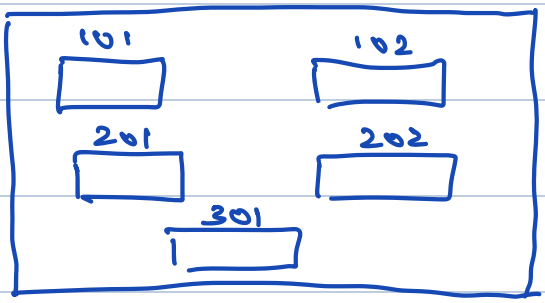
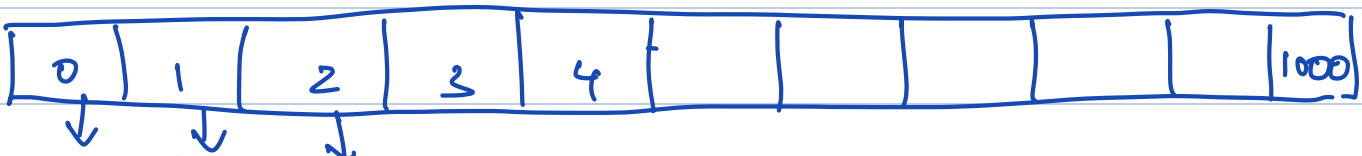
Hashing 1



Receptionist → 101 → Occupied
102 → Occupied
201 → Not occupied
...

For 1000 rooms,



Only 2 occupied → 3 & 1000 (Space Wastage)
No custom room numbers.

Hashmap: Key → Value
{ 3 → True
1000 → True }

Some key points about Hashmaps

- Search in an Hashmap happens in O(1) time
- Key must be unique. ↳ Key
- Value can be anything
- Only immutable objects are accepted as keys.

Python → Dictionary
JAVA → Hashmap
C++ → Unordered Map

```
hm = {}
Hashmap<KeyType, ValType>
```

Quiz 1: Population of every country.
String → 10^9 (1 Billion).
↓
Long

Quiz 2: Every countries → No. of states (50)
↓ String ↓ Integer

Quiz 3: Every country → Names of states
↓ String ↓ List(String)

Quiz 4: Every country → Each state → Population
↓ String ↓ String ↓ Integer
Hashmap.

Hashset: Similar to Hashmap, but only holds keys.

- Search → O(1)
- Keys should be unique
- Any datatype can be stored.
- Both in HM & HS → order is not preserved

Methods of Hashmap	JAVA	Python
- Insert (key, value)	→ put (k, v)	hm[k] = v
- Size	→ size()	len(hm)
- Delete (key)	→ remove(key)	del hm[k]
- Update (key, value)	→ put (k, v)	hm[k] = v
- Search (key)	→ get(k)	hm[k]

Methods of Hashset	JAVA	Python
- Insert (key)	add (k)	add (k).
- Size	size()	len(hs)
- Delete (key)	remove(k)	del hs[k]
- Search (key)	contains(k)	in

Question: Given N elements and Q queries, find the frequency of elements in a query.

arr = [2, 6, 3, 8, 2, 8, 2, 3, 8, 10, 6]

Queries O/P

2 → 3

8 → 3

6 → 2

Brute Force Approach

- For each query, iterate through the array and count occurrences of the element.

T.C = O(N*Q) S.C = O(1)

Optimised

- Iterate over array, count the occurrence of an element and store it an hashmap.
- For each query, retrieve the value in the hashmap

T.C = O(N+Q) S.C = O(N)

```
func getFreq(int[] arr, int[] queries){
    freq_map = {}
    for(int i = 0; i < arr.length; i++){
        if(freq_map.search(arr[i])){
            freq_map[arr[i]]++;
        }
        else{
            freq_map.insert(arr[i], 1);
        }
    }
    ans = []
    for(int i=0; i<queries.length; i++){
        ele = queries[i];
        ans.add(freq_map[ele]);
    }
    return ans;
}
```

Question: Given N elements, find the first non-repeating number

ex: arr = [1, 2, 3, 1, 2, 5] → O/P = 3

ex: arr = [4, 3, 3, 2, 5, 6, 4, 5] → O/P = 2

Idea:

- Iterate over elements in array and create frequency map.
- Go through all keys in frequency map and return the key whose value is 1

arr = [1, 2, 3, 1, 2, 5]

hm = { 1 → 2
2 → 2
5 → 1
3 → 1 }

O/P = 5 ✗

Flaw: Order of insertion is not maintained in hashmaps.

Idea 2:

- Instead of iterating over keys in hashmap, iterate over the array itself.
- Return the element whose freq == 1.

```
func firstNonRepeating(int[] arr){
    hm = {}
    for(int i=0; i<arr.length; i++){
        if(hm.search(arr[i])) hm[arr[i]]++;
        else hm.insert(arr[i], 1);
    }
    for(int i=0; i<arr.length; i++){
        if(hm[arr[i]] == 1) return arr[i];
    }
    return -1;
}
```

T.C = O(N) S.C = O(N)

Question: Given an array of N elements, find the number of unique elements

ex: arr = [3 , 5 , 6 , 5 , 4] → O/P = 4

ex: arr = [3 , 3 , 3] → O/P = 1

Idea:

- Store elements in hashset
- Final hashset will have unique elements.

```
func countUnique (int[] arr) {
    hs = {}  // hashset
    for (int i = 0; i < arr.length ; i++) {
        hs.insert (arr[i]);
    }
    return hs.size();
}
```

T.C = O(N) S.C = O(N)

Question: Subarray sum zero

Given an array of N elements, check if there exists a subarray with a sum equal to 0.

arr = [2, 2, 1, -3, 4, 3, 1, -2, -3, 2]

O/P = True (2, 1, -3)

Brute Force approach:

- Go through all subarray sums using carry forward and check if sum == 0.

T.C = $O(N^2)$ S.C = $O(1)$

Idea:

arr = [2, 2, 1, -3, 4, 3, 1, -2, -3, 2]

PFSum = [2, 4, 5, 2, 6, 9, 10, 8, 5, 7]

Jan 1st 2024 → 5,00,000 INR

[Jan, Feb, March] → Subarray. => Sum = 0

April 1st 2024 → 5,00,000 INR

Subarray sum = PF[j] - PF[i-1]

⇓

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

PF[j] = PF[i-1] → Repeating PF Sum.

for i=0 ⇒ PF[j] = 0

```
func subarraySumZero(int[] arr){
    PF[N]
    PF[0] = arr[0]
    for(int i=1; i<arr.length; i++){
        PF[i] = arr[i] + PF[i-1];
    }
    hm = {}
    for(int i=0; i<PF.length; i++){
        if(hm.search(PF[i])) return True;
        else hm.insert(PF[i], 1);
    }
    return False;
}
```

T.C = O(N) S.C = O(N)

Edge case: arr = [2, 3, -5, 4, 6]

PFSum = [2, 5, 0, 4, 10]

⇓

PF[j]

Way 1: While calculating PF Sum check if PF[i]==0
Then return true.

Way 2:

1) Add 0 to hm initially
2) Both loops run through same elements
So combine them in one.
3) PF value is not needed, so carry forward.

```
func subarraySumZero(int[] arr){
    PFSum = arr[0];
    if(PFSum==0) return true;
    hm = {0:1, PFSum:1};
    for(int i=1; i<arr.length; i++){
        PFSum = PFSum + arr[i];
        if(hm.search(PFSum)) return true;
        else hm.insert(PFSum, 1);
    }
    return false;
}
```

T.C = O(N)
S.C = O(N)

arr = [0, 1, 2, 3, 4]
PFSum = 0, 1, 3, 6, 10
loop starts

arr = [1, -2, 2, 3]
PFSum = 1, -1, 1, 4
loop starts
hm = {0:1, 1:1, -1:1}

To get subarray with sum = 0

```
func subarraySumZero(int[] arr){
    PFSum = arr[0];
    if(PFSum == 0) return [0,0];
    hm = { 0 : -1, PFSum : 0 };
    for(int i = 1 ; i < arr.length; i++){
        PFSum = PFSum + arr[i];
        if(hm.search(PFSum))
            return [hm[PFSum]+1, i];
        else hm.insert(PFSum, i);
    }
    return [] ; }
```

arr = [2 , 3 , -5 , 4 , 6]

PFSum = [2 , 5 , 0 , 4 , 10]

PF[j]

hm = { 0:-1, 2:0, 5:1,

arr = [2 , 2 , 1 , -3 , 4 , 3 , 1 , -2 , -3 , 2]

PFSum = [2 , 4 , 5 , 2 , 6 , 9 , 10 , 8 , 5 , 7]

hm = { 0:-1, 2:0, 4:1, 5:2

Question: Given an array of integers, find the count of least repeating integers.

arr = [101, 102, 103, 101, 102, 101, 104, 105, 106, 105, 105]

Least Repeating = [103, 104, 106]

O/P = 3

Approach :- Use freq_map. Keep storing min freq
- Go through freq_map again look for elements with min_freq.

```
func countMinFreq(int[] arr) {
    min_freq = INT_MAX;
    hm = {}
    for (int i=0; i<arr.length; i++) {
        if(hm.search(arr[i])) {
            hm[arr[i]]++;
        }
        else {
            hm.insert(arr[i], 1);
        }
    }
```

```
        ans_count = 0;
        for ( ele : hm ){
            min_freq = min (min_freq , hm[ele]);
        }
        for (ele : hm){
            if (hm [ele] == min_freq)  ans_count++;
        }
        return  ans_count;
    }
```

arr = [101, 102, 103, 101, 102, 101, 104, 105, 106, 105, 105]

min_freq = 1 hm = {101 : 3, 102 : 2, 103 : 1

ans_count = 1 + 3 104 : 1 ; 105 : 3 , 106 : 1}

Doubts

Mutable → Values in the reference can be changed

→ ArrayList , hashset , hashmap.

Immutable →

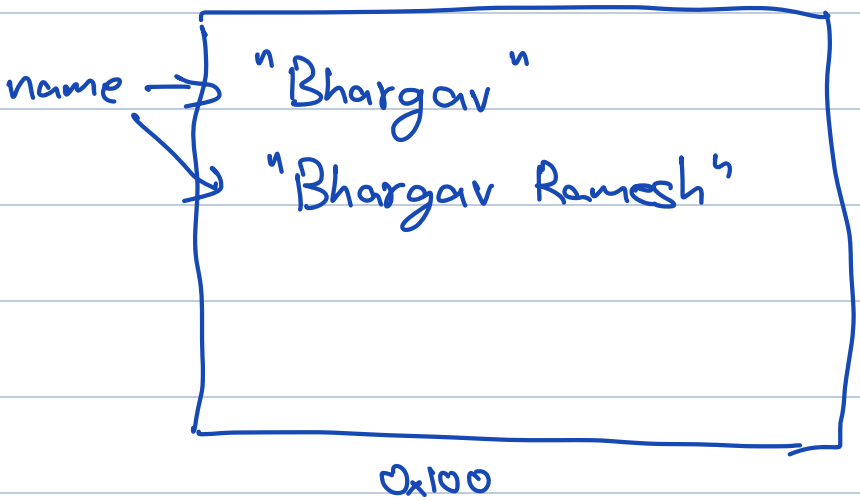
→ int, String , double

hm = { [1, 2, 3] → 0 ⇒ { [1, 2, 3, 4] → 0 ,
[1, 2, 3, 4] → 1 } [1, 2, 3, 4] → 1 }

{1, 2, 3} → 0
{1, 2, 5} → 1

int a = 5

a = 10



String name = "Bhargav"

name += " Ramesh"

int a = 5;