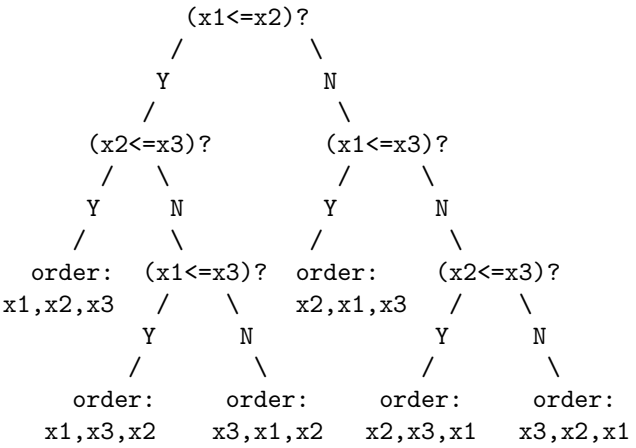
A Lower Bound for Sorting

If we restrict the kind of sorting algorithms we consider, we can give a lower bound of $\Omega(n \log n)$ on the number of operations to sort.

Comparison algorithms and decision trees

A *comparison algorithm* to sort uses *only* comparisons between pairs of elements to decide on a correct ordering. That is, we assume the inputs are $x_1, x_2, \ldots, x_n$ and the only type of operation the algorithm does that involves the inputs is to compare two of them, which we denote by the query $(x_i \leq x_j)$?, which is answered either Y or N. Suppressing all the other operations done by the sorting algorithm, we can represent a comparison algorithm by a *decision tree*, in which each internal node represents a pairwise comparison and each leaf node gives a correct ordering of the inputs. For example, here is a decision tree that represents an algorithm that sorts three inputs x_1, x_2, x_3.



The first comparison made by the algorithm is the one at the root (top node) of the tree: $(x_1 \leq x_2)$?. Depending on whether the answer is Y or N, the algorithm follows the left or right branch from that node. If the answer is Y, the algorithm proceeds to make the comparison $(x_2 \leq x_3)$?. If the answer to that comparison is also Y, then a correct ordering is $x_1 \leq x_2 \leq x_3$, so the algorithm halts and outputs this ordering. However, if the answer to the $(x_2 \leq x_3)$? comparison is N, the algorithm knows that $x_1 \leq x_2$ and $x_3 < x_2$, but it doesn't know how to order x_1 and x_3 yet. Hence, it makes an additional comparison $(x_1 \leq x_3)$? to determine the ordering of x_1 and x_3, and then halts and outputs the ordering x_1, x_3, x_2 or the ordering x_3, x_1, x_2 as appropriate. If instead the answer to the initial comparison $(x_1 \leq x_2)$? is N, the algorithm follows the steps specified in the right hand part of the tree.

This representation ignores all the other computation that is done by the sorting algorithm, and focuses just on the comparisons the algorithm does to determine a correct ordering of the inputs. Thus, both insertion sort and merge sort can be understood as comparison algorithms, by just focusing on the comparisons they do to determine the correct output.

A lower bound

The sorting algorithm represented by the decision tree above does 3 comparisons in the worst case to sort its 3 inputs. In particular, 4 of the 6 outcomes require 3 comparisons, and 2 of them require 2 comparisons. Is

there any comparison algorithm to sort 3 inputs using only 2 comparisons in the worst case? No, because the decision tree of any algorithm to sort 3 elements must have $3! = 6$ possible outcomes, one for each possible ordering of 3 elements. However, if the decision tree has a worst case of 2 comparisons, then it can have at most $2^2 = 4$ leaves. But $4 < 6$, so there would not be enough leaves to accomodate the 6 different outcomes.

This idea generalizes immediately to give a lower bound on the number of comparisons used by any comparison algorithm to sort n inputs. There are $n!$ possible different outcomes (one for each possible ordering of n things.) If the worst case number of comparisons in the decision tree is d, then there are at most 2^d leaves in the decision tree. In order to have at least one leaf for each possible outcome, we need

$$2^d \geq n!,$$

or, taking the logarithm base 2 of both sides:

$$d \geq \log_2(n!).$$

To deal with $\log_2(n!)$, we'd like an approximation (or at least a lower bound) on $n!$ that involves operations of addition, subtraction, multiplication, division, and exponentiation.

One way to get a lower bound (assuming n is even) is to observe that in

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

the first $n/2$ terms in this product are at least $n/2$, and the rest of them are at least 1, so we have

$$n! \geq (n/2)^{n/2}.$$

Taking the logarithm base 2 of both sides (using the monotonicity of log), we get

$$\log_2(n!) \geq (n/2)\log_2(n/2),$$

or

$$\log_2(n!) \geq (n/2)(\log_2(n) - 1).$$

This is enough to show that the worst case number of comparisons done by any comparison algorithm that sorts n inputs is bounded below by $\Omega(n \log n)$. (Note that because we are ignoring multiplicative constants, the base of the logarithm does not need to be specified.) Thus, no comparison algorithm can use asymptotically fewer comparisons than mergesort uses.

For a more accurate lower bound, we can use an inequality based on Stirling's approximation to the factorial. In particular, the limit as n goes to infinity of the quotient of $n!$ and the expression $\sqrt{2\pi n}(n/e)^n$ is 1. Taking the logarithm of this expression, we get a lower bound of the form $n\log_2(n) - nlog_2(e)$ plus lower order terms.

A non-comparison sorting algorithm

To see the kinds of things we can do if we don't limit ourselves to pairwise comparisons, consider the following case. We need to sort n integers whose values are between 1 and $2n$. We initialize an array $A[i] = 0$ for $i = 1, 2, \ldots, 2n$. Then we read in the inputs $x_1, x_2, \ldots, x_n$, and increment $A[x_i]$ by 1 when we read in x_i. (Here we are using the input as an index into an array rather than comparing two inputs.) Finally, for $i = 1, \ldots, 2n$, we output $A[i]$ copies of i. This gives an $O(n)$ time algorithm to sort n numbers in this case. This is not a comparison algorithm for sorting, so the lower bound does not apply to it. It also makes rather restrictive assumptions about the numbers to be sorted.