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

var countOuter = 0;
var countInner = 0;
var countSwap = 0;

function resetCounters() {
  countOuter = 0;
  countInner = 0;
  countSwap = 0;
}

// basic implementation (pivot is the first element of the array)
function quicksortBasic(array) {
  countOuter++;
  if(array.length < 2) {
    return array;
  }

  var pivot = array[0];
  var lesser = [];
  var greater = [];

  for(var i = 1; i < array.length; i++) {
    countInner++;
    if(array[i] < pivot) {
      lesser.push(array[i]);
    } else {
      greater.push(array[i]);
    }
  }

  return quicksortBasic(lesser).concat(pivot, quicksortBasic(greater));
}

quicksortBasic(arrayRandom.slice()); // => outer: 13 inner: 25 swap: 0
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();

quicksortBasic(arrayOrdered.slice()); // => outer: 19 inner: 45 swap: 0
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();

quicksortBasic(arrayReversed.slice()); // => outer: 19 inner: 45 swap: 0
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();

// swap function helper
function swap(array, i, j) {
  var temp = array[i];
  array[i] = array[j];
  array[j] = temp;
}

// classic implementation (with Hoare or Lomuto partition scheme, you can comment
either one method or the other to see the difference)
function quicksort(array, left, right) {
  countOuter++;
  left = left || 0;
  right = right || array.length - 1;

  // var pivot = partitionLomuto(array, left, right); // you can play with both partition
  var pivot = partitionHoare(array, left, right); // you can play with both partition

  if(left < pivot - 1) {
    quicksort(array, left, pivot - 1);
  }
  if(right > pivot) {
    quicksort(array, pivot, right);
  }
  return array;
}
// Lomuto partition scheme, it is less efficient than the Hoare partition scheme
function partitionLomuto(array, left, right) {
  var pivot = right;
  var i = left;

  for(var j = left; j < right; j++) {
    countInner++;
    if(array[j] <= array[pivot]) {
      countSwap++;
      swap(array, i, j);
      i = i + 1;
    }
  }
  countSwap++;
  swap(array, i, j);
  return i;
}
// Hoare partition scheme, it is more efficient than the Lomuto partition scheme because it does three times fewer swaps on average
function partitionHoare(array, left, right) {
  var pivot = Math.floor((left + right) / 2 );

  while(left <= right) {
    countInner++;
    while(array[left] < array[pivot]) {
      left++;
    }
    while(array[right] > array[pivot]) {
      right--;
    }
    if(left <= right) {
      countSwap++;
      swap(array, left, right);
      left++;
      right--;
    }
  }
  return left;
}

quicksort(arrayRandom.slice());
// => Hoare: outer: 9 inner: 12 swap: 12 - Lomuto: outer: 10 inner: 35 swap: 28
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();
```

```javascript

quicksort(arrayOrdered.slice());
// => Hoare: outer: 9 inner: 9 swap: 9 - Lomuto: outer: 9 inner: 45 swap: 54
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();

quicksort(arrayReversed.slice());
// => Hoare: outer: 9 inner: 13 swap: 13 - Lomuto: outer: 10 inner: 54 swap: 39
console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
resetCounters();

```