

Lower Bound on Sorting

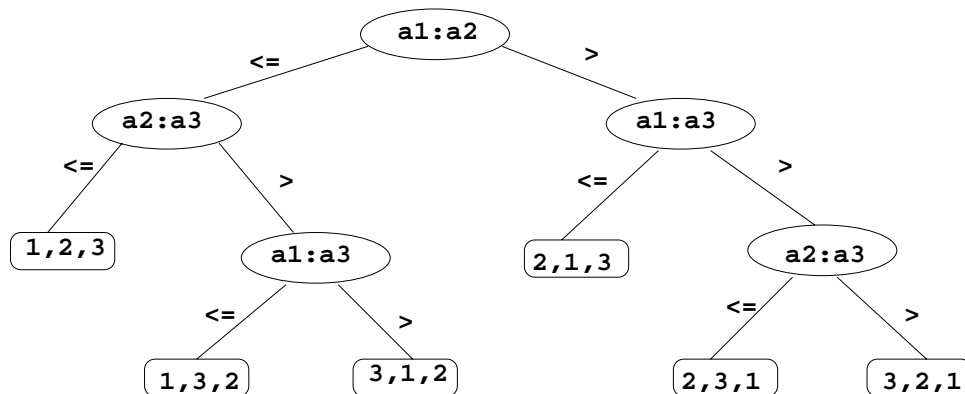
- Sorting numbers is a fundamental and common problem.
- We have seen several algorithms (MergeSort, InsertionSort, QuickSort) that run in time $O(n \log n)$.
- Is this the best possible time? (asymptotically speaking) Or, is there some clever $O(n)$ time algorithm waiting to be discovered?
- Failing to find such an algorithm after much effort, people try to prove that none exists. A *lower bound* means that no algorithm with running time *less* than the lower bound is possible.
- Since lower bound rules out *any* conceivable algorithm (now or in the future) from beating a stated bound, one needs to specify rigorously what *kinds* of algorithms are ruled out.
- A large number of algorithms, including those for sorting, fall into a class called *Comparison Based Algorithms*.
- Comparison-Based algorithms make their decision (branchings) using tests such as “Is $A[i] > A[j]$?”

Sorting Lower Bound

- Notice that all the sorting algorithms discussed so far belong to this class.
- We will show that **no** comparison-based algorithm to sort n numbers can have worst-case asymptotic complexity less than $cn \log n$, for some constant c .
- Thus, $\Omega(n \log n)$ is a lower bound for sorting.
- Outside the Comparison Model there do exist algorithms that beat this lower bound. These algorithms lack the generality of the comparison model, however.
- Since our lower bound applies to *all* possible algorithms (within the model), we must take an abstract view of an algorithm. We don't have the luxury to dissect "every algorithm" statement-by-statement.
- The correct abstraction turns out to be a *binary decision tree*. That is, every comparison-based algorithm can be viewed as a decision tree. Each comparison corresponds to a node in this tree; and each execution of the algorithm corresponds to a path from the root (the first comparison) to a leaf (the last comparison). The information collected by the algorithm in traversing this path is sufficient to compute the answer that execution's input.

Lower Bound by Decision Tree

- Decision Tree has no relationship to the **recursion tree** for mergesort or quicksort.
- Decision tree *models* a comparison-based sorting alg A .
 1. one tree for each n ;
 2. each execution of A corresponds to a path in the tree;
 3. tree represents all possible executions of A .
- The output of a sorting algorithm is a **permutation** of $\{1, 2, \dots, n\}$. In other words, sorting “marks” the elements as “smallest,” “second smallest,” “third smallest,” etc.

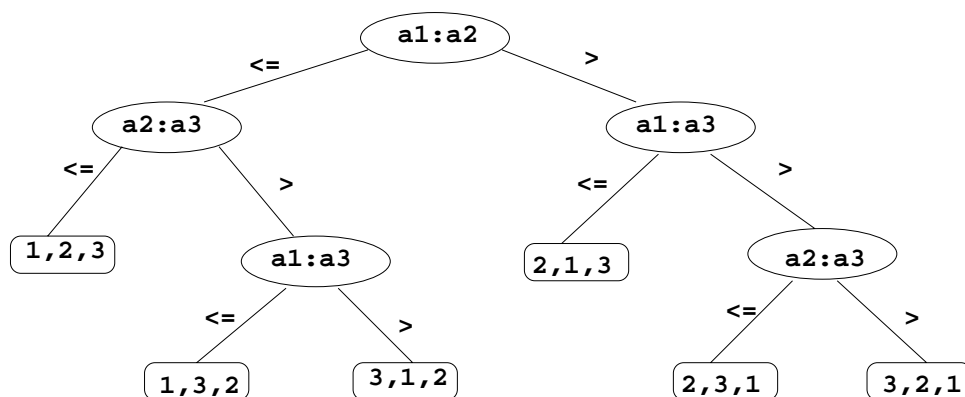


Lower Bound by Decision Tree

- Algorithms chooses which comparisons to make: outcome of earlier comparisons may determine the future comparison, and even randomization may be used.
- The main point, however, is: The algorithm's outcome depend **only** on the comparisons made or deduced by the algorithm.
- Only the relative order of numbers matter for comparison-based sorting, not their actual magnitudes. Thus, a (deterministic) sorting algorithm will behave exactly the same on the following two sequences:

$\{71, 33096, 456, 321, 2\}$ and $\{0.7, 330.9, 4.5, 3, 0\}$.

- The output is a **permutation** of $\{1, 2, \dots, n\}$.

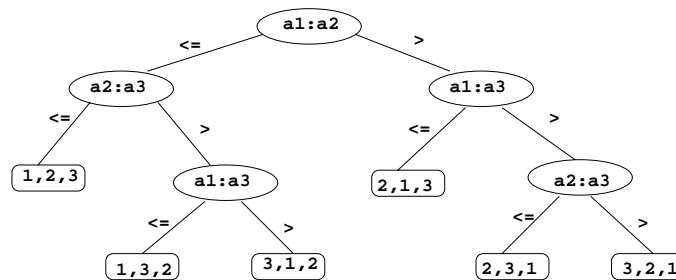


Lower Bound by Decision Tree

- How many permutations of $\{1, 2, \dots, n\}$ are there?

$$n! = n(n-1)(n-2)\cdots 2 \cdot 1.$$

- Each permutation must result in the algorithm taking a different path—otherwise the algorithm cannot distinguish between at least two permutations.



- Thus, the tree must have at least $n!$ leaves.
- The height of a tree is the length of its longest path.
- What's the height of a binary tree with $n!$ leaves?
- The height is a lower bound on the worst-case running time of the algorithm.

Lower Bound by Decision Tree

- Prove by induction that a tree of height H has at most 2^H leaves.
- Since the number of leaves is at least $n!$, we have

$$\begin{aligned}2^H &\geq n! \\H &\geq \lg n! \\&\geq \lg(n/e)^n && \text{(Sterling's Formula)} \\&\geq n \lg(n/e) \\&= n \lg n - n \lg e \\&= \Omega(n \log n)\end{aligned}$$

- In the worst-case the input to the algorithm is the one that causes the algorithm to execute the longest path of its decision tree.
- Since the argument is **independent** of any particular algorithm, **every** comparison-based algorithm has worst-case running time $\Omega(n \log n)$.
- Thus, MergeSort is an asymptotically optimal sorting algorithm.

What Lower Bound Tells Us

- First, it reassures us that widely used sorting algorithms are asymptotically optimal. Thus, one should not needlessly search for an $O(n)$ time algorithms (in the comparison-based class).
 - Second, decision tree proof is one of the few non-trivial lower-bound proofs in computer science.
 - Finally, knowing a lower bound for sorting also allows us to get lower bounds on other problems. Using a technique called **reduction**, any problem whose solution can indirectly lead to sorting must also have a lower bound of $\Omega(n \log n)$.
1. Straightforward application of decision tree method does not always give the best lower bound.
 2. [**Closest Pair Problem:**] How many possible answers (or leaves) are there? At most $\binom{n}{2}$. This only gives a lower bound of $\Omega(\log n)$, which is very weak. Using more sophisticated methods, one can show a lower bound of $\Omega(n \log n)$.
 3. [**Searching for a key in a sorted array:**] Number of leaves is $n + 1$. Lower bound on the height of the decision tree is $\Omega(\log n)$. Thus, binary search is optimal.