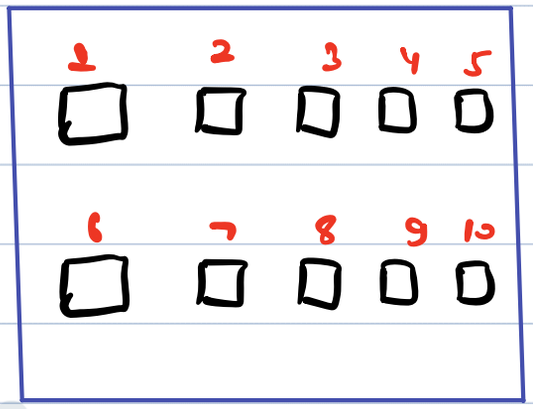
Today's agenda

- HashMap Intro
- frequency of each query
- first non-repeating element
- HashSet
- number of distinct elements
- Pair sum == K

// HashMap Intro

(i) Shubh

M1



boolean r1 : true

r2 : true

r3 : false

.

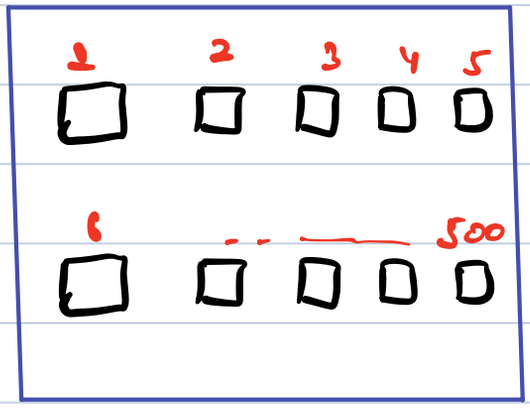
.

.

r10 : false

(ii) Mayur

M2

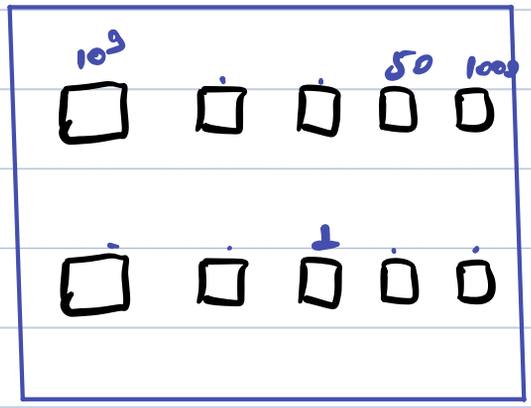


boolean room[501];

room[500] = true;

(11) Anand

$\{1, 10^9\}$ ↙ 500 rooms

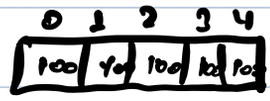


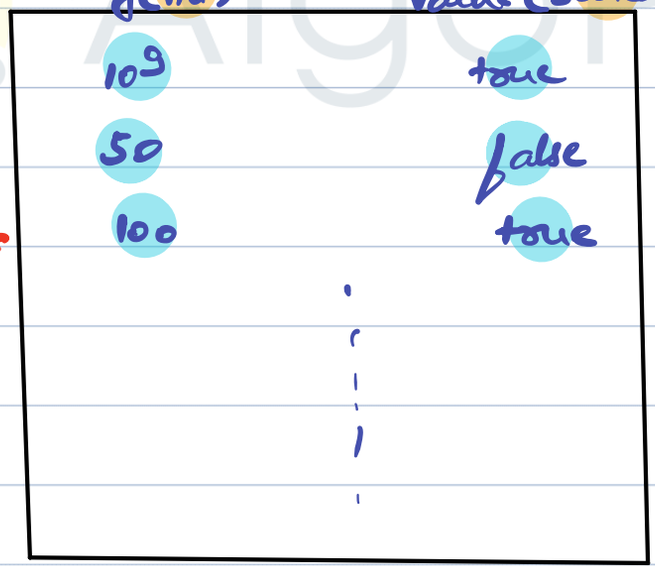
M3

boolean rooms $[10^9+1]$;

↳ Space wastage

↳ HashMap will solve this issue.





→ (key, value) Pair

no. of keys will be 500 for above problem.

Q) Store Population of every country:

Key: Country name : String
value: Population : int/long

Q) Store All the School with their Principal name.

key: School name: String
value: Principal name: String

facts:

(i) Keys can be only of following types: (Datatype)

(i) int → Integer (wrapper class)
(ii) long → Long
(iii) boolean → Boolean
(iv) char → Character
(v) double → Double
(vi) String → String

not allowed: Arrays, Array List.

(ii) values can be of any type.

Syntax:

```
HashMap<Integer, String> hm = new HashMap<>();
```

(Integer = key, String = value, hm = name)

```
add
hm.Put(key, value);
hm.Put(10, 50);
hm.Put("Subhesh", 60); → error
hm.Put(20, 60);
hm.Put(10, 70); → update
hm.Put(30, 60);
```

key (Integer)	value (Integer)
10	50 → 70
20	60
30	60

T.C: O(1)

↳ keys are just like index of array.

```
// get
↳ hm.get(20); → 60
↳ hm.get(40); → null
```

T.C: O(1)

```
// size
↳ hm.size(); → 3
```

↳ T.C: O(1)

```
// Contains Key

hm.containsKey(20); → true
hm.ContainsKey(60); → false
```

T.C: O(1)

// remove

↳ hm.remove(10); ← key

T.C: O(1)

↳ In hashmap, the order of storing key is not necessarily same as order of adding them.

↳ Predictable.

// iterate

	0	1	2	3	4
arr[N]:	100	200	300	400	500

i → { 0-4 }

```
for (int i=0; i<N; i++){
    s.o.p(arr[i]);
}
```

v → {100 200 300 .. 500}

```
for (int v: arr){
    s.o.p(v);
}
```

In HashMap iterating on Keys

v → {10 20 30}

```
for (int v: hm.keySet()){   // keys
}
```

T.C: O(n)

Q) find frequency

↳ Given N array elements & Q queries. for every query find frequency of element in array.
↳ no. of occurrence

Ex: arr[11]: { 2 6 3 8 2 8 2 3 8 10 6 }
0 1 2 3 4 5 6 7 8 9 10

queries[4] = { 2 8 3 5 }
ans[4] = { 3 3 2 0 }

//idea1

↳ iterate and count for every query.

T.C: O(Q * N)
↳ Q :: N
O(N^2)

S.C: O(1)

//idea2

↳ Create HashMap of given array.

Ex: arr[11]: { 2 6 3 8 2 8 2 3 8 10 6 }
0 1 2 3 4 5 6 7 8 9 10

hm

array element	freq
2	x3
6	x2
3	x2
8	x3
10	1

queries[4] = { 2 8 3 5 }
3 3 2 0

O(N) + O(Q) = O(N+Q)
Q ≈ N ⇒ O(N)

//Pseudo Code

```
void PrintFrequency (int arr[N], int queries[Q]){

    HashMap<Integer, Integer> hm = new HashMap<>();

    for (int i=0; i<N; i++) {
        if (hm.containsKey (arr[i]) == false){
            hm.put (arr[i],1);
        }
        else{
            int temp = hm.get (arr[i]);
            hm.Put(arr[i], temp+1);
        }
    }

    for (int i=0; i<Q; i++ ) {
        if (hm.containsKey (query[i] ) == false){
            s.o.p ("0");
        }
        else{
            s.o.p ( hm.get (queries[i]) );
        }
    }
}
```

T.C: O(N+Q)

S.C: O(N)

```
for (int i=0; i<N; i++) {
    if (hm.containsKey(arr[i]) == false) {
        hm.put(arr[i], 1);
    }
    else {
        int temp = hm.get(arr[i]);
        hm.Put(arr[i], temp+1);
    }
}
```

arr[11]: { 2 6 3 8 2 8 2 3 8 10 6 }
temp = 2

hm

2	x23
6	x2
3	x2
8	x23
10	1

```
for (int i=0; i<Q; i++) {
    if (hm.containsKey(query[i]) == false) {
        s.o.p("0");
    }
    else {
        s.o.p(hm.get(queries[i]));
    }
}
```

queries[4] = { 2 8 3 5 }
i: 0 1 2 3
3 3 2 0

Break till 10:50 Pm

Q) Find the first non-repeating elements

↳ return -1 if all the elements are repeating.

Ex1: arr[6]: {1 2 3 1 2 5} →3

arr[8]: {5 4 4 3 6 7 5 6} →3

//idea

arr[8]: { 5 4 4 3 6 7 5 6 }
i

5	x2
4	x2
3	1
6	x2
7	1

↳ iterate on array and get the first element with freq 1

// Pseudo code

```
int firstnonrepeatingelement(int arr[N]){

    HashMap<Integer, Integer> hm = new HashMap<>();

    for(int i=0; i<N; i++){
        if(hm.containskey(arr[i])==false){
            hm.put(arr[i],1);
        }
        else{
            int temp = hm.get(arr[i]);
            hm.Put(arr[i], temp+1);
        }
    }

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

    return -1;
}
```

T.C: O(N)

S.C: O(N)

// HashSet

→ only the key Part of HashMap

```
HashSet<Integer> hs = new HashSet<>();
```

T.C: O(1)

hs.size() ← 10 20 ~~30~~

↓ 2

hs.contains(20) → T

T.C: O(1)

hs.add(10) → add

hs.add(20) → add

hs.add(30) → add

hs.add(20) → Nothing will happen

T.C: O(1)

hs.remove(30)

T.C: O(1)

Q) Given arr[N], find no. of distinct elements.

Ex: arr[5] = {4 6 7 6 5} → ans: 4

arr[5] = {10 10 10 20 20} → ans: 2

//idea

arr[5] = {10 10 10 20 20} i:

hs: | 10 20 |

↳ hs.size()

//Pseudo code

```
int distinctElements (int arr[N]){

    HashSet<Integer> hs = new HashSet<>();

    for (int i=0; i<N; i++) {
        hs.add(arr[i]);
    }

    return hs.size();

}
```

T.C: O(N)

S.C: O(N)

Q) Pair Sum == K

↳ Given arr[N], check if there exists a Pair(i,j) such that arr[i] + arr[j] == K and (i != j).

	0	1	2	3	4	5	6	7	8	9
arr[10]:	8	9	1	-2	4	5	11	-6	7	5

K = 11 arr[4] + arr[8] ⇒ 4 + 7 = 11 → true

K = 6 arr[0] + arr[3] ⇒ 8 - 2 = 6 → true

K = 22 ~~arr[6] + arr[6]~~ i != j → false

Hideal

↳ Nested loop.

T.C: O(N²)

Idea 2 → wrong idea

	0	1	2	3	4	5	6	7	8	9
arr[10]:	8	9	1	-2	4	5	11	-6	7	5

i

(1) insert all elements in hashset.

hs: { 8 9 1 -2 4 5 11 -6 7 }

k = 11

(i) $a + b = 11$

$b = 11 - a$

a	b	is b Present?
8	3	NO
9	2	NO
1	10	NO
-2	13	NO
4	7	yes → return true

i

	0	1	2	3	4	5	6	7	8	9
arr[10]:	8	9	1	-2	4	5	11	-6	7	5

hs: { 8 9 1 -2 4 5 11 -6 7 }

(ii) $a + b = -4$

a	b	is b Present?
8	-12	NO
9	-13	NO
1	-5	NO
-2	-2	yes → return true

Idea 3

↳ insert all elements of array with frequency of it.

	0	1	2	3 (i)	4	5	6	7	8	9
arr[10]:	8	9	1	-2	4	5	11	-6	7	5

8 → 1	11 → 1
9 → 1	-6 → 1
1 → 1	7 → 1
-2 → 1	
4 → 1	
5 → 2	

(11) $a + b = -4$

a	b	is b Present?
8	-12	NO
9	-13	NO
1	-5	NO
-2	-2	yes but freq is 1 even though a = -b.

// Pseudo code

```
boolean PairSum (int arr[N], int K){

    HashMap<Integer, Integer> hm = new HashMap<>();

    for (int i=0; i<N; i++){
        if (hm.containsKey(arr[i]) == false){
            hm.put(arr[i],1);
        }
        else{
            int temp = hm.get(arr[i]);
            hm.Put(arr[i], temp+1);
        }
    }
    for (int i=0; i<N; i++){
        int a = arr[i];
        int b = K-a;

        if (a!=b && hm.containsKey(b) == true){
            return true;
        }
        else if (a==b && hm.get(b) >=2){
            return true;
        }
    }
    return false;
}
```

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