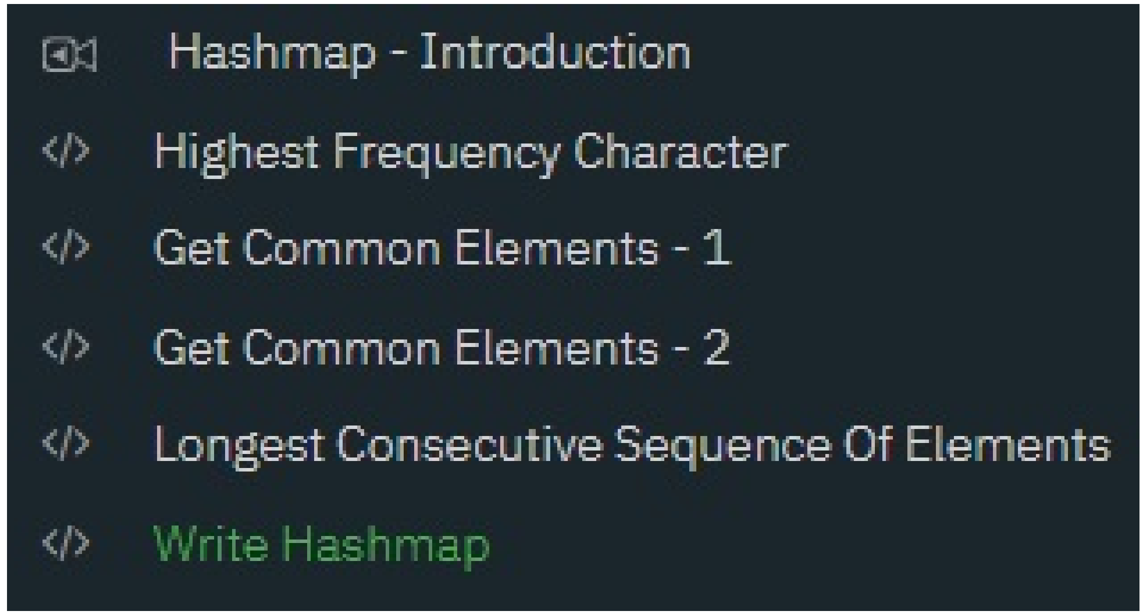
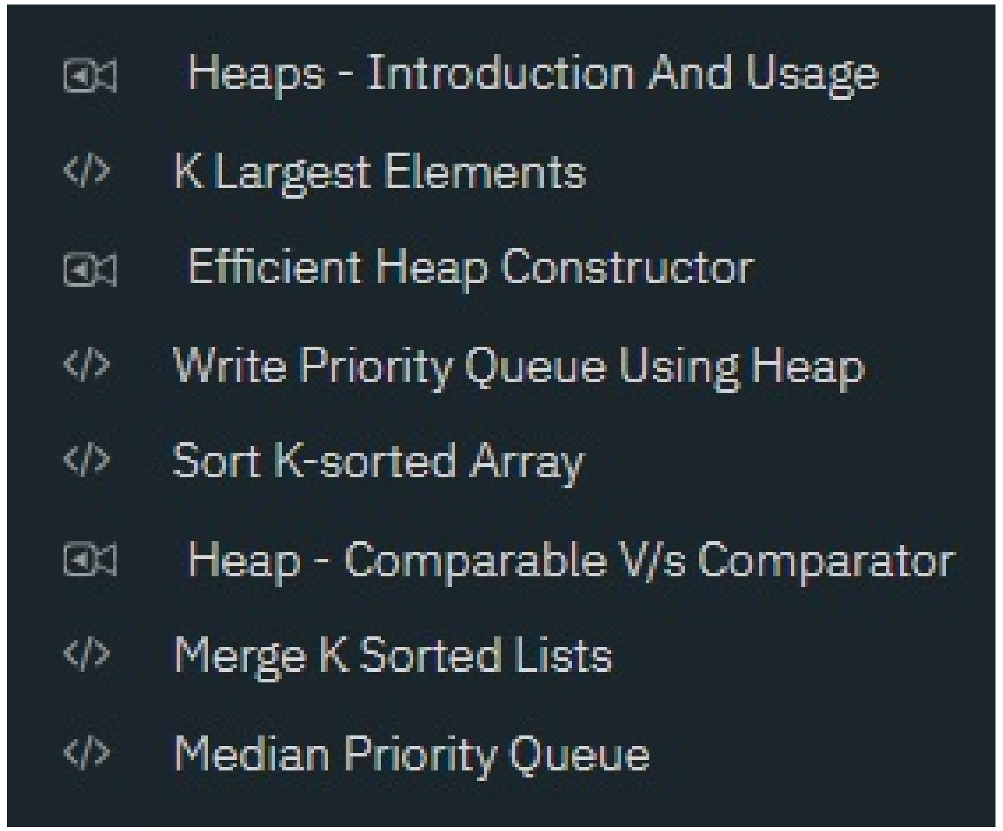
Hashmap


Hashmap - Introduction
Highest Frequency Character
Get Common Elements - 1
Get Common Elements - 2
Longest Consecutive Sequence Of Elements
Write Hashmap

Heap/Priority Queue


Heaps - Introduction And Usage
K Largest Elements
Efficient Heap Constructor
Write Priority Queue Using Heap
Sort K-sorted Array
Heap - Comparable V/s Comparator
Merge K Sorted Lists
Median Priority Queue

#Hashmap

Insert, Update → put ⇒ $\boxed{O(1)}$

Delete → remove ⇒ $\boxed{O(1)}$

Read → get ⇒ $\boxed{O(1)}$

↳ if key exist then return value
else return null;

Display

size ⇒ O(1)

Containskey → find ⇒ O(1)

keyset → keys

Key → value

eg IPL teams → trophies count
String → int

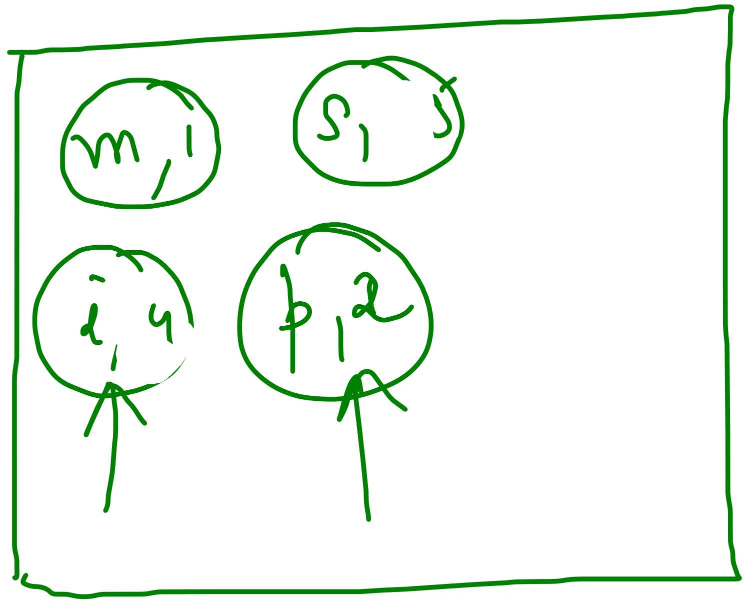
CSK	4
MI	5
SRH	2
DC	0

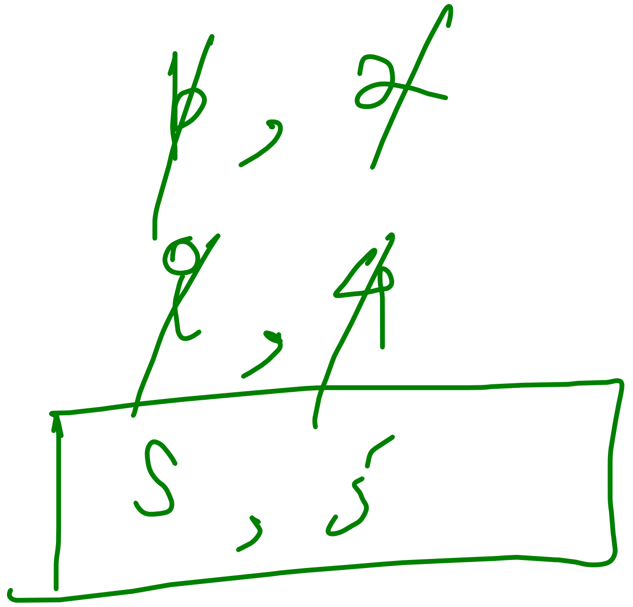
eg countries → population
String → Integer

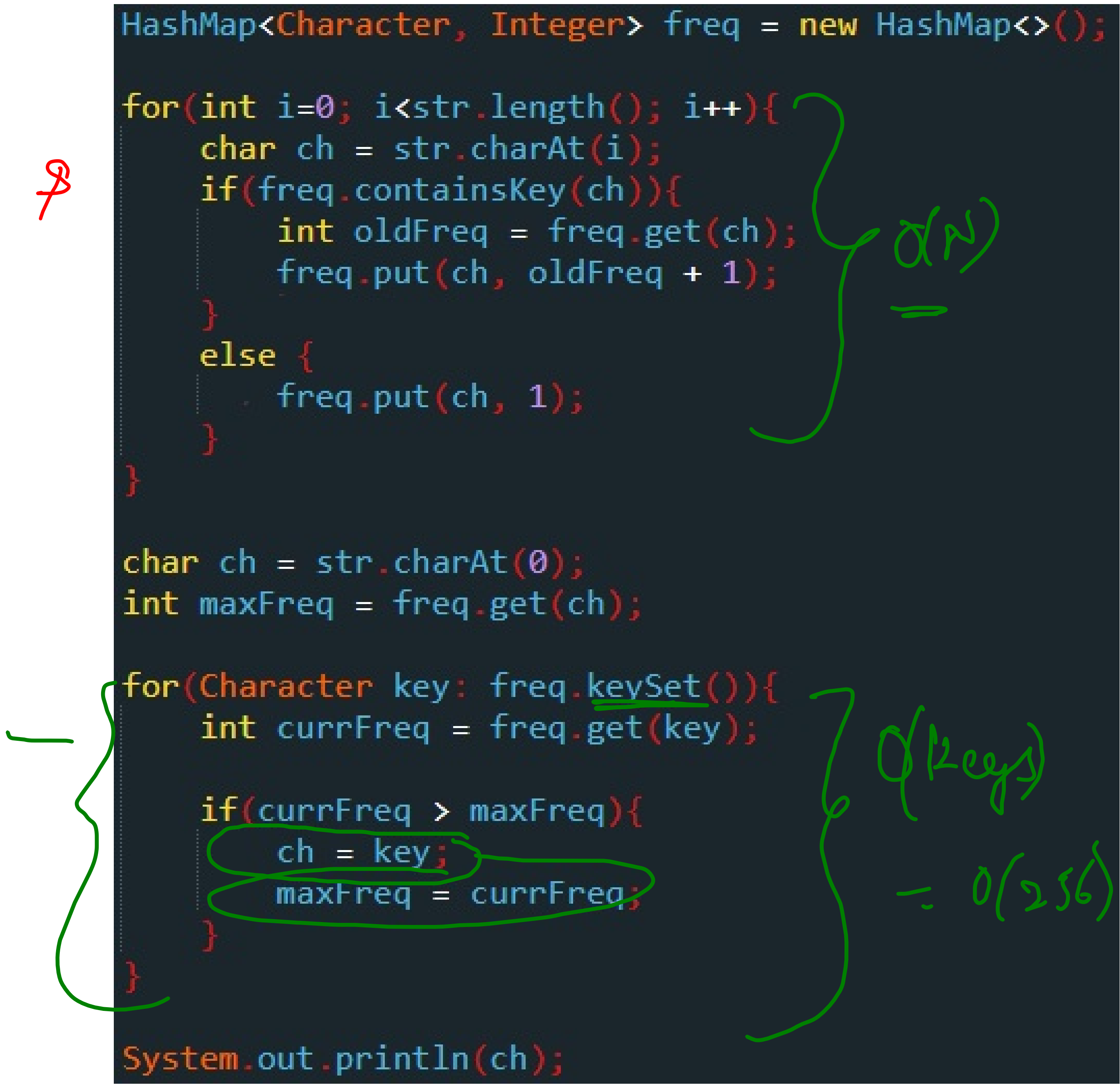
India	130
USA	40
China	200

Highest Frequency Character

m i s s i s s i p p i s

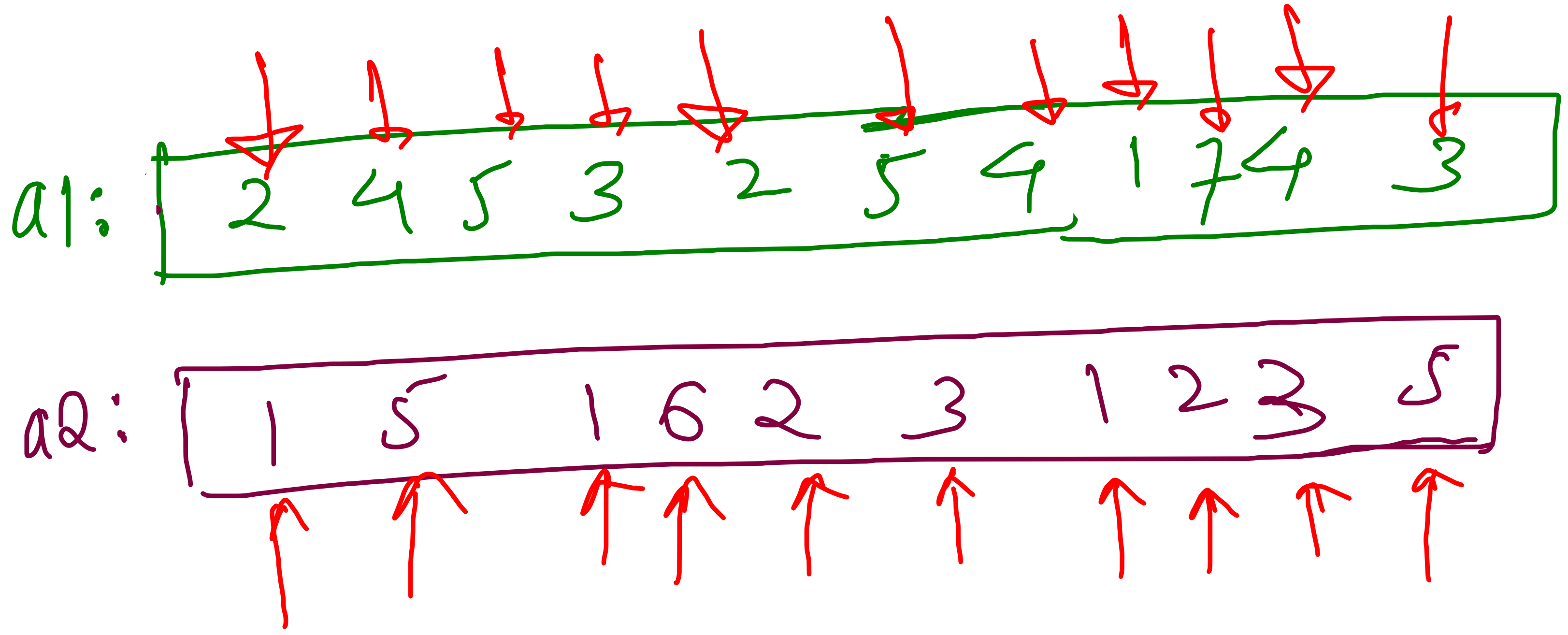




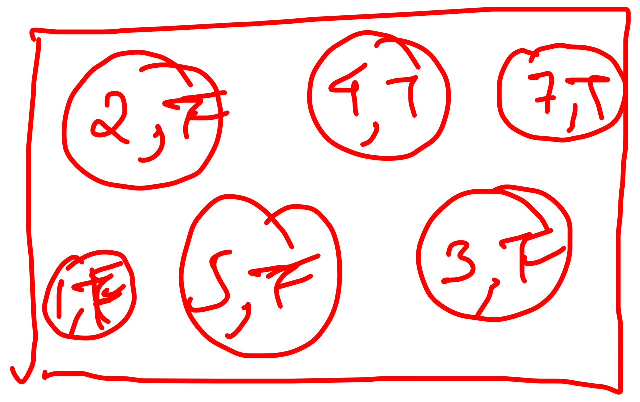


```java
HashMap<Character, Integer> freq = new HashMap<>();

for(int i=0; i<str.length(); i++){
    char ch = str.charAt(i);
    if(freq.containsKey(ch)){
        int oldFreq = freq.get(ch);
        freq.put(ch, oldFreq + 1);
    }
    else {
        freq.put(ch, 1);
    }
}

char ch = str.charAt(0);
int maxFreq = freq.get(ch);

for(Character key: freq.keySet()){
    int currFreq = freq.get(key);

    if(currFreq > maxFreq){
        ch = key;
        maxFreq = currFreq;
    }
}

System.out.println(ch);
```

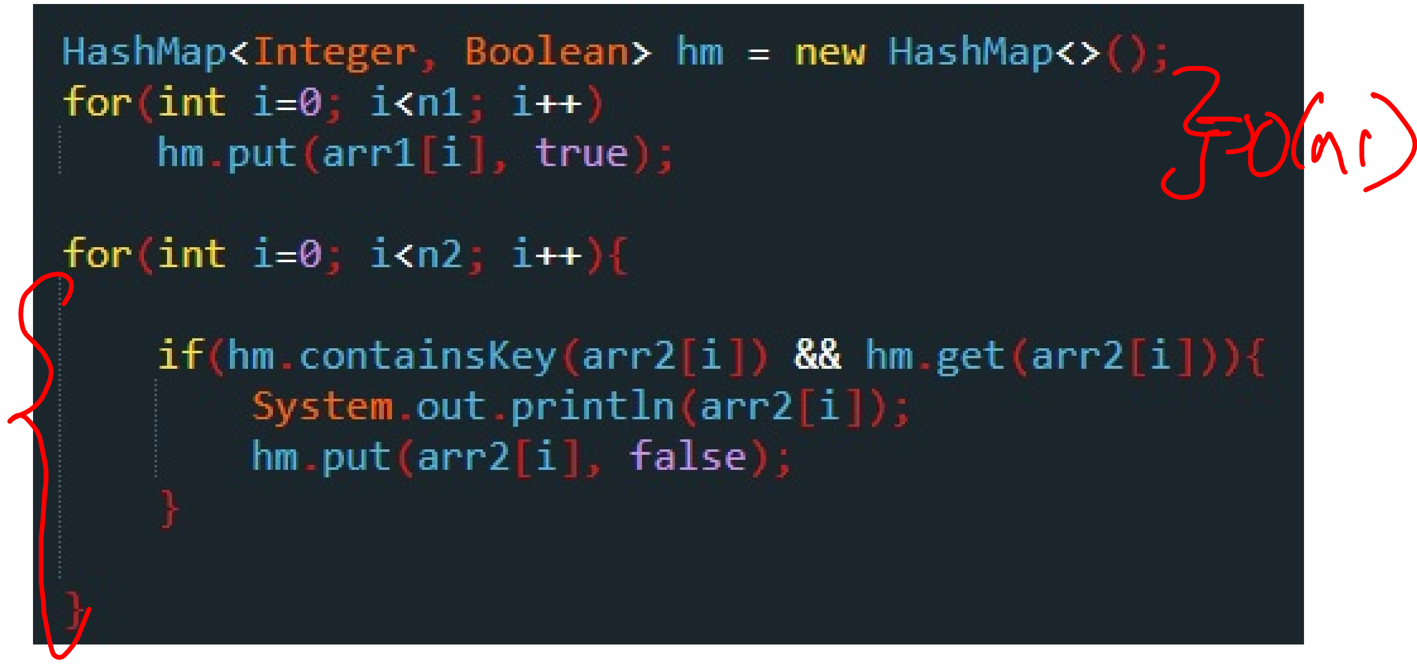
Get Common Elements - 1



1, 5, 2, 3



<Integer, Boolean>



```
HashMap<Integer, Boolean> hm = new HashMap<>();
for(int i=0; i<n1; i++)
    hm.put(arr1[i], true);

for(int i=0; i<n2; i++){

    if(hm.containsKey(arr2[i]) && hm.get(arr2[i])){
        System.out.println(arr2[i]);
        hm.put(arr2[i], false);
    }

}
```

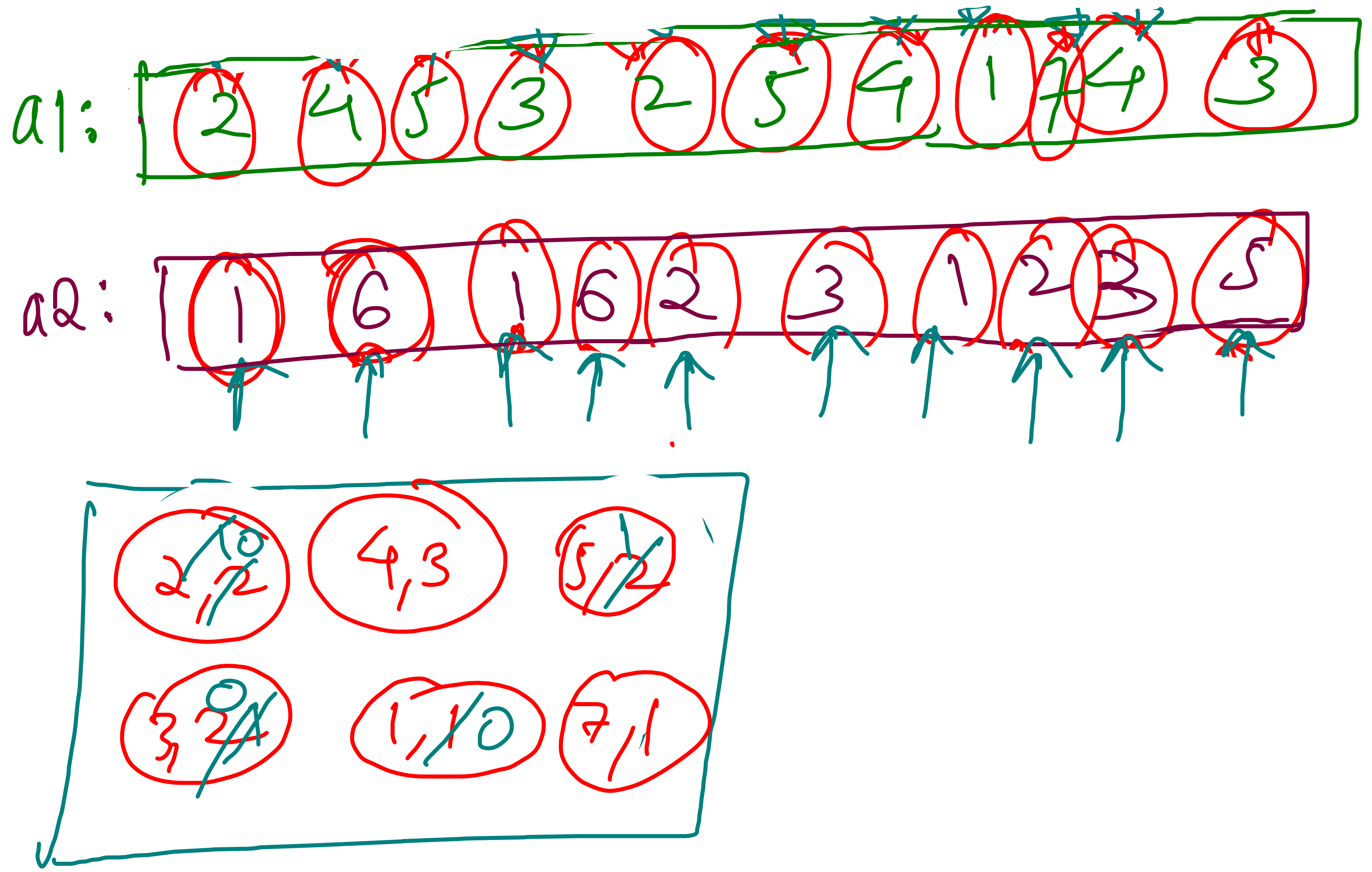
$\} \Rightarrow O(n_1)$

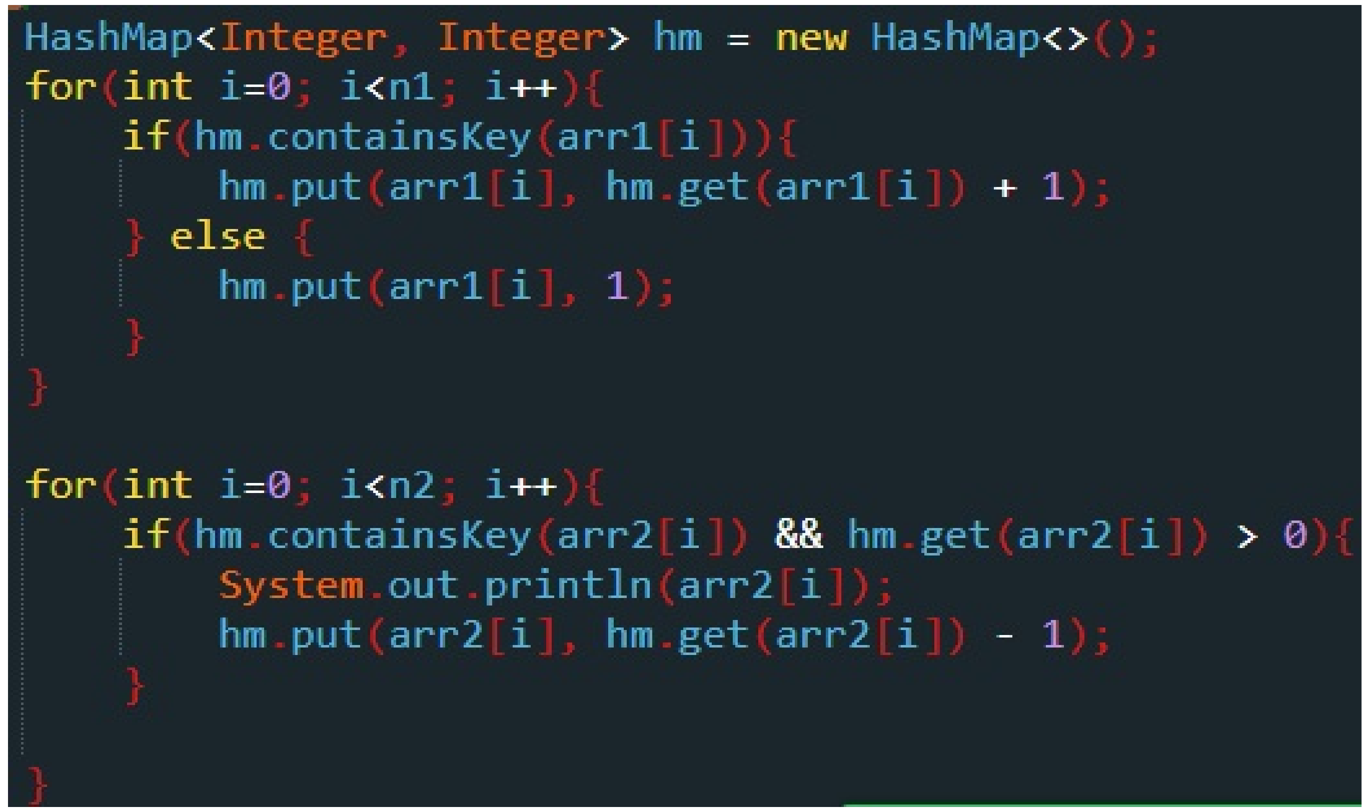
$O(n_2) \Leftarrow$

$\boxed{O(n_1 + n_2)}$

Get Common Elements - 2

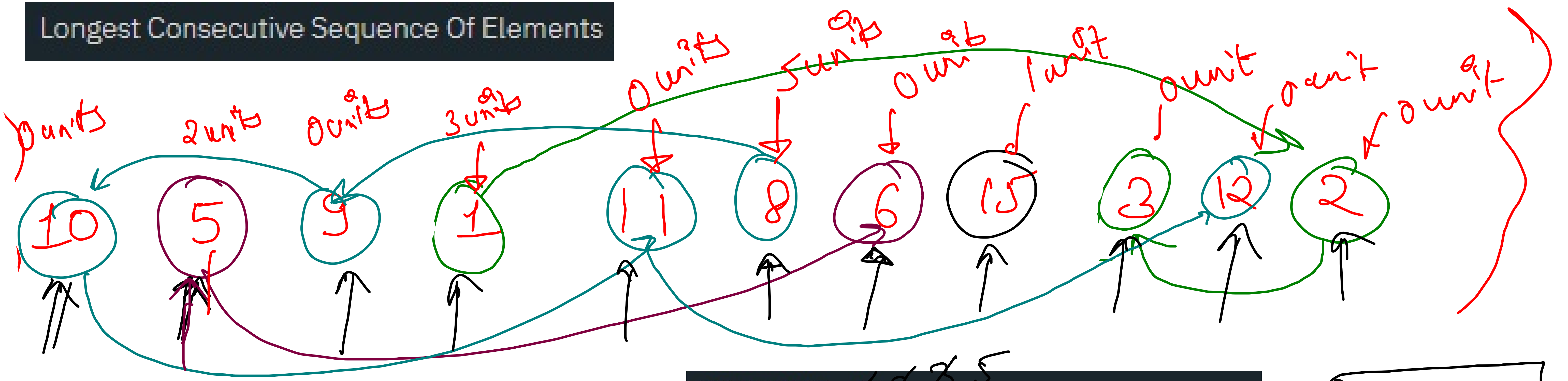
1, 2, 3, 2, 3, 5

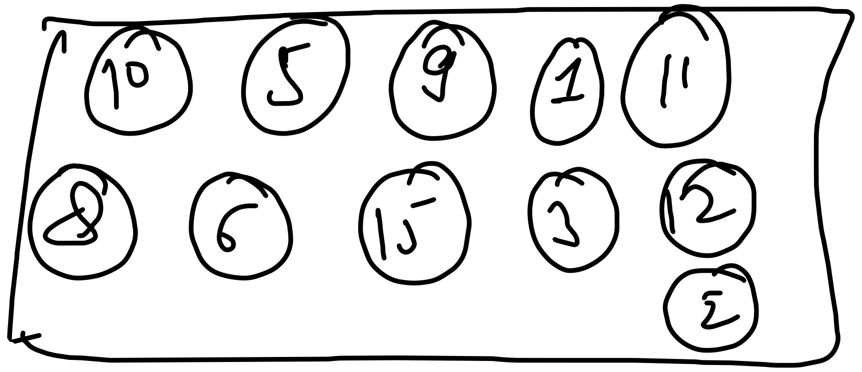


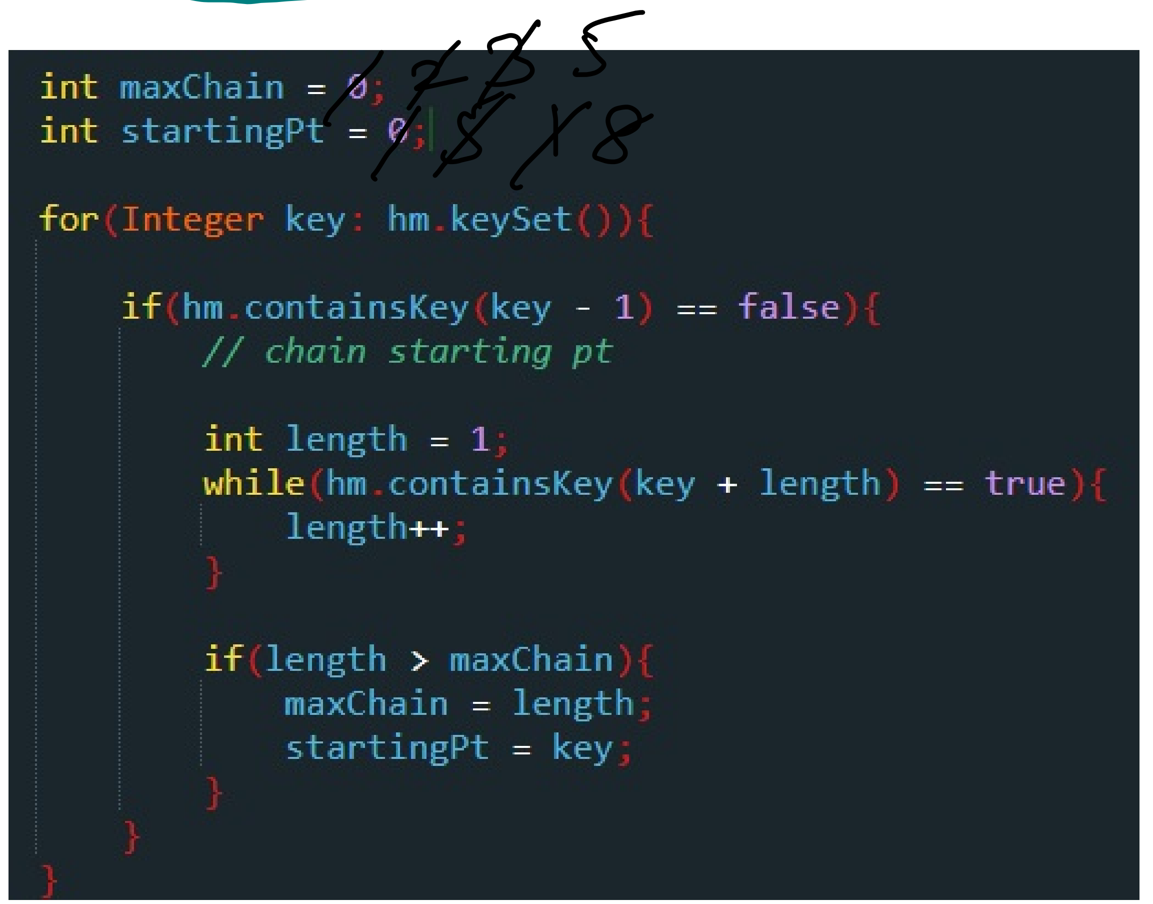


```
HashMap<Integer, Integer> hm = new HashMap<>();
for(int i=0; i<n1; i++){
    if(hm.containsKey(arr1[i])){
        hm.put(arr1[i], hm.get(arr1[i]) + 1);
    } else {
        hm.put(arr1[i], 1);
    }
}

for(int i=0; i<n2; i++){
    if(hm.containsKey(arr2[i]) && hm.get(arr2[i]) > 0){
        System.out.println(arr2[i]);
        hm.put(arr2[i], hm.get(arr2[i]) - 1);
    }

}
```

Longest Consecutive Sequence Of Elements







~~0~~ ~~1~~ ~~2~~ ~~3~~ 5

~~0~~ ~~1~~ ~~5~~ ~~1~~ 8

```
int maxChain = 0;
int startingPt = 0;

for(Integer key: hm.keySet()){

    if(hm.containsKey(key - 1) == false){
        // chain starting pt

        int length = 1;
        while(hm.containsKey(key + length) == true){
            length++;
        }

        if(length > maxChain){
            maxChain = length;
            startingPt = key;
        }
    }
}
```

✓ $\boxed{1 \to 2 \to 3}$

✓ $\boxed{8 \to 9 \to 10 \to 11 \to 12}$

✓ $\boxed{5 \to 6}$

✓ $\boxed{15}$

$\boxed{O(N)}$ Time