Stefan Dierauf

```java
//I had to. It's one of my favorite songs.
public static void partition(int[] beyonce) {
	if (beyonce != null) {
		int n = 0;
		int i = 1;
		int end = beyonce.length - 1;
		while (i <= end) {
			if (beyonce[i] < 0) {
				swap(n, i);
				n++;
				i++;
			} else if (beyonce[i] > 0) {
				swap(i, end);
				end--;
			} else {
				i++;
			}
		}
	}
}
```

Wow, I honestly spent way more time on this than I thought I would. It's been a while since interview season.

Why I think this algorithm works:
First, it'll do nothing if beyonce is null. If beyonce has one or fewer ints, it'll also do nothing. Basic edge cases, check! Then we get into the while loop. The logic goes like this:

1. If beyonce[i] is less than zero (the middle value) then swap it with beyonce[n], pushing i to the front of the array. As we no longer care about i, we can increment both indeces.
2. if beyonce[i] is greater than zero, then it belongs at the end of the array, so put it at the last 'good' index. Decrement the end index because we know whatever is at the last place should stay there
3. if neither of these cases happen, then beyonce[i] is 0, and should stay where it is, so we go and look at the next element in beyonce to see if it should swap with the n index, which we leave either pointing to a) some unknown thing if it was the first index of the array or b) something either 0 or less that was swapped in from i. In the case of a), if it is a number >= 0, we don't have to worry because it will either 1) get swapped with a number lower than zero if i finds one or b) it's supposed to be there because the array is all positive numbers with *q* zeros at the front by the time i equals the end.

Time to completion:
This started out as a really convoluted for loop (I was decrementing i and stuff, it was gross), then I went on a run and figured most of it out (and I remembered that while loops are a thing!). Total actual time it took was maybe an hour and a half, as I got really lost in the weeds with the for loop.