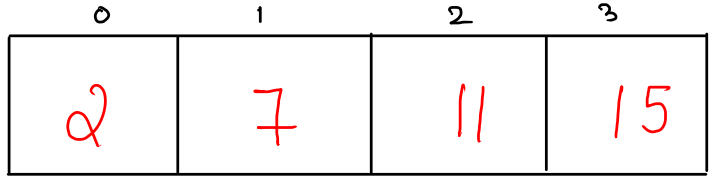
Two Sum 14



target = 9

num1 + num2 == target

num2 == target − num1

map.get(num1)

map.get(num2)

0, 1
1, 0

map

Integer, Integer

2	0
7	1
11	2
15	3

target = 9

num1 = 2, num2 = 7

num1 = 7, num2 = 2

num1 = 11, num2 = -2

num1 = 15, num2 = -6

Code

T.C = O(N)

S.C = O(N)

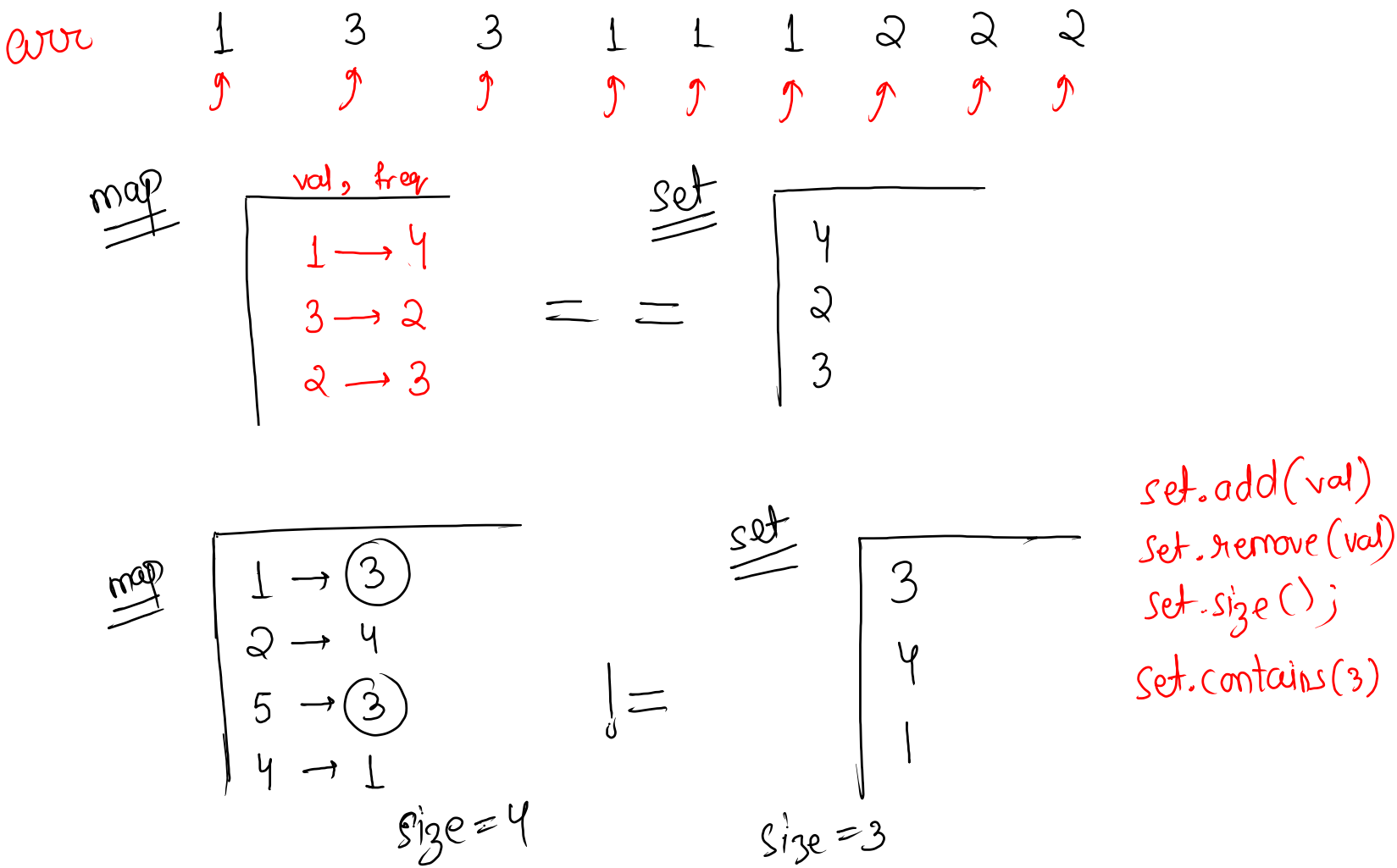
```java
public static void twoSum(int[] arr, int n, int target) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < n; i++) {
        map.put( arr[i], i );
    }

    int[] answer = new int[2];
    for (int i = 0; i < n; i++) {
        int num1 = arr[i];
        int num2 = target - num1;
        if ( map.containsKey( num2 ) ) {
            if ( i != map.get( num2 ) ) {
                answer[0] = i;
                answer[1] = map.get(num2);
                Arrays.sort(answer);
                System.out.println(answer[0] + " " + answer[1]);
                return;
            }
        }
    }
}
```

O(N)

O(N)

Unique Number of Occurrences



Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    System.out.println(uniqueOcc(arr, n));
}
public static boolean uniqueOcc(int[] arr, int n) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < n; i++) {
        if ( !map.containsKey(arr[i]) ) {
            map.put( arr[i], 1 );
        } else {
            map.put( arr[i], map.get(arr[i]) + 1 );
        }
    }

    HashSet<Integer> set = new HashSet<>();
    for (int i : map.values()) {
        set.add(i);
    }

    if ( map.size() == set.size() ) return true;
    else return false;
}
```

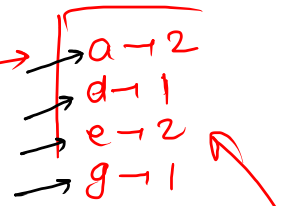
T.C = O(N)

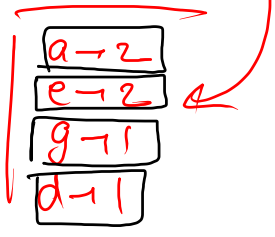
S.C = O(N)

Valid Anagram 5

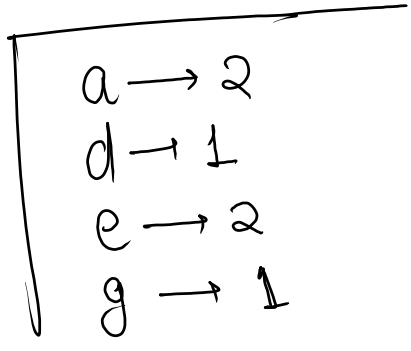
str1 = "adeega"

str2 = "dgeaea"





str1 map

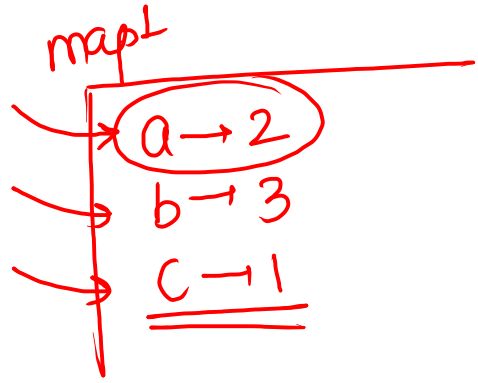


code

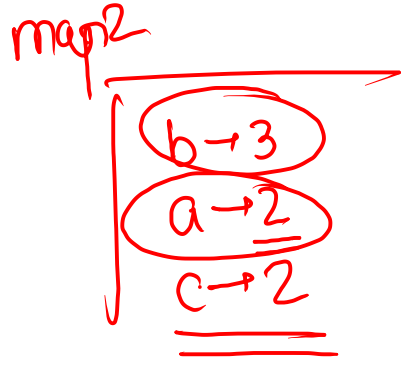
```java
public static boolean validAnagram(String str1, String str2) {
    HashMap<Character, Integer> map1 = new HashMap<>();
    for (int i = 0; i < str1.length(); i++) {
        if ( !map1.containsKey( str1.charAt(i) ) ) {
            map1.put( str1.charAt(i), 1 );
        } else {
            map1.put( str1.charAt(i), map1.get( str1.charAt(i) ) + 1 );
        }
    }

    HashMap<Character, Integer> map2 = new HashMap<>();
    for (int i = 0; i < str2.length(); i++) {
        if ( !map2.containsKey( str2.charAt(i) ) ) {
            map2.put( str2.charAt(i), 1 );
        } else {
            map2.put( str2.charAt(i), map2.get( str2.charAt(i) ) + 1 );
        }
    }

    for (Map.Entry<Character, Integer> e : map1.entrySet()) {
        char ch = e.getKey();
        int freq = e.getValue();
        if ( !map2.containsKey(ch) ) {
            return false;
        }
        if ( map2.get( ch ) != map1.get( ch ) ) {
            return false;
        }
    }
    return true;
```

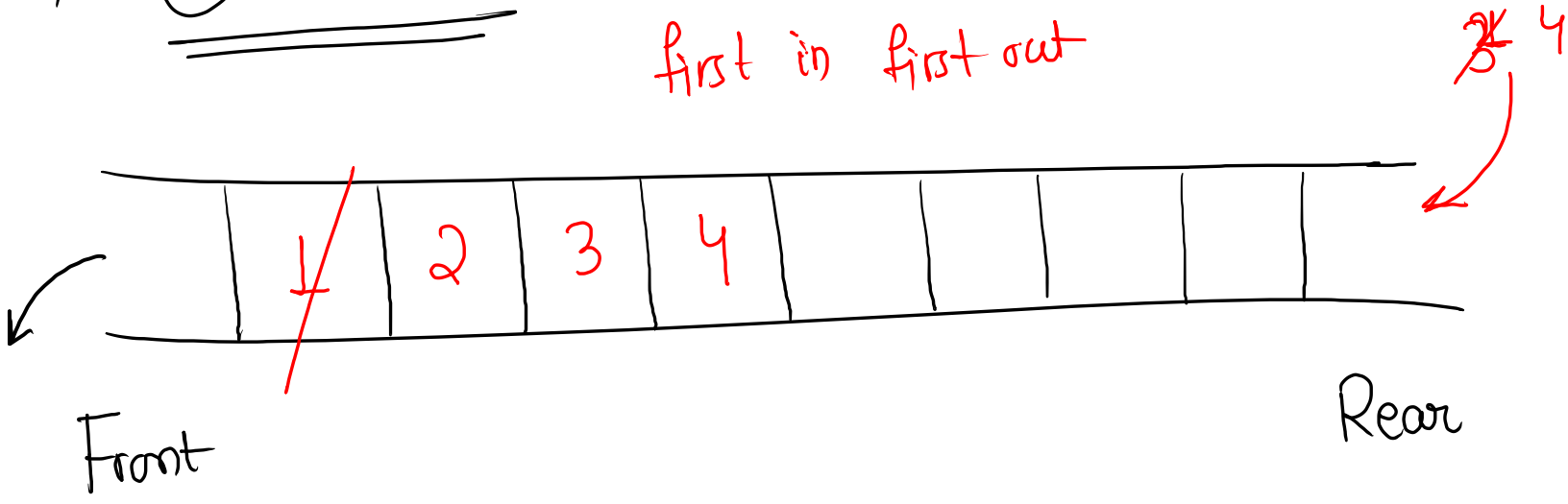


ch = a b c
freq = 2 3 1



⇒ Queue (FIFO)

first in first out



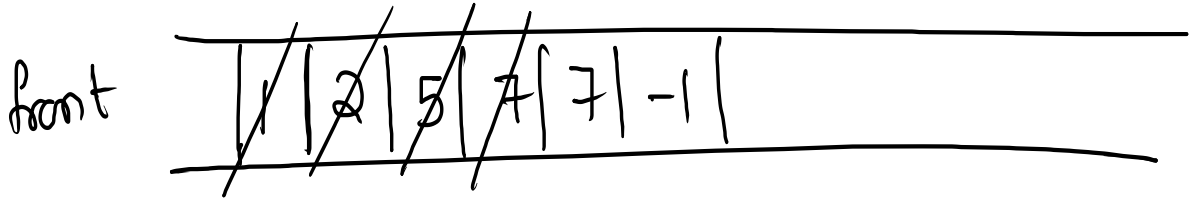
always add elements from rear
and remove elements from front

Syntax :-

```
Queue<Integer> que = new LinkedList<>();
```

Inbuilt functions

- `que.add(x);` // to add an element in rear
- `que.remove();` // to remove element from front
- `que.poll();`
- `que.peek();` // return value at front
- `que.size(), que.isEmpty();`



que.add(1)
add(2)
add(5)
add(7)

→ que.peek() → 1

print(que.poll()) → 1

print(que.poll()) → 2

que.add(7)

que.add(-1)

que.peek() → 5

print(que.poll()) → 5

(que.poll()) → 7