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

theorem
proof
counterexample

Claim

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

...in the comparison model

Comparison Model of Computation

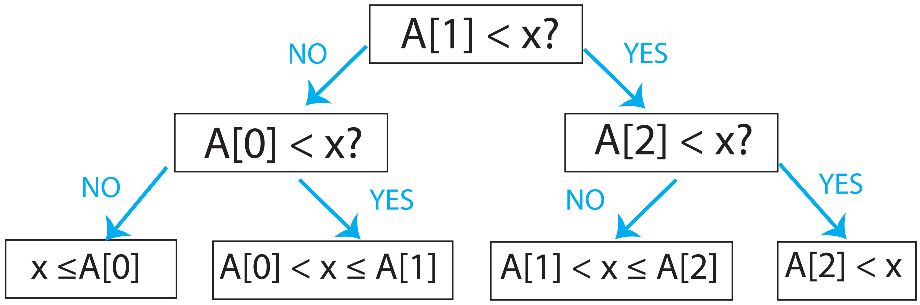
- input items are black boxes (ADTs)
- only support comparisons ($<, >, \leq$, etc.)
- time cost = # comparisons

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 $n = 3$:

| A[0] | A[1] | A[2] |



비교

결과

Decision Tree Algorithm

- internal node = binary decision (여기에서는 comparison만 다룸)
- leaf = output (algorithm is done) found answer
- root-to-leaf path = algorithm execution
- path length (depth) = running time
- height of tree = worst-case running time
  = Deepest leaf까지의 길이 (즉, longest route of root to leaf)

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

n개의 preprocessed item이 있고, 그 중에서 given item을 찾으려고 할 때,
비교만 가능한 경우 omega(lg(n))이 최소한 필요

Search Lower Bound

핵심은, preprocess 된 경우에도 lg(n)을 필요로 한다는 것!

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

1) Decision Tree는 binary
2) 그리고 반드시 최소 n개의 leaves를 가짐
   (답 하나 당 적어도 leaf 하나는 가지니까)

Tree의 height는 worst case에서의 running time

이 두 가지 특성을 조합했을 때,
최소 n개의 leaf를 가지는 Binary Tree의 높이는 lg(n)
= 탐색을 할 때 최소 lg(n)만큼은 필요하다,,,

Sorting Lower Bound

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

1) Decision Tree는 Binary
2) 그럼 몇 개의 leaf 노드를 가질까?
   (Leaf 노드: 순서에 따라 정렬된 것
   그렇다면, 정렬이 가능한 경우의 수는 n!)

여기서는 비교를 통해 정렬된 순서를 알아내는 것이 목표이고,
최소 nlg(n)을 필요로 함!

- # leaves $\geq n!$

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

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

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

Comparison Model은 이제 끝! 모델을 좀 더 크게 RAM으로 해보자: 배열 특정 원소 접근은 linear time임을 이용

Linear-time Sorting

If n keys are integers (fitting in a word) $\in 0, 1, \ldots, k-1$, 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?

1) n개의 키가 integer
2) 특정 범위(편의를 위해 non - negative)에 있음
3) 키는 word에 대응 = 상수 시간에 다룰 수 있음
4) 비교 외에 다른 것도 상수 시간 안에 할 수 있음

For not too big k, can sort in linear time! : 이를 위한 방법이 counting sort와 radix sort!

Counting Sort

만약, n개의 아이템이 key와 다른 무언가를 같이 가진다면...?

<original counting sort>
e.g.,) 3 5 7 5 5 3 6
(3,3) (5,5,5) (6) (7)

L = array of k empty lists
    — linked or Python lists
} $O(k)$

for j in range n:
    $\underbrace{L[\text{key(A[j])}]}$.append(A[j]) $\rightarrow O(1)$
    random access using integer key
} $O(n)$

1) 메모리에 배열을 할당(배열 크기=k)
2) 각 숫자가 몇개 씩 있는지 검사

original counting sort와 같은데, 배열이 아니라 list를 이용하는 방식

output = [ ]
for i in range k:
    output.extend($L[i]$) O(|L| + 1)
} $O(\sum_i (1 + |L[i]|)) = O(k+n)$

L[i]를 다 더한 건 항목의 개수

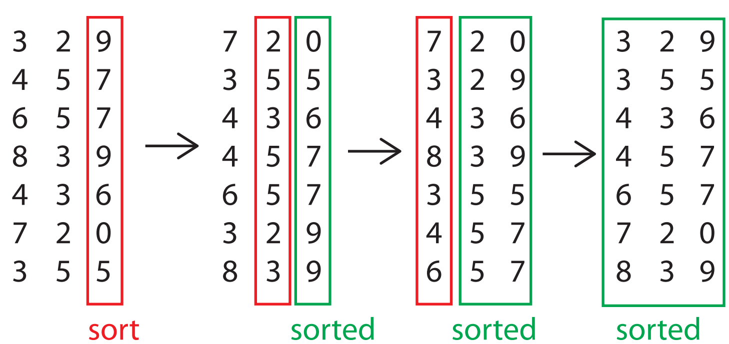
Time: $\Theta(n+k)$ — also $\Theta(n+k)$ space k가 n개랑 비슷하면 linear time을 가짐

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 integer를 여러 개의 column으로 나눈다고 생각 - b 진수
  $\Longrightarrow$ $d = \log_b k$ digits $\in \{0, 1, \ldots, b-1\}$ maximum k가 있다면, 자릿수는 #digit = logb(k)
- sort (all n items) by least significant digit $\rightarrow$ can extract in $O(1)$ time mod든 divide든 다 constant time
- ... d 자릿수니까 그만큼 반복
- 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
  $\Longrightarrow$ don't mess up previous sorting
  For example:



- use counting sort for digit sort
  normal: n+k인데, 여기서는 모든 자릿수가 0 ~ b - 1 사이에 있어서
  - $\Longrightarrow$ $\Theta(n+b)$ per digit
  - $\Longrightarrow$ $\Theta((n+b)d) = \Theta((n+b)\log_b k)$ total time
  - minimized when $b = n$ (b 크기를 키우고 싶은데, 또 n보다 크기를 원하지는 않음)
  - $\Longrightarrow$ $\Theta(n \log_n k)$
  - $= O(nc)$ if $k \leq n^c$