LECTURE 7:
SORTING V: LOWER BOUNDS ON SORTING, COUNTING SORT, RADIX SORT

6.006
L07.1
02/25/2016

Reading CLRS 8.1–8.3

Quiz 1 Information Sheet – on Stellar site

Last time: "Balanced" trees (AVL) give $\Theta(\log n)$ height and operations, and $\Theta(n \log n)$ sorting.

To satisfy AVL invariant

Can we improve on $\Theta(n \log n)$?

TODAY:
- $\Theta(n \log n)$ is best possible for comparison sort
- Non-comparison can be $\Theta(n)$

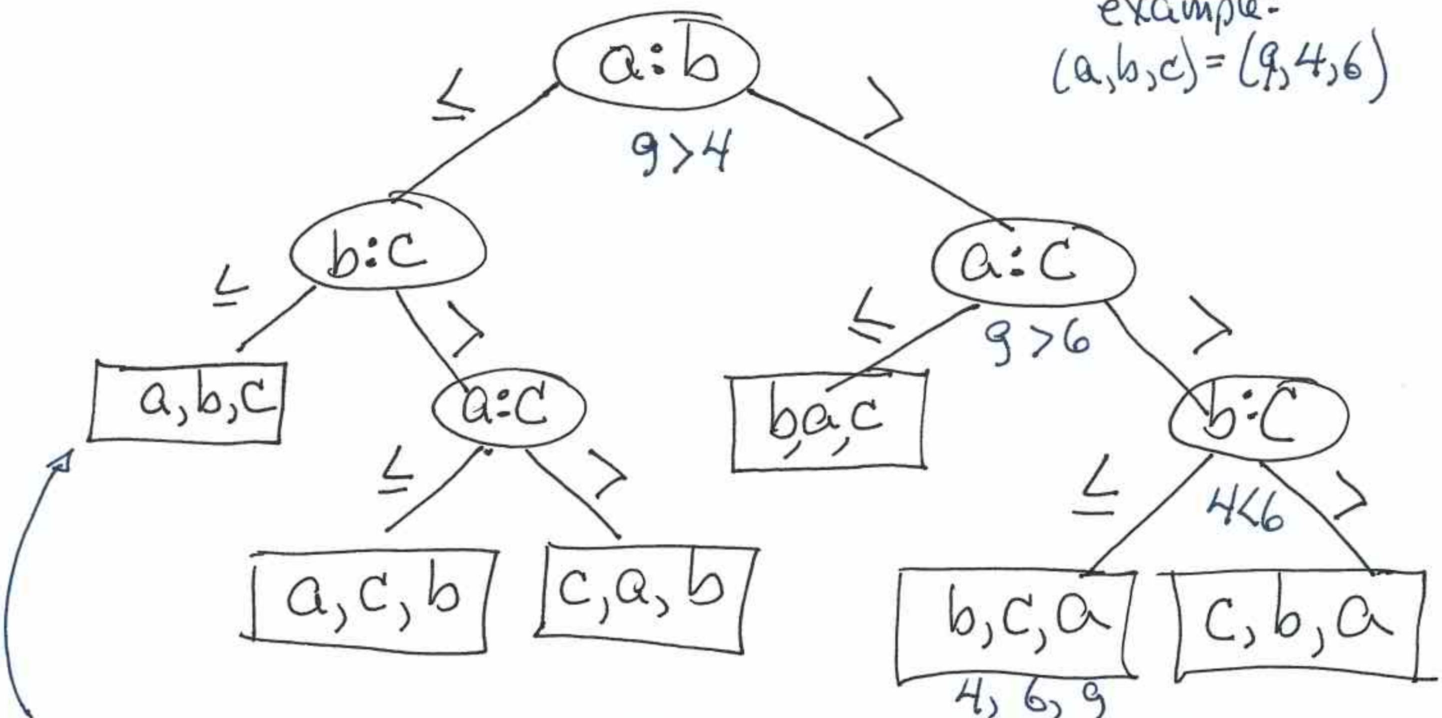
Our sorting algorithms use comparisons between items (explicitly or implicitly)
$\rightarrow$ merge sort, heap sort, AVL sort
$\Longrightarrow \Theta(n \log n)$

Abstract comparison sort to Decision Tree

Each pairwise comparison takes place at a node; branch left or right based on outcome of comparison.

sort three items a, b, c

example:
(a,b,c) = (9,4,6)



The 6 = 3! leaves of tree give all possible permutations of the input array and correspond to all possible outcomes

The length of the path taken is the # of comparisons and proportional to running time of algorithm.

Worst-case running time $\propto$ "proportional to" height of tree

Lower Bound for Decision-Tree Sorting

Theorem: Comparison-based sorting requires $\Omega(n \log n)$ comparisons worst case

Proof:

- # leaves $\geq n!$ (# of permutations = possible outputs)
- binary tree with height h has # leaves $\leq 2^h$
- $2^h \geq n!$

$$
\begin{aligned}
h &\geq \log_2(n!) && \text{(log is monotonically increasing)} \\
&\geq \log_2\left(\left(\tfrac{n}{e}\right)^n\right) && \text{(Sterling)} \\
&= n \log_2 n - n \log_2 e \\
&= \Omega(n \log n)
\end{aligned}
$$

So, comparison-based sort can't be better than $n \log n$!! [But I can sort subsets of a deck of playing cards in $\Theta(n)$ time.]

There is no inconsistency — a linear sort isn't carried out through comparisons. More like each object "goes to pre-assigned place."

We will formalize this today and in recitation:

- Counting Sort
- Radix Sort

Counting Sort

Input: $A[1..n]$, with $\underline{A[j] \in \{0, 1, \ldots, k\}}$

sort from "limited set"

Output: $B[1..n]$ } sorted permutation of A

Storage: $C[0..k]$

Intuition

A: 4 1 3 4 3

	0	1	2	3	4
C:	0	1	0	$\cancel{1}$ 2	$\cancel{1}$ 2

B: 1 3 3 4 4

→ linear time

→ no comparisons made

Improvements

- Need to copy elements from A into B so can copy auxillary data
- Advantageous to add <u>stable</u> sorting, which preserves input order for equal elements

---

$\Theta(k)$ for $i \leftarrow 0$ to k
     $C[i] \leftarrow 0$
} Initialize array C to zero

$\Theta(n)$ for $j \leftarrow 1$ to n
     $C[A[j]] \leftarrow C[A[j]] + 1$
} Count # of each type of element in A. Store in C.

$\Theta(k)$ for $i \leftarrow 1$ to k
     $C[i] \leftarrow C[i] + C[i-1]$
} Make C cumulative, so $C[i]$ contains # of elements $\leq i$ (in sorted order)

$\Theta(n)$ for $j \leftarrow n$ downto 1
     $B[C[A[j]]] \leftarrow A[j]$
     $C[A[j]] \leftarrow C[A[j]] - 1$
} Copy input to proper place in output

+
$\Theta(n+k)$

	1	2	3	4	5
A:	4	1	3	4	3

$\longrightarrow$

	0	1	2	3	4
C:	0	1	0	2	2

"cumulative" counts

C: 0 1 1 3 5

B[3] will be our last "3"

B[5] will be our last "4"

Read backwards from end of A, copying each element into proper place in B

B: _ _ 3 _ _

update C[3] (so next "3" will go to B[2])

C: 0 1 1 2 5

& iterate.

- Achieves copy of elements (& auxillary data)
- STABLE sort (equal elements preserve input order)

Running Time Analysis

$T(n,k) = \Theta(n+k)$. If $k = O(n)$, then counting sort is $\Theta(n)$ time.

Radix Sort

Imagine want to sort d-digit number by sequentially sorting digits.

Wrong way: Sort digits most significant to least

1 2 1	1 2 1	1 1 2	2 1 1
2 1 1	1 1 2	2 1 1	1 2 1
1 1 2	2 1 1	1 2 1	1 1 2
sort 100s	sort 10s	sort 1s	

3-digit #s in wrong order (actually backwards, by coincidence only)

Right way: Sort digits least significant to most!!

1 2 1	1 2 1	2 1 1	1 1 2
2 1 1	2 1 1	1 1 2	1 2 1
1 1 2	1 1 2	1 2 1	2 1 1
sort 1s	sort 10s	sort 100s	

- Produces correct sorted order
- It is important that a stable sort is used.

Radix-Sort(A, d)
for $i \leftarrow 1$ to d
  use a stable sort to sort array A on digit i

where digit 1 is least significant and n is most significant

Running time: If stable sort is $\Theta(n+k)$, then Radix-Sort is $\Theta(d(n+k))$

Can choose to group digits in pairs, triples, etc. and sort on these rather than individual digits.

In general, sort $\underline{n}$ computer words of $\underline{b}$ bits each
- Each word can be viewed as having $d = \lceil b/r \rceil$ digits of $\underline{r}$ bits each

  Example: 32-bit word | 8 | 8 | 8 | 8 |

- Break each $\underline{b}$-bit word into $\underline{d}$ $\underline{r}$-bit pieces and each pass takes $\Theta(n + 2^r)$ time, so $\underline{d}$ passes is $\Theta(d(n+2^r)) = \Theta\left(\frac{b}{r}(n+2^r)\right)$
- Letting $r = \log n \Rightarrow \Theta(bn / \log n)$