CMPSCI 187, Spring 2015 Discussion #12: A Lower Bound for Sorting: Individual Handout

Marc Liberatore and John Ridgway

22 April 2015

Remember that every student gets a copy of this handout, but every pair still hands in one response, on a separate answer sheet provided. Do not hand in this paper.

We have been looking at algorithms for sorting sets of elements, from a class that has a `compareTo` method. A comparison-based sorting algorithm works for any such class because it only deals with the elements by comparing one to another with this method. We've shown that simple algorithms use $O(n^2)$ comparisons to sort n elements, and we'll see that more sophisticated algorithms use $O(n \log n)$ comparisons.

It turns out that no comparison-based algorithm can sort using fewer than $O(n \log n)$ comparisons. We will show this using an adversary argument, like the one we used to show that searching a sorted array requires at least $\log n$ tests in the worst case. As you'll recall from our cheating guess-a-number game, any search algorithm for the numbers 1 through n that gets too-high or too-low responses, and uses fewer than $\log n$ tests, fails on some possible input, the one that the cheating program claims after the fact to have picked at the beginning.

We can similarly argue that since there are $n!$ possible linear orders for n elements, any comparison-based algorithm that asks fewer than $\log(n!)$ questions cannot in the worst case give the right answer. To be sure of that answer, the algorithm must have identified one of the $n!$ orders and ruled out the other $n! - 1$. The only tool we have to rule out orders is to compare one element against another.

For example, consider a list with the three elements a, b, and c. They might be originally in six possible orders: abc, acb, bac, bca, cab, or cba. If we find out that a comes before c, we know that the order must be abc, acb, or bac. If we then find that b comes before c, we know that it is either abc or bac. Testing a against b tells us which order it was. In this case we needed to do all three possible comparisons, but sometimes we can find out the order without doing all of them.

If we ask k possible yes-or-no questions, there are 2^k possible outcomes; we divide the $n!$ orders into 2^k categories. If our sorting algorithm is correct in every case, each of the orders is in its own category, so $n!$ must be less than or equal to 2^k. The number k, therefore, must be at least $\log_2(n!)$, which is $O(n \log n)$ by an argument we won't do today.

Your specific questions are on the other side of this handout.

Question 1: List the $4! = 24$ possible orders of four elements a, b, c, and d.

Question 2: Describe (in English) how you could list all the possible orders of $n + 1$ elements, if you had a list of all the possible orders of n elements. Given this idea, describe (in English) a recursive algorithm to list the orders of n elements. What is the base case of this recursion?

Question 3: Draw a binary tree where the internal nodes are comparisons of two elements from $\{a, b, c\}$, each internal node has two children for the two possible outcomes of the comparison, and the leaves are the $3! = 6$ orderings of the three elements.

Question 4: Repeat Question 3 for four elements. Your tree should have depth as small as possible while still having a leaf for each of the $4! = 24$ possible orders. You may be able to describe parts of the tree using symmetry rather than drawing out every node.