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	

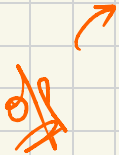
```
int pos = (ch - 'a')
```

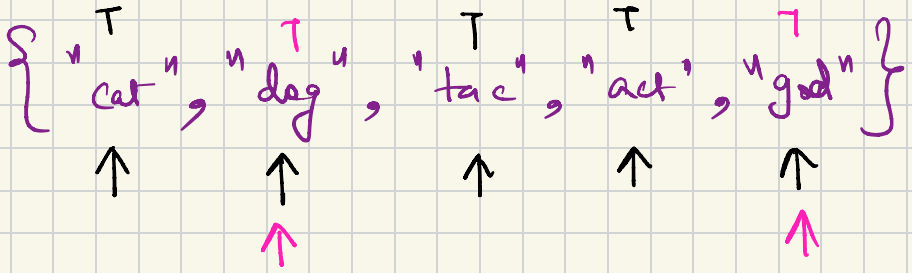
TC : O(N) SC : O(26) ~ O(1)

Group Anagrams

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

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





Brute force

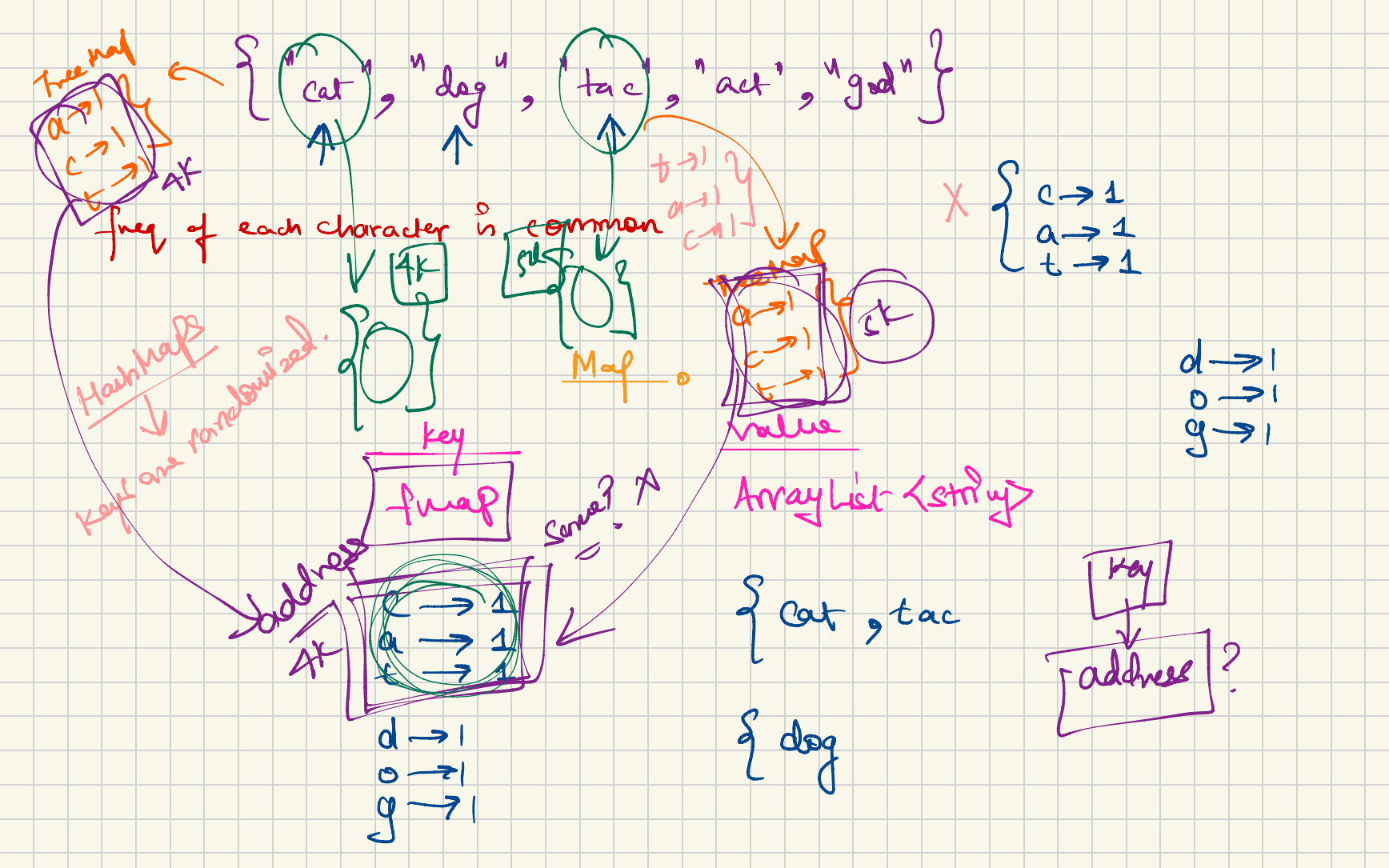
everyone will get a chance to start a group.

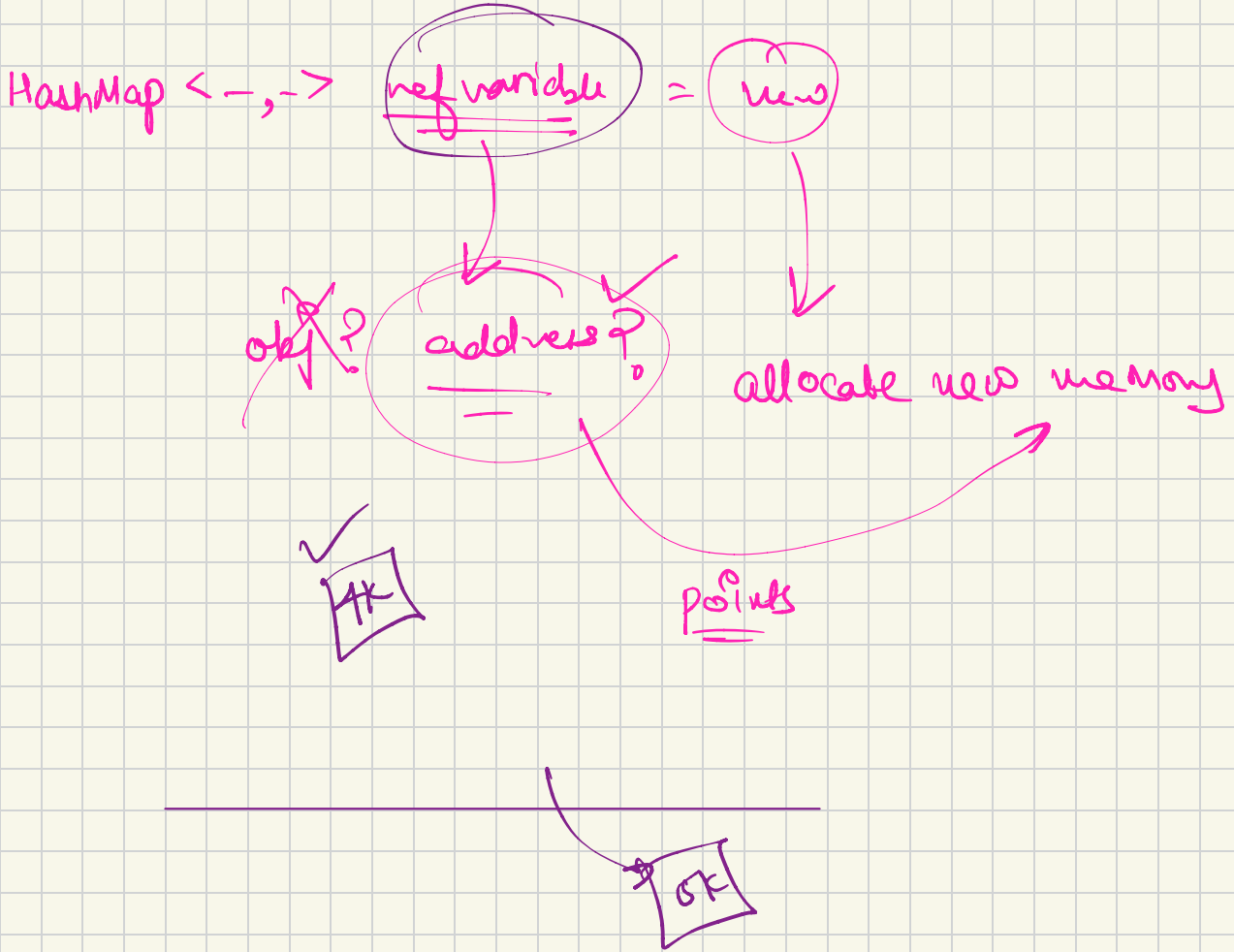
TC : O(N²) × O(M)

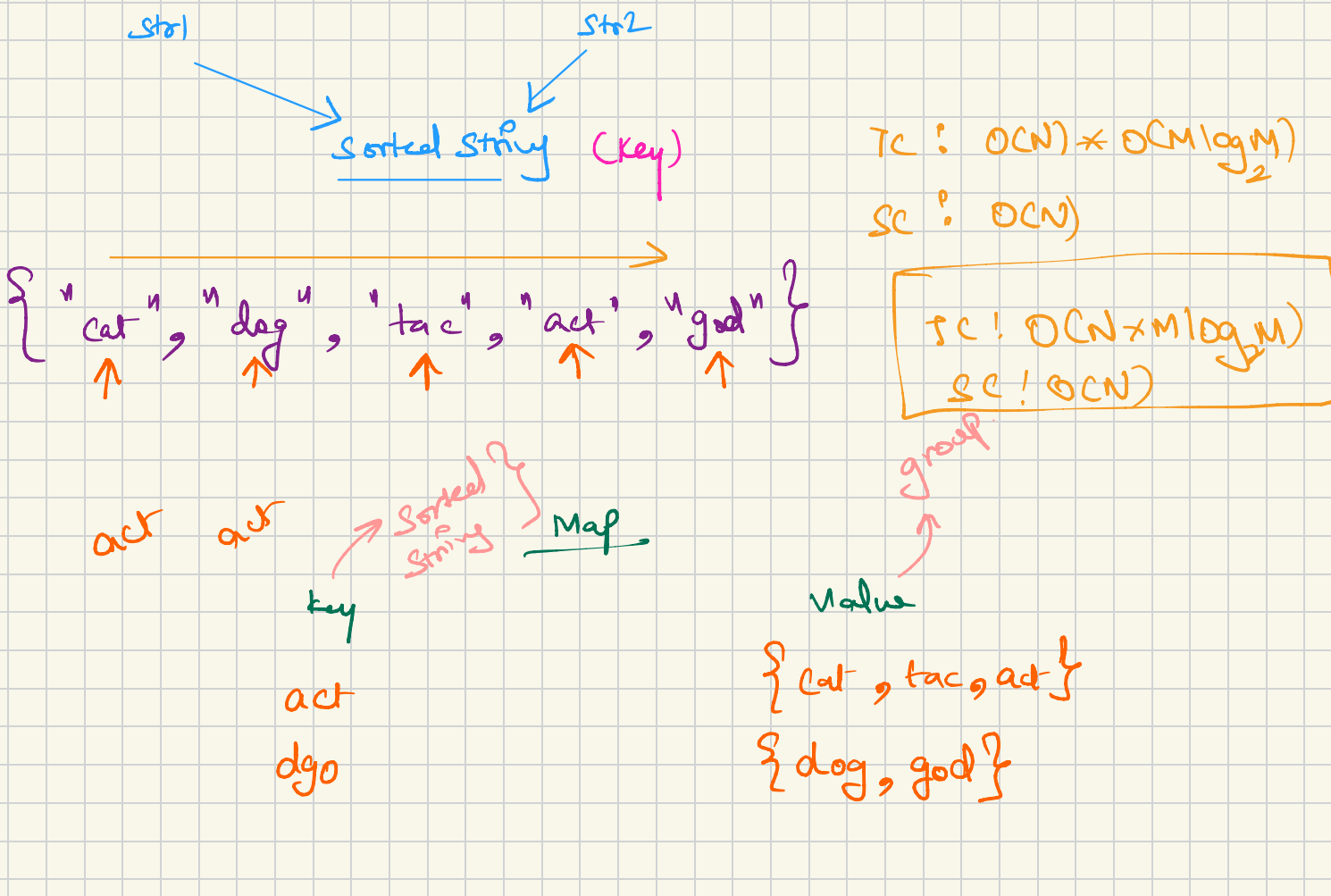
≈ O(N² M)

→ 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 }

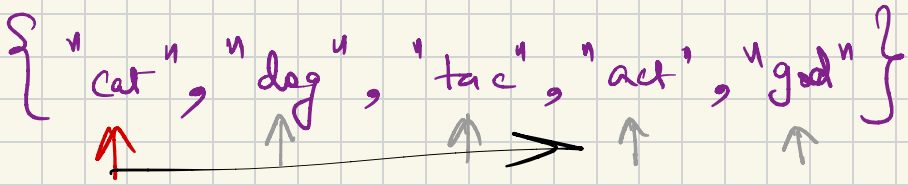
{ 1 2 1 1 }

→

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 : _____

str2 : _____

a d b ae bb a ca f da

cb ab d