ASSIGNMENT 2 : Partner Assignment 1 Review

1. Paraphrase the problem in your own words

The problem is about moving zeros to the end of a list without changing the order of the non zero elements.

2. Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

```
My example:
Input: [0, 7, 0, 0, 8, 9]
Output: [7, 8, 9, 0, 0, 0]

Partner's example:
Taking the first example he did which is:
Input : [0,1,0,3,12]
Output: [1, 3, 12, 0, 0]

        1.      i = 0 → value = 0
                - pop(0) removes the first 0, list becomes [1, 0, 3, 12], then append(0) makes it [1, 0, 3, 12, 0]
                - length_nums becomes 4, i = 0
        2.      i = 0 → value = 1
                - not zero → i += 1
        3.      i = 1 → value = 0
                - pop(1) removes 0, list becomes [1, 3, 12, 0], then append(0) → [1, 3, 12, 0, 0]
                - length_nums becomes 3, i = 1
        4.      i = 1 → value = 3
                - not zero → i += 1
        5.      i = 2 → value = 12
                - not zero → i += 1
        6.      i = 3 → loop ends because i >= length_nums
```

3. Copy the solution your partner wrote.

In [ ]:
```python
from typing import List

def move_zeros_to_end(nums: List[int]) -> List[int]:
    # TODO
    length_nums = len(nums)
    i = 0

    while i < length_nums:
        if nums[i] == 0:
            nums.pop(i)
            nums.append(0)
            length_nums -= 1
        else:
            i += 1
    return nums

list1 = [0, 1, 0, 3, 12]
list2 =[4, 0, 5, 0, 0, 6]
print(move_zeros_to_end(list1))
print(move_zeros_to_end(list2))
```

```
[1, 3, 12, 0, 0]
[4, 5, 6, 0, 0, 0]
```

4. Explain why their solution works in your own words.

My partner’s solution works because it iterates through the list once, and every time it encounters a zero, it removes (pop) that zero from the list and adds it to the end (append(0)). This process ensures that: • All non-zero elements are preserved in their original order • All zeros are moved to the end of the list

By using a while loop with a manual index (i) and keeping track of the list length (length_nums), the function dynamically adjusts for the shrinking list as items are removed.

5. Explain the problem’s time and space complexity in your own words.

The pop(i) operation removes an item from the middle of the list. This takes O(n) time because all the items after index i need to shift left. This pop() happens every time a zero is found, so in the worst case, there could be many zeros. Since the loop runs up to n times, and each pop() can cost up to O(n), the total worst-case time is $O(n^2)$. The function doesn’t use extra memory or storage — it just moves things around inside the same list. So space complexity is O(1).

6. Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

The solution correctly solves the problem — it moves all 0s to the end while keeping the order of non-zero elements.The use of a while loop and conditional logic is easy to follow. The function modifies the list in place, which saves memory (space complexity is O(1)).

I think the issue is if the list is large. It will then become an expensive operation for each pop in the middle. So it will be inefficient for large lists. The code should run through once the list and take all the non zero elements, put it in another increased list accordingly. The code should determine how many zeros were found add all that to the end of the new list.

REFLECTION

Please write a 200 word reflection documenting your process from assignment 1, and your presentation and review experience with your partner at the bottom of the Jupyter Notebook under a new heading "Reflection." Again, export this Notebook as pdf.

Working on Assignment 1 helped me build a deeper understanding of how algorithmic thinking and abstract data types can be used to solve problems efficiently. I was assigned Question 1, where I had to find the first duplicate in a list. At first, I was unsure how to approach it, but after learning about sets as an abstract data type, I realized they could help track seen values with constant lookup time. This significantly simplified my code and improved performance. I also practiced testing the solution with different input lists and analyzing the time and space complexity to understand the trade-offs of different approaches.

For the peer review portion, I was assigned to evaluate my partner's solution to Question 3 (moving all zeros to the end of a list). My partner’s logic worked correctly, though it used a pop/append technique that I learned has higher time complexity. I walked through his solution, created my own test case, and suggested an alternative route. It's a great way to reinforce the learning. It also gave me the chance to practice explaining algorithms clearly, which I think is just as important as writing the code itself.