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

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

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

- mergesort
- heapsort
- quicksort

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

Lower Bounds for Sorting Algorithms

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

But how do we prove this?

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 $\Omega(n \log n)$ time.

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

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 full or proper binary tree is one in which every node has 0 or 2 children.

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 – a proper binary tree.

A decision tree is a binary tree such that

- each internal node in the decision tree corresponds to one of the comparisons in the algorithm.
- each node represents a comparison of 2 values (e.g., $x : y$) s.t.
  - if $x \leq y$, take left branch, else if $x > y$, take right branch.
- each leaf in the decision tree represents one possible ordering of the input.

$\Rightarrow$ One decision tree exists for each algorithm and input size

Binary tree height and upper bound on number of leaves

The height of a node $x$ is the maximum number of edges on a path from a leaf to $x$.

Theorem: A proper binary tree (pbt) of height $h$ has at most $2^h$ leaves.

Basic: a pbt of height 0 has $2^0 = 1$ leaf
Inductive hypothesis: a pbt of height $k \geq 1$ has at most $2^k$ leaves.
Inductive step: Show a pbt of height $k+1$ has at most $2^{k+1}$ leaves.

By the IHOP, we know that a pbt of height $k$ has at most $2^k$. A pbt of height $k+1$ is a pbt of height $k$ in which one or more leaves has 2 children.

So the number of leaves in a pbt of height $2^{k+1}$ is at most $2(2^k) = 2^{k+1}$.

QED

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 (since other operations only make the algorithm run longer)
The $\Omega(n \log n)$ Lower Bound

**Theorem:** Any decision tree for sorting $n$ elements has height $\Omega(n \log n)$ (therefore, any comparison-based sorting algorithm requires $\Omega(n \log n)$ 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

The number of leaves is upper bounded by $2^h$ and lower bounded by $n!$

so we have:

$$2^h \geq n!$$

taking $\log$ of both sides:

$$\log(2^h) \geq \log(n!)
\Rightarrow
h \geq \Omega(n \log n) \quad \text{(Eq. 3.18)}$$