09/02/22

DSML Intermediate – DSA

Hashing - 2

Today's Content :

- Q1 — Count distinct for all subarrays of size K
  - ↳ Sliding Window.
- Q2 — Pair with given sum
- Q3 — Longest Consecutive Sequence
- * — optional + HW.

(1 hr)

Q1

Given N array elements,

Calculate the no. of distinct elements in every window (subarray) of size K.

Return an array

I/p 1: [1, 1, 2, 2] — contiguous

N = 4

K = 1

ans = [1, 1, 1, 1] → K = 1 ⇒ N subarrays.

N = 4

K = 2

[1, 1, 2, 2]

[1, 1] → 1 distinct

[1, 2] → 2 distinct

[2, 2] → 1 distinct

ans = [1, 2, 1]

N = 7

K = 3

A = [6, 3, 7, 3, 8, 6, 9]

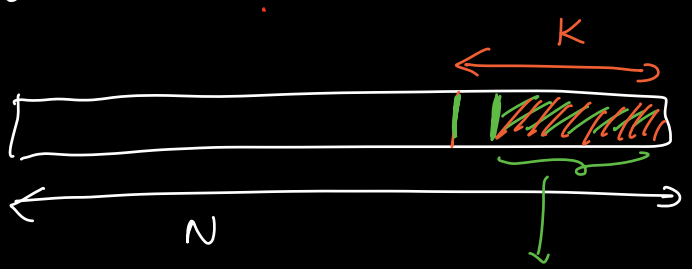
Sliding window

ans = [3, 2, 3, 3, 3]

N − K = 7 − 3 = 4

N − K + 1 = 7 − 3 + 1 = 5

Why $N-K+1$?



$0, 1, 2, \ldots N-1$

N

Possible starting index.

$N - (K-1)$

$= N-K+1$ = # subarrays of size K.

$K-1$ indices can never act as starting pt.

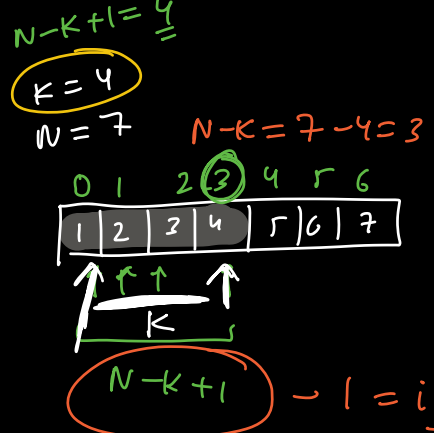
∵ K elements are needed.

Brute Force

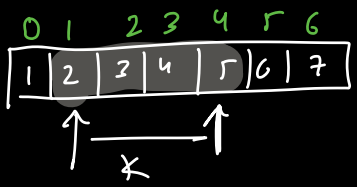
For every window of size K

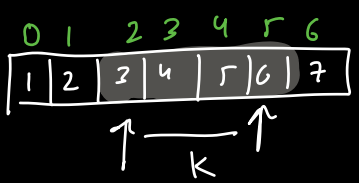
get the no. of distinct elements

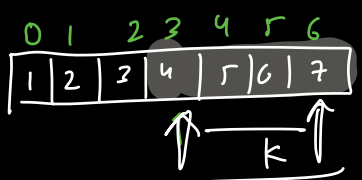
by inserting the elements in a set.




```
ans = []
for start in range(0, N-K+1):
    my_set = set()
    for idx in range(start, start+K):
        my_set.add(A[idx])
    ans.append(len(my_set))
```







Pseudo-Code

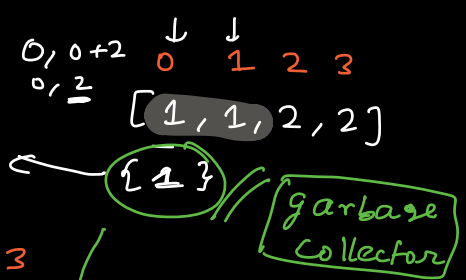
- → count elts that have freq = 1
- → count elts that have freq > 1

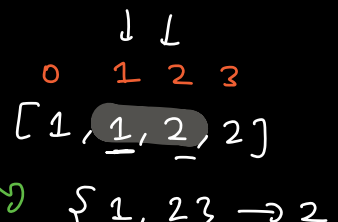
$N = 4$

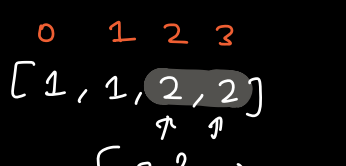
$K = 2$

ans = [1, 2, 1]

```
ans = []
for start in range(0, N-K+1):        # 4-2+1 = 3
    my_set = set()  # starting from scratch
    for idx in range(start, start+K):
        my_set.add(A[idx])
    ans.append(len(my_set))
    # set.clear() →
return ans;
```







$[a, b) \Rightarrow b - a$

start	idx	# iterations
0	[0, K)	K − 0 = K
1	[1, K+1)	K+1−1 = K
2	[2, K+2)	K+2−K = K
⋮	⋮	
N−K	[N−K, N)	N − (N−K) = K

(N−K+1) * K

Total iterations = $(N-K+1)*K$

$N-K+1 \uparrow$

$K \uparrow$

Worst case

→ $K=1$ $\quad (N-1+1)*1 = N$ (N subarray)

mid

$K = N/2$

$\left(N-\frac{N}{2}+1\right)*\frac{N}{2} = \frac{(2N-N+2)}{2}*\frac{N}{2}$

$= \frac{(N+2)N}{4} = \frac{N^2+2N}{4}$

→ $K=N$ $\quad (N-N+1)*N$

$= N$ (1 subarrays)

N
N-1
N-2
1

1

TC: $O(N^2)$

SC: $O(K + (N-K+1))$

K ↑ set

$N-K+1$ ↑ ans array

$\Rightarrow O(N+1)$

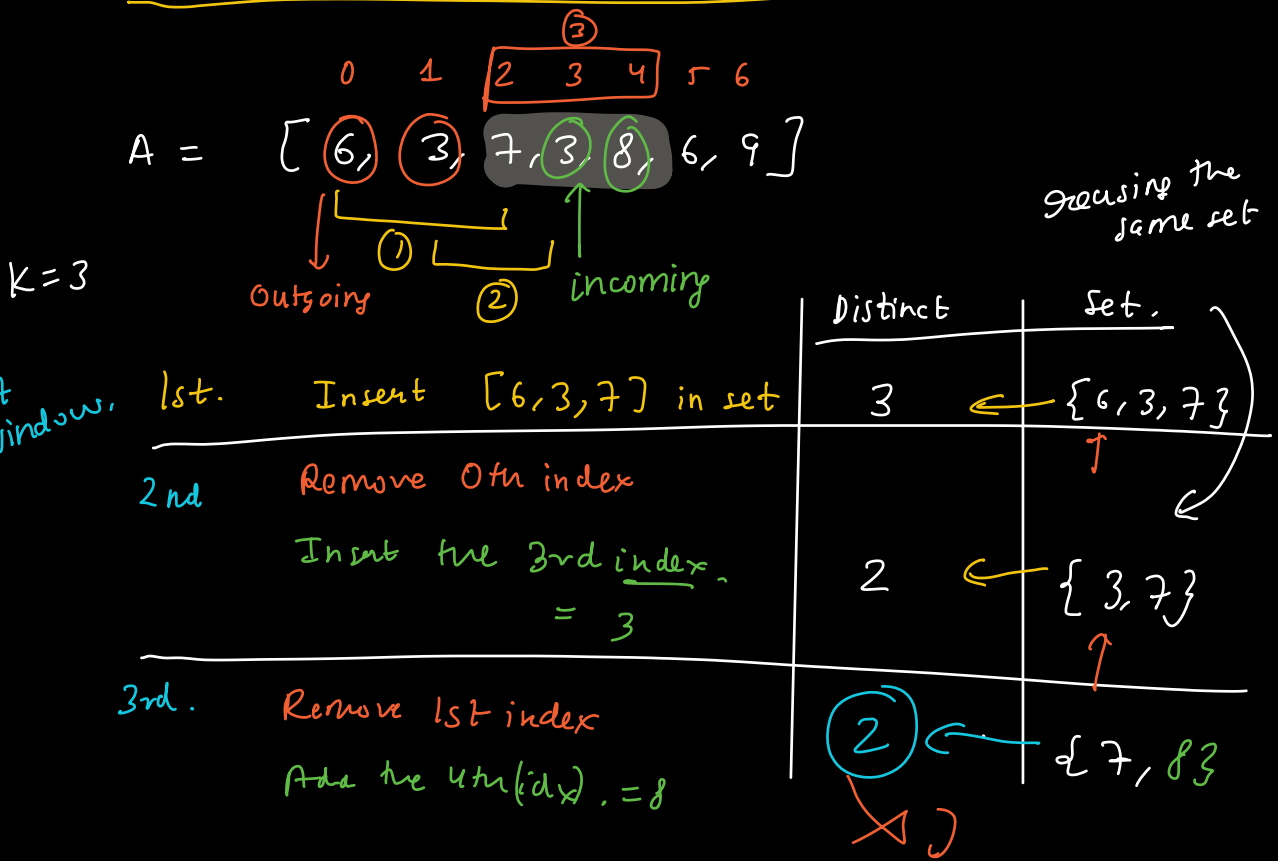
$\Rightarrow$ SC = $O(N)$

SC:

$(N-K+1)*K$

if set is not deleted from memory after every iteration.

Optimisation using Sliding Window



Q) Why HashSet with sliding window does not work?

→ When we delete an elt, we indirectly delete all its occurrences.

→ It does not maintain count/freq.

⇒ Use a HashMap ⇒ dict in Python.

Revenge of HashSet

Sliding Window Technique

0 to (K-1) already inserted.

0 1 2 3 4 5 6

A = [6, 3, 7, 3, 8, 6, 9]

K = 3

K will come next idx

dict { key : val } (elt : freq)

① Insert first K elts in dict : freq_dict

freq_dict

default dict
default 2000
, check might not be needed.

```
freq_dict = { }
ans = [ ]
for i in range(0, K):
    freq_dict[A[i]] += 1
    if A[i] in freq_dict:
        freq_dict[A[i]] += 1
    else:
        freq_dict[A[i]] = 1
```

freq_dict: 6 : 1, 3 : 1, 7 : 1

update. ← add.

ans.append(len(freq_dict))

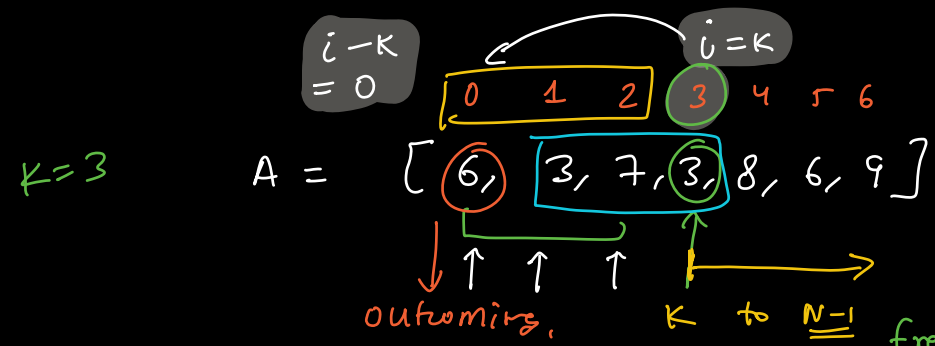
no. of keys

6, 7, 7

6 : 1
7 : 2

[6, 7]

len( ) → # keys



freq_dict

6 : 1
3 : 1
7 : 1

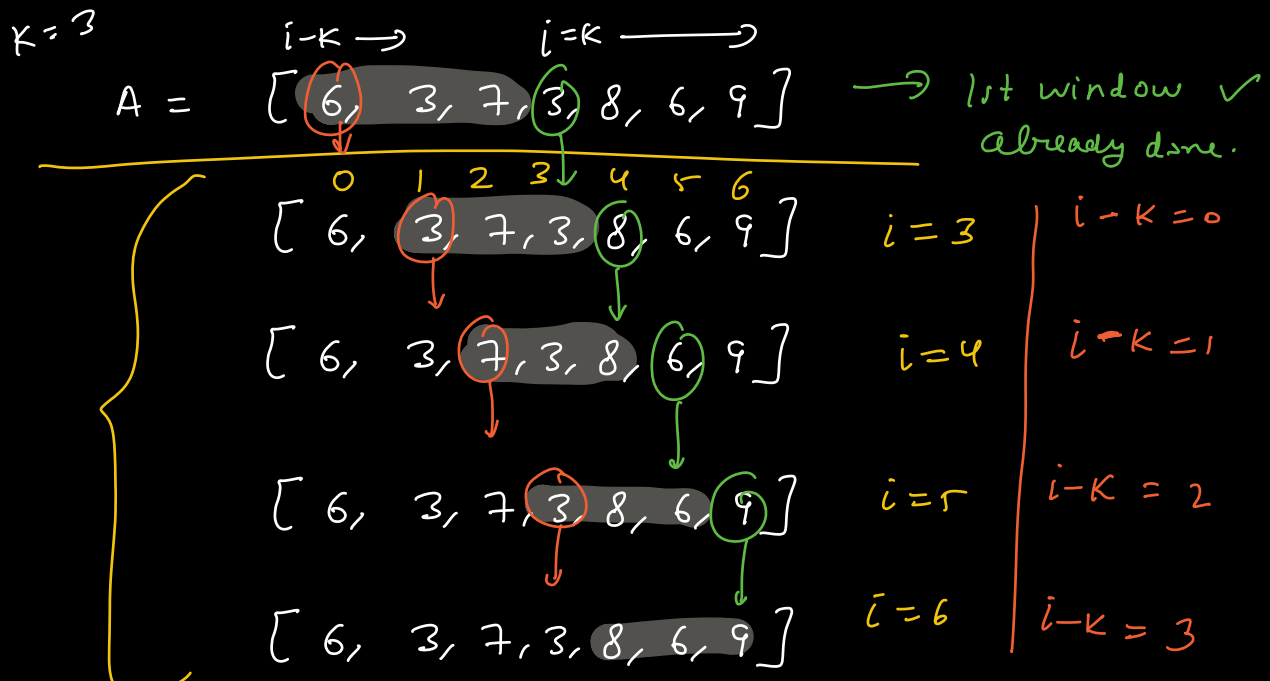
freq_dict

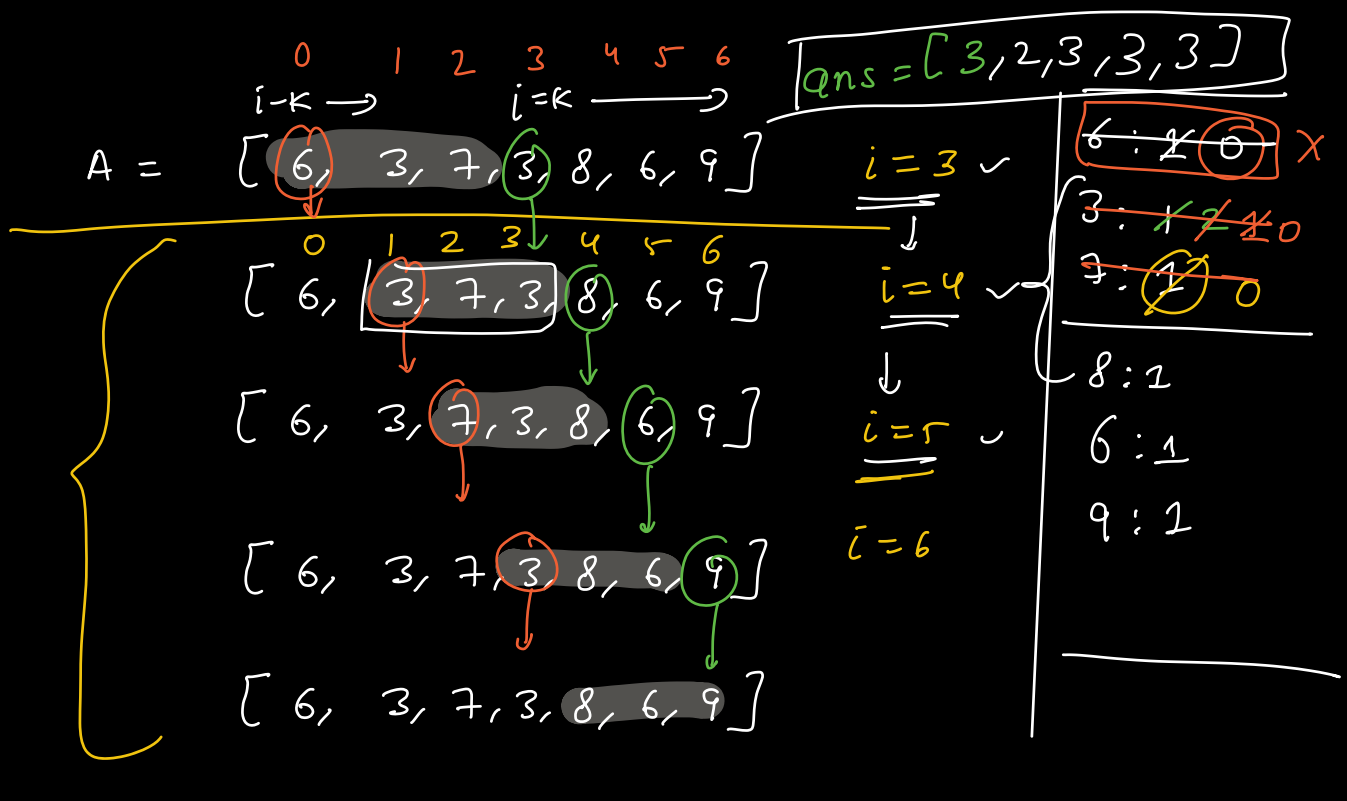
② SLIDING WINDOW STEP :

incoming index : i=K index

Outgoing index : (i-K) index.

```
for i in range(K, N):
    # insert A[i] in HashMap.
    # removing A[i-K] in HashMap.
```






```
for i in range(K, N):
    # insert A[i] in HashMap.
    if A[i] in freq_dict:
        freq_dict[A[i]] += 1
    else:
        freq_dict[A[i]] = 1
    # removing A[i-K] in HashMap.
    freq_dict[A[i-K]] -= 1
    if freq_dict[A[i-K]] == 0:
        # delete the key
        del freq_dict[A[i-K]]
    ans.append(len(freq_dict))
```

elts with freq > 0 (val)

to get correct no. of keys

① Initialize Window

```
freq_dict = {}        (0, K)
ans = []
for i in range(0, K):
    freq_dict[A[i]] += 1
    if A[i] in freq_dict:
        freq_dict[A[i]] += 1
    else:
        freq_dict[A[i]] = 1
ans.append(len(freq_dict))
```

iterations
$k - 0 = K$
$* O(1)$
$= O(K)$

---

② Sliding window step.

```
for i in range(K, N):
    # insert A[i] in HashMap.
    if A[i] in freq_dict:
        freq_dict[A[i]] += 1
    else:
        freq_dict[A[i]] = 1
    # removing A[i-K] in HashMap.
    freq_dict[A[i-K]] -= 1
    if freq_dict[A[i-K]] == 0:
        # delete the key
        del freq_dict[A[i-K]]
    ans.append(len(freq_dict))
```

iterations
$[K, N)$
$N - K$
$* O(1)$
$= O(N-K)$

Total Time = $O(K) + O(N-K) = O(N)$

Total space = dict + ans

dict: $K - 1 + 1$, $K - 1 + 1$, $K - 1 + 1$ — does at any time not exceed K.

ans: $(N - K + 1)$

$$K + (N - K + 1) = O(N + 1) = O(N)$$

Only dict space = $O(K)$

Q 2

True | False.

Google
Amazon
Microsoft

Given N array elements, check if there exists a pair (i, j) such that
arr[i] + arr[j] = K (target)

① i != j

Ip. 1 A = [2, 7, 11, 13] (0 1 2 3)

K = 18 (i, j) = (1, 2) 3 ✓.
K = 9 (0, 1)
K = 17 ✗

K = 18 7 + 11 ✓ ans = True
K = 9 2 + 7 ✓ ans = True
K = 17 ✗ ans = False.

Ip 2 A = [8, 9, 1, -2, 4, 5, 11, -6, 7, 5] (0 1 2 3 4 5 6 7 8 9)

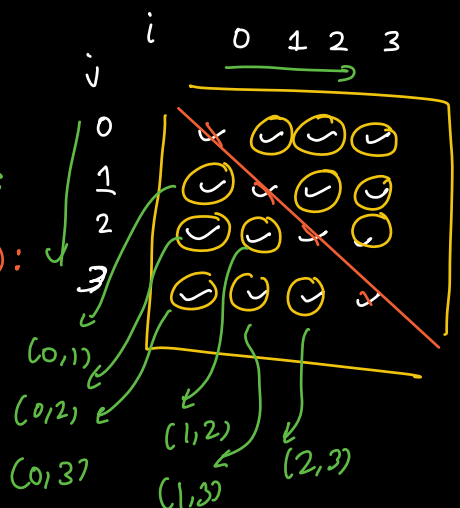
K = 6 ↓ True
(0, 3) 8 - 2 = 6
(2, 5) 1 + 5 = 6
(2, 9) 1 + 5 = 6

K = 100 → False.

Soln 1 → Loop over all the pairs.

N = 4

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



TC: O(N^2)
SC: O(1)

Quiz-4* $A = [7, 1, 10, -3, 9, 7]$ (indices 0 1 2 3 4 5)

$-2 \rightarrow (1, 3)$

$10 \rightarrow (1, 4)$

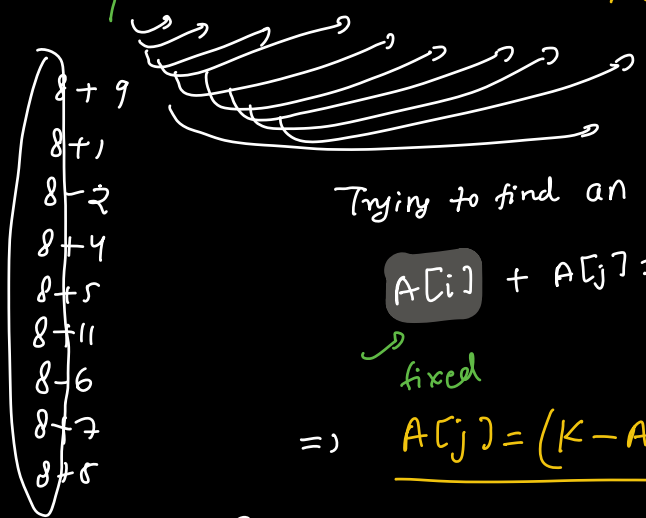
~~18~~ ×

$14 \rightarrow (0, 5)$

Issue

$K = 11$

$A = [8, 9, 1, -2, 4, 5, 11, -6, 7, 5]$ (indices 0 1 2 3 4 5 6 7 8 9)



Trying to find an elt j for i

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

$A[i]$: fixed; K: fixed.

$\Rightarrow A[j] = (K - A[i])$

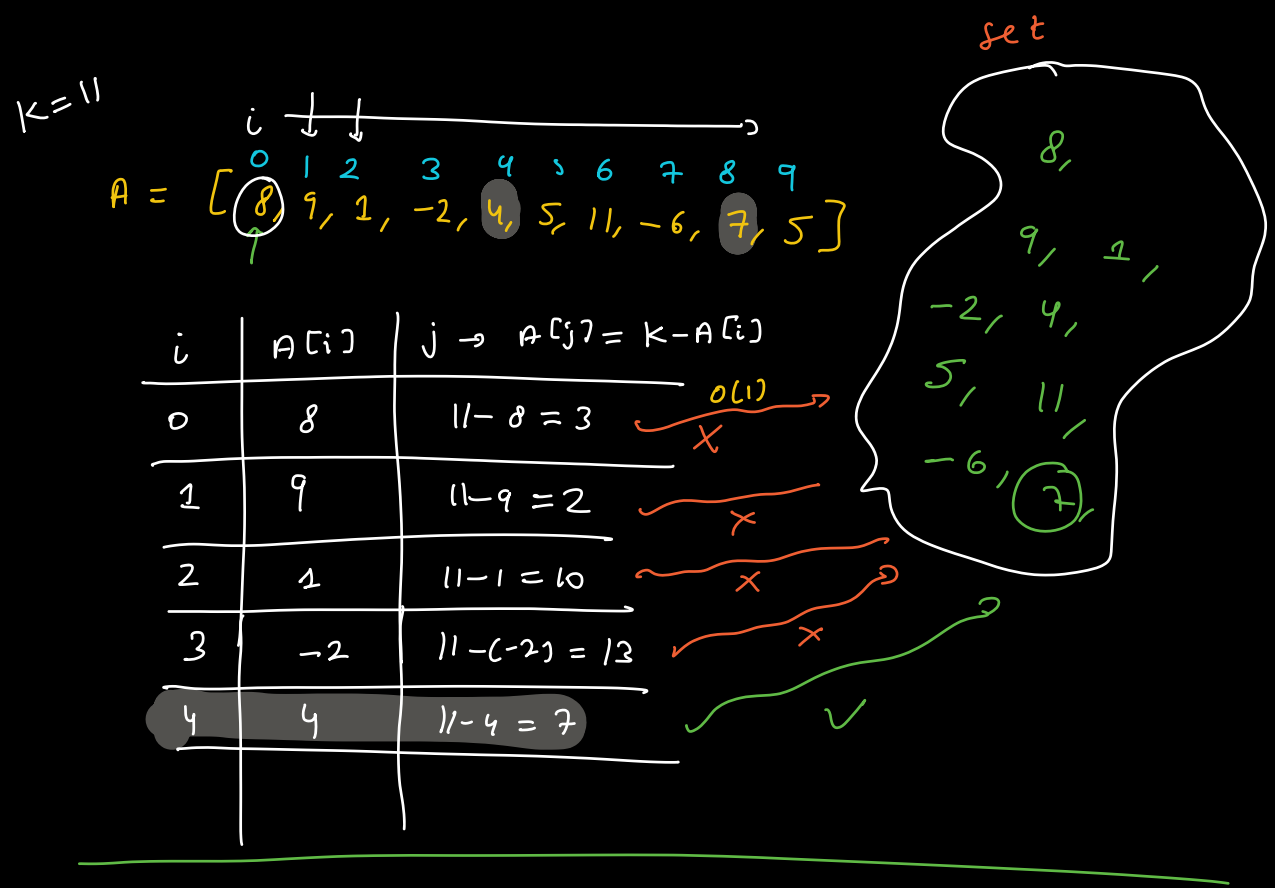
8 : search for 3

(Q) For given $i \rightarrow A[i]$

Check if $j \rightarrow A[j] = K - A[i]$ (known)

Hashset → allows to check in O(1) time. ⇒ doesn't track duplicates

① Add all elements in set. ⟹ O(N)



i	A[i]	j → A[j] = K - A[i]	
0	8	11 − 8 = 3	O(1) ✗
1	9	11 − 9 = 2	✗
2	1	11 − 1 = 10	✗
3	−2	11 − (−2) = 13	✗
4	4	11 − 4 = 7	✓

Reverse of Hashset!

K = 22

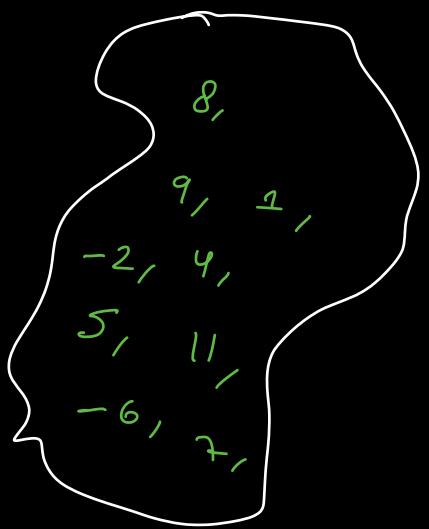
A = [8, 9, 1, -2, 4, 5, 11, -6, 7, 5]

(6, 6) i != j

11 + 11 = 22

i	A[i]	j -> A[j] = K - A[i]
0	8	22 - 8 = 14 X
1	9	22 - 9 = 13 X
2	1	22 - 1 = 21 X
3	-2	22 - (-2) = 24 X
4	4	22 - 4 = 18 X
5	5	22 - 5 = 17 X
6	11	22 - 11 = 11 Issue

set



(Q) Can we handle this case using a freq_map / freq_dict

When 2 elts equal

A[i] == A[j]

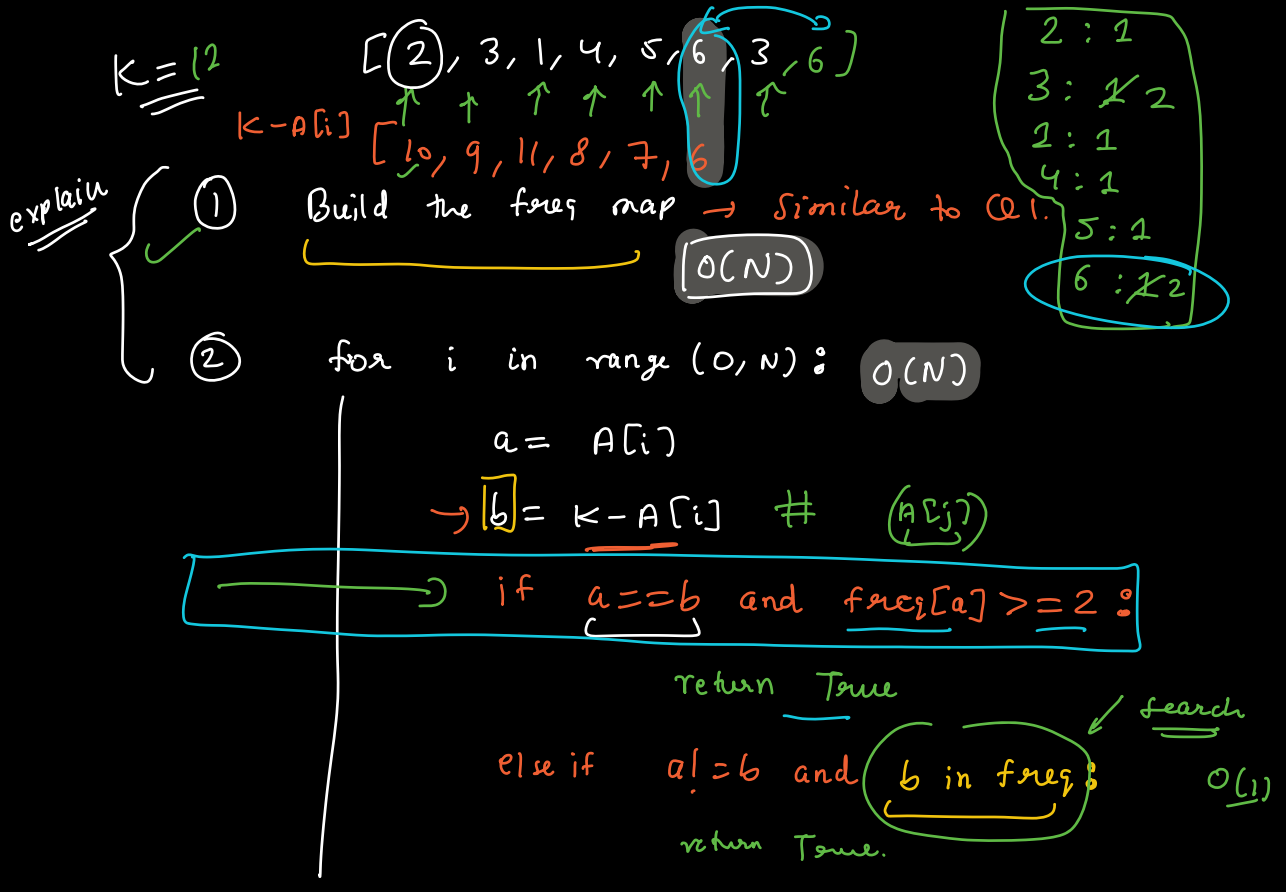
freq[A[i]] >= 2

Pseudo Code

explain

1. Build the freq map → similar to Q1.

2.
```
for i in range(0, N):
    a = A[i]
    b = K - A[i]   # (A[j])
    if a == b and freq[a] >= 2:
        return True
    else if a != b and b in freq:   # search  O(1)
        return True.
return False.
```



Time = O(N)

Space = O(N)

Todo

1. It can be done using Hashset.
   - → While iterating add to set.
   - → Try to think

   Single Pass

2. (i, j) => $A[i] - A[j] = K$

(*) Optional

Q 3

Tricky

Advance level problem

Given N array elements, find the length of the longest sequence (might not contiguous ≠ subarray.) which can be rearranged in a Strictly Increasing order where each element is one more than the prev. element.

Ip 1: A = [100, 4, 200, 1, 3, 2]

Max length = 4

4, 5, 6

4, 100, 200

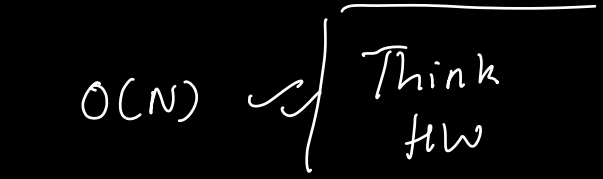
4, 1, 3, 2 => [1, 2, 3, 4]

Seq. => deleting any # elts from anywhere in the array.

sort

[1, 2, 3, 4, 100, 200]

Idea 1 : Sorting. O(NlogN)

Idea 2 : HashMap / HashSet O(N) 

Discuss in Next.

Doubts

2 → 2

3 → 3

2,3 → 6

(2 3)

2*3 = 6

236

$2^3 - 1 = 7$

2

3

6

2,3 → 6

Idea subarrays

O(N)

O(log N)

2,2

2

2

6,3,2

(6)

False

3

2

6,3

3,2 → (6)

(6)

32 → (6)

3245

3

2

4

5

32 → 6

24 → 0

45 → 20

324 → 24

245 → 240

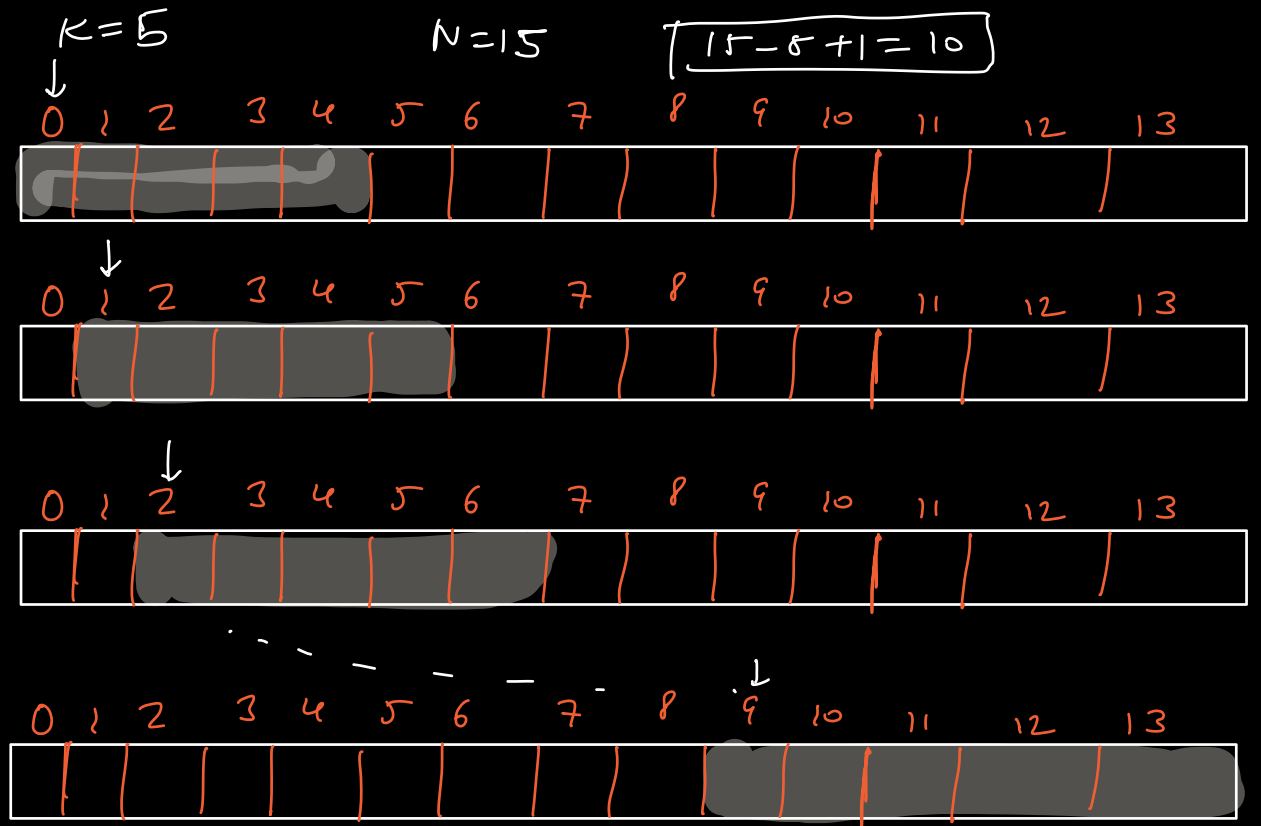
(2,(6,3))

6/3 = 2

2

6

3



Count of duplicate (freq >=2) for all windows of size K.

N = 10

K = 4

2, 3, 2, 1, 4, 2, 3, 4, 5, 6 — ans = 1

2, 3, 2, 1, 4, 2, 3, 4, 5, 6 — ans = 0

2, 3, 2, 1, 4, 2, 3, 4, 5, 6 — ans = 1

2, 3, 2, 1, 4, 2, 3, 4, 5, 6 — ans = 0

2, 3, 2, 1, 4, 2, 3, 4, 5, 6 — ans = 2

2 : ~~2~~ ~~1~~ 1

3 : ~~1~~ 0 ~~1~~

1 : ~~1~~ 0

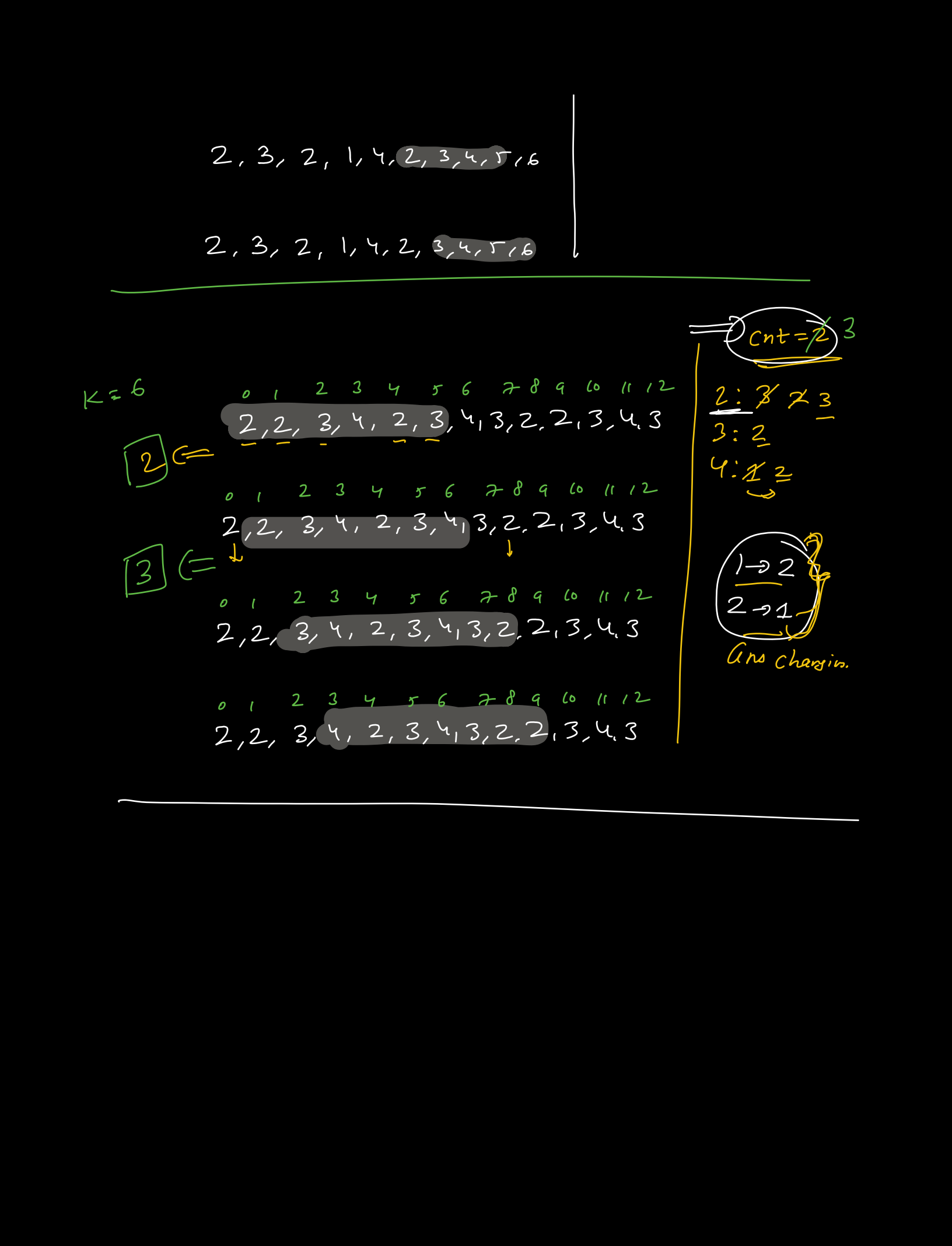
4 : ~~1~~ 2

cnt_dup = 1

cnt_dup = 0

cnt_dup += 1 = ~~1~~ 0

cnt_dup += 1


2, 3, 2, 1, 4, 2, 3, 4, 5, 6
2, 3, 2, 1, 4, 2, 3, 4, 5, 6
K = 6
0 1 2 3 4 5 6 7 8 9 10 11 12
2, 2, 3, 4, 2, 3, 4, 3, 2, 2, 3, 4, 3
2
0 1 2 3 4 5 6 7 8 9 10 11 12
2, 2, 3, 4, 2, 3, 4, 3, 2, 2, 3, 4, 3
3
0 1 2 3 4 5 6 7 8 9 10 11 12
2, 2, 3, 4, 2, 3, 4, 3, 2, 2, 3, 4, 3
0 1 2 3 4 5 6 7 8 9 10 11 12
2, 2, 3, 4, 2, 3, 4, 3, 2, 2, 3, 4, 3
Cnt = 2 3
2: 3 2 3
3: 2
4: 1 2
1 → 2
2 → 1
Ans Changing.