1. store the numbers in a Trie. for each number i, query $\max_j a[j]\ xor\ a[i]$ takes $O(w)$, by walking down the Trie. total time $O(nw)$.
2. determine the result bit by bit. We can verify whether we can get an xor result with prefix t in $O(n)$ by hashing. total time $O(nw)$.
3. w-ary Trie with depth $O(\frac{w}{\log w})$, use (nonstandard?) word operations to walk down. $O(\frac{nw}{\log w})$.
The space complexity can be reduced to $O(\frac{nw}{\log w})$ without using hashing (the naïve implementation needs $O(\frac{nw^2}{\log w})$ space), by 2-level indexing technique. So this algorithm is deterministic.
4. construct a patricia tree of the array in $O(n)$ after sorting, by word operations (e.g. clz). the tree has $O(n)$ nodes and edges. recursively find the maximum xor by a function $f(T, e)$, which returns the maximum xor between an element in T and an element in the subtree below e, where T is a subtree in the patricia tree and e is an edge. during each recursion we will either remove a node or an edge, so by amortized analysis, the total running time after sorting is $O(n)$.

Q: Could you do this in $O(n)$ runtime?
A: No I can't.

References