Hashing 1 : Introduction

=> EXAMPLE

Register

Room No.	occupied?
1	yes
2	No
3	yes
4	yes
5	No

↓ 1 year

Suppose we have 1000 Rooms [Register is not Extensible]

1st Possible Solution

Boolean Array :- idx denotes room No
↳ True → occupied
↳ False → Not occupied.

Suppose Now Astrologer Suggest to name room No. randomly from [1 — 10^9] to gain popularity.

Issue 1 :- Solution of Boolean array is not feasible

Issue 2 :- Space Wastage.

SOLUTION → Hashmap → T.C. Search O(1)

Should be class <Key, Value> Should be class

Eg:- 3 → Occupied
1009 → Occupied
45 → Occupied.

HASHMAP

① In Hashmap, T.C. of Search is O(1) time & S.C. is O(N)

② Key must be unique

③ Value can be anything

Quiz 1 :- which of the following Hashmap will you use to store the population of every country ?

Hashmap < String, Long >

Quiz 2 which of the following Hashmap will you use to store the no. of states of every country ?

Eg:- India → 28
China → 34

Hashmap < String, Integer >

Ques 3 which of the following Hashmap will you use to store the name of all state of every country.

Eg:- India → Delhi , U.P., Bihar .. ...

```
Hashmap < String , List < String >> ;
```

Ques 4 which of the following Hashmap will you use to store the population of each state in every country?

Eg:-

India →

Delhi	– 10 Cr
UP	– 20 Cr
Bihar	– 15 Cr

```
Hashmap < String, Hashmap < String , Long>>;
```

HASHSET

↳ Sometimes we only want to store keys & do not want to associate any values with them, in such a case we use HashSet.

```
Hashset < Key >
```

→ Search in O(1) T.C.

↳ It will be unique.

Hashmap, Hashset → It won't maintain order of key.

Function of Hashmap

→ INSERT (Key, Value): new Key - Value pair is inserted. If the key already exists, it does no change.

→ SIZE: returns the no. of Keys

→ DELETE (Key): delete the key - value pair for given key.

→ UPDATE (Key, Value): previous value associated with the key is overrided by the new value.

→ SEARCH (Key): search for the specified Key.

=> Function of Hashset

→ INSERT (Key): Insert a new key. If key already exists, it does no change.

→ SIZE: return number of keys

→ DELETE (Key): deletes the given key.

→ SEARCH (Key): searches for the specified key.

★ Time complexity of each operation in Hashmap & Hashset is O(1)

Hashing libraries in Different languages

	Java	C++	Python	JS	C#
Hashmap:	Hashmap	unordered_map	dictionary	Map	dictionary
Hashset:	Hashset	unordered_set	Set	Set	Hashset

Question 1 :- Given N elements & Q queries, find the frequency of the elements provided in a query.

Ex 1 :- N = 10

2, 6, 3, 8, 2, 8, 2, 3, 8, 10, 6

Queries :-

		Ans
2	→	3
8	→	3
6	→	2

Brute force

↳ For each query. Iterate over array to get frequency.

TC	→ O(Q * N)
SC	→ O(1)

↳ **OPTIMIZATION**

Idea → Use Hashmap where key will be element of Array & Value will be the count of that element in Array

PSEUDO CODE

```
Hashmap < Int , Int > hmap ;
for ( i = 0;  i < A.length ; i++){
    if ( hmap.search(A[i]) $
        hmap [A[i]++];
    } else {
        hmap.insert(A[i], 1);
    }
```

```
}

for (i=0; i<a.length; i++){
    if (hmap.contains(a[i])){
        print (hmap(a[i]));
    }else{
        print (0);
    }
}
```

TC → O(N+θ)

SC → O(N)

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

Ex :- N = 6

1, 2, 3, 1, 2, 5

Ans = 3

Ex N = 8

4, 3, 3, 2, 5, 6, 4, 5

Ans = 2

IDEA [Hashmap solution]

Wrong.
1. create freq Map of given array
2. Iterate over map & return the element with freq 1.

Flaw :- when we store in Hashmap, the order of element is lost. Therefore, we cannot decide if the element with frequency 1 is first non-repeating in the order described in the array.

CORRECT SOLUTION

1. create freq Map of given array
2. Iterate over given array & print the first element with freq 1.

PSEUDO CODE

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

for (i =0; i < A. length ; i++){
    if ( hmap (A[i]) == 1 ) {
        print (A[i]);
        break;
    }
}
```

TC - O(N)
SC - O(N)

Question3 Given an array of N elements, find the count of distinct elements.

Ex:- N = 5

3, 4, 6, 5, 4 Ans = 4

N = 3

3, 3, 3 Ans = 1

N = 5

1, 1, 1, 2, 2 Ans = 2

IDEA

↳ Store All elements in Set & return size of Set.

PSEUDO CODE

```
Hashset < int > set;
for ( i = 0; i < A.length; i++) {
    set.insert(A[i]);
}
return set.size();
```

TC ⟶ O(N)

SC ⟶ O(N)

Question 4 Given an array of N elements, check if there exists a subarray with a sum equal to 0.

Ex :- N = 10

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

IDEA1 :- Traverse over each subarray & check sum == 0.

TC → O(N³)

SC → O(1)

IDEA2 :- Prefix Sum [To skip going from Start to End]

TC → O(N²)

SC → O(N)

IDEA3 :- Carry forward

TC → O(N²)

SC → O(1)

IDEA4 :- Lets once see prefix sum.

Ex N = 10

	2	2	1	-3	4	3	1	-2	-3	2
PSum:	2	4	5	2	6	9	10	8	5	7

Lets use Real life Example

In Jan start → [Bank] 5 lakh

↓

In Feb Start → 5 lakh

Maths

$sum[i-j] \longrightarrow P[j] - P[i-1]$

$\Rightarrow \quad 0 = P[j] - P[i-1]$

$\Rightarrow \quad \boxed{P[j] = P[i-1]}$

OBSERVATION

↳ If all distinct then No subarray with 0 sum

↳ achieve → Set.size == Array.size.

↳ else

↳ Definately their is subarray with 0 sum.

PSEUDO CODE

```
int [] Pf
PP[0] = A[0];
for ( i=1; i< A.length; i++){
    pf[i] = Pf[i-1] + A[i];
}
```

```
if ( Pf[i]==0) {
    return true;
}
```

```
Hashset <Int> set;
for ( i = 0; i <Pf.length; i++){
    set. insert ( Pf[i]);
}
if ( set. Size == A. length ) {
    Print False;
} else {
    Print True;
}
```

Edge Case :-

Ex:-	2	3	-5	4	6
Pf Sum	2	5	0	4	10

Q → why we have missed this case ?

Sum [i-j] → Pf[j] - Pf[i-1] (i!=0)

Pf[j] (i==0)

How to handle ?

↳ During PF creation check this

```
if ( PF[i] == 0) {
        return True;
}
```

TC — O(N)
SC — O(N)

Question 5 :- Given an Array A of N integers. Find the Count of the Subarrays in the array which sums to 0.

Note :- return Ans % 10^9+7

Ex 1 A = [1, -1, -2, 2]

↳ H.W.