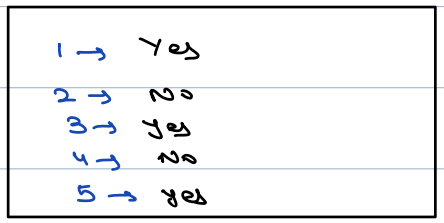
Hotel → Leela Palace .



→ register .

1 years

1000 rooms .

boolean Array → idx denotes the room no .

↳ True → occupied

↳ false → Empty .

[1 - 10^9]

↓

array of len [10^9+1]

Issue :- Space wastage .

Solution → Hashmap . → T.C → search O(1) .

↓

< key, value >

3 → occupied

1009 → occupied

99 → occupied .

- **In HashMap, T.C of search is O(1) time and S.C is O(N)**
- Key must be unique
- Value can be anything

Note :- Internal working of HMap, we'll do it in future class.

In hashmap approach we can search in O(1) time and can have a space complexity of O(N)

HashMap <key, value>
↓ class
↓ class

Ques.

Which of the following HashMap will you use to store the population of every country?

HashMap<String, Double> ✗

HashMap<String, Long> ✓

Ques)

Which of the following HashMap will you use to store the no of states of every country?

India → 28
China → 30

↓

HashMap<String, Int>

Ques)

Which of the following HashMap will you use to store the name of all states of every country?

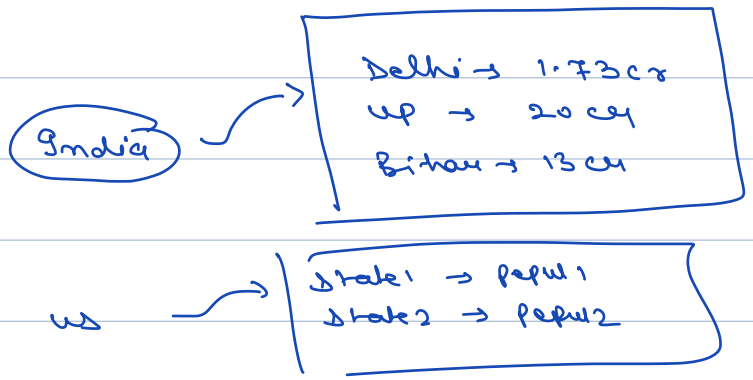
India → Delhi, Maharashtra, Karnataka ~ .

USA → - - - -

HashMap<String, List<String>>

Ques)

Which of the following HashMap will you use to store the population of each state in every country?



HashMap < String, HashMap < String, Long > >

HashSet

Is similar to hashMap, but you can store only keys.

HashSet<KeyType>

HashMap, HashSet → Order is not fixed, key has to be unique.

HashMap :-

O(1)

- **INSERT(Key,Value):** new key-value pair is inserted. If the key already exists, it does no change.
- **SIZE:** returns the number of keys.
- **DELETE(Key):** delete the key-value pair for given key.
- **UPDATE(Key,Value):** previous value associated with the key is **overridden** by the new value.
- **SEARCH(Key):** searches for the specified key.

HashSet :-

O(1).

- **INSERT(Key):** inserts a new key. If key already exists, it does no change.
- **SIZE:** returns number of keys.
- **DELETE(Key):** deletes the given key.
- **SEARCH(Key):** searches for the specified key.

Hashing Library Names in Different Languages

	Java	C++	Python	Js	C#
HashMap →	Hashmap	unordered_map	dictionary	map	dictionary
HashSet →	Hashset	unordered_set	set	set	Hashset

Ques

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

arr → 2 6 3 8 2 8 2 3 8 10 6

Q

2 → 3

8 → 3

6 → 2

idea :-

- For each query, find the frequency of the element in the Array, by traversing
- TC - *O(QN)* and SC - **O(1)**.

idea 2 :-

- Use Hashmap to store the frequency of each element. Store element as key and frequency as value.

```
HashMap<int, int> hmap;
for (i=0; i<n; i++) {
    if (hmap.search(A[i])) {
        hmap[A[i]]++;
    }
    else {
        hmap.insert(A[i],1);
    }
}
```

```
for every Query :- (int x)
    if(hmap.search(x)) {
        print(hmap[x])
    }
    else {
        print(-1);
    }
```

T.C → O(n+q)

S.C → O(n)

Ques)

Given N elements, find the first non-repeating element.

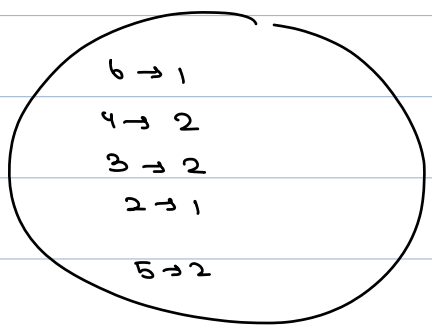
Ex :- 1 2 3 1 2 5 . ans = 3

Ex 2) 4 3 3 2 5 6 4 5 ans = 2

idea 1 :-

- Use Hashmap to store the frequency of each element. Store <**key**:element, **value**:frequency>.
- Iterate over the Hashmap and find the element with frequency 1.

4 3 3 2 5 6 4 5

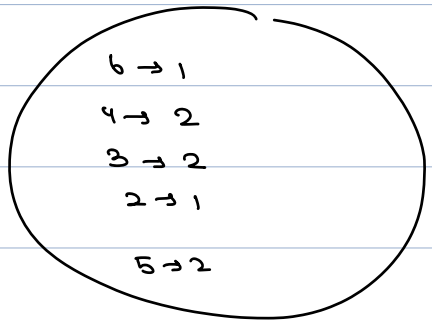


flaw :-

- When we store in Hashmap, the order of elements is lost; therefore, we cannot decide if the element with frequency 1 is first non-repeating in the order described in the Array.

idea2 :-

4 3 3 2 5 6 4 5



- Use Hashmap to store the frequency of each element. Store `<key:element, value:frequency>`.
- Instead of Hashmap, iterate over the Array from the start. If some element has a frequency equal to one, then return that element as answer.

```
Hashmap<int, int> hmap;
for i → 0 to n-1
    if (hmap.search(A[i])) {
        hmap[A[i]]++;
    }
    else {
        hmap.insert(A[i],1);
    }
```

```
for i → 0 to n-1
    if (hmap[A[i]] == 1) {
        return A[i];
    }
```

return -1;

T.C → O(n)

S.C → O(n)

Q3)

Given an array of N elements, find the count of distinct elements.

e.g → 3 5 6 5 4 → 4

e.g → 3 3 3 → 1

idea :- Insert all elements in hashset & return size of it.

```
HashSet<int> set;
for i → 0 to n-1 :-
    set.insert(arr[i]);
return set.size();
```

T.C → O(n)

S.C → O(N)

Ques)

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

N = 10,

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

idea 1:- Traverse for each subarray & get the sum. → $O(n^3)$

We can print all subarray sums,

1) Prefix sum → $O(n^2)$ T.C, → $O(n)$ S.C

2) Carry forward T.C → $O(n^2)$

idea 2 :- sum(i-j) → pf[j] - pf[i-1] (i != 0); → pf[j] (i == 0)

N = 10,

	0	1	2	3	4	5	6	7	8	9
	2	2	1	-3	4	3	1	-2	-3	2
pf sums	2	4	5	2	6	9	10	8	5	7

sum(i-j) = pf[j] - pf[i-1] (i != 0)

=> 0 = pf[j] - pf[i-1]

pf[j] = pf[i-1]

→ If prefix value repeats, it means we have a subarray with sum zero

```
int [] pf →
    pf[0] = arr[0];
    for(i=1; i<n; i++) { pf[i] = arr[i] + pf[i-1]; }

HashSet<int> set;

for i → 0 to n-1 :-
        if( pf[i] == 0) { return True }
        set.insert( pf[i]);

if (set.size() == n) {
        return false;
else {
        return True;
```

T.C → O(n)

S.C → O(n)

edge Case :-

2 3 -5 4 6

pf → 2 5 0 4 10

1, 2, 2 →

1
2
2
1, 2
1, 2
2, 2
1, 2, 2

at every place, we'll make decision for every element starting from its start.

arr, start

(1, 2, 2), -, 0

i > start && i else is equals to prev. elem

(1,2,2), 1, 1

(1,2,2), 2, 2

(1,2,2), 1,2, 2

(1,2,2), 2,2, 3

(1,2,2), (1,2,2), 3

```
helper ( ans list, templist, start) {

        anslist.add( new templist);

        for (i = start; i < arr.len; i++) {

            if (i > start  && nums[i] == nums[i-1] ) {
                                        continue;
            }

            templist.add( nums[i]);

            helper( _ , _ , i+1);

            templist.remove (templist.size()-1);

        }

    }
```