Hashing - 2

Hashmap
O(1)
- Access (key)
- Insert (K, V)
- Update (K, V)
- delete (K) → delete the <K, V> pair.

Set :- Unique Key.

Q.1 Given an Array of size 'N', check if there exists a pair i, j such that

Amazon, FB, Google, MS, Adobe---

$$i != j \;\&\; a[i] + a[j] = K$$

A : { 2, 7, 11, 15 } , K = 18.

↳ True

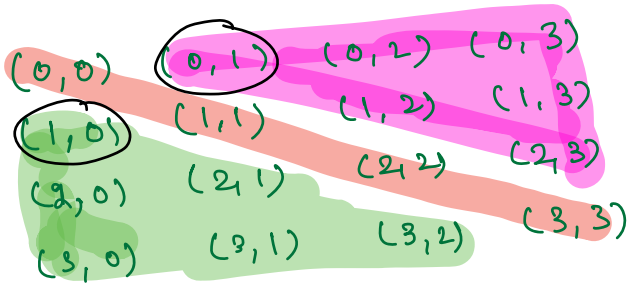
Brute force :-

```
for (i = 0; i < N; i++) {   <- a[i]
    for (j = 0; j < N; j++) {
        if ( i != j && a[i] + a[j] = K)
            return true;
    }
}
return false;
```

TC : $O(N^2)$

SC : $O(1)$

N = 4



a[1] + a[0]

a[0] + a[1]

```
for (i = 0; i < N; i++) {   <- a[i]
    for (j = i+1; j < N; j++) {
        if ( i != j && a[i] + a[j] = K)
            return true;
    }
}
return false;
```

TC: $O(N^2)$

SC: $O(1)$

$a[i] + a[j] = K$.

$a[i]$ — fixing

$a[j] = K - a[i]$

→ for every index 'i', we are fixing $a[i]$ and check for $a[j]$, s.t $a[j] = K - a[i]$

Approach :-

- Insert all the elements in set.
- for every index 'i', check if $(K - a[i])$ is present in the set.

⇒ { 2, 7, 11, 15 }, K = 18

for i=0, a[i] = 2, check for 16 ✗

for i=1, a[i] = 7 —— (11) ↓ Return true

Set { 2, 7, 11, 15 }

⇒ ✗ { 2, 5, 7 }, K = 10. ⇒ false

K - a[i]

10 - 2 = 8 ✗

10 - 5 = 5 ✓

Set { 2, 5, 7 }

{ 2, 7, 11, 15 }, K = 18.

i

1) first check if K - a[i] is present?

2) Insert a[i].

Set { 2, 7 }

{ 2, 5, 7 }

K = 10

10 - 2 = 16. ✗

10 - 5 = 5

a[i] a[j].

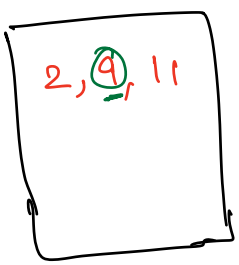
map { <2,1> <5,1> <7,1> }

Approach :

1. Check if K - a[i] is present in the set, if yes, return true.
2. Add a[i] to the set.

⇒ { 2, 9, 11, 9, 15 }, K = 18. ⇒ 9 + 9 = 18.

⇒ true



18 - 2 = 16 ? ✗
18 - 9 = 9 ? ✗
18 - 11 = 7 ? ✗
18 - 9 = 9 ✓ → TRUE

TC : O(N)
SC : O(N)

H.W

0. 1) Count no. of such pairs, a[i] + a[j] = K, i != j.

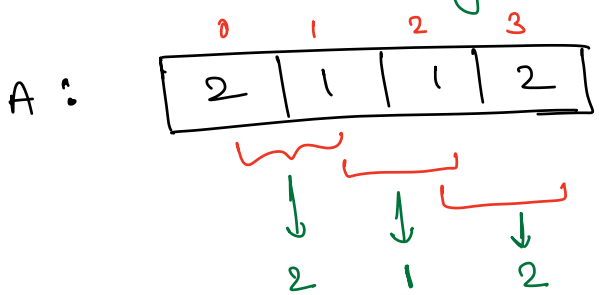
PB * 2) Count the no. of pairs s.t a[i] - a[j] = K. i != j.

30-45 mins

Q.2 Given N array elements and a number 'K' Count the no. of distinct elements in every window of size 'K'.

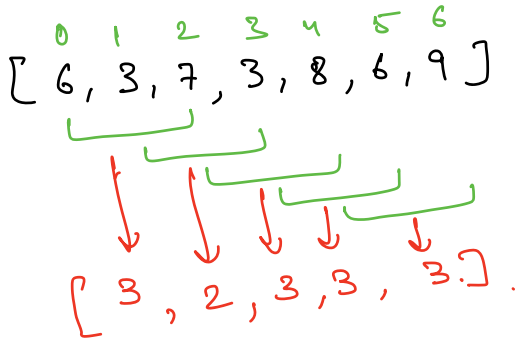
Google, Amazon, FB, Adobe

↳ subarray.



return [2, 1, 2].

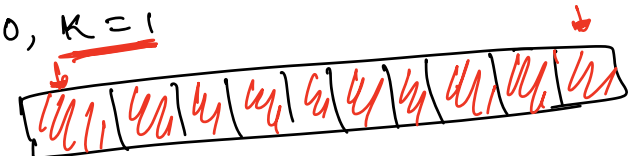
Quiz :-



Brute force :-

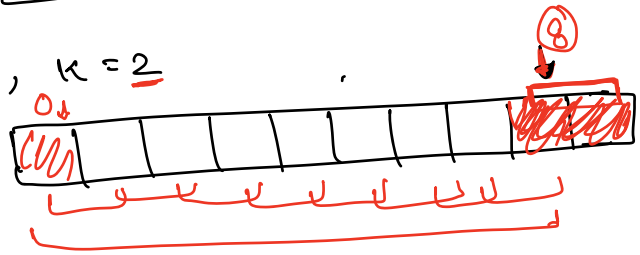
- for every window of size 'K', put all the elements to an empty set.
- add set.size() to ans.

Quiz N = 10, K = 1



⇒ 10 subarrays.

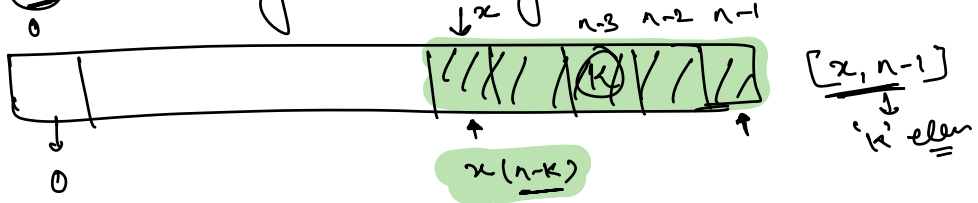
Quiz N = 10, K = 2



[0, 8]

8 - 0 + 1 = 9

Quiz N, (K). No. of subarrays. ⇒ N-K+1



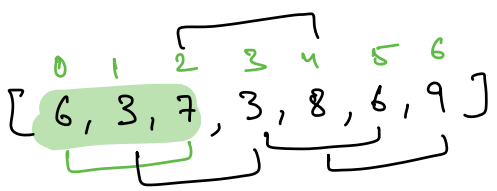
$n - x - x + 1 = k$.

$x = n - k$.

$[0, n-k]$

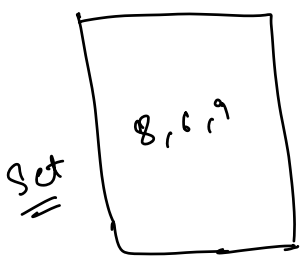
$n - k - 0 + 1 = n-k+1$

↳ No of subarray



K = 3.

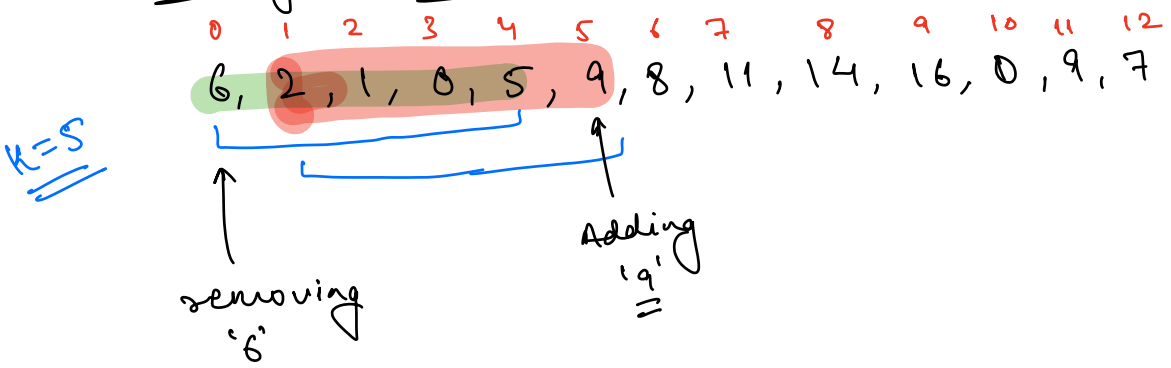
ans : [3, 2, 3, 3, 3].



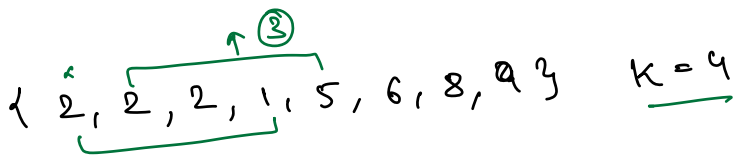
TC : $((N-K+1) * K) = O(N \cdot K)$

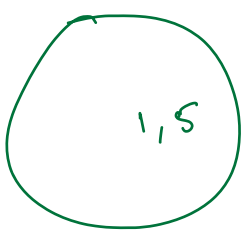
SC : $O(N)$

Sliding Window approach :-

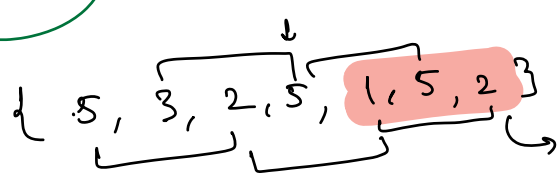


- Push first 'k' elements into the set
- for next window, remove the first element and add the new element to set.

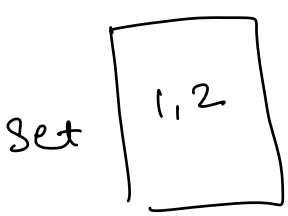




ans : [2, 2 ✗



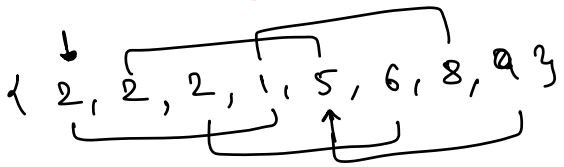
ans : [3, 3, 3, 2, 2]



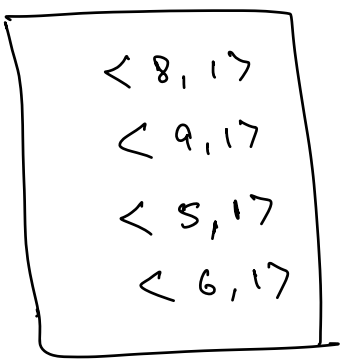
→ This approach won't work.

Correction Using Map :-

↳ Maintain a frequency Map for the array elements.



Map



ans: [2, 3, 4, 4, 4]

Approach :-

O(K) ← { (I) Build a window of size K.
→ Create a map of frequency.

O(N-K) { (II) Iterate over the array :-
→ Remove the 1st element of previous window.
→ Add the new element.

TC : O(N)

SC : O(K).

Code :-

```
// Hashmap for first 'k' elements.
for ( i = 0; i < k; i++ ) {
        if( map.contains(a[i]))
                map[a[i]]++;
        else
                map.insert({a[i], 1})
}
// Sliding window technique.
for (i = 0; i <= N-K ; i++) {
```

- Remove a[i] from Map
- Add a[i+K] to map.

```
ans.push(map.size());
}
```

→ Reduce the frequency of a[i] in map

→ if freq == 0, remove the the entry from map.

Q Given an array, find the length of largest sequence which can be rearranged to a sequence of consecutive numbers.

Google, FB, Amazon, MS, Directi

A: | 100 | 4 | 200 | 1 | 3 | 2 | ⇒ 4

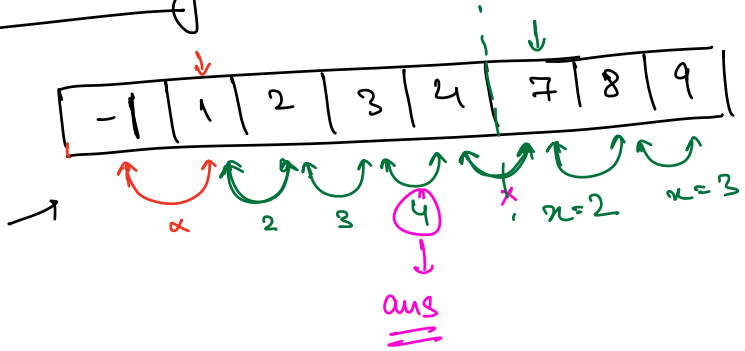
4, 1, 3, 2 → 1, 2, 3, 4
consecutive no's

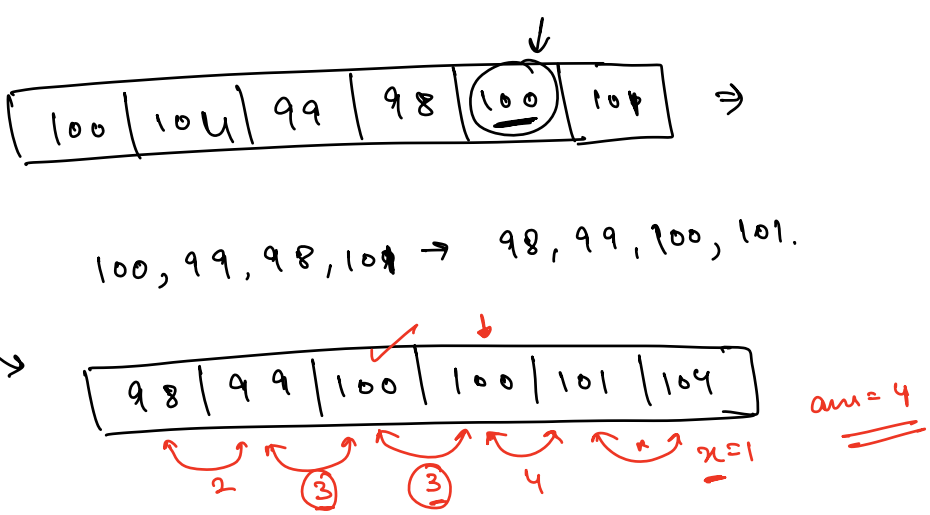
A: | -1 | 8 | 2 | 3 | 7 | 1 | 4 | 9 | ⇒ 4

2, 3, 1, 4 ⇒ 1, 2, 3, 4.

⇒ SORTing Approach :-

1, 2 ⇒ 2



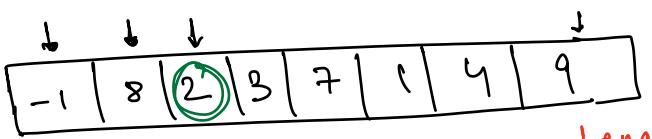


TC : $O(N \log N)$ →

*** SC : $O(1)$ → depends on the sorting Algo.

Brute force :-

- Pick every element from array and try to see the length of consecutive sequence starting with that element.



Length of the consecutive seq.

$O(N^2)$ −1 ⇒ −1, ⇒ ①

8 ⇒ 8, 9 ⇒ ②

$O(N^2)$ ← 2 ⇒ 2, 3, 4 ⇒ ③

$O(N^2)$ ← 3 ⇒ 3, 4, ⇒ ②

$O(N^2)$ ← 7 ⇒ 7, 8, 9 ⇒ ③

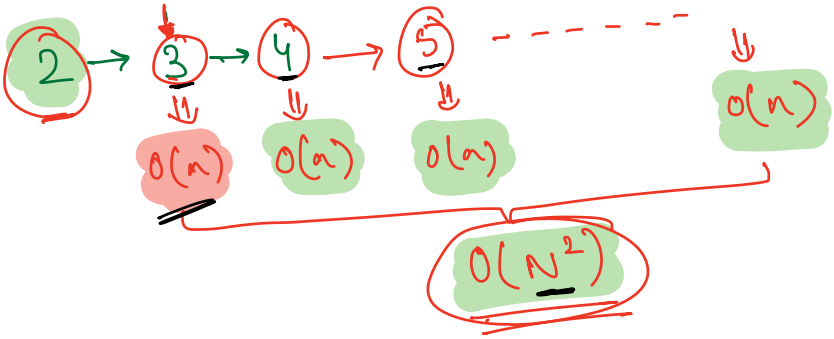
1 ⇒ 1, 2, 3, 4. ⇒ ④

4 ⇒ 4, ⇒ ①

$O(N^2)$ 9 ⇒ 9 ⇒ ①

⇒ ④.

Search TC in array ⇒ $O(N)$



TC : $O(N^3)$

SC : $O(1)$.

⇒ Searching is very costly in above approach.
↳ $O(N)$

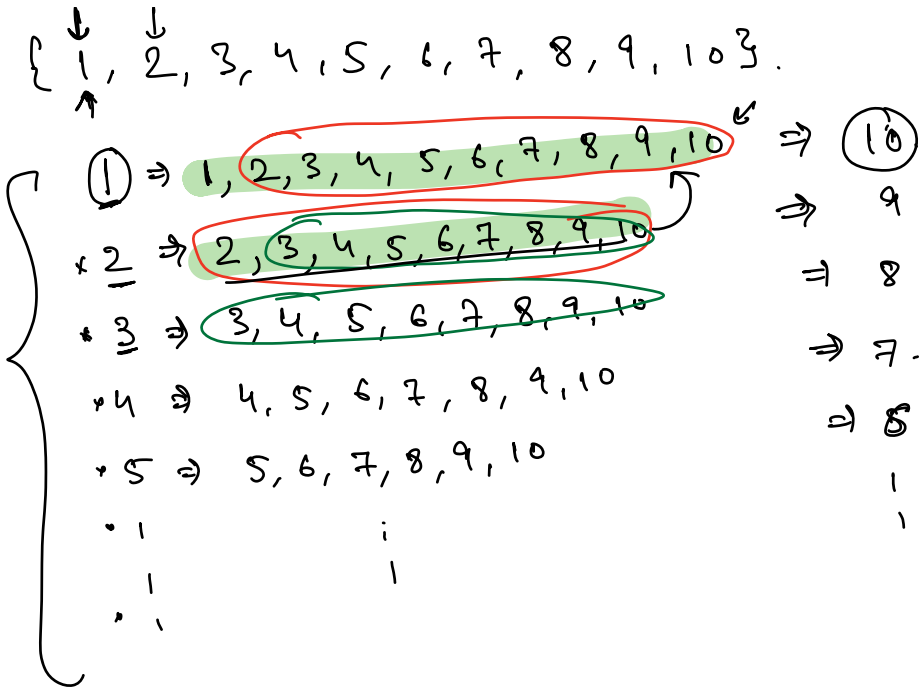
⇒ We can use SET to optimize searching.

Code :-

```
// Build the SET.
for( i=0; i<N; i++){
    set.add(a[i]);
}                                   } → O(N)

// Count the consecutive sequence length for every a[i].
for( i=0; i<n; i++) {  → O(N)
    ans = 0; x = a[i]
    while ( set.contains( x )){   → O(1)
        ans++;
        x++;
    }
    maxL = max(ans, maxL);
}
```

TC ⇒ $O(N^2)$, SC ⇒ $O(N)$



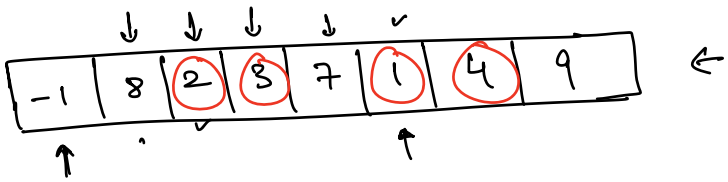
⇒ for any a[i],
if (a[i] - 1) present in the set
then there's no need to check for
a[i].

```
// Build the SET.
for( i=0; i< N; i++){
        set.add(a[i]);
}                                   } → O(N)
// Count the consecutive sequence length for every a[i].
for( i= 0; i< n; i++ ) { → O(N)
    if( !set.contains(a[i]-1) ){
        ans = 0; x=a[i]
        while( set.contains( x )){
                ans++;                  } → O(1)
                x++;
        }
        maxL = max(ans, maxL);
    }
}
```

TC ⇒ O(N)

SC ⇒ O(N)

#



-1 ⇒ -1, ⇒ ①

8 ⇒ X

2 ⇒ X

3 ⇒ X

7 ⇒ 7, 8, 9 ⇒ ③

! ⇒ 1, 2, 3, 4 ⇒ ④

④ → X

9 → X