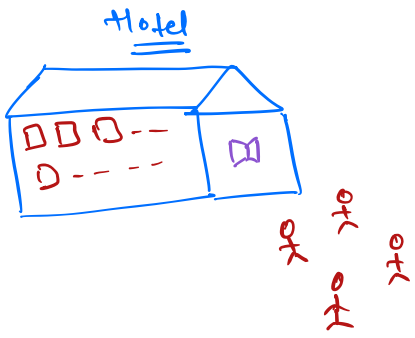
Hashing !!

- Why Hashmap ?
- How the Hashmap is implemented ? *
- Pattern of problems for Hashmap.

Rahul ☹ Aditya ☹

⇒ Hotel with 1000 rooms.



Roomno	free
101	f
102	0
103	≠0
⋮	⋮

→ 1000 entries.

a[555]

int arr[1001];

arr[i] = 0 ⇒ free
1 ⇒ Occupied

Check, update → O(1)

⇒ * Numerologist :- [1 - 10^9] ⇒ 1000005

{ 37, 10000, 97, 10004, ... } 1000000

→ 1000 random numbers ∈ [1, 10^9]

int arr[10^9 + 1]

arr[10004] ?

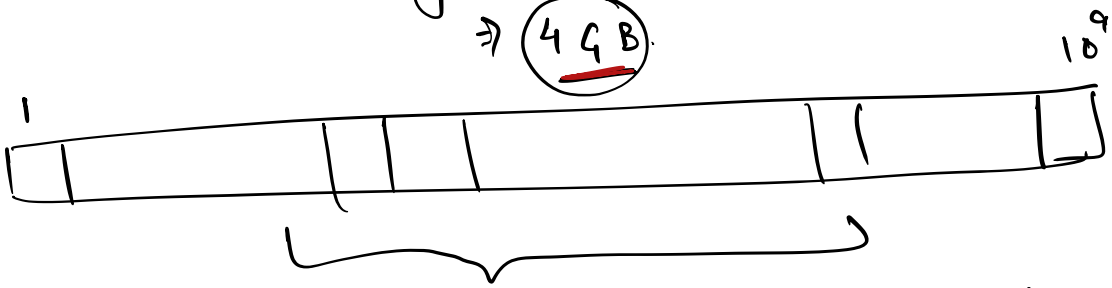
↓

x

1 integer ⇒ 4B

10^9 size array ⇒ 4×10^9 B.

⇒ (4GB)



1000 indexes will be used.

Space wasted = $10^9 - 10^3$ ≃ 10^9

1000000000
−1000
≃ 10^9

Requirements:

- No space wasted.
- Easier access.
  - Check → O(1)
  - Update → O(1)

⇒ Average TC : O(1)

⇒ **HashMap** Data Structure

<key, value>

K : V

1 : 4s
2 : 2s
3 : 1s
4 : 1s.

avg :- 2 secs

HashMap < Hotel room no, Availability >

Key — Value.

⇒ 1000 entries in HashMap.

HashMap :-

Check
Update } O(1)

#

Hotel Room No ⇒ Room name.

= HashMap < String, Boolean >.

String → Room name

Boolean → free / occupied.

⇒ HashMap < Key, Value >.

1) HashMap for country & its population.

HashMap < String, Long >.

String → Country

Long → Population.

2) Country name, name of states.

HashMap < String, List < String > >

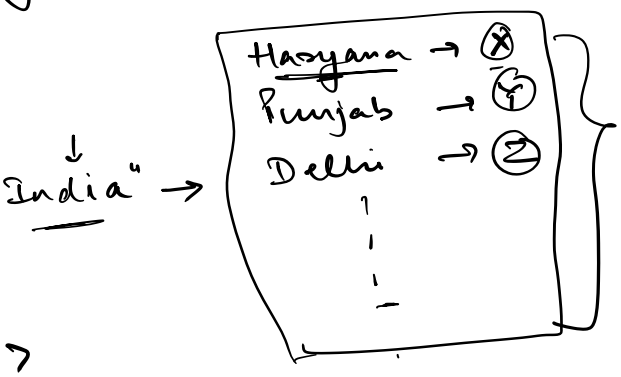
India → { ---, ---, ---- }

List of String.

3) Country Name, population of each state.

⇒ HashMap < String , HashMap < String, Long >>

HashMap → unordered_map.



Java → HashMap<K, V>
C++ → unordered_map<K, V>.
Python → dict
Ruby/JS → map.

www.google.com

⇒ HashMap functions :-

- insert (key, value) → O(1)
- get (key) ⇒ value → O(1)
- size() ⇒ No. of key's.
- delete (key) → O(1)

HashMap have unique keys.

Q. Given an array of size 'N', and 'Q' queries.
Every query → integer 'x'.
return frequency of 'x' in the array.

A : {2, 6, 3, 8, 2, 8, 5, 8}

Q : 2 → 2
8 → 3
3 → 1
5 → 1
⋮

Brute force :-

for every query : → Q
traverse array and → N.
count of 'x'.

⇒ TC : O(Q·N)
SC : O(1)

Approach #2 :

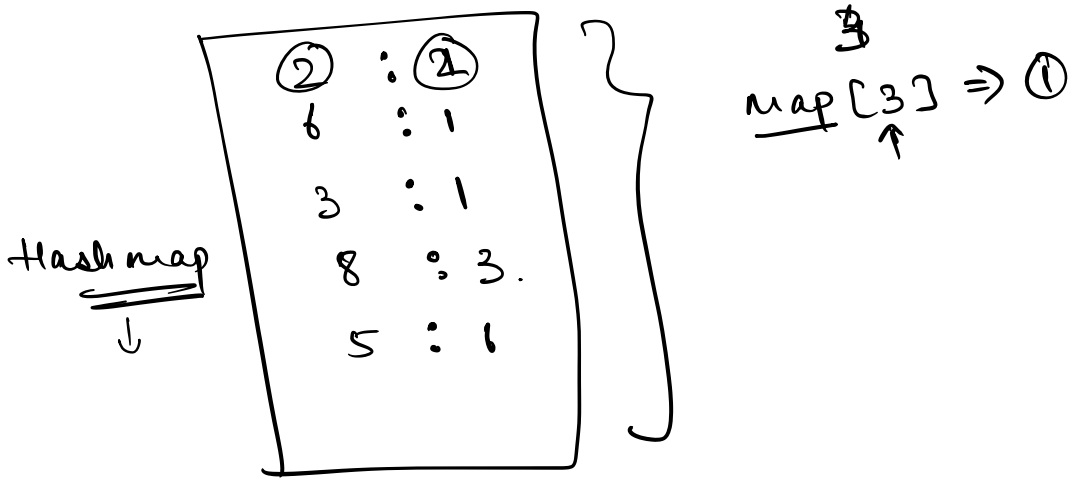
1) Create Hashmap of element & its frequency. → O(N)
2) Check Hashmap of each query.

2	2
6	1
3	1
8	1
⋮	

element ↑ freq.

unordered_map <int, int> mp;

: {2, 6, 3, 8, 2, 8, 5, 8}



TC: O(N + Q)

SC: O(N)

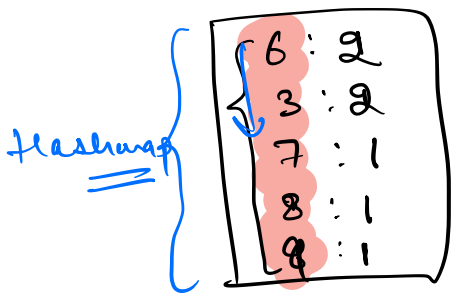
Q: Given an array, count the no. of distinct elements in the array.

A: [7, 3, 2, 1, 3, 7, 0] → 5.

Quiz A: 6, 3, 7, 3, 8, 6, 9 ⇒ 5.

Approach:- → Build frequency hashmap → O(N)

→ Size of the Hashmap →

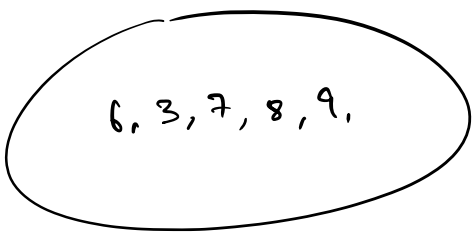


→ Return the size of Hashmap.

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

* ⇒ Sets / HashSet :- DS to have unique value.

{6, 3, 7, 3, 8, 9, 7}

Set 

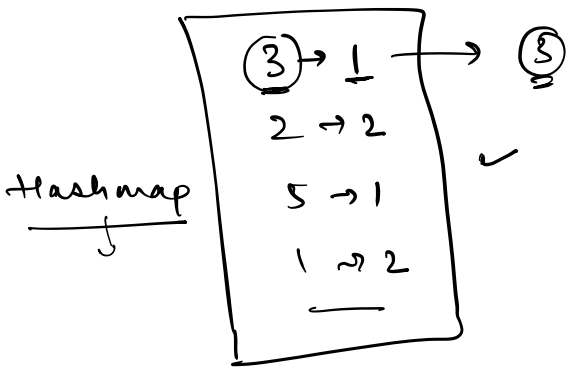
⇒ Hashmap doesn't preserve order.

Q. Amazon Adobe GS.

Given an array of size N, find the first Non repeating element in the array.

A : 8, 2, 8, 3, 1, 2, 6, 5.

Quiz {1, 2, 3, 1, 2, 5} → 3



⇒ Approach :-

1. Build the hash map for element & frequency. → O(N)
2. Traverse the array and find the first element with freq == 1. → O(N)

```
for(i=0; i<n; i++){          → O(1)
    if(map[arr[i]] == 1)
        return arr[i];
}
```

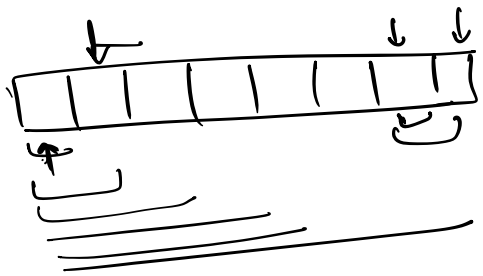
TC : O(N+N) → O(N)

SC : O(N)

Q: Given an array of size N, check if there exists a subarray with sum = 0.

Amazon, FlipKart, Adobe, Ola, Uber, MMT ...

A: {2, 2, 1, -3, 4, 3, 1, -2, -3} → TRUE.



No. of subarrays ⇒ $N + (N-1) + (N-2) + \dots 2 + 1$

⇒ $N * (N+1) / 2$ ⇒ N^2

Brute force :-

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

TC : $O(N^3)$

SC : $O(1)$.

```
for( i=0; i<N; i++){
    for(j=i; j<N; j++){
        sum = PS[j] - PS[i-1]; → O(1)
        sum(i,j)
        if( sum==0) return true;
    }
}
return false;
```

TC : $O(N^2)$

SC : $O(N)$

A : {2, 2, 1, -3, 4, 3, 1, -2, -3}

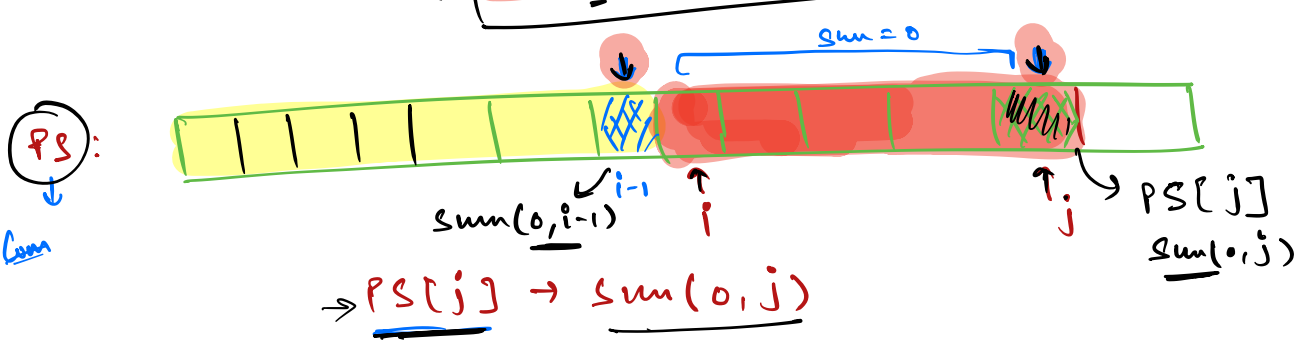
PS : {2, 4, 5, 2, 6, 9, 10, 8, 5}.

$sum(i,j) = PS[j] - PS[i-1]$

⇒ Lets Assume $sum(i,j) = 0$.

$PS[j] - PS[i-1] = 0$

⇒ $PS[j] = PS[i-1]$ ⇒



→ $PS[j] \rightarrow sum(0,j)$

→ PS[i-1] = sum(0, i-1)

sum(i, j) → 0 ≡ [i, j]

↓

→ sum(0, j) − sum(0, i-1)

```
if (PS[j] == PS[i-1]) {
        sum(i, j) = 0
}
```

→ Build PS array.

→ Check if there's any duplicate pair.

↳ Set / HashSet.

↓

Set.insert(arr[i])

↳ O(1)

TC : O(N)

SC : O(N)

(I) ✓ A : [1, 2, 0, 8].

PS : [1, 3, 3, 8]

(II) X A : [0, 3, 2, 8]

PS : [0, 3, 5, 13].

↳ No dup.

Edge Case

→ If there's a ZERO in array then always return true.

(iii) A : [3, -1, -2, 4]

PS : [3, 2, 0, 4]

sum(0,2) = 0.
3 - 1 - 2 = 0.

sum(i,j) = PS[j] - PS[i-1] = 0.

i = 0

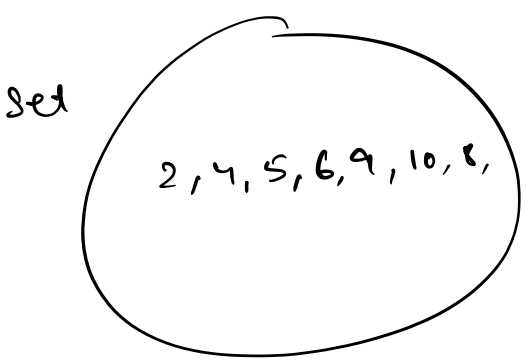
sum(0, i) = PS[i]

→ If PS array contains ZERO then return TRUE.

⇒

A : {2, 2, 1, -3, 4, 3, 1, -2, -3}

PS : {2, 4, 5, (2), 6, 9, 10, 8, (5)}.

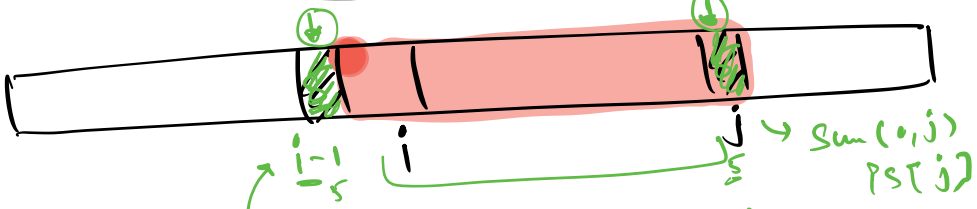



```
for(i=0; i<N; i++){
    set.insert(PS[i]);
}
if(set.size() != N)
    return true;
else
    return false;
```

⇒ sum(i,j) = PS[j] - PS[i-1].

0 = PS[j] - PS[i-1]

PS[j] = PS[i-1] ✓



PS

↓

duplicate values.

PS[i-1] → sum(0, i-1)

PS[j] → sum(0, j)

PS[j] = PS[i-1].

↳ sum(i,j) = 0.