Agenda!

1. First Non Repeating Element
2. Pair Sum
3. Count of Pair Sum
4. check if there exists a subarray with Sum = 0
5. check if there exists a subarray with Sum = K.

1. First Non-Repeating Element

O(N) O(N)

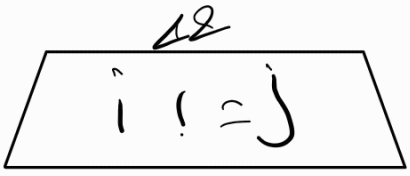
1. Cout the element frequencies in freq map
2. iterate over A again and check the element freqency if it is == 1

2.

Given int[] A and int k

check if there exist a pair (i, j)

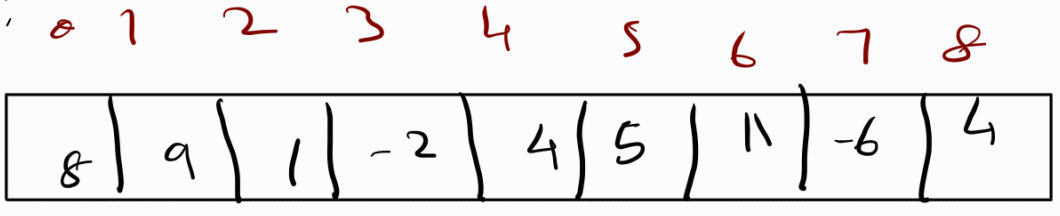
such that $\boxed{arr[i] + arr[j] = k}$



Brute force : $O(N^2)$

1. lets create all possible pairs,
   using nested loop one for i one for
   j from i+1 to N. $O(N^2)$
2. and check if element exist or not

optimized :



k=6 ✓
k=22 ✗
k=8 ✓

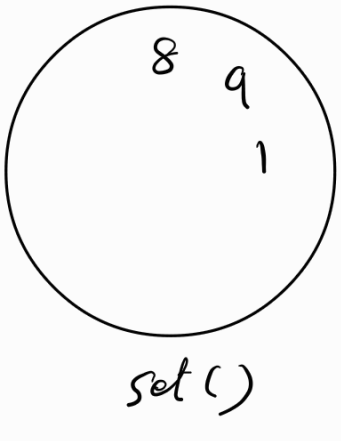
6-8
-2

6-9 6-1
-3 5

6+2
8

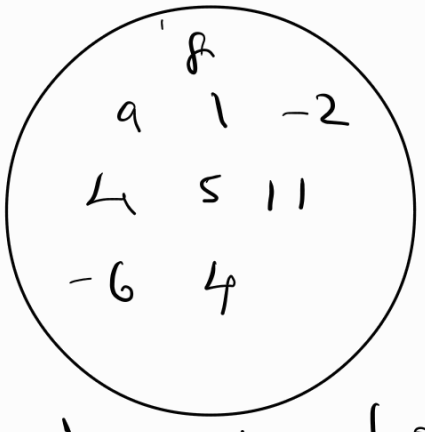
b = k - a

Return True



& $b = k - a \Rightarrow -2$

$\Rightarrow$ do we have it in set?



Create set out of A

1. # create st from A
2. # iterate over A and check if b is found in set.

TC: O(N)

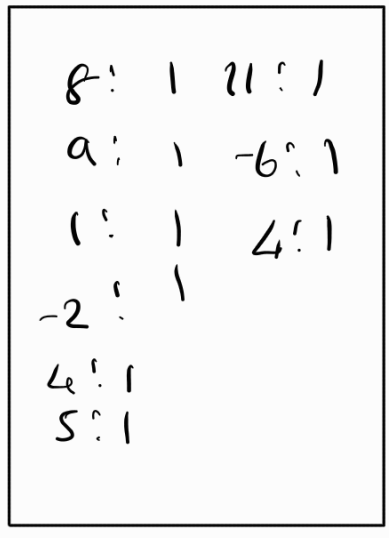
SC: O(N)

issue : above approach will not work $\boxed{i != j}$ condition

1. use dictonary to store element and frequency. and keep $i != j$

k = 18

18 - 9 = 9



we sould keep check

if freMap[9] == A[i]

- if value == 1
  - Pass
- else:
  - Consider

TC: O(N)

SC: O(N)

Count the number of pairs with sum = K

0	1	2	3	4	5	6	7
2	5	2	5	8	5	2	8
2	2	2	1	1	0	1	0

$k = 10$

8

2	5	2	5	8	5	2	8
8	5	8	5	2	5	8	2

$10 - 2 = 8$ ✓ 5

$a + b = k$

$b = k - a$

2	3
5	4
8	1

ans = 2 + 3 + 2 + 3 + 3 + 3 + 2 + 3

5 5 6 5

10 11

21

all the possible pairs = First create dictionary iterat A and all all frequency.

$a + b = k$

$a + b - a = k - a$

$b = k - a$

---

from left to right = iterate A and create dictionary. on the fly and add if it finds.

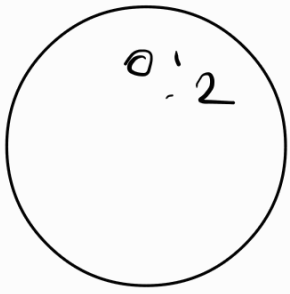
freqMap = { }, ans = 0, k = 6

for i in range(N)

b = k - A[i]

value = freqmap.get(b, 0)

ans += value



if A[i] in freqMap.

↓ ans += value

increase freqeny also.

else:

freqmap[A[i]] = 1

[0 0 0]

ans = 5

Subarray with Sum Zero

1. Given int [] A
2. Find if subarray with sum zero. return True

Approach 1 : BF $O(N^3)$

1. going through all the subarray and $O(N^2)$
2. check if it's sum = 0 $O(N)$

$A = [2\ \ 2\ \ 1\ \ -3\ \ 4\ \ 3\ \ 1\ \ -2\ \ -3\ \ 2]$ $N = 11$

(indices: 0 1 2 3 4 5 6 7 8 9)

2
2

2 2
2 2 1
2 2 1 -3
:

2 1 -3

$\frac{N(N+1)}{2} = 66$

```
N = len(A)
for i in range(N)
    for j in range(i, N)
        temp = 0
        for k in range(i, j+1)
            temp += A[k]
        if temp == 0:
            return True
return False.
```

approch 2 Prefix Sum $O(N^2)$, $O(N)$ space

1. creating Prfix Sum
2. for every Sub array get sum from prefix Sum

PrefixSum = [ ], prefixsum.append(A[0])
for i in range(1, N): prefixSum.apend (prefixSum[i-1] + A[i])

```
for i in range(N)
    for j in range( i, N)
        temp = 0
        if i == 0: temp = prefixSum[j]
        else:
            temp = prefixsum[ j] - prfixSum[i-1]
        if temp == 0: return True
return False
```

approch 3: carry forword $O(N^2)$

```
for i in range(N)
    temp = 0
    for j in range( i, N)
        temp += A[j]
        if temp == 0
            return True
```

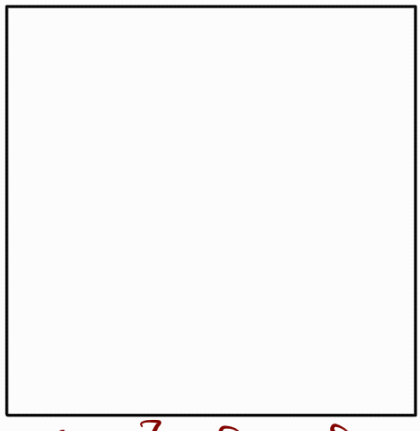
approch 4:

Sum(i, j) = 0

	0	1	2	3	4	5	6	7	8	9
A = [	2	2	1	-3	4	3	1	-2	-3	2] N = 11

i

j



	0	1	2	3	4	5	6	7	8	9
PrefixSum = [	2	4	5	2	6	9	10	8	5	7]

i j

= Prefix[j] - Prefix[i-1]

= 6 - 2

=> 4

idea:

Sum(i, j) = 0

Prefix[j] - Prefix[i-1] = 0

Prefix[j] = Prefix[i-1]

So, if Prefix[j] and Prefix[i-1] are same. they will make "Zero"

get frequency of all element in prefixsum

2 : 2 = True

4 : 1

5 : 2

6 : 1

9 : 1

10 : 1

8 : 1

7 : 1

A = [-2 -1 3 5]

PF = [-2 -3 0 5]

```
if value = 0 :
        return True
```

-2 : 1

-3 : 1

0 : 1

5 : 1

Subarray with sum → k

⇒ Given int [] A, check if there is a subarray with sum `k'

$1 \le N \le 10^8$

arr = [2 , 3, 9 , -4 , 1, 5, 6 , 2, 5]

k = 11

k = 10

sum (i, j) = k

PrfixSum[j] − prfixsum[i-1] = k

PF = [2 5 14 10 11 16 22 24 29]

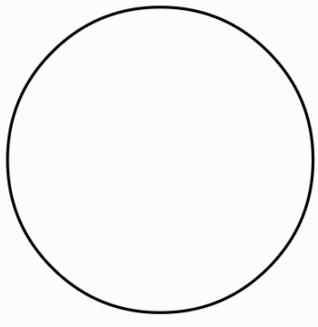
arr = [5 10 20 100 105]

PF = [5 15 35 135 240]

k = 10

PF [j] - PF[i-1] = k

15 - 5 = 10



Approch 1 : BF $O(N^3)$

1. going through all subarray and check if
2. sum is == k.

Approch 2 :

$\Rightarrow a - b = k$

add b both sides

$\Rightarrow a - b + b = k + b$

simplify it.

$\Rightarrow a = k + b$

subtract k from both sides

$\Rightarrow a - k = k + b - k$

Simplify

$a - k = b$

$b = a - k$

$Sum(i, j) = k$

$PrfixSum[j] - prfixsum[i-1] = k$

(a) - (b) = k

as per derived equation:

$Prefix[i-1] = Prefix[j] - k$

$b = a - k$

$Prefix[i-1] = Prefix[j] - k$

Dry run!

arr = [2 3 9 -4 1 5 6 2 5] (j at 5)

Subarray = 15

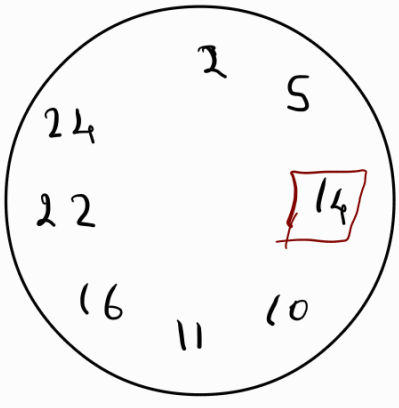
k = 15

PF = [2 5 14 10 11 16 22 24 29]

29 - 15 = 14

2 - 15 = -13

PF = [2 5 14 10 11 16 22 24 14]



arr = {2 3 9 -4 1}

PF = [2 5 14 10 11] k = 10

8 5 4 0 1

if PF[j] == k

return True

one way

2nd way

add 0 beforehand

Code :

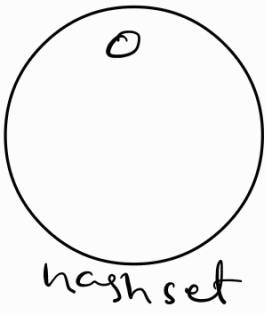
```
# creatR prefixsum O(N)
Pf = [ ]
Pf.append( A[0])
for i in range( 1, N)
    ↓ Pf.append( Pf[i-1] + A[i])
# iterate through the Pf array and O(N)
find if sum = 0 exists
hashset = 0,
for j in rang(N)
    # b = a - K formula
    b = Pf[j] - K
    if b in hashset :
        ↓ return True
    else:
        ↓ hashset.add( PF[j])
```

TC : O(N)

SC : O(N)

without using PF array

```
sum = 0
hashset = {}
for j in range(N)
    sum += A[j]
    b = sum - k
    if b in hashset:
        return True
    else:
        hashset.add(sum)
return False
```



TC: O(N)

SC: O(N)

① check pair sum.

Given int[] B, int A as sum.

=> check if there exists a pair (i, j) such that

$B_i + B_j = A$ and $i \neq j$

$a + b = k$

$b = k - a$ (a = B[i])

B[i]

A = 8

B = [8 5 1 2 1 2]

8 8
0

$1 \leq$ length of $B \leq 10^5$

$0 \leq B_i \leq 10^9$

$0 \leq A \leq 10^9$

code:

```
N = len(B), eleSet = set()
for i in range(N)
    b = A - B[i]
    if b in eleSet:
        return 1
    else
        eleSet.add(B[i])
return 0
```

② Count pair Difference

1. given int [] A , int B
2. Count no. of pairs (i,j) such that

$$A[i] - A[j] = B \quad \& \quad (i \neq j)$$

B = 4

	1	2	3	4
A	3	5	1	2

3 - 4 / -1 ; 5 - 4 / 1

$1 \leq N \leq 10^5$

$1 \leq A[i] \leq 10^9$

$1 \leq B \leq 10^9$

$A[i] - A[j] = B$

$a - b = k$

	1	2	3	4
A =	1	2	1	2

B = 1

(1+1)/2 ; (2+1)/3 ; (1+1)/2

1 ; 2

2 ; 1

$a - b = k$ # Find a

$a - b + b = k + b$

$a = k + b$

ans = + 2

$A[i] = k + A[j]$

$A = [2 \quad 1 \quad 21 \quad 21 \quad 21 \quad 2] \qquad B = 1$

$\frac{2+1}{3} \quad \frac{1+1}{2} \quad \frac{2+1}{3} \quad \frac{1+1}{2} \quad \cdots \quad \frac{1+1}{2} \quad \frac{1+1}{2}$

2 : 5

1 : 4

$A[i] = A[j] + K$

Code:

```
def pairSumB(A, B):
    N = len(A)
    ans = 0, freqmap = { }
    for i in rang(N):
        freqmap[A[i]] = freqmap.get(A[i], 0)+1

    for i in range(N)
        b = A[i]+B
        ansf= freqmap.get(b, 0)

    return ans.
```

Sub array with sum zero

1. given int[] A,
2. return array contains subarray with sum zero

$1 \leq |A| \leq 1000$

$A = [1, 2, 3, 4, 5]$ ans=0

$Pf = [1 \quad 3 \quad 5 \quad 9 \quad 14]$

sum(i,j)=0

Pf[j] - P[i-1] = 0

a - b = 0

a - b + b = b

a = b

create Pf array (optional)

Pf = []

iterate through array and check if condition is matching.

Handling zero

while creating set add 0 before the hand

④. Subarray sum = k

1. given int[] A, int B
2. Find Num subarray sum equal to B

PF = [1 , 1 , 0]
1-1 = 0 , 1-1 = 0 , 0-1 = -1

Sum(i, j) = k

PF[j] - PF[i-1] = k

a - b = k # find b

a - b + b = k + b

a = k + b

a - k = k + b - k

b = a - k

PF[i-1] = PF[j] - k

[0 0 0] | [1 0 1]

Count = 0 + 3 + 3 = 6

currentSum = 0

prefixMap = { 0 : 3 }

b = a - k

0

Count = 0 + 2 + 4

sum = 0 + 1 + 2

b = a - k

2 - 1 = 1

0 : 1
1 : 2

steps: iterat over array, add the element to prefix sum.

```
ans = 0
prefixSum = 0
PfMap = {0: 1}
```

↓

because there might be subarray sum that is equal to k from the start.

```
for a in A:
    prefixSum += a
    # b = a - k
    b = prefixSum - B
    # check if b present in PfMap.
    if b in PfMap:
        anst = PfMap[b]
    #add the PfixSum into PfMap:
    Value = PfMap.get(PriffSum, 0) + 1
    PfMap[PfiSum] = value
return ans
```

Count pair Sum

1. Given int [] A, int B
2. Count the number of pairs (i,j) such that A[i] + A[j] = B and i != j
3. return ans % 10^9+7

A = [3 5 1 2] B = 8 ans = 1

8-3 = 5, 8-5 = 3, 8-1 = 7, 8-2 = 6

A[i] + A[j] = B

a + b = k

a + b - a = k - a

b = k - a

3 : 1
5 : 1
1 : 1
2 : 1

ans = 1

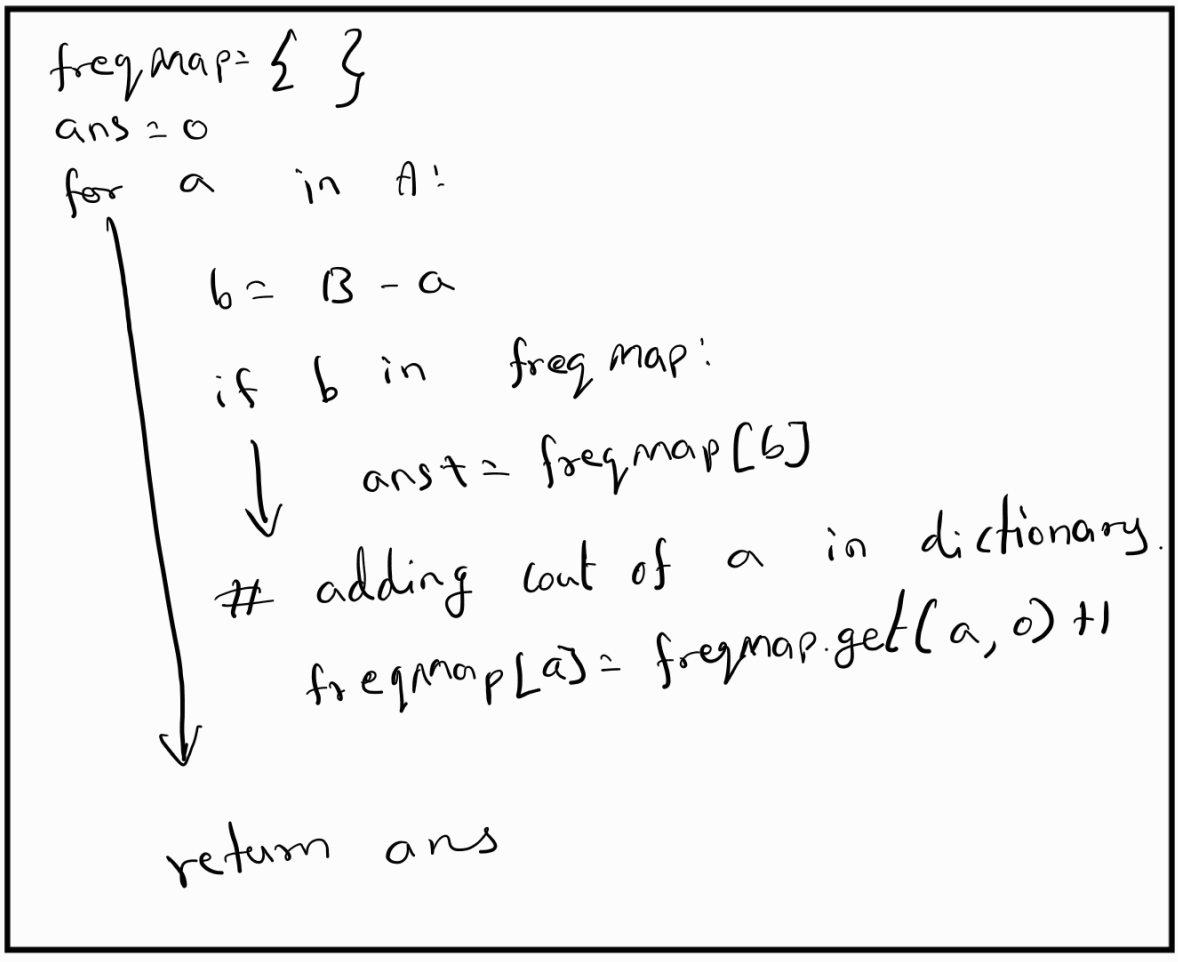
A = [1, 2, 1, 2] B = 3

3-1 = 2, 3-2 = 1, 3-1 = 2, 3-2 = 1

1 : x2
2 : 1

ans = x2 + 2 = 4

1. iterat over array
2. b = K - A[i]
3. if b in dictionary
   ans += value
4. dictionary[a] += 1


freqMap = { }
ans = 0
for a in A:
b = B - a
if b in freqmap:
ans += freqmap[b]
# adding count of a in dictionary.
freqmap[a] = freqmap.get(a, 0) + 1
return ans

Count Subarray zero Sum

1. Given int [] A,
2. Find count of Subarray sum = zero
3. return ans % (10^9 + 7)

ex: 1 $A = [1, -1, -2, 2]$

$PF = [1, 0, -2, 0]$

equations:

$$sum(i,j) = 0$$

$$Pf[j] - Pf[i-1] = 0$$

$$\boxed{Pf[j] = Pf[i-1]}$$

1 : 1
0 : 2
-2 : 1

=> if element repeating more than 1 they make zero.

ans = 1 + 2 = 3

ex: 2 $A = [-1, 2, -1]$

$Pf = [-1, 1, 0]$

0 : 1
-1 : 1
1 : 1

(x) :
keep (0:1) in the begining.

ans = 1

as there can be subarray = zero from starting.

Code:

```
freq_map = {0:1}
current_sum = 0
ans = 0
for a in A:
    current_sum += a
    if current_sum in freq_map:
        ans += freq_map[current_sum]
    freq_map[current_sum] = freq_map.get(current_sum, 0) + 1
return ans
```

Subarray with given Sum = K

1. Given int [] A, and int B
2. Find and return first continous subarray which adds to B
3. else return [-1]

```
      0  1  2  3  4
A = [ 1, 2, 3, 4, 5 ]      B = 5
      0  1  2  3  4
Pf = [ 1, 3, 6, 10, 15 ]
     1-5 3-5 6-5 9-5 14-5
     -4  -2   1   4   9
```

6 - 5 = 1

6 - 1 = 5

$sum(i,j) = k$

$Pf[j] - P[i-1] = k$

$P[j] = k + [P[i-1]]$

$P[j] - k = P[i-1]$

```
0 : 1
1 : 1
2 : 1
9 : 1
```

current_sum = 0

```
for i in range(N)
    current_sum += A[i]
    b = current_sum - k
    if b in freqmap:
        break.
```

```
0 : 1
1 : 0
2 : 1
3 : 2
```

Curr_Sum = 0 1 3

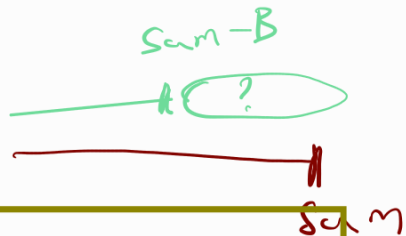
3 - 5 = -2

6 - 5 = 1

j = 2

i =

elem = 1



Pfelefreq = {0 : -1}, Curren_Sum = 0 A[i]

s = -1
e = -1

for i in range(len(A)) O(N)

Curr_Sum += A[i]

b = Curr_Sum - B

if b in Pfelefreq:

s = Pfelefreq[b] +1

e = i

break

Pfelefreq[Curr_Sum] = i

TC: O(N)

SC: O(N)

if s != -1

ans = []

for i in range(s, e+1)

ans.append(A[i])

return ans

else:

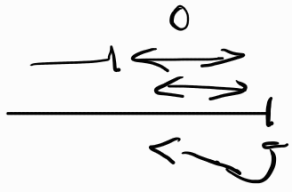
return [-1]

Longest Subarray zero Sum

1. Given int[] A
2. Find the length of longest subarray
3. if there is no subarray sum zero return 0.

$$A = [1, -2, 1, 2]$$

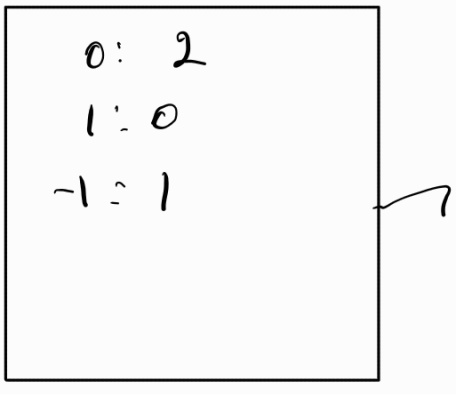
indices: 0, 1, 2, 3



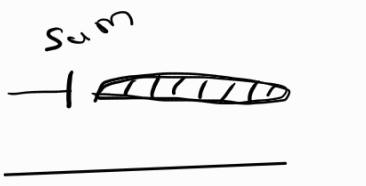
$$Sum = [1, -1, 0, 2]$$

$Pf[j] - P[i-1] = 0$

$Pf[j] = P[i-1]$



ans: 3




```
ans = 0
sum = 0
hm = {0:-1}  # if any subarray sum = k
s, e = -1, -1
for i in range(len(A)):
    sum += A[i]
    if sum in hm:
        s = hm[sum] + 1
        e = i
        length = e - s + 1
        if ans < length
            ans = length
    else: # update the sum occurance into hm if not seen ahead
        hm[sum] = i
return ans
```