Q1. Given N array elements, check if there exists a subarray with sum = 0.

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	2	2	1	-3	4	3	1	-2	-3	2

O/p ⇒ 2 + 1 + (-3) = 0 true

brute force

1) for every subarray find the sum of subarray

if (sum == 0)

return true

TC ⇒ O(N²) * O(N)

≈ O(N³)

↓ using Pfsum / carry forward

TC ⇒ O(N²)

SC ⇒ O(1)

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	2	2	1	-3	4	3	1	-2	-3	2
Pfsum ⇒	2	4	5	2	6	9	10	8	5	7

sum [a b] ⇒ Pf[b] - Pf[a-1] { a > 0

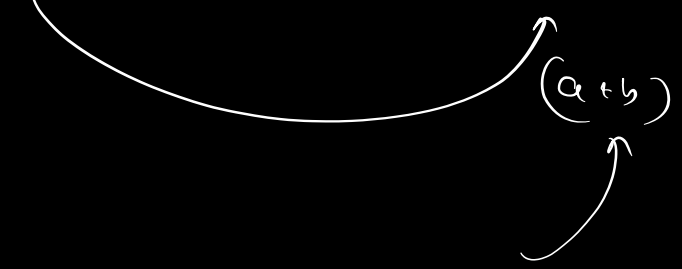
sum [3 1] ⇒ pfsum[3] - pfsum[0]

⇒ 0 = pfsum[3] - pfsum[0]

⇒ pfsum[0] = pfsum[3] ✓

arr: a b [c d] e
(c: 0, d: 0)

pf: a a+b a+b+c a+b+c+d a+b+c+d+e



if pfsum is ever getting repeated, or

if pfsum Arr has any duplicates, that means there exists a subarray whose sum = 0

* if pfsum array has any duplicate
  return true
  else
  return false.

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	2	2	1	-3	4	3	1	-2	-3	2
pfsum ⇒	2	4	5	2	6	9	10	8	5	7

Soln

1) Create pfsum array → TC O(N) + SC O(N)

2) iterate pfsum array → TC O(N) + SC O(N)
keep checking if pfsum[i] in hashset, if not found pfsum[i] in hs

TC ⇒ O(2N) ≈ O(N)
SC ⇒ O(2N) ≈ O(N)

optimize

O(N) + TC (single iteration)
O(N) + SC

	0	1	2	
arr ⇒	1	0	2	⇒ O/p ⇒ true
pfsum ⇒	1	1	3	

arr ⇒	-1	1	2	⇒ O/p ⇒ true
pfsum ⇒	-1	0	2	

pfsum ⇒

0	-1	0	2

Two sum prob

Q2. Given N array elements, check if there exists a pair (i, j), such that arr[i] + arr[j] == k, [i != j]

Soln — Check all pairs:

TC ⇒ O(N^2)
SC ⇒ O(1)

```
for(i=0; i<N; i++) { // i is fixed, arr[i] → a
    for(j=i+1; j<N; j++) {
        if(arr[i] + arr[j] == k) {
            return true
        }
    }
}
return false
```

Check if b is present or not

b ↓ a+b=k

arr[i] + arr[j] == k
⇒ a + b == k
⇒ b = k - a

use hashset

Soln
1) Create a hashset
2) populate the hashset
3) Check for every i [arr[i]] if we have [k - arr[i]] in hashset.

	0	1	2	3	4	5	6	7	8	9
arr[i] =	8	9	1	-2	4	5	11	-6	7	5

k = 11

1) hs & populate

hs [8 9 1 -2 4 5 11 -6 7]

k = 11

a	b (k-a)	present in hs
8	3	X
9	2	X
1	10	X
-2	13	X
4	7	✓

return true

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

k = 22

a	b (k-a)	present in hs
8	14	X
9	13	X
1	21	X
-2	24	X
4	18	X
5	17	X

1) Create hs & populate with arr

[8 9 1 -2 4 5 11 -6 7]

hs.

[11 11 ...] → return true { wrong approach }

⇒ Create a hashmap with freq.

i) create hashmap

ii) update the hm with freqency of each element

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

hashmap < int, int >
key → element
value → freq.

hm

8 : 1	-2 : 1	7 : 1
9 : 1	4 : 1	
1 : 1	11 : 1	
5 : 2	-6 : 1	

K = 22

a	(k-a) b	check in hm
8	14	X
9	13	X
1	21	X
-2	24	X
4	18	X
5	17	X
11	11	→ if (a == b) if (freq(a) > 1) return true

	0	1	2	3	4	5
eg ⇒	7	5	2	5	9	5

↑ (k-a)

k = 10

a	b	Check in hm
7	3	X
5	5	✓

freq(5) > 1

return true

hm

7 : 1

5 : 3

2 : 1

9 : 1

k = 9 — 7 — 2 — ✗

pseudo

TC → O(N) + O(N)

SC

1) Create a hm & populate with freq

```
for( i=0; i<N; i++ ) {
    a = arr[i]
    b = k-a
    if( a != b ) {
        if( b is present in hm )
            return true
    } else {
        if( freq(a) > 1 )
            return true
    }
}
return false;
```

O(N)

TC ⇒ $O(2N) \approx O(N)$

SC ⇒ $O(N)$

: Solving with hashset :

1) create a hashset

2) populate the hashset

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	8	9	1	-2	4	5	11	-6	7	5
							i			

hs [8 9 1 -2 4 5]

⇒ when iterating to populate hashset, if iterated till index i, we have all element in hashset till [0 i-1]

	0	1	2	3	4	5	6	7	8	9 ↓
arr[] =	8	9	1	-2	4	5	11	-6	7	5

k = 22

i	a	b = (k-a)	Check in HS	Current HS
0	8	14	X	[] insert 8
1	9	13	X	[8] insert 9
2	1	21	X	[8,9] insert 1
3	-2	24	X	[8,9,1] insert -2
4	4	18	X	[8,9,1,-2] insert 4
5	5	17	X	[8,9,1,-2,4] insert 5
6	11	11	X	[8,9,1,-2,4,5] insert 11
7	-6	28	X	[8,9,1,-2,4,5,11] i→6
8	7	15	X	[8,9,1,-2,4,5,11,-6] i→7
9	5	17	X	[8,9,1,-2,4,5,11,-6,7] i→5

return false

Pseudo

```
hashset<int> hs;
for(i=0; i<N; i++) {
    a = arr[i]
    b = k-a
    if( b is present hs)
        return true
    else {
        hs.insert(a)
    }
}
return false.
```

TC ⇒ O(N)
SC ⇒ O(N)

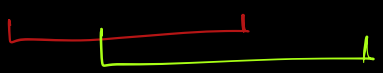
→ Given arr, find no. of pairs (i, j) such that arr[i] + arr[j] = 2k, (i ≠ j)

↓ Hashmap (✓)

Todo ⇒ Calculate if there exists a pair (i, j) such that arr[i] − arr[j] = 2k, (i != j)

Q 3. Given N array elements, Calculate no. of distinct elements in every window of size 'k'.

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



k = 4

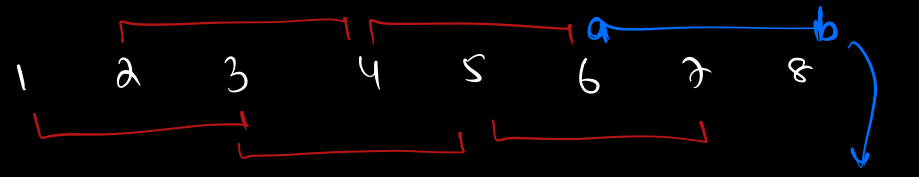
Subarray	# distinct
0-3	4
1-4	3
2-5	3
3-6	4
4-7	3
5-8	2
6-9	3

print

Brute force

for every window / subarray, insert elements into hashset, and find distinct

No. of subarrays of len ⇒ k



k	no. of sub	N = 8
1	8	
2	7 →	8-2+1
3	6 →	8-3+1
4	5 →	8-4+1
⋮	⋮	
K	N-k+1	

NOTE :- No. of subarrays of len k, in arr[N] ⇒ N-k+1

↓

len of sub ⇒ k

⟶ (N-k+1) subarrays

last subarray ⇒ [a b] ⇒ k

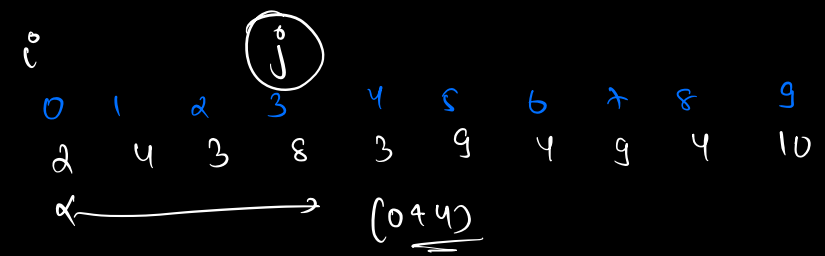
⇒ b-a+1 = k

⇒ (N-1) - a + 1 = k

⇒ N-1-a+1 = k

⇒ a = N-k

last subarray ⇒ [N-k, N-1]



k = 4

2 4 3 8 → 4

1 4 3 8 → 3

3 8 9 → 3

pseudo

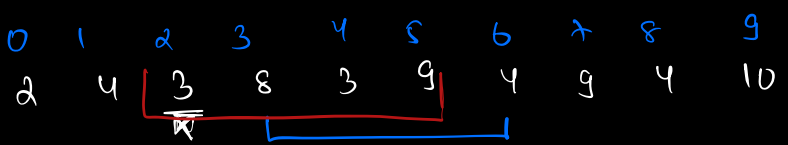
```
for( i=0; i<=(N-k); i++){
    HashSet<INT> hs;
    for( j=i; j<(i+k); j++) {
        hs.insert( arr[j])
    }
    print (hs.size())
}
```

TC ⇒ O((N-k+1) * k)

⇒ O(N²) ⇒ k = N/2

SC ⇒ O(k)

1. Sliding window with hashset



K = 4

subarray	remove	add	Hashset	#
[0-3]	–	–	{2, 4, 3, 8}	4
[1-4]	0th	4th	{4, 3, 8}	3.
[2-5]	1st	5th	{3, 8, 9}	3
[3-6]	2nd	6th	{4, 8, 9}	3

Sliding window with Hashmap.

	0	1	2	3	4	5	6	7	8	9
arr =	2	4	3	8	3	9	4	9	4	10.

K = 4

subarray	remove	add	HM	distinct
[0-3]	–	–	[2:1, 4:1, 3:1, 8:1]	4
[1-4]	0th (2)	4th (3)	[2:0, 4:1, 3:2, 8:1]	3.
[2-5]	1st (4)	5th (9)	[4:0, 3:2, 8:1, 9:1]	3.

[3-6]	2nd (3)	6th (4)	[3:1, 8:1, 9:1, 4:1]	4
[4-7]	3rd (8)	7th (9)	[3:1, 8:0, 9:2, 4:1]	3.
[5-8]	4th (3)	8th (4)	[3:0, 9:2, 4:2]	2

i) for new element,
insert (arr[i], 1)

ii) if freq = 0, remove (arr[i])

iii) if element exists,
increase freq by 1

TODO

pseudo

TC => O(N)
SC => O(N)