Anagram

↳ HEART EARTH

EARTH

(both are permutation of each other)

Ex → abaac abacc

LISTEN SILENT

S and t

rat car

cases 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 occurance

<k,v>

→ frequency map

Note → Order doesn't matter

Ex S → ANAGRAM (i)

t → NAGAARAM (i)

{

'A' : 0,

'N' : 0,

'G' : 0,

'R' : 0

'M' : 0

}

(i) → insert delete seen → O(1)

Key, value

char → freq

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

freq.

Once freq becomes 0, remove the char from mappy.

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

s = a a b

t = a b b → if we don't find a char of t inside mapping of s, we don't have anagrams.

↑ i

{ a : ~~2~~ 1

~~b : 0~~

3

Space Complexity → O(1)

Time Complexity → O(N + N) ⇒ O(N)

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

2

Ips ⇒ [eat, tea, tan, ate, nat, bat] ⇒ Op ⇒ (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.

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

2

curr → bat

2

sort

abt

{

"aet": [eat, tea, ate]

"ant": [tan, nat]

"abt": [bat]

}

unique sorted permutation

key - value

↳ set of anagrams

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

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

Span →

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

Every string will form a new key value pair.

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

↳ $O(NK)$ ← Space

$O(NK \log K)$ ← Time → $10^4 \times 10^2 \log 10^2$

↳ $10^6 \times 7$

Subarray With Sum 0.

PROBLEM STATEMENT

Try Problem

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

$N \leq 10^6$

For Example :

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

[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

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

i

2d

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

1 2 3 4 5

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

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

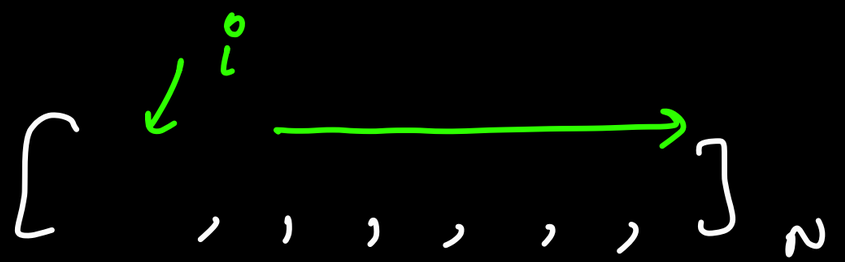
[1,2,3,4] [2,3,4,5]

[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 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 ++
    }
}
```

Time

$\rightarrow O(n^2)$

Space

$O(1)$

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

TLE

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

0 ⟶ i

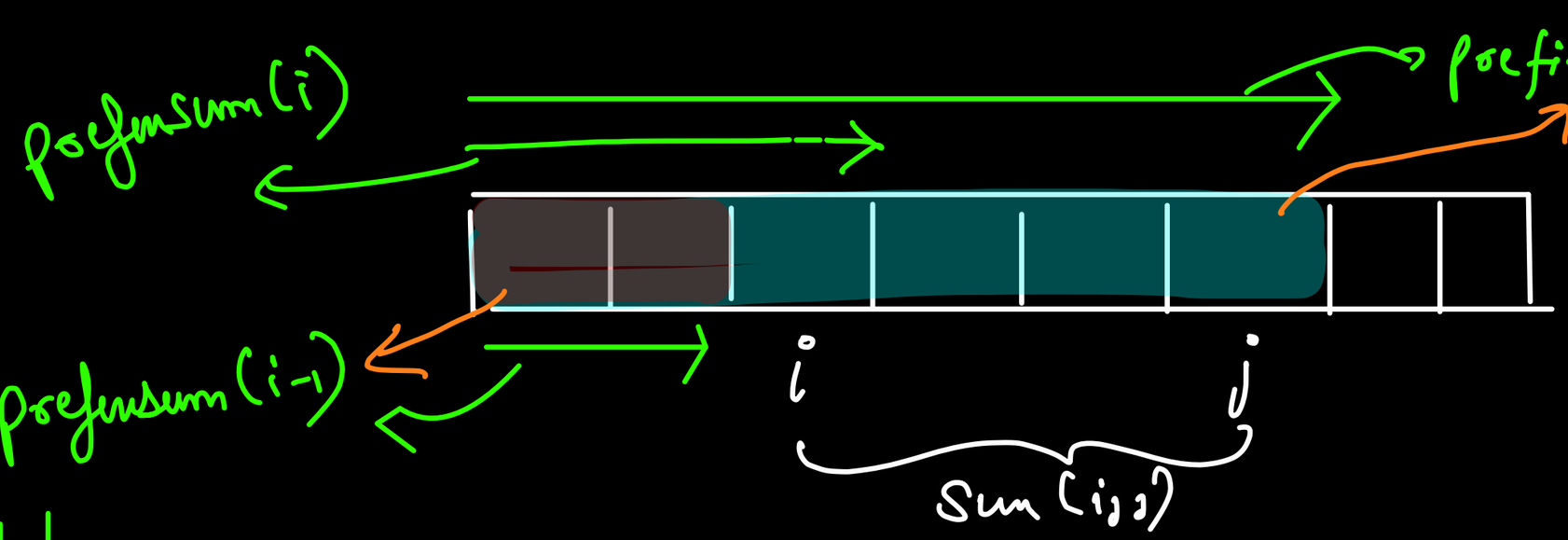
[1, 2, -3, 4, 5] ⟶ true

[1, 2, 3, 4, 5] → false

[1, 2, -2, 5] → true

i ⟶ j

$N \leq 10^6$



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

Technique

Property about Subarray Sum.

$Sum(i, j) = prefixsum(j) - prefixsum(i-1)$

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

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 elem 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 }

Prefix Sum

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

freq map

$$Sum(i,j) = prefixsum(j) - prefixsum(i-1)$$

↳ we are looking for Sum 0.

$$0 = prefixsum(j) - prefixsum(i-1)$$

$$\Rightarrow prefixsum(i-1) = prefixsum(j)$$

$O(N+N+N)$

arr ⇒

2	5	-7	3	2

↓

prefixsum

2	7	0	3	5

Time → $O(N)$

Space - $O(N)$

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.

$i = x \not< \not> y$

0	1	2	3	4
1	2	3	-5	6

arr →

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

1	0	3	7	12	3	5	6	3	9	3

prefixsum

```
{
    1%1      5:1
    0%1      6:1
    3:4      9:1
    7:1
    12:1
}
```

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

$\longrightarrow {}^4C_2 \rightarrow \frac{4!}{2!2!} \rightarrow \frac{4\times3}{2} \rightarrow 6$

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

arr

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

prefix sum

3 4 3 5 8 5 12 3 5 4 5

{
3 : 3
5 : 4
4 : 2
8 : 1
12 : 1
}

$3C_2 + 4C_2 + 2C_2 \rightarrow \frac{3 \times 2}{2} + \frac{4 \times 3}{2} + \frac{2 \times 1}{2}$

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

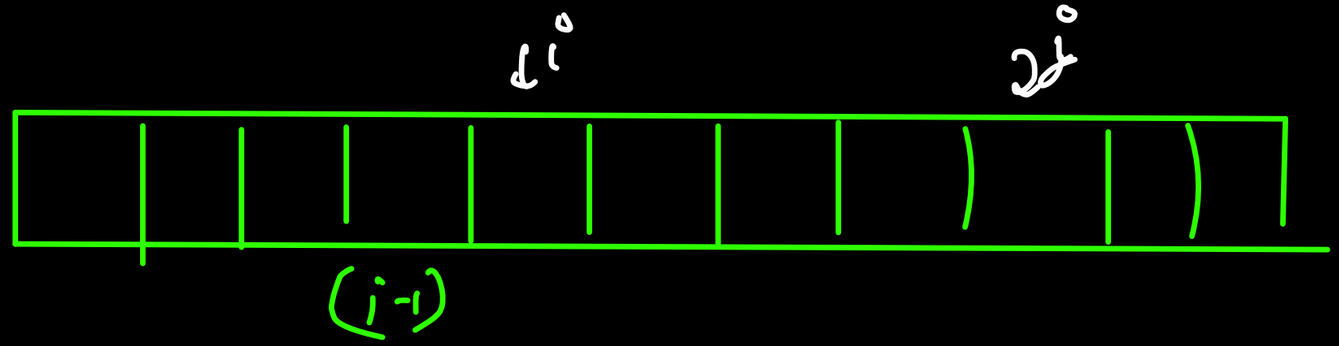
$3 + 6 \rightarrow 9 + 1 \rightarrow 10$

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

$\sum \frac{v \times (v-1)}{2}$ $\forall$ $v \in (K, v) \rightarrow (v \geq 1)$

Time $\rightarrow O(n)$

Space $\rightarrow O(n)$



2 prefixsum

Sum (i, j) = prefixsum [j] − prefixsum [i-1]

Sum of subarray from i to j

if we can find two index (i-1) and j that follows this eqn then we can say Sum (i,j) == K

Sum (i, j) = K

prefixsum [j] − prefixsum [i-1] = K

prefixsum [i-1] = prefixsum [j] − K

K = 3 arr → [1, 6, 2, -5, 4, 3, -7, 7, -4]

i

ans = 3 / 1 / 2

j

prefix → [1 7 9 4 8 11 4 11, 7]

Prefix map - 2

proyensum → ~~0~~ 7 9

(i-1) ↔ (j)

prefix(i-1) == prefix(j) - K

(i-1) < j

mp → {

1 : 1

7 : 1

9 : 1

⋮

O(n)

O(n)

}

if (mp[arr[j] - K]) {

ans += mp[arr[j] - K];

}

[1, 1, 1]

k = 2

↳ [1, 2, 3]

prefixSum == k

↳ There might be some players who never loose i-e always win.

↳ there might be some players who never player

↳ there might be some player who loose multiple time

State of the players

- unplayed
- always winning
- who lost once
- who loose more than once

[[1,6] [3,5]] : [x, y] [2, 3]

win lost

[4, 5]

[4, 2]

[6, 2]

0	1	2	3	4	5	6	7	8	9	10
-1	-1	2	1	0	1	0	-1	-1	-1	-1

O(N) O(max(win, lost)) ← 10^5

a[i] == -1 no match played by player i.

a[i] == 0 player is winning always upto now.

a[i] == 1 player lost atmost one match

a[i] > 1 player lost more than one match.

```
if ( a[x] == -1 or a[x] == 0)
    a[x] = 0
if(a[y] == -1)
    a[y] = 0;
else if(a[y] >= 0)
a[y]++;
```

i
2
[1, -3, 4, -6, 7, 8, 9]

O(n)

result = [-3, -6, 1, 4]

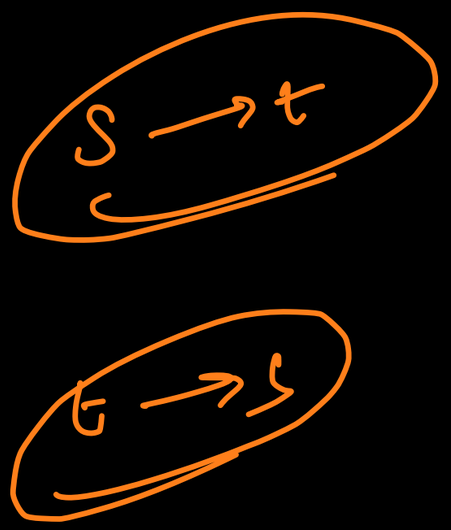
O(n) time
O(1) space

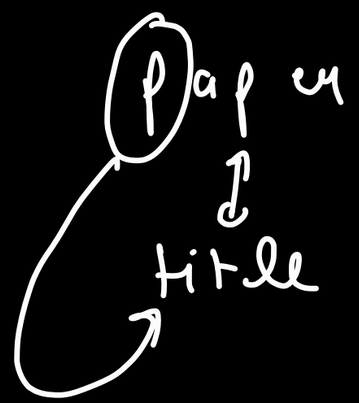
[-3, -6, -9, 1, 7, 8, 4]
j

i

less than i are negative

2 pointers






```
( p : t  a : i , t : l ,  m : e )

( t : p  i : a , l : e , e : x )
```

```
papers
tides
```

S → x y z a

t → x y x y

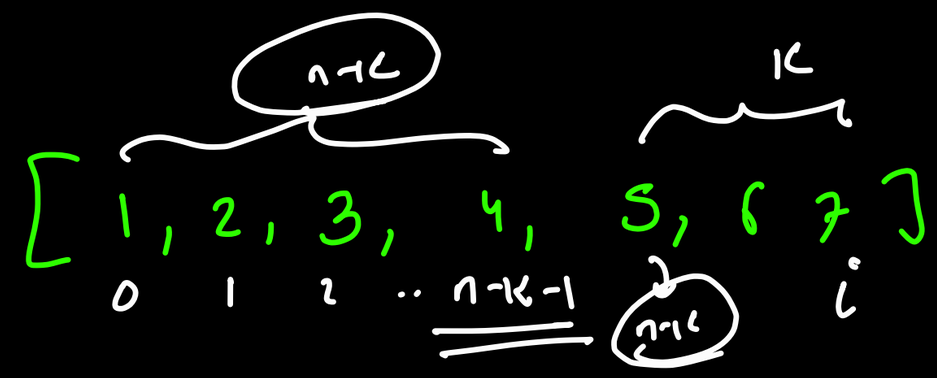
{ x : x , y : y , : : }

{ x : x , y : y }

O(N)

O(1)

space



k = 3

Ans ⇒ [5, 6, 7, 1, 2, 3, 4]

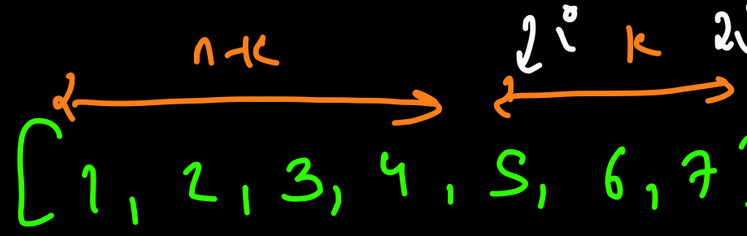
Approach 1 → Another array → O(n)

```
for (i = n-k; i < n; i++)
for (i = 0; i < n-k; i++)
```

[5, 6, 7, 1, 2, 3, 4]

Time → O(n)

Space → O(n)



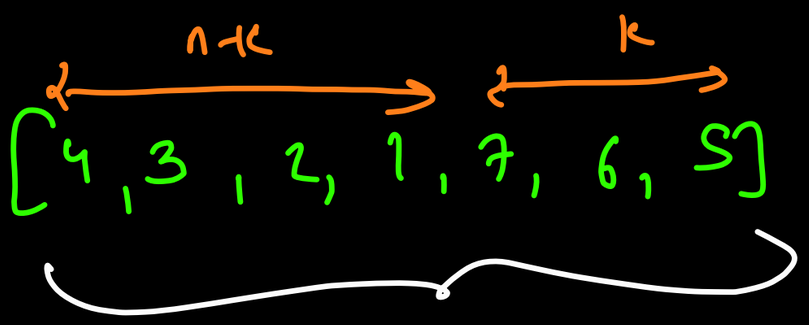
[4, 3, 2, 1 | 7, 6, 5]

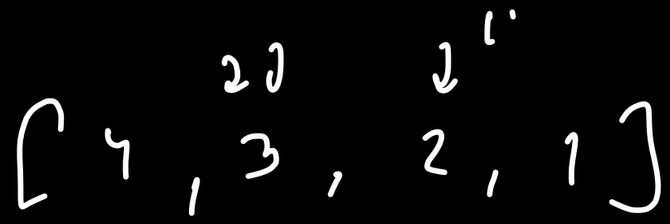
Ans → [5, 6, 7, 1, 2, 3, 4]

Reverse Anarray

[1, 2, 3, 4]

[4, 3, 2, 1]





$O(n-k + k + n)$

$\Rightarrow O(2n) \approx O(n)$

Space → $O(1)$