Obligatory Exercises

This document contains the list of obligatory exercises for the re-examinations of the 2013 offering of the course Efficient Algorithms and Problem Complexity. Please prepare your solutions as a single pdf document, and submit the document [here]. Note that you only need to solve the exercises corresponding to those you failed on during the ordinary examination. The exercises are not the same (except for one), but Exercise $n$ below corresponds to Exercise $n$ in the original list of exercises. If you want me to grade your solution before the due date, please send me an e-mail and I will do that asap.

Exercise 1 (due 2014-04-27)

This exercise is derived from Exercises 6.2.17-23 in the textbook by Johnsonbaugh and Schaefer.

Let $m3quicksort(a, i, j)$ be the version that uses the median of the three array cells $a[i]$, $a[(i+j)/2]$, and $a[j]$ in order to partition $a[i,...,j]$, if $i < j$ (instead of always using $a[i]$ for that purpose).

(1) Write down suitable pseudocode for $m3quicksort$. (If $m3quicksort$ uses the procedure $partition$ from the textbook without changes, you do not need to include that one.)

(2) Determine the recurrence relation that describes the running time of the algorithm $m3quicksort$, if it is applied to a sorted array of $n$ pairwise distinct elements. Argue convincingly why this recurrence relation indeed describes the running time of the algorithm in this special case. (In particular, you have to argue that the running time of each recursive call is governed by the same recurrence relation! This is somewhat similar to a proof by induction.)

(3) Use the recurrence relation to determine the running time of the algorithm on this type of input.

Exercise 2 (due 2014-04-27)

Consider a strongly connected digraph with positive edge weights. (A digraph is strongly connected if each pair of nodes is connected by a directed path.) Here is an example:
A spanning tree of a digraph is a directed tree, i.e., there is one node, the root, from which every other node is reachable on exactly one directed path. Clearly, every strongly connected weighted digraph has a spanning tree, and therefore it has a minimal spanning tree (MST). One may now attempt to apply Prim’s algorithm to determine such an MST $S$, as follows: Initially, let $S$ consist of a single node (which is arbitrarily chosen). Now, as long as the graph contains nodes not in $S$, expand $S$ by an edge $e$ such that
(a) the addition of $e$ to $S$ results in a directed tree and
(b) the weight of $e$ is minimal among all edges with this property.

For instance, starting with the rightmost node in the example above, the algorithm would choose the edges with weights 2, 1, 3 (in this order), resulting in an MST of the graph.

(1) Show that this algorithm does not always yield the correct result.
(2) Consequently, the correctness proof of the greedy rule for the undirected case cannot carry over to the directed case. Recall the invariant established by that proof: After any number of steps, $S$ is a subtree of an MST of the input graph. Explain why this is not true any more for directed graphs.

Exercise 3 (due 2014-04-27)

Give an example that illustrates Huffman codes and Huffman’s algorithm, consisting of the following parts:

(1) a set (or multiset) of frequencies,
(2) an optimal Huffman frequency tree that could result from applying Huffman’s algorithm to these frequencies, and
(3) another optimal Huffman frequency tree that could never result from applying Huffman’s algorithm to these frequencies, regardless of the specific implementation of the data structures used. Explain why this tree, which would be a correct solution of the problem, cannot result from applying the algorithm.

Note that you have to choose the frequencies in a way that makes (3) possible.

Exercise 4 (due 2014-04-27)

Let $f$ be a polynomial-time reduction of a decision problem $A$ to a decision problem $B$. We know that, if $B \in P$ then $A \in P$. Similarly, if $B \in NP$ then $A \in NP$. However, what about the other direction? Assume that $A \in NP$ and consider the following nondeterministic algorithms to decide whether $y \in B$:

1. “Guess” nondeterministically some $x$.
2. Verify that $f(x) = y$ by computing $f(x)$ in polynomial time and comparing it with $y$. If $f(x) \neq y$, reject.
3. Check (using the polynomial-time nRAM for $A$) whether $x \in A$ and return the answer.

Why does this not qualify as a proof that $B \in NP$? Explain!

Exercise 5 (due 2014-04-27)

Let $M$ be a deterministic random access machine $M$ that decides a problem $A$. For every input $x$, let $used_M(x)$ denote the set of addresses of those registers of $M$ that occur at least once as the target of some write instruction during the
computation that processes $x$. If the computation consists of $l$ steps, it can be represented as a sequence of $l + 1$ configurations $C_0, \ldots, C_l$, where

- $C_0$ is the initial configuration,
- $C_l$ is the final one, and
- each configuration is of the form $C_i = (\kappa_i, R_i)$, where $\kappa_i$ is the value of the program counter (the number of the next instruction to be executed), and a mapping $R_i: used_M(x) \rightarrow \mathbb{N}$, where $R_i(j)$ is the contents of register $R_j$ at that point in time.

Now, for $m \in \mathbb{N}$, let $|m|$ denote the number of bits of the binary representation of $m$. (Thus, $|0| = 1$ and $|m| = \lfloor \log m \rfloor + 1$ for $m > 0$.) We say that the memory space consumed by $M$ with input $x$ is

$$space_M(x) = \max\{ \sum_{j \in used_M(x)} |R_i(j)| \}.$$ 

Thus, $space_M(x)$ is the maximum number of bits ever stored in memory during the computation, not counting the bits that are only required for storing the input $x$. We say that $M$ runs in logarithmic space if $space_M(x) \in O(\log |x|)$ (where $|x|$ denotes the size of $x$). The class $L$ is the set of all decision problems that can be solved in logarithmic space.

Prove that $M$ runs in polynomial time if it runs in logarithmic space. Thus, $L \subseteq P$. (Hint: Ask yourself how many configurations $C_0, \ldots, C_l$ a computation of $M$ may consist of, i.e., how many steps there can be in the computation.)

**Exercise 6 (due 2014-04-27)**

Is the following problem NP-complete?

**Input:** A digraph $G = (V, E)$ and two nodes $u, v \in V$.

**Question:** Is there a path in $G$ from $u$ to $v$?

Note: One sentence is sufficient as an answer, but neither yes nor no alone qualify. Be careful and think! It is quite easy to answer this question too quickly and carelessly.