still open for other values of n.'

There are non-trivial non-evasive graph properties, but all known examples are non-monotone. One such property—'scorpionhood'—is described in an exercise at the end of this lecture note.

28.7 Finding the Minimum and Maximum

Last time, we saw an adversary argument that finding the largest element of an unsorted set of n numbers requires at least $n-1$ comparisons. Let's consider the complexity of finding the largest *and* smallest elements. More formally:

> Given a sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$ of n distinct numbers, find indices i and j such that $x_i = \min X$ and $x_j = \max X$.

How many comparisons do we need to solve this problem? An upper bound of $2n-3$ is easy: find the minimum in $n-1$ comparisons, and then find the maximum of everything else in $n-2$ comparisons. Similarly, a lower bound of $n-1$ is easy, since any algorithm that finds the min and the max certainly finds the max.

We can improve both the upper and the lower bound to $\lceil 3n/2 \rceil - 2$. The upper bound is established by the following algorithm. Compare all $\lfloor n/2 \rfloor$ consecutive pairs of elements x_{2i-1} and x_{2i}, and put the smaller element into a set S and the larger element into a set L. if n is odd, put x_n into both L and S. Then find the smallest element of S and the largest element of L. The total number of comparisons is at most

$$\underbrace{\left\lfloor \frac{n}{2} \right\rfloor}_{\text{build } S \text{ and } L} + \underbrace{\left\lceil \frac{n}{2} \right\rceil - 1}_{\text{compute } \min S} + \underbrace{\left\lceil \frac{n}{2} \right\rceil - 1}_{\text{compute } \max L} = \left\lceil \frac{3n}{2} \right\rceil - 2.$$

For the lower bound, we use an adversary argument. The adversary marks each element $+$ if it *might* be the maximum element, and $-$ if it *might* be the minimum element. Initially, the adversary puts both marks $+$ and $-$ on every element. If the algorithm compares two double-marked elements, then the adversary declares one smaller, removes the $+$ mark from the smaller element, and removes the $-$ mark from the larger one. In every other case, the adversary can answer so that at most one mark needs to be removed. For example, if the algorithm compares a double marked element against one labeled $-$, the adversary says the one labeled $-$ is smaller and removes the $-$ mark from the other. If the algorithm compares to $+$'s, the adversary must unmark one of the two.

Initially, there are $2n$ marks. At the end, in order to be correct, exactly one item must be marked $+$ and exactly one other must be marked $-$, since the adversary can make any $+$ the maximum and any $-$ the minimum. Thus, the algorithm must force the adversary to remove $2n-2$ marks. At most $\lfloor n/2 \rfloor$ comparisons remove two marks; every other comparison removes at most one mark. Thus, the adversary strategy forces any algorithm to perform at least $2n-2-\lfloor n/2 \rfloor = \lceil 3n/2 \rceil - 2$ comparisons.

28.8 Finding the Median

Finally, let's consider the *median* problem: Given an unsorted array X of n numbers, find its $n/2$th largest entry. (I'll assume that n is even to eliminate pesky floors and ceilings.) More formally:

> Given a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of n distinct numbers, find the index m such that x_m is the $n/2$th largest element in the sequence.

To prove a lower bound for this problem, we can use a combination of information theory and two adversary arguments. We use one adversary argument to prove the following simple lemma: