CSCI 314 – Fall 2014 Handout: Sample Proof
Algorithms
Michael H. Goldwasser
Saint Louis University Wednesday, 27 August 2014

We consider the problem of finding the second largest of n unique elements, in a setting in which the only information that can be gained is through a comparison of two elements. We will show that the best possible algorithm can located the second largest element using at most $\mathbf{n + \lceil \lg n \rceil - 2}$ **comparisons**. (We use notation $\lg n$ to mean $\log_2 n$.)

We begin by showing the upper bound, namely that there exist such an algorithm. We can find the second largest element by first running a standard, balanced "single-elimination tournament" to find the largest element. Since everyone except the winner loses exactly once, this part of our procedure uses exactly $n - 1$ comparisons.

Since all elements other than the largest must have been defeated, we know that the second largest element must have lost in a direct comparison with the largest element. We can run a secondary tournament for all elements which lost to the winner. The original tournament will require at most $\lceil \lg n \rceil$ rounds (giving first-round byes, if not exactly a power of two), and so at most that many elements will have lost to the winner. Therefore, the secondary tournament will use at most $\lceil \lg n \rceil - 1$ comparisons, and together, we have used at most $n + \lceil \lg n \rceil - 2$ comparisons.

Next, we show that *any* algorithm that correctly locates the second-largest element must use at least $n + \lceil \lg n \rceil - 2$ comparisons in the worst-case. For the sake of notation, let A denote the largest element and B the second largest. For any algorithm to be sure of its answer, it must have identified the group of $(n-2)$ elements which lie below B. In order to verify that all of these elements do indeed lie below B, each element of this set must have lost a comparison either to B or else to some other member of this group (as a comparison between such an element and A would not help place it below B). This implies that for any input, an algorithm must do at least $n - 2$ comparisons that do *not* involve the maximum element.

Now we will show that an adversary can force any algorithm to do at least $\lceil \lg n \rceil$ comparisons which do involve the maximum element, and therefore such an algorithm performs at least $n + \lceil \lg n \rceil - 2$ comparisons overall. To do so, we will consider the following adversary. Given a set of previous comparison answers, we will say that an answer to a new comparison is "known" if that answer is the same for all total orders which are consistent with the previous comparisons. Our adversary will maintain for each element x, the set $L(x)$ of elements which are "known" to be less than or equal to x based on previous comparisons. Initially, $L(x) = \{x\}$ for all elements, as the only thing we can be sure of is that x is less than or equal to itself. Now, when asked the question, "is $a \leq b$?", if this answer is known, our adversary will give the known answer. Otherwise, our adversary will say that a wins the comparison exactly when $|L(a)| \geq |L(b)|$. By answering questions in this way, we can be sure that there will always be at least one total ordering which is consistent with all of our adversary's answers.

Using this adversary, we claim that if a particular element x has been involved in exactly k comparisons, then $|L(x)| \leq 2^k$. We prove this by induction. As a base case, we have seen that when $k = 0$, $|L(x)| = 1 = 2^0$. For the inductive step, assume that $|L(x)| \leq 2^{k-1}$ after $k-1$ comparisons involving x, and we consider the k^{th} such comparison between x and some y. In the case where x loses the comparison or if it was already "known" that x was at least as large as y, then no new elements are added to $L(x)$, and so $L(x) \leq 2^{k-1} \leq 2^k$. If the answer was not previously known, and the adversary answers that x wins the comparison, then it must have been the case that $|L(y)| \leq |L(x)|$. If $L'(x)$ is equal to the resulting set after the k^{th} comparison, then we see that $L'(x) = L(x) \cup L(y)$. But these are the only elements which can be added to $L(x)$. Therefore, $|L'(x)| \leq |L(x)| + |L(y)|$ (some of $L(y)$ may have already been in $L(x)$). However, by induction we know that $|L(x)| \leq 2^{k-1}$, and by the adversary's rules, we know that $|L(y)| \leq |L(x)| \leq 2^{k-1}$, and therefore, $|L'(x)| \leq 2^{k-1} + 2^{k-1} = 2^k$.

Finally, we are ready to show that any algorithm will have made at least $\lceil \lg n \rceil$ comparisons which involve the maximum element. Notice that since any algorithm eventually knows the maximum element A, then it must be that $L(A) = n$. If A was involved in k comparisons, we know that $n = L(A) \leq 2^k$, and so rearranging, we see that $k \geq \lg n$. Since k must be integral, we see that $k \geq \lceil \lg n \rceil$, and therefore, the maximum element was involved in at least $\lceil \lg n \rceil$ comparisons. This completes our proof.