FB, Google ...

Q1 Given an array of N integers and a no. K. Return true if there exists a pair (i, j) s.t.

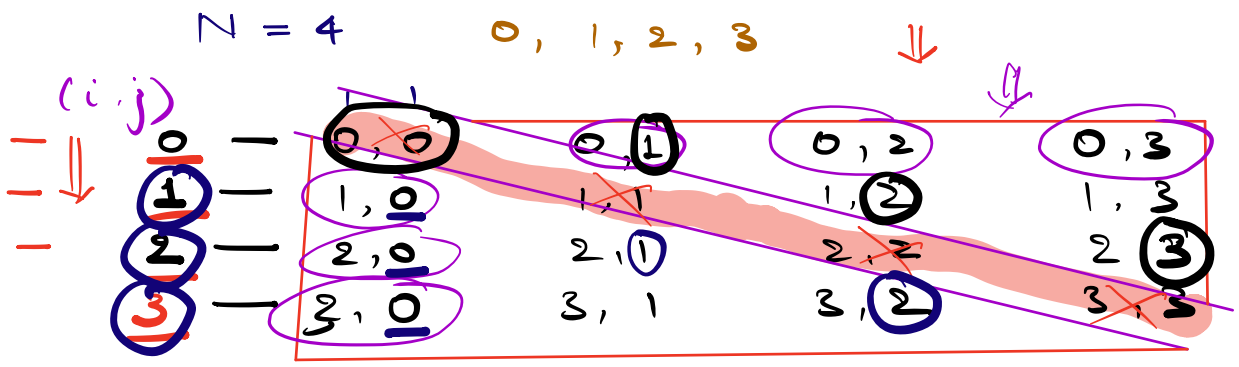
$$A[i] + A[j] = K \qquad (i \neq j)$$

Eg: $A \Rightarrow [2, 7, 11, 15, 7]$ (indices 0 1 2 3 4)

$K = 18 \Rightarrow$ True $(1, 2)$
$K = 14 \Rightarrow$ True $(1, 4)$
$K = 20 \Rightarrow$ false.

Sol^n 1) Brute force

$\Rightarrow$ Check for all pairs if sum = K



$i \Rightarrow 0$ to $i-1 \Rightarrow$ n^2 possibilites.

$\Rightarrow$ It is okay to solve for one quadrant.

Upper

```
for(i=0; i<N; i++){
```

Lower

```
for(i=0; i<N; i++){
```

```
// for (j = i+1 ; j<N ; j++){
    if (A[i] + A[j] == K){
        return true;
    }
}
}
```

```
for (j = 0 ; j<i ; j++){
    if (A[i] + A[j] == K){
        return true;
    }
}
}
```

$i = N-1$

$j = N$

$i = 0$

$j = 0$

$T.C. = O(n^2)$

$S.C. = O(1)$

2) Can you optimize T.C. ??

$\Rightarrow$ (i) $\Rightarrow$ $A[i] + X = K$

$X = K - A[i]$

$\forall i$, search for $(K - A[i])$ using hashset

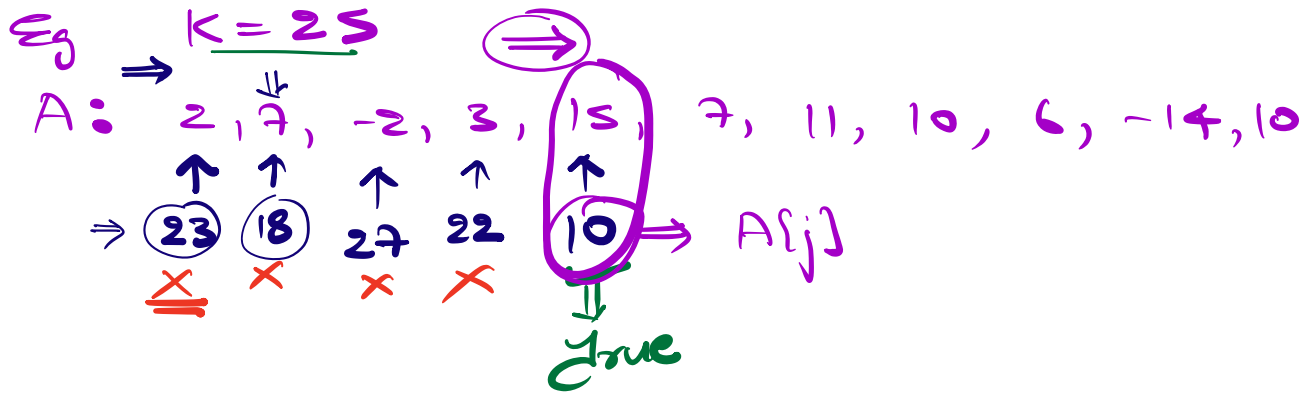
Steps

1) Insert all the array elements in a hashset

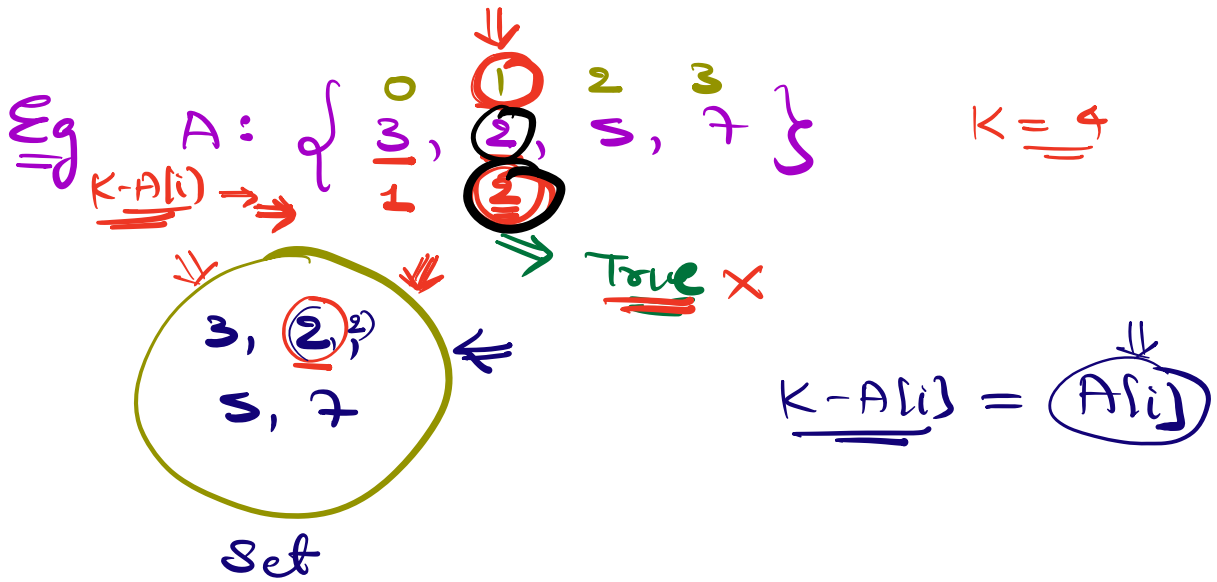
$\Rightarrow$ 2) Iterate over the array &

$\forall_i$, search for (K - A[i]) in hashset.

If present, return true. Else false.



T.C. = O(N), S.C. = O(N)



Soln

→ HashMap

Step 1 → Create a freq map from the array.

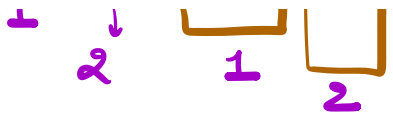
(freq Map)

Step 2

```
for (i=0; i<N; i++) {
    x = K - A[i];
    if ( freqMap.containsKey (x)){
        count = freqMap [x];
        ⇒ if (A[i] == x) {
            if (count > 1) {
                return true;
            }
        }
        else {
            return true;
        }
    }
}
return false;
```

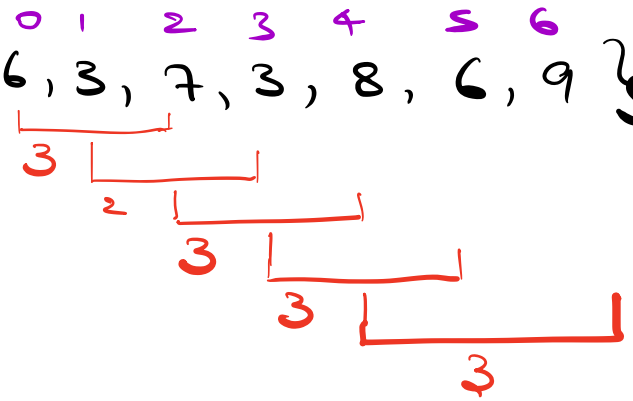
Q-2 Given an array of size N
Calculate no. of distinct element in every sliding window of size K.

Eg A: [1, 1, 2, 2, 3] (indices 0 1 2 3 4) K = 2



Return [1, 2, 1, 2]

Eg A: { 6, 3, 7, 3, 8, 6, 9 } K=3



Return [3, 2, 3, 3, 3]

N

length	start index first	start index last
1	0	n-1
2	0	n-2
3	0	n-3
⋮		
K	0	n-K

1) Hash Set

(i, j) ⇒ K = j - i + 1

j = K + i - 1

⇒ for (i = 0; i <= n-K; i++) {

```
D        j = K+i-1
        // set<Int> s;
// SW) ⇒ ⇒ for (k = i; k <= j; k++){
K                s.add(A[k]); ⇒
                }
                // ans = s.size()
        }
```

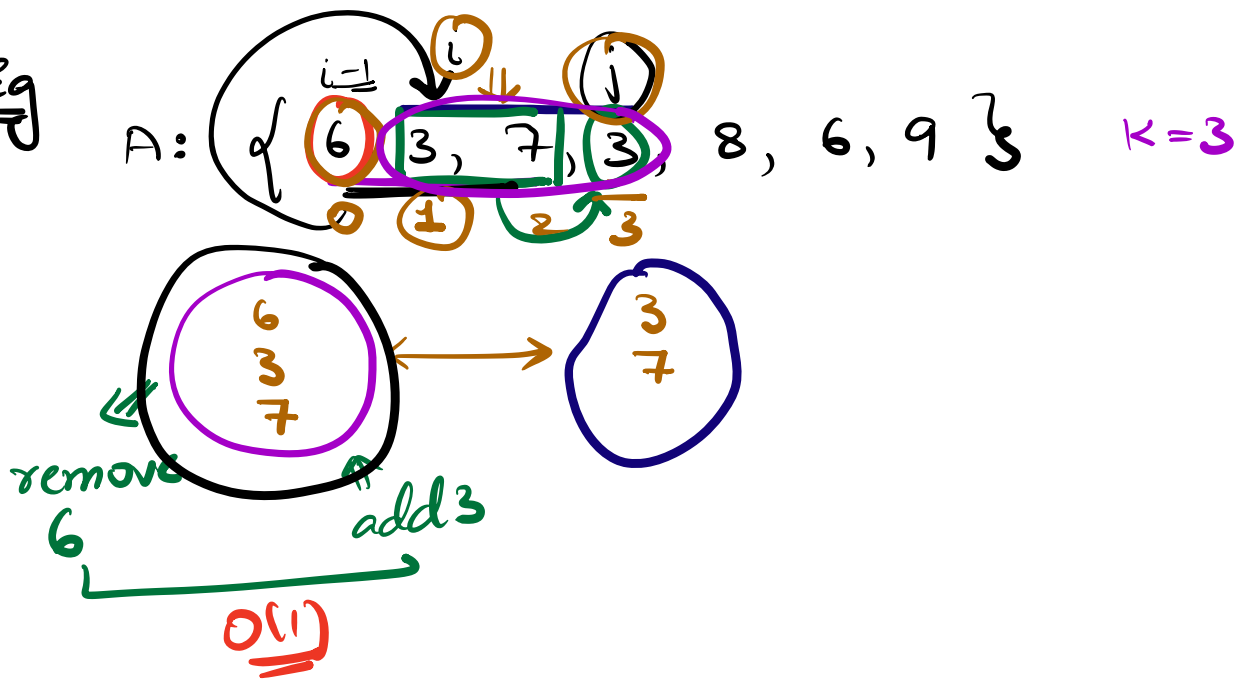
$T.C. = O((n-k) \times k)$

$= O(nk - k^2)$

$= O(nk)$

$K \leq N$

Eg A: { 6, 3, 7, 3, 8, 6, 9 } $K = 3$



Code

```
for (i = 0; i < K ; i++) {
        set.add(A[i]);
}
```

Ans = set.size();

⇒

```
for (i = 1; i <= (n-K); i++) {
        j = K+i-1;
        set.remove(A[i-1]);
        set.add(A[j]);
        // ans = set.size()
}
```

T.C. = O(N)
S.C. = O(K)

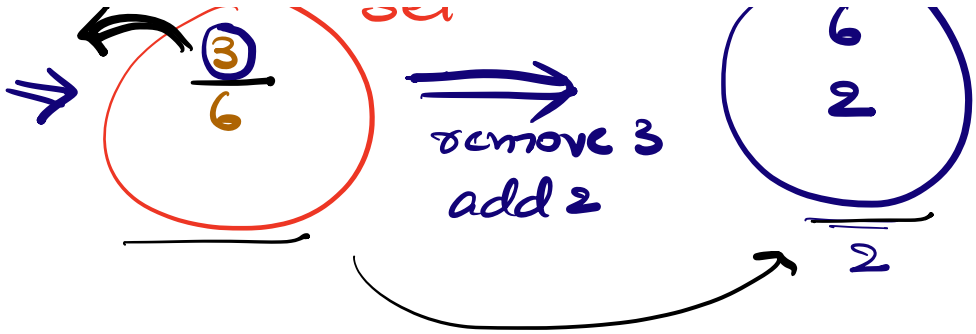
Any missing corner cases ??

Eg ⇒ A: [3, 6, 3, 2, 1, 3, 5, 7, 9]
indices: 0 1 2 3 4 5 6 7 8

K = 3

2 2

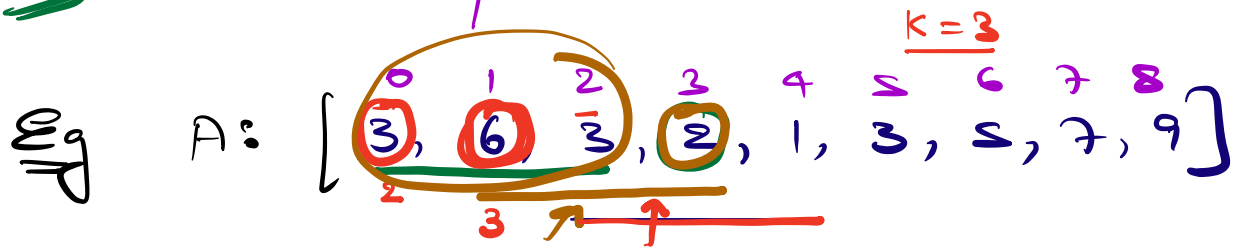
set

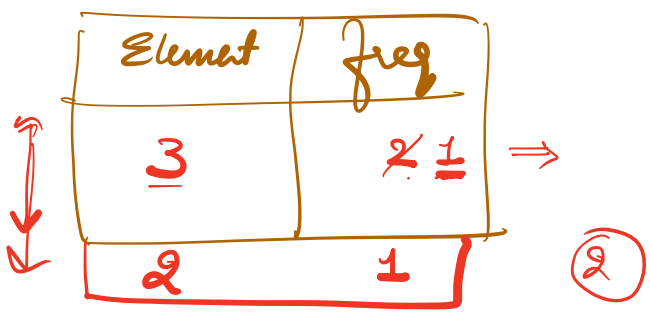


Soln

Hash Map

Eg





Removing 3
Adding 2

0 2

0, 2 ⇒ 3

Code

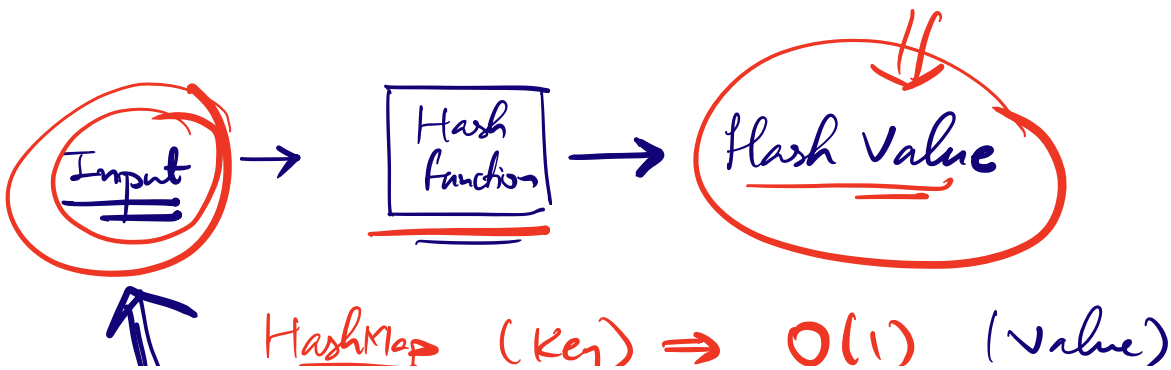
```
for (i=0; i < k; i++) {
        if ( freqMap.contains(A[i]) ){
                freqMap[A[i]]++;
        }
        else {
                freqMap.put (A[i], 1);
}
```

```
for (i = 1;  i <= (n-k) ;  i++ ){
        j = k+i-1;
        freqMap[A[i-1]]--;
        if ( freqMap[A[i-1]] == 0){
                freqMap.remove(A[i-1]);
        }

        if ( freqMap.contains(A[j])){
                freqMap[A[j]]++;
        }
        else {
                freqMap.put(A[j], 1);
        }

}
```

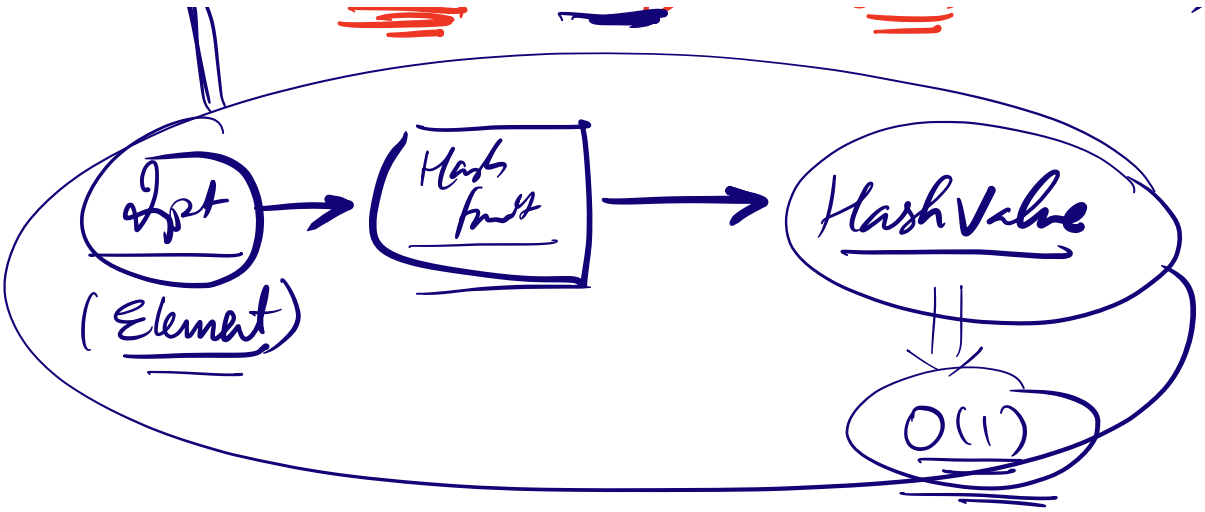
Key Value

T.C. = O(N)

S.C. = O(K)



HashMap (Key) ⇒ O(1) (Value)



$N \Rightarrow O(N)$

$N \rightarrow O(2N) = O(N)$

Q Given an array of size N

Return TRUE if there exists a subarray with sum = K
↳ given

Eg A: [4, 2, -1, 3, 10, 5, 6, 3] (indices 0 1 2 3 4 5 6 7)

PS: 4, 6, 5, 8, 18, 23, 29, 32

K=12 TRUE

12

j = 4

i-1 = 1

i = 2

Soln 1) Brute force

∀ subarrays, check if sum = K

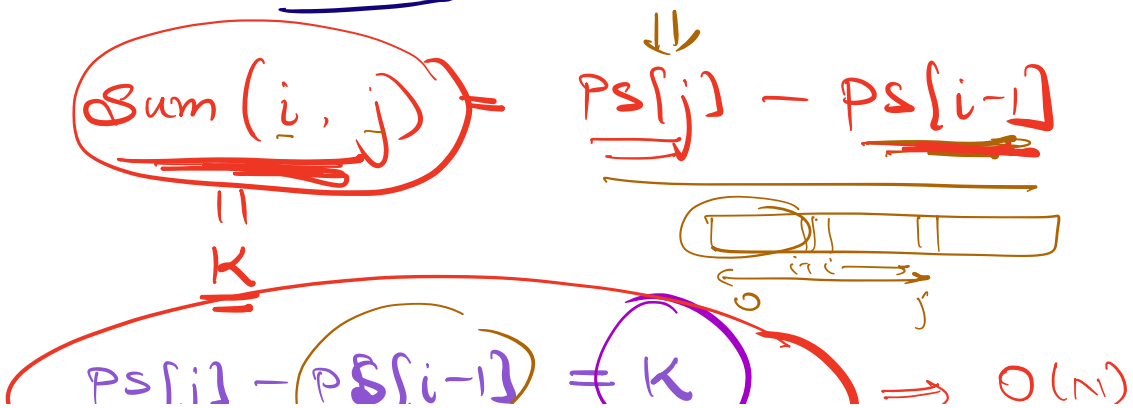
$O(n^2)$

T.C. = $O(n^2)$

```
for (i = 0; i < N; i++) {
    sum = 0;
    for (j = i; j < N; j++) {
        sum += A[j];
        if (sum == K)
            return true;
    }
}
return false;
```

T.C. $O(n^2)$

2) Optimise !?



A ⇒ PS

Find 2 ele from PS array s.t.

diff = K

H.W.

Ques 2