```
// array to sort
const array = [9, 2, 5, 6, 4, 3, 7, 10, 1, 8];

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

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

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

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

console.log(quicksortBasic(array.slice())); // => [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

// 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) {
  left = left || 0;
  right = right || array.length - 1;

  // 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(let j = left; j < right; j++) {
    if(array[j] <= array[pivot]) {
      [array[i], array[j]] = [array[j], array[i]];
      i = i + 1;
    }
    last = j + 1;
  }
  [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) {
    while(array[left] < array[pivot]) {
      left++;
    }
    while(array[right] > array[pivot]) {
      right--;
    }
    if(left <= right) {
      [array[left], array[right]] = [array[right], array[left]];
      left++;
      right--;
    }
  }
  return left;
}

console.log(quicksort(array.slice())); // => [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

```