Sorting Lower Bound

Andreas Klappenecker
based on slides by Prof. Welch
Insertion Sort Review

- How it works:
  - incrementally build up longer and longer prefix of the array of keys that is in sorted order
  - take the current key, find correct place in sorted prefix, and shift to make room to insert it

- Finding the correct place relies on comparing current key to keys in sorted prefix

- Worst-case running time is $\Theta(n^2)$
Insertion Sort Demo

- [http://sorting-algorithms.com](http://sorting-algorithms.com)
Heapsort Review

- How it works:
  - put the keys in a heap data structure
  - repeatedly remove the min from the heap
- Manipulating the heap involves comparing keys to each other
- Worst-case running time is $\Theta(n \log n)$
Heapsort Demo

- http://www.sorting-algorithms.com
Mergesort Review

- How it works:
  - split the array of keys in half
  - recursively sort the two halves
  - merge the two sorted halves
- Merging the two sorted halves involves comparing keys to each other
- Worst-case running time is $\Theta(n \log n)$
Mergesort Demo

- http://www.sorting-algorithms.com
Quicksort Review

- How it works:
  - choose one key to be the pivot
  - partition the array of keys into those keys < the pivot and those ≥ the pivot
  - recursively sort the two partitions
  - Partitioning the array involves comparing keys to the pivot
  - Worst-case running time is Θ(n²)
Quicksort Demo

- http://www.sorting-algorithms.com
Comparison-Based Sorting

- All these algorithms are comparison-based
  - the behavior depends on relative values of keys, not exact values
  - behavior on \([1,3,2,4]\) is same as on \([9,25,23,99]\)
- Fastest of these algorithms was \(O(n \log n)\).
- We will show that's the best you can get with comparison-based sorting.
Decision Tree

- Consider any comparison based sorting algorithm
- Represent its behavior on all inputs of a fixed size with a decision tree
- Each tree node corresponds to the execution of a comparison
- Each tree node has two children, depending on whether the parent comparison was true or false
- Each leaf represents correct sorted order for that path
Decision Tree Diagram

first comparison:
check if $a_i \leq a_j$

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

second comparison
if $a_i \leq a_j$: check if $a_k \leq a_j$

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

third comparison
if $a_i \leq a_j$ and $a_k \leq a_j$: check if $a_x \leq a_y$

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td></td>
</tr>
</tbody>
</table>

second comparison
if $a_i > a_j$: check if $a_m \leq a_p$

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>YES</td>
</tr>
</tbody>
</table>
Insertion Sort

for j := 2 to n to
    key := a[j]
    i := j - 1
    while i > 0 and a[i] > key do
        a[i + 1] := a[i]
        i := i - 1
    endwhile
    a[i + 1] := key
endfor
Insertion Sort for $n = 3$

- $a_1 \leq a_2$?
  - YES
  - NO

- $a_2 \leq a_3$?
  - YES
  - NO

- $a_1 \leq a_3$?
  - YES
  - NO

- $a_2 \leq a_3$?
  - YES
  - NO

- $a_1 \leq a_3$?
  - YES
  - NO

- $a_2 \leq a_3$?
  - YES
  - NO

- $a_1 \leq a_3$?
  - YES
  - NO

- $a_2 \leq a_3$?
  - YES
  - NO

Insertion Sort for $n = 3$:

1. $a_1 \leq a_2$?
   - YES: $a_1 a_2 a_3$
   - NO: $a_2 a_1 a_3$

2. $a_2 \leq a_3$?
   - YES: $a_1 a_3 a_2$
   - NO: $a_3 a_1 a_2$

3. $a_1 \leq a_3$?
   - YES: $a_1 a_2 a_3$
   - NO: $a_2 a_3 a_1$

4. $a_2 \leq a_3$?
   - YES: $a_1 a_3 a_2$
   - NO: $a_3 a_2 a_1$
Insertion Sort for $n = 3$

```
<table>
<thead>
<tr>
<th>a1 ≤ a2?</th>
<th>a1 ≤ a3?</th>
<th>a2 ≤ a3?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>a1, a2, a3</td>
<td>a1, a3, a2</td>
<td>a2, a3, a1</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>a1, a3, a2</td>
<td>a3, a1, a2</td>
<td>a3, a2, a1</td>
</tr>
</tbody>
</table>
```
How Many Leaves?

- Must be at least one leaf for each permutation of the input
  - otherwise there would be a situation that was not correctly sorted
- Number of permutations of $n$ keys is $n!$.
- Idea: since there must be a lot of leaves, but each decision tree node only has two children, tree cannot be too shallow
  - depth of tree is a lower bound on running time
Key Lemma

Height of a binary tree with $n!$ leaves is $\Omega(n \log n)$.

Proof: The maximum number of leaves in a binary tree with height $h$ is $2^h$.

$h = 1$, $2^1$ leaves
$h = 2$, $2^2$ leaves
$h = 3$, $2^3$ leaves
Proof of Lemma

- Let $h$ be the height of decision tree, so it has at most $2^h$ leaves.
- The actual number of leaves is $n!$, hence
  
  $2^h \geq n!$
  
  $h \geq \log(n!)$
  
  $= \log(n(n-1)(n-1)...(2)(1))$

  $\geq (n/2)\log(n/2)$ \text{ by algebra}

  $= \Omega(n \log n)$
Finishing Up

- Any binary tree with \( n! \) leaves has height \( \Omega(n \log n) \).
- Decision tree for any c-b sorting alg on \( n \) keys has height \( \Omega(n \log n) \).
- Any c-b sorting alg has at least one execution with \( \Omega(n \log n) \) comparisons
- Any c-b sorting alg has \( \Omega(n \log n) \) worst-case running time.