- Set ⟵ (Hashset)
  Map ⟵ (Hashmap)

```
String a = "abc";
```

0 1 2

a[0] = 'd' //

a = "dbc"

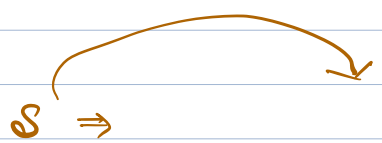
StringBuilder sb = new StringBuilder();

sb.charAt(i); // access char at index i

Syntax

sb.setCharAt(i, 'c');
sb.setCharAt(0, 'd');

sb.toString() // returns the string that you are building.



Set ⇒ Unique values

HashSet

Search / find ⇒ O(1)

⇒ ⇓

Set s = [5, 7, 9, 31, 42, 6, 17]

⇒ O(1) search

Java / C#	C++	Python / Ruby / JS	C
HashSet	Unordered_set	Set	✗

Set
- get / find (x) ⇒ O(1)
- add (x) ⇒ O(1)
- size() ⇒ O(1)
- delete (x) ⇒ O(1)

A: [1, 2, 1, 2, 1, 2, 3, 3, 4, 4]

```
HashSet s = new HashSet();
for (i=0; i<A.size(); i++) {
    s.add(A[i]);
}
```

s ⇒ {1, 2, 3, 4} ⇒

Set only stores unique elements.

Q. Given an array of size N.
Return the count of distinct elements in the array.

Eg ⇒ A: [7, 3, 2, 1, 3, 7, 0]

⇒ N dist

7, 3, 2, 1, 0 ⇒ 5

Set
- Tree based implementation (ordered) — log(n)
- Hash based implementation (unordered) — O(1)

```
Hashset s = { }   // Extra space
for (i=0; i<N; i++){
        s.add(A[i]);
}
return s.size();
```

T.C = O(N)

S. C. = O(N)

Q-2 Given an array of size N.

Given Q queries

⇒ x ⇒ Return the frequency of x

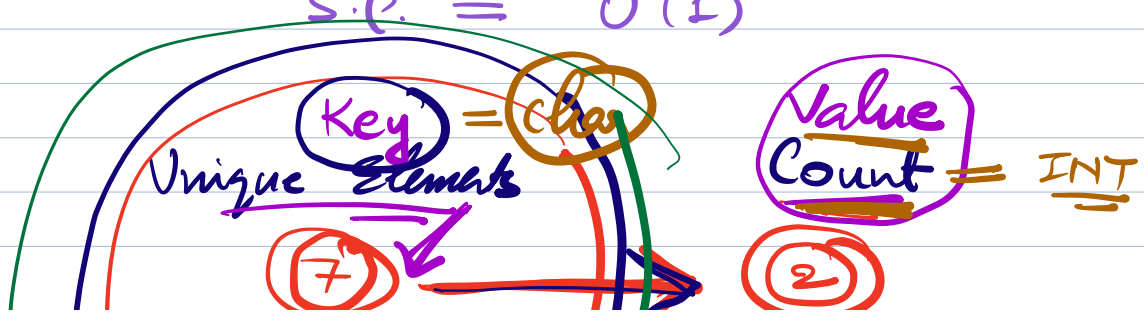
A : 7, 2, 7, 1, 2, 8, 1, 0, 8

Q → 7 = 2
1 = 2
9 = 0
⋮

Soln 1) Iterate & count

T. C = O(N × Q)

S. C. = O(1)



Map Pair < Key , Value >

Hashing as internal implementation

HashMap

Java	C++	Python
HashMap	unordered-map	Dictionary

Ruby / JS

Map

C ⇒ C++ ⇒ STL

Map < Key , Value >

- Store Countries – Capital

  Map < String , String >

- Store country – population

  Map < String , Long >

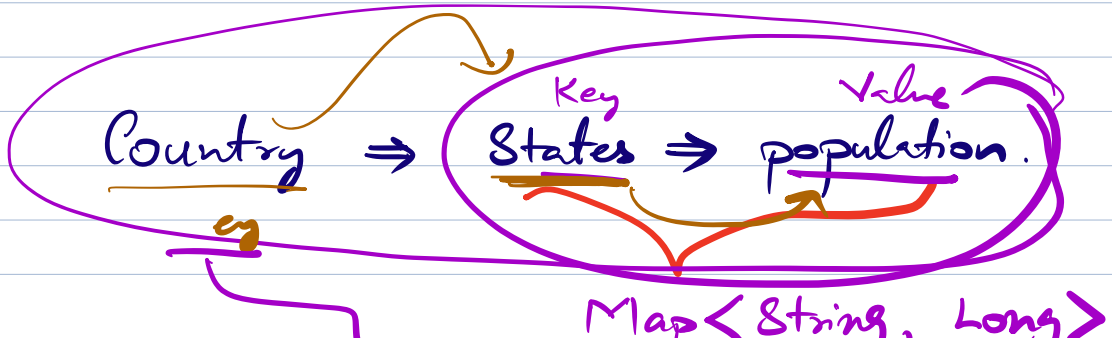
- Store County – all states of county.

  Key

  Map < String, Array [String] >

India = 29

All the Keys of a map will be unique.

- 

Map < String, Map < String, Long > >

India ⇒ UP =
Rajasthan =
WB =
Orissa =

USA ⇒ DC =
Calif =
Tex =

India, UP

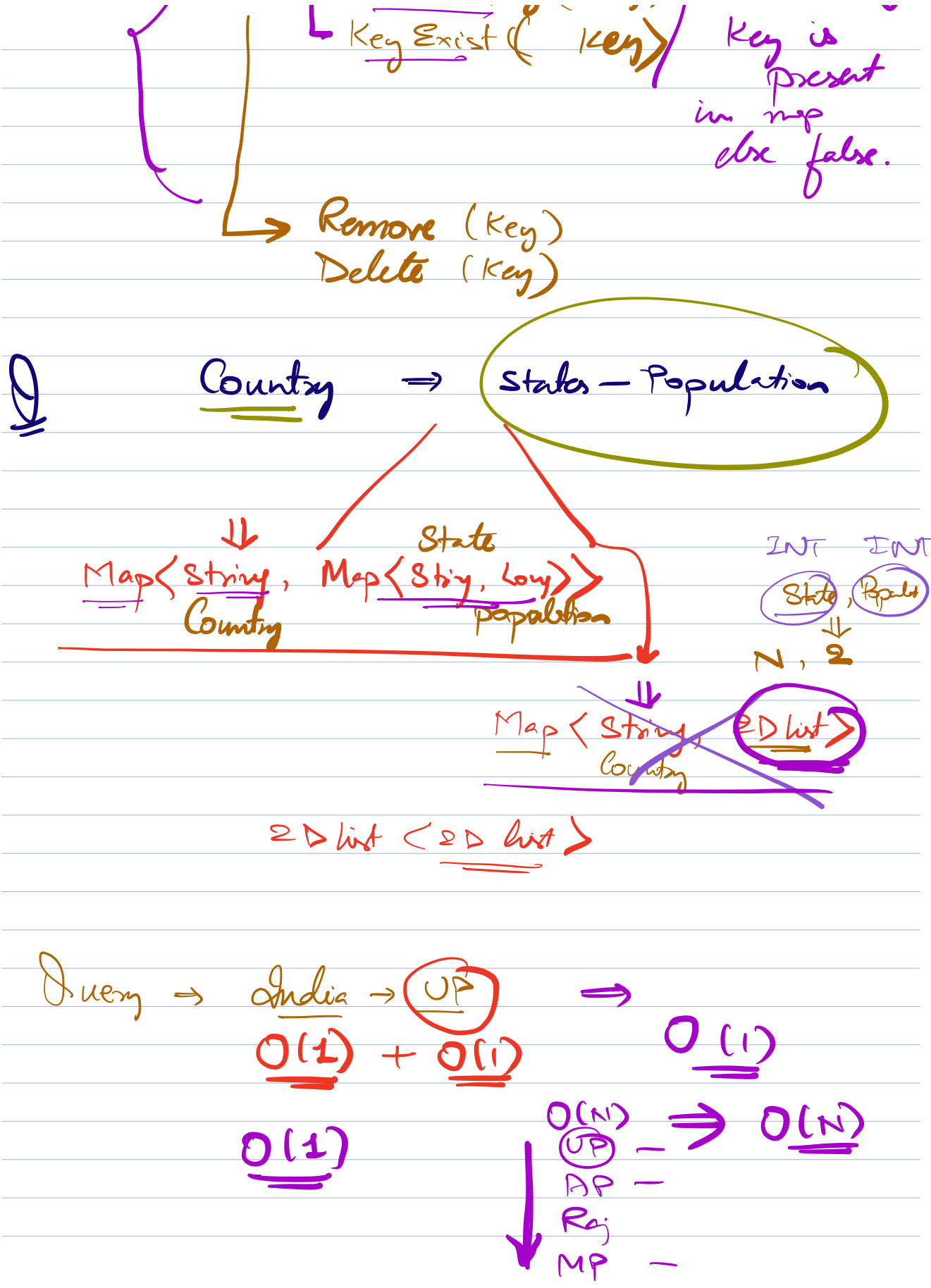
map.containsKey('India')

Map < Country, Capital > map.get(India) = New Delhi

Hash Map

- get / find (x) // Value associated with the key X
- add / put (Key, Value)
- size () // Total no. of Keys
- update (Key, Value)
- isPresent (Key) / contains Key (Key) / Returns true if

O(1)


KeyExist(key) Key is present in map else false.
Remove (Key)
Delete (Key)
Country ⇒ States – Population
Map<String, Country
Map<Sting, Long> Population
State
INT INT
State, Popula
N, 2
Map<String, 2D list>
Country
2D list < 2D list >
Query → India → UP ⇒
O(1) + O(1)
O(1)
O(1)
O(N)
O(N)
UP –
AP –
Raj
MP –

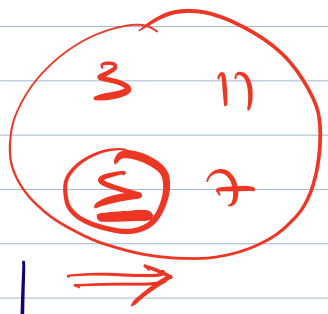
HashMap HashSet

Unordered manners

⇒ Iterate

Set (S)

set.add (3) ✓
set.add (5) ✓
set.add (7) ✓
set.add (11) ✓



Ordered set (Increasing)

7, 15, 3, 2, 1.

1, 2, 2, 7, 15

Given an array of size N.

Find the first non repeating element in the array.

freq = 1

Eg ⇒ A: 8, 2, 8, 3, 1, 2, 6, 5

Soln

1) ⇒ Iterate from i = 0 to i = n-1 — O(N) ✗

O(N) ⇒ Count the freq of element

O(1) ← HashMap

T.C = O(N²)

S.C = O(1)

Optimized soln

1) Create a frequency map from the given array.

Key — Element

Value — No. of times element is present (frequency)

2) Iterate over the array & the first element with freq = 1 is the answer.

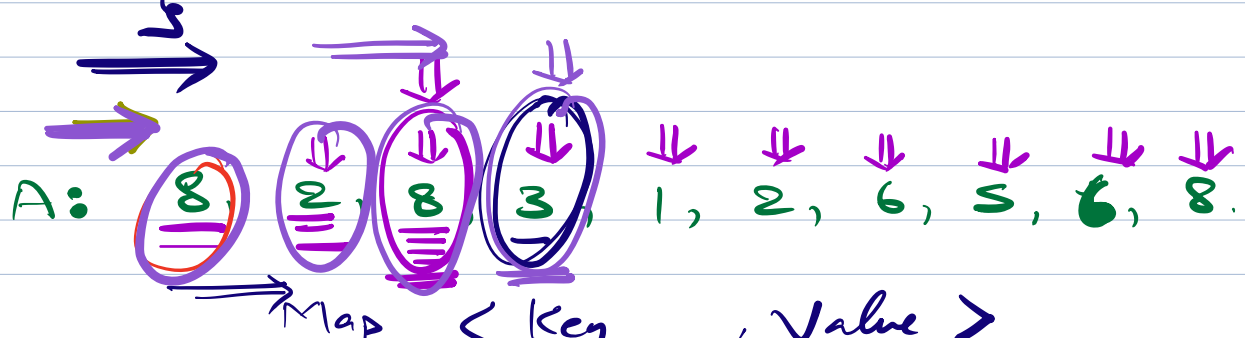
Code

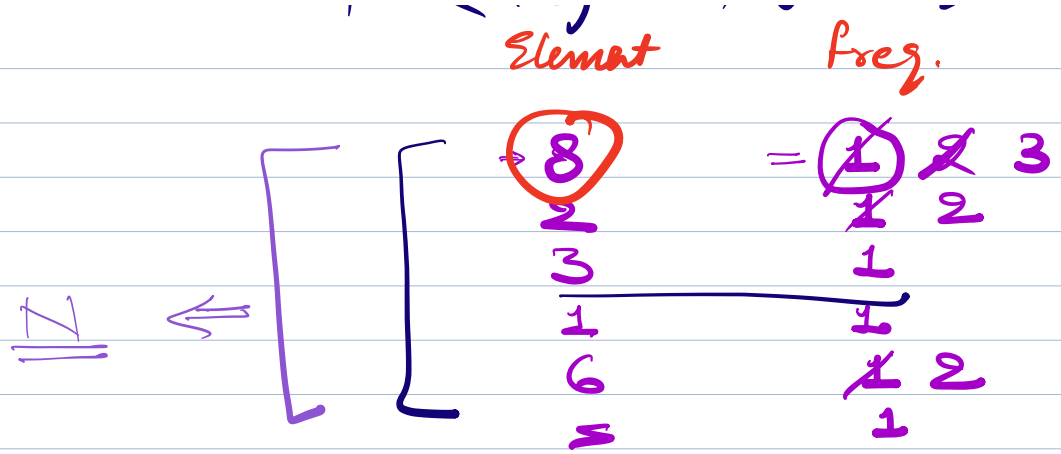
1) `Map<int, int> freqMap = new HashMap.`

```
for (i=0; i<N; i++) {
    if (!freqMap.contains(A[i])) {
        freqMap.put(A[i], 1);
        continue;
    }
    count = freqMap.get(A[i]);        // Read
    freqMap.update(A[i], count+1);    // Write
    or
    (freqMap[A[i]]++;)  //another way to update
}
```

2)

```
for (i=0; i<N; i++) {          → O(N)
    if (freqMap.get(A[i])==1){ ← O(1)
        return A[i];
    }
}
```





freqMap.put (8, 2) ✗ ⇒ Error

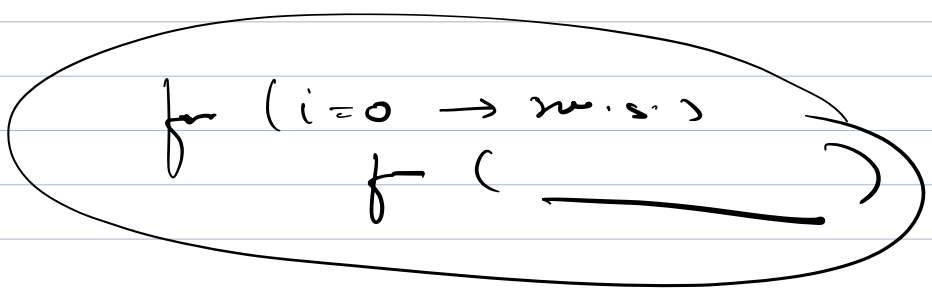
T.C. = O(N + N)

= O(N)

S.C. = O(N)

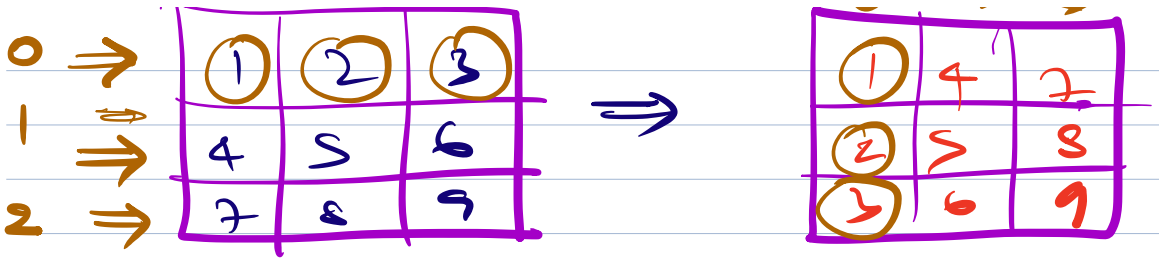
Doubt

2D matrix ⇒ Iterate row wise



Transpose ⇒

0 1 2



0,0 ⇒ 0,0
0,1 ⇒ 1,0
0,2 ⇒ 2,0

1,0 ⇒ 0,1
1,1 ⇒ 1,1
1,2 ⇒ 2,1

2,0 ⇒ 0,2
2,1 ⇒ 1,2
2,2 ⇒ 2,2

A: [ajosh, ajeshman, ajman, ajda]

bool goOn = true

aj

index = 0

1

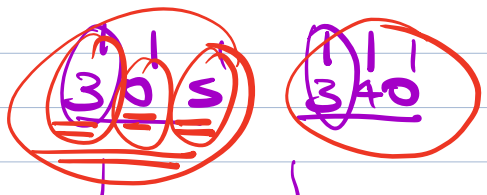
while (goON) {

for (i=0; i<N; i++) {

if (!A[i].charAt(index) == A[0].charAt(index))

goON = false;

}



30, 34, 5, 9

9, 5, 34, 30

sort (,) ⟹

a > b ? a : b

comp (int1, int2) {

digit1 = log(d1)

digit2 = log(d2)

while (digit1 != 0 && digit2 != 0) {

d1 = int1 / 10^digit1

int1 = int1 % 10^digit1

d2 = int2 / 10^digit2

int2 = int1 % 10^digit2

if (d1 > d2)