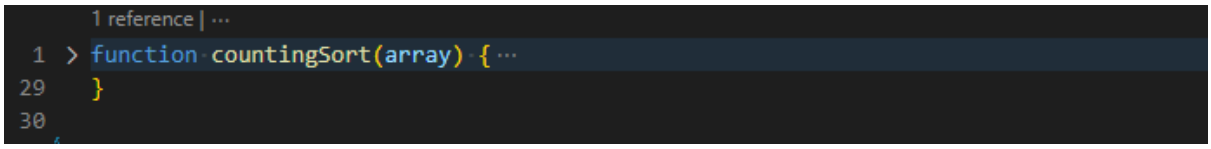
Counting Sort

Para implementar o algoritmo, primeiramente criamos a função countingSort que receberá como parâmetro um array:



Declaração da função

Então inicializamos um array vazio, chamado counts. Esse array será utilizado para amazenar o numero de ocorrências de cada digito.

Ao invés de criar um array com varios 0s, podemos criar um array nulo que funcionará da mesma forma.

Declaração do array counts

Após criar o array vazio, criamos um loop for que irá percorrer todo o array passado como parâmetro da função.

Este array irá funcionar da seguinte forma:

- O valor n irá representar cada indice do array;
- Ao acessar o valor do indice n do array counts, será avaliado se ja possui algum valor nesse indice;
- Caso ja possua algum valor, o valor irá aumentar em 1;
- Caso esteja vazio ainda, irá receber o valor 1.

```
for (n of array) {
  if (counts[n]) {
    counts[n]++;
  } else {
    counts[n] = 1;
  }
}
```

Loop que percorre o array original e gera o array counts

Utilizando o array = [1, 4, 1, 2, 7, 5, 2], o output do counts é :

```
Array counts :
[ <1 empty item>, 2, 2, <1 empty item>, 1, 1, <1 empty item>, 1 ]
```

Output counts

Onde <empty item> significa que não possui aquele valor no array, e o numero é o numero de ocorrências daquele valor

Declaramos um array result que irá armazenar o array original de forma ordenada

```
let result = [];
```

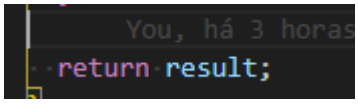
Declaração do array result

Criamos um loop for que irá percorrer o array counts, utilizando a variavel i , e vai adicionar os valores j vezes ao vetor result.

```
for (let i = 0; i < counts.length; i++) {
  for (let j = 0; j < counts[i]; j++) {
    result.push(i);
  }       You, há 6 dias • Adicionado algoritmo …
}
```

Loop que preenche o vetor result

Por fim, retornamos o array result

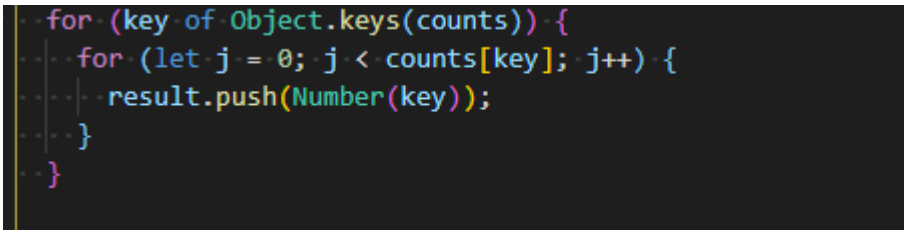


Retorno do vetor result

Lidando com o problema de intervalos grandes

O método countSort funciona melhor em casos em que o vetor possui valores em um intervalo pequeno, como o menor valor sendo 0 e o maior valor sendo 1000. Em casos em que o intervalo é muito grande, o algoritmo ou possui um longo tempo de execução, ou não funciona, como em casos em que o menor valor é 0 e o maior valor é 99999999. O algoritmo será executado, porém o resultado surgirá após muito tempo. Este problema surge devido a forma que o algoritmo funciona: todos os índices devem ser acessados para ter seu valor verificado, tornando necessário que todos os 99999999 índices sejam acessados.

Utilizando javascript, podemos superar esse problema utilizando o método chamado Object.keys( ), que irá acessar as chaves do array, não mais os seus índices. Desta forma, ao invés de acessar todos os índices, só serão verificados os valores únicos das chaves, ja que os índices vazios não terão valor.



```
for (key of Object.keys(counts)) {
    for (let j = 0; j < counts[key]; j++) {
        result.push(Number(key));
    }
}
```

Loop for que acessa as chaves do array