

3.
5

Move Zeros to End

EASY

■ What You Will Learn

- ✓ Two-pointer approach to partition zeros and non-zeros
- ✓ How to maintain relative order of non-zero elements
- ✓ In-place solution without extra space

■ Intuition First

Think of sorting a bag of marbles where you want all the clear (zero) marbles at one end. You use two hands — one scans forward looking for non-zero marbles, the other points to where the next non-zero marble should go. When the scanner finds one, swap!

■ The Approach

- j = pointer to position where next non-zero should go (starts at 0)
- i scans through the array
- When `arr[i] != 0`: swap `arr[i]` and `arr[j]`, then `j++`
- Zeros naturally fill the tail

■ Visual Trace on [0,1,0,3,12]

Step 1	i=0: arr[0]=0, skip				
	0	1	0	3	12
Step 2	i=1: arr[1]=1≠0, swap(1,j=0), j=1				
	1	0	0	3	12
Step 3	i=2: arr[2]=0, skip				
	1	0	0	3	12
Step 4	i=3: arr[3]=3≠0, swap(3,j=1), j=2				

1	3	0	0	12
---	---	---	---	----

Step 5 i=4: arr[4]=12≠0, swap(4,j=2), j=3

1	3	12	0	0
---	---	----	---	---

■ Java Code

```
public static void moveZerosToEnd(int[] arr) {
    int j = 0; // Points to next non-zero position
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] != 0) {
            // Swap arr[i] with arr[j]
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
            j++; // Move write pointer forward
        }
    }
}
```

■ ■ Complexity

Case	Time	Space	What happens?
All Cases	O(n)	O(1)	One pass — every element touched once

■ ■ Common Mistakes

- Don't move zeros to front — problem asks end
- Relative order of non-zeros MUST be preserved (swap maintains this)
- Don't create a new array — solve in-place

■ Optimization

- If i == j, you're swapping with yourself — skip with: if (i != j) swap
- This avoids unnecessary operations on already-correct elements

■ Practice Problems

[Easy] Move all negatives to front (same pattern)

[Med] Move zeros to front instead of end

[Med] Segregate even and odd numbers while preserving relative order

■ Quick Revision

- j tracks next write position for non-zeros
- Scan with i : when $\text{arr}[i] \neq 0$, swap $\text{arr}[i]$ and $\text{arr}[j]$, $j++$
- Zeros fill the tail naturally
- $O(n)$ time, $O(1)$ space — in-place
- Preserves relative order of non-zero elements