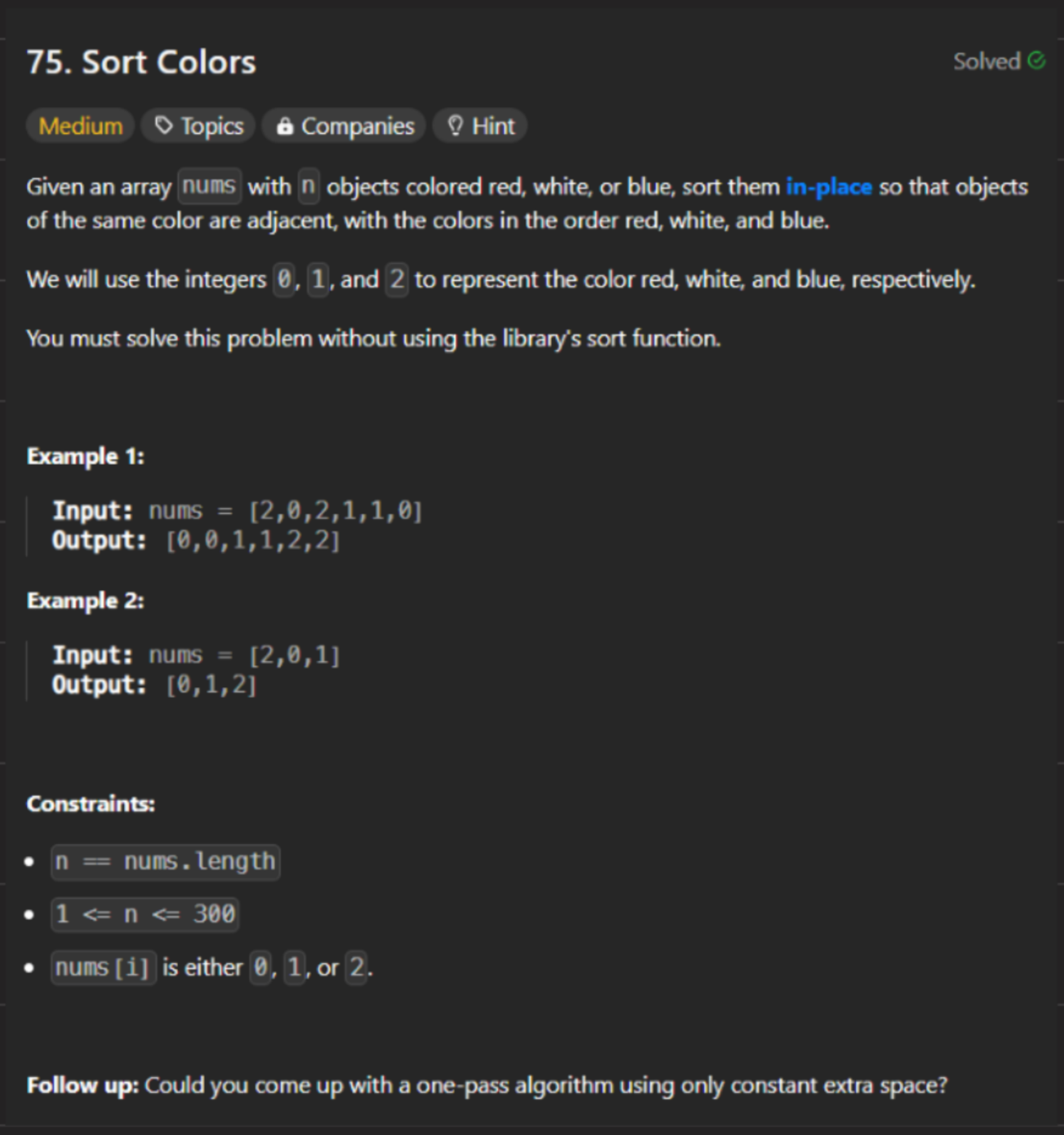
Sort an array of 0's, 1's, 2's

↳ Dutch National Flag Algorithm



75. Sort Colors

Solved

Medium · Topics · Companies · Hint

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?

Algorithm

① This algorithm contains 3 pointers,
i.e ... (low, mid, high)

② And 3 main rules,

* arr[0 ... low − 1] → contains 0

* arr[low .... mid − 1] → contains 1

* arr[high+1 .... n−1] → contains 2

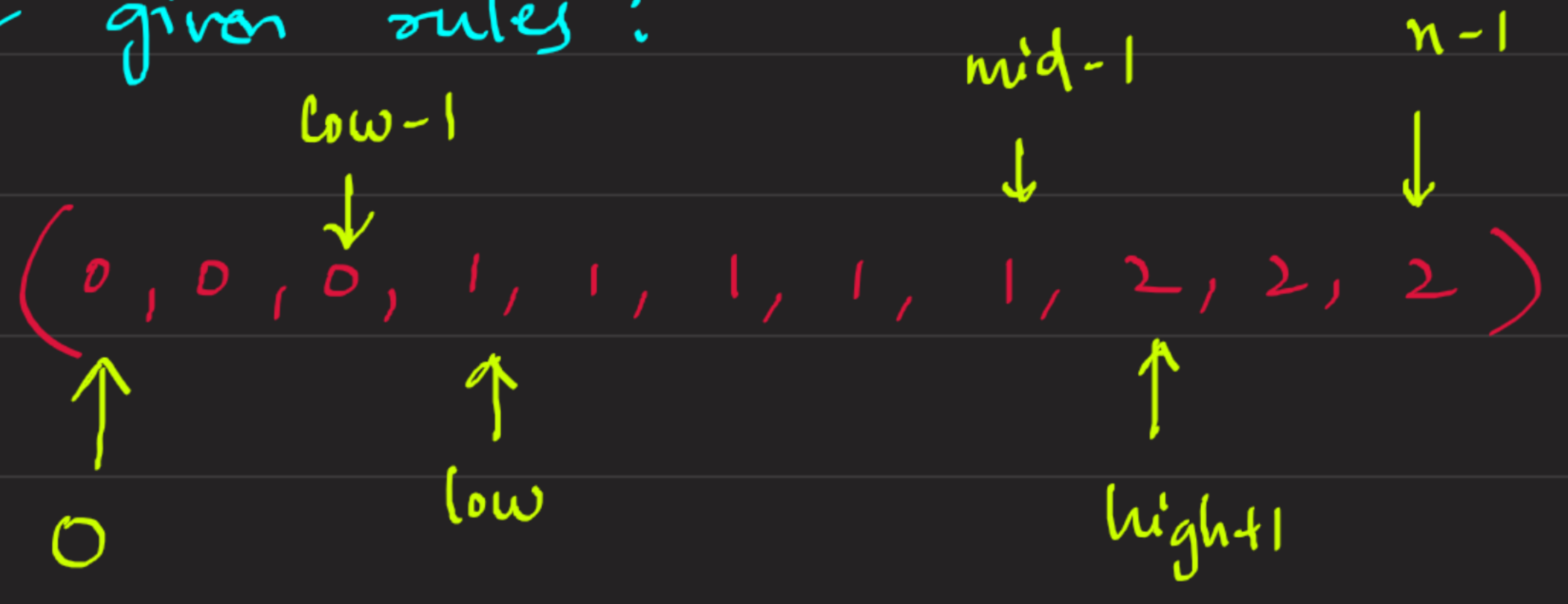
* Whatever array you are given, make them into above structure

input array : (1, 0, 2, 0, 0, 1, 2, 2, 1, 1, 1)

↓

final array : (0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2)

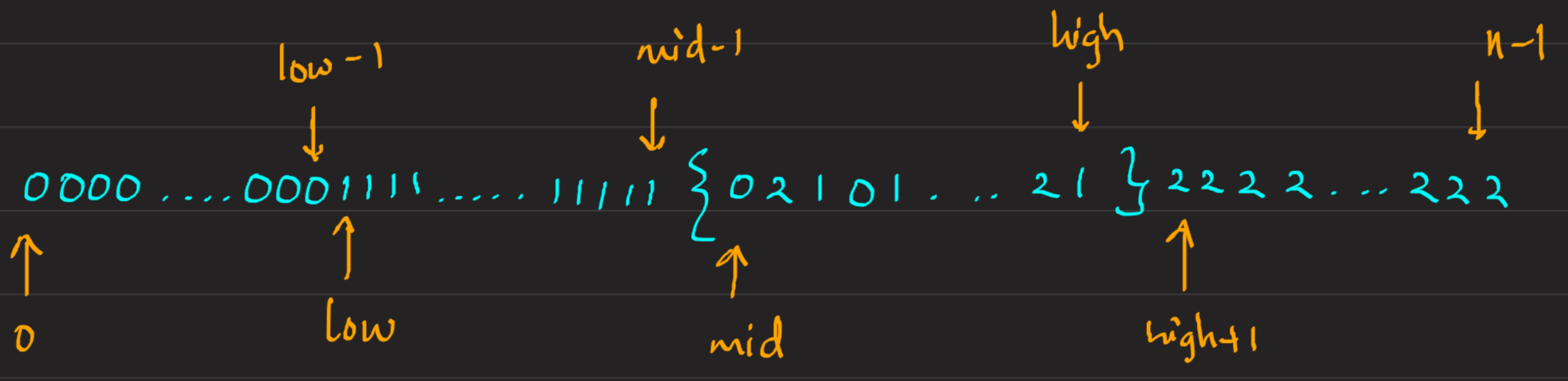
place pointers as per given rules :



* If you observe carefully, we don't find (mid to high) portion,

that is because, this is already a sorted array

Generalizing

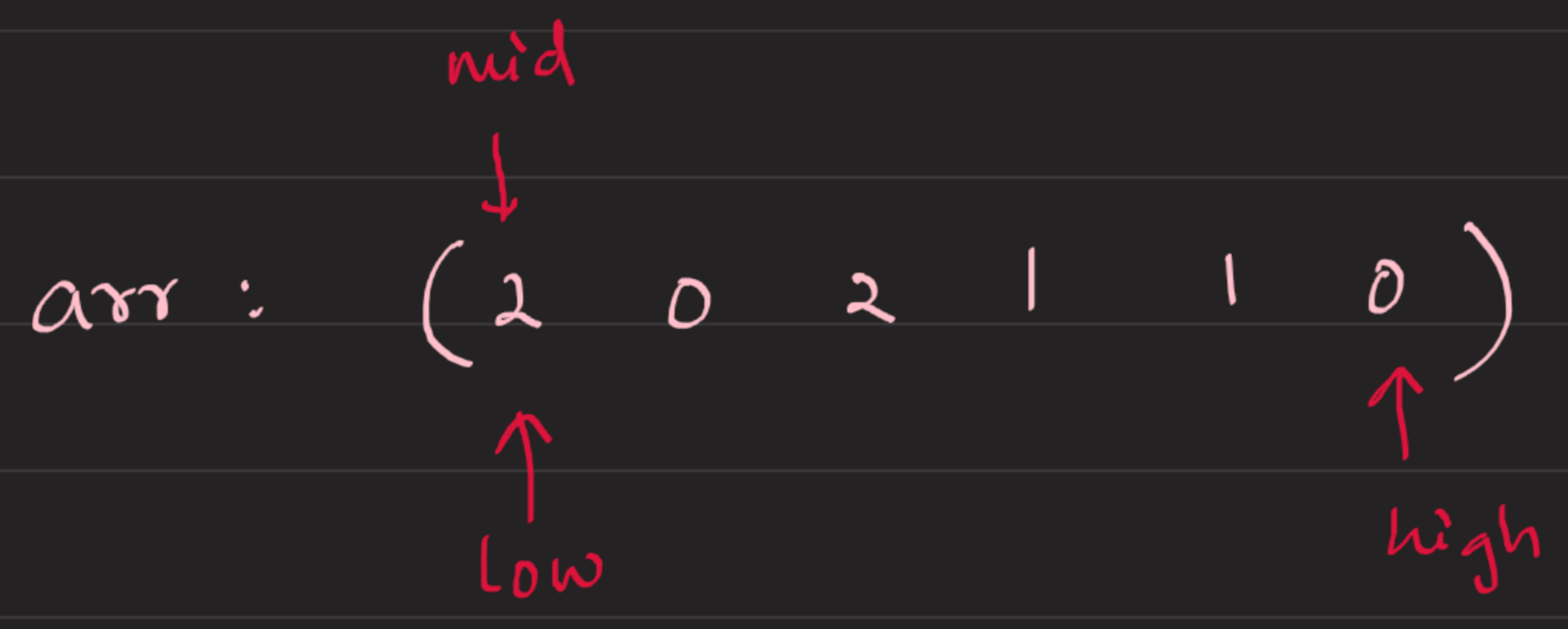


We have to clear that (mid to high portion) → then our array will be sorted

// The unsorted portion lies b/w (mid, high)

* We have to arange these mid to high values, so, that our top 3 rules won't break

* In our case, we assume whole array is unsorted, so we place pointers accordingly



After all the values move to the original places,
↓
(high, mid) will cross
↳ so erases

Observations

① Case - I

if (arr[mid] == 0)
↳ ① we will swap(arr[low], arr[mid]]
② increment both mid & low

② Case - II

if (arr[mid] == 1)
↳ we will just increment mid

③ Case - III

if (arr[mid] == 2)
↳ ① swap(arr[mid], arr[high])
② decrement high

Dry Run

$$arr[\,]:(\underset{low}{\overset{mid}{0}}\ 1\ 1\ 0\ 1\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

① $arr[mid] == 0$ (case - I)

$$arr[\,]:(0\ \underset{low}{\overset{mid}{1}}\ 1\ 0\ 1\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

② $arr[mid] == 1$ (case - II)

$$arr[\,]:(0\ \underset{low}{1}\ \overset{mid}{1}\ 0\ 1\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

③ $arr[mid] == 1$ (case - II)

$$arr[\,]:(0\ \underset{low}{1}\ 1\ \overset{mid}{0}\ 1\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

④ $arr[mid] == 0$ (case - I)

$$arr[\,]:(0\ \underset{low}{\overset{0}{\cancel{1}}}\ 1\ \overset{mid}{\underset{1}{\cancel{0}}}\ 1\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

$$arr[\,]:(0\ \overset{0}{\cancel{1}}\ \underset{low}{1}\ \underset{1}{\cancel{0}}\ \overset{mid}{1}\ 2\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

⑤ $arr[mid] == 1$ (case - II)

$$arr[\,]:(0\ 0\ \underset{low}{1}\ 1\ 1\ \overset{mid}{2}\ 1\ 2\ 0\ 0\ \underset{high}{0})$$

⑥ $arr[mid] == 2$ → (case - III)

$$arr[\,]:(0\ 0\ \underset{low}{1}\ 1\ 1\ \overset{mid}{\underset{0}{\cancel{2}}}\ 1\ 2\ 0\ 0\ \underset{high}{\overset{2}{\cancel{0}}})$$

$$arr[\,]:(0\ 0\ \underset{low}{1}\ 1\ 1\ \overset{mid}{\underset{0}{\cancel{2}}}\ 1\ 2\ 0\ \underset{high}{0}\ \overset{2}{\cancel{0}})$$

⑦ $arr[mid] == 0$ → case (I)

$$arr[\,]:(0\ 0\ \underset{low}{\overset{0}{\cancel{1}}}\ 1\ 1\ \overset{mid}{\underset{1}{\cancel{0}}}\ 1\ 2\ 0\ \underset{high}{0}\ 2)$$

$$arr[\,]:(0\ 0\ \overset{0}{\cancel{1}}\ \underset{low}{1}\ 1\ \underset{1}{\cancel{0}}\ \overset{mid}{1}\ 2\ 0\ \underset{high}{0}\ 2)$$

likewise --> it goes

when mid & high crosses our array goes sorted

T.C : O(n)

S.C : O(1)