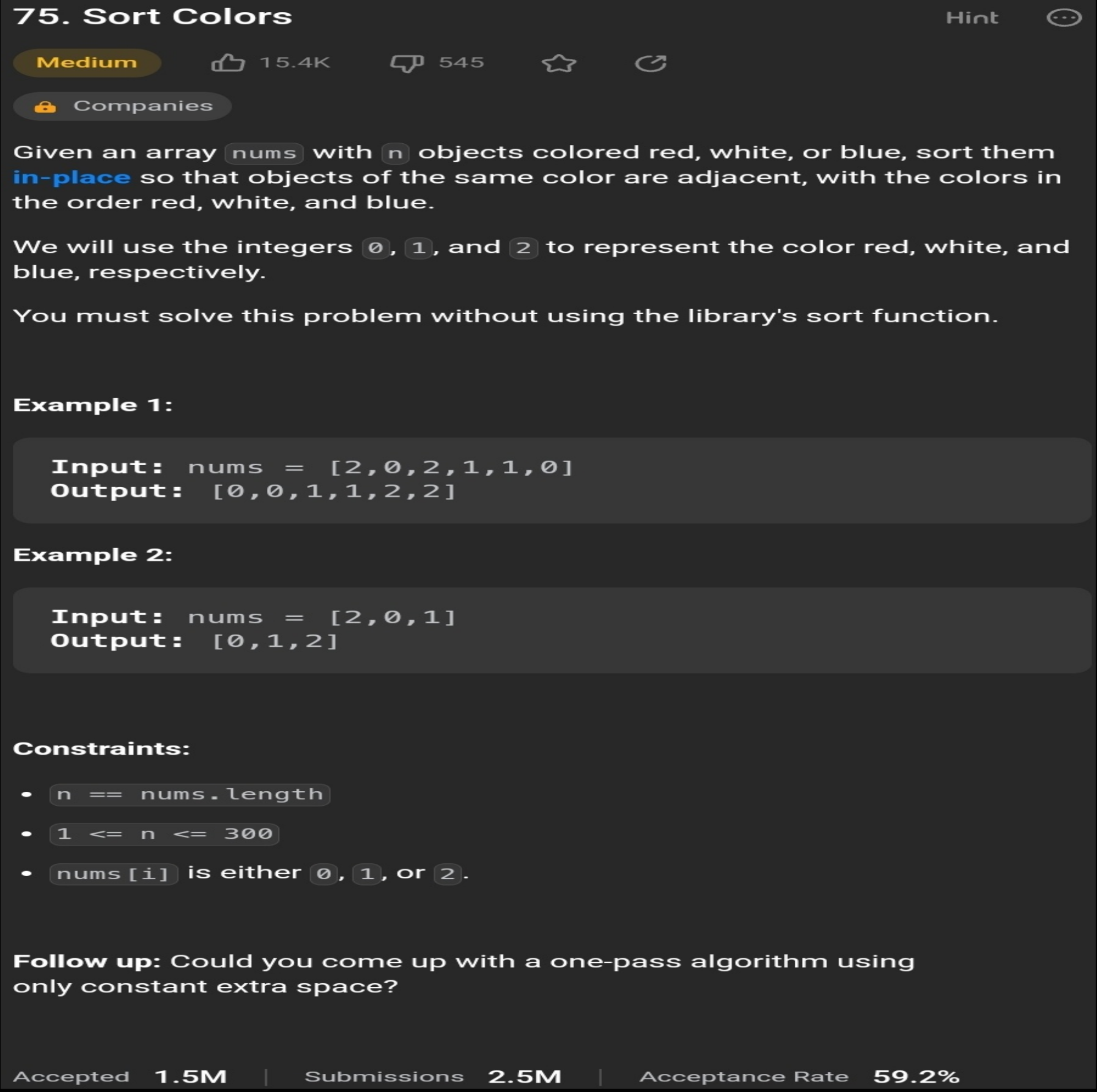


75. Sort Colors

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?

Accepted 1.5M | Submissions 2.5M | Acceptance Rate 59.2%

Approach 1:

Sort

$O(n\log n)$

Approach 2:

Using three frequency variables

0	2	1	0	1	2	0	1	1	2

Red =0

white = 0
Blue = 0

Scan the array once and accordingly increase the variable values.

After scanning the array,

Red = 3
white = 4
Blue = 3

now just run 3 while loops and fill the array with required no. of 0's, 1's & 2's.

i.e. i = 0

```
while (Red--) nums[i++] = 0
while (white--) nums[i++] = 1
while (Blue--) nums[i++] = 2
```

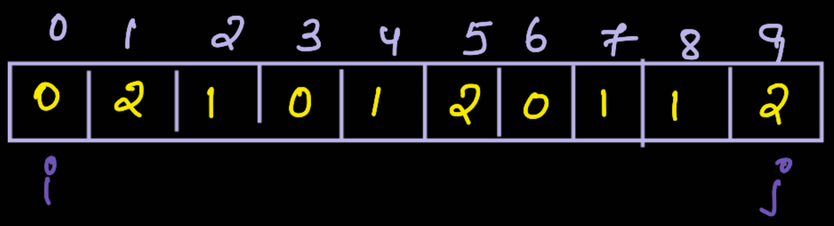
O(n) but requires two passes on the array.

Approach 3:

in two passes

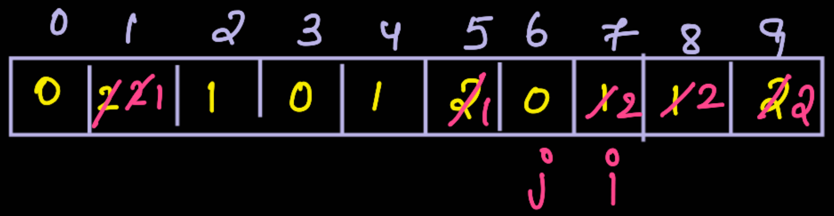
1st pass: make all 2's to come to end.

2nd pass: make all 1's to come to end.



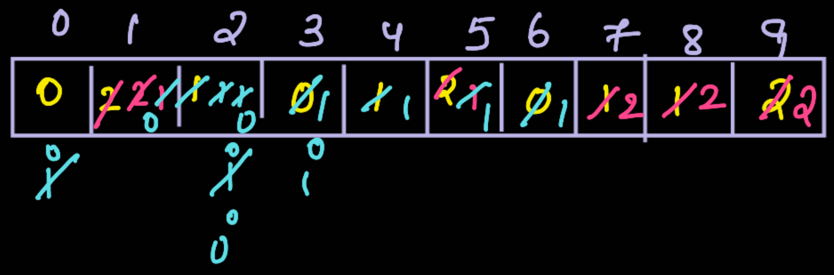
1st pass:

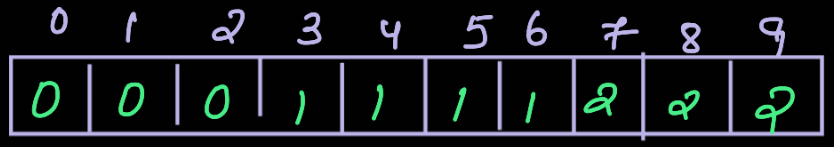
```
while (i ≤ j)
    if (nums[i] == 0 || nums[i] == 1) i++
else   Swap(nums[i], nums[j--])
```



2nd pass:

```
i = 0
while(i ≤ j)
    if(nums[i] == 0)    i++;
    else   Swap(nums[i], nums[j--])
```

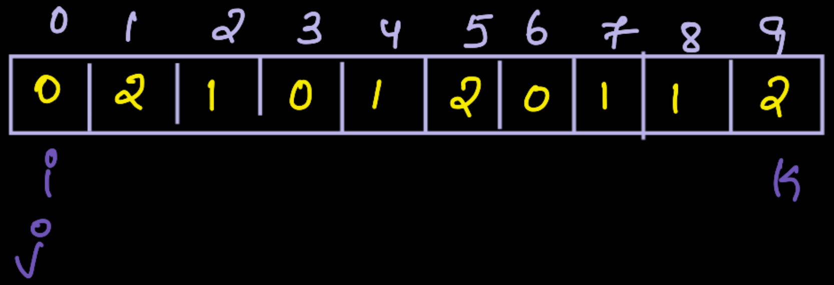




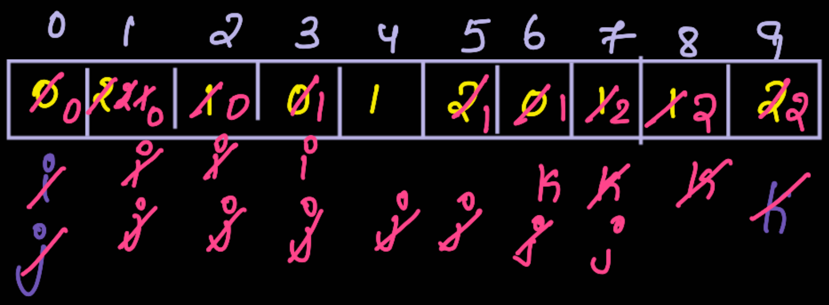
$O(n)$ but requires two passes.

Approach 4:

Using 3 pointers and in single pass.



```
while(j ≤ k)
    if(nums[j] == 0)
    swap(nums[i++], nums[j++])
else if(nums[j]==1)
        j++
else
    swap(nums[j], nums[k--])
```



0	1	2	3	4	5	6	7	8	9
0	0	0	1	1	1	1	2	2	2

O(n) and in
Single pass