1. Problem description omitted.

**Algorithm:** Our divide and conquer algorithm will be called $\text{findLast}(A)$. Say $A$ has length $N$. First we check if $N = 1$: if so, we return the only element. Otherwise, we look at the middle ($\lceil N/2 \rceil$) element of the array. Split the array into two halves: $L$, containing everything up to but not including the middle element, and $R$, containing everything else. If the middle element is less than the first, recurse on the first half calling $\text{findLast}(L)$. If the middle element is greater than the first, recurse on the second half calling $\text{findLast}(R)$.

**Proof of Correctness:** The key observation is that, since the elements of the array are in increasing order up to the location of the largest element, the largest element belongs to whichever half of the array starts with a larger number. We prove correctness using induction on the array size.

The base case is trivial. Let $N > 1$ and assume that for all $0 < k < N$, $\text{findLast}$ returns the largest integer for any circularly shifted array of length $k$. Let $A = [a_1, a_2, ..., a_N]$ be an array of length $N$. Upon execution of $\text{findLast}(A)$, we consider two cases:

- First suppose $a_1 > a_{\lceil N/2 \rceil}$. Then the largest element must be in the first half, so we recurse on that side. By the inductive hypothesis, this returns the largest integer.
- Next suppose $a_1 < a_{\lceil N/2 \rceil}$. Then the largest element must be in the second half, so we recurse on that side. Again by the inductive hypothesis, this returns the largest integer. $\square$

**Proof of Runtime:** The algorithm runs in $\Theta(\log(N))$ time, where $N$ is the size of the input array. The recurrence can be summarized as $T(N) = T(N/2) + d$, where $d$ is a constant. By the Master Theorem $T(N) \in \Theta(\log(N))$.

2. Problem description omitted.

Below is an answer at the bleeding edge of informality. Nevertheless, as you can sense, it is rigorous, i.e., it addresses everything meaningful. Observe the impact of the very first sentence. It contains the key observation enabling efficient solution.

Overall, try to mimick this example in terms of structure and, ideally, the one above in terms of presentation.

Since $A[1..n]$ is sorted, the first and second elements of each pair appear in odd and even indexes of the array $A$, respectively, up to the “loner” element, and in the opposite order beyond that. Therefore, by examining “the middle two” indices of the array we can determine if they loner element is in the left subarray (including the examined two elements) or the right subarray (including the examined two elements).

Concretely, if $n \leq 4$, we do a linear scan of the array i time $O(1)$. Otherwise, let $k = \lfloor \frac{n}{4} \rfloor \geq 1$ and consider the elements $(A[2k - 1], A[2k])$. If they are identical, then the “loner” element is at a position strictly greater than $2k$, otherwise at a position at most $2k$. As we discard (almost) half of the array in $O(1)$ time in each step we achieve $O(\log n)$ time algorithm.