QUIZ 1 Information Sheet - on Stellar site

\[ \text{To satisfy AVL invariant} \]

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

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) → merge sort, heap sort, AVL sort → \( \Theta(n \log n) \)
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:
$(a, b, c) = (9, 4, 6)$

$6 = 3!$ 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 $\propto L$ height of tree
**Lower Bound for Decision-Tree Sorting**

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

**Proof:**
- \( \# \text{leaves} \geq n! \) (\# of permutations + possible outputs)
- binary tree with height \( h \) has \( \# \text{leaves} \leq 2^h \)
- \( 2^h \geq n! \)
- \( h \geq \log_2 (n!) \) (\( \log \) is monotonically increasing)
- \( h \geq \log_2 ((\frac{n}{e})^n) \) (Sterling's)
- \( = n \log_2 n - n \log_2 e \)
- \( = \Omega (n \log n) \)

So, comparison-based sort can't be better than \( n \log n \)!! [But I can sort subsets of a deck of playing cards in \( O(n) \) 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 \( A[i] \in \{0, 1, ..., k\} \)

Output: \( B[1..n] \)

Storage: \( C[0..k] \)

sort from "limited set"

sorted permutation of \( A \)

Intuition

A: 4 1 3 4 3

B: 1 3 3 4 4

\[ \begin{array}{c|cccc} 0 & 1 & 2 & 3 & 4 \\ \hline \text{C:} & 0 & 1 & 0 & +2 & +2 \end{array} \]

\( \rightarrow \) linear time
\( \rightarrow \) no comparisons made

Improvements

- Need to copy elements from \( A \) into \( B \) so can copy auxiliary data
- Advantageous to add stable sorting, which preserves input order for equal elements
for $i \leftarrow 0$ to $k$
    \[ C[i] \leftarrow 0 \]  
} \{ Initialize array $C$ to zero \}

for $j \leftarrow 1$ to $n$

\[ C[A[j]] \leftarrow C[A[j]] + 1 \]  
} \{ Count # of each type of element in $A$. Store in $C$. \}

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) \}

for $j \leftarrow n$ down to $1$

\[ B[C[A[j]]] \leftarrow A[j] \]  
[\[ C[A[j]] \leftarrow C[A[j]] - 1 \] \} \{ Copy input to proper place in output \}

\[ \frac{1}{n-k} \]  
\[ 2 \]  
\[ 3 \]  
\[ 4 \]  
\[ 5 \]  
$A$:  
\[ 4 \]  
\[ 1 \]  
\[ 3 \]  
\[ 4 \]  
\[ 3 \]  
$C$:  
\[ 0 \]  
\[ 1 \]  
\[ 0 \]  
\[ 2 \]  
\[ 2 \]  
\[ \text{"cumulative" counts} \]

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

\[ B: \]  
\[ 3 \]  
\[ \text{update } C[3] \]  
\[ \text{(so next } \frac{1}{3} \text{ will go to } B[2]) \]

\[ \# \text{ iterate.} \]

- Achieves copy of elements (auxiliary 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

Right way: Sort digits least significant to most!!

- Produces correct sorted order
- It is important that a stable sort is used.
Radix-Sort \((A, d)\)

\[
\text{for } i \leftarrow 1 \text{ to } d
\]

use a stable sort to sort array \(A\) on digit \(i\)

where \(i = 1\) 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 \(n = \) computer words of \(b\) bits each

- Each word can be viewed as having \(d = \left\lceil \frac{\log \frac{b}{r}}{\log 2} \right\rceil\) digits of \(\leq b\) bits each

Example: 32-bit word \([8, 8, 8, 8, 8]\)

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

- Letting \(r = \log \frac{n}{\log n}\)