10.2 Decision Trees

Many important algorithms, especially those for sorting and searching, work by comparing items of their inputs. We can study the performance of such algorithms with a device called the *decision tree*. As an example, Figure 10.1 presents a decision tree of an algorithm for finding a minimum of three numbers. Each internal node of a binary decision tree represents a key comparison indicated in the node, e.g., $k < k'$. The node's left subtree contains the information about subsequent comparisons made if $k < k'$, while its right subtree does the same for the case of $k > k'$. (For the sake of simplicity, we assume throughout this section that all input items are distinct.) Each leaf represents a possible outcome of the algorithm's run on some input of size n. Note that the number of leaves can be greater than the number of outcomes because, for some algorithms, the same outcome can be arrived at through a different chain of comparisons. (This happens to be the case for the decision tree in Figure 10.1.) An important point is that the number of leaves must be at least as large as the number of possible outcomes. The algorithm's work on a particular input of size n can be traced by a path from the root to a leaf in its decision tree, and the number of comparisons made by the algorithm on such a run is equal to the number of edges in this path. Hence, the number of comparisons in the worst case is equal to the height of the algorithm's decision tree.

The central idea behind this model lies in the observation that a tree with a given number of leaves, which is dictated by the number of possible outcomes, has to be tall enough to have that many leaves. Specifically, it is not difficult to prove that for any binary tree with l leaves and height h,

$$h \geq \lceil \log_2 l \rceil. \qquad \textbf{(10.1)}$$

Indeed, a binary tree of height h with the largest number of leaves has all its leaves on the last level (why?). Hence, the largest number of leaves in such a tree is 2^h. In other words, $2^h \geq l$, which immediately implies (10.1).

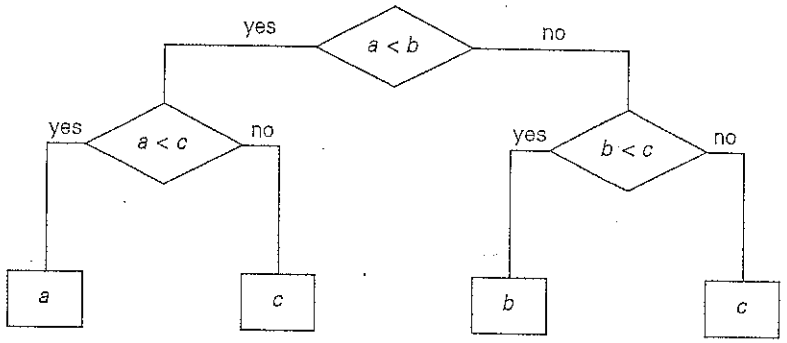

FIGURE 10.1 Decision tree for finding a minimum of three numbers

Inequality (10.1) puts a lower bound on the heights of binary decision trees and hence the worst-case number of comparisons made by any comparison-based algorithm for the problem in question. Such a bound is called the ***information-theoretic lower bound*** (see Section 10.1). We illustrate this technique below on two important problems: sorting and searching in an ordered array.

Decision Trees for Sorting Algorithms

Most sorting algorithms are comparison-based, i.e., they work by comparing elements in a list to be sorted. Moreover, with the notable exception of binary insertion sort (Problem 9 in Exercises 5.1), comparing two elements is the basic operation of such algorithms. Therefore, by studying properties of decision trees for comparison-based sorting algorithms, we can derive important lower bounds on time efficiencies of such algorithms.

We can interpret an outcome of a sorting algorithm as finding a permutation of the element indices of an input list that puts the list's elements in ascending order. For example, for the outcome $a < c < b$ obtained by sorting a list a, b, c (see Figure 10.2), the permutation in question is 1, 3, 2. Hence, the number of possible outcomes for sorting an arbitrary n-element list is equal to $n!$.

Inequality (10.1) implies that the height of a binary decision tree for any comparison-based sorting algorithm and hence the worst-case number of com-

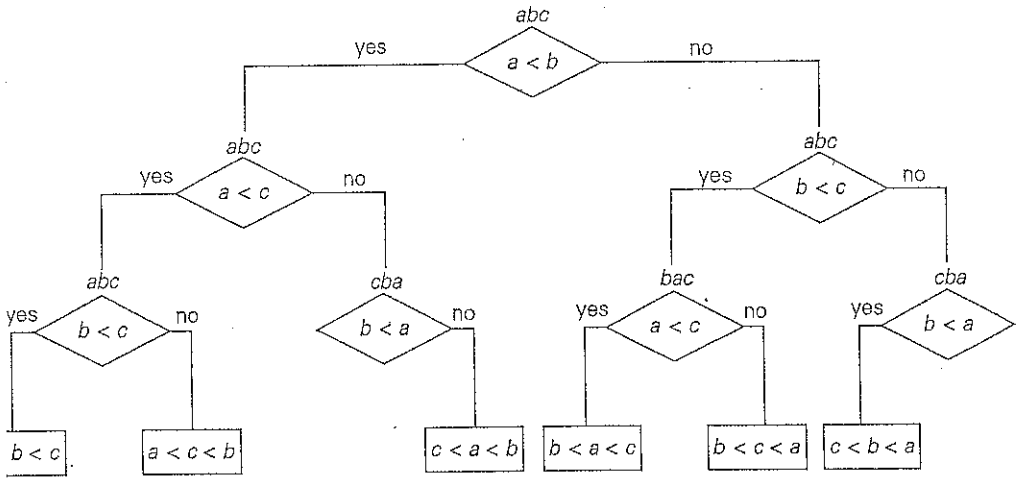

FIGURE 10.2 Decision tree for the tree-element selection sort. A triple above a node indicates the state of the array being sorted. Note two redundant comparisons $b < a$ with a single possible outcome because of the results of some previously made comparisons.

parisons made by such an algorithm cannot be less than $\lceil \log_2 n! \rceil$:

$$C_{worst}(n) \geq \lceil \log_2 n! \rceil. \tag{10.2}$$

Using Stirling's formula for $n!$, we get

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n = n \log_2 n - n \log_2 e + \frac{\log_2 n}{2} + \frac{\log_2 2\pi}{2} \approx n \log_2 n.$$

In other words, about $n \log_2 n$ comparisons are necessary to sort an arbitrary n-element list by any comparison-based sorting algorithm. Note that mergesort makes about this number of comparisons in its worst case and hence is asymptotically optimal. This also implies that the asymptotic lower bound $n \log_2 n$ is tight and therefore cannot be substantially improved. We should point out, however, that the lower bound of $\lceil \log_2 n! \rceil$ can be improved for some values of n. For example, $\lceil \log_2 12! \rceil = 29$, but it has been proved that 30 comparisons are necessary (and sufficient) to sort an array of 12 elements in the worst case.

We can also use decision trees for analyzing the average-case behavior of a comparison-based sorting algorithm. We can compute the average number of comparisons for a particular algorithm as the average depth of its decision tree's leaves, i.e., as the average path length from the root to the leaves. For example, for the three-element insertion sort whose decision tree is given in Figure 10.3, this number is $(2+3+3+2+3+3)/6 = 2\frac{2}{3}$.

Under the standard assumption that all $n!$ outcomes of sorting are equally likely, the following lower bound on the average number of comparisons C_{avg}

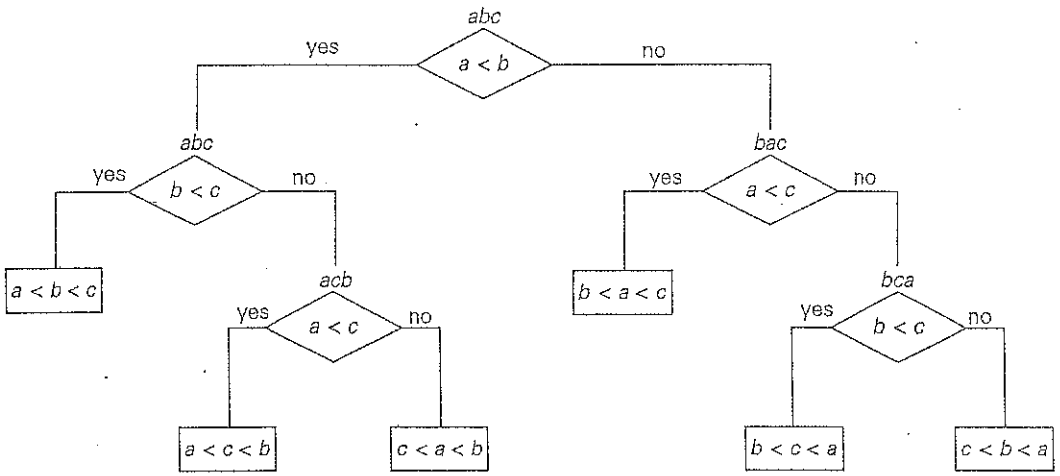

FIGURE 10.3 Decision tree for the three-element insertion sort

made by any comparison-based algorithm in sorting an n-element list has been proved:

$$C_{avg}(n) \geq \log_2 n!. \tag{10.3}$$

As we saw earlier, this lower bound is about $n \log_2 n$. You might be surprised that the lower bounds for the average and worst cases are almost identical. Remember, however, that these bounds are obtained by maximizing the number of comparisons made in the worst and average cases, respectively. For individual sorting algorithms, the average-case efficiencies can, of course, be significantly better than their worst-case efficiencies.

Excerpt from: A. Levitin, Introduction to the Design and Analysis of Algorithms (2003), pp. 339-342.