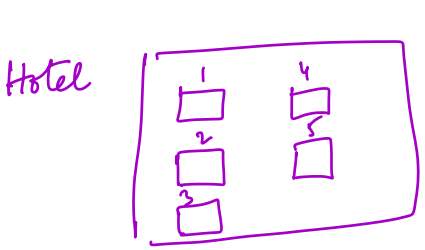


Room nr.	Occupied
1	✓
2	✗
3	✗
4	✓
5	✓

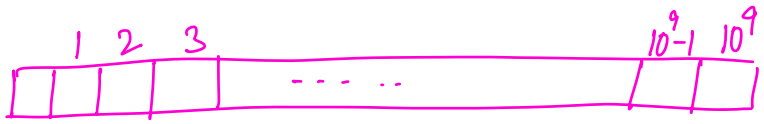
1000 rooms →

covid → drop in business

1000 room nr.

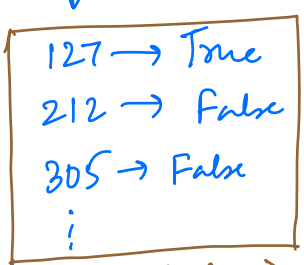
127
212
305
⋮
10321
⋮

each room nr. $\in [1, 10^9]$



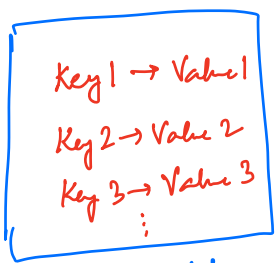
Problem → We're occupying space for 10^9 elements, while we're using only 1000 of them.

Solution → HashMap.



<Integer, Boolean>
<int, bool>

<key, value>



HashMap → For one key, the HM stores only 1 value. (almost)

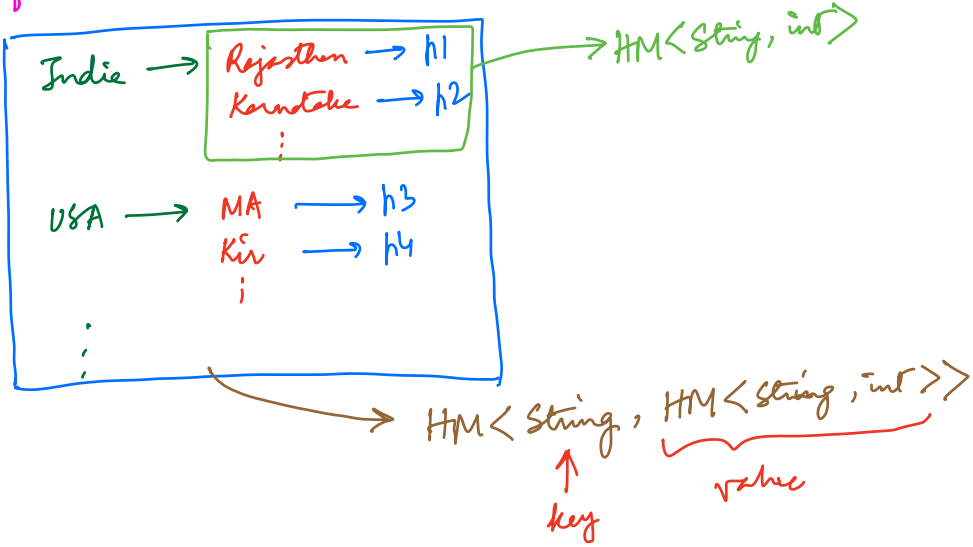
HashMap → T.C. of search is O(1) on average, S.C. is O(n).
Key must be unique
Value can be anything

1) Population of each county.
HashMap<String, Integer/Long> popCounty

2) No. of states of every county
HashMap<String, Integer> noOfStatesPerCounty

3) Names of states of every county
HashMap<String, List<String>> statesPerCounty

4) Population of each state in each county.



HashMap<String, HashMap<String, Integer>> popOfStatesPerCounty;

HashSet → Helps in string keys.
→ Key must be unique
→ Like HashMap, we can search a key in O(1) time in HashSet.

HashMap :-

Java

Insert (key, value) → put (key, value)

size ( ) → size ( )

Delete (key) → remove (key) → remove the key

Update (key, value) → put / replace (key, value)

put / replace → adds (key, value)

Search (key) → get (key) → returns the value

containsKey (key) → checks whether the key is present or not.

→ O(1) T.C

HashSet

Java

Insert (key) → add (key)

size ( ) → size ( )

Delete (key) → remove (key)

Search (key) → contains (key) T/F

→ O(1) T.C

Adding n (key, value) pairs in a HashMap takes O(n) T.C , O(n) S.C

	Java	C++	Python	JS	C#
HashMap	HashMap	unordered_map	dict	map	dict
HashSet	HashSet	unordered_set	set	set	HashSet

Q1) Given n elements and q queries, find the frequency of each element in the queries.

n=10

2 6 3 8 2 8 2 3 8 10

q=4

2	8	3	5
↓	↓	↓	↓
3	3	2	0

Brute force → For each query, iterate the array and count.

O(q*n) T.C.

O(1) S.C.

Optimal approach → Store <Element, Frequency> in HM.

2 6 3 8 2 8 2 3 8 10

2 → ~~1~~ ~~2~~ 3

6 → 1

3 → ~~1~~ 2

8 → ~~1~~ ~~2~~ 3

10 → 1

2	8	3	5
↓	↓	↓	↓
3	3	2	0

```
fn freqQuery(A[], Q[]){
    HashMap<int, int> hm;
    q = len(Q)
    n = len(A)
    fn(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)
        }
    }
    fn(i -> 0 to q-1){
        if(hm.containsKey(Q[i])){
            Print(hm.get(Q[i]))
        }
        else{
            Print(0)
        }
    }
}
```

O(n) TC

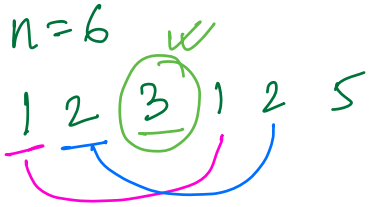
O(q) TC

O(n+q) TC.

O(n) S.C

Q2) given n elements, find the first non-repeating element.

n = 6



2 → 2
1 → 2
5 → 1
3 → 1

HM.

```
fn firstNonRepeating(A[]){
    HashMap<int, int> hm;
    n = len(A)
    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 n-1){
        if(hm.get(A[i])==1)
            return A[i]
    }
    return -1
}
```

O(n) T.C
O(n) S.C

[Break till 10:41 PM]

Q3) given n elements, find the count of distinct elements.

n = 5

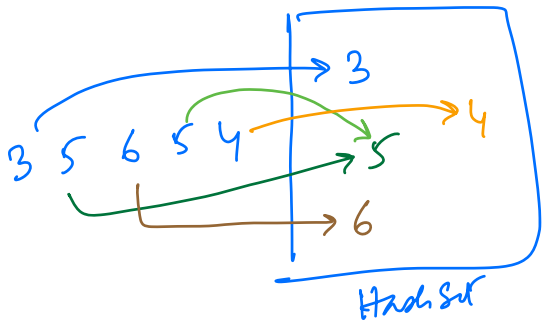
3 5 6 5 4

ans = 4

n = 3

4 4 4

ans = 1




```
fn countDistinct(A[]){
    HashSet<int> set
    n = len(A)
    for(i -> 0 to n-1){
        set.add(A[i])
    }
    return set.size()
}
```

O(n) TC

O(n) S.C

Q 4) given n elements, check if there exists a subarray with sum equal to 0.

n = 10

2	2	1	-3	4	3	1	-2	-3	2

Ans → True.

Brute Force → Find the sum of each of $O(n^2)$ subarrays

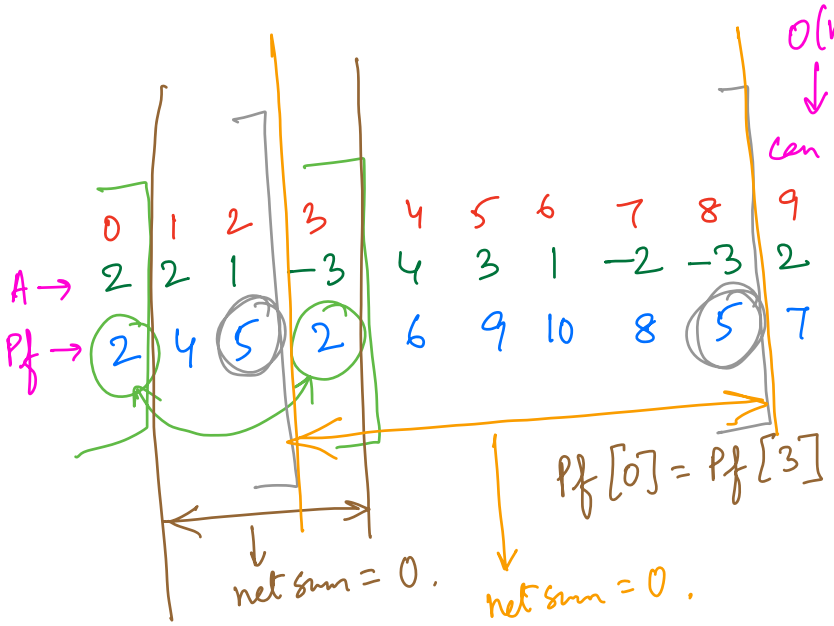
$O(n^3)$ TC

↓ Prefix Sum / Carry Forward Technique.

$O(n^2)$

↓

can we optimise further?



Pf[0] = Pf[3]

Pf[2] = Pf[8]

⇒ Pf[2] = Pf[2] + sum(A[3] ~ A[8])

⇒ sum(A[3] ~ A[8]) = 0.

Pf[0] = Pf[3]

↓

Pf[0] = [Pf[0] + sum(A[1] ~ A[3])]

⇒ sum(A[1] ~ A[3]) = 0

Pf[l] = Pf[r] ⇒ sum(A[l+1] ~ A[r]) = 0

Pf[l] = 0 ⇒ sum(A[0] ~ A[l]) = 0.

sum = 0

A →	-2	-1	3	4
Pf → 0	-2	-3	0	4

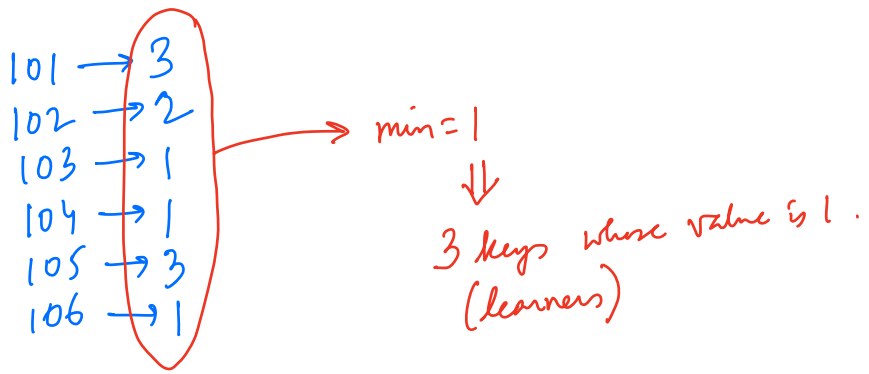
```
fn findZeroSubarraySum(A[]){
    HashSet<int> set
    n = len(A)
    sum = 0
    for(i -> 0 to n-1){
        sum += A[i]
        if(sum == 0 || set.contains(sum))
            return true

        set.add(sum)
    }
    return false
}
```

O(n) T.C

O(n) S.C

Q5) You have the list of IDs where each ID (integer) represents a learner who tried a contest. A learner's ID shows up x times if they try x contests. Write a program that looks at this list and finds out how many students have tried the least nr. of contests.

l_ids = {101, 102, 103, 101, 102, 101, 104, 105, 106, 105, 105}



- store all elems with freq. in HM (IDs)
- iterate the HM/array and find min value
- iterate the HM and find how many keys have the min value.

} together

```
fn findLeast(A[]){
    HashMap<int, int> hm;
    n = len(A)
    fn (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)
        }
    }
    int min = Integer.MAX_VALUE;
    int cnt = 0
    fn (int x : hm.keySet()){
        if(hm.get(x) == min)
            cnt ++
        else if(hm.get(x) < min){
            min = hm.get(x)
            cnt = 1
        }
    }
    return cnt
}
```

101 → 3
102 → 2
103 → 1
104 → 1
105 → 3
106 → 1

min = $2*10^9$
cnt = 0

3 < min

min = 3, cnt = 1
min = 2, cnt = 1

1 < 2 → min = 1, cnt = 1

1 == 1 → min = 1, cnt = 2

1 == 1 → " , cnt = 3