Lower Bounds for Comparison-Based Sorting Algorithms (Ch. 8)

We have seen several sorting algorithms that run in $\Omega(nlgn)$ time in the worst case (meaning there is some input on which the algorithms run in *at least* $\Omega(nlgn)$ *time*).

- mergesort
- heapsort
- quicksort

In all comparison-based sorting algorithms, the sorted order results *only from comparisons between input elements.*

Is it possible for any comparison-based sorting algorithm to do better?

Lower Bounds for Sorting Algorithms

Theorem: Any *comparison-based sort must make* $\Omega(nlgn)$ *comparisons in the worst case to sort a sequence of n elements.* (Across all comparison-based sorting algorithms, no worst case runs faster than nlgn time.)

<u>But how do we prove this?</u>

We'll use the *decision tree model* to represent any sorting algorithm and then argue that no matter the algorithm, there is some input that will cause it to run in $\Omega(nlgn)$ time.

Question: How many ways are there to order n elements?

Binary tree

Recall that a binary tree is a tree data structure in which each node has at most 2 children, a left child and a right child.

Sources differ, but most authors agree that a proper binary tree is one in which every node has 0 or 2 children. A complete or full binary tree has every level completely filled.

Binary tree height and upper bound on number of leaves

The *height* of a node is the longest root to leaf path to that node.

Theorem: A full proper binary tree of height h has at most 2^h leaves.

<u>Basis</u>: a binary tree of height 0 has $2^0 = 1$ leaf

<u>Inductive hypothesis</u>: a binary tree of height k ≥ 1 has at most 2^k leaves.

<u>Inductive step</u>: Show a binary tree of height k+1 has at most 2^{k+1} leaves.

By the IHOP, we know that a binary tree of height k has at most 2^k leaves. A binary tree of height k+1 is a tree of height k in which every leaf has 2 children. So the number of leaves in a binary tree is $2(2^k) = 2^{k+1}$

The Decision Tree Model

Given any comparison-based sorting algorithm, we can represent its behavior on an input of size *n* by a decision tree. Note: we need only consider the *comparisons* in the algorithm (the other operations only make the algorithm take longer).

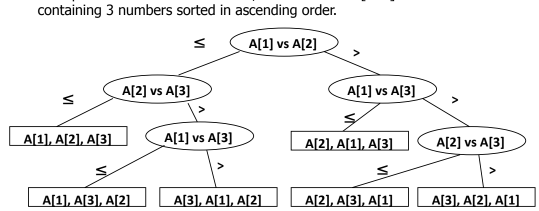
A decision tree is a binary tree.

- each internal node in the decision tree corresponds to one of the comparisons in the algorithm.
- start at the root and do first comparison (e.g., x:y)
  - if x ≤ y take left branch, if x > y take right branch, etc.
- each leaf represents one possible ordering of the input

⇒ One decision tree exists for each algorithm and input size

The Decision Tree Model

Example: decision tree with n = 3, with elements A[1..3] has 3! = 6 leaves containing 3 numbers sorted in ascending order.



Let the length of the *longest* root to leaf path in this tree be h

= worst-case number of comparisons

≤ worst-case number of operations of algorithm

The $\Omega(n\lg n)$ Lower Bound

Theorem: *Any decision tree for sorting n elements has height Ω(nlgn) (therefore, any comparison-based sorting algorithm requires Ω(nlgn) comparisons in worst case).*

Proof: Let *h* be the height of the tree. Then we know

- *the tree has at least (≥) n! leaves*
- *the tree is binary, so it has at most (≤) 2^h leaves*

# of leaves is upper bounded by 2^h and lower bounded by n!

$$2^h \geq \text{number of leaves} \geq n!$$

so we have:

$$2^h \geq n!$$

taking lg of both sides:

$$\lg(2^h) \geq \lg(n!)$$

$$h \geq \Omega(n\lg n) \quad \text{(Eq. 3.18)} \quad \square$$

Optimal Sorting Algorithms

- This lower bound proof tells us that heap-sort and merge-sort are asymptotically optimal comparison-based sorting algorithms.
- Randomized-Quick-Sort is asymptotically optimal with high probability.
- We also know that insertion-sort, selection-sort, and bubble-sort are not asymptotically optimal comparison-based algorithms.