```
1  // sample of arrays to sort
2  const arrayRandom = [9, 2, 5, 6, 4, 3, 7, 10, 1, 8];
3  const arrayOrdered = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
4  const arrayReversed = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1];
5
6  let countOuter = 0;
7  let countInner = 0;
8  let countSwap = 0;
9
10 function resetCounters() {
11   countOuter = 0;
12   countInner = 0;
13   countSwap = 0;
14 }
15
16 // basic implementation (pivot is the first element of the array)
17 function quicksortBasic(array) {
18   countOuter++;
19   if(array.length < 2) {
20     return array;
21   }
22
23   const pivot = array[0];
24   const lesser = [];
25   const greater = [];
26
27   for(let i = 1; i < array.length; i++) {
28     countInner++;
29     if(array[i] < pivot) {
30       lesser.push(array[i]);
31     } else {
32       greater.push(array[i]);
33     }
34   }
35
36   return quicksortBasic(lesser).concat(pivot, quicksortBasic(greater));
37 }
38
39 quicksortBasic(arrayRandom.slice()); // => outer: 13 inner: 25 swap: 0
40 console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
41 resetCounters();
42
43 quicksortBasic(arrayOrdered.slice()); // => outer: 19 inner: 45 swap: 0
44 console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
45 resetCounters();
46
47 quicksortBasic(arrayReversed.slice()); // => outer: 19 inner: 45 swap: 0
48 console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
49 resetCounters();
50
51 // classic implementation (with Hoare or Lomuto partition scheme, you can comment
   either one method or the other to see the difference)
52 function quicksort(array, left, right) {
53   countOuter++;
54   left = left || 0;
55   right = right || array.length - 1;
56
57   // const pivot = partitionLomuto(array, left, right); // you can play with both
   partition
```

```
  const 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) {
  const pivot = right;
  let i = left;
  let last = left;

  for(var j = left; j < right; j++) {
    countInner++;
    if(array[j] <= array[pivot]) {
      countSwap++;
      [array[i], array[j]] = [array[j], array[i]];
      i = i + 1;
    }
    last = j + 1;
  }
  countSwap++;
  [array[i], array[last]] = [array[last], array[i]];
  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) {
  const 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++;
      [array[left], array[right]] = [array[right], array[left]];
      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();

quicksort(arrayOrdered.slice());
// => Hoare: outer: 9 inner: 9 swap: 9 - Lomuto: outer: 9 inner: 45 swap: 54
```

```
116 console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
117 resetCounters();
118
119 quicksort(arrayReversed.slice());
120 // => Hoare: outer: 9 inner: 13 swap: 13 - Lomuto: outer: 10 inner: 54 swap: 39
121 console.log('outer:', countOuter, 'inner:', countInner, 'swap:', countSwap);
122 resetCounters();
123
```