# Sort 0 1 2

---

***Problem Description:*** You are given an integer array containing only 0s, 1s and 2s. Write a solution to sort this array using one scan only.

---

***Sample Input:***

```
7
0 1 2 0 2 0 1
```

***Sample Output:***

```
0 0 0 1 1 2 2
```

***How to approach?***

Trivia time! In case you didn't know, this problem is a variation of a famous problem called the 'Dutch National Flag Problem'

Let's discuss what we need to do in this problem. Basically, we need to push all 0s in the array towards the front of the array and push all the 2s towards the end of the array. Notice that if we do this, the 1s will automatically come in between the 0s and 2s.

We'll use a three pointer approach to solve this problem; the three pointers will be: `current`, `zeroPos` and `twoPos`. Let's discuss what will be the purpose of these three pointers:

1. `current` - This will hold the position of the current element that we're on during the iteration of the array. This will be initialised to 0.
2. `zeroPos` - This will hold the index where we will push any 0s that we may encounter. This will be initialised to 0.
3. `twoPos` - This will hold the index where we will push any 2s that we may encounter. This will be initialised to `n - 1`, where `n` is the size of the array

Let's discuss the approach to this problem in some detail. We'll iterate through the array using the `current` pointer. Every element is either 0, 1 or 2 so let's see what we'll be doing in each of these cases.

1. `arr[current] = 0` - In this case, we need to push the element towards the front of the array. To do that we can swap `arr[current]` and `arr[zeroPos]`, then we will increase both `current` and `zeroPos` by 1.

2. `arr[current] = 1` - In this case, we will just increase `current` by 1, since we are only concerned with push 0s to the front and 2s to the end of the array.

3. `arr[current] = 2` - In this case, we need to push the element towards the end of the array. Again, to do this, we'll just swap `arr[current]` and `arr[twoPos]`. We will decrease `twoPos` by 1. However, in this case we will **not** increase `current` by 1 (we'll discuss why later).

Now let's discuss why we didn't increase `current` by 1 in the case where `arr[current] = 2`. But before that there's one more thing that we need to discuss. What will be the condition that must be satisfied so that our loop can end? You might think that it's when current reaches the end of the array but that's not the case here. Let's see why.

Can you see what exactly the two pointers, `zeroPos` and `twoPos` are doing? As we go through the array, every element before `zeroPos` is a 0 and every element after `twoPos` is a 2. Also, every element after `zeroPos` but before current is a 1. Therefore, all these elements are 'sorted'. The element that remain to be sorted are the ones that lie between the indices `current` and `twoPos`. Therefore our loop will terminate when current reaches the value of `twoPos`.

Now, let's understand why we can't increase the value of current when `arr[current] = 2`. When we swap `arr[current]` with `arr[twoPos]`, we don't know what value was initially at index `twoPos` (before the swap happened), it could be any of the values 0, 1, or 2. So, we can't increase the value of current without checking what value was swapped with `twoPos`. We didn't have to worry about this in the case where we were swapping `arr[current]` with `arr[zeroPos]` because then we would always be swapping 0 and 1. The pseudo code at the next page will make this clear.

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

```
function sort012(arr):

  current <- 0
  zeroPos <- 0
  twoPos <- arr.size - 1

  while(current < twoPos):

    if(arr[current] == 0):
      swap(arr[current], arr[zeroPos]
      current <- current + 1
      zeroPos <- zeroPos + 1
   else if(arr[current] == 1):
      current <- current + 1
   else:
      swap(arr[current], arr[twoPos])
      twoPos <- twoPos - 1
```