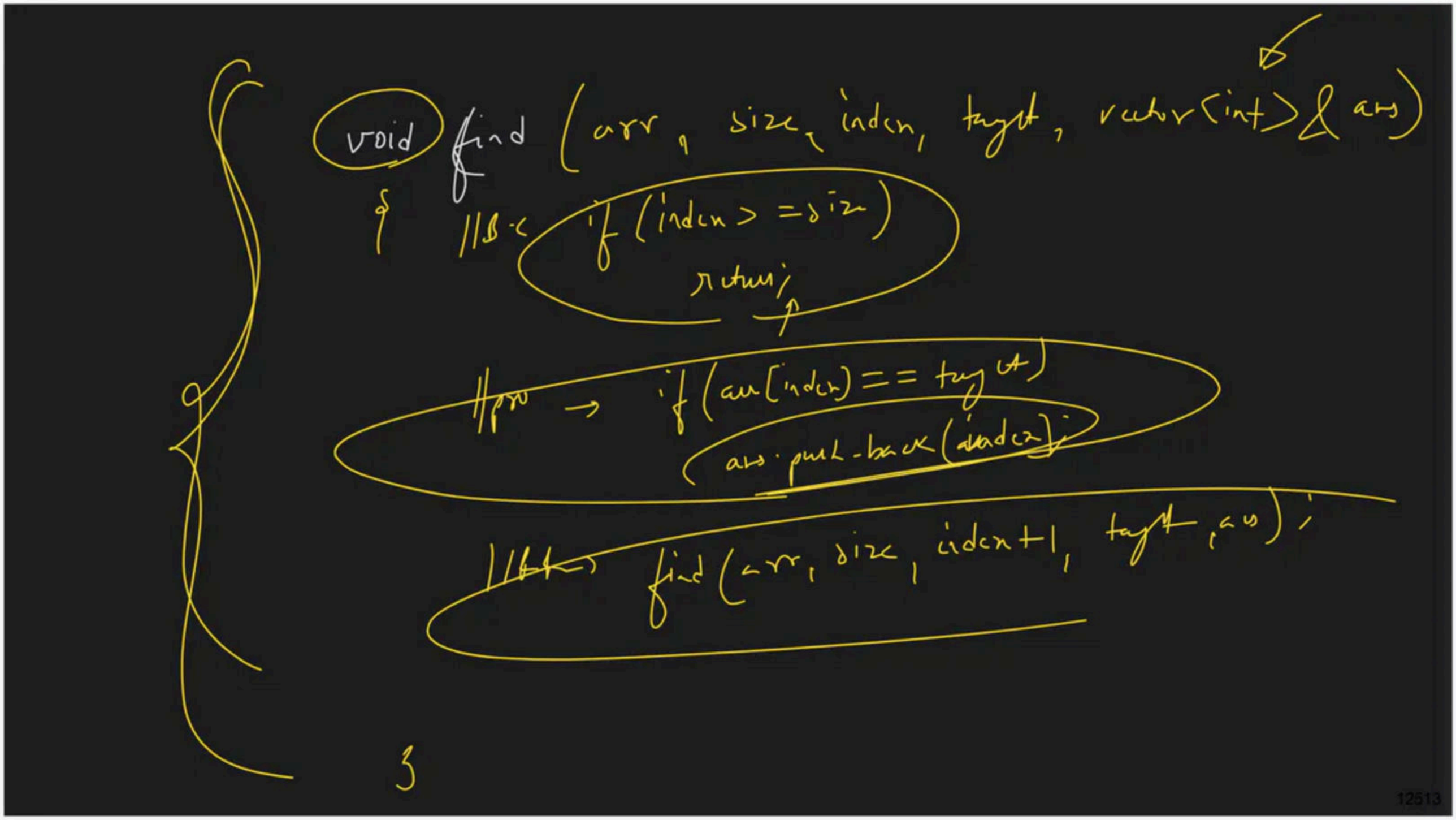
```
void find(arr, size, index, target)
{
    //B.C →  if(index >= size)
               return;

    //process → if(arr[index] == target)
                  cout << index;

    //R.R → find(arr, size, index+1, target);
}
```



target → 10

0 | p → 0 2 3


func → return → vector → index of all occurrence
vector<int> find(arr, size, index, target)
vector<int> ans;
//B.C → if (index >= size)
return ans;
//proving → if (arr[index] == target)
ans.push_back(index);
vector<int> ans = //R.C → find(arr, size, index+1, target);
return ans;
}
I can


void find (arr, size, index, target, vector<int>& ans)
{
//B.C if (index >= size)
return;
//pro → if (arr[index] == target)
ans.push_back(index);
//Ask → find (arr, size, index+1, target, ans);
3

→ ~~int~~ i/p → integer

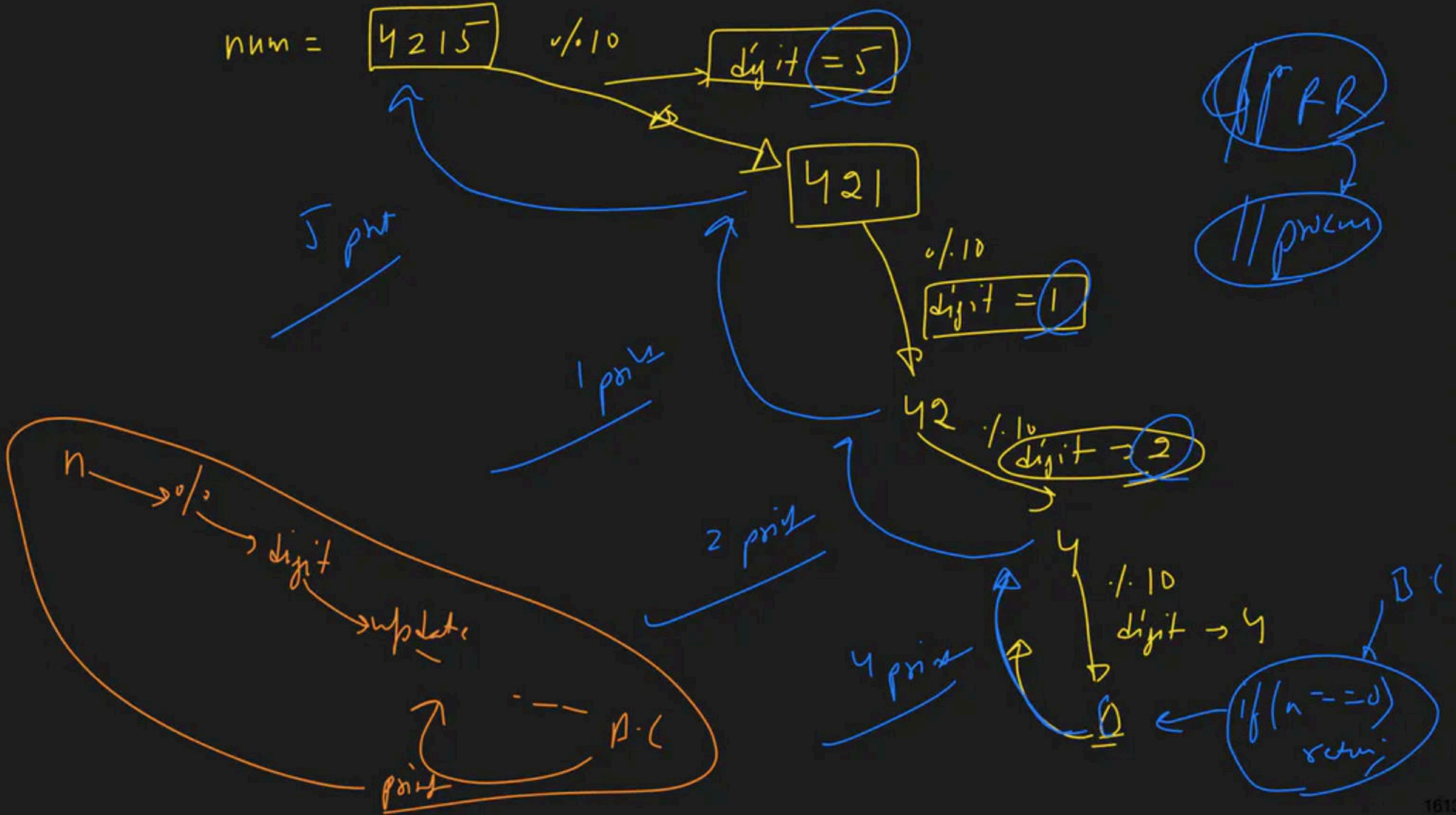
o/p → vector → digits stored

i/p → 4215

→ print digits

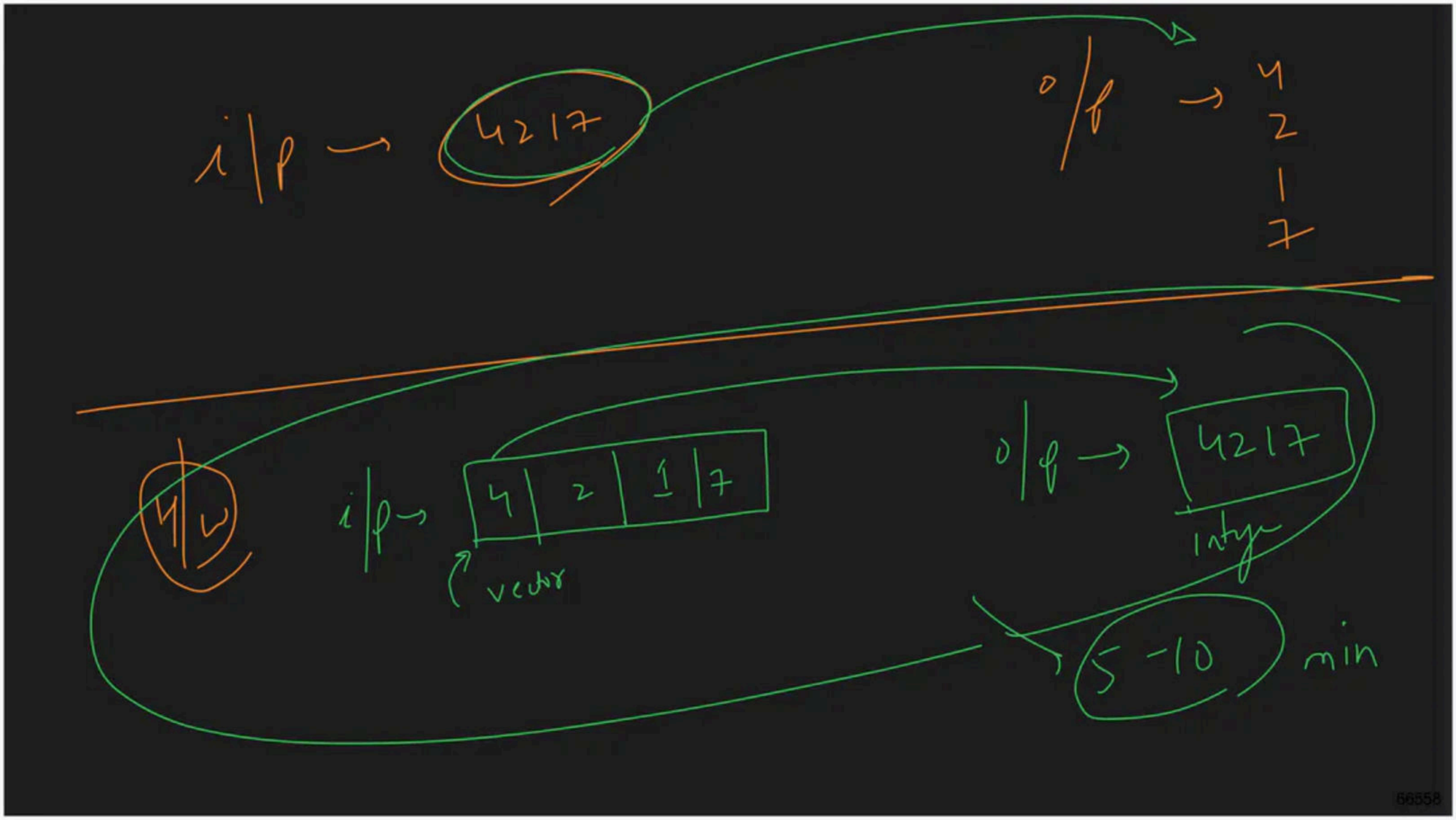
2 min

4
2
1
5



num = 4215
%10
digit = 5
421
%10
digit = 1
42
%10
digit = 2
4
%10
digit → 4
0
if(n==0)
return;
B.C
5 print
1 print
2 print
4 print
n
%
digit
update
print
B.C

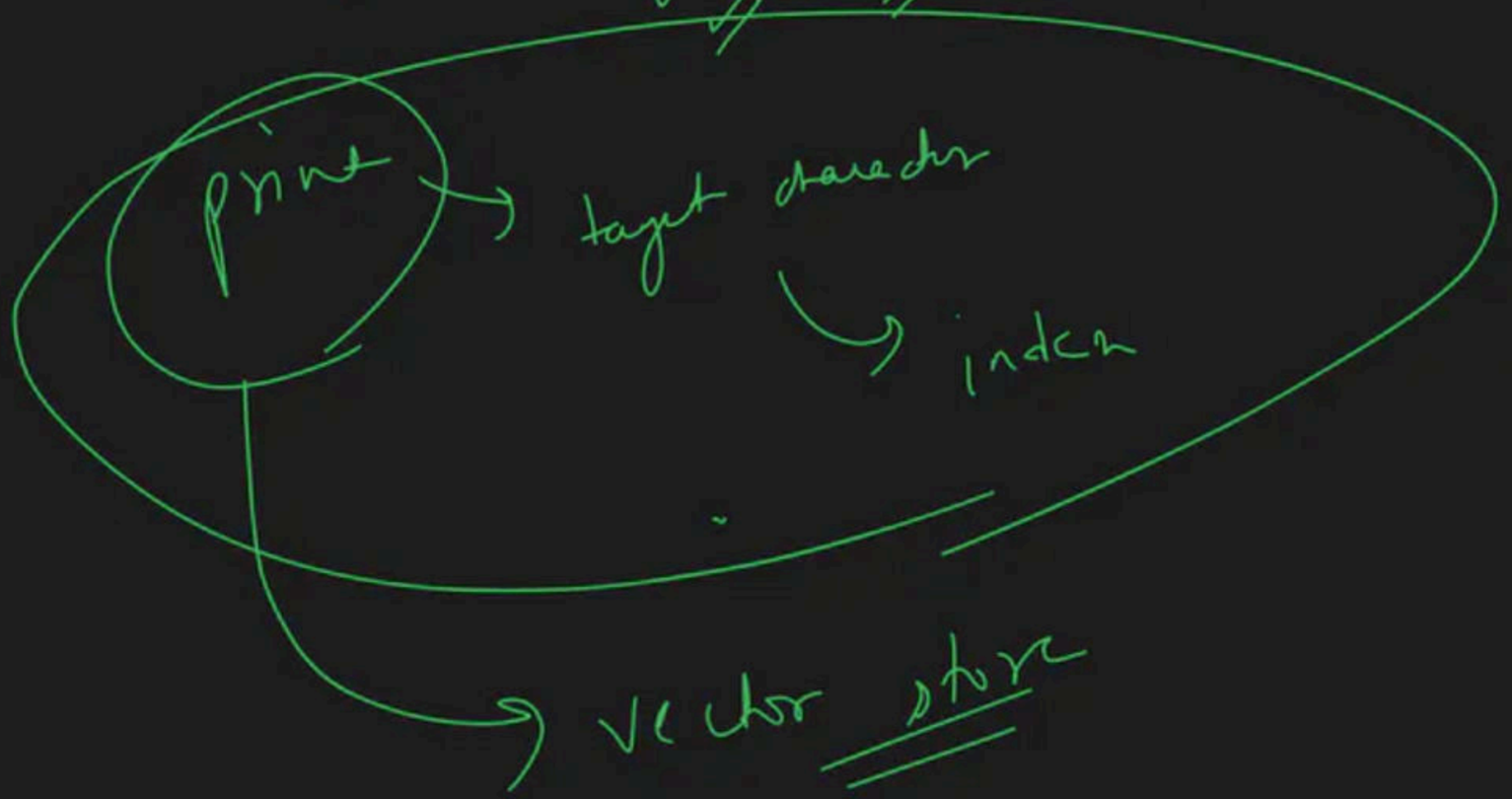
16130


i/p →
4217
o/p →
4
2
1
7
4 | 4
i/p →
4 | 2 | 1 | 7
vector
o/p →
4217
int*
5 -10
min

Friday

string = "Baabbaar"
0 1 2 3 4 5 6

text = 'a'



RE = Recorded.

① Binary Search ∈

② → iterative ∈

RE

O(log n)

O(1)

Space complexity

→ DnC

Quick Sort

⇒ Subsequences of a String → Pattern

↓ RE

↳ Naive approach w/o RE

↓

Incl Excl pattern

a b c

N -->

2^n digits → n = 3

	A	B	C
0 →	0	0	0
1 →	0	0	1
2 →	0	1	0
3 →	0	1	1
4 →	1	0	0
5 →	1	0	1
6 →	1	1	0
7 →	1	1	1

3

n		
2 →	no. of subseq.	
1	1	
..	..	
	c	c
	b	b
	bc	bc
	a	a
	ac	ac
	ab	ab
	abc	abc

1	
2	a
3	ab
4	ac
5	b
6	bc
7	c
8	abc

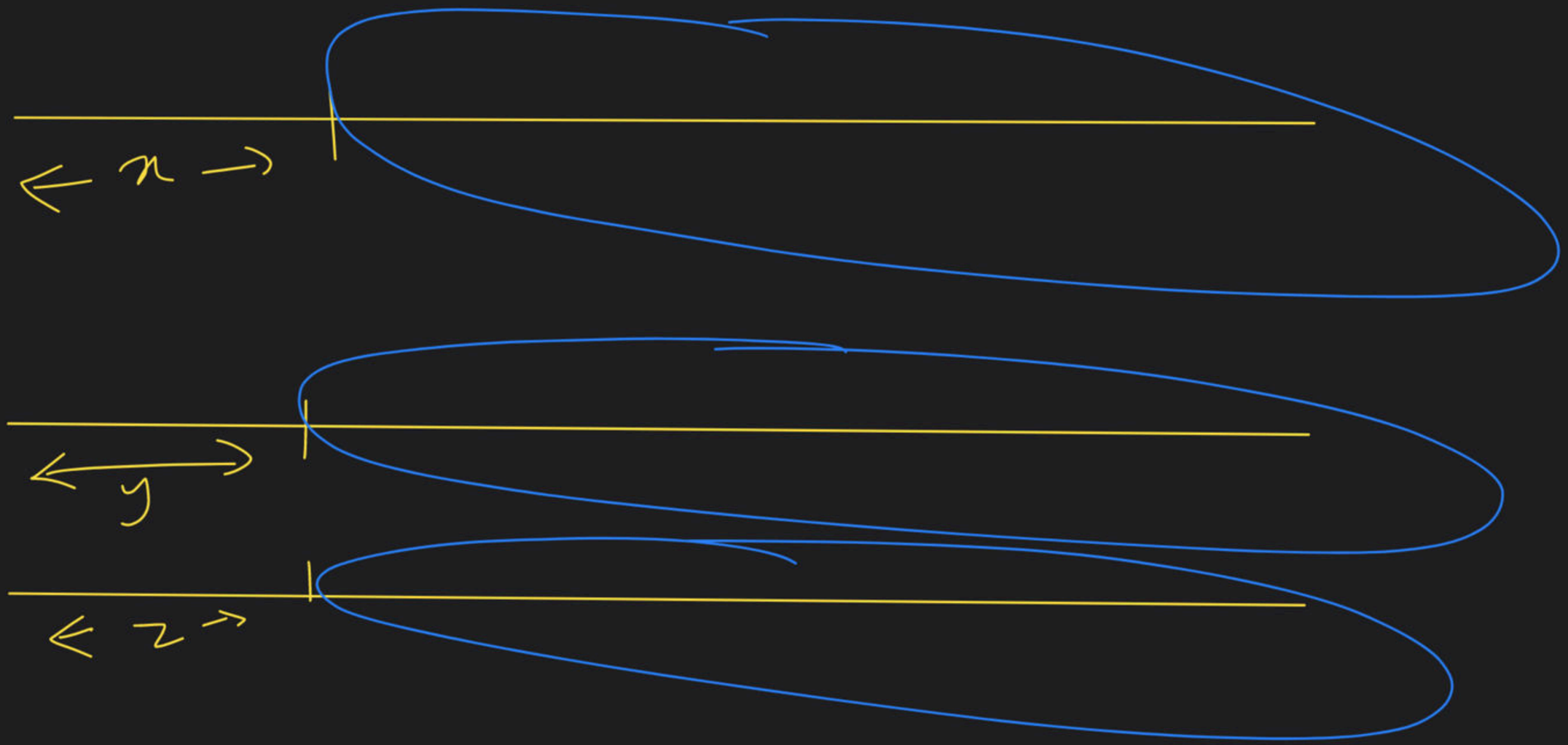
2^n

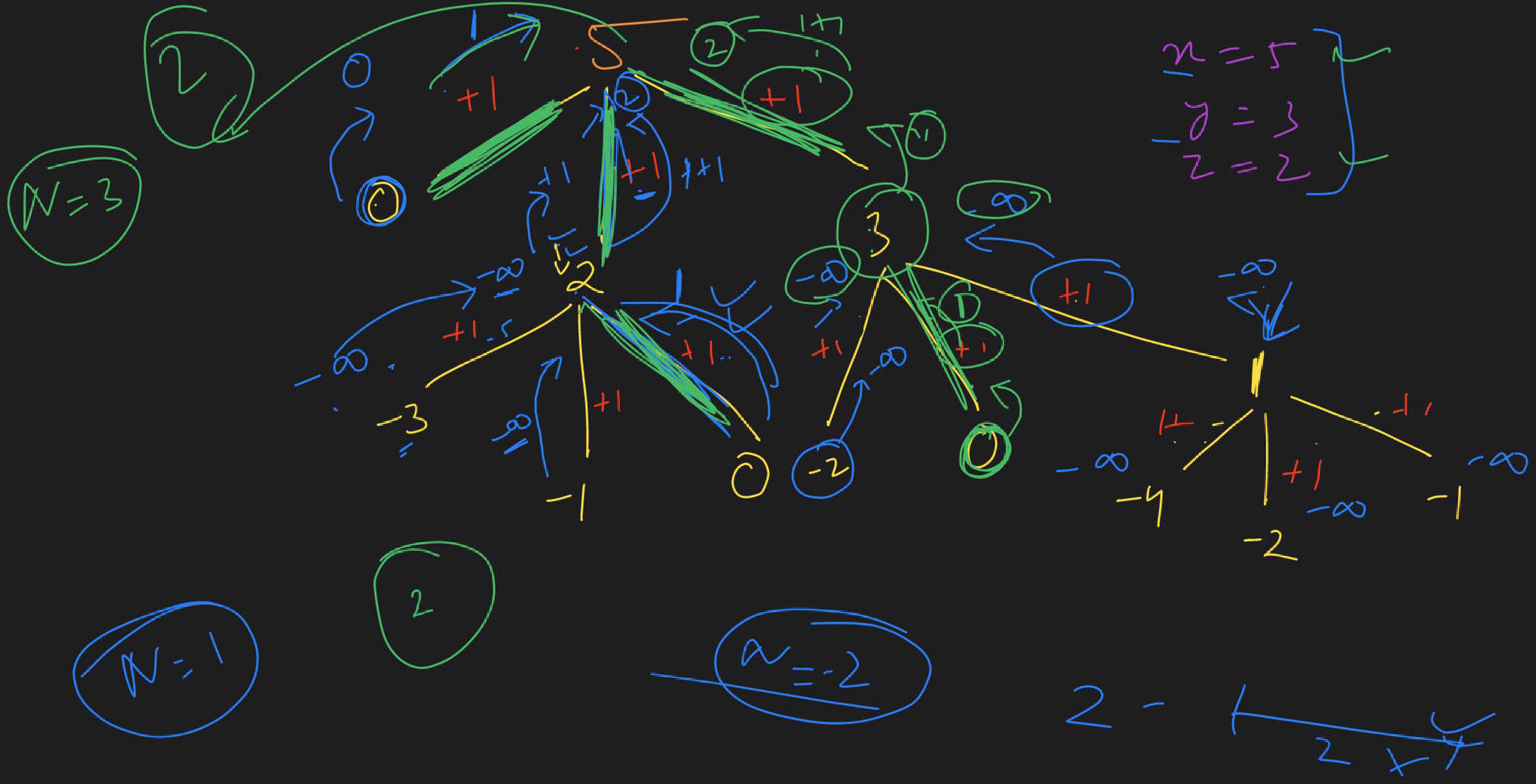
* Cut segments

* Coin change

$\Rightarrow$ $N = 5$

$x = 5, \; y = 3, \; z = 2$





N=3
x=5
y=3
z=2
N=1
a=-2

num = 0 → 0 x

num = 1 → 1 → i = 0 → 'a'

num = 2 → b → 0, x → i = 0 → i++ → h - i - 1

b → 1 → ✓ s[1] → 'b'

num = 5 → 1 → i = 0, → s[0] → a i++

0 → i = 1 → s[1] X i++

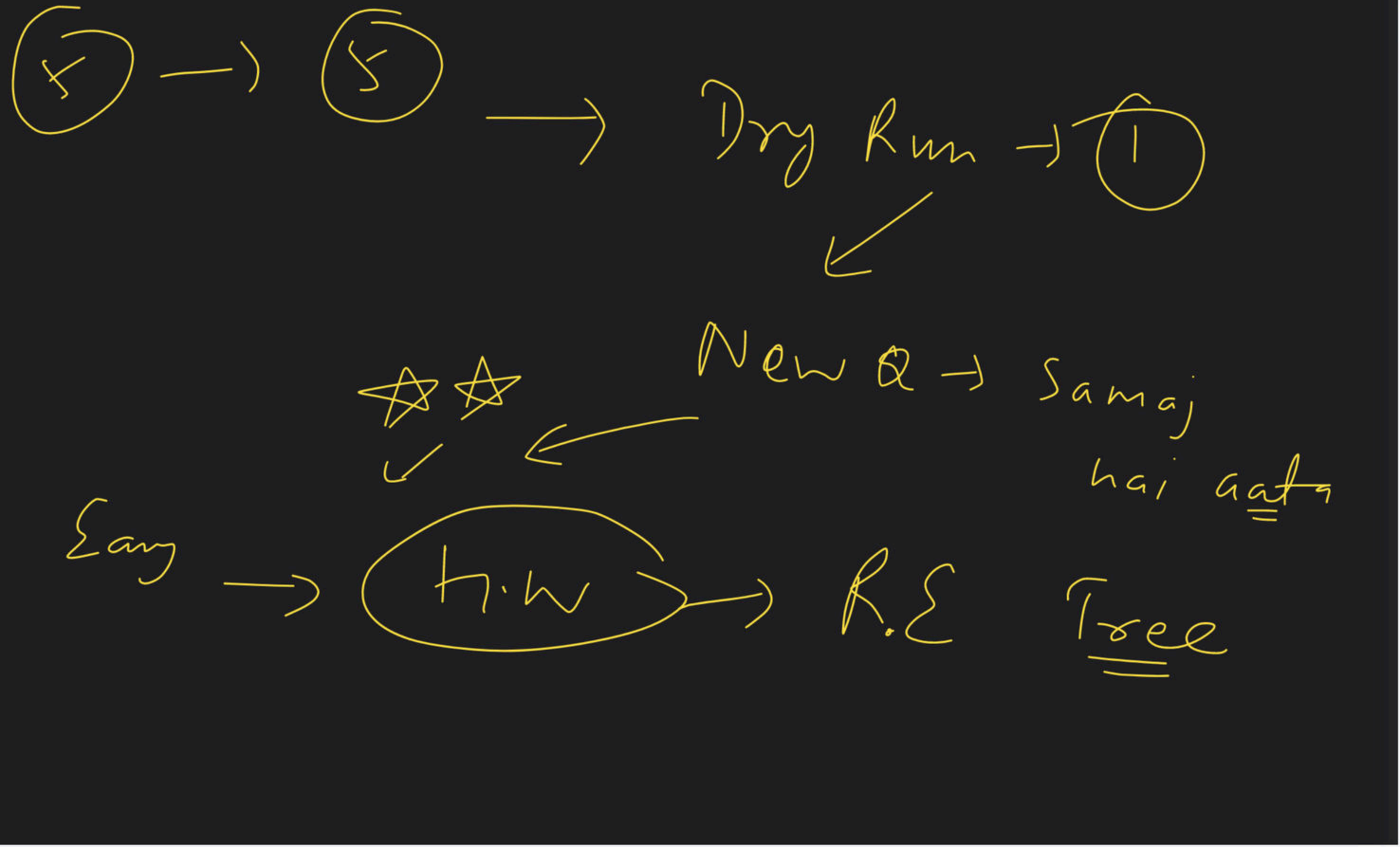
1 → i = 2 → s[2] ⇒ c

a c

s →

a	b	c
0	1	2

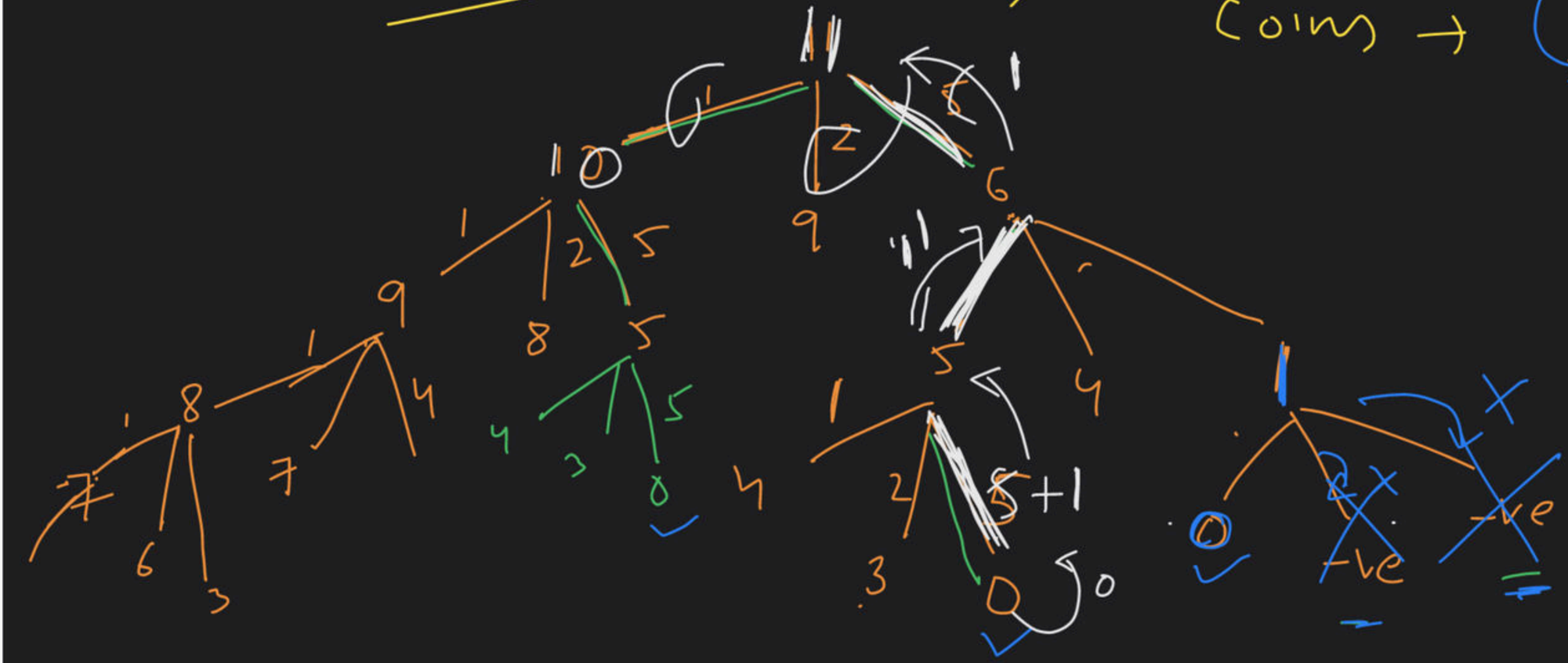
↑ 0 ↑

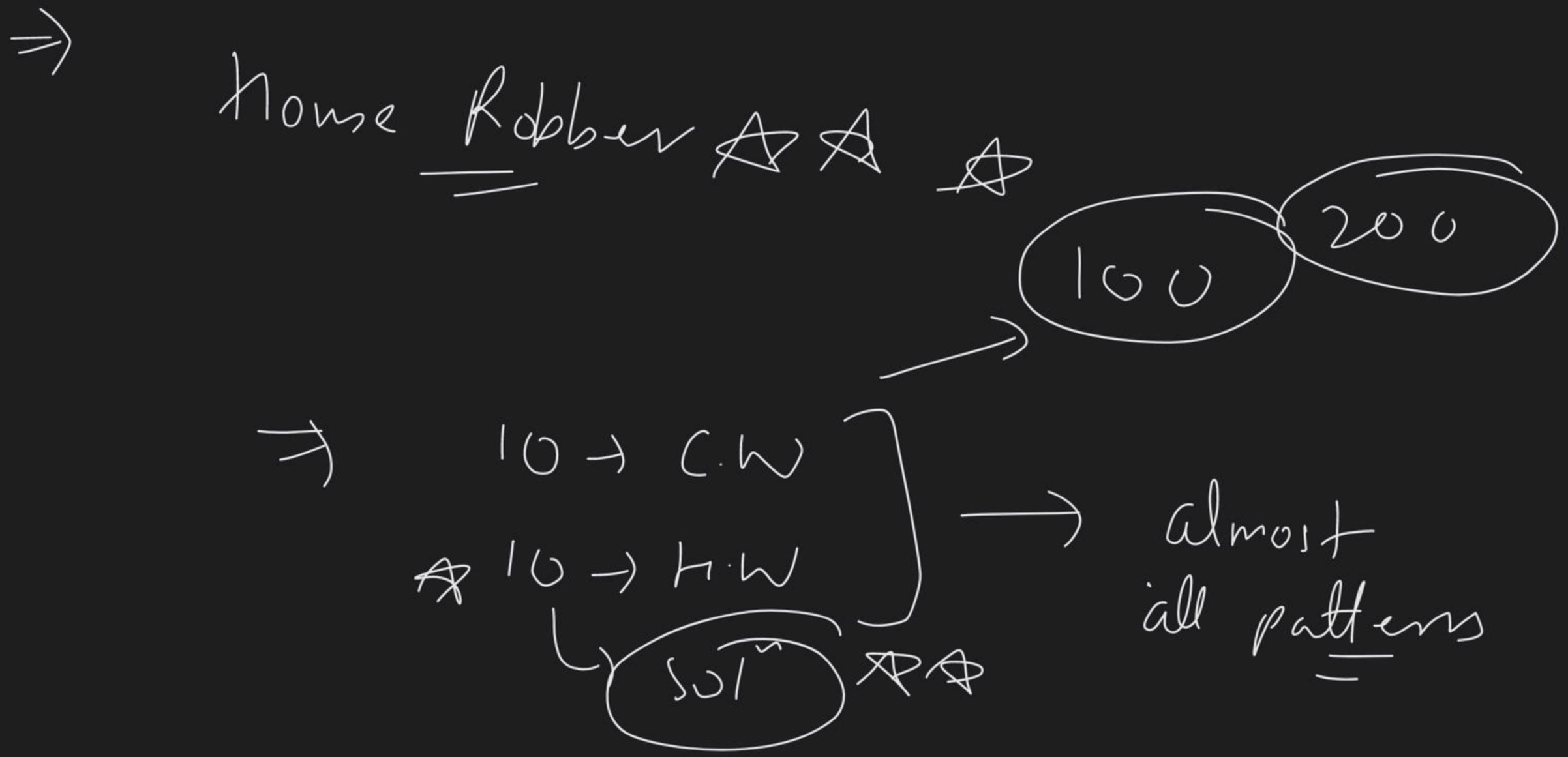

Dry Run
New Q → Samaj
hai aata
Easy
H.W
R.E
Tree

⑭ Coin Change

amt = 11

coins → 1, 2, 5




House Robber
100
200
10 → C.W
10 → H.W
SOL

Algo

=> (1) n = S.size() → ⊻

(2) no. of subsq. 2^n => Nos

(3) for i = 0 → < Nos

3.1) Convert i to Binary → ⊻ → %2

→ >>

3.2) Tab Tab 'b' → = char to be Taken