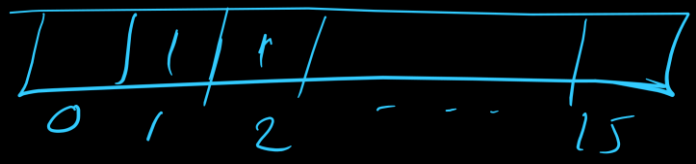
Q.1. arr [1, 1, 2, 2, 15, 15, 15] $\xrightarrow{3}$ 15 * 4 = 60 byte

Counts - neg X

- Space = O(n)



Bitwise Operator

2 bits same result = 0

XOR = ^ =

a = 5 → 101

b = 5 → 101

000

A	B	R
T	T	F
T	F	T
F	T	T
F	F	F

⇒ 1 1 = 0 ←

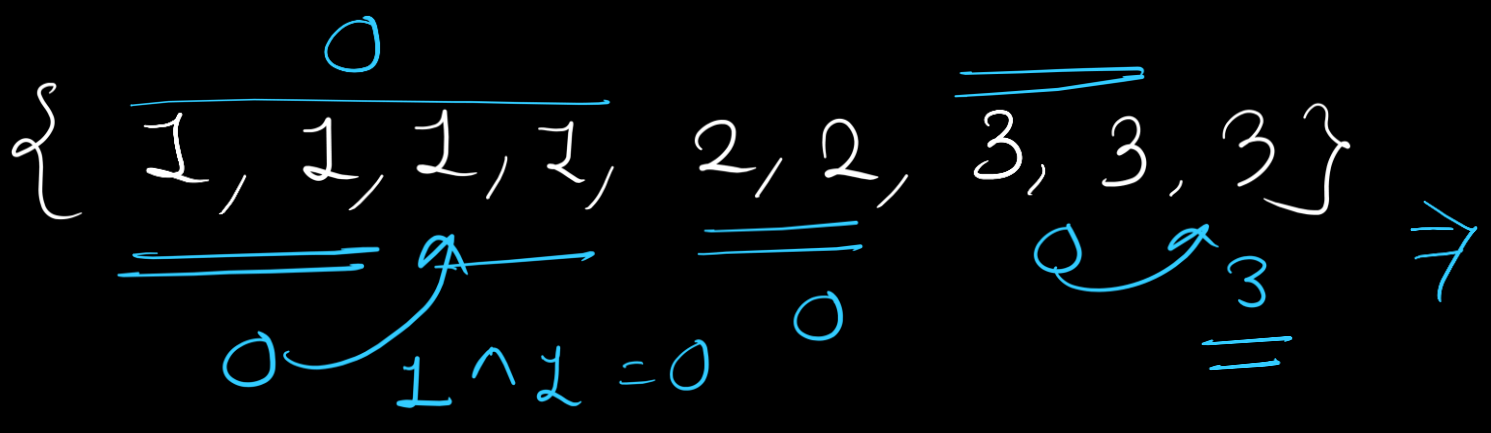
1 0 = 1

0 1 = 1

0 0 = 0 ←

XOR = Agar aap ki 2 value same hai to result 0 ayega

else = Any other number



even = 8 = 2 + 2 + 2 + 2

odd = 7 = 2 + 2 + 2 + 1

0 ans

$0 \wedge x = x$

(zero)

$= 0 \wedge 10$

1010

1010

0000

$0000 \leftarrow 0$

1010

0000

$1010 = 10$

$\{1, 1, 2, 2, 2\}$

$0 \wedge 1 \quad 1 \wedge 1 \quad 0 \wedge 2 \quad 2 \wedge 2 \quad 0 \wedge 2$

$1 \quad 0 \quad 2 \quad 0 \quad 2$

$XR = 0 \wedge 1 \wedge 0 \wedge 2 \wedge 0 \wedge 2$

$ans = 2$

$\{10, 10, 20, 20, 30, 10, 30\}$

$0 \wedge 10 \quad 10 \wedge 10 \quad 20 \wedge 0 \quad 20 \wedge 20 \quad 0 \wedge 30 \quad 30 \wedge 10$

$10 \quad 0 \quad 20 \quad 0 \quad 30 \quad 30 \wedge 10$

$XOR = 0 \wedge 10 \wedge 0 \wedge 20 \wedge 0 \wedge 30$

$30 \wedge 10$

$xor = xor \wedge arr[i]$

$30 \wedge 30 \wedge 10$

$30 \wedge 30$

$0 \wedge 10 = 10$

```java
class Solution {
    int getSingle(int arr[]) {
        int ans = 0;
        for(int i = 0; i < arr.length; i++){
            ans = ans ^ arr[i];
        }
        return ans;
    }
}
// } Driver Code Ends
```

Majority Element

Difficulty: **Medium** Accuracy: **27.82%** Submissions: **657K+** Points: **4**

Given an array **arr**. Find the majority element in the array. If no majority exists, return -1.

> A majority element in an array is an element that appears **strictly** more than **arr.size()/2 times** in the array.

Examples:

Input: arr[] = [3, 1, 3, 3, 2]
Output: 3
Explanation: Since, 3 is present more than n/2 times, so it is the majority element.

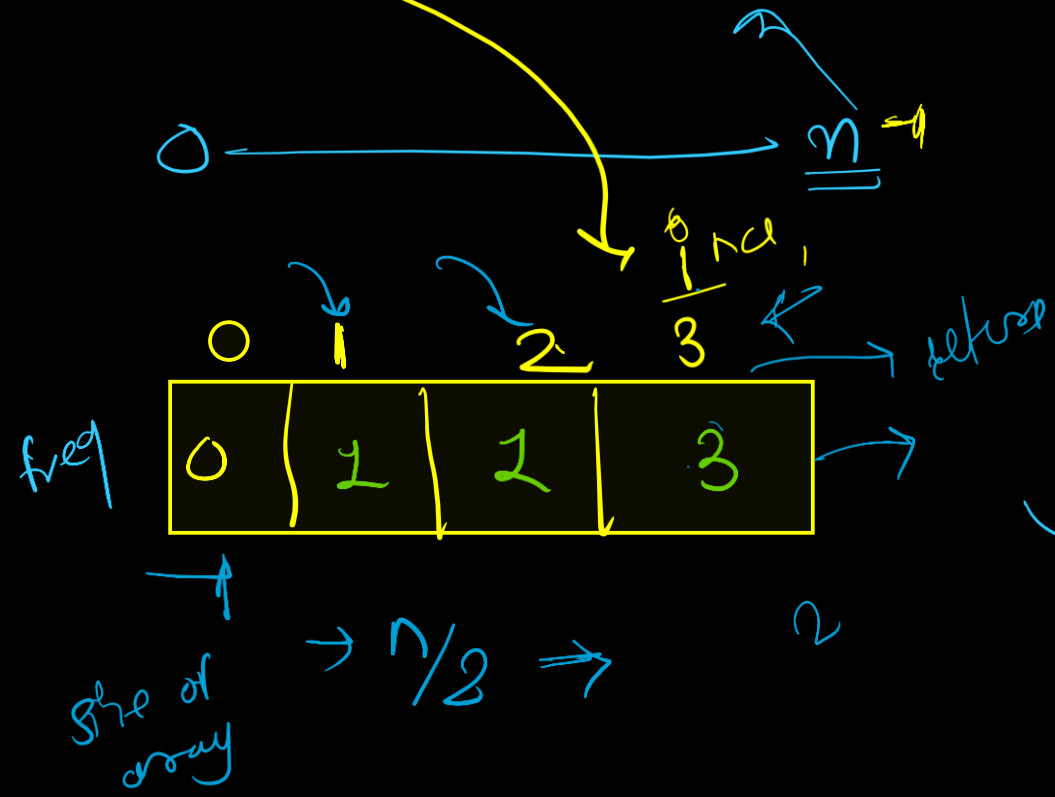
Input: arr[] = [7]
Output: 7
Explanation: Since, 7 is single element and present more than n/2 times, so it is the majority element.

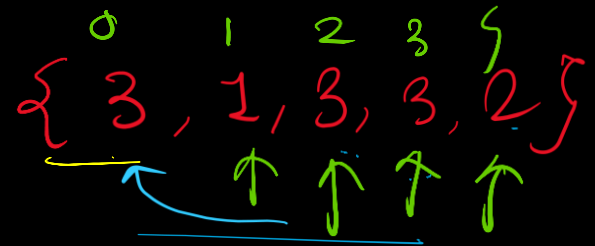
majority => arr(size)/2

{ 3, 1, 3, 3, 2 } => Hashmap → Array →

0 → n+1

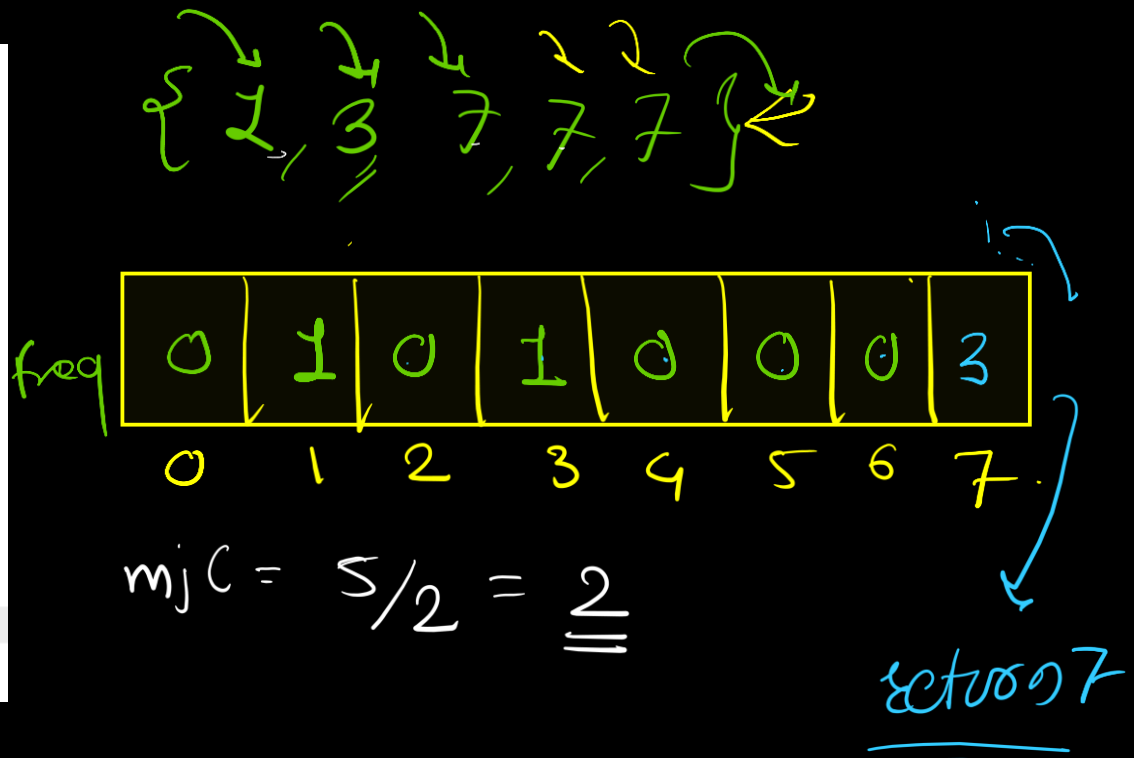
3 + 1 = 4





→ freq[arr[i]]++

```java
class Solution {
    static int majorityElement(int arr[]) {
         int maxi = arr[0];
        for(int i = 1; i < arr.length; i++){
            maxi = Math.max(arr[i], maxi);
        }
        int freq[] = new int[maxi + 1];
        for(int i=0; i < arr.length; i++){
            freq[arr[i]]++;
        }
        int majorityCount = arr.length/2;
        for(int i = 0; i < freq.length; i++){
            if(freq[i] > majorityCount)return i;
        }
        return -1;
    }
}
```



Kth Smallest

Difficulty: **Medium** Accuracy: **35.17%** Submissions: **649K+** Points: **4**

Given an array **arr[]** and an integer **k** where k is smaller than the size of the array, the task is to find the **k^{th} smallest** element in the given array.

Follow up: Don't solve it using the inbuilt sort function.

Examples :

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3
Output: 7
Explanation: 3rd smallest element in the given array is 7.

Input: arr[] = [2, 3, 1, 20, 15], k = 4
Output: 15
Explanation: 4th smallest element in the given array is 15.

Expected Time Complexity: O(n+(max_element))
Expected Auxiliary Space: O(max_element)

kth position

{7, 10, 4, 3, 20, 15}

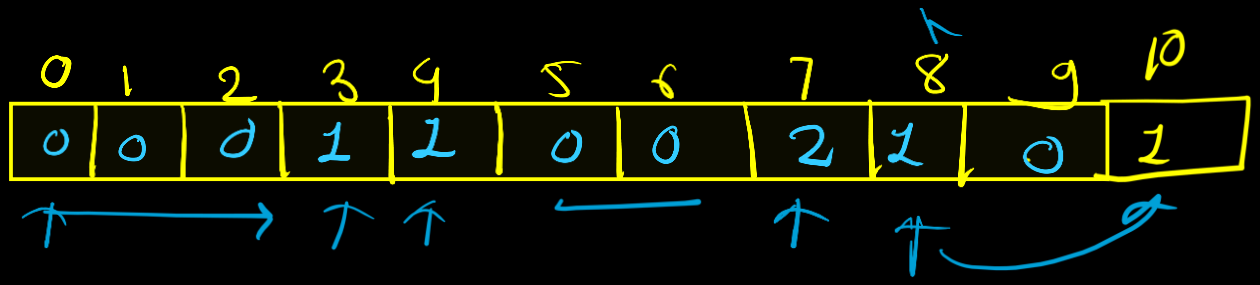
k=3

15

{3, 4, 7, 10, 15, 20}

✓ Bubble sort
select^n sort
insert^n sort

Count Sort

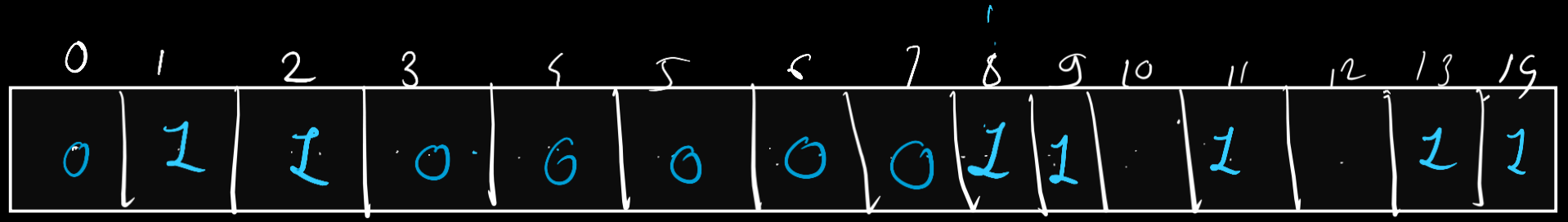


- K=0

K=4

return i

{1, 2, 8, 9, 11, 13, 14}

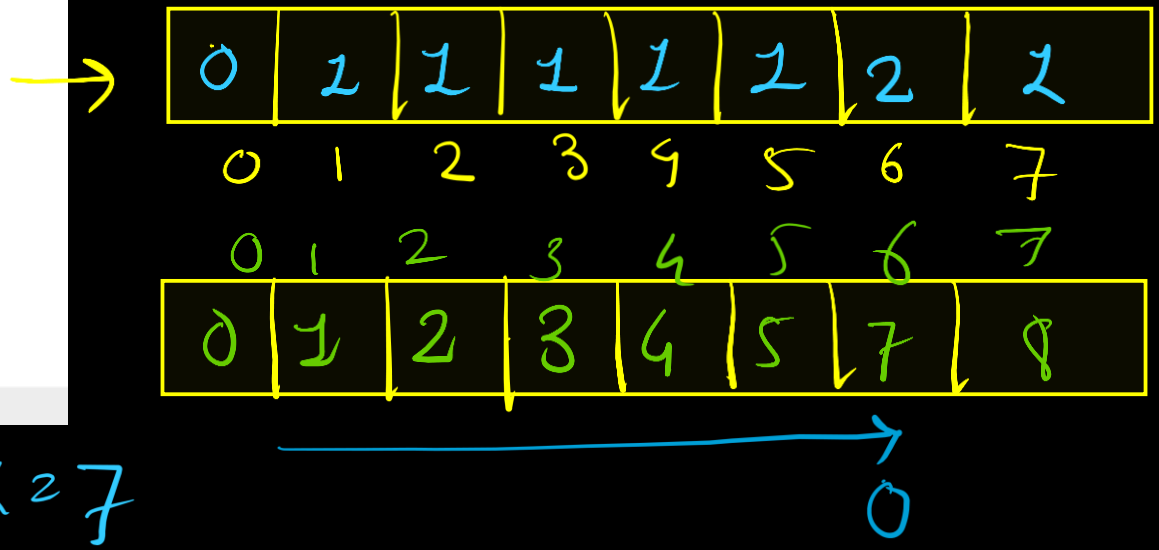


K: 4 3 2 2 2 2 2 2 1 0

→ return i

```java
class Solution {
    public static int kthSmallest(int[] arr, int k) {
        int maxi = arr[0];
        for(int i = 1; i < arr.length; i++){
            maxi = Math.max(arr[i], maxi);
        }
        int freq[] = new int[maxi + 1];
        for(int i=0; i < arr.length; i++){
            freq[arr[i]]++;
        }
        for(int i = 0; i < freq.length; i++){
            k = k - freq[i];
            if(k == 0)return i;
        }
        return -1;
    }
}
```

{ 2, 4, 3, 7, 5, 7, 6, 6 }



k = 7

{ 1, 2, 3, 4, 5, 6, 6, 7 }