Sorting without comparisons

By counting # of smaller category elements, we can get the rank.

If an input is from range 1, 2, ..., m then, in an input of size n
we can count #1's #2's ... #m
from which the output can be computed uniquely.

Input: 4 1 3 1 10 100 ...

#1's #2's #3's #4
3 5 1 0

Output: 111 1112 2222 ...

Count sort scans the input to find the #i's
in the input and then writes the output.

Running time: \(O(n + m)\)

Assuming that
fashioning to the
correct counter takes \(O(1)\) time
When \( m = n \) then 'ideal',
say \( m = n \) \( \Theta(n) \).
What happens is \( \Omega(n \log n) \) lower bound.

Comparison Tree model

Any comparison input will traverse a fixed path in the tree

\[ x_i : x_j \]

\[ x_i : x_j \]

\[ x_i : x_k \]

Sorted output

\[ \geq n! \text{ leaf nodes corresponding to each perm} \]

Obs A binary tree with \( L \) leaf nodes must have a path of length at least

\[ \log_2 L \]

\[ L = n! \]

Since any comparison based algorithm can be represented by a \( n! \) leaf node binary tree (the worst case bound is \( \Omega(n \log n) \)),
Sorting strings of fixed size, say, each string has length \( l \) \((l \text{ bits})\)

\[ m = 2^l \]

Time: \( O(n+2^l) \)

\( l \leq \log n \)

Read, sort, partition

Use \( l \) bits in chunks \( l \)

Count sort on these chunks starting from the lowest significant bit.

\[ 56, 21, 58, 31, 35, 29, 19, 24 \]

First round

\[ 21, 31, 24, 35, 56, 58, 29, 19 \]

Second round

\[ 19, 21, 24, 29, 31, 35, 56, 58 \]

Starting with the second round, the sorting must be “stable”
Time for radix sort is:
\[ \frac{l}{\log n} \times \text{Time for each round} = O(n \cdot \frac{l}{\log n}) \]

If \( l = c \cdot \log n \), \( c \) is constant:

Time to sort \( n \) en = \( O(n) \)

What about strings with variable lengths?

\( l_1, l_2, l_3, \ldots, l_n \)

Length of string \( S_i = l_i \)

\( S_1 : \text{\textcolor{red}{\texttt{00101}}} \)

\( l_1 = 3 \)

\( S_{10} = \text{\texttt{110110000}} \)

\( L_{10} = 100 \)

Look at \( \max l_i = L \)

Convert all strings to length \( L \) by leading 0’s:

Time: \( O\left(\frac{L}{\log n} \times n\right) = O\left(\frac{L \log n}{\log n}\right) \)
Input size of the problem was

\[ \leq l_i : N \]

By appending leading bits, it becomes \[ L \cdot n \]

Compare \[ N \cdot \ln \]

Blowup can be by a quadratic factor

Goal: \( O(N) \) algorithm for sorting variable size strings

Tries: Can they be used to sort?

(Digital Trees)