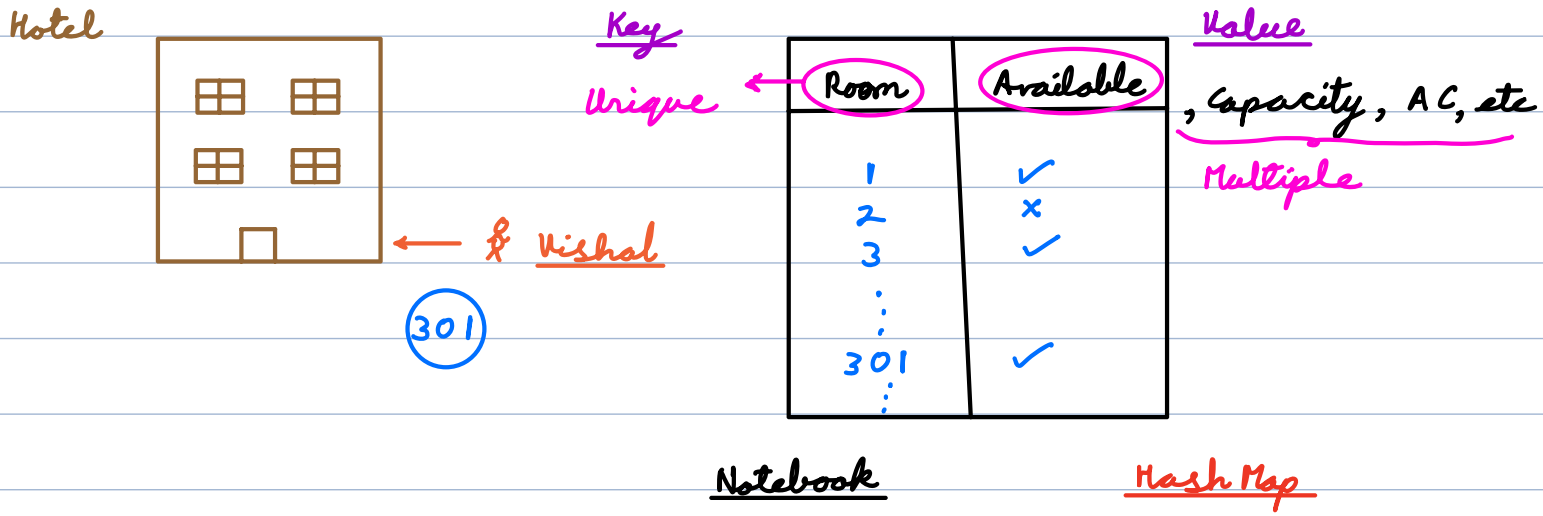


Q → Store population of every country →

country name → population

String → Long

Q → Store # states of every country →

country name → # states

String → int

Q → Store name of all states of every country →

country name → list of states

String → list of string

Q → Store population of each state in every country →

country name → (state → population)

String → HM < String, int >

< Key, Null > → < Key > Hash Set

✓ HashSet / HashMap → Hashing → store keys in random order. TC = O(1)

TreeSet / TreeMap → Maintain keys in Sorted Order (use balanced BST internally) TC = O(log N)

LinkedHashMap / LinkedHashSet → Maintain keys in order of insertion.

Operations

To update → insert the same key again.

HashMap

1) insert(key, value) / put(key, value)
2) size()
3) delete(key) / remove(key)
4) get(key)
5) containsKey(key)
   ⋮

TC = O(1)

HashSet

1) insert(key) / add(key)
2) size()
3) delete(key) / remove(key)
4) contains(key)
   ⋮

TC = O(1)

Q → Given N elements & Q queries.

Find frequency of an element for multiple queries.

	0	1	2	3	4	5	6	7
A =	2	6	3	6	3	8	2	6

(Ans.)

Query	Frequency
6	3

8	1
5	0

Bruteforce → ∀query, travel & find frequency.

TC = O(Q * N) SC = O(1)

Frequency Array → F[i] → frequency of i

(F[i]: ↑ key, ↑ value)

```
for i → 0 to (N-1) {
    F[A[i]] ++
}
```

// Find freq. of x → F[x]

SC = O(Range of A[i])

	0	1	2	3	4	5	6	7
A =	2	1	3	3	0	3	2	2

	0	1	2	3
F =	1	1	3	3

Hash Map → < int, int >

A[i] → frequency of A[i]

```
for i → 0 to (N-1) {
    if (hm.containsKey(A[i])) {
        v = hm.get(A[i])
        hm.put(A[i], v+1)
    } else {
        hm.put(A[i], 1)
    }
}
```

```
for i → 0 to (Q-1) {
    x = B[i]   // B → queries
    if (hm.containsKey(x))
        print(hm.get(x))
    else
        print(0)
}
```

hm.getOrDefault(x, 0)

TC = O(N + Q) SC = O(N)

Q → Given N integers, find first non-repeating value.
Return -1 if all elements are repeating.

	0	1	2	3	4	5
N = 6	A = [1	3	5	1	2	3]

Ans = 5

N = 5 A = [2 3 2 4 4] Ans = 3

Bruteforce → ∀ elements check if that element is repeating in future. TC = O(N²) SC = O(1)

Sol → 1) Store frequency ∀ A[i].

2)

```
for i → 0 to (N-1) {
    if (hm.get(A[i]) == 1)
        return A[i]
}
```

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

Q → Find the count of distinct elements in A[].

```
      0  1  2  3  4
A = [ 3  5  6  5  6 ]   {3, 5, 6}
```

Ans = 3

```
      0  1  2
A = [ 5  5  5 ]
```

Ans = 1

```
for i → 0 to (N-1) {
    hs.add(A[i])
}
return hs.size()
```

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

Q → Given an integer array, check if there exist a subarray with 0 sum.

```
      0  1  2  3  4  5
A = [ 2  2  1 -3  4  3 ]
```

Ans = true

Bruteforce → ∀ subarray, calculate sum & check if sum = 0.

$\frac{N*(N+1)}{2}$

TC = $O(N^3) \rightarrow O(N^2)$

SC = O(1)

subarray sum → prefix sum

subarray sum l ___ r ⇒ P[r] − P[l−1] // l > 0

P[r] − P[l−1] = 0

if (l == 0) → P[r]

⇒ P[r] = P[l−1] ✓ P[r] = 0 ✓

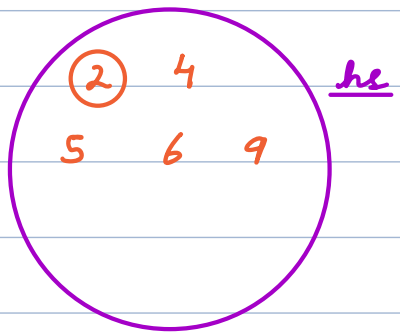
P[0] = A[0]

P[i] = A[0] + A[1] + ... A[i]

```
if (P[0] == 0) return true
hs.add(P[0])
for i → 1 to (N-1) {
        P[i] = P[i-1] + A[i]
        if (P[i] == 0) return true
        hs.add(P[i])
}
if (hs.size() == N) return false
 else   return true
```

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

	0	1	2	3	4	5
A =	2	2	1	-3	4	3
P =	2	4	5	2	6	9



---

contest _ learner id → not attempted / score

Key — Value — int

-1

Q → Given an integer array, check if there exist a pair (i, j) s.t $A[i] + A[j] = K$ & $i != j$.

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

K = 6 A[2] + A[5] = 6 Ans = true

K = 22 Ans = false

	0	1	2	3	4	5
A = [	3	5 ✓	1	2 ✓	1	2]

K = 7 Ans = true

K = 10 Ans = false

Bruteforce → ∀ i, j check if sum = K & i != j.

TC = O(N²) SC = O(1)

```
for i → 1 to (N-1) {
    for j → 0 to (i-1) {
        if (A[i] + A[j] == K)
            return true
    }
}
return false
```

A[i] + A[j] = A[j] + A[i]

i < j or i > j ✓

TC = O(N²) SC = O(1)

check if there exist A[j] st. A[j] = K - A[i]. } Hash Set

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

→ 8 9 2 | -2 4 5 | 11 -6

Search K - A[i] → complete array x

K = 6 6 - 8 = -2 Ans = true

K = 9

9 - 8 = 1 ✗

9 - 9 = 0 ✗

9 - 2 = 7 ✗

9 - (-2) = 11 ✓ Ans = true

K = 4

4 - 8 = -4

4 - 9 = -5

4 - 2 = 2 Ans = ~~true~~ → false

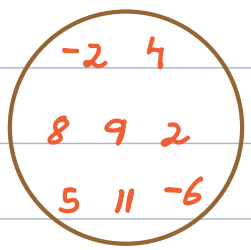
(i == j)

Hashset should contain values from 0 to (i-1).

∀i, 1) Search for K - A[i] ✓

2) Insert A[i] in Hashset ✓ (j < i)

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



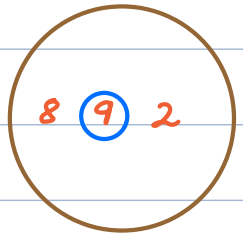
K = 4

K - A[i]	
4 - 8 = -4	4 - 5 = -1
4 - 9 = -5	4 - 11 = -7
4 - 2 = 2	4 - (-6) = 10
4 - (-2) = 6	4 - 4 = 0
4 - 4 = 0	

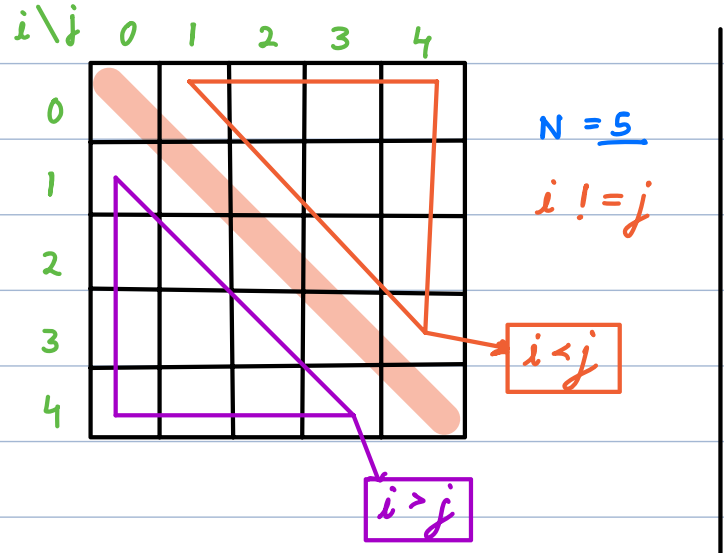
Ans = false ✓

K = 7

7 - 8 = -1	7 - (-2) = 9
7 - 9 = -2	Ans = true
7 - 2 = 5	



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



N = 5

$i != j$

Q → count pairs with sum = K.

$i != j$ & $(i, j) = (j, i)$.

	0	1	2	3	4	5
A = [	3	5	1	2	1	2]

K = 3

i	j
2	3
2	5
3	4
4	5

Ans = 4

Find # times a value is present

⇒ Hash Map (freq.)

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

<A[i], freq.>

8 → 1 4 → 1

9 → 1 5 → 2

2 → 1

-2 → 2

K = 7

K - A[i]	count = 0
7 - 8 = -1	+ 1
7 - 9 = -2	+ 1
7 - 2 = 5	+ 1
7 - (-2) = 9	+ 1

$7 - 4 = 3$ $\quad +2$

$7 - 5 = 2$ $\quad 6$ (Ans)

$7 - (-2) = 9$

$7 - 5 = 2$

$7 - 2 = 5$ $\quad TC = O(N)$ $\quad SC = O(N)$

Q → check if there exist a subarray with sum K.

subarray → continuous part of array.

$A = [\underset{0}{2}\ \underset{1}{3}\ \underset{2}{9}\ \underset{3}{-4}\ \underset{4}{1}\ \underset{5}{5}]$ $\quad K = 8$ $\quad$ Ans = true

$K = 12$ $\quad$ Ans = true

$A = [\underset{0}{5}\ \underset{1}{10}\ \underset{2}{20}\ \underset{3}{100}\ \underset{4}{105}]$ $\quad K = 110$ $\quad$ Ans = false

subarray sum ⇒ prefix sum

$i \_\_ j$ $\quad i <= j$

$P[j] - P[i-1] = K$

$\Rightarrow P[i-1] = P[j] - K$

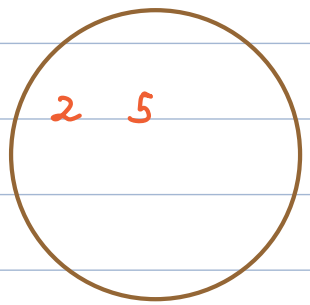
$(i == 0)$ $\quad P[j]$

$\forall j$, check $P[j] - K$ is present in prefix sum or $P[j] = K$. ✓

$A = [\underset{0}{2}\ \underset{1}{3}\ \underset{2}{9}\ \underset{3}{-4}\ \underset{4}{1}]$

P → ~~2~~ ~~5~~ 14

$K = 12$ $\quad$ Ans = True



Hash Set (prefix sum)

$P[j] - K$

$2 - 12 = -10$

$5 - 12 = -7$

$14 - 12 = 2$ ✓ $\quad$ Ans = true

```
// HashSet → hs
p = 0
for i → 0 to (N-1) {
    p += A[i]
    if (p == K) return true
    if (hs.contains(p - K)) return true
    hs.add(p)
}
return false
```

$P[j] - P[i-1] = K$

$\Rightarrow P[j] = K + P[i-1]$

$\Rightarrow \boxed{P[j] - K = P[i-1]}$

TC = O(N) SC = O(N) (hs)

HW → Count subarrays with sum = K.

Q → Given an array, find count of distinct elements in every window of size K.

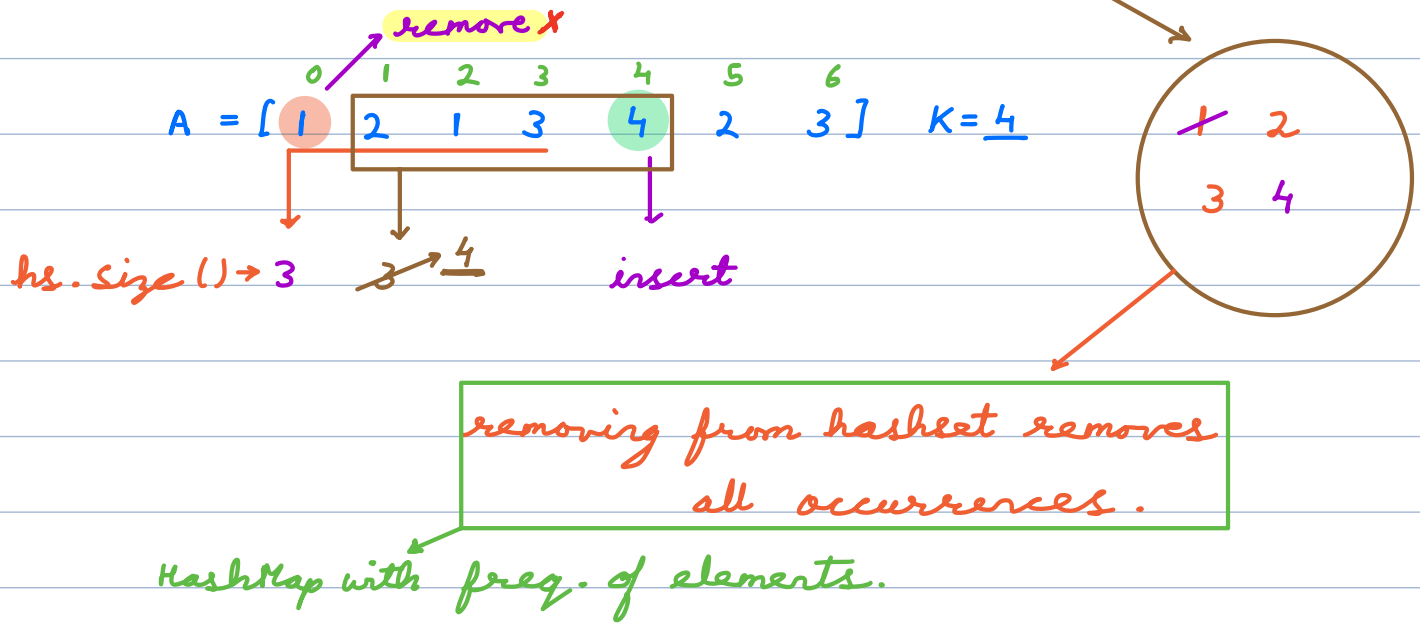
	0	1	2	3	4	5	6	
A = [	1	2	1	3	4	2	3	]

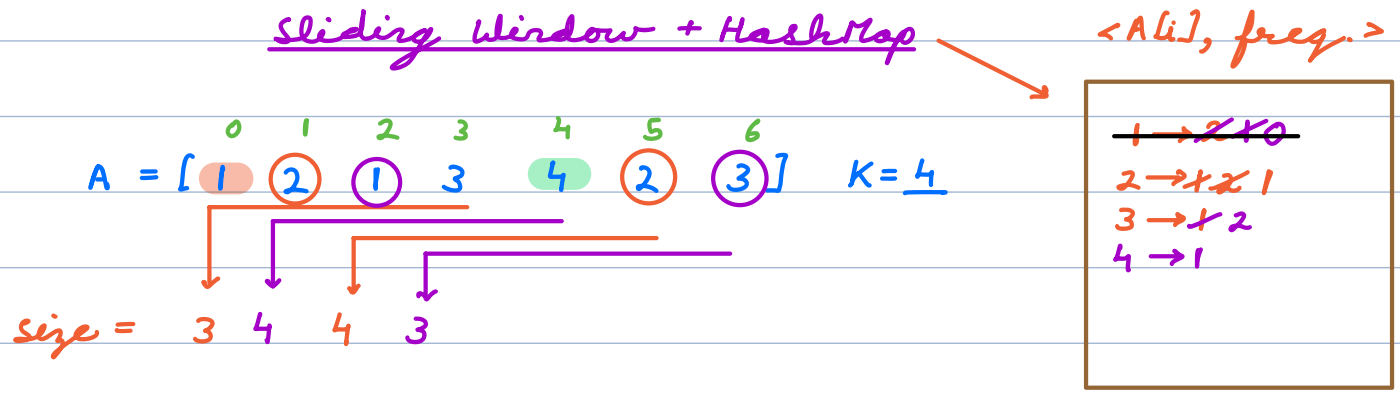
K = 4

Ans → 3 4 4 3

✓ { fixed size subarray → sliding window
distinct elements in array →
i) Insert all elements in hashset
ii) Ans = hs.size()

Sliding window + Hashset






```
for i → 0 to (K-1) {
    f = hm.getOrDefault(A[i], 0)
    hm.put(A[i], f+1)
}

print(hm.size())
for i → K to (N-1) {        // window (i-K+1) ____ i
    f = hm.getOrDefault(A[i], 0)        add A[i]
    hm.put(A[i], f+1)
    f = hm.get(A[i-K])                  remove A[i-K]
    if (f == 1) hm.remove(A[i-K])
    else hm.put(A[i-K], f-1)
    print(hm.size())
}
```

TC = O(N) SC = O(K)