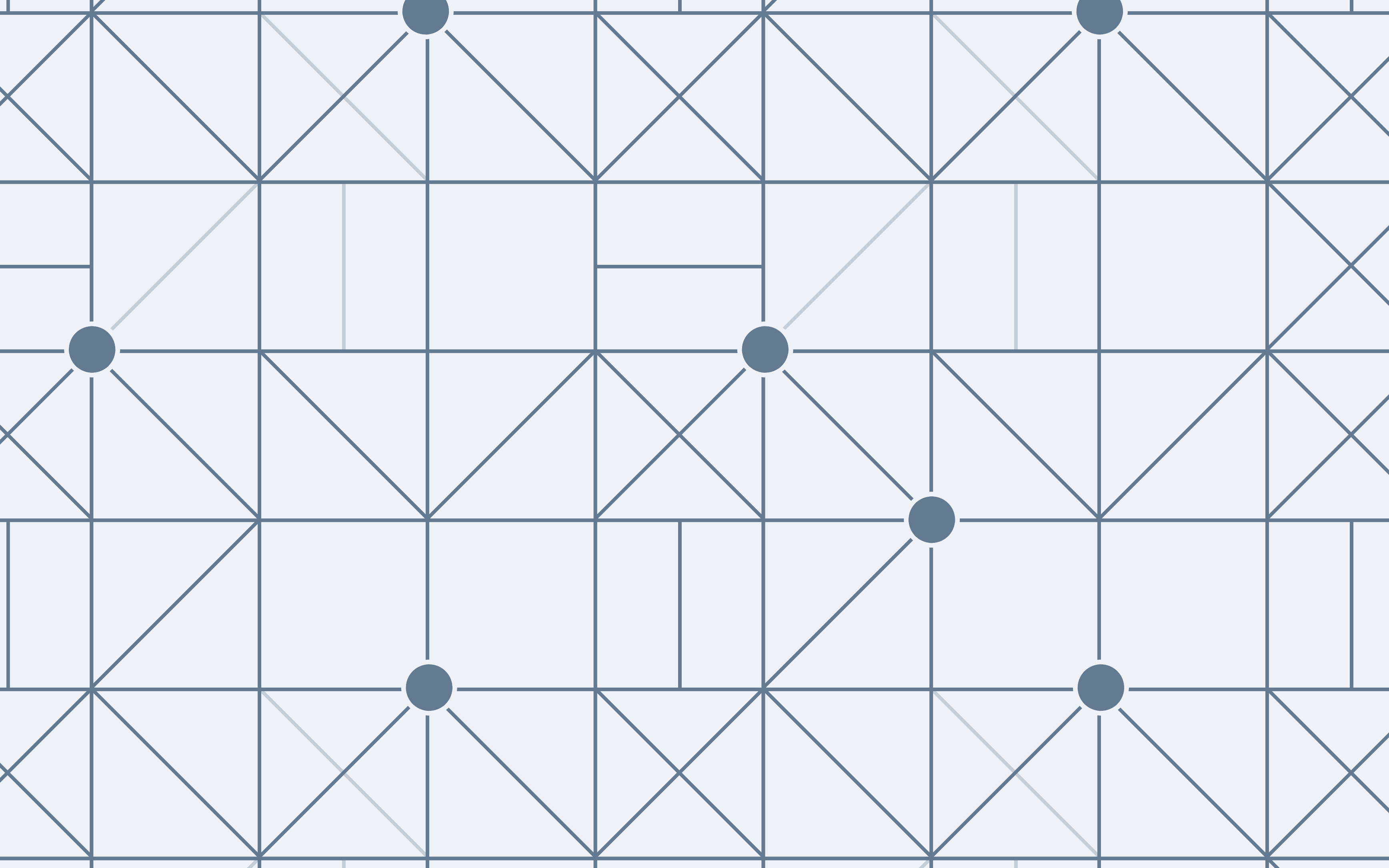


(Imp)

Agenda.

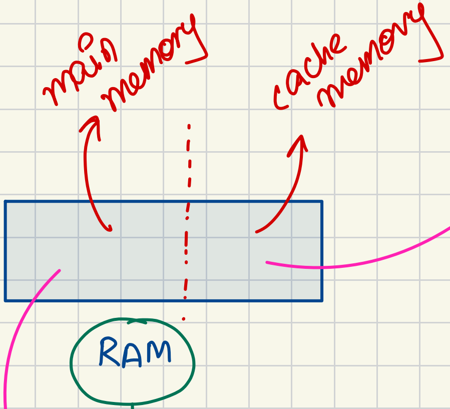
→ LRU Cache

→ Snapshot Array

→ Longest subarray with equal freq.

TOP PBC ✓

LRU Cache.



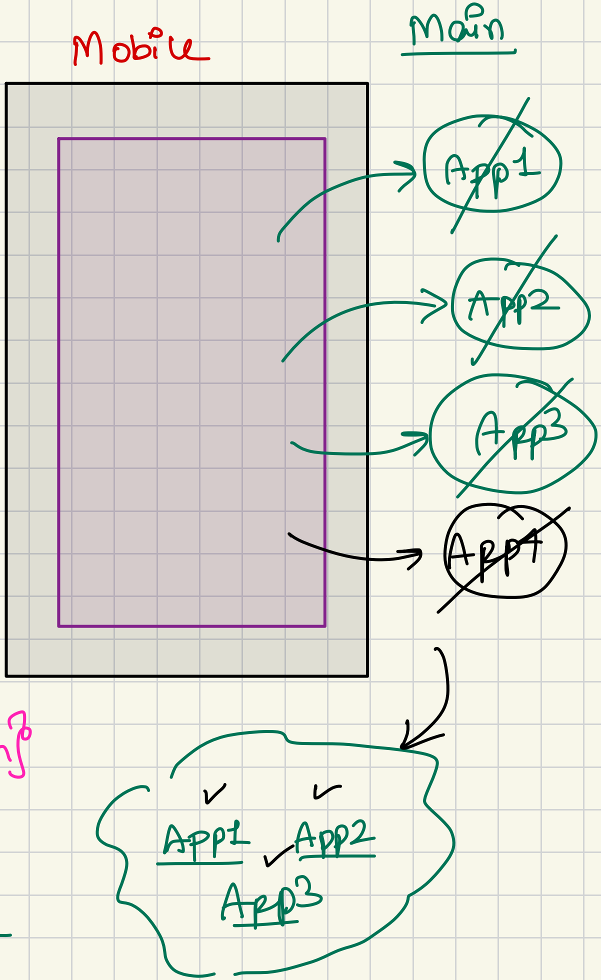
required to run applications in background.

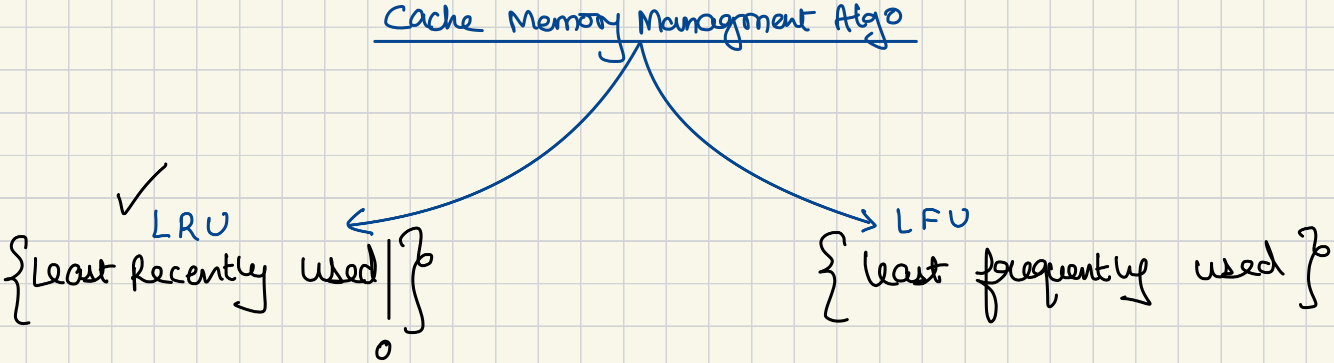
everything runs on RAM

responsible for applications currently in use {on screen}

eventually it will get filled,

Cache





Cache Memory Managment Algo

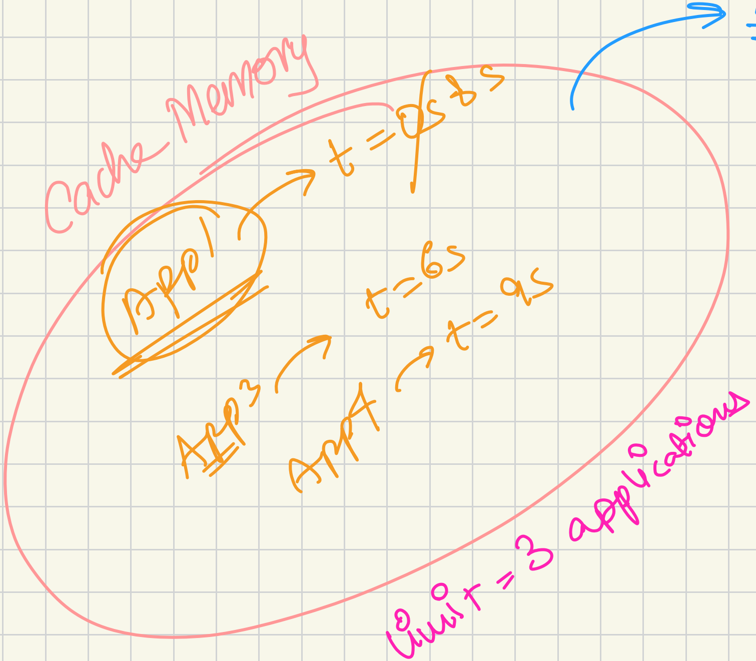
✓ LRU

{Least Recently used|}

0

LFU

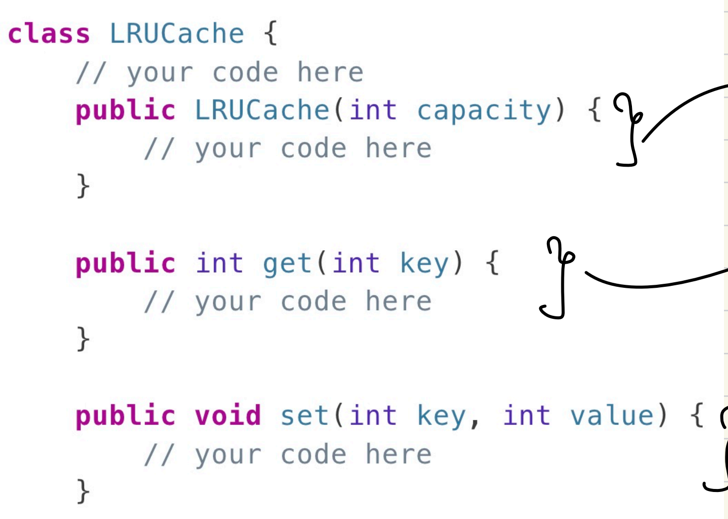
{ least frequently used }



LRU

TC:
O(1)

{ Adding | Removing from Cache
should be TC: O(1)

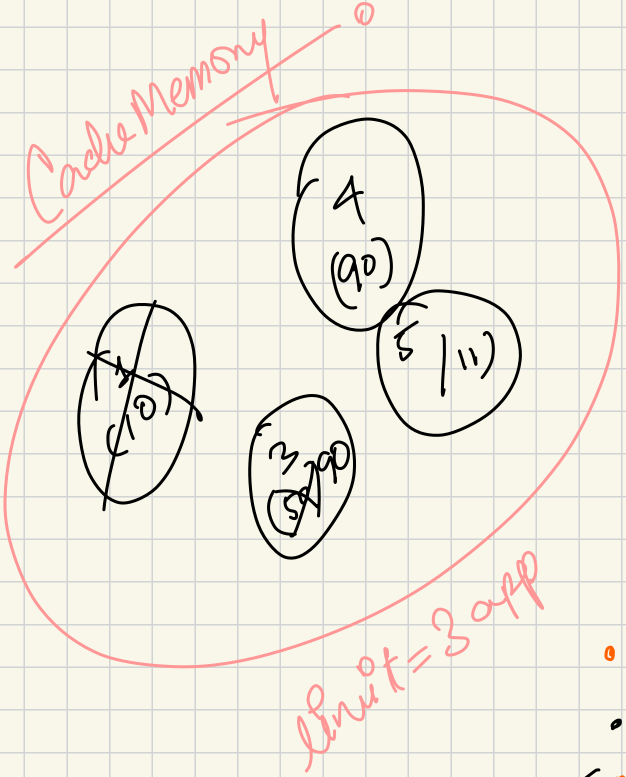


```java
class LRUCache {
    // your code here
    public LRUCache(int capacity) {
        // your code here
    }

    public int get(int key) {
        // your code here
    }

    public void set(int key, int value) {
        // your code here
    }
```

defines max. cap. of your cache memory!

returns the value against an applicatio

adding / Updating an application in cache memory

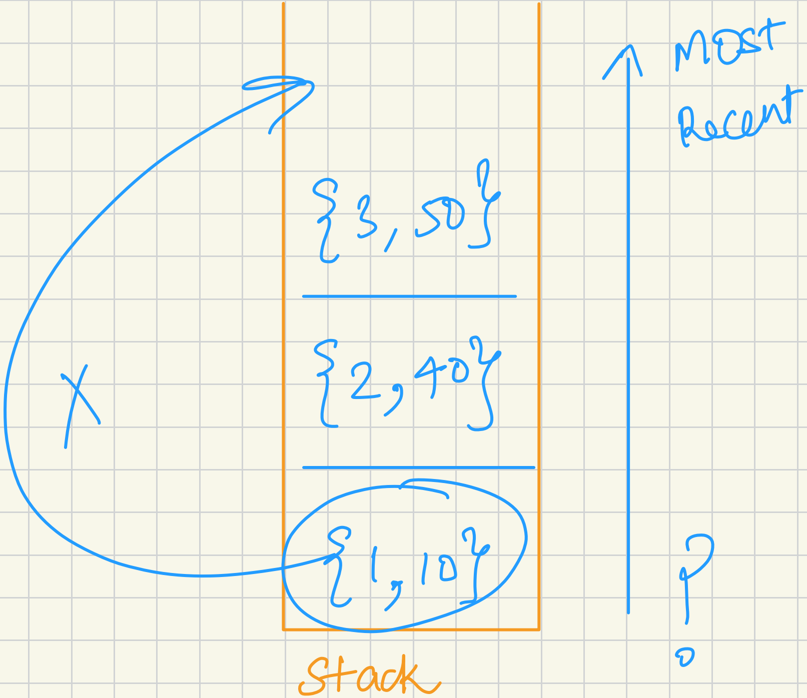


- ✓ set(1, 10)
- ✓ set(2, 40)
- ✓ set(3, 50)
- ✓ get(1) → 10
- ✓ set(4, 90)
- ✓ set(3, 90)
- set(5, 11)

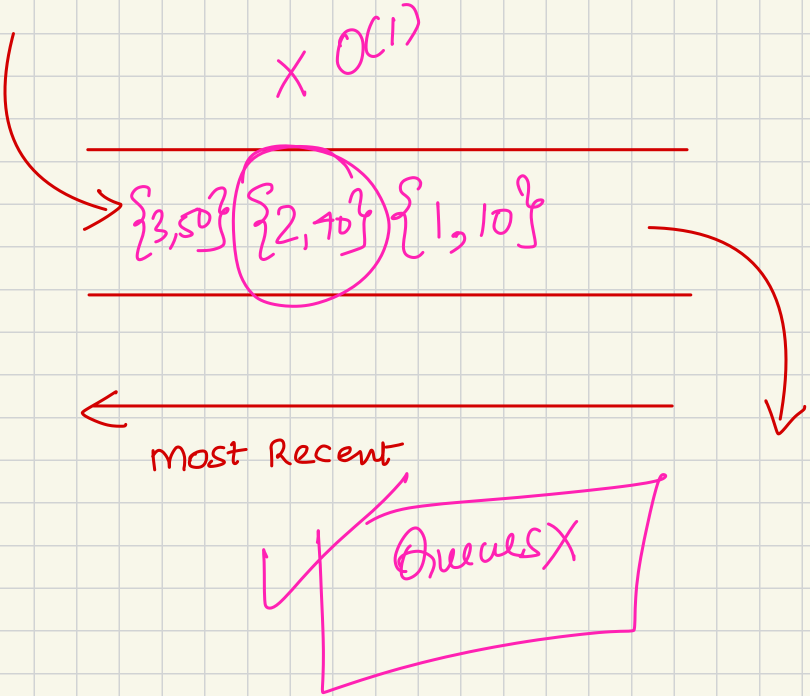
- 3
- 4
- ✓ 1

↑ most Recent

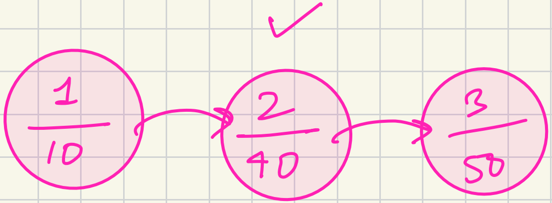
- ✓ set(1, 10)
- ✓ set (2, 40)
- ✓ set (3, 50)
- ✓ get(1) → 10
- set(4, 90)
- set(3, 90)
- set(5, 11)

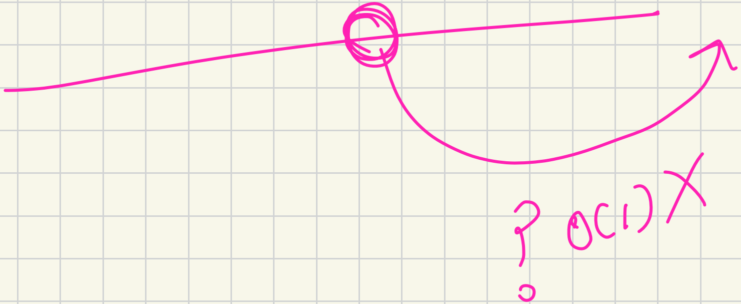


- ✓ set(1, 10)
- ✓ set(2, 40)
- ✓ set(3, 50)
- get(2) → 40
- set(4, 90)
- set(3, 90)
- set(5, 11)



- set(1, 10)
- set(2, 40)
- set(3, 50)
- ✓ get(1) → 10
- set(4, 90)
- set(3, 90)
- set(5, 11)



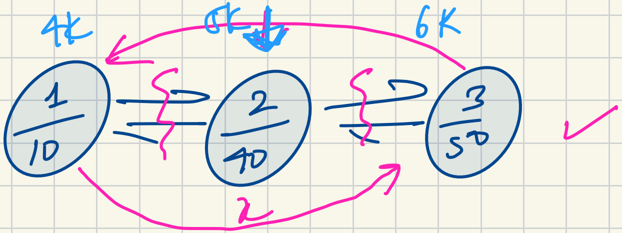


- set(1, 10)
- set (2, 40)
- set (3, 50)
- get(2) → 10
- set(4, 90)
- set(3, 90)
- set (5, 11)

doubly

↓

add TC: O(1) ✓



Node A = x.prev;

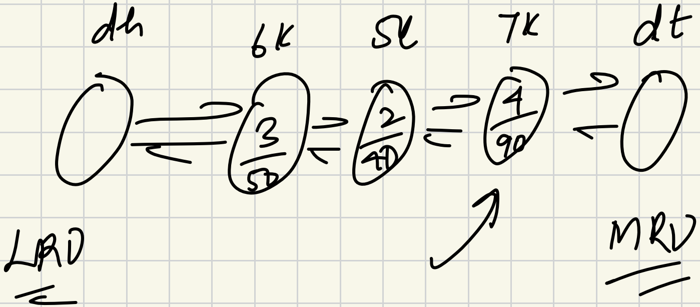
NodeB = x.next;

A.next = B;

B.prev = A;

Key	Add.
2	5k
1	4k
3	6k

- ✓ set(1, 10)
- ✓ set(2, 40)
- ✓ set(3, 50)
- ✓ get(2)
- ✓ set(4, 90)
- set(3, 90)
- set(5, 11)



~~1~~	~~1K~~
2	5K
3	6K
4	7K

addLast() }
remove() }

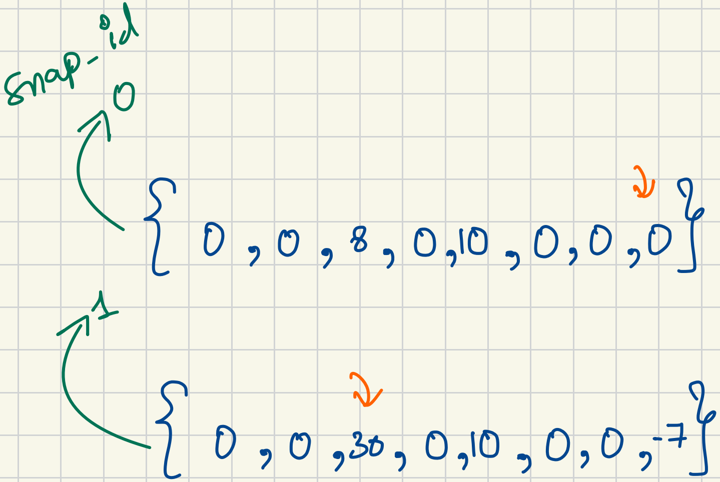
moveAtLast()

Snapshot - Array.

0	1	2	3	4	5	6	7

int[] arr = { 0 , 0 , 30 , 0 , 10 , 0 , 0 , -7 }

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



set (2, 8)

set (4, 10)

snap()

set (2, 30)

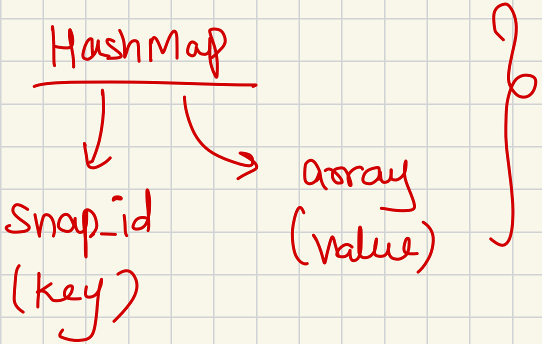
set (7, -7)

snap()

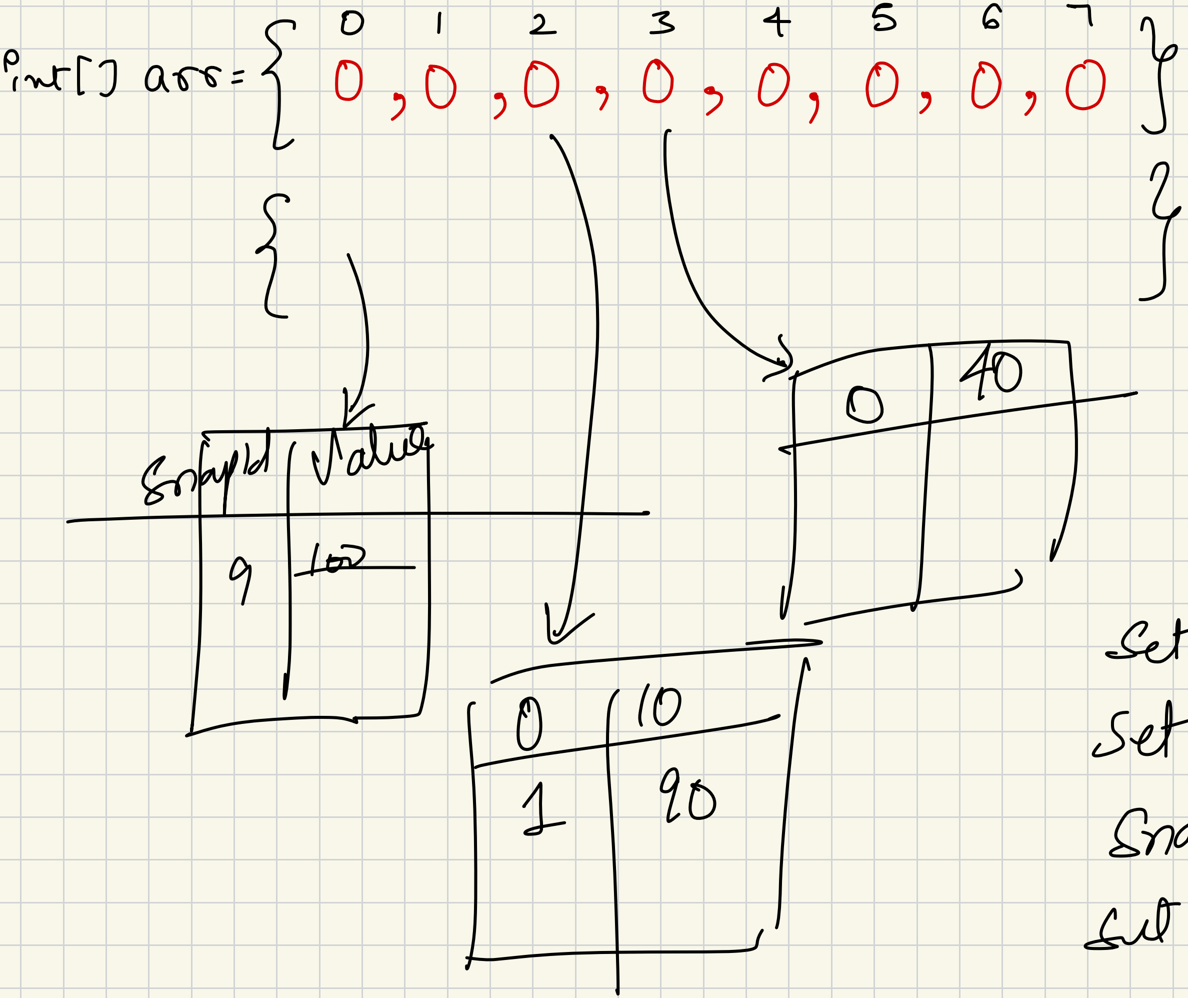
get(2, 1) ⟶ 30

get (7, 0) ⟶ 0

Brute force.



Lot of Memory is again initialized without any delta



set(2, 10)
set(3,40)
snap()
set(2,90)

longest subarray with equal freq of 0's, 1's & 2's.

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

freq 0 → 1
freq 1 → 1
freq 2 → 1

TC: O(N^2)
SC: O(1)

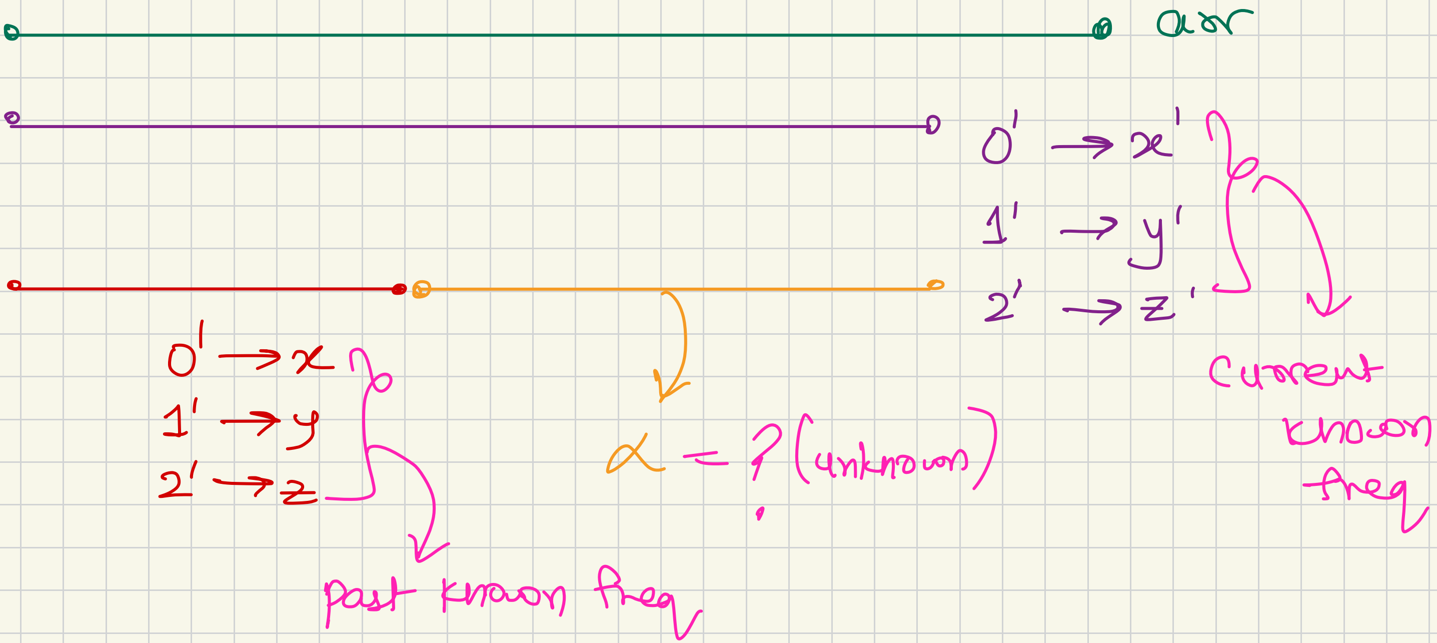
Brute force.

↳ Calc. all the Subarray

int cnt0
int cnt1
int cnt2

→ equal → Store longest among them

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



$$x' = x + \alpha \quad \text{———} \quad \textcircled{1}$$

$$y' = y + \alpha \quad \text{———} \quad \textcircled{2}$$

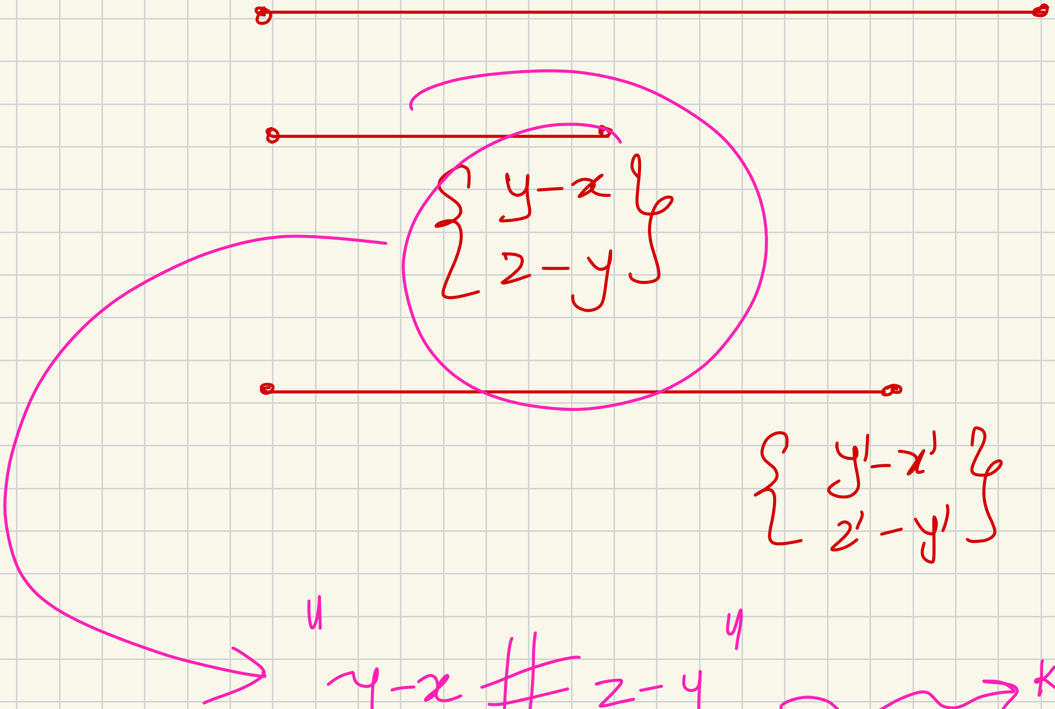
$$z' = z + \alpha \quad \text{———} \quad \textcircled{3}$$

$\textcircled{2} - \textcircled{1}$

$$y' - x' = y - x \quad \text{———} \quad \textcircled{4}$$

$\textcircled{3} - \textcircled{2}$

$$z' - y' = z - y \quad \text{———} \quad \textcircled{5}$$



"$y-x$ # $z-y$" ⟶ Key In HashMap

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

x	0	0	0	1	2	2	2	2	2	2	3	3
y	0	1	2	2	2	3	3	4	4	4	4	5
z	0	0	0	0	0	0	1	1	2	3	3	3
y − x	0	1	2	1	0	1	1	2	2	2	1	2
z − y	0	-1	-2	-2	-2	-3	-2	-3	-2	-1	-1	-2

2#-2

index

Len = i − index