```es6
// sample of arrays to sort
const arrayRandom = [9, 2, 5, 6, 4, 3, 7, 10, 1, 8];
const arrayOrdered = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const arrayReversed = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

// be careful: this is a very basic implementation which is nice to understand the deep principle of bubble sort (going through all comparisons) but it can be greatly improved for performances
function bubbleSortBasic(array) {
  let countOuter = 0;
  let countInner = 0;
  let countSwap = 0;

  for(let i = 0; i < array.length; i++) {
    countOuter++;
    for(let j = 1; j < array.length; j++) {
      countInner++;
      if(array[j - 1] > array[j]) {
        countSwap++;
        [array[j - 1], array[j]] = [array[j], array[j - 1]];
      }
    }
  }

  console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
  return array;
}

bubbleSortBasic(arrayRandom.slice()); // => outer: 10 inner: 90 swap: 21
bubbleSortBasic(arrayOrdered.slice()); // => outer: 10 inner: 90 swap: 0
bubbleSortBasic(arrayReversed.slice()); // => outer: 10 inner: 90 swap: 45

// correct implementation: this is the usual implementation of the bubble sort algorithm. Some loops execution are avoided if not they are not needed
function bubbleSort(array) {
  let countOuter = 0;
  let countInner = 0;
  let countSwap = 0;

  let swapped;
  do {
    countOuter++;
    swapped = false;
    for(let i = 0; i < array.length; i++) {
      countInner++;
      if(array[i] && array[i + 1] && array[i] > array[i + 1]) {
        countSwap++;
        [array[i], array[i + 1]] = [array[i + 1], array[i]];
        swapped = true;
      }
    }
  } while(swapped);

  console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
  return array;
}

bubbleSort(arrayRandom.slice()); // => outer: 9 inner: 90 swap: 21
bubbleSort(arrayOrdered.slice()); // => outer: 1 inner: 10 swap: 0
bubbleSort(arrayReversed.slice()); // => outer: 10 inner: 100 swap: 45
```

```
58
```