Anagrams

str1 = "acaio" str2 = "oiacc" Yes they are anagramic

if all the characters of a string can be rearranged to form another string.

{'a', 'c', 'c', 'i', 'o'}

↓ sort

{a, c, c, i, o}

{a, c, c, i, o}

TC : O(N log$_2$ N)

SC : O(N)

"accio" ⟷ "ciaoc"

Map

Char	Freq
a	1
c	~~1~~ 2
i	1
o	1

Map 2

Char	Freq
[illegible]	~~2~~
[illegible]	1
[illegible]	1
[illegible]	1

TC : O(N) SC : O(26) ~ O(1)

① if size are not same return false
② verify all char has same freq

"aacc10"

a b c d e ... z

0 1 2 3 4 ... 25

1	2		1	1	

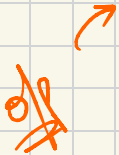
```
int pos = (ch - 'a')
```

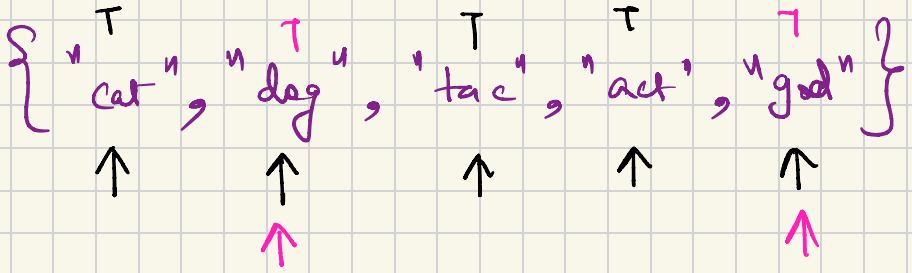
TC : O(N) SC : O(26) ~ O(1)

Group Anagrams

String[] arr = {"cat", "dog", "tac", "act", "god"}

{{cat, tac, act}, {"dog", "god"}}





Brute force

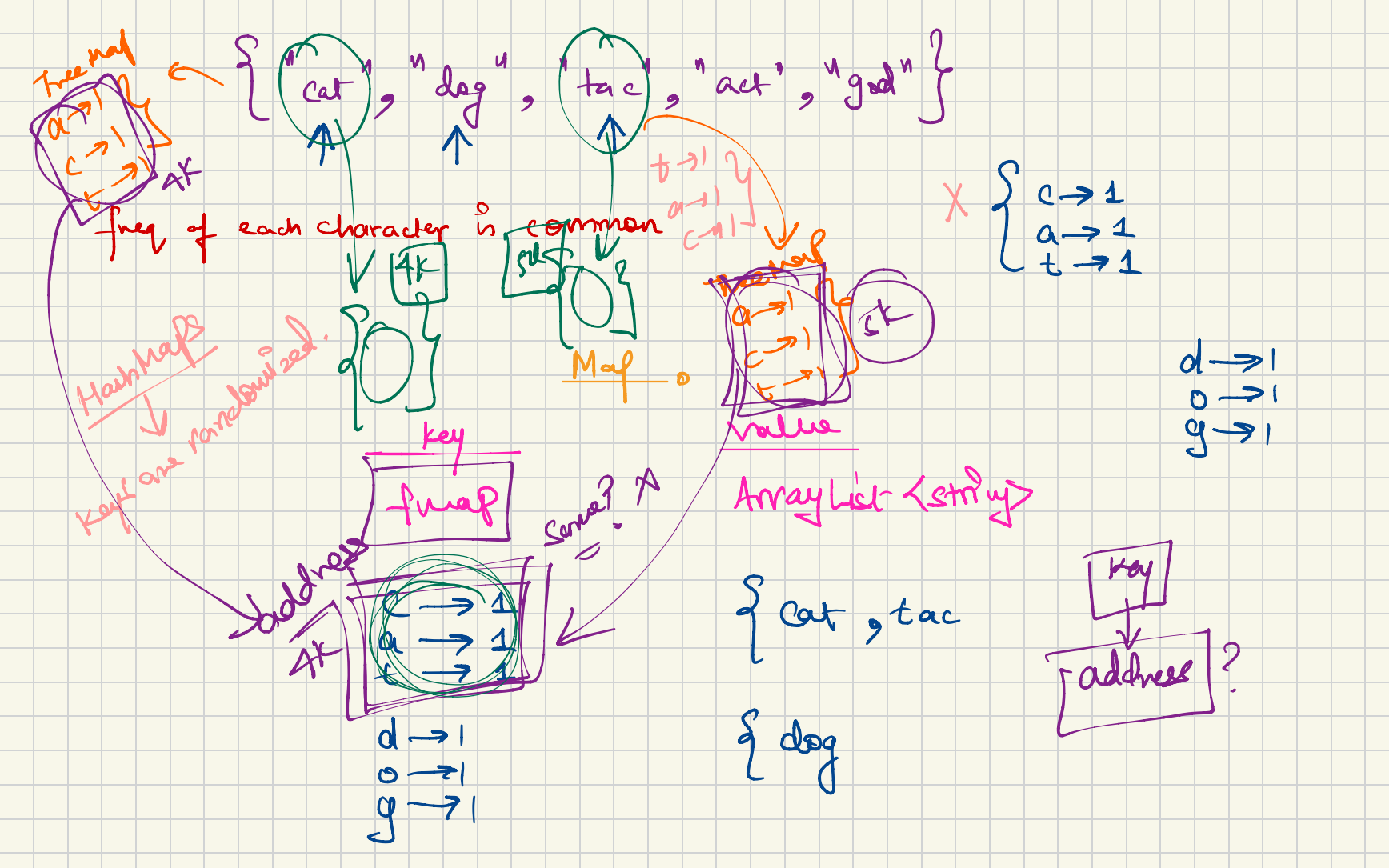
everyone will get a chance to start a group.

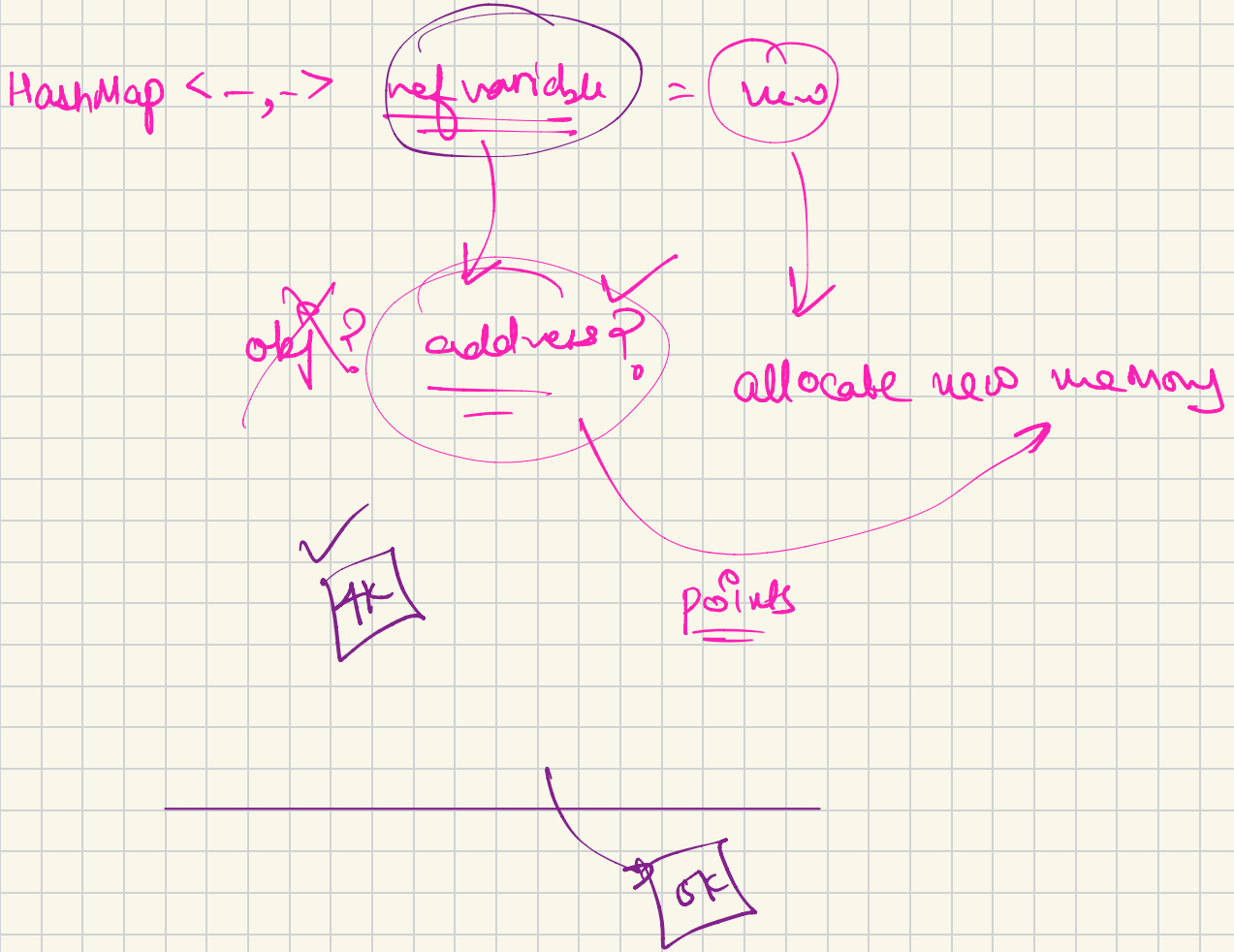
TC : O(N²) × O(M)

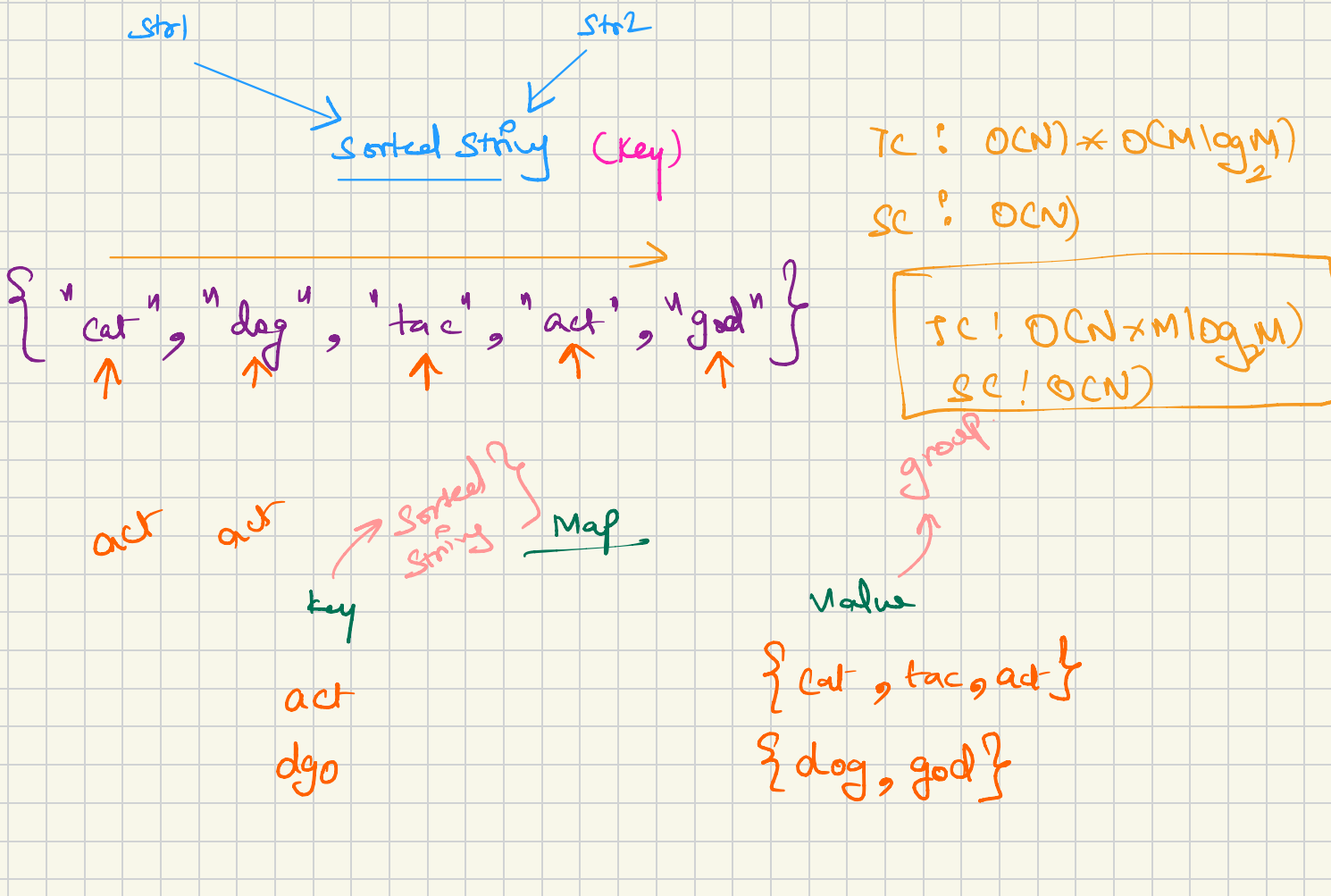
≈ O(N² M)

→ avg. length of the word

SC : O(N)


freq map
a → 1
c → 1
t → 1
4k
{ "cat", "dog", "tac", "act", "god" }
freq of each character is common
4k
50k
Map
t → 1
a → 1
c → 1
freq map
a → 1
c → 1
t → 1
5k
value
X
{ c → 1
a → 1
t → 1
HashMap
Keys are randomized.
key
fmap
address
4k
Same?
c → 1
a → 1
t → 1
d → 1
o → 1
g → 1
ArrayList<String>
{ cat, tac
{ dog
d → 1
o → 1
g → 1
key
address ?


HashMap <-,-> ref variable = new
obj ?
address ?
allocate new memory
FF
points
0F


Str1
Str2
Sorted String (Key)
{ "cat", "dog", "tac", "act", "god" }
act act
Sorted String
Map
key
act
dgo
group
Value
{ cat, tac, act }
{ dog, god }
TC : O(N) * O(M log M)
SC : O(N)
TC : O(N*M log M)
SC : O(N)

"accio"

"caioc"

↓

↓

{ 0 1 2 x y }

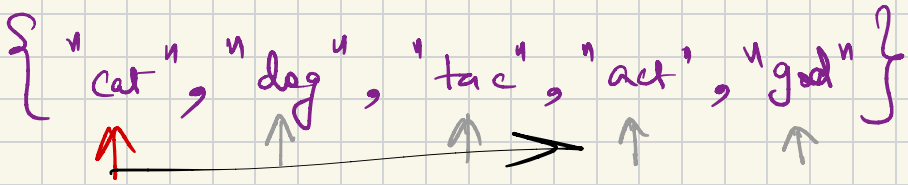
{ 1 2 1 1 }

→

a1c2i101

O(M)

encoding!



a|c|t|

a|c|t|

TC: O(N x M)

SC: O(N)

Map

Key	Value
a|c|t|	{ cat, tac, act }
d|g|o|	{ dog, god }

Minimum window substring

str1 : a d b ae bb a ca f da

str2 : cb ab d

Brute force

(1) generate all the substrings and try to find all char of str2 in window

TC: $O(N^2 \times M)$ SC : $O(N)$

exc

str1 : a d b ac bb a ca f da

inc

len = inc - exc

= 6 5

str2 : cb ab d

freqMap2

c → 1
b → X2
a → 1
d → 1

freqMap1

a → 1 / 2 3
f → 1
b → 1
c → 1
d → 1

TC : O(M) + O(N)

≈ O(N+M)

SC : O(1)

exc

str1 : a d b ac bb a ca f da

str2 : cb ab d

dmcnt = 5

freq Map2

c	→	1
b	→	2
a	→	1
d	→	1

mcnt = ~~1~~ ~~2~~ ~~3~~ 4

freq Map1

a	→	~~1~~ ~~2~~ 3
f	→	1
b	→	1
c	→	1
d	→	1

len = ~~6~~ 5

{ TC : O(N+M) SC : O(26) + O(26) ~ O(1) }

Agenda:

1. LRU Cache
2. SnapShot Array
3. Longest Subarray with Equal Freq of 0, 1, 2

Longest Subarray with equal freq of 0, 1 and 2.

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

Brute Force

- Calc Subarray, store count 0, 1, 2
- store the maxLen

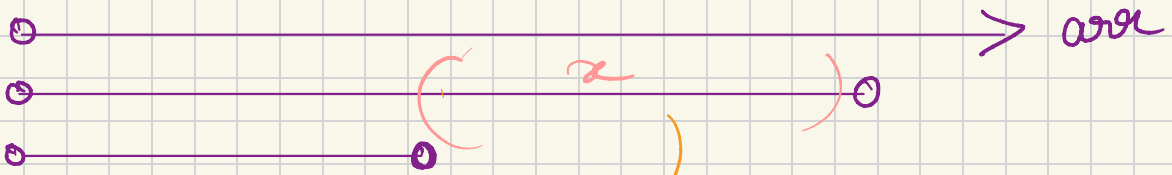
```
for (int i = 0 ⟶ n)
{  int cnt0, cnt1, cnt2
   for (int j = 0 ⟶ i)
   {  ++
      if ____
```

TC: O(N²)
SC: O(1)

len = 3 - (cnt0)

int arr = { 0, 1, 2, 0, 0, 1, 2, 0, 1, 2, 2, 2, 1, 0, 1 }

(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14)



0 → α
1 → β
2 → γ

0 → α'
1 → β'
2 → γ'

unknown

α' = α + x ——— (1)

β' = β + x ——— (2)

γ' = γ + x ——— (3)

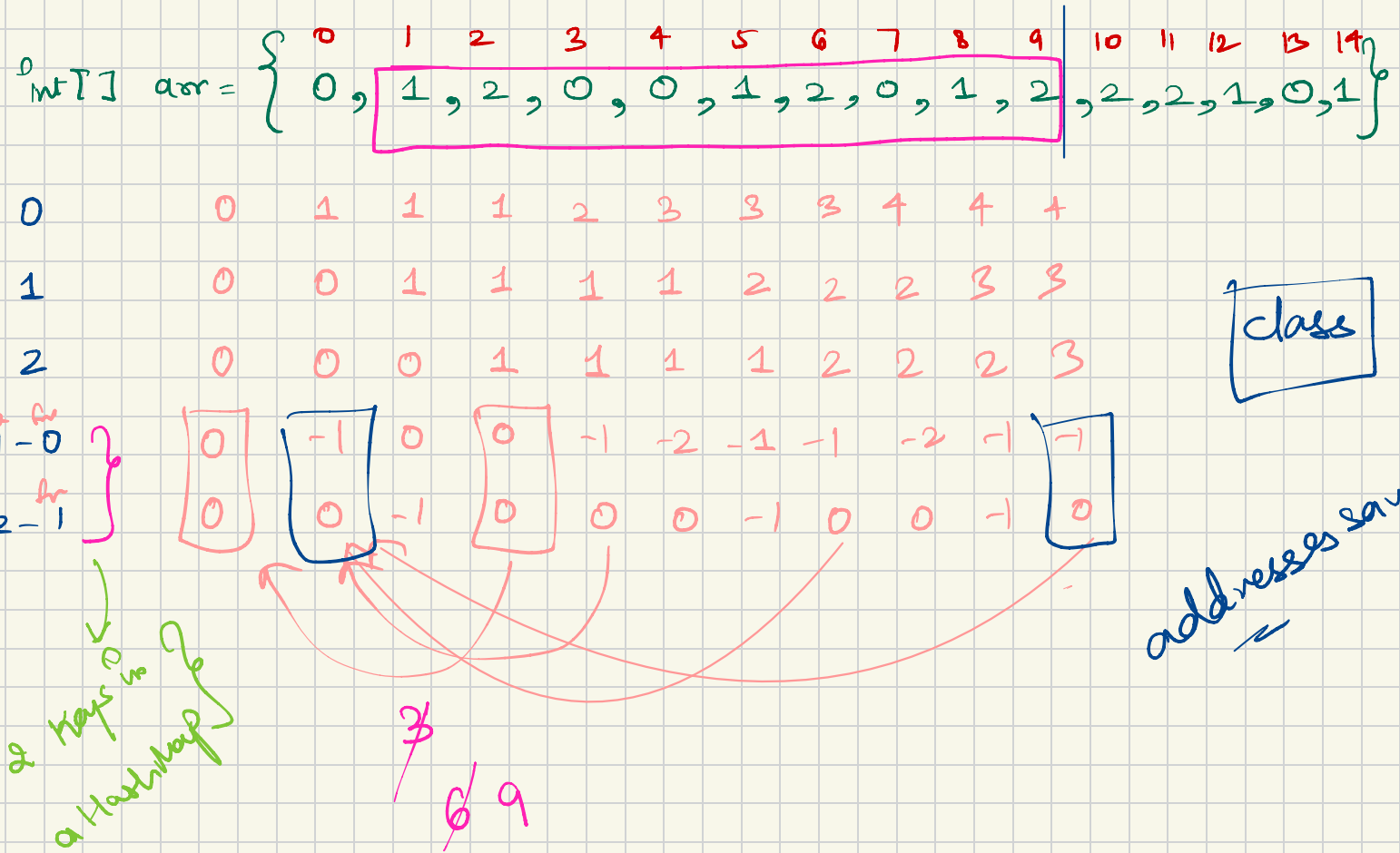
(2) − (1)

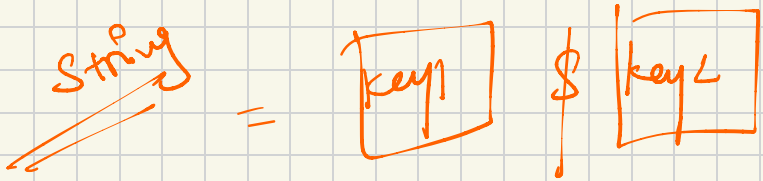
$\beta' - \alpha' = \beta - \alpha$ ——— (4)

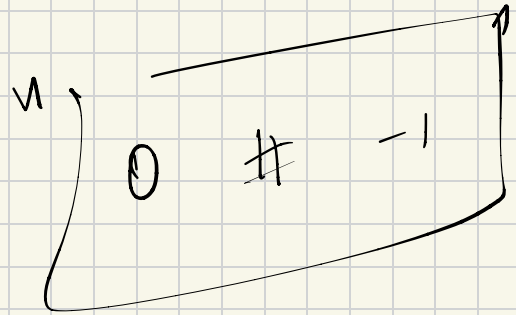
(3) − (2)

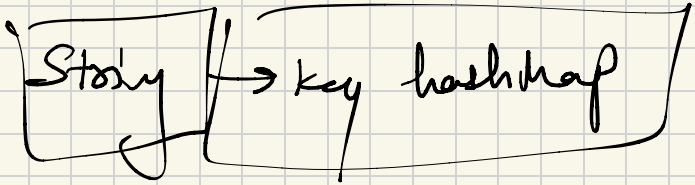
$\gamma' - \beta' = \gamma - \beta$ ——— (3)

difference of current freq 1,0 2,0 → same

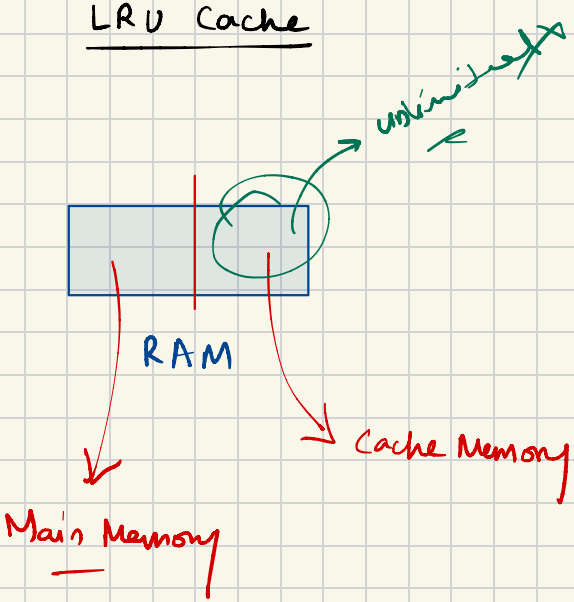

int[] arr = { 0, 1, 2, 0, 0, 1, 2, 0, 1, 2, 2, 2, 1, 0, 1 }
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0 0 1 1 1 2 3 3 3 4 4 4
1 0 0 1 1 1 1 2 2 2 3 3
2 0 0 0 1 1 1 1 2 2 2 3
fr 1 - fr 0
fr 2 - fr 1
0 -1 0 0 -1 -2 -1 -1 -2 -1 -1
0 0 -1 0 0 0 -1 0 0 -1 0
class
addresses same?
2 keys in a HashMap
3
6 9

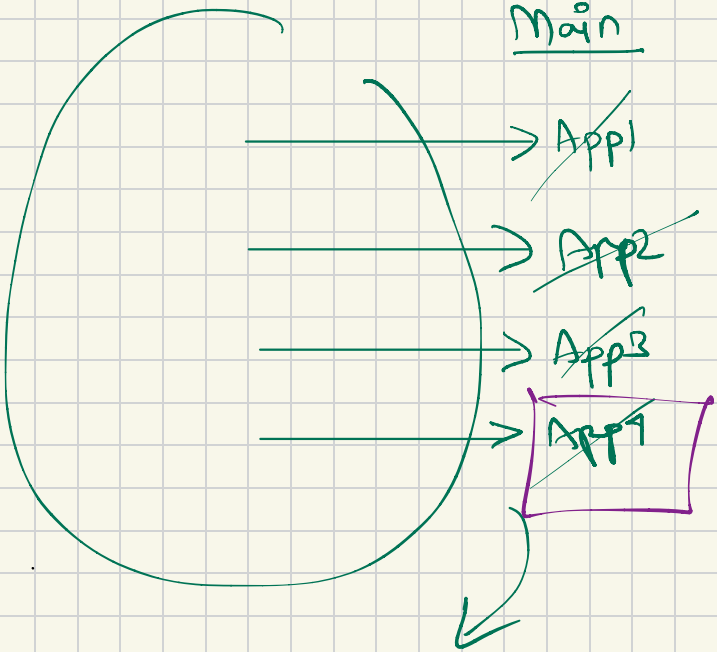

String
=
key1
$
key2


N
0
#
-1


String
key hashMap

LRU Cache





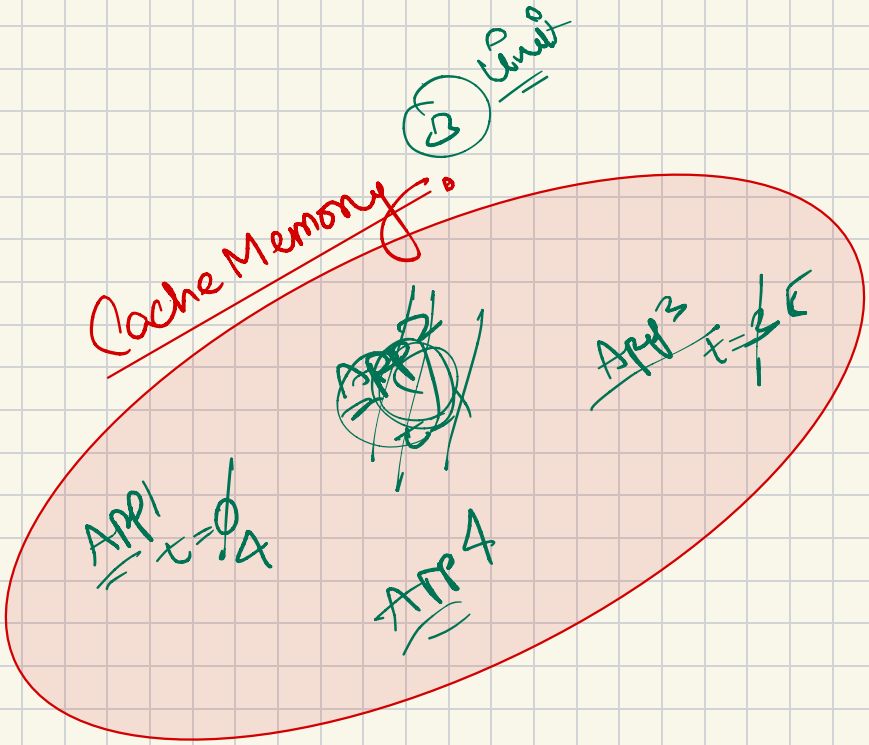
3 App

Cache

App1 App2 App3

Cache Memory Management Algo

- least recently used (LRU)
- least frequently used (LFU)



App1 → opened

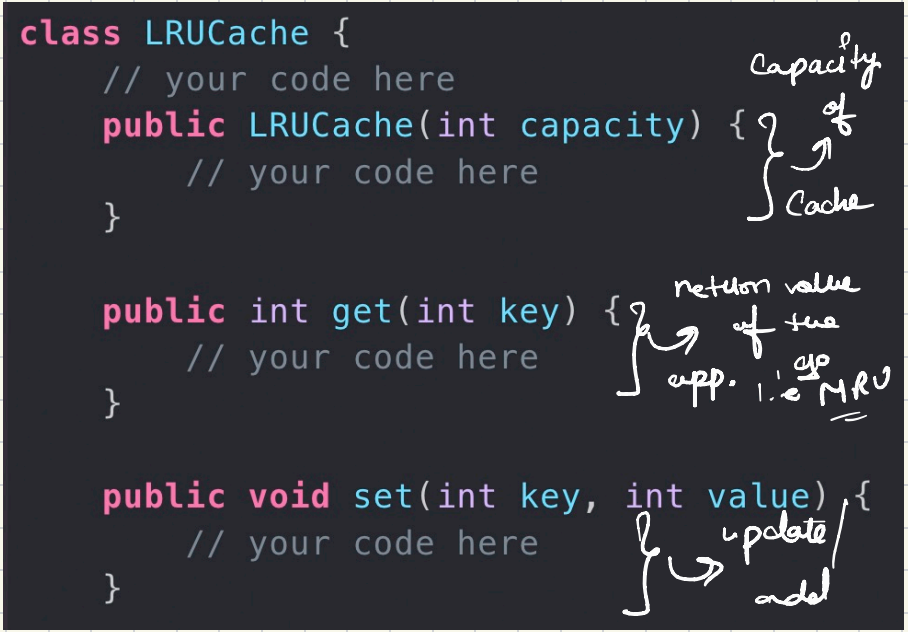
App2 → "

App3 →

App4 → popup

App2 → updated

App1 → open


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
}
Capacity of Cache
return value of the app. i.e MRU
update/ add

set(1, 10)

set(2, 30)

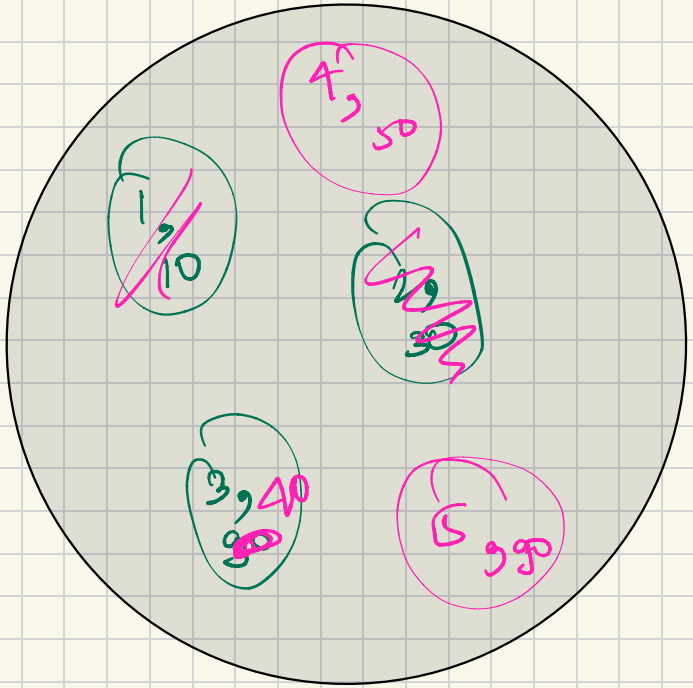
set(3, 90)

get(1) ⇝ 10

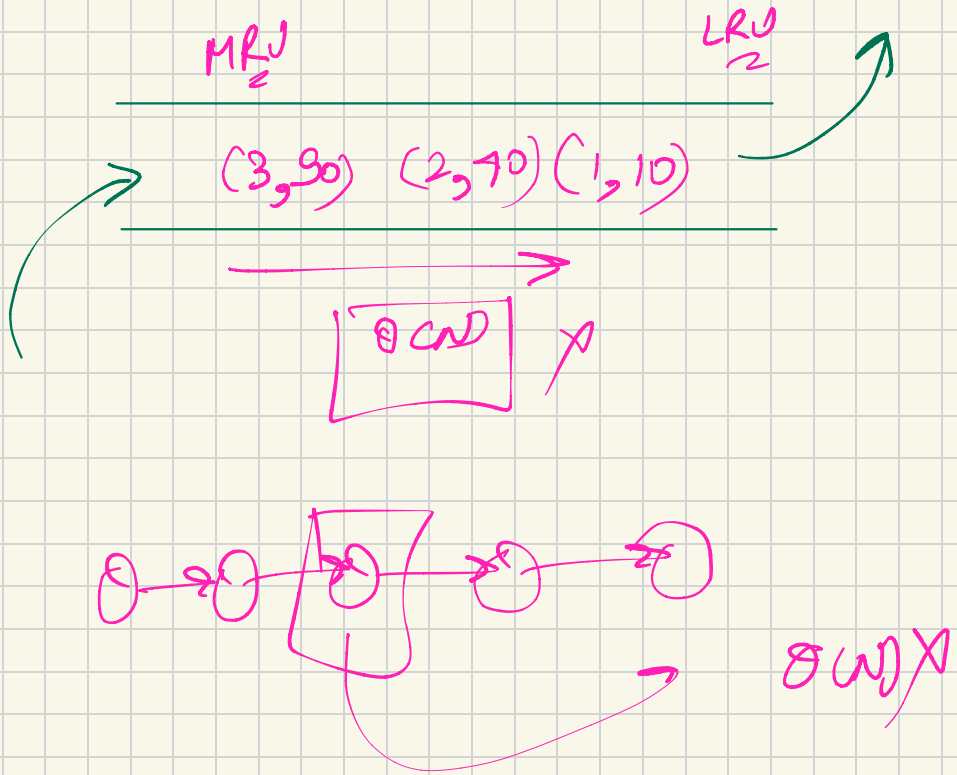
set(4, 50)

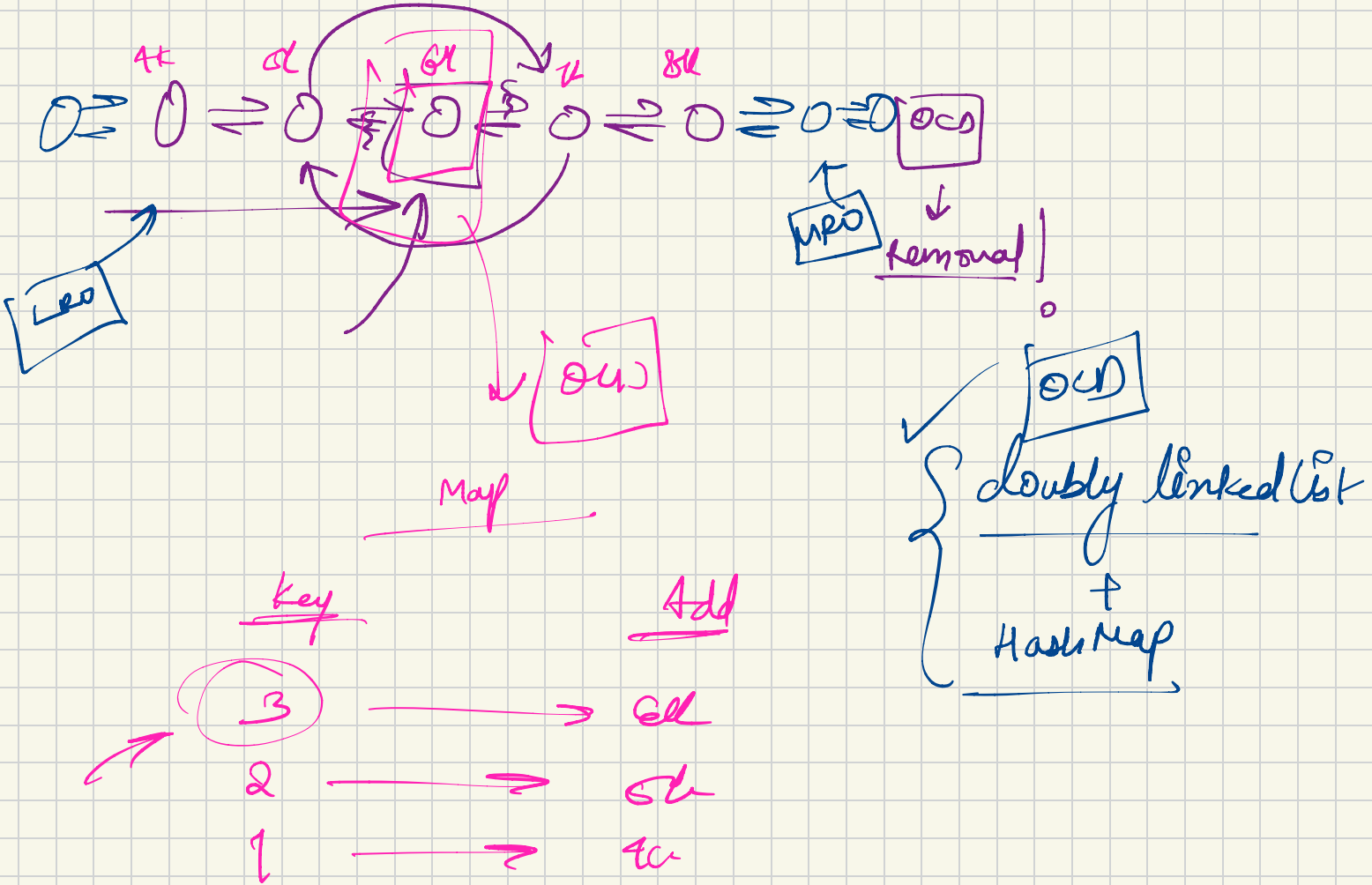
set(3, 40)

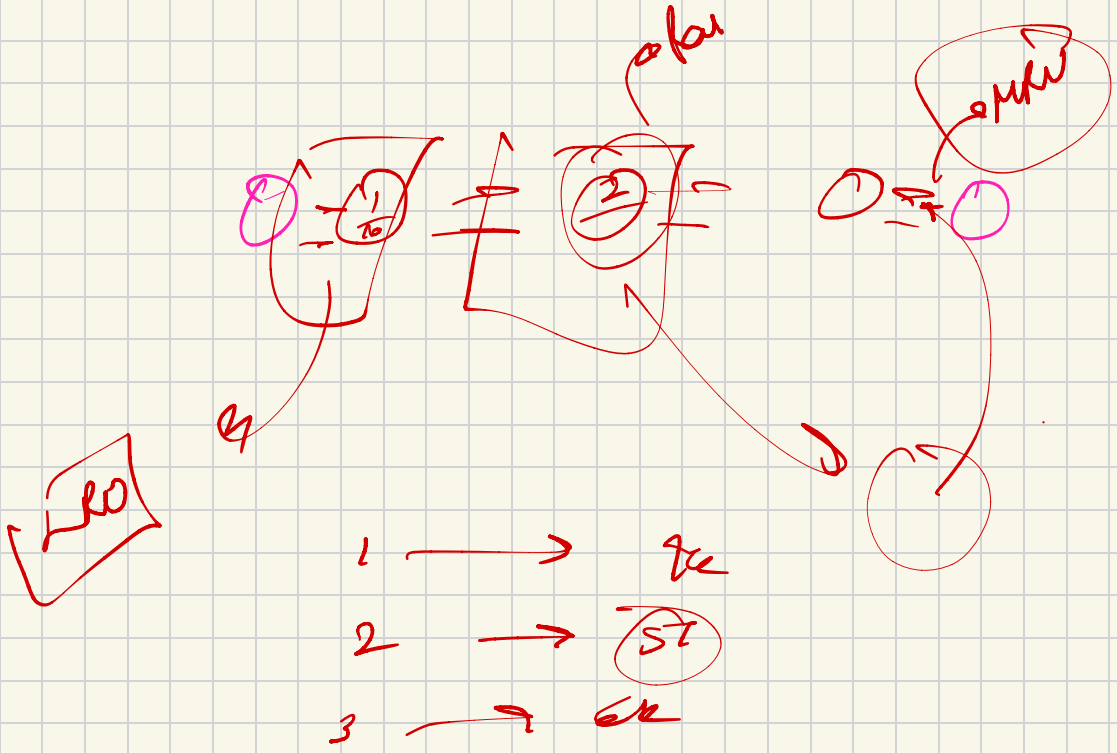
set(5, 90)



Cache


MRU
LRU
(3,30) (2,40) (1,10)
O(N)
O(N)


4K
6K
8K
LRU
MRU
removal
Map
Key
Add
3
2
1
doubly linked list
+
HashMap



Snapshot Array

```
int[] arr = { 0, 0, 8, 0, 3, 0, 0 }
```
(indices 0 1 2 3 4 5 6)

set(2, 10)

set(4, 3)

snap()

set(2, 8)

snap()

get(2, 0) ⇝ 10

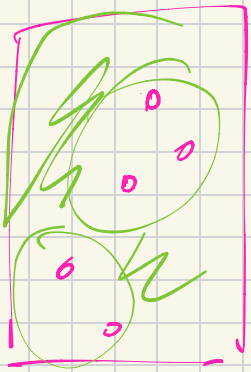
get(3, 1) ⇝ 0

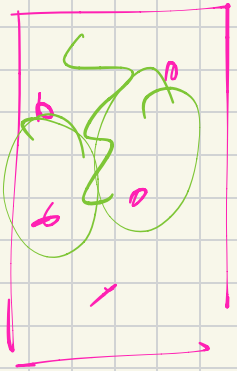
snap = 0

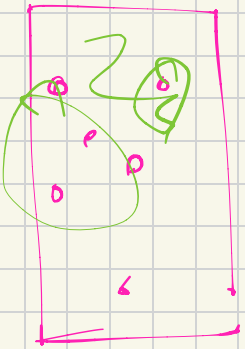
```
{ 0, 0, 10, 0, 3, 0, 0 }
```
(indices 0 1 2 3 4 5 6)

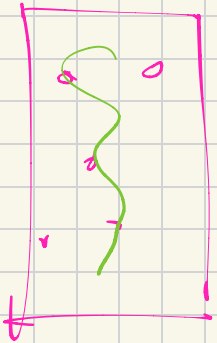
snap = 1

```
{ 0, 0, 8, 0, 3, 0, 0 }
```
(indices 0 1 2 3 4 5 6)

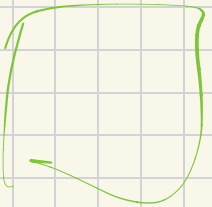


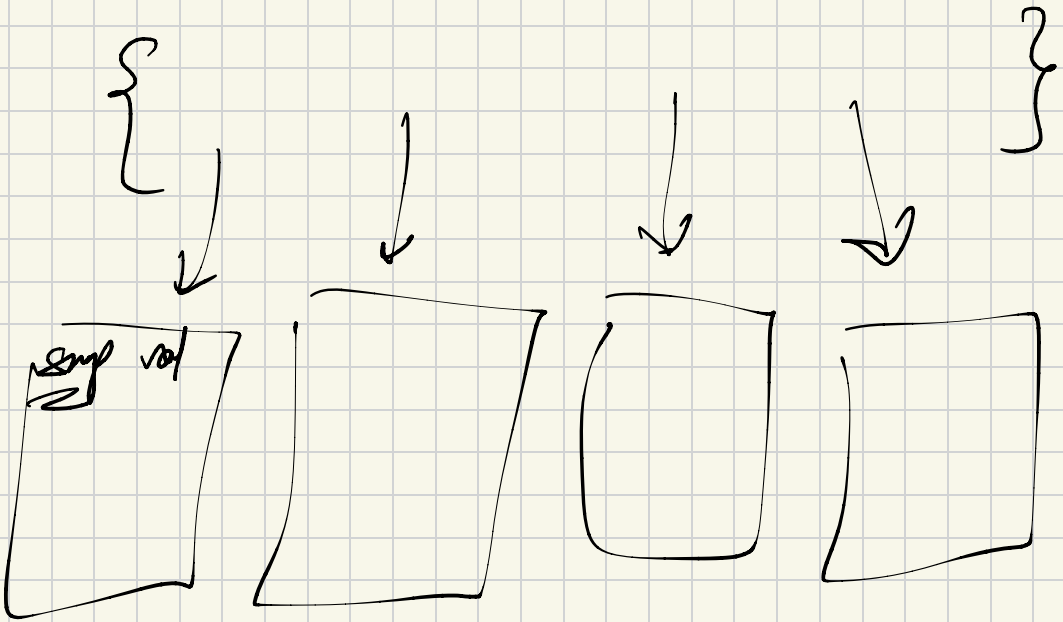


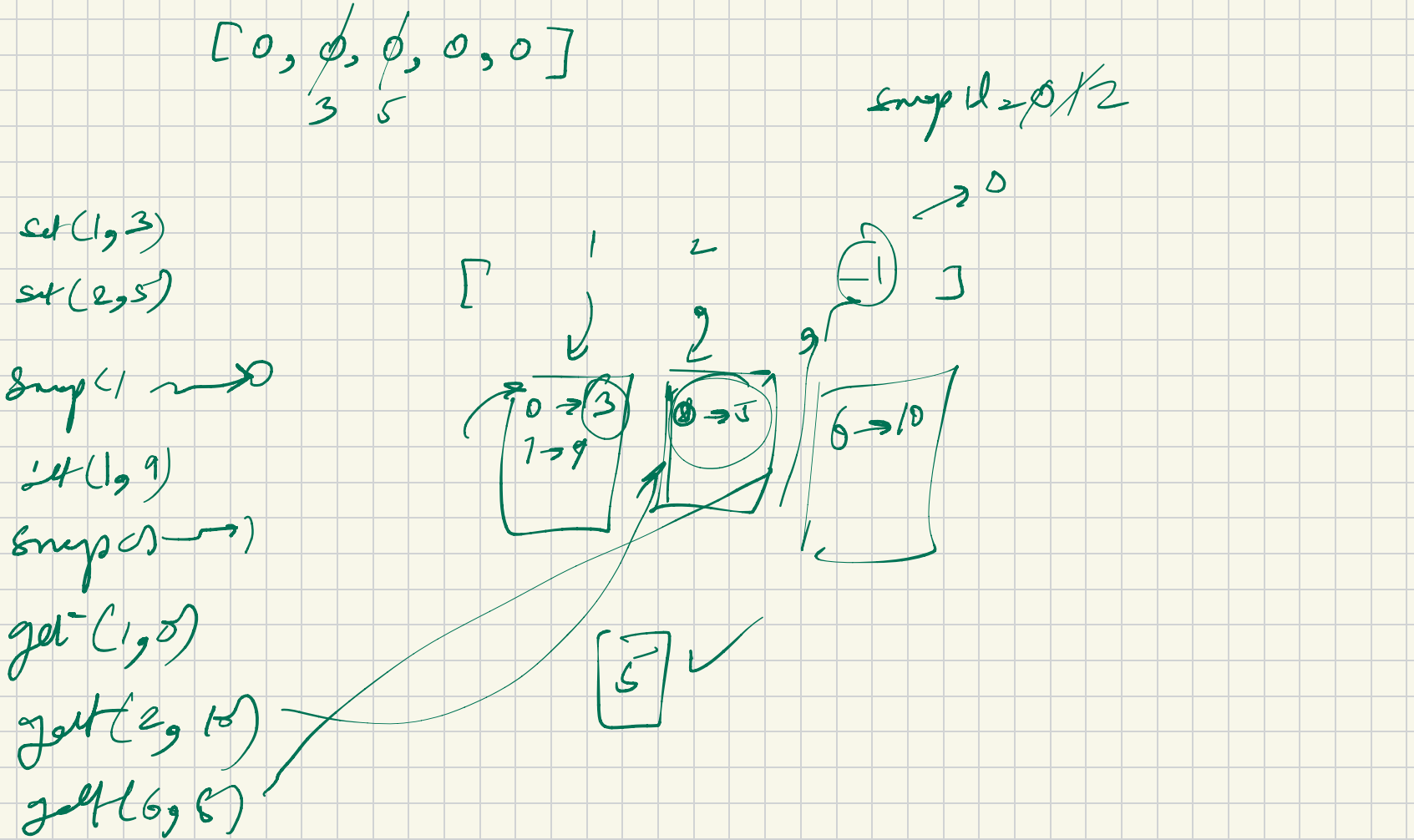




4k ⟹ Seperate






[0, ∅, ∅, 0, 0]
3 5
swap id=∅ 1/2
set(1,3)
set(2,5)
swap(1
get(1,9)
swap(5
get(1,0)
get(2,10)
get(6,5)
[1 2]
0 → 3
1 → 9
∅ → 5
6 → 10
5