Quicksort (Implementation)

(Reading time: under 3 minutes)

First, we create a function that receives the array we want to sort as an argument. It should also be able to gwt the *left* and *right* value. However, as we don’t pass these values at the beginning, it should have a default value as well.

```
function quicksort(array, left, right) {
   left = left || 0;
   right = right || array.length - 1;
}
```

Next, we need to create a function that chooses the pivot. In this example, I use the **Hoare partition scheme**. Again, this function should receive the array we want to sort, the *left* value, and the *right* value.

```
function partitionHoare(array, left, right) {
}
```

Here, we first choose our random pivot.

```
const pivot = Math.floor(Math.random() * (right - left + 1) + left);
```

Then, we need to check whether the left and right values are in the correct position. If the left is bigger or equal to right, they need to swap.

```
while (left <= right) {
 //Swap logic
}
```

Now, we need to check whether it’s necessary to swap. If the left value is smaller than the pivot, we increase the value of the left by one, so that we move on to the next element. If the right is bigger than the pivot, we decrease move on to the next element. If the right is bigger than the pivot, we decrease the value of right by one, so we move on to the previous element. No

swapping is needed: they're already in the correct place! If we finally find elements for which the conditions return false, we swap the elements.

```
while (array[left] < array[pivot]) {
  left++;
}
while (array[right] > array[pivot]) {
  right--;
}
if (left <= right) {
  [array[left], array[right]] = [array[right], array[left]];
}
```

We return the left value, as this will be our pivot. Right now, the entire function looks like:

```
function partitionHoare(array, left, right) {
   const pivot = Math.floor(Math.random() * (right - left + 1) + left);
   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]];
     }
   }
   return left;
}
```

Back to the quicksort function. Here, we now set the value of the pivot equal to whatever gets returned from the `partitionHoare` function.

```
const pivot = partitionHoare(array, left, right);
```

We need to invoke the quicksort function recursively, to keep on sorting every *left* and *right* side. We invoke the quicksort function with the new *left* and *right* values every time. For the left side, the *left* value is just the left element, and the right value is one element before the pivot. For the right side, the left value is the pivot, and the right value is the right element. After that, we return the sorted array.

```
if (left < pivot - 1) {
   quicksort(array, left, pivot - 1);
}
if (right > pivot) {
   quicksort(array, pivot, right);
}
```

The quicksort function now looks like:

```
function quicksort(array, left, right) {
   left = left || 0;
   right = right || array.length - 1;

   const pivot = partitionHoare(array, left, right);

   if (left < pivot - 1) {
      quicksort(array, left, pivot - 1);
   }
   if (right > pivot) {
      quicksort(array, pivot, right);
   }
   return array;
}
```

Now, let’s move on to the time complexity of this algorithm.