6.006	Lecture 7	Sept. 29, 2011

TODAY: Linear-Time Sorting

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

theorem
proof
counterexample

Lower bounds: claim

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

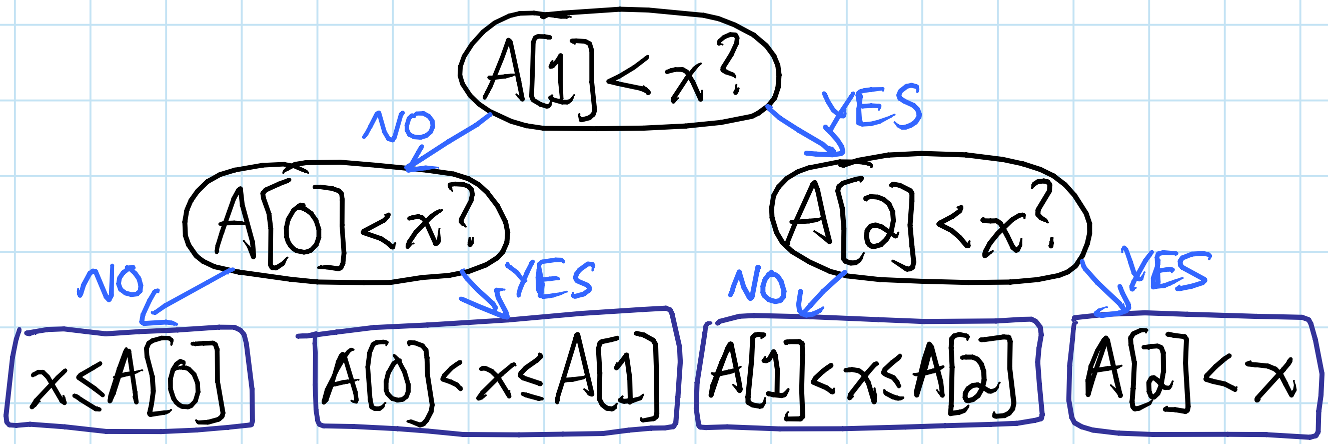
... in the comparison model

Comparison model of computation:

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

<u>Decision tree</u>: any comparison algorithm can be viewed/specified as a tree of all possible comparison outcomes & resulting output, for a particular n:

- e.g. 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

$\Rightarrow$ height $\geq \lg \Theta(n) = \lg n \underbrace{\pm \Theta(1)}_{\lg \Theta(1)}$

Sorting lower bound:

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

$\Rightarrow$ # leaves $\geq n!$

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

- in fact $\lg n! = n \lg n - O(n)$ via:

Sterling's formula: $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$\Rightarrow \lg n! \sim n \lg n - \underbrace{(\lg e) n + \frac{1}{2} \lg n + \frac{1}{2} \lg (2\pi)}_{O(n)}$

Linear-time sorting:

if n keys are integers $\in \{0, 1, \ldots, k-1\}$, ↗ fitting in a word
can do more than compare them
$\Rightarrow$ 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):
    L[key(A[j])].append(A[j]) $\} O(1) \} O(n)$
random access using integer key
– output = []
– for i in range(k):
    output.extend(L[i])
$\} O(\sum_i (1 + |L[i]|)) = O(k+n)$

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

$\Rightarrow d = \log_b k$ digits $\in \{0, 1, \dots, b-1\}$

- sort by least significant digit → can extract in $O(1)$ time
- ... → all n items
- sort by most significant digit

↳ sort must be stable: preserve relative order of items with the same key

$\Rightarrow$ don't mess up previous sorting

e.g.

3 2 9	⇒	7 2 0	⇒	7 2 0	⇒	3 2 9
4 5 7		3 5 5		3 2 9		3 5 5
6 5 7		4 3 6		4 3 6		4 3 6
8 3 9		4 5 7		8 3 9		4 5 7
4 3 6		6 5 7		3 5 5		6 5 7
7 2 0		3 2 9		4 5 7		7 2 0
3 5 5		8 3 9		6 5 7		8 3 9
sort		sorted		sorted		sorted

- use counting sort for digit sort

$\Rightarrow \Theta(n+b)$ per digit

$\Rightarrow \Theta((n+b)d) = \Theta((n+b)\log_b k)$ total time

- minimized when $b = n$

$\Rightarrow \Theta(n \log_n k)$

$= O(nc)$ if $k \le n^c$