longest subarray with equal number of 0's, 1's, and 2's

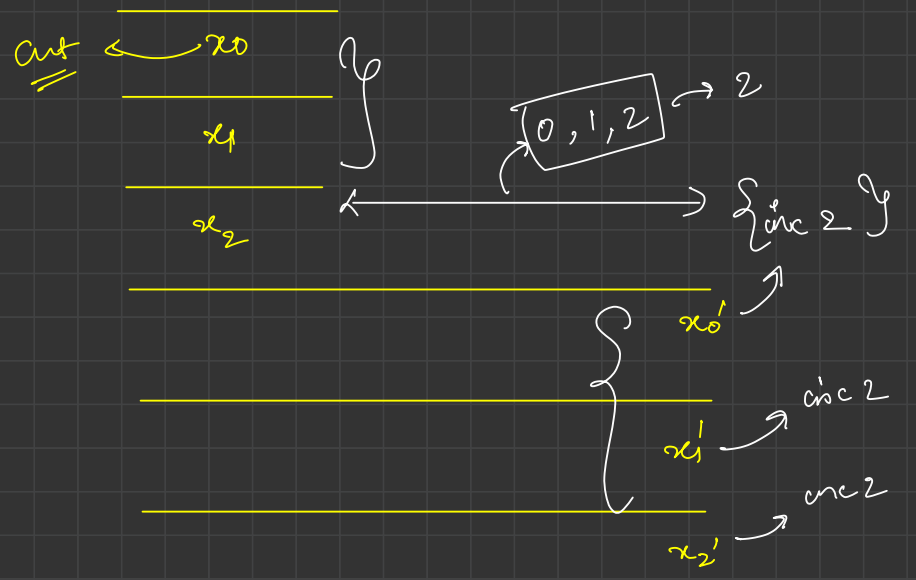
arr[] = { 1, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 0, 1 }

Brute force

- find all the subarray → O(N²)
- cnt no. of 0's, 1's, & 2's → O(N)
- if cnt are equal
  - you got a potential
  - store max. len. ans

- TC: O(N³)
- SC: O(1)

arr [ ] = { 1, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 0, 1 }



$$x_0' = x_0 + y$$
$$x_1' = x_1 + y$$
$$x_2' = x_2 + y$$

$$x_0' = x_0 + y \quad \text{——} (1)$$

$$x_1' = x_1 + y \quad \text{——} (2)$$

$$x_2' = x_2 + y \quad \text{——} (3)$$

(2) − (1)

$$(x_1' - x_0') = (x_1 - x_0)$$

(3) − (2)

$$(x_2' - x_1') = (x_2 - x_1)$$

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr[] =	1	1	2	0	1	0	1	2	1	2	2	0	1

x_0	0	0	0	0	0	1	1	2	2
x_1	1	0	1	2	2	2	3	3	4	4
x_2	2	0	0	0	1	1	1	1	1	2
$x_1 - x_0$		0	1	2	2	1	2	1	2	2
$x_2 - x_1$		0	-1	-2	-1	-1	-2	-2	-3	-2

Key

0#0 1#-1 2#-1 2#-2 2#-2 1#-1 1#-2 2#-3

$(x_1 - x_0) \# (x_2 - x_1)$

Key	value
0#0	-1
1#-1	0
2#-2	1
2#-1	2
1#-2	5
2#-3	0

maxlen = 3 − 0

= 6

7 − 1 = 6

Rabbit in forest

	0	1	2	3	4	5	6	7	8
arr[]:	2	2	3	1	0	2	2	3	1

a ✓ ✓ ✓

b ✓ ✓ . .

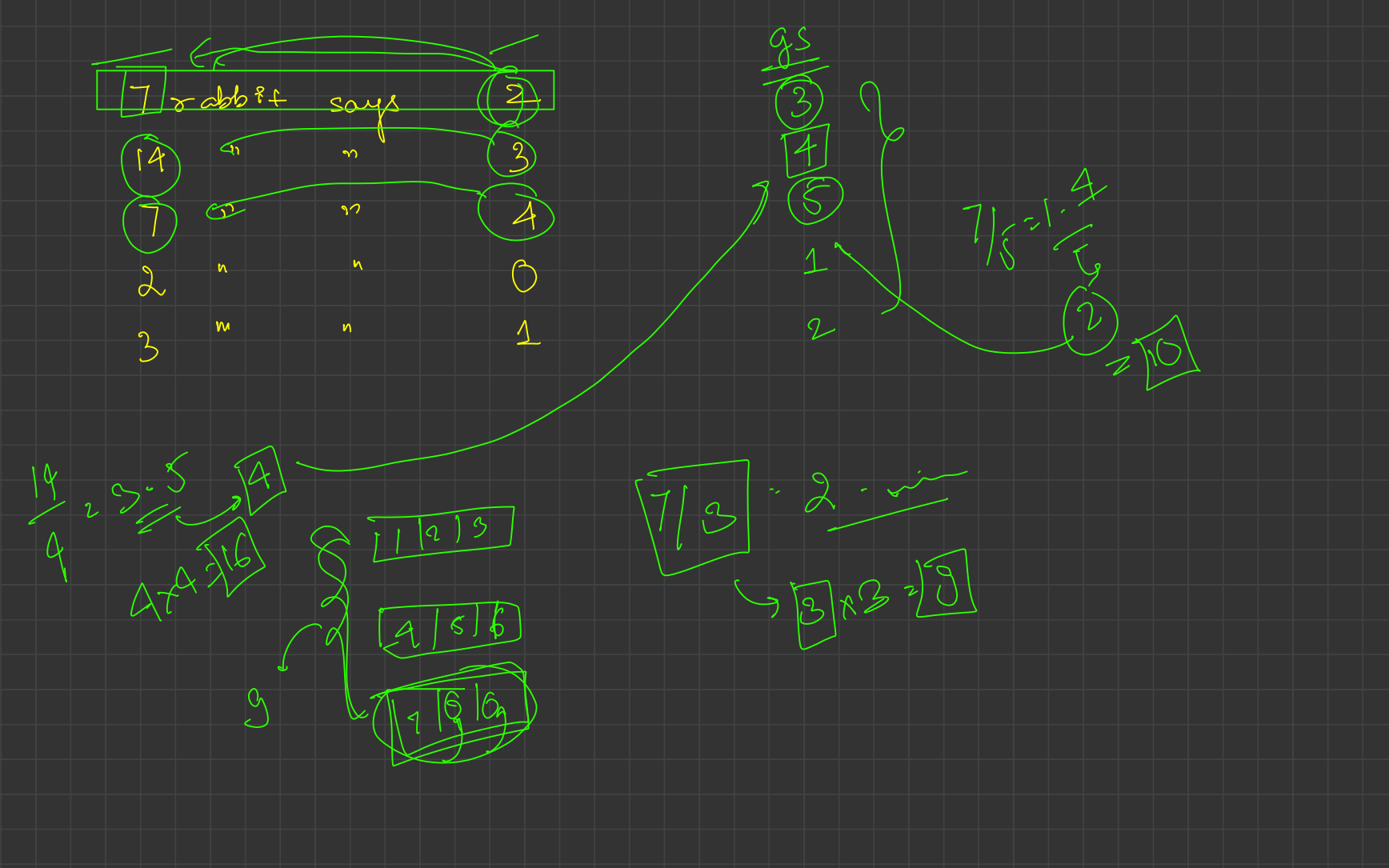
c ✓ ✓

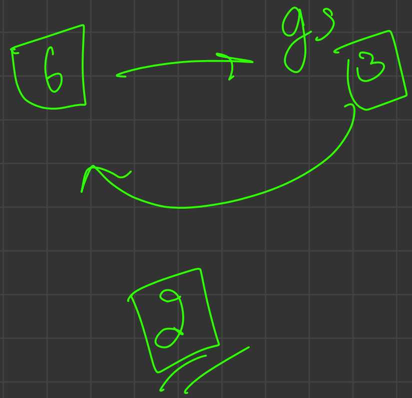
d ✓

e ✓ . .

3 + 4 + 2 + 1 + 3

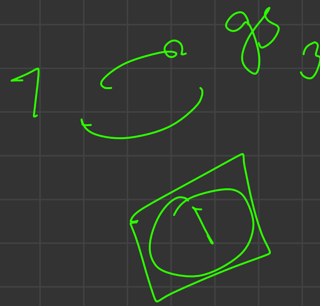
= 13 rabbits


rabbit says
14
7
2
3
gs
3
4
5
1
2
7/5 = 1 · 4
2
×10
14/4 = 3 · 5
4
4×4=16
1 2 3
4 5 6
7 8 9
7/3 = 2 · ...
3×3=9

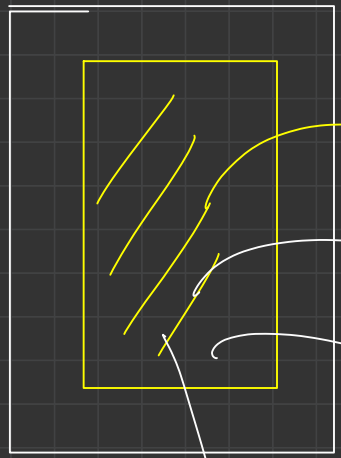


ans / g = fully filled grp

ans % g = 20 → partially filled grp



LRU cache



App1 minimize

App2 minimize

App3 minimized

App4 minimize

Cache
=
memory

{ (App1) (App2) (App3) }

(App4)

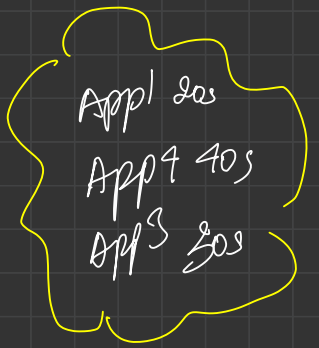
Cache Memory Managment System

OS

LRU { least recently used }

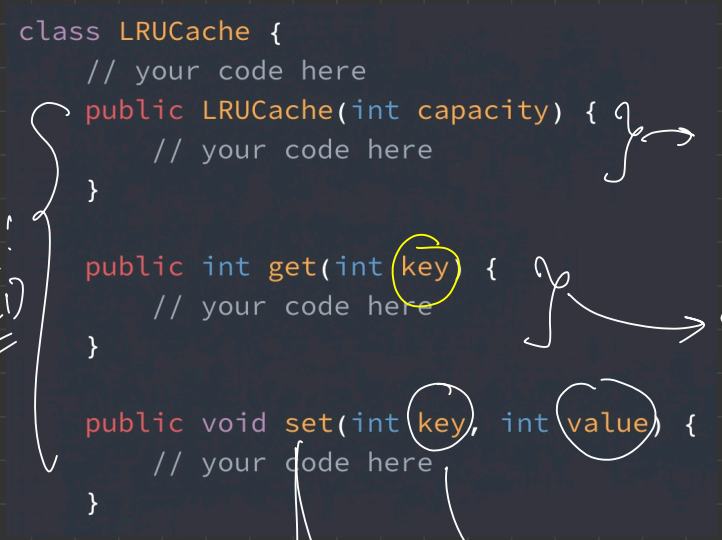
LFU { least freq. used }

LRU



t = 0s App1 (opened)
t = 10s App2 (opened)
t = 20s App1 (output)
t = 30s App3 (opened)
t = 40s App4 (opened)

Cache Memory { 3 Application is Capacity }



TC: O(1)

initialize capacity of cache memory

applications gives some output

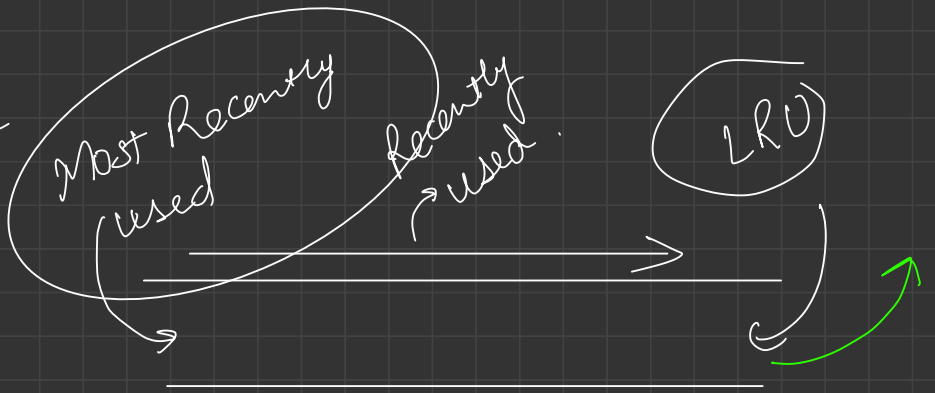
Hence it becomes Most Recently Used

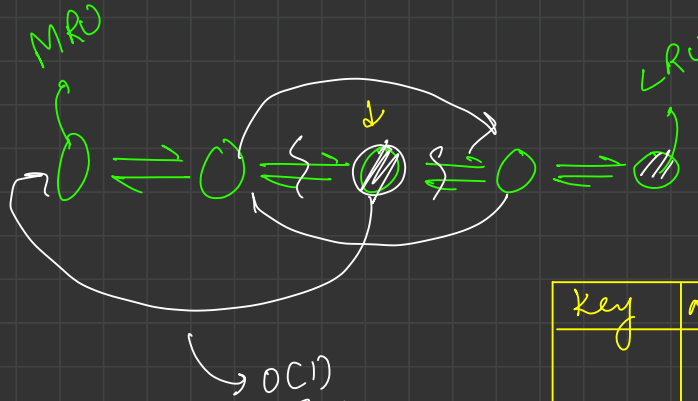
opens a new application

to update value of a application

Most Recently Used

linear data structure





O(1)
removeNode()
addtofront()

key	address

DLL ✓
Stack ✗
queue ✗
Array ✗
LL ✗

① set (1, 10)

② set (2, 30)

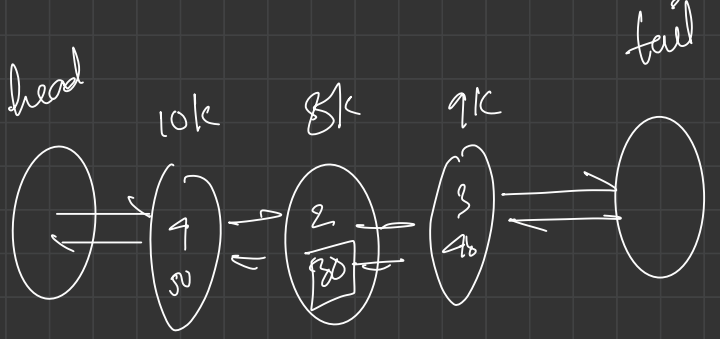
③ set (3, 40)

④ get (2) → 30

⑤ set (4, 50)

⑥

capacity = 3

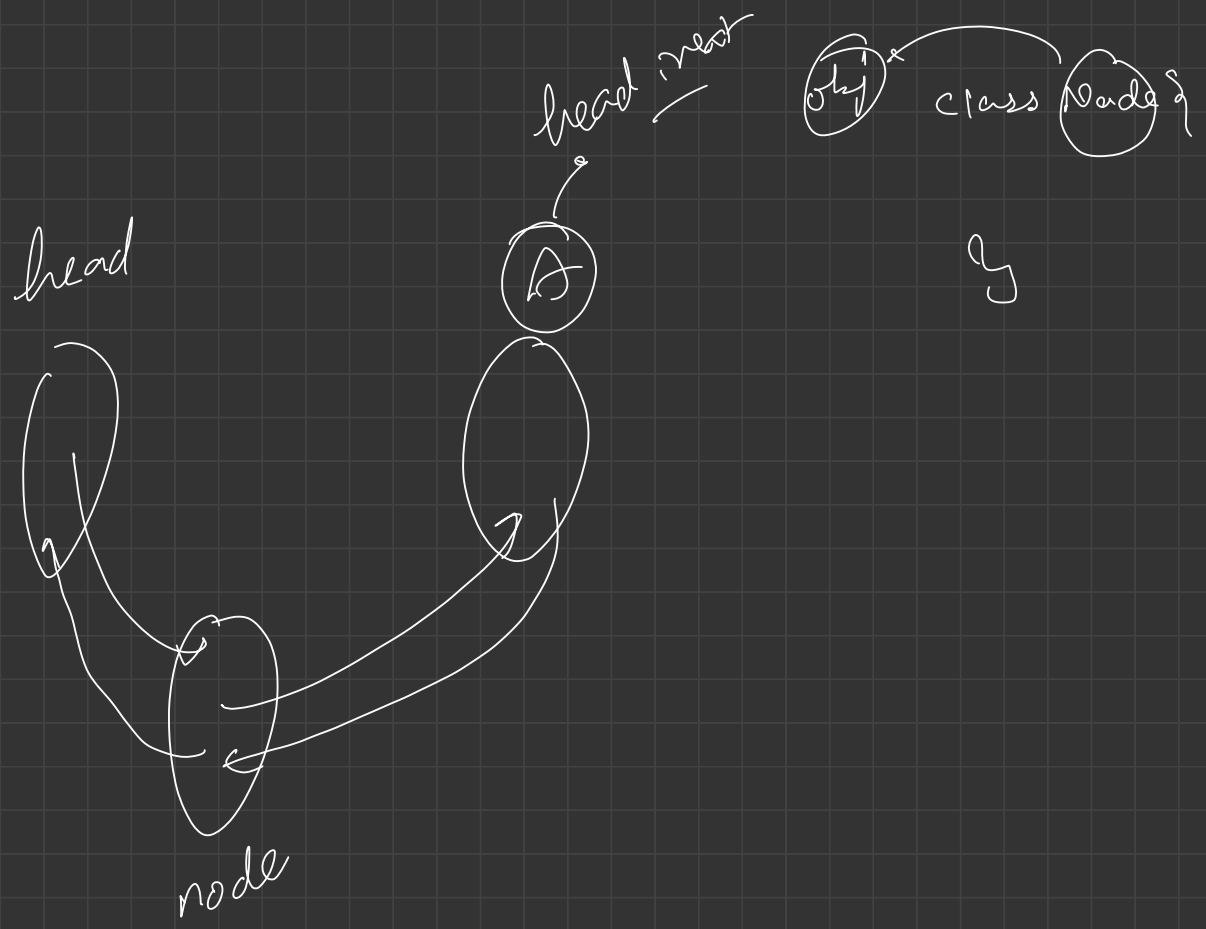


all methods

remove ()

addFirst ()

id	Add
2	8K
3	9K
4	10K


head.next
obj
class Node
head
node

SnapShot Array.

arr[] = { 0, 20, 30, 0, 0 } (indices 0 1 2 3 4)

```
class SnapshotArray {

    public SnapshotArray(int length) {

    }

    public void set(int index, int val) {

    }

    public int snap() {

    }

    public int get(int index, int snap_id) {

    }
}
```

snap_id = ~~0~~ 1 2 ...

initially 0

{ 0, 10, 30, 0, 0 } (indices 0 1 2 3 4) → 0

{ 0, 20, 30, 0, 0 } (indices 0 1 2 3 4) → 1

① set(1, 10)

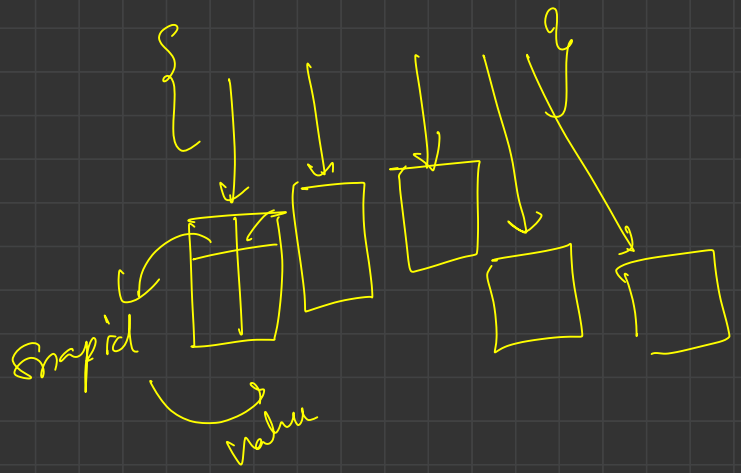
② set(2, 20)

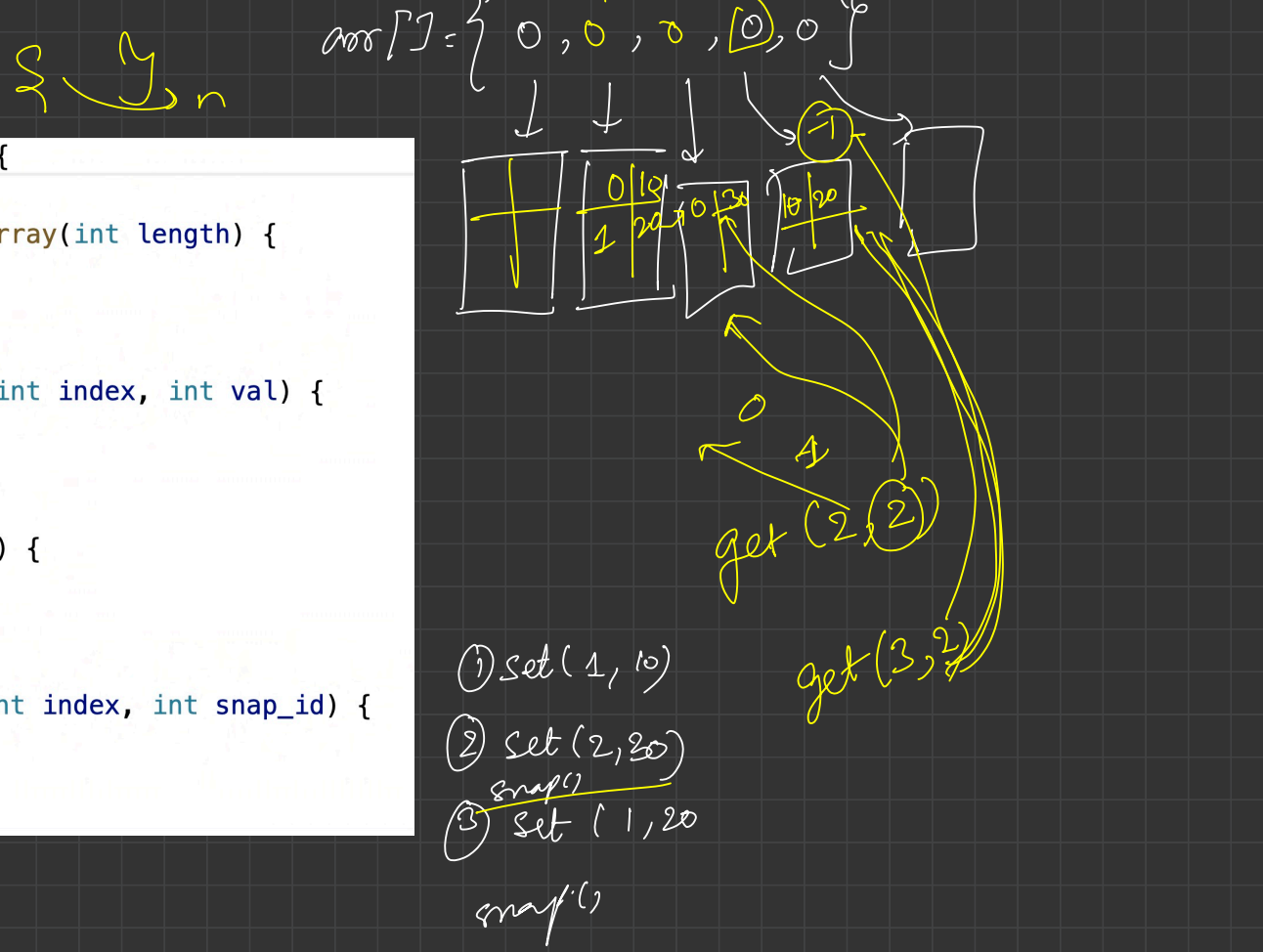
③ snap() set(1, 20)

snap()

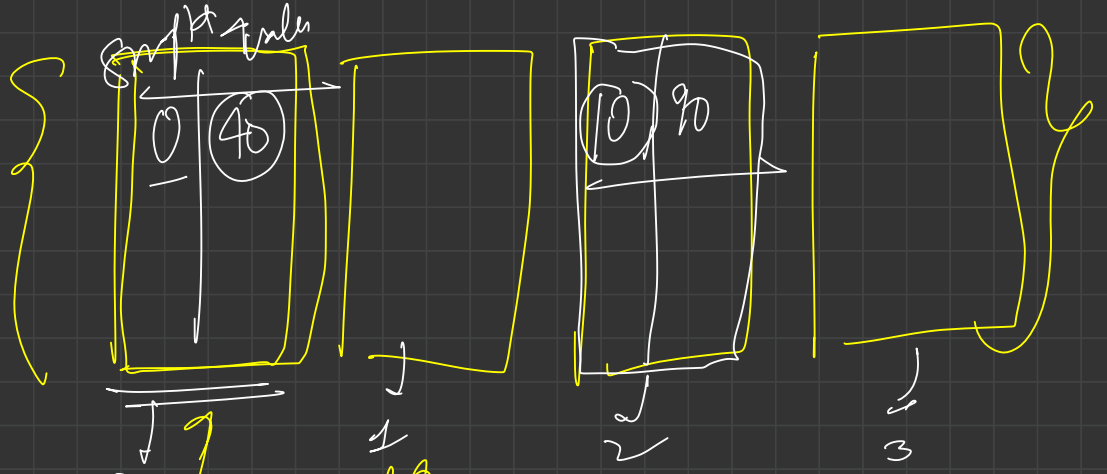
④ get(1, 0)

arr[] = { 0, 20, 30, 0, 0 }





```
class SnapshotArray {
    snap_id,
    public SnapshotArray(int length) {

    }

    public void set(int index, int val) {

    }

    public int snap() {

    }

    public int get(int index, int snap_id) {

    }
}
```



set (0, 20)
=
set (0, 40)