

Lecture 7: Linear-Time Sorting

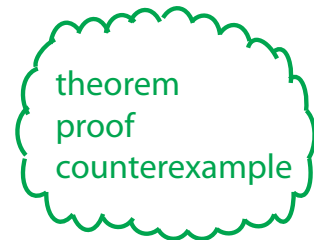
Lecture Overview

- Comparison model
- Lower bounds
 - searching: $\Omega(\lg n)$
 - sorting: $\Omega(n \lg n)$
- $O(n)$ sorting algorithms for small integers
 - counting sort
 - radix sort

Lower Bounds

Claim

- searching among n preprocessed items requires $\Omega(\lg n)$ time
 \implies binary search, AVL tree search optimal
- sorting n items requires $\Omega(n \lg n)$
 \implies mergesort, heap sort, AVL sort optimal



...in the comparison model

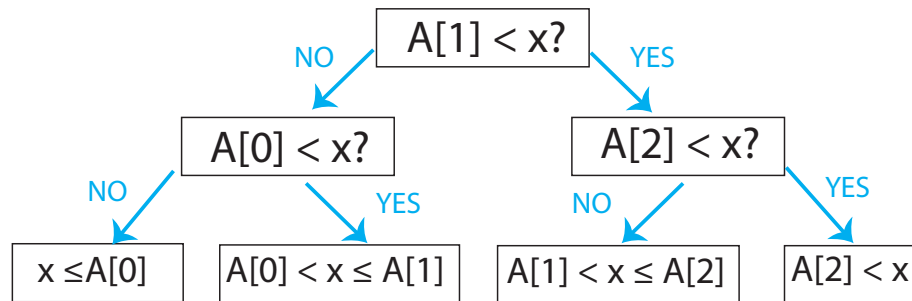
Comparison Model of Computation

- input items are black boxes (ADTs)
- only support comparisons ($<$, $>$, \leq , etc.)
- time cost = # comparisons

Decision Tree

Any comparison algorithm can be viewed/specified as a tree of all possible comparison outcomes & resulting output, for a particular n :

- example, binary search for $n = 3$:



- internal node = binary decision
- leaf = output (algorithm is done)
- root-to-leaf path = algorithm execution
- path length (depth) = running time
- height of tree = worst-case running time

In fact, binary decision tree model is more powerful than comparison model, and lower bounds extend to it

Search Lower Bound

- # leaves \geq # possible answers $\geq n$ (at least 1 per $A[i]$)
- decision tree is binary
- $\implies \text{height} \geq \lg \Theta(n) = \lg n \underbrace{+ \Theta(1)}_{\lg \Theta(1)}$

Sorting Lower Bound

- leaf specifies answer as permutation: $A[3] \leq A[1] \leq A[9] \leq \dots$
- all $n!$ are possible answers

- # leaves $\geq n!$

$$\begin{aligned}
 \implies \text{height} &\geq \lg n! \\
 &= \lg(1 \cdot 2 \cdots (n-1) \cdot n) \\
 &= \lg 1 + \lg 2 + \cdots + \lg(n-1) + \lg n \\
 &= \sum_{i=1}^n \lg i \\
 &\geq \sum_{i=n/2}^n \lg i \\
 &\geq \sum_{i=n/2}^n \underbrace{\lg \frac{n}{2}}_{=\lg n - 1} \\
 &= \frac{n}{2} \lg n - \frac{n}{2} = \Omega(n \lg n)
 \end{aligned}$$

- in fact $\lg n! = n \lg n - O(n)$ via [Sterling's Formula](#):

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \implies \lg n! \sim n \lg n - \underbrace{(\lg e)n + \frac{1}{2} \lg n + \frac{1}{2} \lg(2\pi)}_{O(n)}$$

Linear-time Sorting

If n keys are integers ([fitting in a word](#)) $\in 0, 1, \dots, k-1$, can do more than compare them

- \implies lower bounds don't apply
- if $k = n^{O(1)}$, can sort in $O(n)$ time
[OPEN](#): $O(n)$ time possible for all k ?

Counting Sort

L = array of k empty lists	}	$O(k)$
— linked or Python lists		
for j in range n :	}	$O(n)$
$L[\underbrace{\text{key}(A[j])}] \text{.append}(A[j])$		
random access using integer key		
output = []	}	$O(\sum_i (1 + L[i])) = O(k + n)$
for i in range k :		
output.extend($L[i]$)		

Time: $\Theta(n + k)$ — also $\Theta(n + k)$ space

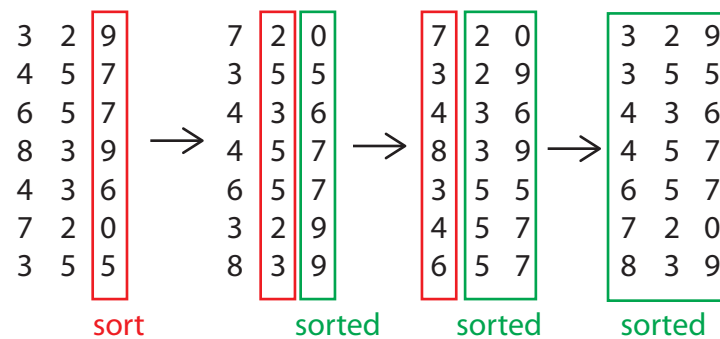
Intuition: Count key occurrences using RAM output <count> copies of each key in order ... but item is more than just a key

CLRS has cooler implementation of counting sort with counters, no lists — but time bound is the same

Radix Sort

- imagine each integer in base b
 $\implies d = \log_b k$ digits $\in \{0, 1, \dots, b - 1\}$
- sort (all n items) by least significant digit \rightarrow can extract in $O(1)$ time
- ...
- sort by most significant digit \rightarrow can extract in $O(1)$ time
sort must be stable: preserve relative order of items with the same key
 \implies don't mess up previous sorting

For example:



- use counting sort for digit sort
 - $\implies \Theta(n + b)$ per digit
 - $\implies \Theta((n + b)d) = \Theta((n + b) \log_b k)$ total time
 - minimized when $b = n$
 - $\implies \Theta(n \log_n k)$
 - $= O(nc)$ if $k \leq n^c$

MIT OpenCourseWare
<http://ocw.mit.edu>

6.006 Introduction to Algorithms
Fall 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.