Equilibrium Index

```
int[] arr = { 5, 4, 6, 7, 3, 9, 3 }
            0  1  2  3  4  5  6
```

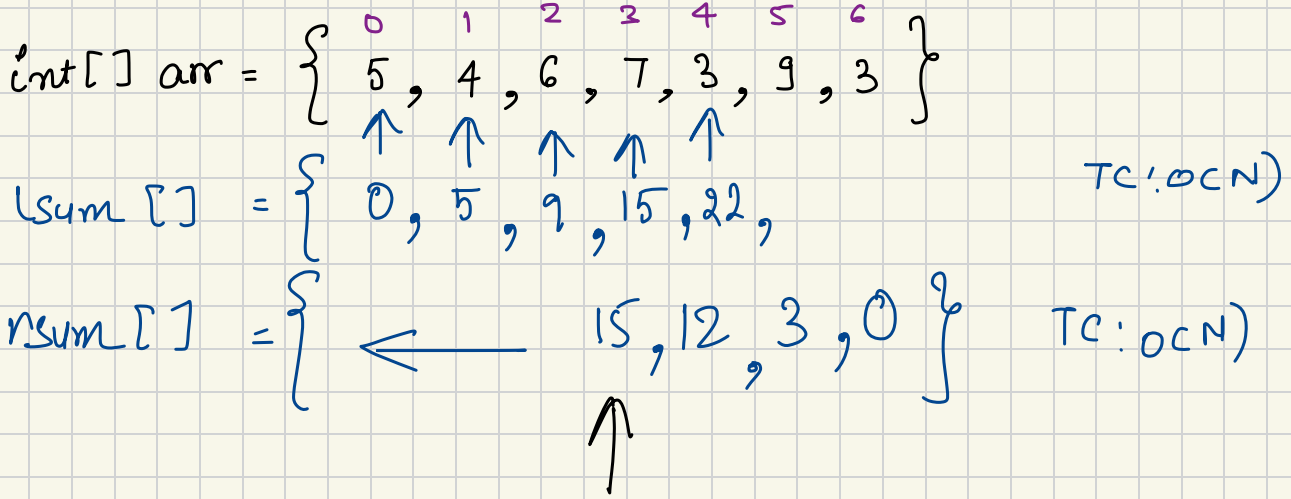
15 | 7 | 15

equi. index = 15

Brute force

TC: $O(N^2)$

SC: $O(1)$



TC : O(N) + O(N) + O(N) ≈ O(N)

SC : O(N) + O(N) ≈ O(N)

int[] a = { 5, 4, 6, 7, 3, 9, 3 }

0 1 2 3 4 5 6

↑

int totalSum = 37 TC: O(N), SC: O(1)

int lsum = ~~0~~ ~~5~~ ~~9~~ 15

rsum = totalSum - lsum - arr[i];

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

Count Number of Pairs with Absolute diff. K

int[] arr = { 14, 29, 29, 39, 49, 49, 10 } K = 10

Cnt = 0 → 2

(x, y)

↳ x − y = K

y = x − K

Key	freq
14	1
29	2
39	1
49	2
10	1

x = 14, y = 4

x = 29, y = 19

x = 39, y = 29

x = 49, y = 39

x = 10, y = 0

arr = { 14, 29, 29, 39, 49, 49, 10 } k = 0

cnt = ~~0~~ ~~1~~ 2 ✓

14	1
29	2 ✓
39	1
49	2
10	1

$x - y = k$

$y = x - k$

x = 14
y = 14

x = 29
y = 29

{(29, 29) (49, 49)}

Agenda

1. group anagrams
2. Minimum window substring
3. Distinct windows
4. Substring with K unique characters

Group Anagrams.

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

o/1 : { { "act", "tac", "cat" }, { "dog", "god" } }

Brute force

TC : O(N) x O(N) x O(M) ≈ O(N² x M)

SC : O(1)

N → size of array

M → avg length of the word

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

HashMap

key (String)	value (AL)
"act"	{"act", "tac", "cat"}
"dog"	{"dog", "god"}

TC: O(N x N x M) ≈ O(N²M)

SC: O(N)

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

"act", "tac", "cat"

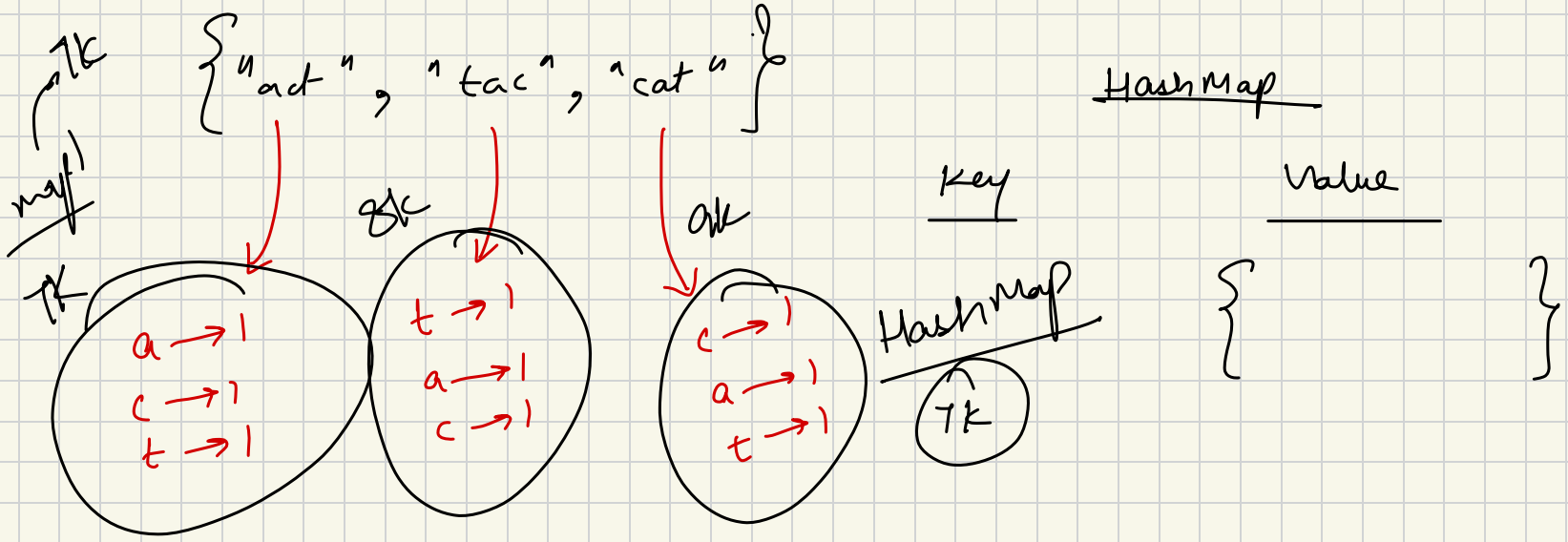
sort(str) → "act"

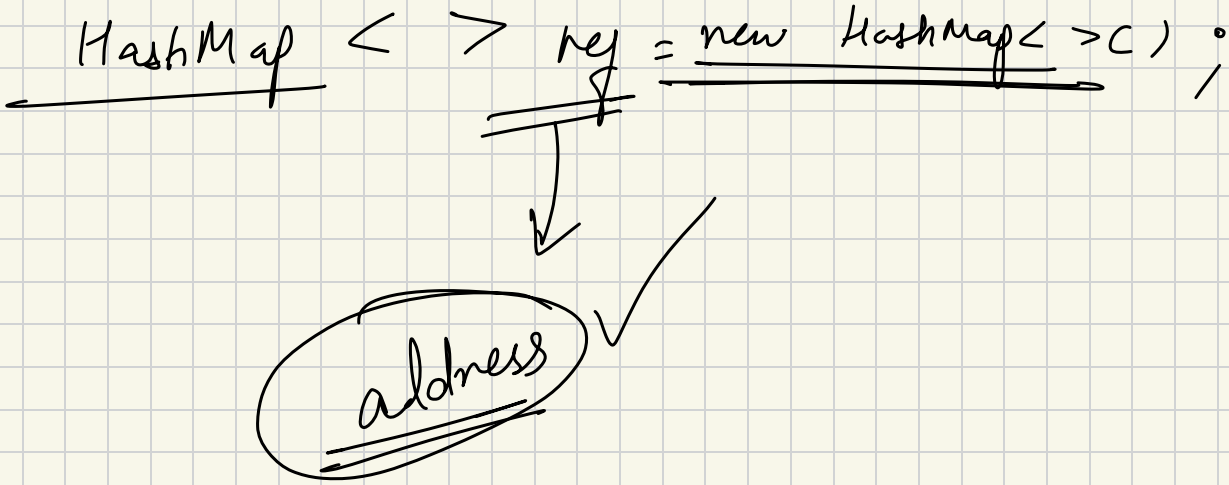
key	value
"act"	{"act", "tac", "cat"}
"dgo"	{"dog", "god"}

dgo

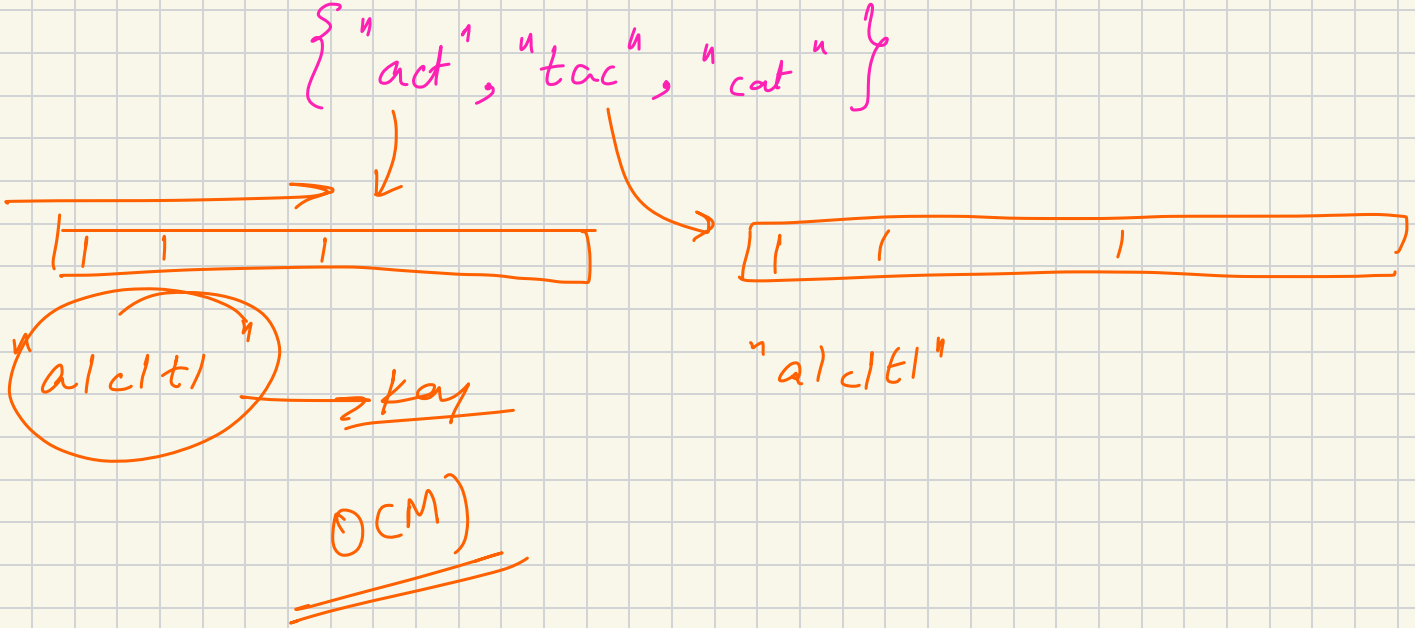
TC: O(N × M log M) SC: O(N)

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




HashMap < > ref = new HashMap< >() ;
address

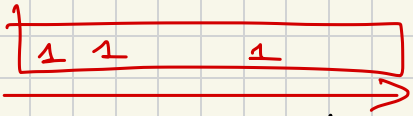
String[] words = {"act", "tac", "dog", "cat", "god"}



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

freqMap

"a1c1t1"



"a1c1t1" "d1g1o1"

Key (String)	Value (AL)
a1c1t1	{ "act", "tac", "cat" }
d1g1o1	{ "dog", "god" }

TC: O(N x M) SC: O(N)

Minimum window substring.

str1 = " d b e a c d b c c a b "

str2 = " a b b c d c "

Brute force

↳ generate map of all the substring of str1 to try to accumulate str2 in it.

TC: O(N²M) SC: O(1)

str1 = " d b e a c d b c c a b "

enc

inc

dbccab

len

substring(idx1, enc+1)

str2 = " a b b c d c "

fmap1

fmap2

a → 1

c → 2

b → 2

d → 1

b → X 2

a → 1

c → X 2

str1 = " d b e a c d b c c a b "

str2 = " a b b c d c "

end ↓ (d)

mc ↑ (o)

len = ~~beacdbc~~ dbccab

mct = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

dmcnt = 6

fmap1

b → ~~1~~ 2

a → 1

c → ~~1~~ 2

fmap2

a → 1

b → 2

c → 2

d → 1

TC: O(M+N)

SC: O(1)

H.W.

- Smallest substring with all character
- Max^m Ones after modification