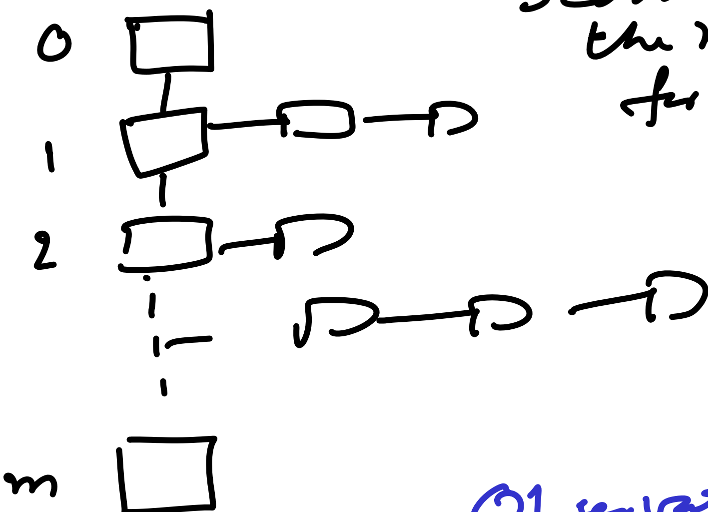
CSL 630 Lecture 6 Aug 11

Sorting strings over an alphabet Σ

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

Concatenate the list in order of the buckets

Observation: If $m = O(n)$ then total time is $O(n)$

But if $m = \omega(n)$

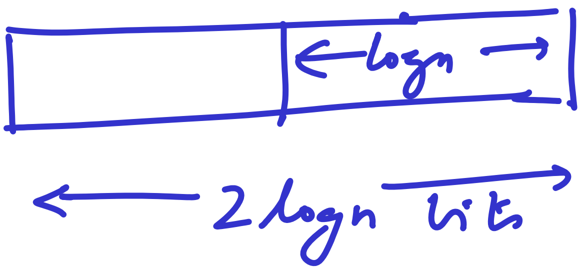
small omeaga

$\frac{m}{n} \to \infty$ $n \to \infty$, then run-time is $\Omega(m)$

To sort numbers in range $[1 .. n^2]$ we don't apply bucket sort

$m = n \times n$

Each number is split into two equal parts, each of size at most



① Then we sort the lower half

② Do a stable sort on the upper half

5 1	1^{st} pass →	5 1	2^{nd} pass → Stable	1 8	More passes if required
2 6		4 3		2 6	
9 5		9 5		4 3	
4 3		4 5		4 5	
1 8		2 6		5 1	
4 5		1 8		9 5	

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

K passes $\Rightarrow O(K(m+n))$

Since $m = O(n) \Rightarrow O(Kn)$

We can sort nos in the range $[1, n^k]$ in $O(k(n))$ steps

$k \log n \cdot b$

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

Σ : alphabet ℓ : length of each number

n : strings

If we use radix sort,

$$O\left[\left(|\Sigma| + n\right)\ell\right] = O(n\ell) \text{ if } |\Sigma| \text{ is } O(1)$$

This is linear-time with input size

Input size $N = n\ell$ $O(N)$

In lexicographic sorting, when strings are of different lengths, say $l_1, l_2, l_3 \dots l_n$ then we can artificially transform the strings into length l_{max} and then apply radix sort

$$\begin{array}{ccc} d & n & a \\ c & & \end{array} \Rightarrow \cancel{\begin{array}{ccc} d & n & a \\ - & - & c \end{array}} \quad \begin{array}{ccc} c & - & - \\ d & n & a \end{array} \checkmark$$

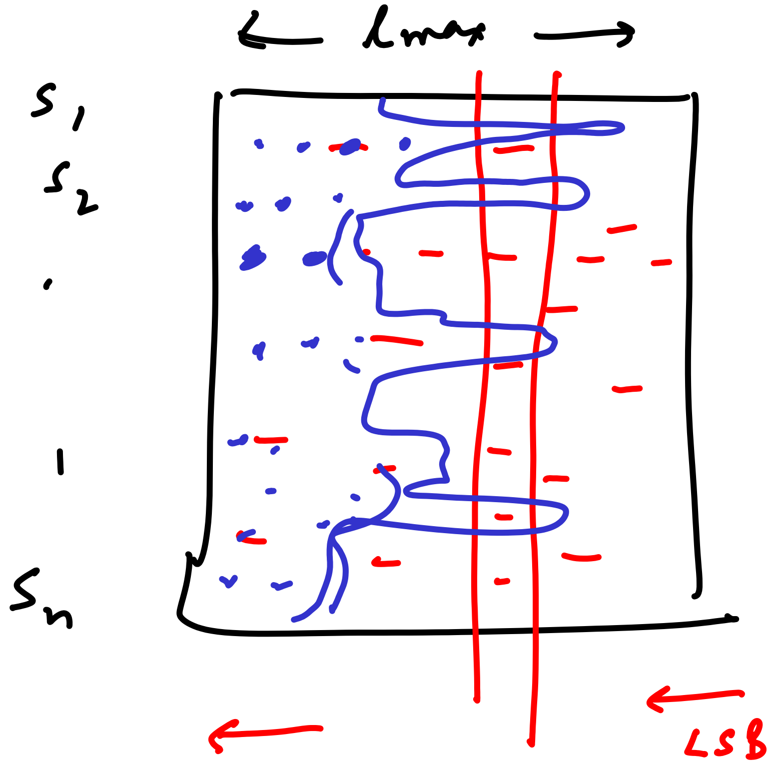
Running time : $O(l_{max} \cdot n)$

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

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

Time to sort is n^2! Too wasteful

This is the price 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

$$\sum_j m_j = \sum l_i$$

Goal: Sort in time proportional to $O(m_j)$ in pass j

If we knew that S^j is the set of strings involved in the j^{th} pass, then we are done! j^{th} pass takes time $O(|S^j|) = O(m_j)$

We will some preprocessing to construct S^j for all j

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 c a v e

$(4, e)$ $(3, v)$ $(2, a)$ $(1, c)$

$\Rightarrow$ Total of $\sum l_i$ tuples $= N$

Do a radix sort on the tuples

$\Rightarrow$ we have all the symbols corresponding to a index clubbed together

<u>Time</u> : <u>First pass</u> : $O(|\Sigma| + N) = O(N)$

<u>Second pass</u> : $(l_{max} + N) = O(N)$

Overall $O(N)$

c a v e $= {}^{1}(1,c)\ {}^{1}(2,a)\ {}^{1}(3,v)\ (4,e)$

b a t $\quad {}^{2}(1,b)\ {}^{2}(2,a)\ {}^{2}(3,t)$

a t $\quad {}^{3}(1,a)\ {}^{3}(2,t)$

$^{1}(2,a)$	$(1,a)$
$^{2}(2,a)$	$(1,b)$
$^{3}(1,a)$	$(1,c)$
$^{2}(1,b)$	$(2,a)$
$^{1}(1,c)$	$(2,a)$
$^{1}(4,e)$	$(2,t)$
$\vdots$	$(3,t)$
$^{2}(3,t)$	$(3,v)$
$^{3}(2,t)$	$(4,e)$
$^{1}(3,v)$	