Counting Sort

Let $\Delta = \{0, \ldots, m - 1\}$

input: $a_1, \ldots, a_n \in \Delta$

initialize $T[j] \leftarrow \text{"empty list" } (j = 0 \ldots m - 1)$

for $i \leftarrow 1$ to $n$ do
    $T[a_i] \leftarrow T[a_i] \parallel a_i$

output $T[0] \parallel T[1] \parallel \cdots \parallel T[m - 1]$

Running time: $O(m + n)$

A stable sort: if two items are equal, the sort does not affect their relative order
Lexicographic Sort (1)

input: $A_1, \ldots, A_n \in \Delta^t$

for $j \leftarrow t$ down to 1 do
    counting sort the $A_i$’s using $j$th entry as the “sort key”

Correctness: follows from stability of Counting Sort

Running time: $O(nt + mt)$

Improvements:

• reduce running time to $O(nt + m)$
• handle variable length inputs
Lexicographic Sort (2)

Input: $A_1, \ldots, A_n \in \Delta^*$, where $t_i := |A_i| > 0$, $t_{\text{max}} := \max \{t_i\}$, $N := \sum_i t_i$

Step 1: for $j = 1 \ldots t_{\text{max}}$, create a list $L[j]$ of all $A_i$’s of length $j$

Step 2: create a list of $N$ pairs $(j, a_{ij})$, where $a_{ij}$ is the $j$th component of $A_i$ [Time $= O(N)$]

Step 3: sort pairs lexicographically — Counting Sort twice, first in the second component ($m$ slots), and then in the first component ($t_{\text{max}}$ slots) [Time $= O(N + m)$]
Step 4: run lex sort as before, except that we use the data from step 3 to ignore empty slots

\[ L \leftarrow \text{empty list} \]
\[ \text{for } j \leftarrow t_{\text{max}} \text{ down to } 1 \text{ do} \]
\[ L \leftarrow L[j] \parallel L \]
\[ \text{counting sort } L \text{ using } j\text{th component as the “sort key”, ignoring empty slots} \]

**Running Time Analysis**

The running time of loop iteration \( j \) is proportional to the number of pairs \((j, a_{ij})\)

The total cost is proportional to the total number of pairs, which is \( N \)
Putting it all together: total running time is $O(N + m)$

For constant $m$, or $m = O(N)$, this is linear in the input size

Does not contradict the sorting lower bound