Problem: Move Zeros to Beginning

Level: Easy

You are given an array of integers. Rearrange the array so that all zeroes are at the beginning of the array.

For example,
a = [4,2,0,1,0,3,0] -> [0,0,0,4,1,2,3]

Questions to Clarify:
Q. What if there are no zeroes?
A. Then the array will be unchanged.

Q. After the re-arrangement, do non-zero elements need to be in the same order as they were before?
A. No, they need not be in the same order.

Variation of this Problem: If you want to keep non-zero elements in the same order, you can modify this algorithm a bit - instead of moving zeros to the beginning (creating a beginning partition), you move all non-zeros to the end (creating an end partition). This is similar to *Question 2* in the Lecture Videos.

Solution:
We keep one variable to track the boundary. The boundary represents the partition between zero and non-zero elements. We loop through the array's elements. For every zero we encounter, we move it into the boundary and expand the boundary by 1. At the end, all elements in the boundary will be zeroes.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

boundary = 0
for i : 0 to a.length-1
    if a[i] is 0:
         swap a[i] with boundary
         increase boundary by 1
```

Test Cases:
Edge Cases: Empty array, null array
Base Cases: one element (zero/non-zero)
Regular Cases: All zeroes, all non-zeroes, mix of zeroes and non-zeroes

Time Complexity: O(n)

Space Complexity: O(1)

```
public static void moveZeroesToBeginning(int[] a) {
    int boundary = 0;

    for (int i = 0; i < a.length; i++) {
        if (a[i] == 0) {
            Utils.swap(a, i, boundary);
            boundary += 1;
        }
    }
}
```

Problem: Move Zeros to End

Level: Easy

Given an array of integers, rearrange the elements such that all zeros are moved to the end of the array.

Questions to Clarify:
Q. Does it matter what order we place the non-zero numbers?
A. No, you can place them in any order.

Q. What if there are no zeroes?
A. Keep the array as it is.

Solution:
Keep a pointer named 'boundary' that tracks the zero-boundary at the end of the array. Everything after this boundary contains only zeros. We loop through the array from the end and place all zeroes into this boundary.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

boundary = a.length - 1
for i: a.length-1 to 0
 if a[i] is 0
     put a[i] in the zero boundary
     expand boundary by 1
```

Test Cases:
Edge Cases: empty array, null array
Base Cases: single element (0/non-0), 2 elements
Regular Cases: more than 2 elements, 0 already at end, etc.

Time Complexity: O(n)

Space Complexity: O(1)

```
public static void moveZeroesToEnd(int[] a) {
    int boundary = a.length - 1;

    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] == 0) {
            Utils.swap(a, i, boundary);
            boundary--;
        }
```

```
    }
}
```

Problem: Dutch National Flag

Level: Medium

You are given an array of integers and a pivot. Rearrange the array in the following order:
[all elements less than pivot, elements equal to pivot, elements greater than pivot]

For example,
a = [5,2,4,4,6,4,4,3] and pivot = 4 --> result = [3,2,4,4,4,4,5,6].

Questions to Clarify:
Q. Do numbers on each side need to be sorted?
A. No, they need not be sorted

Q. What if there are no numbers less than pivot?
A. Then that portion (< a[X]) will not exist.

Q. Do the numbers need to be in original order after re-arrangement?
e.g, if [4,1,4,3,5] and pivot=4, does 1 need to be before 3? E.g [1,3,4,4,5] or can it also be [3,1,4,4,5]
A. They need not be in original order. Any order is ok.

Solution:
In this problem, we have to divide the array into 3 sections.

We keep 2 boundaries - *low* and *high*. *Low* starts at 0, *High* starts at the end of the array. *Low* contains numbers less than pivot. *High* contains numbers greater than pivot. We walk through the array. If we encounter a number less than pivot, we put it in the *low* boundary. If we encounter a number greater than pivot, we put it in the *high* boundary. If the number is equal to pivot, we ignore it, effectively placing it in the middle of the two boundaries.

Pseudocode:

```
(Note: Never write pseudocode in an actual interview. Unless you're writing a
few lines quickly to plan out your solution. Your actual solution should be in
a real language and use good syntax.)

low_boundary = 0, high_boundary = a.length - 1
i = 0
while i <= high_boundary
     if a[i] < pivot:
         add a[i] to low boundary and expand boundary by 1
```

```
        i++
    else if a[i] > pivot:
        add a[i] to high boundary and expand boundary by 1
        don't increment i because the current i needs to be processed
               as it came from ahead
    else // equal to pivot
        i++
```

Test Cases:
Edge Cases: empty array, null array, pivot not in array
Base Cases: single element, two elements
Regular Cases: Single element in pivot, multiple pivots, pivot is max/min element

Time Complexity: O(n)

Space Complexity: O(1)

```
public static void dutchNationalFlag(int[] a, int pivot) {
    int low_boundary = 0, high_boundary = a.length - 1;

    int i = 0;
    while (i <= high_boundary) {
        if (a[i] < pivot) {
            Utils.swap(a, i, low_boundary);
            low_boundary++;
            i++;
        } else if (a[i] > pivot) {
            Utils.swap(a, i, high_boundary);
            high_boundary--;
        } else {
            i++;
        }
    }
}
```