Solution no. 4

1. Solved in class.

2. Let M be an algorithm for the element-distinctness problem, and fix any sequence $\mathbf{a} = (\mathbf{a_1}, \ldots, \mathbf{a_n})$ of distinct numbers. It is not hard to observe that when applied to $\mathbf{a}$, M must for every k, compare the k'th order-statistic in it to the $(k-1)$'th order-statistic and to the $(k+1)$'th order statistic. Otherwise the k'th element can be replaces by either the $(k-1)$'th or the $(k+1)$'th element, and M would still decide that all elements are distinct, as in the original sequence.

   Now let $\mathbf{b} = (\mathbf{b_1}, \ldots, \mathbf{b_n})$ be obtained from $\mathbf{a}$ by applying any permutation but the identity. Then there must be some k such that if a_i is the k'th order-statistic in $\mathbf{a}$ and a_j is the $(k+1)$'th order-statistic in it, then $b_i > b_j$ (although $a_i < a_j$!). To prove this, assume that there is no such k and prove that $\mathbf{a} = \mathbf{b}$.

   From the previous two observations it easily follows (prove!) that M runs differently on every permutation of $\mathbf{a}$, and therefore the tree which represents the running of M on inputs of size n with distinct elements has $n!$ leaves, each containing exactly one permutation. From a lemma mentioned in class, it follows that the average depth of the leaves in that tree is at least $\log(n!)$. This yields the desired result.

3. Solved in class.

4. Divide the array into pairs and compare the elements in each pair. Then divide the smaller elements from all comparisons into pairs, and compare each pair, and so on. After $n-1$ comparisons, the minimal element is found. Note that the minimal element took part in at most $\log n$ comparisons, and that at some point it must have been compared to the second order statistic (!). Therefore by finding the minimum within the elements compared to the minimum, the second order-statistic is found. Overall, this takes at most $n + \log n - 2 \leq n + \log n + 5$ comparisons.

5. Find the k'th order statistic among the distances from (a, b). Then print all the points whose distance from (a, b) is at most k. This gives $\Theta(n)$ complexity.