E-Question

Isak Nyberg
December 1, 2022

Problem Description

Given a list of positive integers $L = a_1, a_2, \cdots, a_n$ with $maxL = 10n$ where n is the length of the list. Sort the list L in linear time.

Counting Sort

Given that the maximum value of the list is only a multiple of n a simple linear sorting algorithm such as counting sort can be used.

Algorithm

Create a list with zeroes for each possible value of the elements in the list and fill each index with the number of instances from elements of the list. In the end reconstruct the list in a sorted fashion using the counting list.

Algorithm 1 Counting Sort

```
Input: $L$
1: $n \leftarrow L.length$
2: $count \leftarrow \{0, \overset{10n}{\dots}, 0\}$ // fill an array of length $10n$ with zeroes
3: for $e \in L$ do
4:     $count[e] \leftarrow count[e] + 1$
5: end for
6:
7: for $i \in 2$ to $10 \times n$ do // for $i$ from 2 (inclusive) to 10n (inclusive)
8:     $count[i] \leftarrow count[i] + count[i-1]$
9: end for
10:
11: $res \leftarrow [n]$ // Set res to array of length $n$
12: for $e \in L.reverse()$ do
13:     $res[count[e]] \leftarrow e$
14:     $count[e] \leftarrow count[e] - 1$
15: end for
16: return $res$
```

This implements counting sort, the first *for* loop will increment the index in the array for each instance of a number in input list. For example if there are seventeen different fours in the list L the value of count at index four will be seventeen. $count[4] = 17$. (This assumes that the array count is indexed from 1.) When this is done the array is iterated over and for each index the number of elements corresponding to the value at that index is added to the list res, so for example when the index 4 is reached the number 4 will be appended to res 17 times.

Proof of Correctness

The two first *for* loops are rather trivial, the first one will count the number of instances of a number for every index less than or equal to $10n$. The second *for* will just make the list accumulated, so that instead of each index being the number of instances of that number, instead it will be the number of instances of that number or less. The third *for* loop on line 12 is the more interesting one. An invariant is that for each element e that has been seen in L it will have been placed in the correct spot in res. This is achieved by the initial lookup from $count[e]$ which find out what the spot the element should be in since it knows how many elements that are less than or equal to e, thereafter it is places in the appropriate position in line 13. Line 14 also decrements $count[e]$ since there is now one less element less that or equal to e

Space Complexity

There are two relevant uses of memory, the first is the array $count$ and the second is res. Count is defined to be of size $10n$ and res of size n. From this it can be determined that the space complexity is linear $\mathcal{O}(n)$ with respect to the size of the input.

Time Complexity

Thankfully we know the maximum element of L is less than or equal to ten times the length of L. This means that the count array size only needs to be that size. Initializing the array count will run in $10n$ times. The *for* loop on line 3 will trivially run $10n$ times, same for the second *for* on line 7. The third *for* loop on line 12 will also trivially run n times. Thus the time complexity of the algorithm as a whole is

$$\mathcal{O}(n + 10n + n) = \mathcal{O}(n)$$

which is linear.