Valid Anagram

- given strings S & T, check if T anagram of S = True
  otherwise = False

=> Create 2 hashmaps (1 for S, 1 for T)

=> { keys: characters of string : values: freq. of the char. (key) }

Count S	
c	1
a	1
r	1

Count T	
a	1
r	1
c	1

S= car
T= arc

=> If keys are similar as well as values similar
↳ return True

Two Sum

- Given Int. Array return Indices that equal to Target
- The array has exactly 1 solution

=> One Pass Method

- use hashmap to track differences

```
       0    1     2     3
ex: [ 2,   7,   11,   15 ]      Target = 9
```

∴ Keys = Value of elements ; Values : Index of the element

prevMap

2	0
7	1

=> 9 - 2 = 7   => 9 - 7 = 2 → If 2 is in hashmap we found Indices that = target

↗ If 7 is in Array we found Indices that = target

return [prevMap[diff], i]

-Group Anagrams

//good for learning fundementals

∴ two strings are anagrams if we take e/a & sort them they will be the same

I.E: $Tan = S$, $nat = T$ (sort S&T)

$\Rightarrow anT = S$, $anT = T$ (These are equal)

∴ This method takes nlogn time, where n is the average length of the strings. We also have to do this M times, and M is the collection of strings

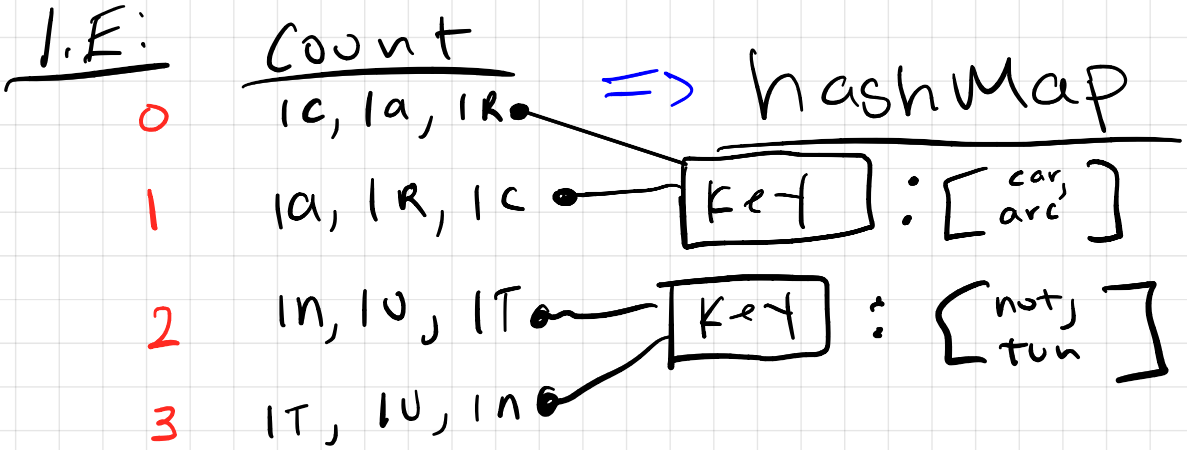
$$\text{I.E} \Rightarrow M = [\overset{0}{\underbrace{\text{"car"}}_{n=3}}, \overset{1}{\underbrace{\text{"arc"}}_{n=3}}, \overset{2}{\underbrace{\text{"nut"}}_{n=3}}, \overset{3}{\underbrace{\text{"tun"}}_{n=3}}, \ldots,]$$

$$\Rightarrow O(M \cdot n\log n)$$

$\Rightarrow$ We can do better than this...

∴ Strings will be (a-z), thus at most we have 26 unique chars.

↳ This lets us create an array → count where we count frequency of e/a unique character



// keys are freq. of chars of a str.

// values are the list of anagrams w/ similar count keys

∴ Because using hashMap, & just counting chars & there is a limit of 26 chars

$$\Rightarrow O(\underbrace{m}_{\text{total \# of input str.}} \cdot \underbrace{n}_{\text{avg. len of a str}} \cdot 26) = \Theta(m \cdot n)$$