BReaK

↳ Slack/email for confirmation

↳ Wed / Friday => OFF to clear backlogs this weeks. :)

Hashing

Data Structures

- Hashmap <Key, Value>
- Hashset <Keys> {1, 3, 7, 8, 3}

Unique Keys

Q Given an array, check if there is a pair (i, j) such that

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

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

K = 10

1. Brute Force — i, j — $O(N^2)$

```
for(i)
  for(j)
    if(a[i]+a[j]==k)
```

2. Sort + 2 Pointers

0, 1, 2, 4, 5, 6, 7, 8, 10

s e

```
while(s<e){
  ==
  ==
  ==
}
```

1+8 = 9 s++

7+4 = 11 e--

(0,10)
(2,8)
(4,6)

// Sort NLog N

+ N

= O(NLog N)

```
s = 0
e = n-1

while ( s < e ) {
        if(a[s] + a[e] == k) { print (a[s], a[e]), s++, e-- }
        else if ( a[s] + a[e] > k ) {   e--          }
        else                {   s++                  }
}

return false ;      // No such pair exists.
```

return True for current problem

(3) arr = [8, 1, 2, 4, 5, 10, 0, 6, 7]

Find 5
↳ Linear search (N)
↳ Binary search
↓ Sort N log N +
↓ Search Log N

expensive

hashset or hashmap

Attempt - 1

(1) Put all elements in the hashset

hs
8
1
2
4
5
10
0
6
7

O(N + N)
= O(N)

① hashset<Integer> hs;

```
for(int x : arr){
    hs.put(x);
}
```

hashset offers O(1) search

② iterate over every element in the array,
s.t. x + y == k

x → a(i)
y → a(j)

```
for(i=0, i<=N-1; i++){
    x = arr[i]
    // Look for y = K-x in the hashset
```

x + y = K
⇒ y = K - x

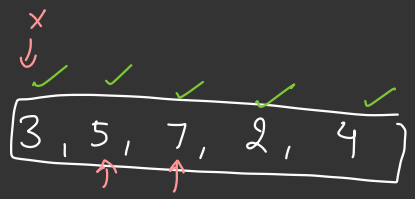
$8 + y = 10$

$\Rightarrow y = 10 - 8$

$= 2$

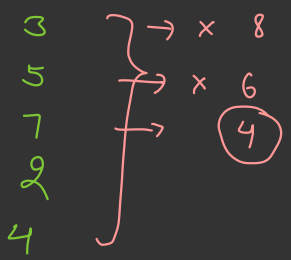
```
if( hs.contains(y) ){
    // got the pair x,y
    return true.
}
```

return false,



$K = 11$

hashset



$\rightarrow x + y = 11$

$3 + y = 11$

$\Rightarrow y = 11 - 3$

$= 8$

$7 + 4 = 11$

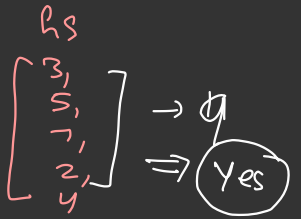
(7, 4)

Problem

$K = 14$

arr → 3, 5, 7, 2, 4

hs



But (7,7) is a valid pair

In

$i \neq j$

Current Solution using hashset fails.

Another Example 3, 5, 7, 2, 4

3, 5, 7, 2, 4, 7

frequency is 2

store unique keys

⇒ hashset

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

we actually need a hashmap here. ① Step.

```
hashmap <int, int> hm;
```

key (number)	value (freq)
3	1
5	1
7	2
2	1
4	1

```
for(i=0, i<n-1, i++) {
    if(hm.contains(a[i])) {
        hm.put(a[i], prevFreq+1);
    }
    else {
        hm.put(a[i], 1);
    }
}
```

repeating element

first time

3

3

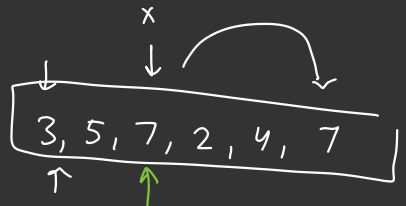
//

```
hm.put(a[i], hm.getOrDefault(a[i], 0) + 1)
```

key, default

0 + 1

prev + 1



~~key~~

(2) find a pair (

~~k=5~~

k=14

```
for( int x : arr ) {
    ⇒ y = k - x;
    → if ( x != y ) {
            if ( hm.contains(y) {
                return true;
            }
        else if( x == y ) {
            if ( hm.get(x) >= 2) { return True },
        }
    }
return false;
```

arr[i] → hashmap

freq

7 + 7 = 14

↓

freq > 1

C++

```
unordered_map <int, int > m,
```

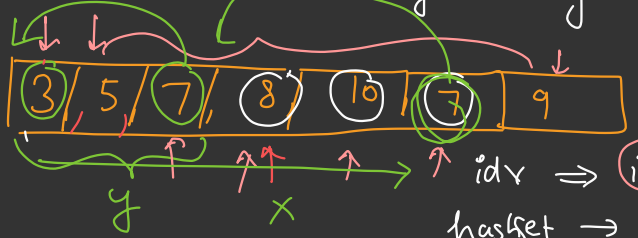
C++

m[arr[i]] = freq

hm.insert(key, value)

pair

// Pair Sum — one more try using Hashset



7 ↷ 2 times / 1 times. Problem

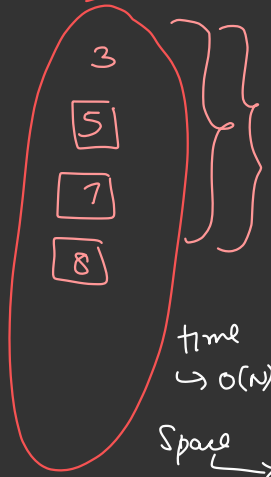
hashset → 0 — (i-1) | j < i | K = 14

↳ ensures two elements are diff

One Step algorithm

For every we try to combine it some element we have already seen

↳ HS



time ↳ O(N)

Space ↳ O(N) because of hashset

✓ hashset<int> hs;

for (i=0, i<=n-1; i++) { K=14

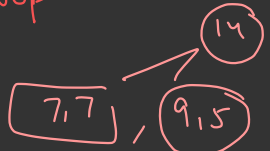
x = a[i]

y = k - x

→ if (hs.contains(y)) { return true; }

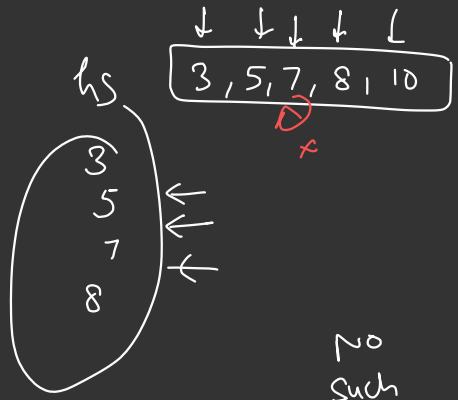
hs.put(x); // wop

}



DRY Run hs

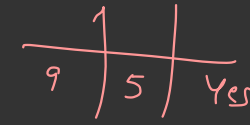
	y	
3	11	NO
5	9	NO
7	7	NO
8	6	NO
10	4	NO
→ 7	7	yes



K=14

3	11	NO
5	9	NO
7	7	NO
8	6	NO
10	4	NO

No such pair



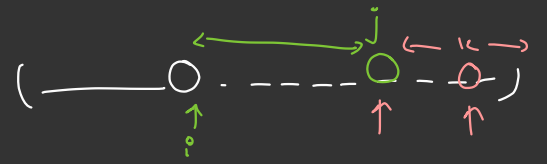
(9) Find if there is a triplet $a[i] + a[j] + a[k] == 9$ in a given array $i < j < k$

Input

1, (3), 4, 7, (6), 2, (8)

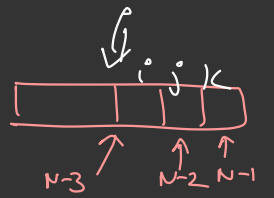
$S = 17$

a(i) a(j) a(k)



(1) Brute Force

```
for( i=0 , i<= N-3 , i++) {
    for( j= i+1 ; j<= N-2; j++){
        for( k=j+1 ; k<=N-1 , k++){
            if (a(i)+a(j)+a(k) == S){
                // got the pairs triplets
            }
        }
    }
}
```

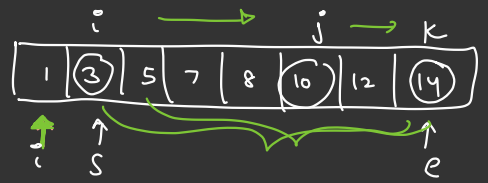


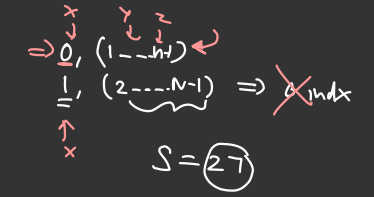
Time
$O(N^3)$

Space
$O(1)$

2

Sort the array $(n \log N)$, $QS \rightarrow O(\log N)$





$a(i), a(j), a(k)$ $\quad i<j<k$

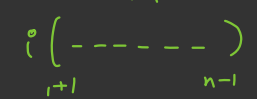
$x + y + z = S$

$y + z = S - x$

Pairs Sum Problem for every x, starting from $i+1 \dots n-1$

✓ Best Approach → 1

Pair $S - a[i]$

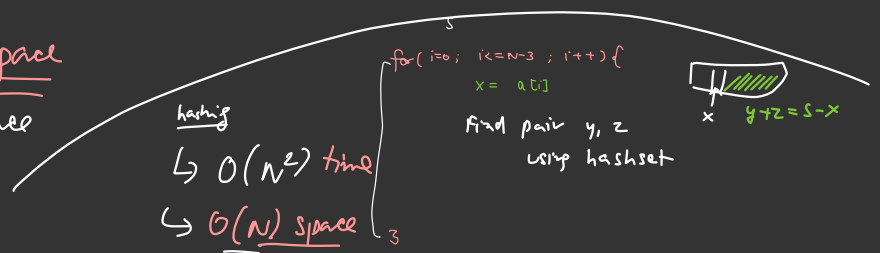


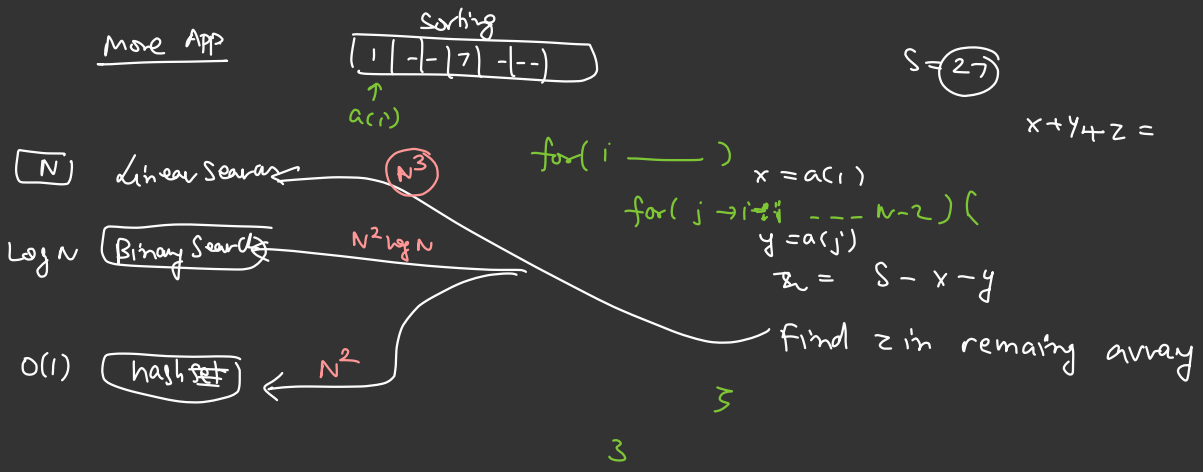

```
for( i=0 ; i<=N-3 ; i++){
    x = a[i]
    ✓ s = i+1
    ✓ e = n-1
    while(s<e){
        // Two pointer to find (y & z)
    }
}
```

Time $\Rightarrow O(N^2)$ + $N \log N$

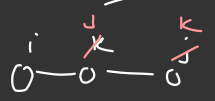
$= O(N^2)$

$= O(1)$ Space + $\log N$ Space of Sorting




More App
Sorting
1 - - 7 - - -
a(1)
S = 27
x+y+z =
N
Linear Search
N^3
for(i ———)
x = a(1)
for(j → i+1 ——— N-2) {
y = a(j)
z = S - x - y
find z in remaing array
Log N
Binary Search
N^2 log N
O(1)
hash set
N^2
3
3

DRY Run for triplet using hashing



create once

$Z = (S-x) - y$

$S = 25$

3, 15, 7
3, 7, 15

hashset<int> hs;

for(i=0, ——— i<=N-3; i++){

x = a(i)

for(j=i+1, j<=N-1, j++){

y = a(j),

if hs.contains(S-x-y) → yes ⇒ :)

hs.put(y)

}

hs.clear() → clear the hashmap for next i

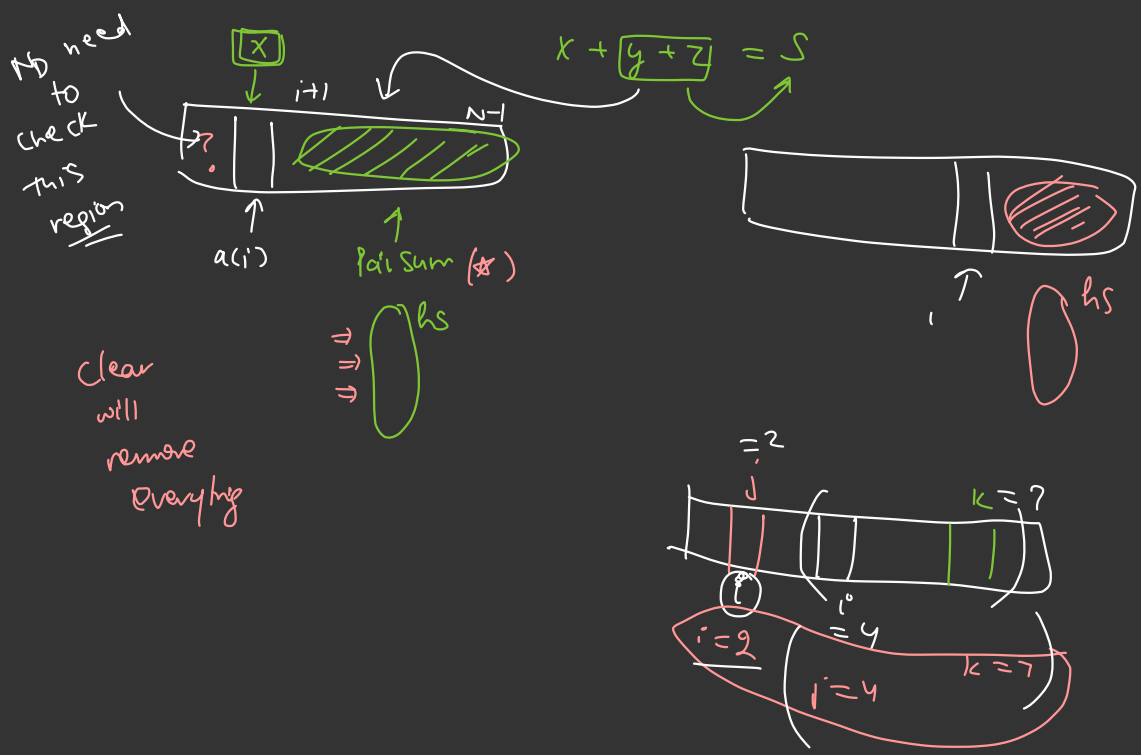
Reuse

Pair Sum using hashing (i+1, N-1)

hs

5
7
10
12
15

x	S-x	y, z =
1	24	3, 24-3
1,	24,	5, 19
⋮		
3	22	5, 16
3	22	7, 15
⋮		
3	22	15, 7 (Yes)


No need to check this region
x
i+1
N-1
x + y + z = S
a(i)
Pal Sum (*)
lhs
Clear will remove everything
lhs
i=2
j
k=?
i=4
i=2
k=7
i=4

Find triplets in the array

st $x + y + z \leq S$] TODO

↑

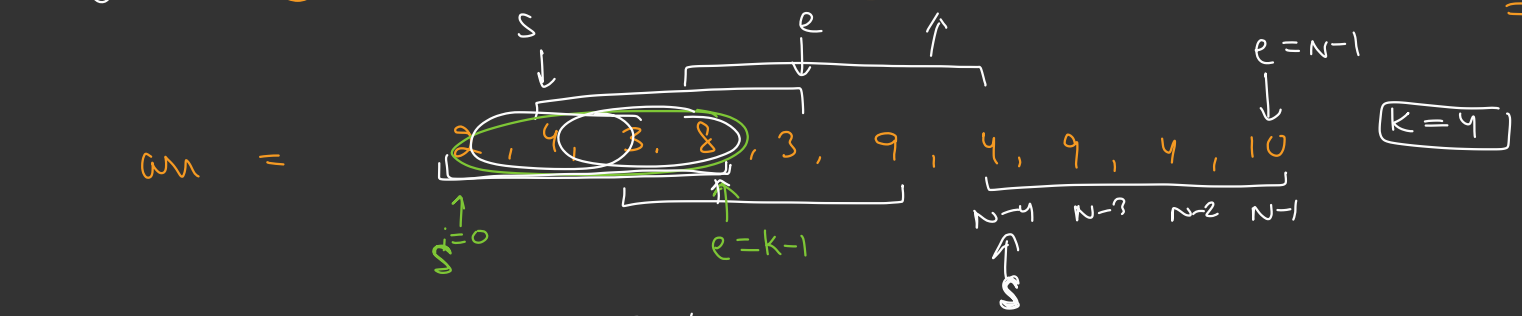
Google Interview Problem

⇒ Count triplets

input

Given N array elements, calculate the no of distinct elements in every subarray of size K.

K = 4

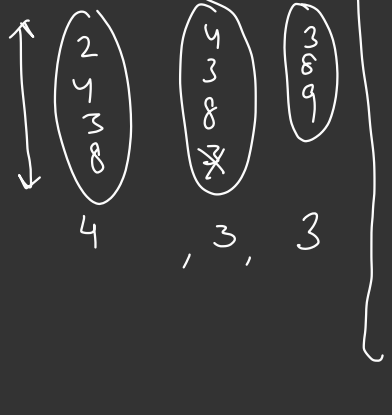


Brute Force
+ hashset

s = 0
e = k-1

hashset<int> hs;

while(e <= N-1) {



⇒ hs.clear();

for(j=s, j<=e, j++) {
hs.put(a[j]);
}

print(hs.size()); // Unique elements

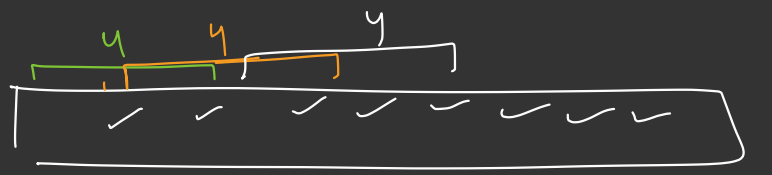
s = s+1, e = e+1

}

K times

Time
O(NK)

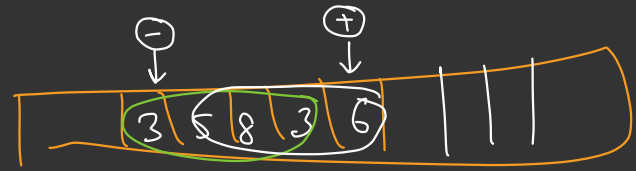
Space
O(K)



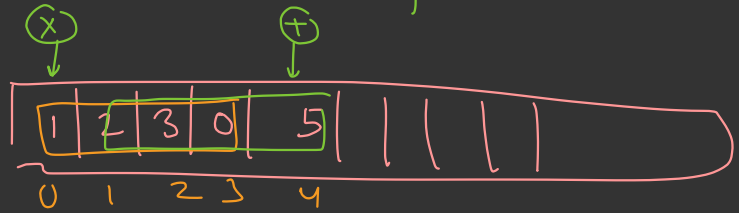
$N \cdot K$

$= O(N \times K)$

5 mins to think → Better?



Challenge?

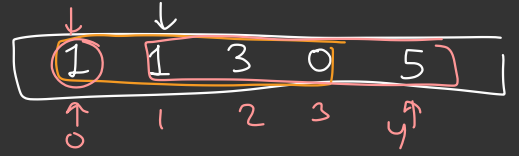


[0-3] hashset (1,2,3,0) Size = 4

[1-4] (2,3,0,5) Size = 4

⋮ ⋮ ⋮

Problem



[0,3] (x,3,0) size = 3

[1-4] (3,0,5) size = 3 ?

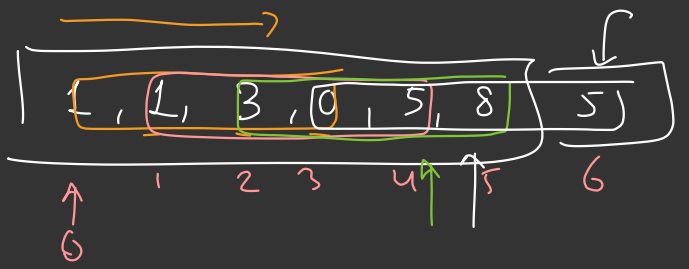
How should we handle this ⇒ Reduce freq of outgoing element by 1.

↳ if freq becomes 0 → Remove element

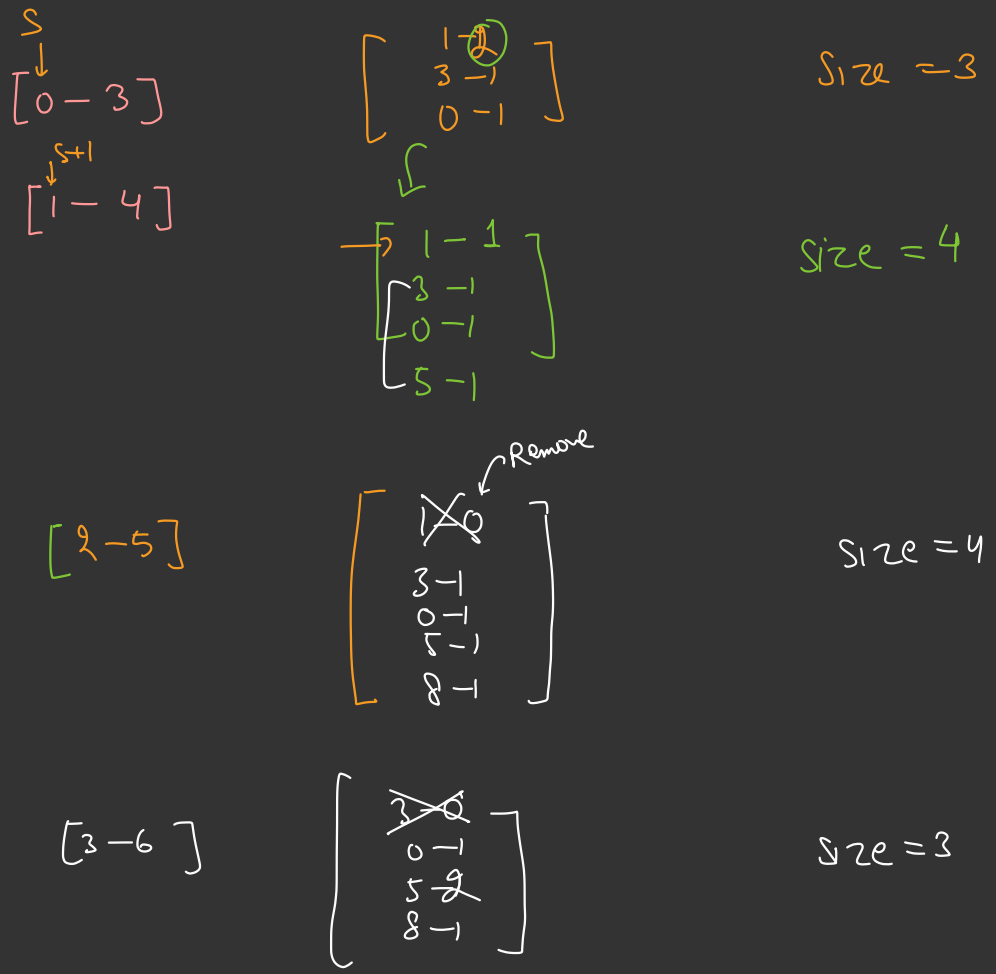
Hashmap + Sliding Window

<element, freq>

↳ only in the subarray



K=4


S
[0 – 3]
S+1
[1 – 4]
[2 – 5]
[3 – 6]
Remove
Size = 3
Size = 4
Size = 4
Size = 3

Hashmap <int, int> hm;

O(N-K)

for(i=0 —— // TO DO){

Add new element

if(arr(i) is in hm)

↳ update freq + 1

else {

hm put (arr(i), 1)

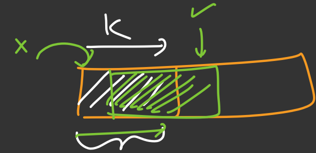
}

Remove outgoing element

- outgoing element → ?
- hm update Freq (outgoing, -1)
- if ⤴ == 0 { hm remove (outgoing Element);

hm.size()

}



Do it separately → O(K)

⇒ for(i=0 — i<k){
hm. put(arr(i), 1)
} or hm update(arr(i) +1)

→ O(N) time

→ O(K) space