Sort 0 1

Problem Description: You are given an integer array A that contains only integers 0 and 1. Write a function to sort this array. Find a solution which scans the array only once. Don't use extra array.

Sample Input:

```
10
0 1 1 0 1 0 0 1 0 0
```

Sample Output:

```
0 0 0 0 0 0 1 1 1 1
```

How to approach?

There's a lot of ways to solve this problem. We could use any of the sorting algorithms to do the job or we could use two variables to store the frequency of 0s and 1s and then reiterate the array to fill the array according to the frequencies.
Now all these approaches either scan the array more than one time or use an extra array.

We are looking for a solution that scan the array only once and has a constant space complexity.

Before we get into the approach, let's think about what we want to achieve. We want to push all 0s towards the front of the array and all 1s towards the end of array.

Let's use two pointers. One that iterates through the array and one that decides the next location for a 0 that we may encounter while iterating through the array

The algorithm will look something like this:
1. Keep two pointers 'zeroPos' for the next location of a 0, and 'current' for the current element that we're iterating through. Both of them will be initialised to 0

2. Start iterating through the array. If we encounter a 1, we'll do nothing. Just increase 'current' by 1. If we counter a 0, we need to push this zero to the front of the array. More specifically, we need to send this zero to zeroPos and bring zeroPos' element to the current location. So basically we do `swap(arr[zeroPos], arr[current])`. And then we increase 'current' by 1 **and** increase 'zeroPos' by 1 too.

The pseudo-code for this approach is shown on the next page.

```
function sort01(arr):
  zeroPos <- 0
  current <- 0
  while(current < arr.size):
    if(arr[current] = 0):
      swap(arr[current], arr[zeroPos])
      zeroPos <- zeroPos + 1
    current <- current + 1
```