Advanced Algorithms. Assignment 3

Exercise 5.
This exercise shall illustrate by some minimalistic examples what randomization can do or not do.

We are given n elements with a hidden total ordering, and the only way to obtain information on it is to choose pairs of elements and compare them (as in the sorting problem). We consider mainly the case $n = 3$.

5.1. Explain why every deterministic algorithm has to do, in the worst case, all 3 comparisons, in order to determine the median of $n = 3$ elements.

5.2. Give a Las Vegas algorithm that determines the median of $n = 3$ elements using an expected number of comparisons that is strictly smaller than 3. – But do not make any probabilistic assumptions on the input, e.g., that all 6 possible orderings are equally likely. The assumed scenario is rather that we do not know anything about the input distribution, and the bound on the expected number of comparisons must hold for every possible ordering.

5.3. Now consider the problem of determining the minimum of $n = 3$ elements. How many comparisons does a deterministic algorithm need in the worst case? Can you improve this number by randomization also here? That is: Either give a Las Vegas algorithm with a smaller expected number of comparisons, or explain why this is not possible.

Optional: You might have reflections on larger n, or on generalized versions of the problem where the comparisons have different costs that are known in advance.

See reverse page.

Exercise 6.
Let G be an undirected graph. Every edge has a cost, and every pair of incident edges has also a turn cost. All these costs are given non-negative numbers. The problem is to compute a cheapest path that visits every node of G at least once. The start and end nodes are not prescribed. This problem naturally appears in practice in some manufacturing or maintenance procedures.
For clarity, we specify more formal details. Let $c(x, y)$ denote the cost of edge (x, y). For any two incident edges (x, y) and (y, z), let $t((x, y), (y, z))$ denote the turn cost. Now, the cost of a path $(v_1, \ldots, v_k)$ is defined as $\sum_{i=1}^{k-1} c(v_i, v_{i+1}) + \sum_{i=2}^{k-1} t((v_{i-1}, v_i), (v_i, v_{i+1}))$.
Describe an algorithm that solves this problem in $O^*(2^n)$ time, where n is the number of nodes. Prove its correctness and the time bound.