Lec 05 线性时间排序

how fast can we sort ?

it depends on what we call the computational model

' what you are allowed to do with the elements'

can we do better than $\Theta(n\lg n)$?

Comparison Sorting Model :

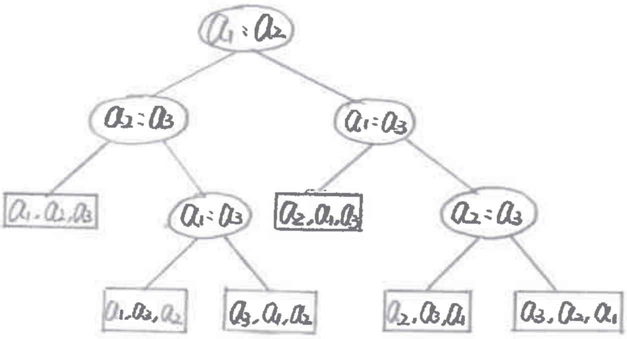
only use comparisons to determine the relative order of elements

Decision - Tree Model : (决策树)

more general than comparison model

Example:

to sort $<a_1, a_2, a_3>$



what this tree means is that each node you're making a comparison $(x:y)$. if $x<y$, go left, and go right otherwise. when you get down to a leaf, this is the answer.

Def. in general, $<a_1, a_2, \cdots, a_n>$,

- △ each internal node (non-leaf node) has a label of the form "$i:j$" where $1 \le i, j \le n$, means we compare a_i and a_j . and we have two subtrees from every such node ,
  - left subtree which tells you what the algorithm does when $a_i \le a_j$
  - right subtree which gives subsequent comparisons if $a_i > a_j$
- △ each leaf node gives a permutation , $<\pi(1), \pi(2), \cdots, \pi(n)>$, such that $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$

用决策树的表达方式构建比较排序算法：(转换的过程)

"基于比较的排序可以转换成决策树，但是有一些排序算法无法以决策树的形式表现出来"

- one tree for each n
- view algorithm as splitting into two forks (subtrees) whenever it makes a comparison
- tree lists comparisons along all possible instruction traces
- running time (the number of comparisons) = the length of path
- the worst-case running time = the height of the tree

lower bound on decision-tree sorting:

“下界” any decision-tree sorting n elements has height $\Omega(n\lg n)$ “≥”

proof: the number of leaves must be at least $n!$
(as there are n factorial permutations of an input)
the height of the tree $:= h$, then it has at most 2^h leaves

$\Longrightarrow \#\text{leaves} \le 2^h$

$\Longrightarrow n! \le 2^h$

$\Longrightarrow h \ge \log_2(n!)$

“斯特林公式” $\ge \log_2\left(\frac{n}{e}\right)^n$

$= n \log_2 \frac{n}{e}$

$= n\log_2 n - n\log_2 e$

$= \Omega(n\lg n)$

corollary: merge sort and heapsort are asymptotically optimal $(n\lg n)$, 优
but this is only in the comparison model,
Randomized QuickSort is too in expectation.

“基于比较模型的话，nlgn 就是极限了”

Sorting in Linear Time “不可能比线性时间更快完成排序，因为得遍历数据”

2 algorithms for instance

① counting sort

input: $A[1\ldots n]$, each $A[i]$ is an integer from the range of 1 to k

“当k较小时，性能比较好”

output: $B[1\ldots n]$ = sorting of A

auxiliary storage: $C[1\ldots k]$

Counting Sort:

```
for i ← 1 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]]+1          // C[i]表示数值i出现的次数
for i ← 2 to k
    do C[i] ← C[i] + C[i-1]         // C[i]表示 小于等于i的元素的数目 (对前缀做加法)
                                                                  “prefix sum”
for j ← n downto 1                  // distribution
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]]-1
```

Example:

$A = [4, 1, 3, 4, 3]$

$$C = \begin{array}{|c|c|c|c|}\hline 0&0&0&0\\ \hline \multicolumn{1}{c}{1}&\multicolumn{1}{c}{2}&\multicolumn{1}{c}{3}&\multicolumn{1}{c}{4}\end{array} \xrightarrow{\text{counting}} C = \begin{array}{|c|c|c|c|}\hline 1&0&2&2\\ \hline \multicolumn{1}{c}{1}&\multicolumn{1}{c}{2}&\multicolumn{1}{c}{3}&\multicolumn{1}{c}{4}\end{array}$$

$$\xrightarrow{\text{prefix sum}} C' = \begin{array}{|c|c|c|c|}\hline 1&1&3&5\\ \hline \multicolumn{1}{c}{1}&\multicolumn{1}{c}{2}&\multicolumn{1}{c}{3}&\multicolumn{1}{c}{4}\end{array}$$

$j=5$, $A[j]=3$, $B = \begin{array}{|c|c|c|c|c|}\hline \ & \ & 3 & \ & \ \\ \hline \end{array}$

$C'[A[j]]=3$ ← "值为3的元素应该放在位置3或者更靠前"

$C'=[1,1,\textcircled{2},5]$

$j=4$, $A[j]=4$, $B = \begin{array}{|c|c|c|c|c|}\hline \ & \ & 3 & \ & 4 \\ \hline \end{array}$

$C'[A[j]]=5$

$C'=[1,1,2,\textcircled{4}]$

. . . .

$B = \begin{array}{|c|c|c|c|c|}\hline 1&3&3&4&4\\ \hline \end{array}$

$T(n) = O(k+n)$,

it is a great algorithm if k is relatively small, like at most n.

so you could write a combination algorithm that if $k > n\lg n$ and if $k \le n\lg n$

a stable sorting algorithm preserves the relative order of equal elements, counting sort is stable

② radix sort 基数排序 by Herman Hollerith in 1890

radix sort is going to work for a much larger range of numbers in linear time

first: sort by the most significant digit first (need many boxes)

right: (by Hollerith) sort by the least significant digit first, using stable sorting

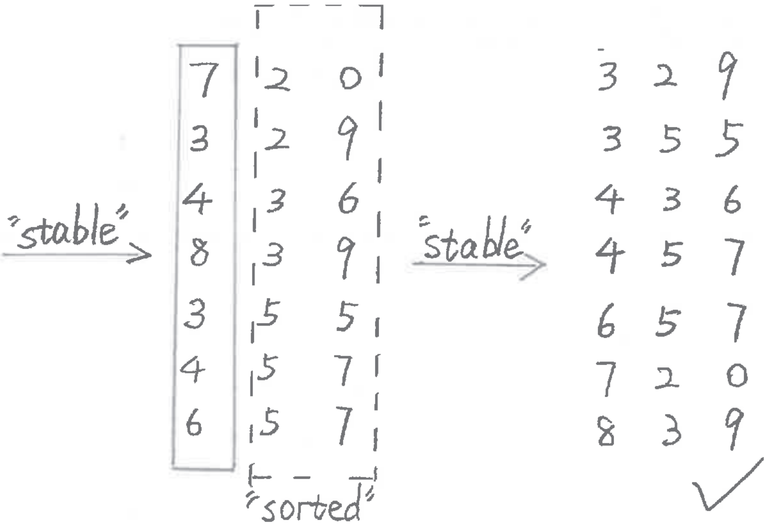
Hollerith 在1911年创建制表机公司 (tabulating machine company),
然后在1924年合并了其他几个公司, 组成了 IBM

the whole idea is that we are doing a digit-by-digit sort,
from least significant digit to most significant digit

the nice thing about this algorithm is that there are no bins, it's one big bin at all times

Example:

$$\begin{array}{ccc} 3&2&9\\ 4&5&7\\ 6&5&7\\ 8&3&9\\ 4&3&6\\ 7&2&0\\ 3&5&5 \end{array} \xrightarrow{\text{"LSB"}} \begin{array}{cc|c|} \cline{3-3} 3&2&9\\ 4&5&7\\ 6&5&7\\ 8&3&9\\ 4&3&6\\ 7&2&0\\ 3&5&5\\ \cline{3-3} \end{array} \xrightarrow{\text{"stable"}} \begin{array}{c|c|c} \cline{2-2} 7&2&0\\ 3&5&5\\ 4&3&6\\ 4&5&7\\ 6&5&7\\ 3&2&9\\ 8&3&9\\ \cline{2-2} \end{array}$$



7	2 0	3 2 9
3	2 9	3 5 5
4	3 6	4 3 6
8	3 9	4 5 7
3	5 5	6 5 7
4	5 7	7 2 0
6	5 7	8 3 9

"when I have equal elements here, I have already sorted the suffix"
"好的部分是我们不用分成一个个箱子了，而是始终把它们放在一个大箱子里面。"

Correctness:

to induct on the digit position t that we are currently sorting,
assume that by induction that it is already sorted on lower $t-1$ digits,
and then the next thing we do is to sort on the t-th digit.
if two elements are the same (has the same t-th digit),
stability ⇒ keep the order ⇒ still sorted
else,
put them in the right order

next we are going to use counting sort for each round
(we could use any sorting algorithm we want for individual digits)

Analysis:

- use counting sort ($O(k+n)$)
- say n integers, each b bits long ($0 \sim 2^b-1$) "可以把几个比特放在一起当做一位数处理，相当于八进制、十六进制等"
- split each integer into $\underbrace{b/r}_{\text{"需要运算的轮数"}}$ "digits", each "digit" is r bits long "基于2^r进制来表示这个数" "counting sort 的 $k=2^r$"

$$T(n) = O\left(\frac{b}{r}\cdot(n+2^r)\right)$$

$$\min_r T(n)$$

$$r=\log_2 n,\quad T(n) = O\left(\frac{bn}{\log_2 n}\right)$$

if numbers (integers) are in the range $0 \sim 2^b-1$
$0 \sim n^d-1$
then $T(n) = O(d\cdot n)$