Counting Sort (Implementation)

(Reading time: 3 minutes)

We create a function that takes 3 arguments: the array, the minimum value, and the maximum value. In the example above, the minimum value was 0, and the maximum value was 9.

```
function countingSort(arr, min, max) {
}
```

First, we need to initialize an empty index array based on the minimum and maximum values that have been passed as parameters. The initial value of all elements in this array is 0.

```
function countingSort(arr, min, max) {
  let count = [];

  for (let i = min; i <= max; i++) {
     count[i] = 0;
  }
}
```

After initializing the index array, we need to fill this array with the occurrences of the elements in the original array. We do this by looping over the original array, and incrementing the value of the element in the index array that corresponds to the value in the original array.

```
for (let i = 0; i < arr.length; i++) {
  count[arr[i]]++;
}
```

Then, we start modifying the original array so that the elements are in the correct position. We do this by checking whether the elements in the index array aren't 0, meaning that they never occur in the original array. If they do, we place them in the correct position in the array. We know the correct position, as we declare a new variable.

```
let z = 0;

for (let i = min; i <= max; i++) {
  while (count[i]-- > 0) {
    arr[z++] = i;
  }
}
```

We loop over the index array, and check whether the element's value is bigger than 0. If that's the case, then we increment the value of variable z by one, as we go to the next element, and set it equal to the index of the value in the index array.

The entire function looks like:

```
function countingSort(arr, min, max) {
  let count = [];
  let z = 0;

  for (let i = min; i <= max; i++) {
    count[i] = 0;
  }

  for (let i = 0; i < arr.length; i++) {
     count[arr[i]]++;
  }

  for (let i = min; i <= max; i++) {
    while (count[i]-- > 0) {
      arr[z++] = i;
    }
  }
  return arr;
}
```

In the next lesson, I will talk about the time complexity of this algorithm.