# Individual Assignment 2

David Östling
dostl@kth.se
DD2440

December 11, 2022

## Problem description

Consider a list $A$ of positive integers $\{a_1, \ldots, a_n\}$ where it is known that $\max\{a_1, \ldots, a_n\} \leq$ 10n. Describe and analyze an algorithm that sorts such lists in linear time.

## Solution

When presented with a list $A$ of the above kind there are multiple ways of sorting it, some ways more efficient than others. Here however, given that all inputs are postive integers and that $\max\{a_1, \ldots, a_n\} \leq$ 10n, one of the most simple solutions which allows $A$ to be sorted in linear time would be the *Counting Sort algorithm.* Although, to further simplify the algorithm we will make some slight modifications to it. Note that *counting sort* can only run in linear time if the maximum value is a *constant* or proportional to $n$ *(which is the case above; max*$\{a_1, \ldots, a_n\} \leq$ *10n*), in any other instance we would need to resort to other methods instead.

The simplified variant of the *Counting Sort* algorithm can be described as follows:

- The first step is identifying the max element of the input list $A$. As of above it is known that $\max\{a_1, \ldots, a_n\} \leq$ 10n.
- Now the algorithm creates a new array *Count* going from 0 to *10n + 1.* This array will be used to track all occurences of each unique element in $A$. Here, all values of *Count* are initialized to 0.
- The algorithm then fills all indexes of *Count* with the occurences of each unique element of $A$. This is achieved by incrementing each corresponding element for every index in *Count* by *1* everytime the index is encountered in $A$. Hence, the number assigned to each index in *Count* becomes the amount of times each particular index appears as an element in $A$. As a result, all indexes of *Count* that do not correlate with any unique element of $A$ will have their values remain at 0 which they were intitialized to.

- In a normal circumstance of *Counting Sort*, at this stage the algorithm would compute a prefix sum in order to permute the elements into sorted order. However, since all items in $A$ are positive integers *(where the items themselves can be used as indexes for the Count array)* we can resort to a simplified version of the algorithm...
- Once all elements have been counted and the *Count* array has been finalized it is time to begin the sorting process. Here, it is important to note that each unique element of $A$ is used as an index for *Count*. As a result, we can simply go through all elements in *Count*, extract and append each index of *Count* to a new array we call *SortedA* by the number of the index's corresponding value. If an element is equal to 0 the algoritm ignores it *(as its index is not present in A)* and continues iterating. This is done in increasing order going from *0 to 10n*.
- Once all elements have been extracted and appended into *SortedA* the algorithm simply returns the list and $A$ has been sorted.

## Pseudocode

**Algorithm 1** Counting Sort variant algorithm

1: $A \leftarrow$ a list of positive integers $\{\mathrm{a}_1, \ldots, a_n\}$ where max s$\{a_1, \ldots, a_n\} \leq$ 10n<br>
2: *SortedA* $\leftarrow$ output array of length n *(same as A)*<br>
3: *Count* $\leftarrow$ a "count" list of maximum length *10n + 1*<br>
4: **for** (i $\leftarrow$ 0 **TO** 10n) **do**<br>
5:     Count[i] $\leftarrow$ 0<br>
6: **for** (j $\leftarrow$ 0 **TO** n) **do**<br>
7:     Count[A[j]]++<br>
8: **for** (i $\leftarrow$ 0 **TO** 10n) **do**<br>
9:     **for** (j $\leftarrow$ 0 **TO** Count[i]) **do**<br>
10:         SortedA.append(i)<br>
11: **return** SortedA

## Time Complexity

The above algorithm heavily relies on for loops where it first goes through the entire *Count* list. Here it initializes all *10n + 1* values to 0; this takes *O(10n)* time where 10n is the maximum value found within the input list $A$. Additionally, the next loop goes through the entirity of $A$ when counting each element of the list. Since $A$ has a length of $n$, this takes *O(n)* time.

For the last two nestled for loops the case is a little different. In the outer loop the algorithm goes through each element in *Count* taking *O(10n)* time. Finally, within the inner loop the algorithm needs to go through all counted elements at each index of the Count array, this takes at most $n$ times as the total number of counts in the *Count* array will always be equal to the length $n$ of the input array $A$ *(or in other words, row 9 in the pseudocode will be ran at most n times).*

This yields an overall time complexity of:

$$\text{O(10n+n+10n+n)} = \text{O(22n)} = \text{O(n)}$$

*(where A* $\{a_1, \ldots, a_n\}$ *is an input list of positive integers and max*$\{a_1, \ldots, a_n\} \leq$ *10n)*

Hence, the algorithm runs in linear time.

## Correctness

**Lemma 1: The algorithm sorts the input list**

***Proof:*** The algorithm creates a new list *Count* which goes between *0 and 10n.* Here, it is important to note that all values in the list are initialized to 0. After this, each index of the *Count* list is filled with the number of occurences of each corresponding unique element of $A$. Then the algorithm simply appends each index of *Count* to a new list *SortedA* by the number of the index's corresponding value *(the amount of times an index of Count appears as an element in A)*. However, this is only the case if the value found at an index $\neq 0$. Although, if an index of *Count* has a corresponding value of 0, we can be sure that the index the 0 was found at is not an element included in the original array $A$, hence such indexes will not be included in the sorted list *(the algorithm will not need to loop through such indexes).* It appends the indexes in increasing order, going from 0 to 10n thus covering all possible values that could be present in $A$. Hence, when all elements have been returned the input list $A$ has been sorted. $\square$