Sort colors

Problem Statement:
We are given three set of colors that we wish to sort on order.

TIP: 3 colors means n-1 number mapping starting from 0 that is 0,1,2.

↳ Red, green, blue
↓ ↓ ↓
0 1 2

We are given colors mapped to a number which needs to be sorted and then presented as an output

[1,1,0,2,0,2] ← Input (Given Input)

[0,0,1,1,2,2] ← output (Expected Output)

1) MergeSort/QuickSort
↳ $\Theta(n \log n)$

Optimized approach

2) Two pointers
- p_0 → to store 0's in extreme left
- p_2 → to store 2's in extreme right

$p_0 = 0$ (index value)

$p_2 = 5$ (index value)

0	1	2	3	4	5
2	0	2	1	1	0

↑

n → size of an array

In 2 pointer since left pointer takes care of lowest number whereas right pointer takes care of highest number so, in the middle section the third value will be automatically placed since it does not have any other location to go.

Since only 1 while loop used that traverses through the whole list then time complexity is equivalent to O(n)

$\Theta(n)$

p_0, curr = 0

$p_2 = n-1$

while curr <= p2:

traversing through the whole list from starting cursor at 0th index up to last index being pointed by p2 initially.

Please note that nums represent the list containing the respective integer color mapped value

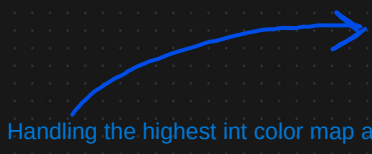
if nums(curr) == 0:

swap(nums(curr), nums(p0))

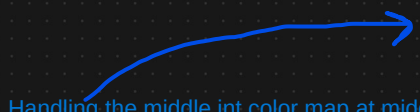
p0 += 1

curr += 1

Handling the lowest int color map at left extreme




```
elif nums(curr) == 2:
    swap(num(curr), nums(p2))
    p2 -= 1
```




```
else:
    curr += 1
```

return nums
