The following problem was shown to me by Adnan Aziz of the ECE department at UT.

It is required to maintain a queue $Q$ that supports the traditional “add-at-the-left” ($add$) and “remove-from-the-right” ($remove$) operations. Additionally, it should support a $max$ operation that returns the maximum value in $Q$. The inclusion of $max$ makes it impossible to implement each operation in constant time in the worst case. What is required is an implementation that runs in amortized $O(n)$ time for $O(n)$ operations.

Call an item a $peak$ if it is strictly larger than all item to its left in $Q$. Let $P$ be the sequence of peaks in the same order as they appear in $Q$. Observe that the peaks are monotone increasing from left to right in $P$. Thus, the rightmost peak is the largest peak and, hence, the largest item in $Q$. Further, the leftmost item of $Q$ is the smallest (leftmost) peak in $P$.

The operations on $Q$ are implemented as follows.

1. $max$: Return the value of the rightmost peak.
2. $remove$: Remove the rightmost item of $Q$. If it is a peak (then it is the rightmost peak), also remove it from (the right end) of $P$.
3. $add(x)$: Add $x$ at the left of $Q$. It is a peak, and the next peak is the first peak to its right in $P$ that is larger than $x$. Therefore, scan over $P$ from left to right removing all peaks that are less than or equal to $x$. Then add $x$ at the left of $P$.

The operations on $P$ are such that $P$ can be implemented as a double-ended queue. Then $max$ and $remove$ incur constant cost. Executing an $add(x)$ may take as many as $O(m)$ steps where $m$ is the length of $Q$; we show that it incurs only constant amortized cost.

For each $add(x)$ operation assign a cost of 1 unit to $x$ and to each peak that is removed from $P$. Since a peak becomes a non-peak at most once, we have the invariant that for each item in $Q$ a peak has a cost of 1 and a non-peak a cost of 2. A $remove$ operation incurs the cost associated with the item removed, which is at most 2. For $O(n)$ operations the queue length is at most $O(n)$ with a total cost of $O(n)$, and the $add(x)$ and $remove$ incur costs of $O(n)$.

For practical implementation a binary search tree is the preferred way of implementing $P$. 