Move Zeros Algorithm

Give DSA notes for Move zero notes to end based on this code

```
public void moveZeroes(int[] nums) {
    int j = 0; // Index for the next non-zero element
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] != 0) {
            // Swap elements only if necessary
            if (i != j) {
                nums[j] = nums[i];
                nums[i] = 0;
            }
            j++;
        }
    }
}
```

DSA Notes: Moving Zeros to the End

Problem Statement

Given an array of integers, move all the zeros to the end of the array while maintaining the relative order of the non-zero elements. This operation should be done in-place without using extra space for another array.

Code Explanation

Code:

```java
public void moveZeroes(int[] nums) {
    int j = 0; // Index for the next non-zero element
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] != 0) {
            // Swap elements only if necessary
            if (i != j) {
                nums[j] = nums[i];
                nums[i] = 0;
            }
            j++;
        }
    }
}
```

Key Points of the Code

1. **Two Pointer Technique**:
    - `i`: Iterates through all elements of the array.
    - `j`: Tracks the position where the next non-zero element should be placed.
2. **Logic Flow**:
    - Traverse the array with pointer `i`.
    - Whenever a non-zero element is found at `nums[i]`:
        - If `i` is not equal to `j`, swap `nums[i]` with `nums[j]` to bring the non-zero element to the correct position.
        - Increment `j` to move to the next position for the next non-zero element.
3. **Swap Condition**:
    - Swapping only occurs when `i` and `j` are different to avoid unnecessary operations.
4. **In-place Modification**:
    - The algorithm modifies the array without using extra space.
5. **Complexity**:
    - Time Complexity: $O(n)$ (Single pass through the array)
    - Space Complexity: $O(1)$ (No extra space used)

Dry Run Example

Input:

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

Execution:

Step	i	j	nums	Action
Initial	0	0	[0, 1, 0, 3, 12]	-
Check	0	0	[0, 1, 0, 3, 12]	$nums[0] = 0$: Skip
Check	1	0	[0, 1, 0, 3, 12]	$nums[1] = 1$: Swap
After	1	1	[1, 0, 0, 3, 12]	$j++$
Check	2	1	[1, 0, 0, 3, 12]	$nums[2] = 0$: Skip
Check	3	1	[1, 0, 0, 3, 12]	$nums[3] = 3$: Swap
After	3	2	[1, 3, 0, 0, 12]	$j++$
Check	4	2	[1, 3, 0, 0, 12]	$nums[4] = 12$: Swap
After	4	3	[1, 3, 12, 0, 0]	$j++$

Final Output:

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

Advantages of This Code

1. Efficient for large arrays.
2. No extra memory allocation.
3. Simple and easy-to-follow logic.

Interview Insights

- **Commonly Asked In**:
  - Google, Microsoft, Amazon, and other product-based companies.
- **Difficulty Level**:

    - Beginner to Intermediate.
- **Follow-Up Questions**:
    1. Can you solve this with the same logic but in reverse order (moving zeros to the beginning)?
    2. What if the array is read-only? How would you handle this?

Additional Practice

- LeetCode: Move Zeroes
- Extend the solution to handle cases with other placeholders (like moving negatives or other specific values).