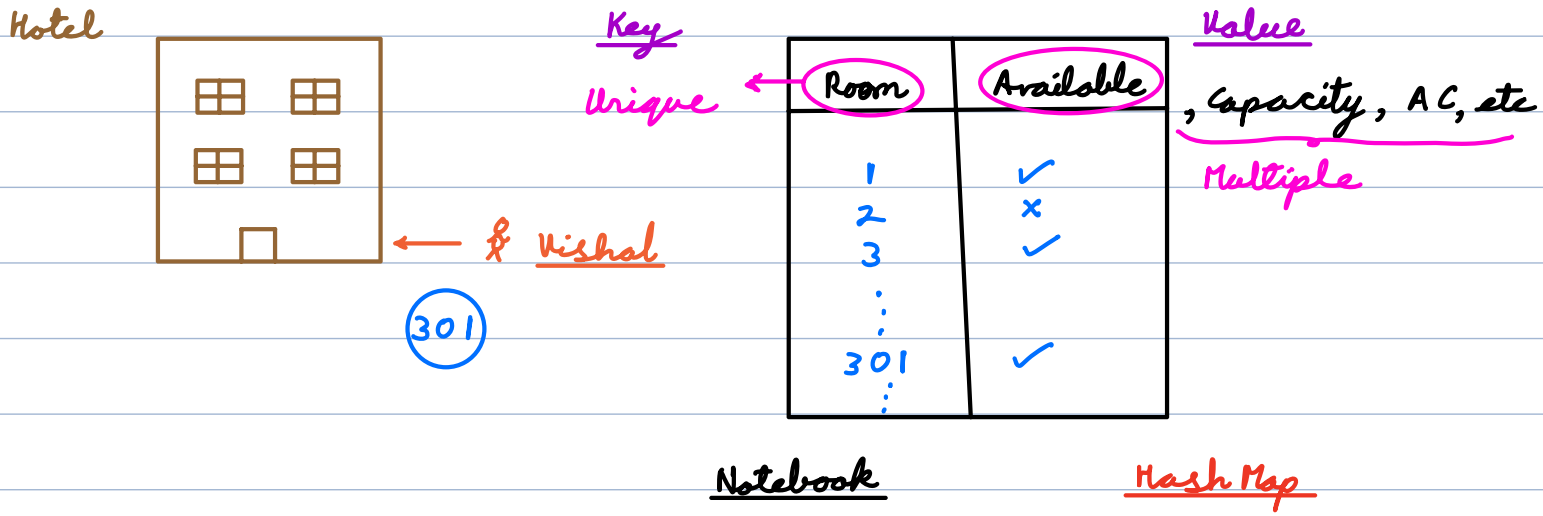


Q → Store population of every country →

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 ✓

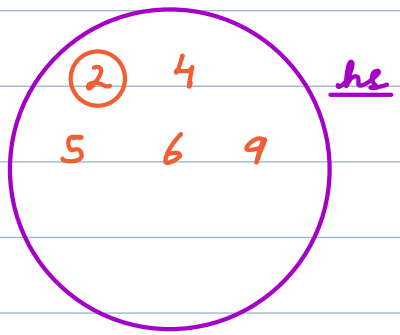
P[0] = A[0]

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
