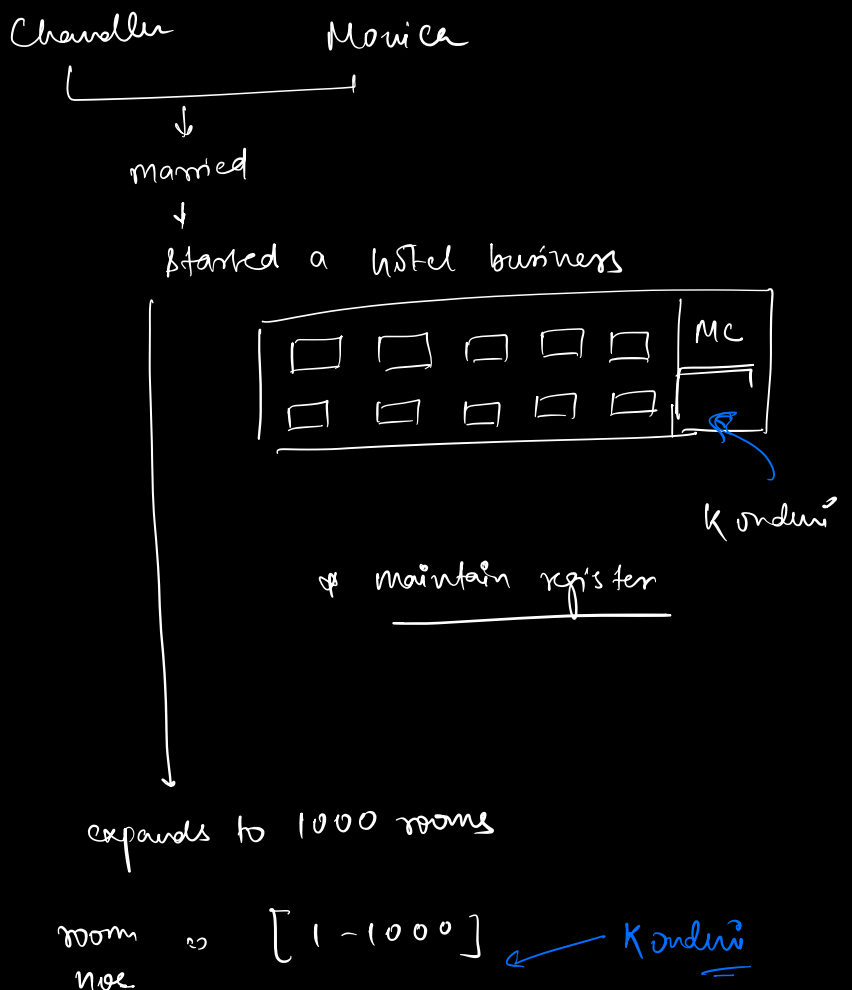
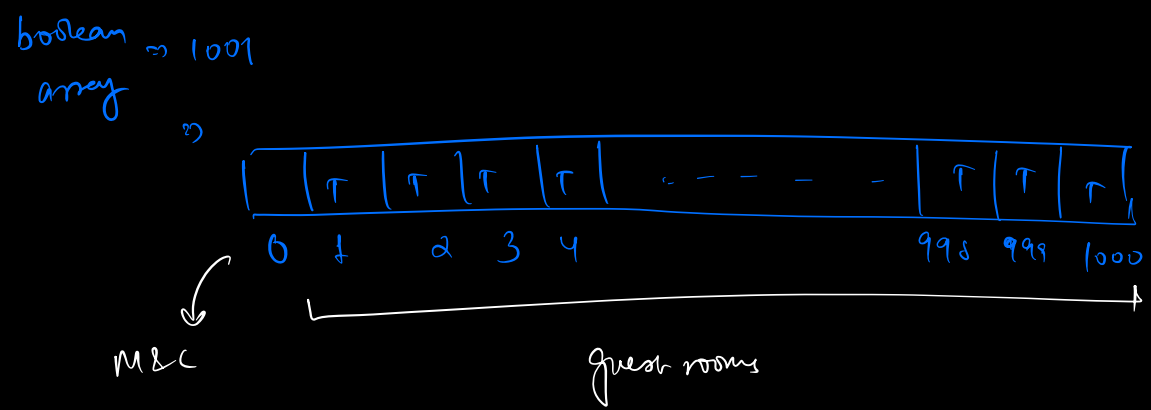




anyone books a room ⇒ i ⇒ arr[i] = false

if to check room i is available ⇒ arr[i] ⇒ 0(1)

pandemic

↓

Phone

↓

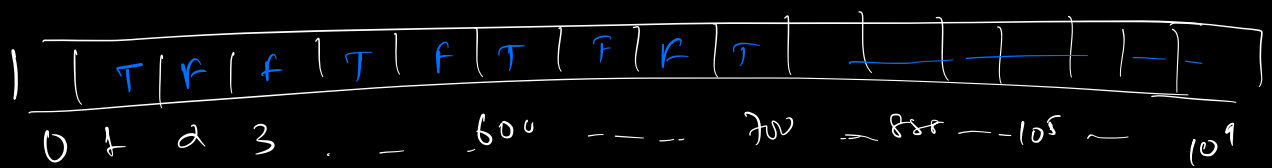
all your room nos. to lucky numbers

↓

Lucky nos. [1-10^9]

↓

10^9 el



arr[i] ⇒ check

1000 rooms ⇒ 10^9

Hashmap

↳ Optimised for space

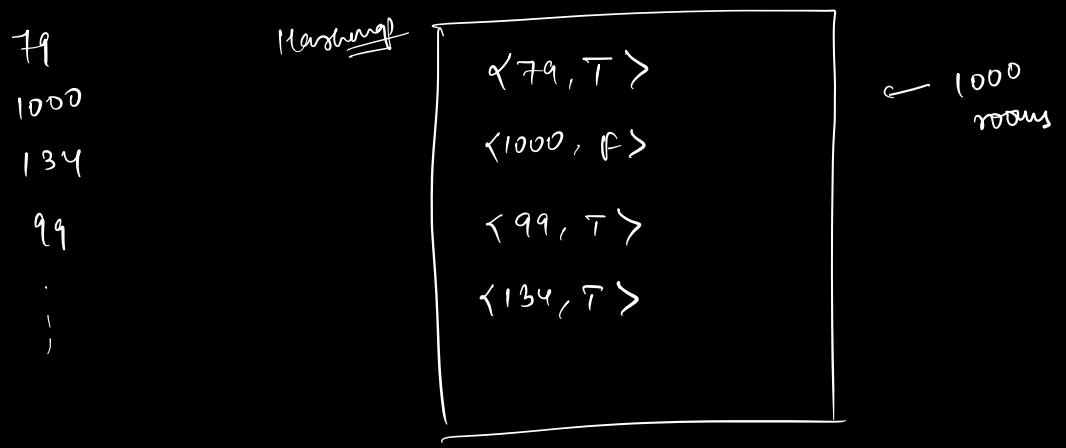
↳ O(1)

// Hashmap < key, value >

↳ Stores data in form of key-value pairs

↳ all keys are unique

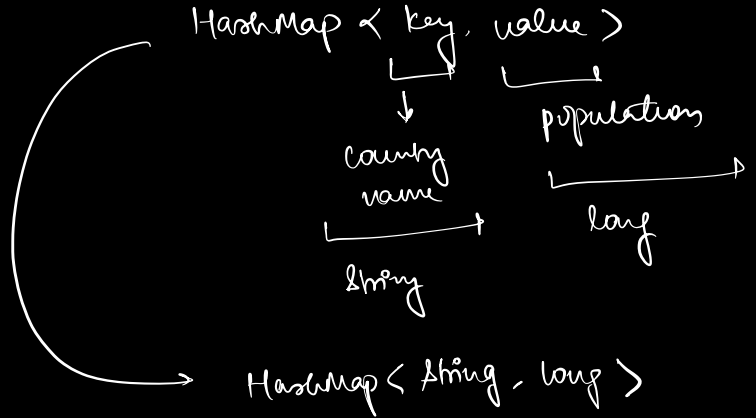
↳ value can be duplicate



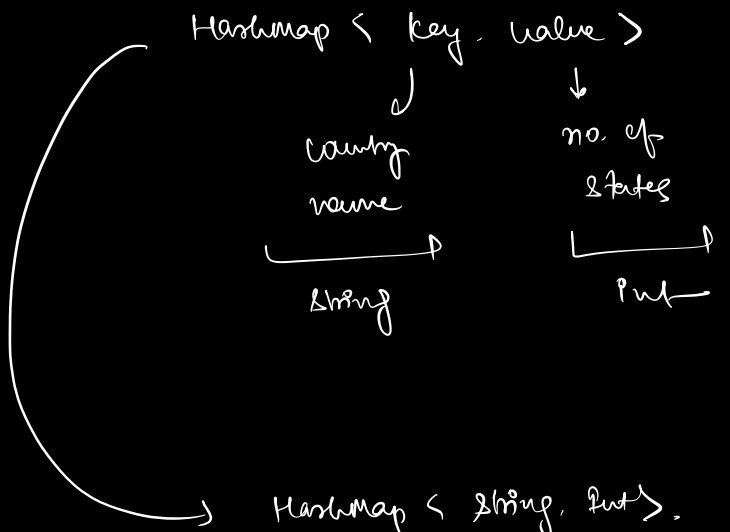
→ stores k-v pairs → Hashmap

→ store keys → HashSet
↳ unique

Q1. Store population of every country :

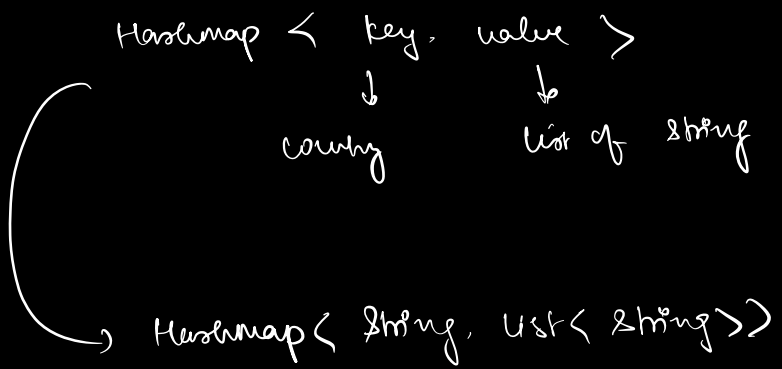


Q 2. Store no. of states of a given country:

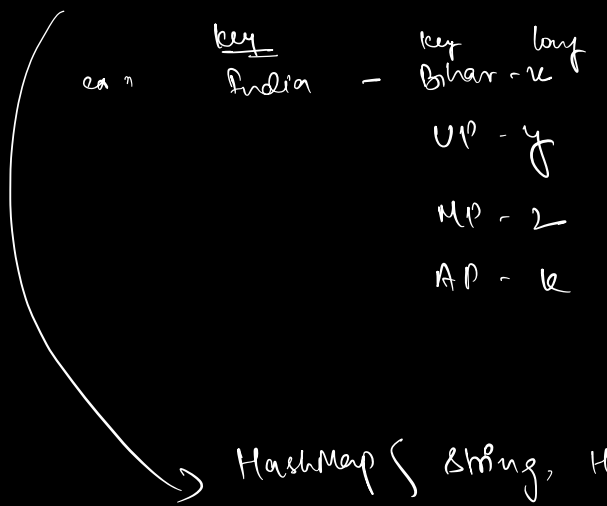


Q 3. For every country, we want to store all State names

India → Bihar, UP, WB - - -



Q 4. For every country, store the population of each state



* value can be anything.

* key → String, int, float, long

any primitive type including String

// Hashmap

functions

O(1) → avg. or amortised time

- insert(key, value) → inserting k-v pair in hashmap
- search(key) - returns value for that key
- remove(key) - remove k-v pair for that key
- update(key, newValue) → update the value for that key
- size() → returns no. of k-v pairs in hashmap.

hashmap ⇒ 10^4 k-v pairs

search (59) → O(1)

Roll No	Marks
4	96
5	98
1	99
2	97
.	
104	56

// Hashset <key>

↓

unique keys

O(1)
- insert (key)
- search (key) → whether a key exists in set or not
- remove (key)
- size()

	Java	C++	Python	JS	C#
Hashmap -	Hashmap	unordered_map	dictionary	map	Dictionary
Hashset -	Hashset	unordered_set	set	set	set

Q: Find frequency of numbers:-

N = 10, arr[10] ⇒ {2, 6, 3, 8, 2, 8, 2, 3, 8}

give op as per queries:

Q = [2, 8, 3, 5]

↓ ↓ ↓ ↓

3 3 2 0

Brute force

for every query element,

search entire, and print freq

↓

TC ⇒ O(Q*N)

⇒ O(NQ)

SC ⇒ O(1)

Hashmap < key, value > → Hashmap < INT, INT >

↓ ↓

all distinct array element — freq of element

2, 6, 3, 8, 2, 5, 2, 3, 5

Q = 2, 8, 3, 5

0

<2, 3>
<6, 1>
<3, 2>
<8, 3>

pseudo

```
HashMap <INT, INT> hm;

for( i=0; i<N; i++) {                          → check
    if( arr[i] is present in hm) {
        // increase the value for arr[i] by 1   → update
    else {
        insert < arr[i], 1>                    → insert
    }
}
```

O(N)

```
for( i=0, i< Q.length; i++ ) {
    if( Q[i] is present in hm ) {
        return value of Q[i].
    else {
        return 0.
    }
```

O(Q)

TC ⇒ O(N+Q)

SC ⇒ O(N)

hashmap

Q2. Find the first non-repeating element.

arr[6] ⇒ [1 2 3 1 2 5]

O/P = 3.

arr[8] ⇒ [4 3 3 2 5 6 4 5]

O/P = 2

arr[7] = [2 6 8 4 7 2 9]

O/P = 6

Soln 1

1) Create a hm

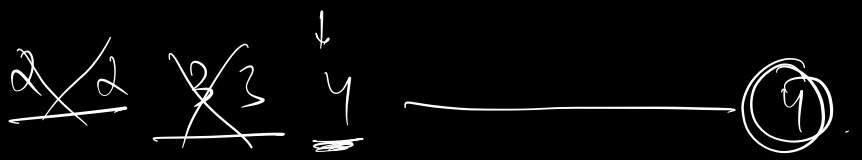
2) update the freq of each element

3) Iterate the hm, return 1st element with freq = 1.

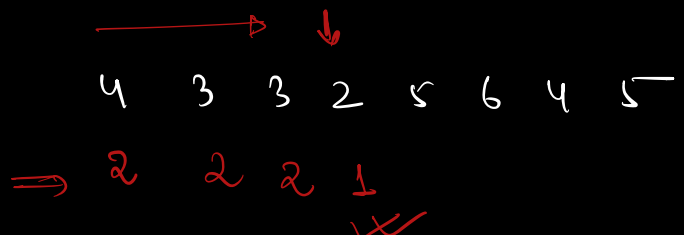
* hashmap | hashset, they never store data in order.

Soln-2

keep storing everything in map, and return if it was duplicate



* create a hm
* update frequency for each element
* iterate the array, and get freq for each element, return the 1st element with freq 1.



4 → 2
6 → 1
5 → 2
2 → 1
3 → 2

⇒ Create a hm containing freq ⇒ hm

```
for(i=0; i<N; i++){
    if( hm[arr[i]] == 1)
        return arr[i]
}
```

O(N)

TC ⇒ O(N) + O(N)

≈ O(N)

SC ⇒ O(N)

Q3. Given N arr elements, find no. of distinct elements.

different

eg ⇒ arr[5] ⇒ {3, 5, 6, 5, 4} ⇒ 4.

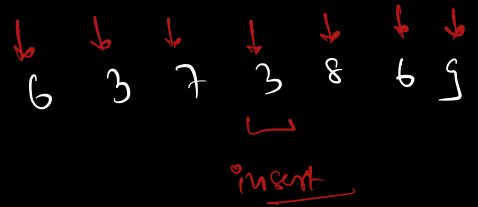
arr[7] ⇒ {6 ✓ 3 ✓ 7 ✓ 3 ✗ 8 ✓ 6 ✗ 9 ✓} ⇒ 5.

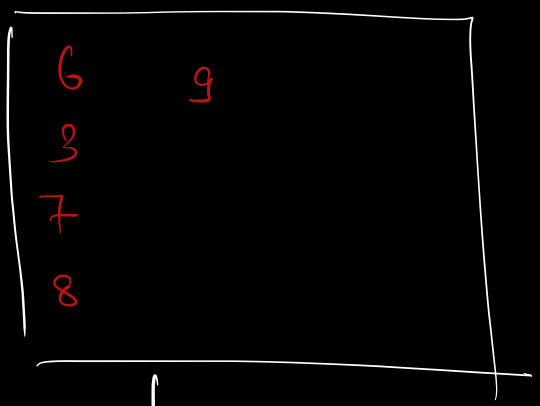
SOL Create a freq hm ⇒ return size

{6-2, 3-2, 7-1, 8-1, 9-1} ⇒ 5.

Soln

Hashset → duplicate keys are not possible





hashset.insert(3)

```
}
// hashset<INT> hs
for( i=0; i<N; i++){
    hs.insert( arr[i] )
}
return hs.size()
}
```

return hashset size

TC → O(N)

SC → O(N)

Q 4. Given N array elements, check if all elements are distinct or not

soln → insert in hashset → hs.size() → distinct elements equal to N (true)

```
// hashset<INT> hs
for( i=0; i<N; i++){
    hs.insert( arr[i] )
}
return hs.size() == N.
}
```

TC ⇒ O(N)
SC ⇒ O(N)

Q5. Given an array[N], return true if there exists any subarray sum == 0, else false.

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	2	2	1	-3	4	3	1	-2	-3	2

= = 0

O/P = true

brute force

find all possible subarrays
, and find sum of subarrays

if sum == 0 return true.

TC ⇒ O(N^3)

SC ⇒ O(1)

	0	1	2	3	4	5	6	7	8	9
arr[] ⇒	2	2	1	-3	4	3	1	-2	-3	2
pfsum ⇒	2	4	5	2	6	9	10	8	5	7

sum [3 1] ⇒ pf[3] - pf[0]

⇒ 2 - 2

⇒ 0

pf[3] - pf[0] = 0

⇒ pf[3] = pf[0]

* if any two pfsums are same
there exists a subarray whose sum = 0

Steps

i) Calculate pfsum

ii) Store pfsum, and keep checking if any no. is repeating

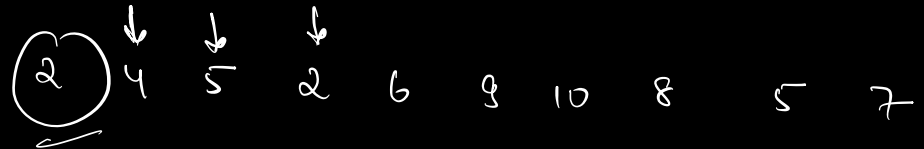
if repeats ⇒ true

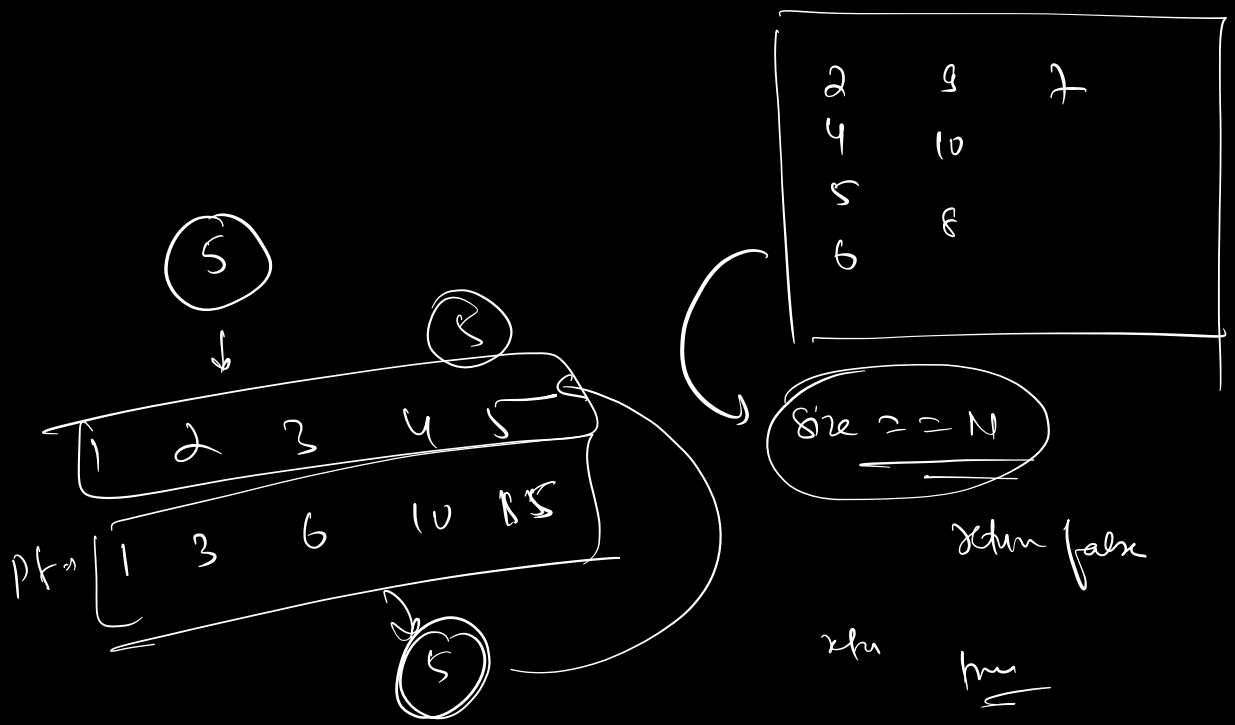
else ⇒ false

TC ⇒	O(N+N)	= O(N)
SC ⇒	O(N+N)	= O(N)

* TODO

[Solve this question in a single loop]





doubt

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

Hashmap →

4 → [0, 6]

3 → [1, 2]

2 → [3, 4]

5 → [4, 7]

6 → [5, 6]

0 sum subarray

0	1	2
0		

Pf → 1 1 2