Sorting strings over an alphabet \( \Sigma \)

\( n \) numbers in range \([0\ldots m]\)
can be sorted in \( O(n+m) \) steps
and \( O(n+m) \) space.

Scan the numbers from the list
Consider the list in order of the buckets

Observation: If \( m = O(n) \) then
total time in \( O(n) \).

But if \( m = \omega(n) \)

\[
\frac{m}{n} \to \infty \quad n \to \infty
\]
then run-time is \( \Omega(m) \).

To sort numbers in range \([1\ldots n^2]\)
we don't apply bucket sort
\[ m = n \times n \]

Each number is split into two equal parts, each of size at most \( \log n \) bits.

1. Then we sort the lower half.

2. Do a stable sort in the upper half.

<table>
<thead>
<tr>
<th>5 1</th>
<th>2 6</th>
<th>1st pass</th>
<th>5 1 2nd pass</th>
<th>1 8</th>
<th>Move passes if required</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 6</td>
<td>1 8</td>
<td>9 5</td>
<td>4 3</td>
<td>2 6</td>
<td></td>
</tr>
<tr>
<td>4 3</td>
<td>9 5</td>
<td>4 3</td>
<td>5 1</td>
<td>4 5</td>
<td></td>
</tr>
<tr>
<td>1 8</td>
<td>4 5</td>
<td>2 6</td>
<td>5 1</td>
<td>1 8</td>
<td></td>
</tr>
<tr>
<td>4 5</td>
<td>2 6</td>
<td>1 8</td>
<td>9 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each pass takes \( O(m+n) \) time, so

\[ K \text{ passes } \Rightarrow O(K(m+n)) \]

Since \( m = O(n^2) \) \( \Rightarrow O(Kn^2) \)
We can sort in the range $[1, n^k]$ in $O(k(n))$ steps. As long as $K$ is constant, we can sort in $O(n)$ time.

This is not comparisons and therefore we can beat the $\Omega(n \log n)$ bound.

To sort strings that have the same lengths, the natural choice is Radix sort.

$\Sigma$: alphabet, $l$: length of each number,

$n$: string

If we use radix sort,

$O\left[\left(|\Sigma| + n\right)l\right]: O(nl)$ if $|\Sigma| \in O(1)$

This is linear-time with input size.

Input size $N = nl \quad O(N)$.
In lexicographic sorting, when strings are of different lengths, say $l_1, l_2, l_3 \ldots l_n$, then we can artificially transform the strings into length $l_{\text{max}}$ and then apply radix sort.

dna $\Rightarrow$ dna $\text{c} \quad \text{cross out}

c

Running time: $O(l_{\text{max}} \cdot n)$

Suppose $n-1$ strings of length 1 and 1 string of length $n$

$N = n-1 + n = 2n-1$

Time to sort = $n^2!$. Too wasteful.

This is the true of "blanks"
When there are blanks, we only need to copy them from the previous pass (stable sort).

We are willing to pay the price for non-blank symbols.

Total size of input: \( \sum_{i=1}^{n} l_i = N \)

For a specific pass, say \( j \), suppose there are \( m_j \) non-blank symbols

\[ \sum_{j} m_j = \sum_{i} l_i \]

Goal: sort in-time proportional to \( O(m_j) \) in pass \( j \)

If we knew that \( S_i \) is the set of things involved in the \( j \)th pass, then we are done! \( j \)th pass takes time \( O(|S_i|) = O(m_j) \)
We will assume preprocessing to construct $S_i$ for all $i$.

For each string $S_i$, let us construct tuples $(j, S_{ij})$ where $S_{ij}$ denotes the $j$th symbol of string $S_i$. Example candidate:

$$(1, c) (3, v) (2, a) (1, c)$$

$\Rightarrow$ Total $O \leq |S_i|$ tuples = $N$

Do a radix sort on the tuples $\Rightarrow$ we have all symbols corresponding to a index clubbed together

Time:
- First pass: $O(|S| + N) = O(N)$
- Second pass: $(\ell_{max} + N) = O(N)$
- Overall $O(N)$
\[ \begin{align*}
\text{c a v e} & = \begin{pmatrix} 1, c \\ 2, a \\ 3, v \\ 4, e \end{pmatrix} \\
\text{b a t} & = \begin{pmatrix} 1, b \\ 2, a \\ 3, t \end{pmatrix} \\
\text{a t} & = \begin{pmatrix} 1, a \\ 2, t \end{pmatrix}
\end{align*} \]