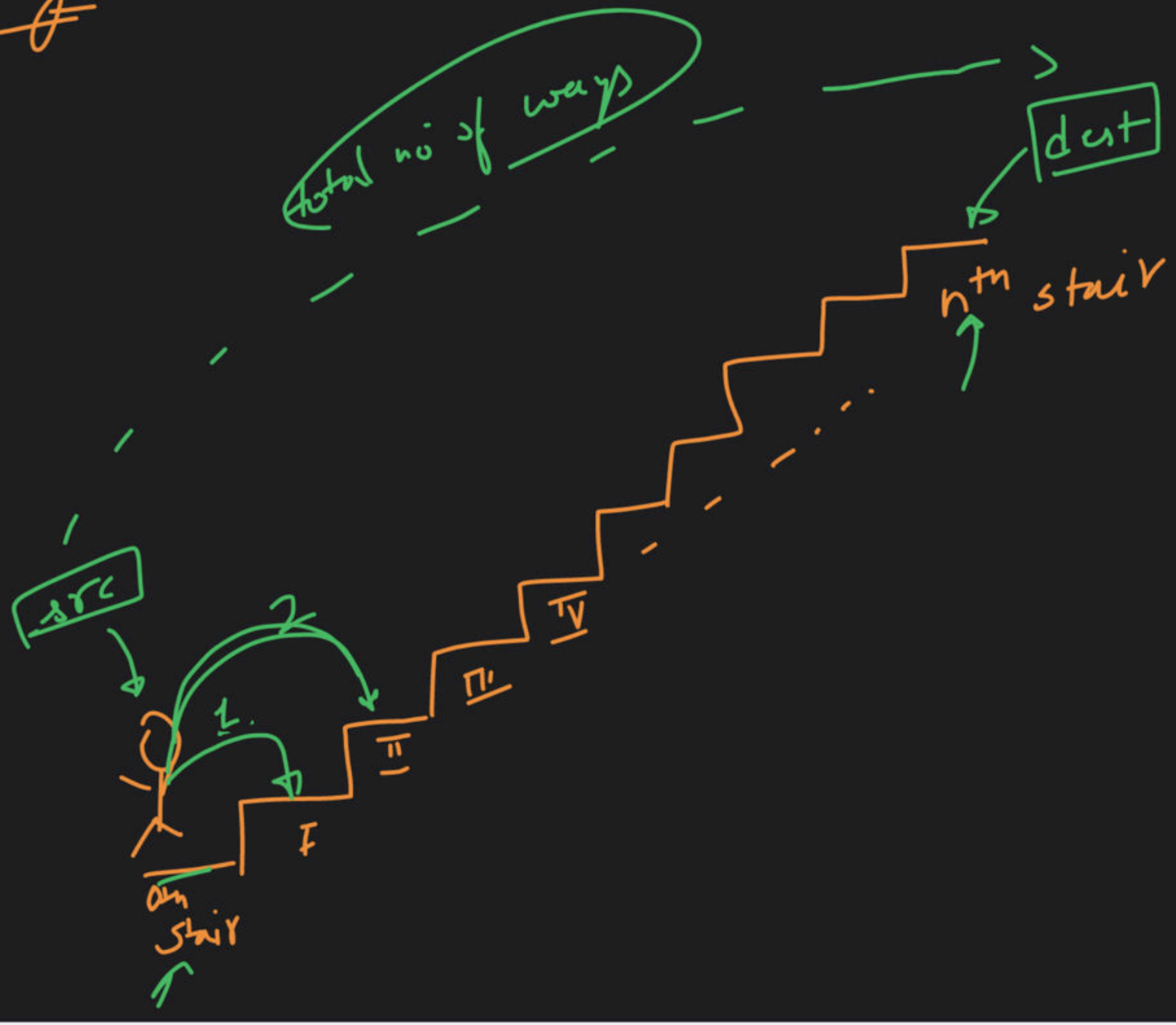
Recursion - Class 2

Special class

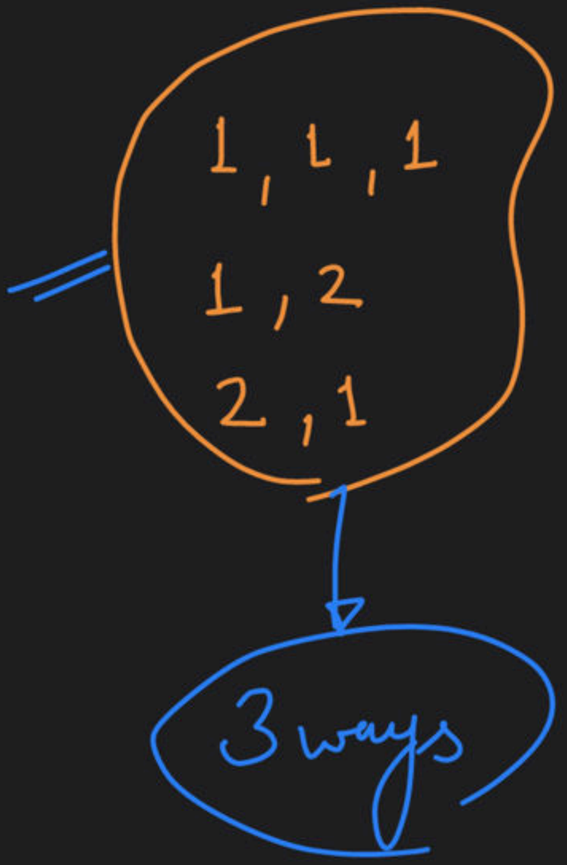
Love Babbar • Oct 10, 2023

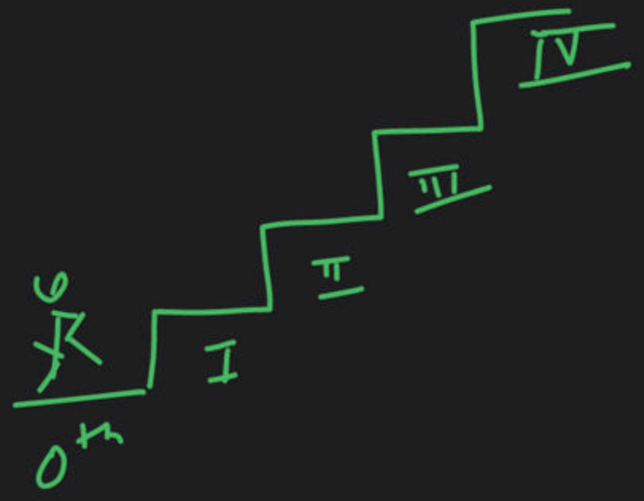
Climbing Stairs




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

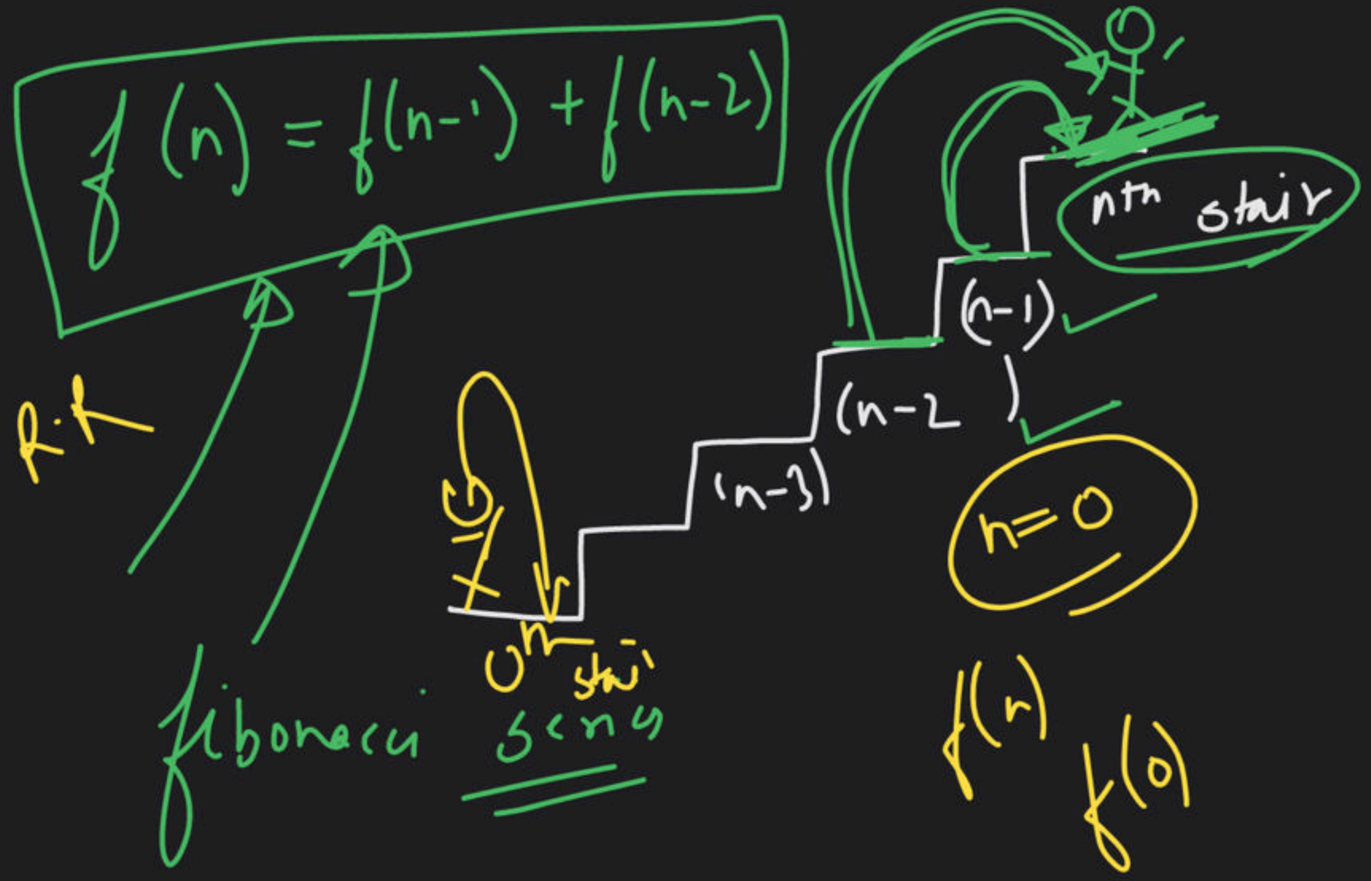
2, 2

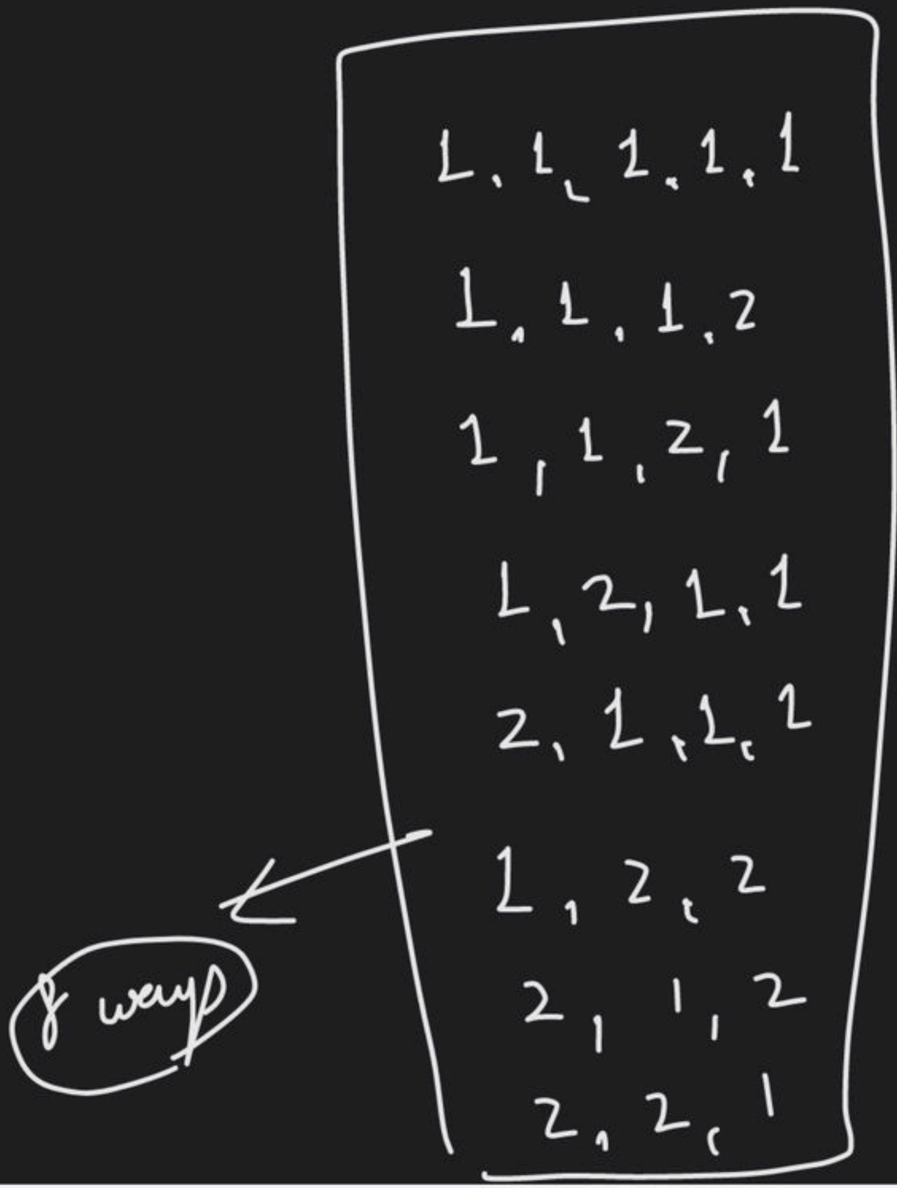
→ 5 ways

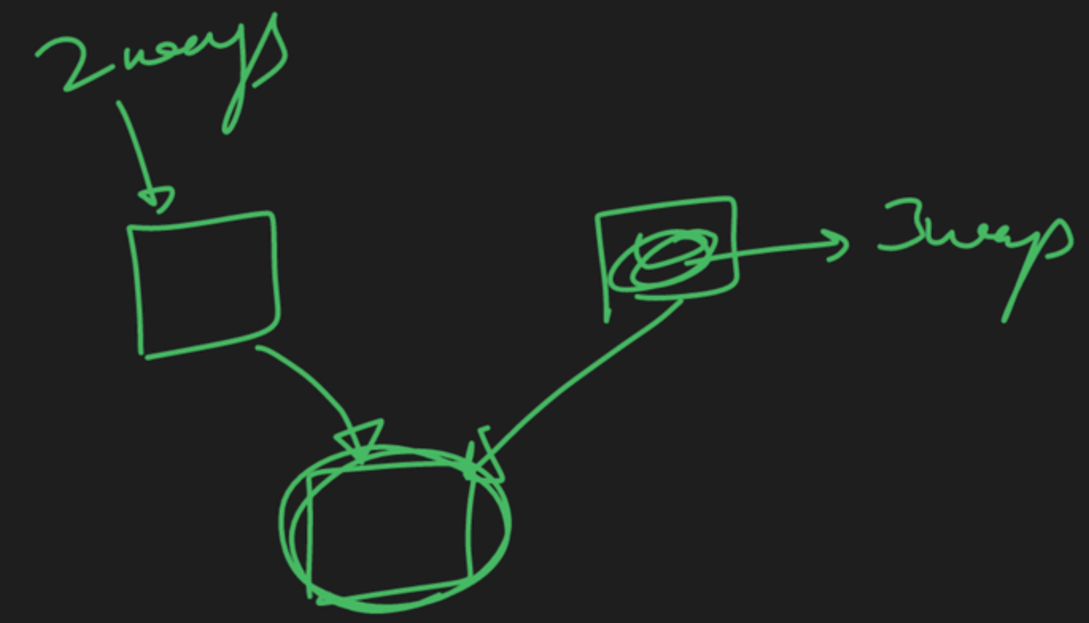
n^{th} Stair → total no of ways

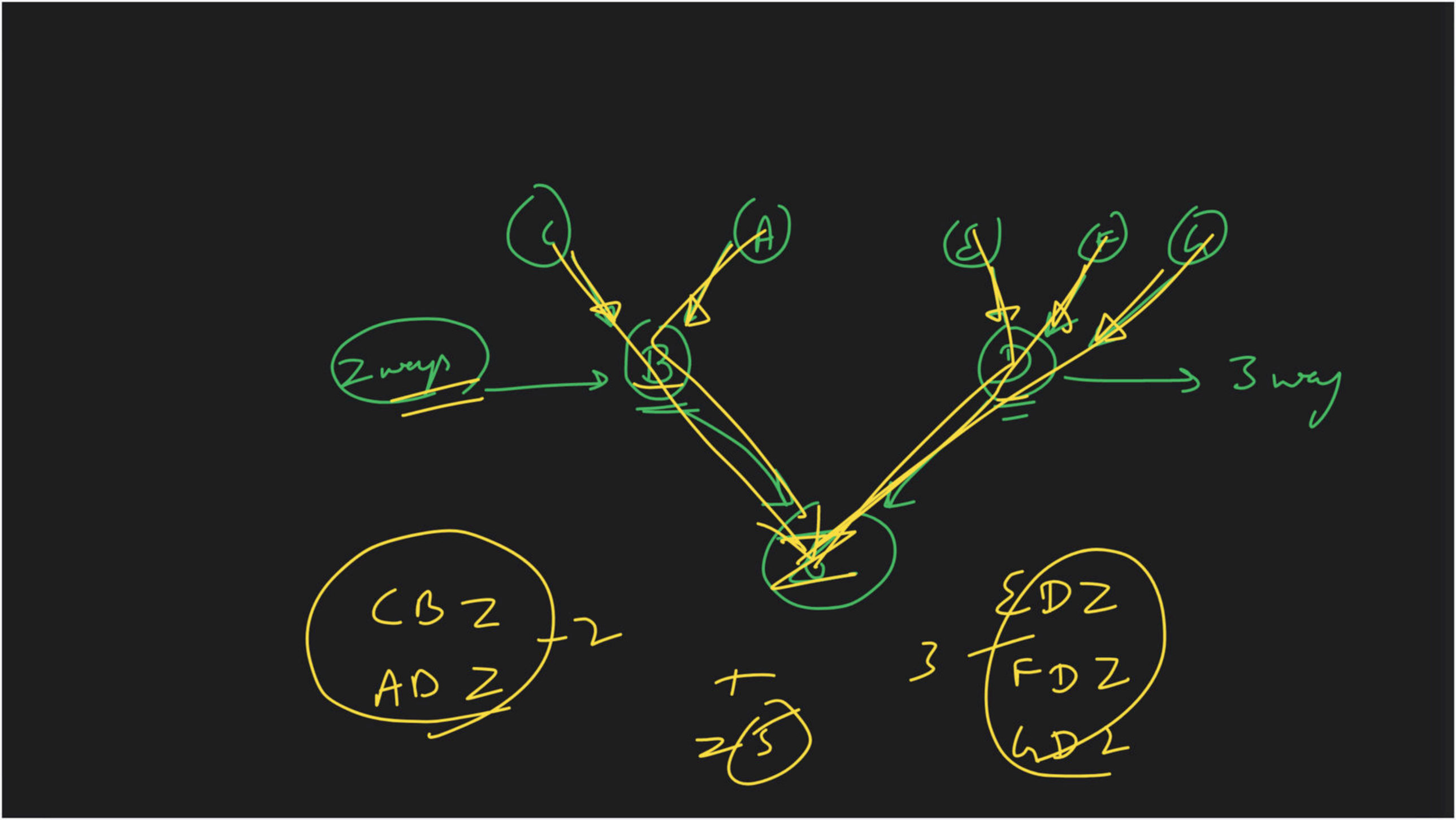
$f(n)$ → no of ways to reach n^{th} stair

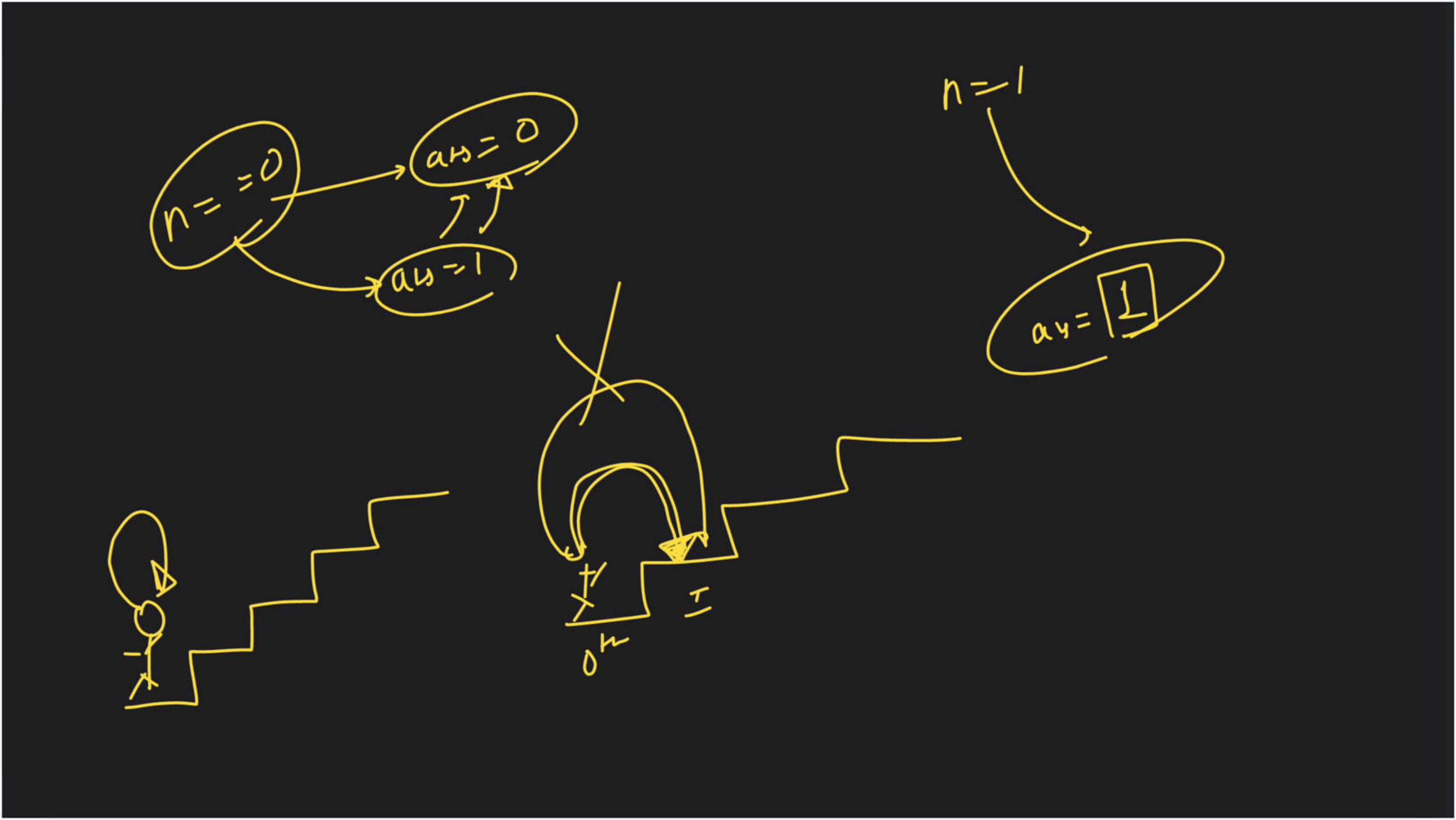
$f(n) = f(n-1) + f(n-2)$

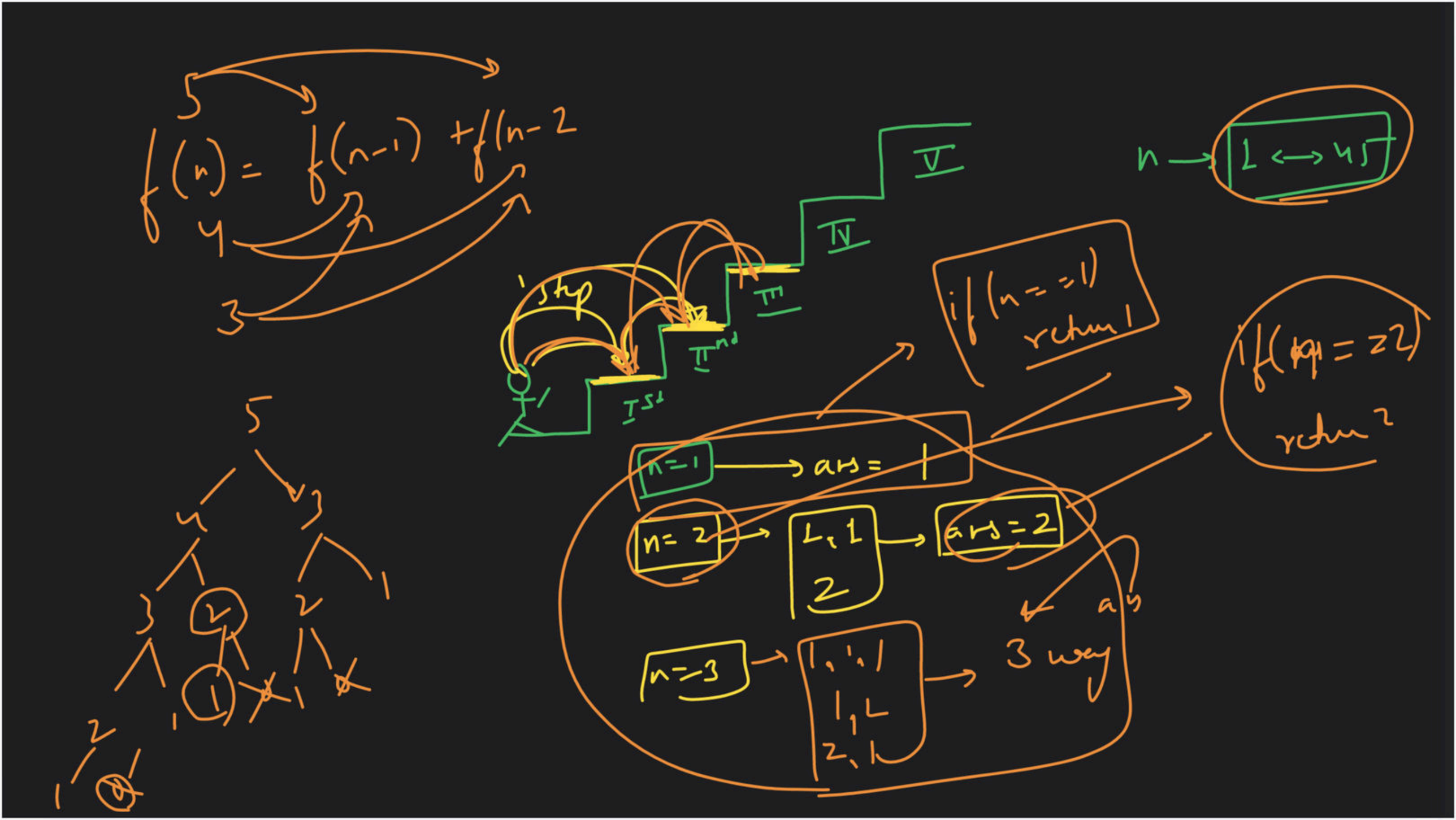


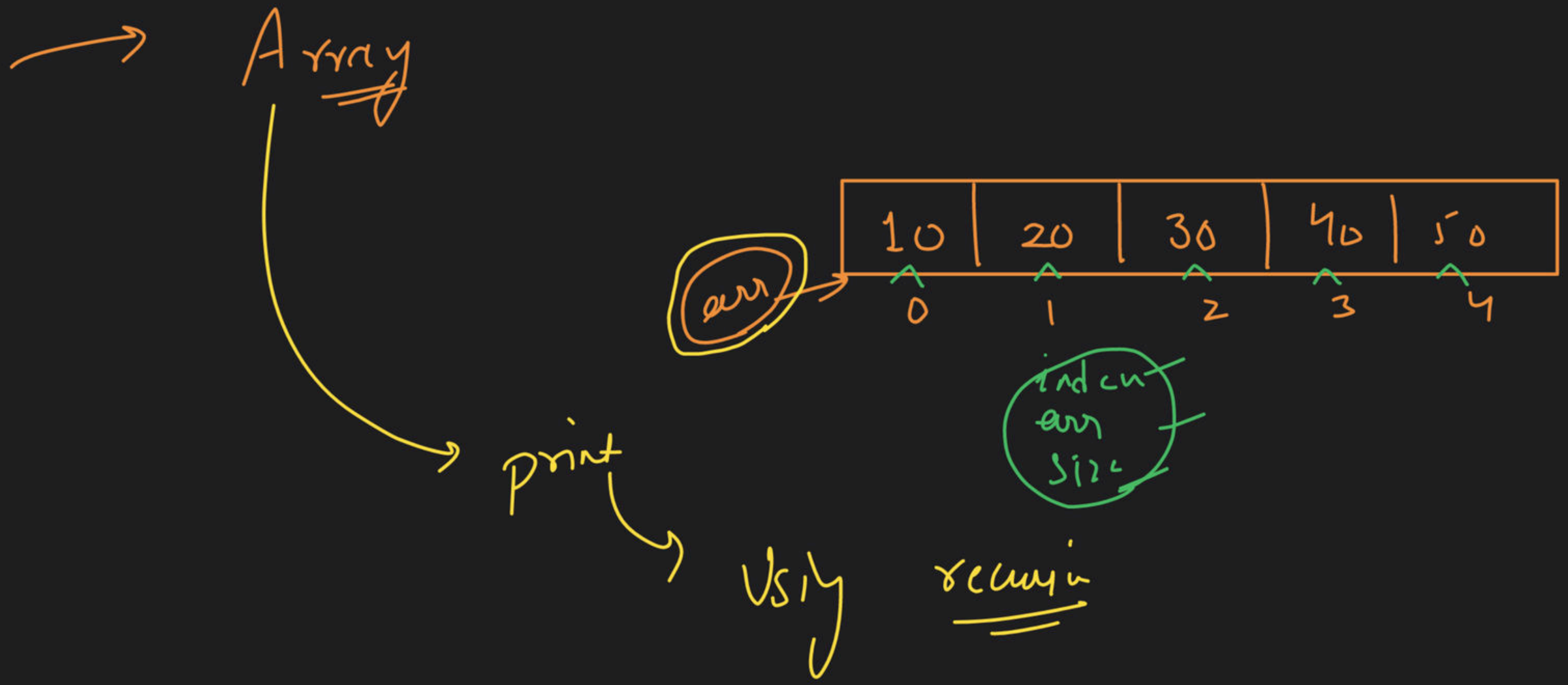


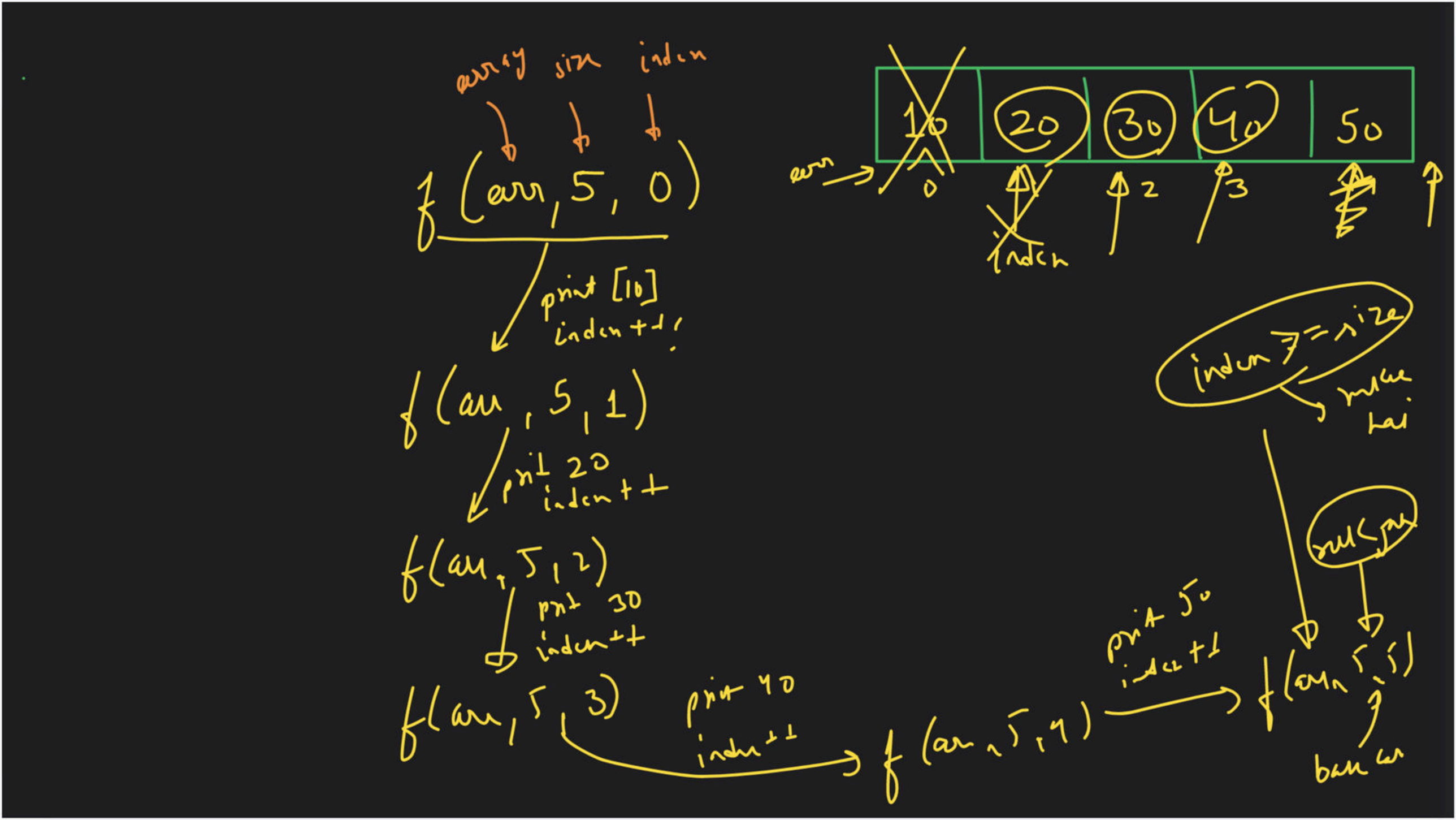

2 ways
3 ways


C
A
B
2 ways
E
F
G
D
3 way
CBZ
ADZ
+2
+
2(5)
3
EDZ
FDZ
GDZ


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


f(n) = f(n-1) + f(n-2)
5
4
3
1 step
I st
II nd
III
IV
V
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


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


array
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
arr
0
1
2
3
index
index >= size
return
sum(arr)
base case

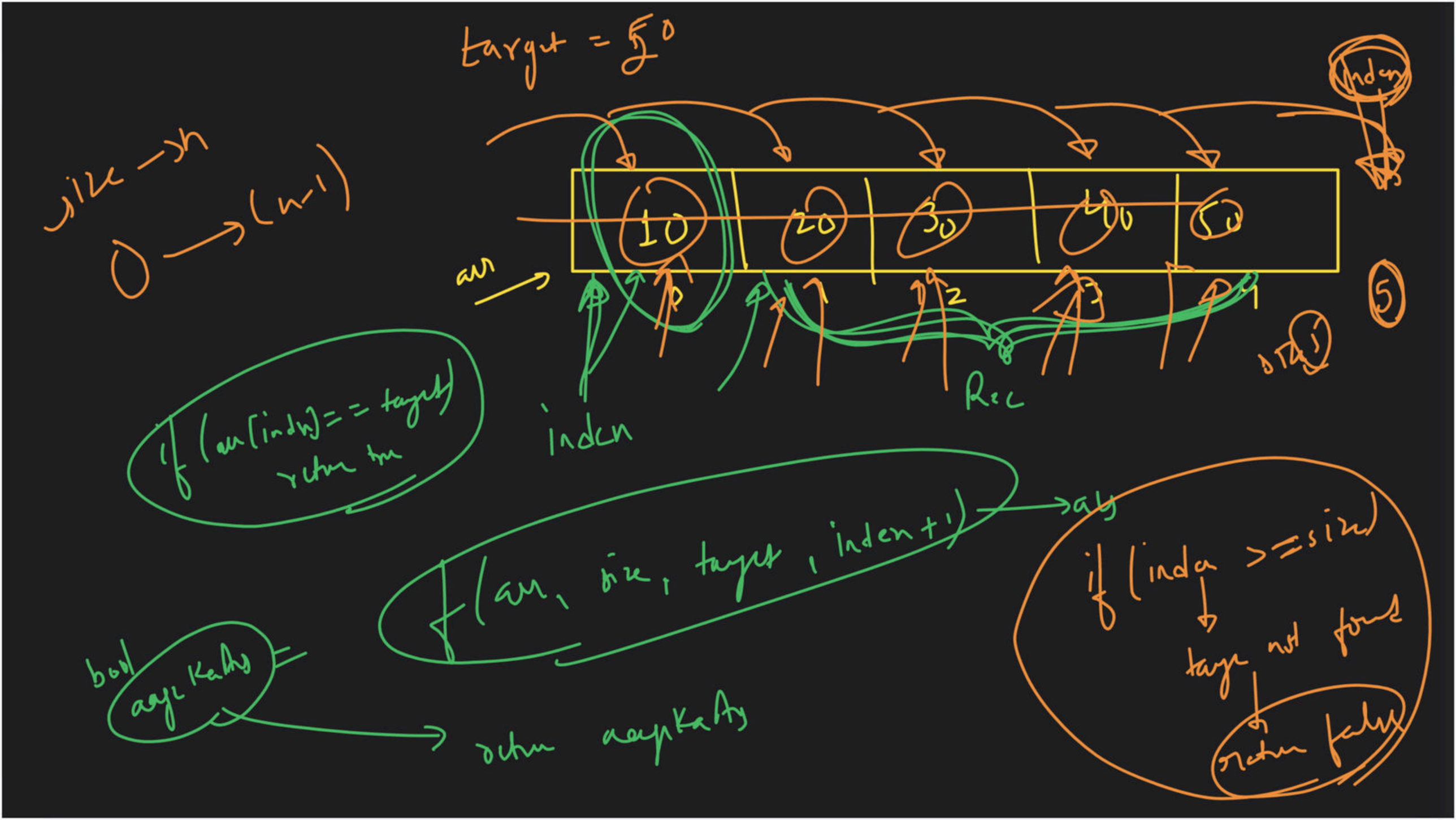
Search in Array

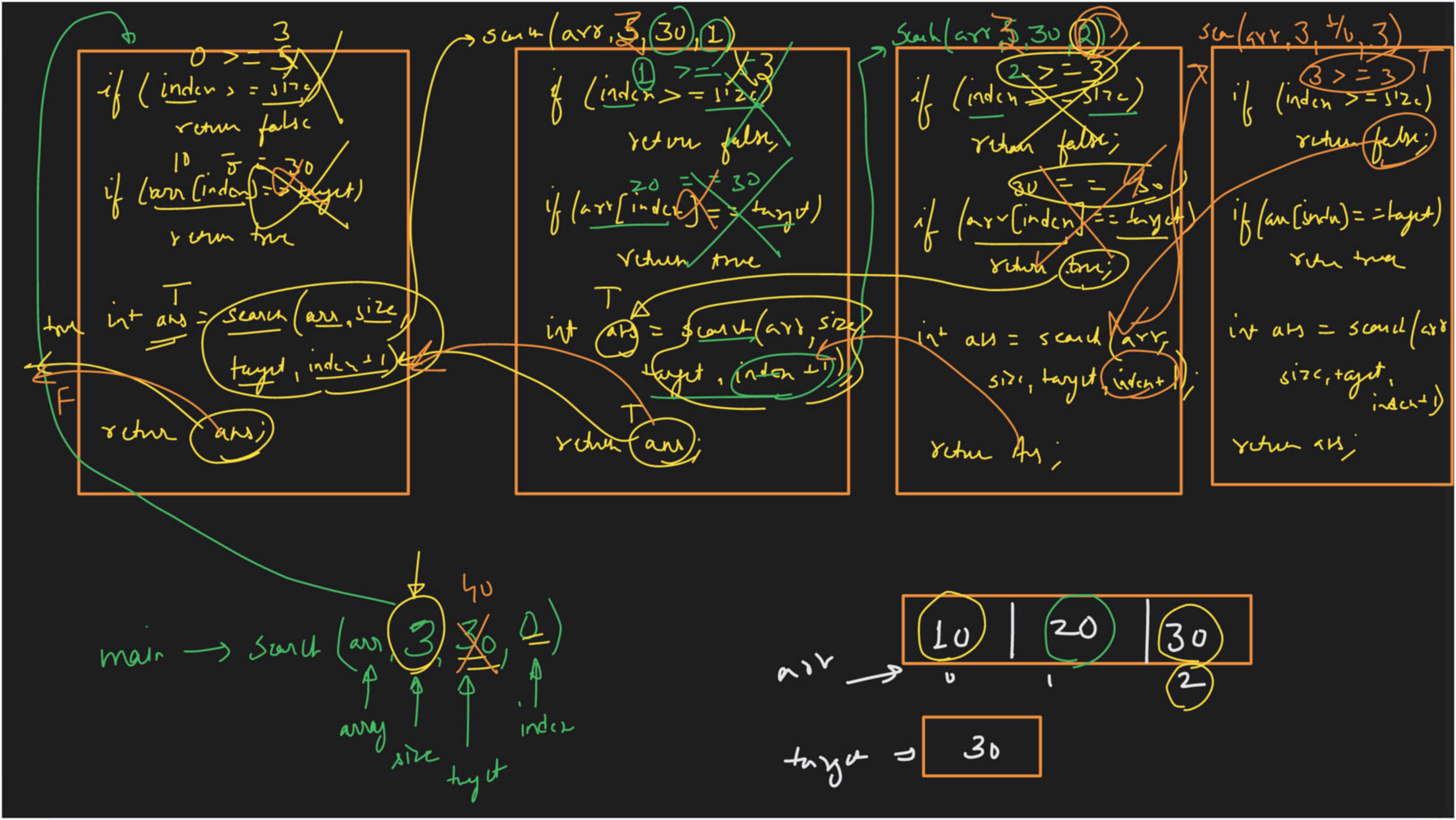
target = 50

T/F

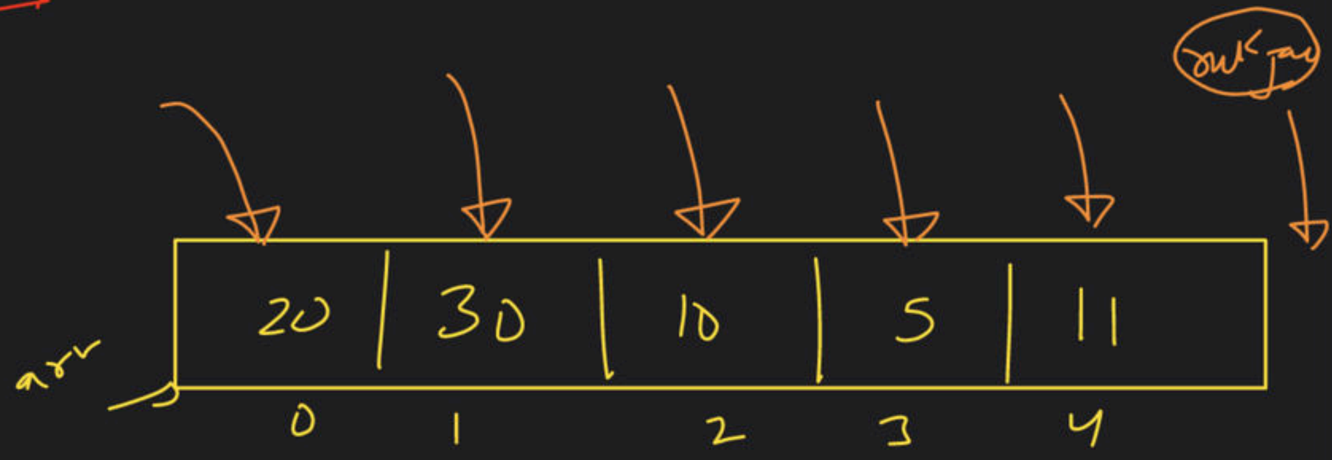
milne pr
↳ return
true

entire array
traverse hogya → B.C
return false


target = 50
size -> n
0 -> (n-1)
arr
10
20
30
40
50
index
5
Rec
if (arr[index] == target)
return true
f(arr, size, target, index+1)
bool
arrKaAny
return arrKaAny
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, 5, 30, 1)
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, 2)
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, 3)
if (index >= size)
return false;
if (arr[index] == target)
return true
int ans = search(arr, size, target, index+1)
return ans;
true
F
T
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;
}
```

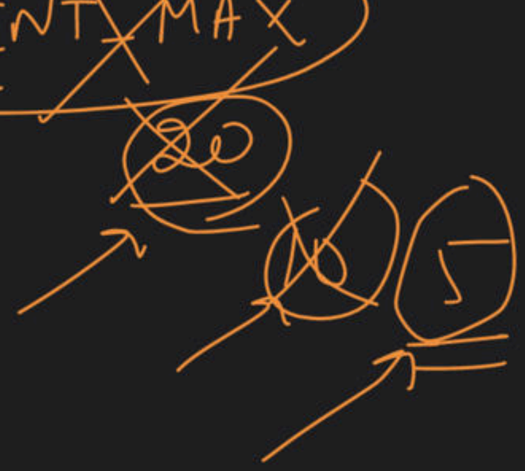
Recursion

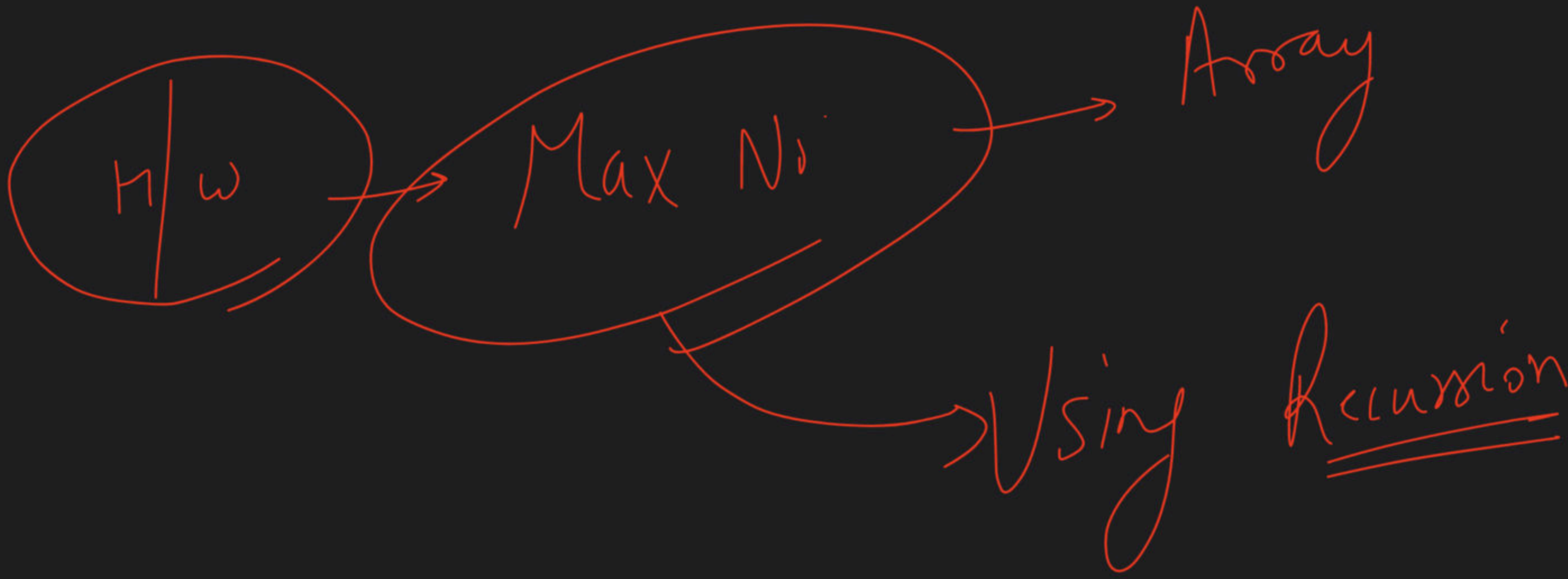
mini = min (mini, arr[index])

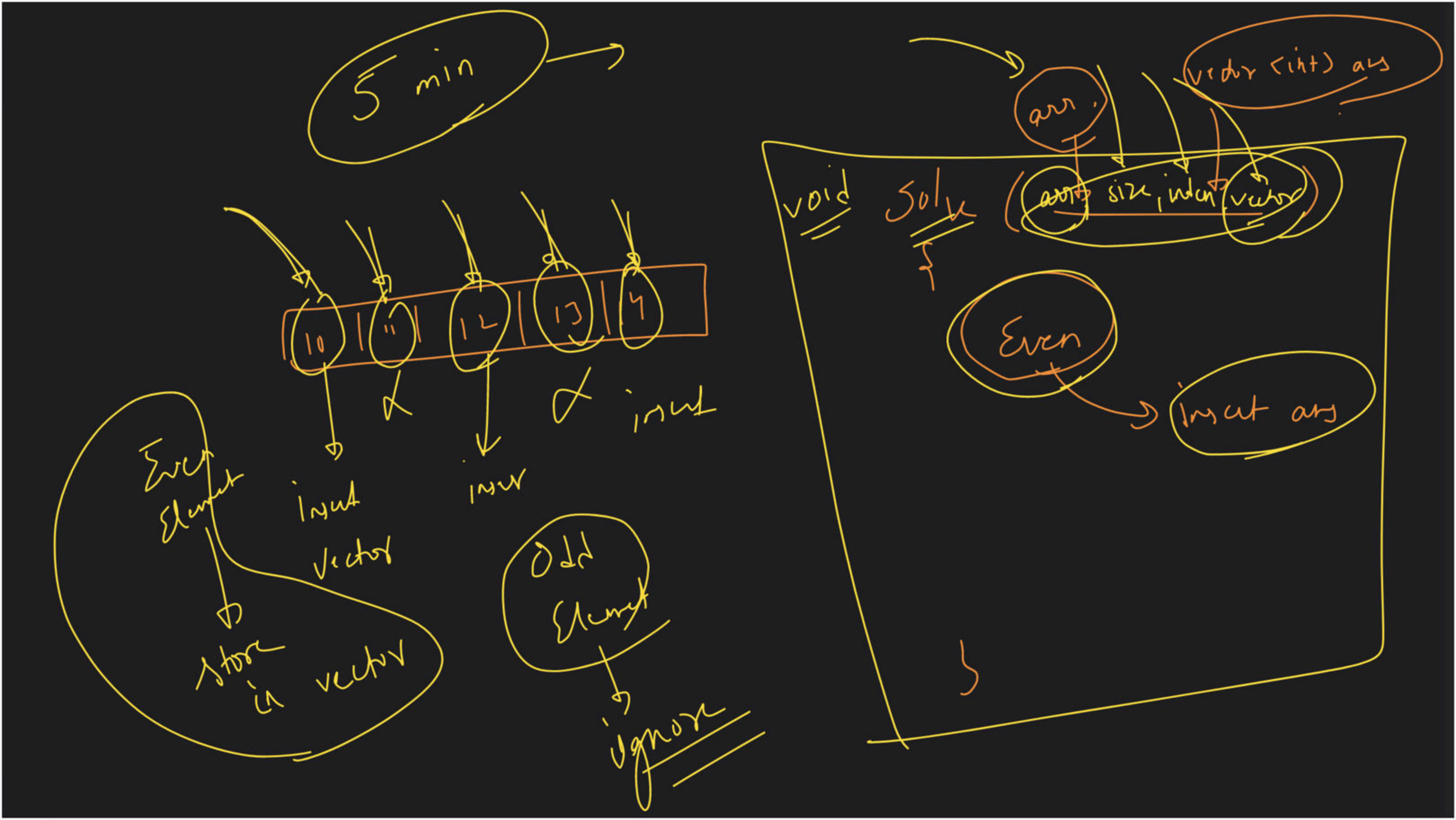
R.f

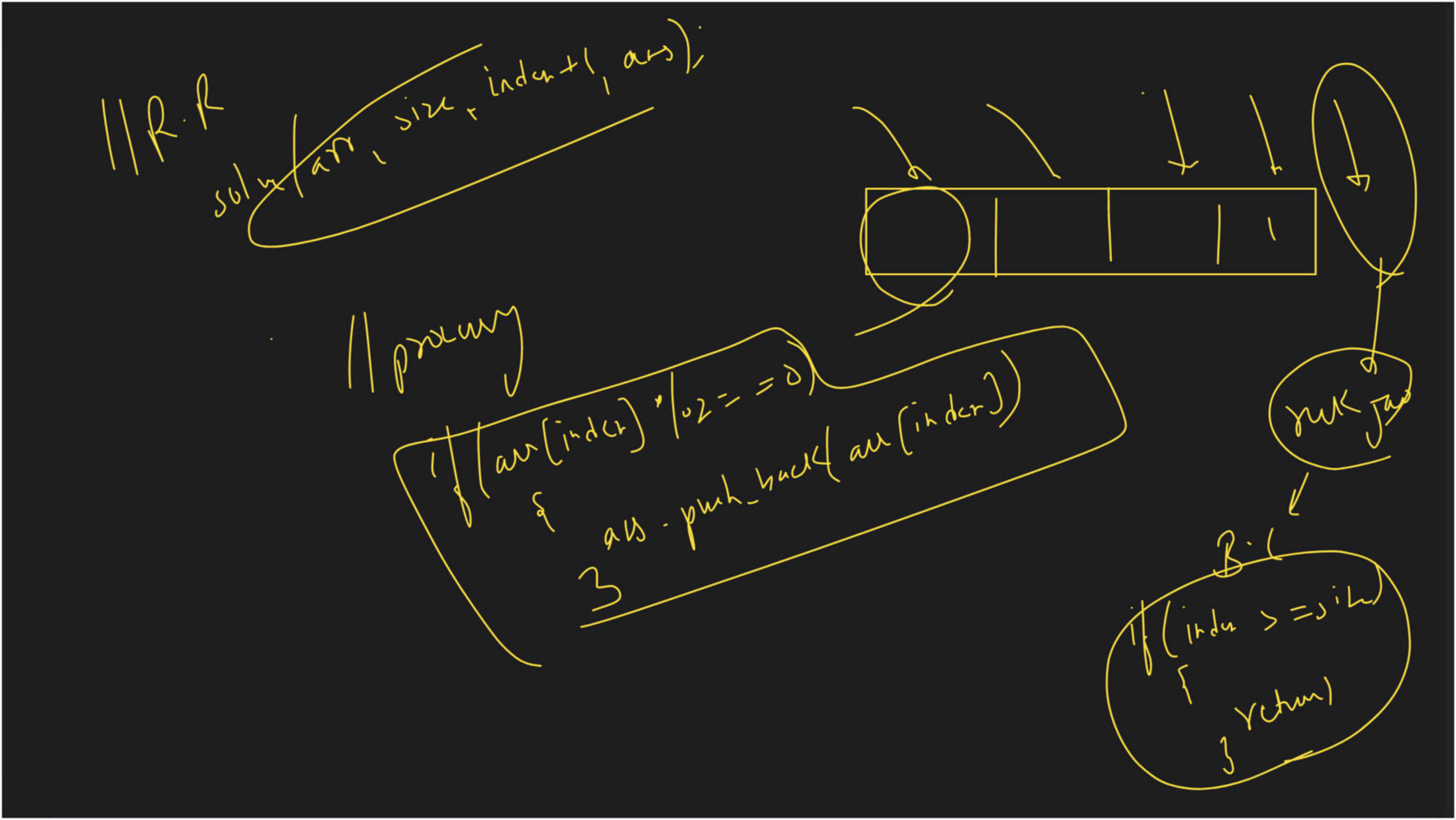
f (arr, size, index+1, mini)

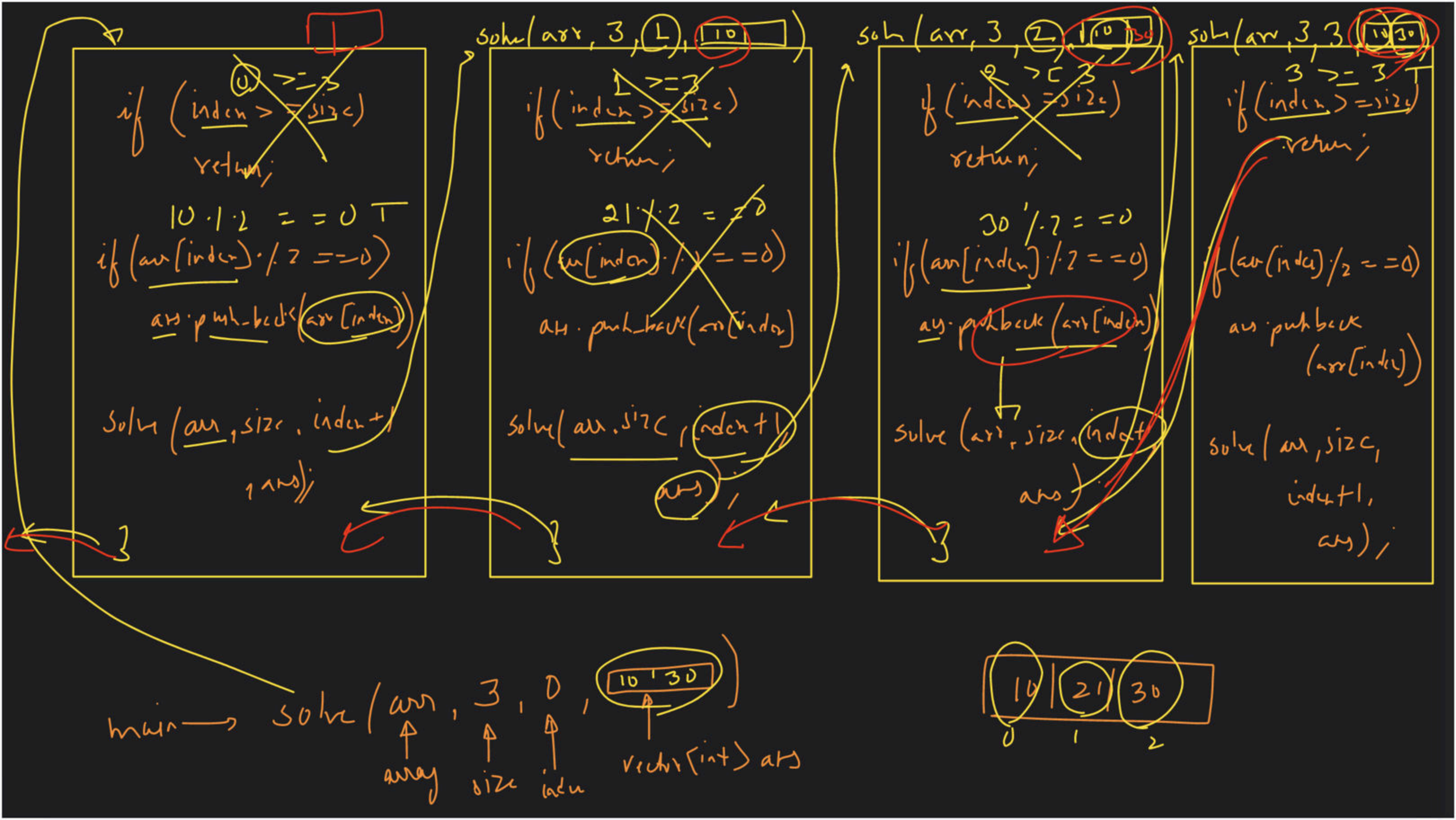
int mini = INT_MAX

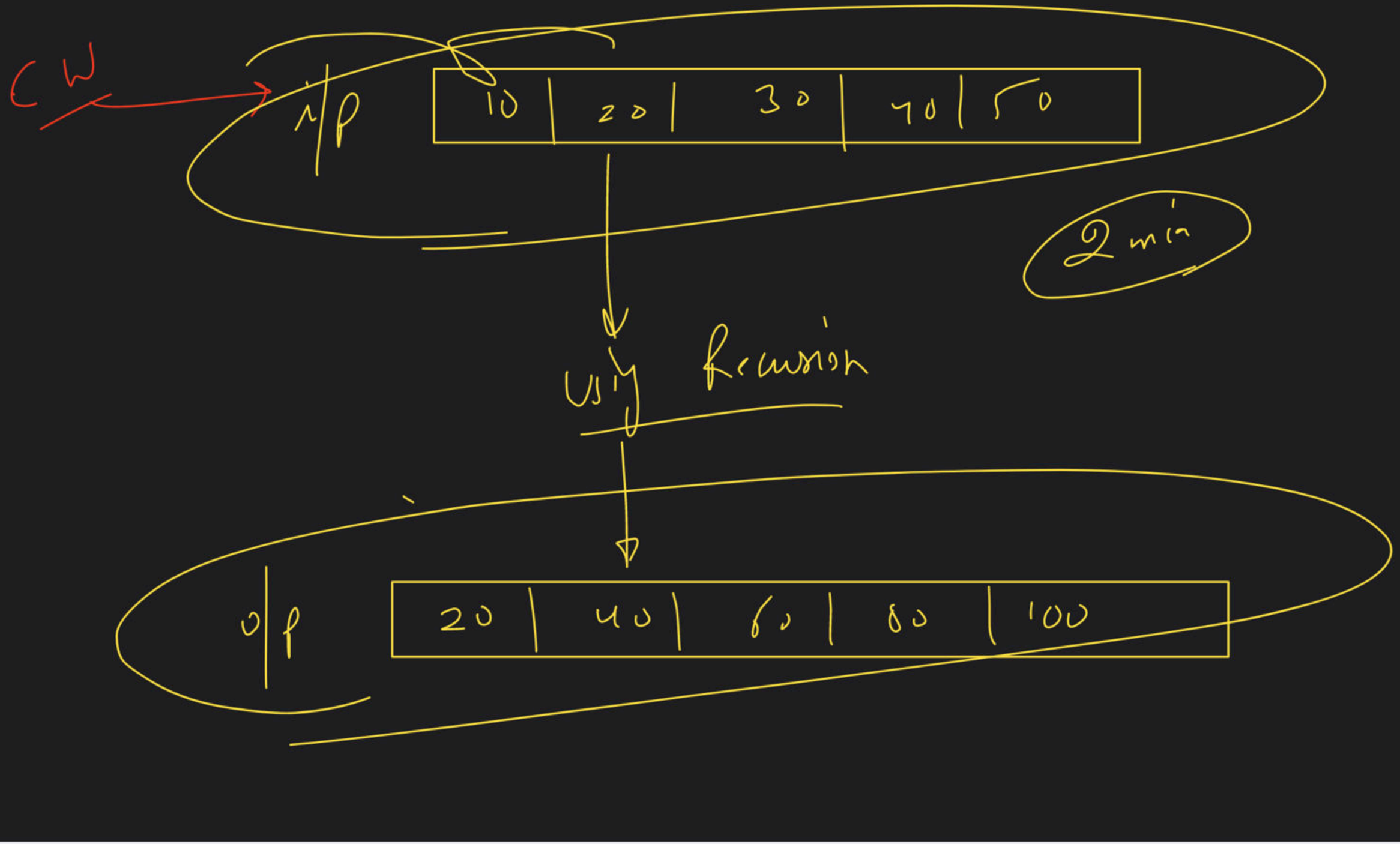



H/W
Max No
Array
Using Recursion


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

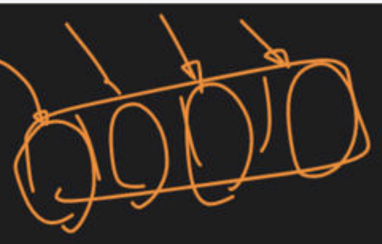

//R.R
sol(arr, size, index, ans);
//proxy
if(arr[index]%2==0)
{
ans.push_back(arr[index])
}
Bit
if(index >=size)
{
return
}


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)
3 >= 3 T
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, indcn)
{
    // Base Case
    if (indcn >= size)
        return;

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

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

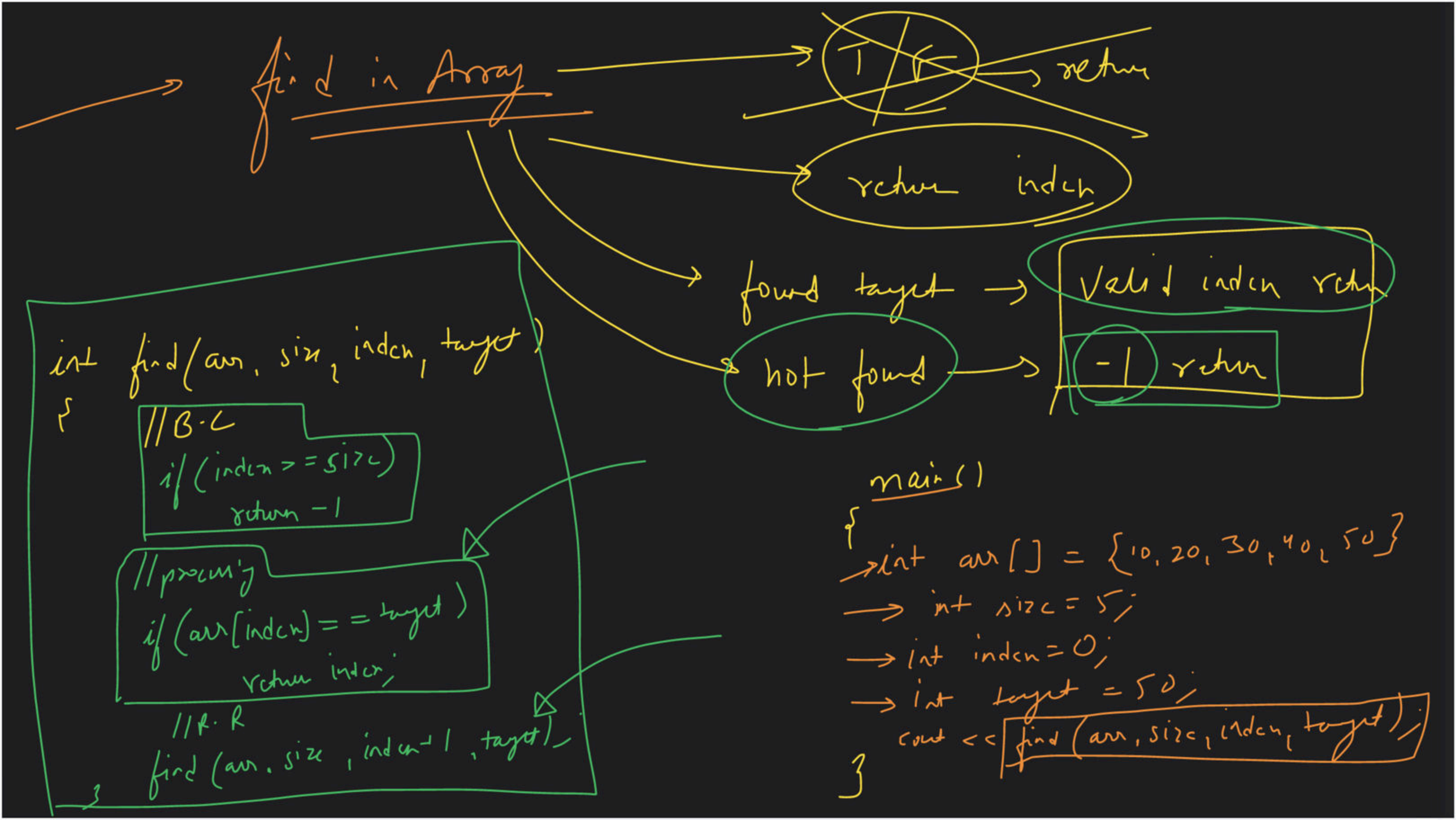
Max no in Array → 2 min



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

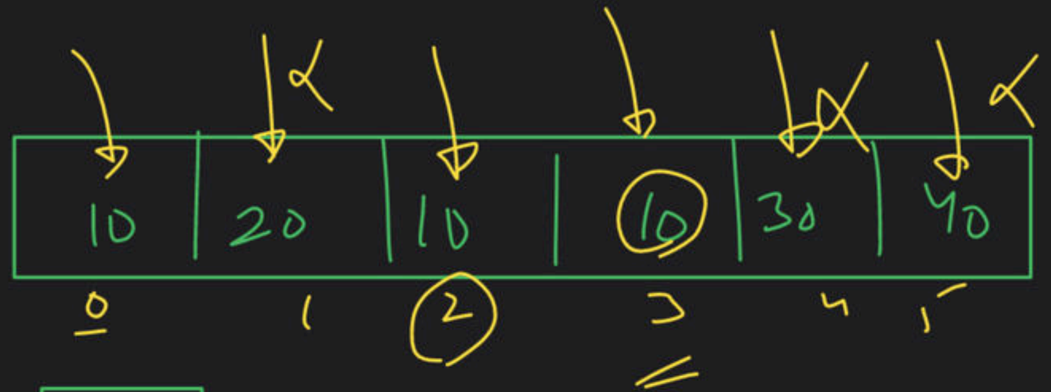
by ref

```
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
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 Occurence of target in Array

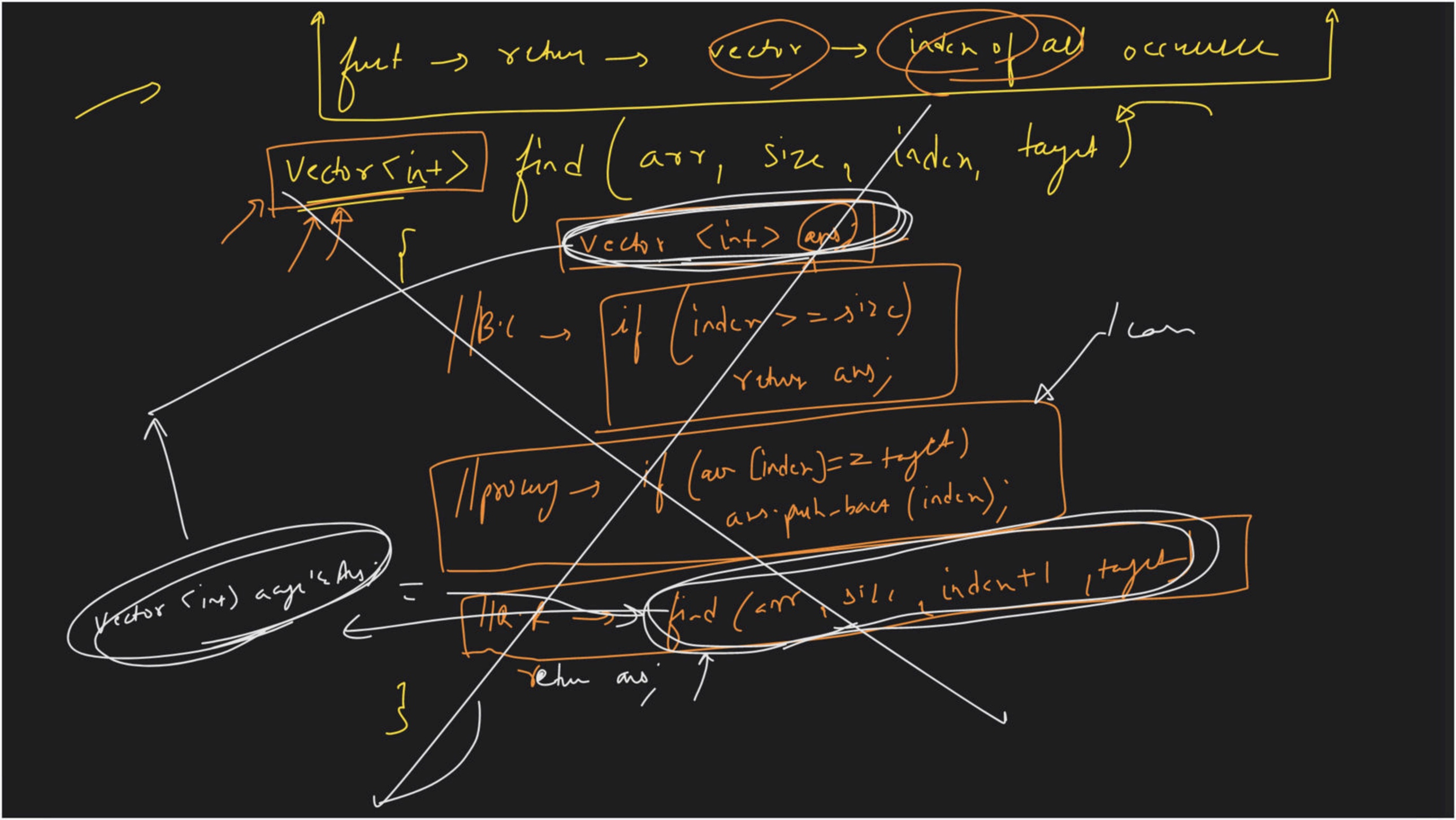
2 min

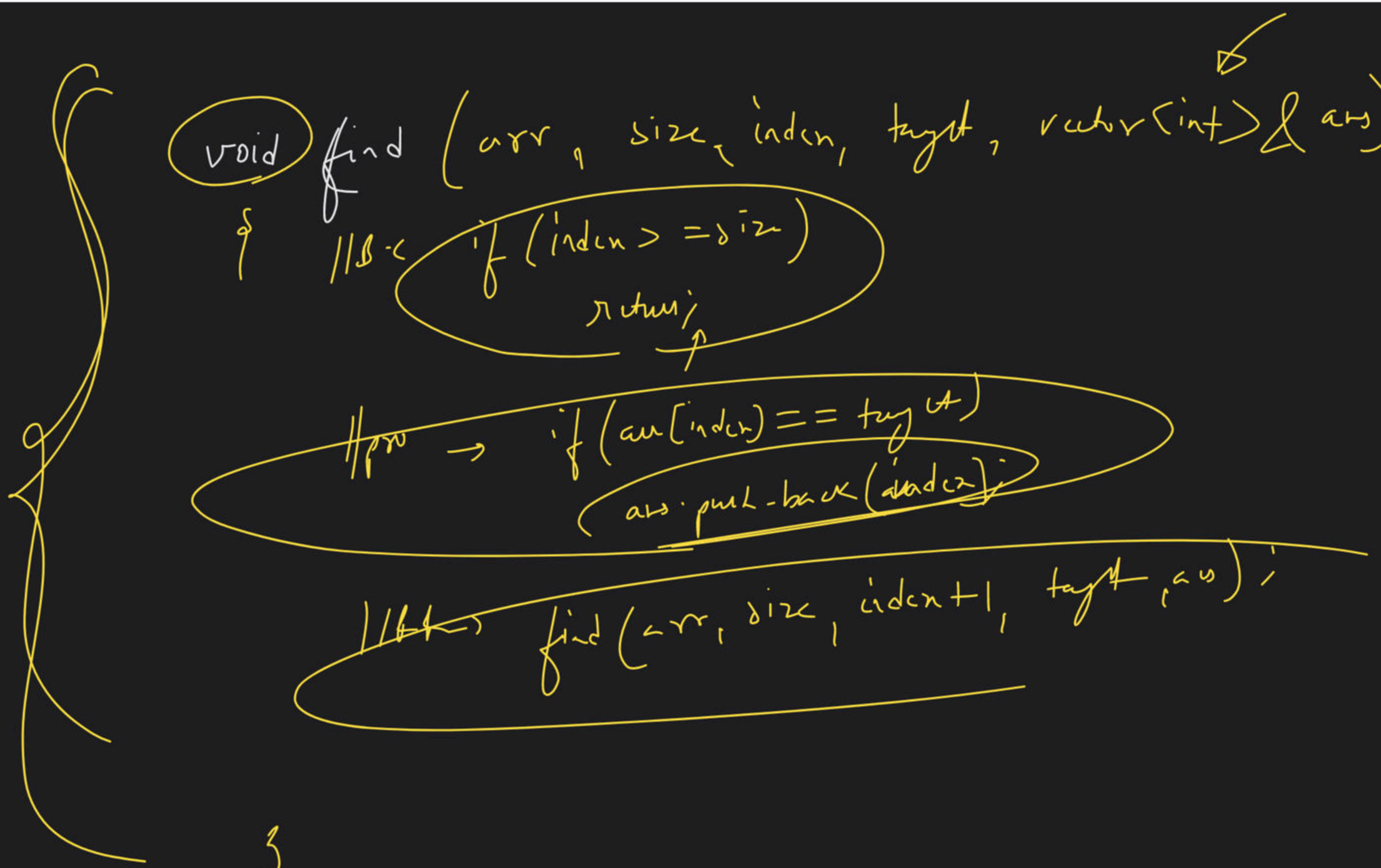


target

0 | p → 0 2 3

```
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);
}
```


find → 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 = this->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

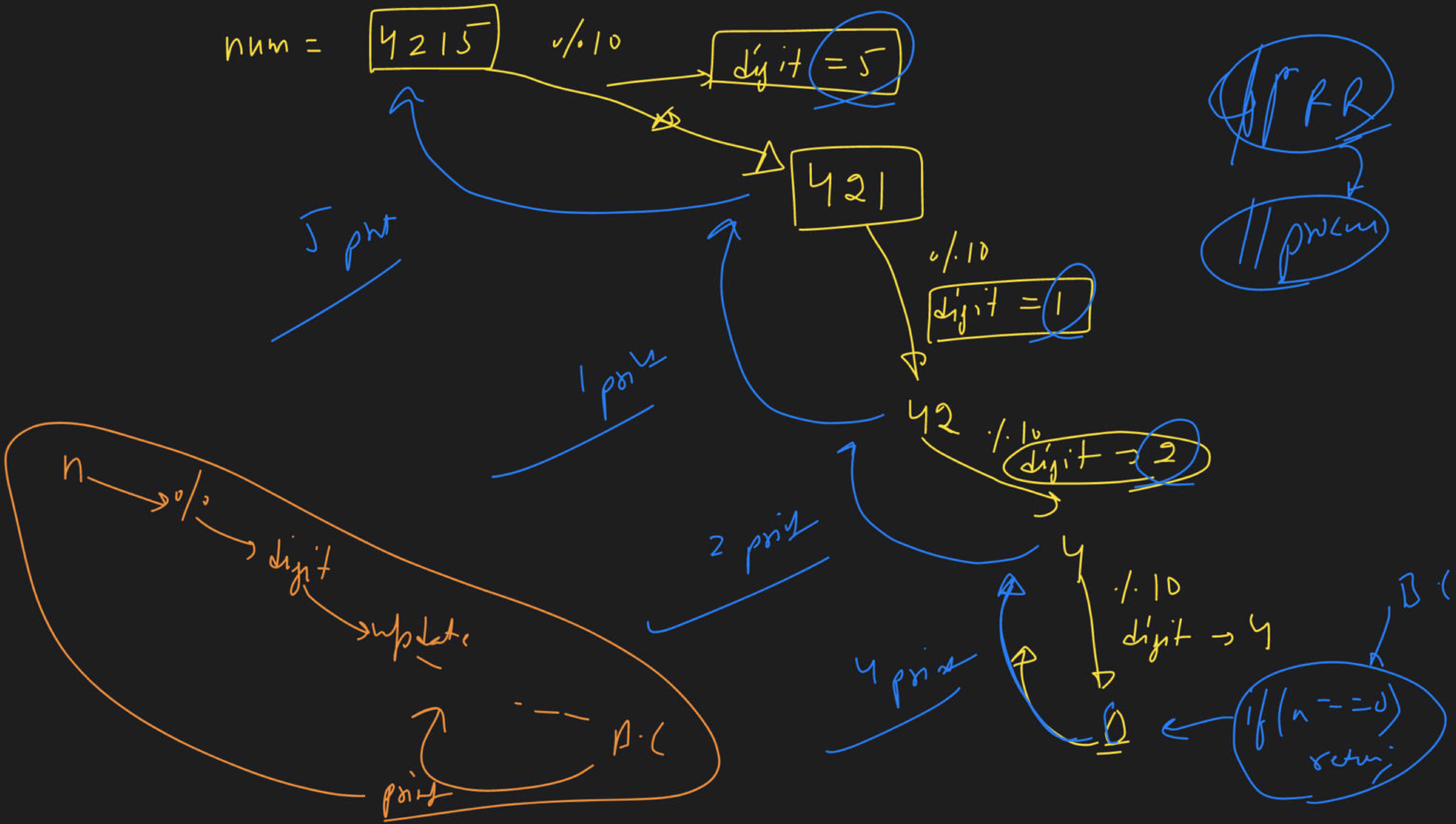
o/p → vector → digits stored

i/p → 4215

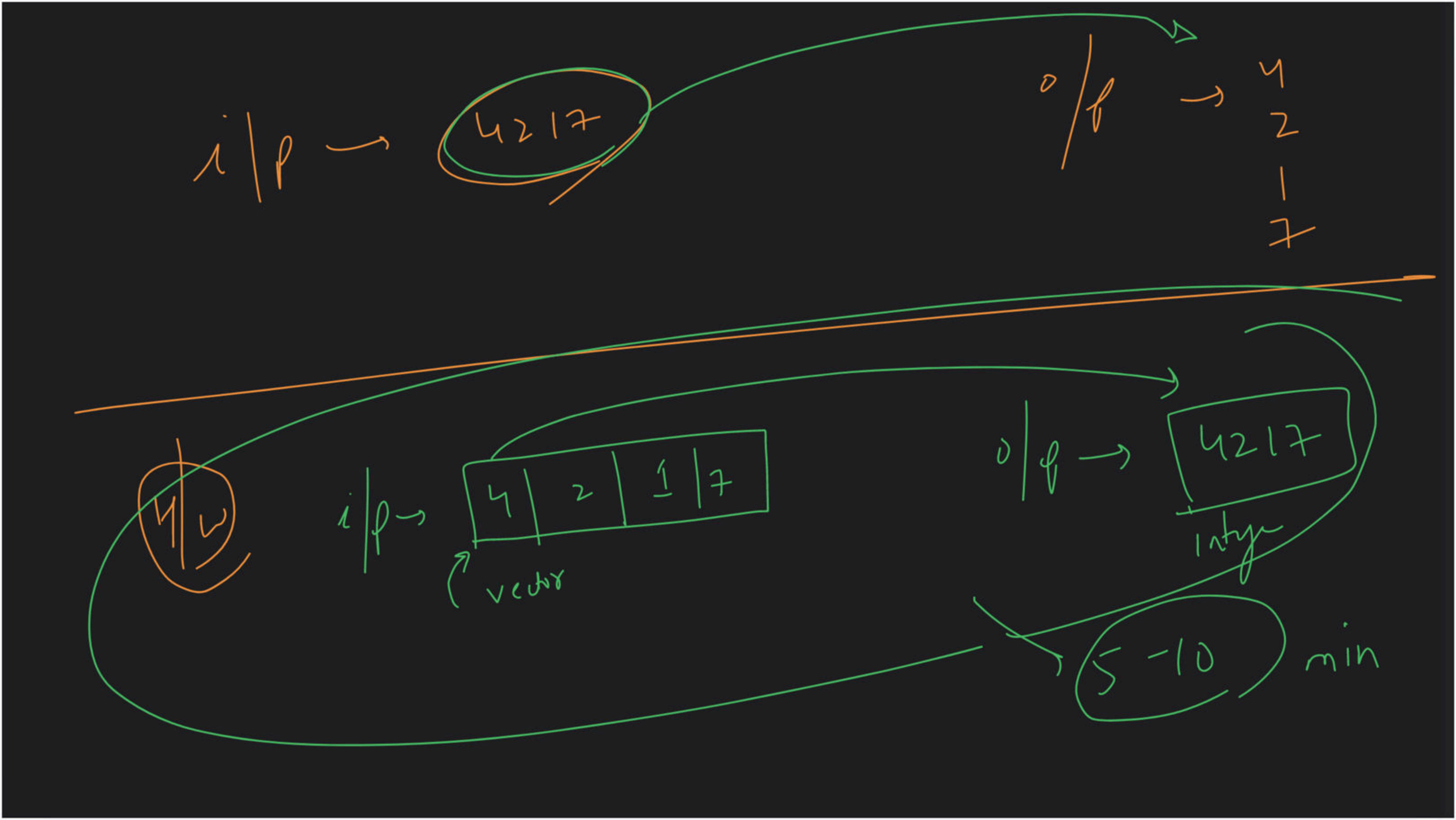
→ print digits

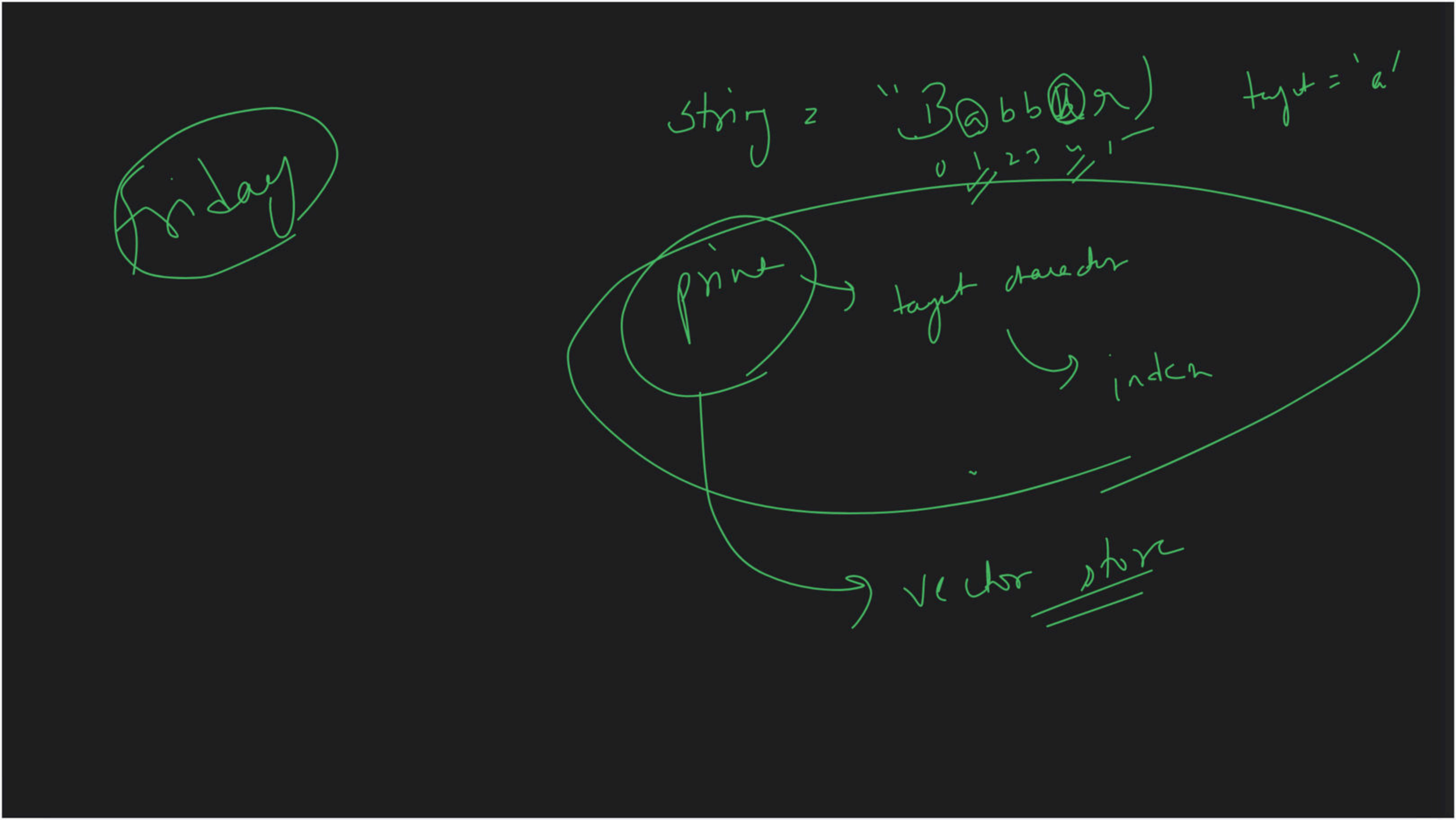
2 min

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
B.C
print


i/p →
4217
o/p →
4
2
1
7
4
2
i/p →
4
2
1
7
vector
o/p →
4217
intyp
5 -10
min


friday
string z = "Baabbaar"
0 1 2 3 4 5 6
target = 'a'
print
target character
index
vector store