Move Numbers

#two-pointers #uber_interview_question

LinkedIn: Md. Ziaul Karim

Find me here:

Problem Statement:

Given an array `nums` consisting of integers, implement a function `moveNumbers(nums: List[int]) -> List[int]` to rearrange the elements of the array. The goal is to move all non-zero elements to the beginning of the array while maintaining their original order. The remaining positions in the array should be filled with zeros.

Consequently, the function should return the modified array.

Function Signature:

```python
def moveNumbers(nums: List[int]) -> List[int]:
    # Implementation goes here
```

Input:

- An array `nums` where 0 <= len(nums) <= 10^4.
- Each element `nums[i]` is an integer, where $-2^{31} <= nums[i] <= 2^{31} - 1$.

Output:

- The function should return a modified array as described above.

Example:

Input:

```
moveNumbers([0, 3, 0, 12, 1])
```

Output:

```
[3, 12, 1, 0, 0]
```

Input:

```
moveNumbers([3, 0, 1, 21, 12, 0, 0, 2, 13])
```

Output:

```
[3, 1, 21, 12, 2, 13, 0, 0, 0]
```

Note:

- The solution should perform the rearrangement in-place without using additional data structures.
- The relative order of non-zero elements should be preserved.
- The function should handle edge cases, such as an empty array.

Solution 1: Using `pop()` and `insert()` method subsequently

you first remove the element at index `i` from the list using `pop`, and then you insert it at index `j` using `insert`. This process effectively moves the non-zero element to the desired position in the array.

It's worth noting that this approach has a time complexity of $O(n^2)$ due to the use of `pop(i)` inside the loop, where `n` is the length of the input list. If you want to improve the time complexity, you may consider an alternative approach, such as maintaining two pointers to iterate through the list in a single pass, avoiding the need for repeated removal and insertion operations.

```python
def moveNumbers(li):
        i=0
        j=0
        while i<len(li):
            if li[i]!=0:
                val = li.pop(i)
                li.insert(j,val)
                j+=1
            i+=1
        return li
if __name__ == '__main__':
        li = [0, 3, 0, 12, 1] #[3,12,1,0,0]
        result = moveNumbers(li)
        print(result)
        li = [3,0,1,21,12,0,0,2,13] #[3, 1, 21, 12, 2, 13, 0, 0, 0]
        result = moveNumbers(li)
        print(result)
```

Output

```
[3, 12, 1, 0, 0]
[3, 1, 21, 12, 2, 13, 0, 0, 0]
```

Solution 2: Two pointers solution ✅

Time Complexity:

- The solution performs a single pass through the list, so the time complexity is $O(n)$, where n is the length of the input list.

Space Complexity:

- The solution uses only a constant amount of extra space, regardless of the input size. Therefore, the space complexity is $O(1)$.

Note:

- This solution optimally achieves the desired rearrangement without using the `pop` and `insert` operations, resulting in improved time complexity compared to the original solution.

```python
def moveNumbers(li):
    left=0
    right=1
    while right<len(li):
        if li[left]!=0:
            left+=1
        li[left],li[right]=li[right],li[left]
        right+=1
    return li

if __name__ == '__main__':
    li = [0, 0, 3, 0, 12, 1, 0] #[3, 12, 1, 0, 0, 0, 0]
    result = moveNumbers(li)
    print(result)
    li = [3,0,1,21,12,0,0,2,13] #[3, 1, 21, 12, 2, 13, 0, 0, 0]
    result = moveNumbers(li)
    print(result)
```

Output

```
[3, 12, 1, 0, 0]
[3, 1, 21, 12, 2, 13, 0, 0, 0]
```

End