Illinois Institute of Technology
Department of Computer Science

Lecture 6: September 15, 2014

CS 430 Introduction to Algorithms
Fall Semester, 2014

1 Lower Bounds on Sorting

All of the sorting algorithms that we have seen have been based on element comparisons $a_i : a_j$ of the items being sorted; all of them use $\Omega(n \log n)$ such comparisons. Today we'll see that this is no coincidence: Any sorting algorithm based on comparisons of the elements being sorted must use that many element comparisons in both the worst case and on the average.

In this discussion we will consider only the case in which no two elements are equal, so such a comparison results in either a $>$ or $<$ answer, never $=$. Any sorting algorithm that works correctly for all inputs must work for this type of input.

For a fixed value of n, we can take such a sorting algorithm and expand it into a binary decision tree in which internal nodes are the comparisons and leaves are the sorted elements—that is, a leaf corresponds to a permutation of the input elements. For the algorithm to sort correctly, each of the possible $n!$ permutations must correspond to a unique leaf (if two or more permutations share a leaf, the algorithm cannot have determined the unique sorted order; if some permutation is missing, the algorithm cannot sort correctly on the corresponding input order).

1.1 Worst Case

Notice that the height of the decision tree is the number of comparisons used by the algorithm in the worst case. Let $S(n)$ denote the minimum number of comparisons required in the worst case by *any* sorting algorithm based on element comparisons. A binary tree of height h can have at most 2^h leaves, so

$$2^{S(n)} \geq n!$$

and hence

$$S(n) \geq \lg n! = \Omega(n \log n).$$

Thus *any* sorting algorithm based on element comparisons must use $\lg n!$ in the worst case.

1.2 Average Case

Now consider the minimum number of comparisons required in the average case by any sorting algorithm based on element comparisons. The average number of comparisons used is the average of the depths of all leaves: if we define $EPL(T)$, the *external path length* of a binary tree T, as the sum of the depths of all leaves, then $EPL(T)/n!$ gives the average number of comparisons used by the sorting algorithm represented by the decision tree T.

Lemma 1. *The binary tree T with the least external path length $EPL(T)$ has all of its leaves on levels l and $l+1$ for some value of l.*

Proof. Suppose not; let T be a binary tree of n leaves with minimal external path length with lowest (deepest) leaf at level L and shallowest (highest) leaf at level $l < L - 1$. Remove two sibling leaves from level L, making their parent internal node into a leaf; replace a leaf at level l with and internal node and two leaves as children. We now have a new tree T' of n leaves satisfying

$$EPL(T') = EPL(T) - 2L + 2(l + 1) < EPL(T)$$

because $l < L - 1$ implies $-2L + 2(l + 1) < 0$. That is, $EPL(T')$ is less than the smallest possible external path length in a tree with N leaves, contradicting our choice of T as a binary tree with minimal external path length. □

Lemma 2. *If $l_1, l_2, \ldots, l_N$ are the depths of the leaves in a binary tree, then*

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1.$$

This is called Kraft's Inequality*.*

Proof. By induction or by water-pouring. □

Now we can compute the minimum external path length in a binary tree with N leaves. Suppose, by the first lemma, that there are k leaves at level l and $N - k$ leaves at level $l + 1$, $1 \leq k \leq N$ (that is, all leaves may be at level l with none on level $l + 1$). The second lemma tells us that

$$k2^{-l} + (N - k)2^{-l-1} = 1$$

and hence

$$k = 2^{l+1} - N.$$

But $k \geq 1$ so that $2^{l+1} > N$; similarly, $k \leq N$ so $2^l \leq N$. Thus

$$l = \lfloor \lg N \rfloor .$$

Then $k = 2^{l+1} - N$ gives

$$k = 2^{\lfloor \lg N \rfloor + 1} - N$$

and the minimal external path length is thus

$$lk + (l + 1)(N - k) = N \lfloor \lg N \rfloor + 2N - 2^{\lfloor \lg N \rfloor + 1}.$$

Define $\delta(N) = \lg N - \lfloor \lg N \rfloor$ and hence $0 \leq \delta(N) < 1$. The minimal external path length is then

$$N \lg N + N(2 - \delta(N) - 2^{1-\delta(N)}).$$

The function $2 - \delta(N) - 2^{1-\delta(N)}$ is small on the interval $[0, 1]$; specifically, $0 \leq \delta(N) \leq 0.0861$ on this interval. We conclude that for a binary tree with $N = n!$ leaves, the external path length, and hence the minimum possible average sorting time (element comparisons) must be at least $EPL(T)/n!$ where T is a tree of $n!$ leaves of least external path length; in other words, $\lg n!$.

2 Sorting in Linear Time

The $\Omega(n \log n)$ lower bounds we just proved do not apply if we can avoid element comparisons. For example, if we know we are sorting a permutation of the numbers $1, \ldots, n$, then as we encounter each number we can put it directly into its correct place in sorted order.

CLRS (sections 8.2–8.4) describes three sorting algorithms that do make comparisons of elements, but rather sort based on information about the values of the input, their representation, or their distribution.