Bubble Sort

Input: array = [8, 5, 2, 9, 5, 6, 3]
Output: [2, 3, 5, 5, 6, 8, 9]

Input: An array of integers
Output: a sorted version of the input array
Use the bubble sort algorithm

```
// // O(n^2) time | O(1) space
function bubbleSort(array) {
  let isSorted = false;
  while (!isSorted) {
    isSorted = true;
    for (let i = 0; i < array.length - 1; i++) {
      if (array[i] > array[i + 1]) {
        [array[i], array[i + 1]] = [array[i + 1], array[i]];
        isSorted = false;
      }
    }
  }
  return array;
}
```

Time: $O(n^2)$ (where n is length of the array) since at each element, we are iterating through the array. O(n) at best case if the array is already sorted since we would only do 1 iteration

Space: We are not using any additional space as input size grows, we are swapping in place

isSorted is used to tell us if we should continue to iterate. If the if-statement gets executed, that means we swapped values and now we can't guarentee that the array is sorted. If we never swap values, then we know the array is sorted
We go to array.length - 1 because we are swapping elements in the array. If we go the end, we cannot swap any values

```
// // O(n^2) time | O(1) space
function bubbleSort(array) {
  let isSorted = false;
  let counter = 0;
  while (!isSorted) {
    isSorted = true;
    for (let i = 0; i < array.length - 1 - counter; i++) {
      if (array[i] > array[i + 1]) {
        [array[i], array[i + 1]] = [array[i + 1], array[i]];
        isSorted = false;
      }
    }
    counter++;
  }
  return array;
}
```

Small improvement.
We know after every loop through the array, the last element is sorted in its final position.
Therefore, as a small optimization, we would not need to check the entire array at every loop.