Anagram

↳ HEART   EARTH   (both are permutation of each other)

EARTH

ex → a b a a c   abacc

LISTEN   SILENT

$\boxed{\text{S and t}}$

rat   car

cares race

Observations

1. if the length of the strings are different then we can never form permutations of each other.

2. Once we know that length is same, then all we need to ensure that there is no mismatch in char.

unique char should be same ← (both string should posses same set of chars with same frequency of occurrence

<K,V>

→ frequency map

Note → Order doesn't matter

Ex

S → ANAGRAM ↓i

t → NAGARAM ↑i

key , value

↓ char ↳ freq

insert delete search → O(1)

{

~~'A' : 0,~~

~~'N' : 0,~~

~~'G' : 0,~~

~~'R' : 0~~

~~'M' : 0~~

}

we can one by one process char of t, & if we find this char in the mapping we will reduce the freq.

Once freq becomes 0, remove the char from mapping.

↓

at last if mapping is empty we have pair of anagram.

$s = aab$ $\quad\quad$ $t = ab\boxed{b}$ → if we don't find a char of t inside mapping of s, we don't have anagrams.

$\downarrow$ $\quad\quad\quad\quad$ $\uparrow_i$

$\{a : \cancel{2}\ 1$

~~b : 1 0~~

3

Space Complexity → $O(1)$

Time complexity → $O(N+N) \Rightarrow O(N)$

```
if ( S.length != t.length)
        return false;
```

2

strs ⇒ [eat , tea , tan, ate, nat , bat] p1 {eat , tea}

club anagrams together → anagrams are permutation of each other.

anagrams have same set of permutations

→ length is always same

→ char set with freq is same.

eat → eat , eta, tea, tae, ate, aet ←

tea → eat , eta, tea, tae, ate, aet

→ this permutation is special

why ?? because we have chars in sorted inc order.

2   curr → bat

[eat, tea, tan, ate, nat, bat],

↓   sort

abt

{
"aet" : [eat, tea, ate]
"ant" : [tan, nat]
"abt" : [bat]
}

↳ return all the values of the mapping by storing in an array.

unique sorted permutation → key - value

↳ set of anagrams

we can find more strings like eat whose sorted permutation will be aet.

Space →

→ In the worst case, we might have all N string different.

Every string will from a new key value pair.

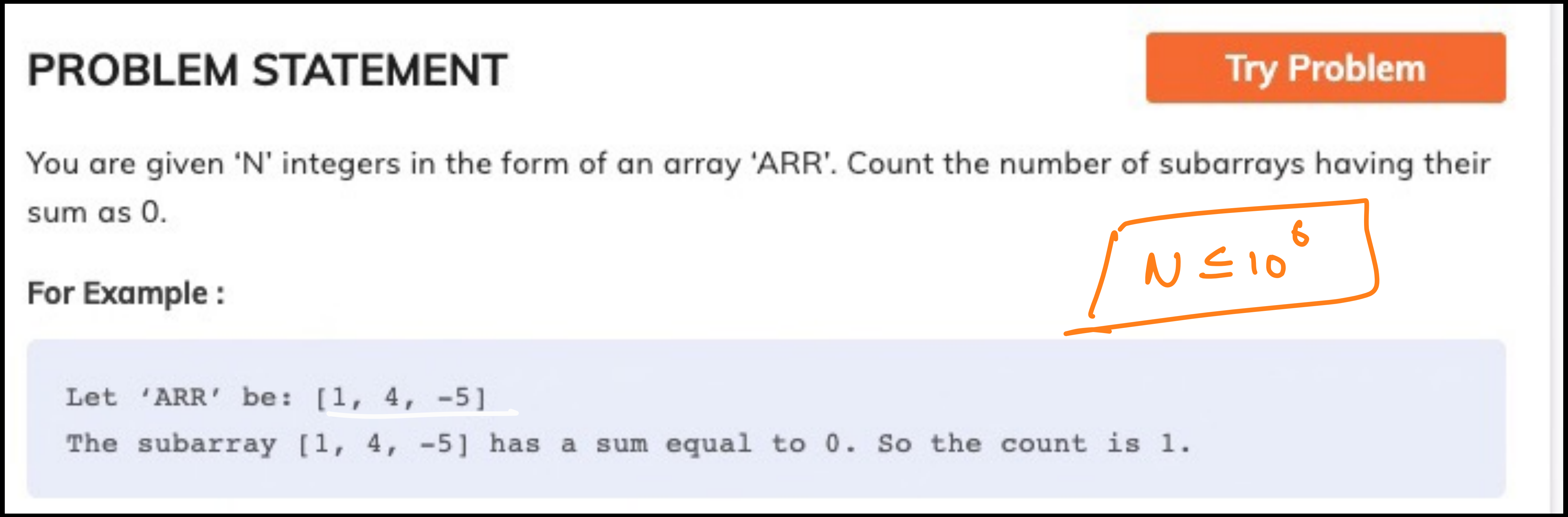
↳ if we assume max length of a string to be K ($K < 10^2$)

↳ $O(NK)$ ← space

$O(N K \log K)$ ← time → $10^4 \times 10^2 \log 10^2$

↳ $10^6 \times 7$

Subarray With Sum 0.



PROBLEM STATEMENT

You are given 'N' integers in the form of an array 'ARR'. Count the number of subarrays having their sum as 0.

For Example :

```
Let 'ARR' be: [1, 4, -5]
The subarray [1, 4, -5] has a sum equal to 0. So the count is 1.
```

$N \le 10^6$

[1, 4, -5] → [1] [1, 4]

[4] [4, -5]

[-5] [1, 4, -5]

Subarray → So subarray is a contiguous cross-section of the given array.

Sum = ~~0~~ ~~1~~ ~~3~~ ~~0~~ ~~4~~ 9

$\overset{i}{[1},\ 2,\ -3,\ 4,\ \overset{j}{5]}$

So, [2, 4] is not a subarray because they are not present consecutively

$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{4}$ $\boxed{5}$

[1, 2] [2, 3] [3, 4] [4, 5]

[1, 2, 3] [2, 3, 4] [3, 4, 5]

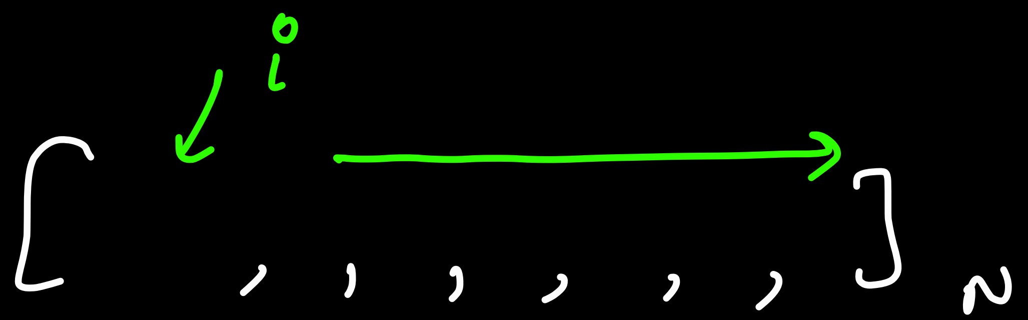
[1, 2, 3, 4] [2, 3, 4, 5]

[1, 2, 3, 4, 5]

nested loop

Brute force

→ We can generate all possible subarrays and then calculate their sum & then check if sum is 0.



every subarray is a contiguous cross-section, so it will be have a start and end.

if we wish to generate all possible subarrays, we can try to form all possible pairs of (start and end.)

```
for( i=0; i<n ; i++) {
    sum=0
    for( j=i ; j<n ; j++) {
        sum += a[j];
        if( sum == 0)
            count ++
    }
}
```

movement of j, helps us to get a new subarray so we add the elemt to get the sum.

Time ?
→ $O(n^2)$

Space ?
$O(1)$

TLE

↳ Given an array of length N, check if there is any subarray with sum 0. Return true if there is even 1 sub array with sum 0 else return false

0 ⟶ i

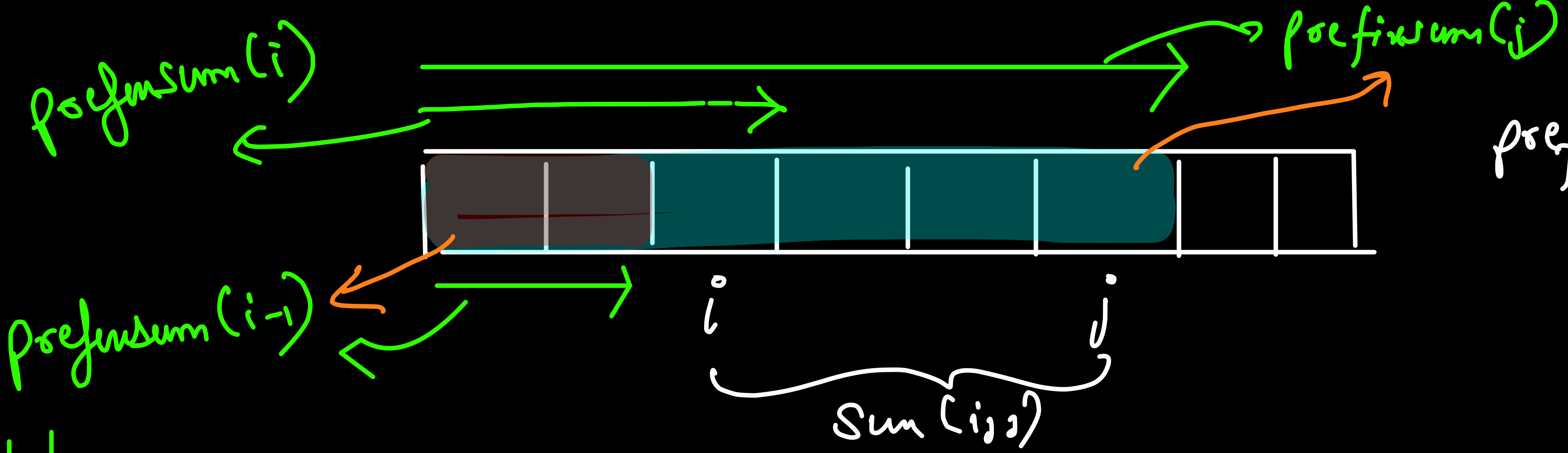
[1, 2, -3, 4, 5] ⟶ true

[1, 2, 3, 4, 5] ⟶ false

{1, 2, -2, 5] ⟶ true

i i ⟶

$N \le 10^6$



$$prefixsum(i) = \sum_{k=0}^{i} arr[k]$$

Technique

property about Subarray Sum.

$$\underbrace{sum(i, j)}_{\downarrow} = prefixsum(j) - \underset{\downarrow}{prefixsum(i-1)}$$

$$= prefixsum(j) - prefixsum(i) + arr[i]$$

Sum of subarray starting with index i and ending at j

(Sum of a range can be calculated by prefix Sums) create freq map & check if any element is present more than once.

arr

2	6	-3	1	-3	2	5

{ 2 : 1
8 : 1
5 : 2
6 : 1
3 : 1
10 : 1 }

prefixsum

prefix sum

		(i-1)	i		j	
2	8	5	6	3	5	10
		x			y	

just check if there are 2 indexes with same value or not.

key-value

freqmap

$$\text{Sum}(i, j) = \text{prefixsum}(j) - \text{prefixsum}(i-1)$$

↳ we are looking for sum 0.

$$0 = \text{prefixsum}(j) - \text{prefixsum}(i-1)$$

$$\Rightarrow \text{prefixsum}(i-1) = \text{prefixsum}(j)$$

$O(N+N+N)$

Time → $O(N)$

Space - $O(N)$

arr ⇒

2	5	-7	3	2

↓

prefixsum

2	7	0	3	5

prefixsum(x) == 0 ⟵ sum(0, x)

↓

subarray starting from 0 ending at x

final approach

→ either 2 elements are same or one element is atleast 0.

i = ~~1~~ ~~2~~ ~~3~~ 4

arr →

0	1	2	3	4
1	2	3	-5	6

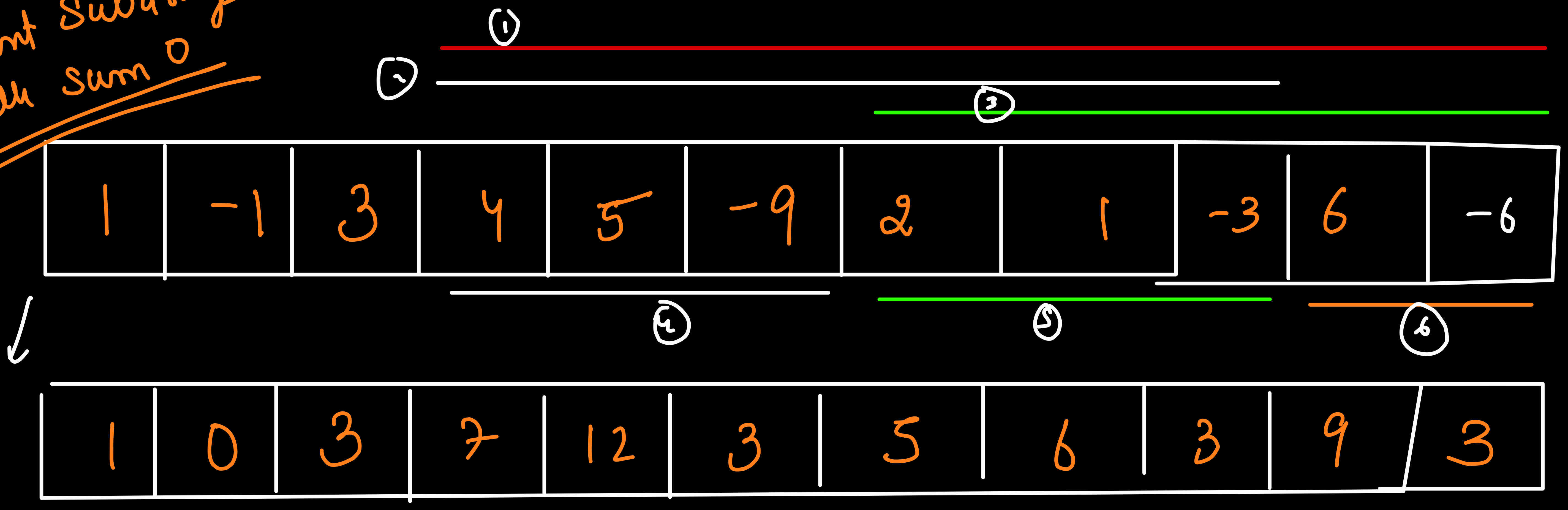
prefixsum →

1	3	6	1	7

```
prefixsum[0] = arr[0];
for(i=1; i<n; i++)
    prefixsum[i] = prefixsum[i-1] + arr[i];
```

→ O(n)

count Subarrays with sum 0



1	-1	3	4	5	-9	2	1	-3	6	-6

efixsum

1	0	3	7	12	3	5	6	3	9	3

{

1	1
0	1
3	4
7	1
12	1
5	1
6	1
9	1

}

① Total no. of 0's in the map tells about count of prefix subarray with sum 0.

Among these 4 occ of 3 we can choose any 2, to form a [(i-1), j] pair

→ $^{4}C_{2} \rightarrow \frac{4!}{2!2!} \rightarrow \frac{4\times3}{2} \rightarrow 6$

$^{x}C_2 \to \frac{x\times(x-1)}{2}$

arr

3	1	-1	2	3	-3	7	-9	2	-1	1

prefix sum 0 3 4 3 5 8 5 12 3 5 4 5

{
3 : 3 ✓
5 : 4
4 : 2
8 : 1
12 : 1
}

$^{3}C_2 + {^{4}C_2} + {^{2}C_2} \to \frac{3\times2}{2} + \frac{\overset{2}{\cancel{4}}\times3}{\cancel{2}} + \frac{2\times1}{2}$

$\overset{2}{\frac{3!}{2!\,1!}} + \frac{4!}{2!\,2!}$

$3 + 6 \to 9 + 1 \to 10$

Sum += $\frac{v\times(v-1)}{2}$

$\Sigma \frac{v \times (v-1)}{2}$ $\quad \forall\ v \in (k,v) \rightarrow \underline{\underline{\boxed{v>1}}}$

Time → $O(n)$

Space → $O(n)$