Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
M. Ahmadi, P. Bamberger, H. Ghodselahi, Y. Maus, P. Schneider

Theoretical Computer Science - Bridging Course
Summer Term 2017
Exercise Sheet 7 – Sample Solution

Exercise 1: $\mathcal{O}$-Notation Formal Proofs (2+2+2 points)

The set $\mathcal{O}(f)$ contains all functions that are asymptotically not growing faster than the function f (when additive or multiplicative constants are neglected). That is:

$$g \in \mathcal{O}(f) \iff \exists c \geq 0, \exists M \in \mathbb{N}, \forall n \geq M : g(n) \leq c \cdot f(n)$$

For the following pairs of functions, check whether $f \in \mathcal{O}(g)$ or $g \in \mathcal{O}(f)$ or both. Proof your claims (you do not have to prove a negative result $\notin$, though).

(a) $f(n) = 100n,\ g(n) = 0.1 \cdot n^2$

(b) $f(n) = \sqrt[3]{n^2},\ g(n) = \sqrt{n}$

(c) $f(n) = \log_2(2^n \cdot n^3),\ g(n) = 3n$ ***Hint**: You may use that $\log_2 n \leq n$ for all $n \in \mathbb{N}$.*

Sample Solution

(a) It is $100n \in \mathcal{O}(0.1n^2)$. To show that we require constants c, M such that $100n \leq c \cdot 0.1n^2$ for all $n \geq M$. Obviously this is the case for $c = 1000$ and $M = 1$.

(b) We have $g(n) \in O(f(n))$. Let $c := 1$ and $M := 1$. Then we have

$$\begin{aligned} & g(n) \leq c \cdot f(n) && (1)\\ \Leftrightarrow\ & \sqrt{n} \leq n^{2/3} && (2)\\ \Leftrightarrow\ & 1 \leq n^{1/6} && (3)\\ \Leftrightarrow\ & 1 \leq n && (4) \end{aligned}$$

The last inequality is satisfied because $n \geq M = 1$.

(c) $f(n) \in O(g(n))$ holds. We give $c > 0$ and $M \in \mathbb{N}$ such that for all $n \geq M : \log_2(2^n \cdot n^3) \leq c \cdot n$. As $cn \in O(g(n))$ holds for every constant $c > 0$ the result will follow with the transitivity of the O-notation.

$$\begin{aligned} & \log_2(2^n \cdot n^3)\\ &= \log_2(2^n) + \log_2(n^3)\\ &= n + 3 \cdot \log_2(n)\\ &\leq n + 3n = 4n. \end{aligned}$$

Thus $\log_2(2^n \cdot n^3) \leq c \cdot n$ for $n \geq M := 1$ and $c := 4$.

We also have that $g(n) \in O(f(n))$ holds because

$$g(n) = 3n \leq 3(n + 3 \cdot \log_2(n)) = 3(\log_2(2^n \cdot n^3)) = 3 \cdot f(n).$$

Thus with $c = 3$ and for $n \geq M := 1$ we have $g(n) \leq cf(n)$.

Exercise 2: Sort Functions by Asymptotic Growth (4 points)

Sort the following functions by asymptotic growth. Write $g <_{\mathcal{O}} f$ if $g \in \mathcal{O}(f)$. Write $g =_{\mathcal{O}} f$ if $f \in \mathcal{O}(g)$ *and* $g \in \mathcal{O}(f)$.

n^2	$\sqrt{n}$	2^n	$\log(n^2)$
3^n	n^{100}	$\log(\sqrt{n})$	$(\log n)^2$
$\log n$	$10^{100}n$	$n!$	$n \log n$
$n \cdot 2^n$	n^n	$\sqrt{\log n}$	n

Sample Solution

	$\sqrt{\log n}$	$<_{\mathcal{O}}$	$\log(\sqrt{n})$	$=_{\mathcal{O}}$	$\log n$	$=_{\mathcal{O}}$	$\log(n^2)$
$<_{\mathcal{O}}$	$(\log n)^2$	$<_{\mathcal{O}}$	$\sqrt{n}$	$<_{\mathcal{O}}$	n	$=_{\mathcal{O}}$	$10^{100}n$
$<_{\mathcal{O}}$	$n \log n$	$<_{\mathcal{O}}$	n^2	$<_{\mathcal{O}}$	n^{100}	$<_{\mathcal{O}}$	2^n
$<_{\mathcal{O}}$	$n \cdot 2^n$	$<_{\mathcal{O}}$	3^n	$<_{\mathcal{O}}$	$n!$	$<_{\mathcal{O}}$	n^n

Exercise 3: The class $\mathcal{P}$ (1+2+1+2 points)

$\mathcal{P}$ is the set of languages which can be decided by an algorithm whose runtime can be bounded by $p(n)$, where p is a polynomial and n the size of the respective input (problem instance). Show that the following languages ($\cong$ problems) are in the class $\mathcal{P}$. Since it is typically easy (i.e. feasible in polynomial time) to decide whether an input is well-formed, your algorithm only needs to consider well-formed inputs. Use the $\mathcal{O}$-notation to bound the run-time of your algorithm.

(a) PALINDROME:= $\{w \in \{0,1\}^* \mid w \text{ is a Palindrome}\}$

(b) LIST:=$\{\langle A, c\rangle \mid A$ is a finite list of numbers which contains two numbers x,y such that $x + y = c\}$.

(c) 3-CLIQUE := $\{\langle G\rangle \mid G$ has a *clique* of size at least 3$\}$

(d) 17-DOMINATINGSET := $\{\langle G\rangle \mid G$ has a *dominating set* of size at most 17$\}$

- A *dominating set* of a graph $G = (V, E)$ is a subset $D \subseteq V$ such that for every vertex $v \in V$: $v \in D$ or v adjacent to a node $u \in D$.
- A *clique* of a graph $G = (V, E)$ is a subset $Q \subseteq V$ such that for all $u, v \in Q : \{u, v\} \in E$.

Sample Solution

(a) We have already seen in exercise sheet 5 that the problem can be solved with a Turing machine with $\mathcal{O}(n^2)$ head movements. The same idea/algorithm shows that the problem is in $\mathcal{P}$.

(b) Assume that the length of A is n. We can simply go through all combinations of tuples of A with two for-loops with indexes i and j where i ranges from 1 to n and j ranges from $i+1$ to n. Then we can simply test whether $A[i] + A[j] == c$ and accept if this ever evaluates to true, otherwise we reject. This has a runtime of $\mathcal{O}(n^2)$ because the test can be done in $\mathcal{O}(1)$ and there are $\mathcal{O}(n^2)$ tuples.

(c) Let $G = (V, E)$ and $|V| = n$. Then we know $|E| = \mathcal{O}(n^2)$. Upon input G, we can enumerate all possible triples (v_1, v_2, v_3) such that $v_1 \neq v_2 \neq v_3 \neq v_1$. There exist at most $\binom{n}{3} = \mathcal{O}(n^3)$ such triples. For each such triple (v_1, v_2, v_3), we examine whether $(v_1, v_2) \in E$, $(v_1, v_3) \in E$, and $(v_2, v_3) \in E$. Since $|E| = O(n^2)$, this examination can be done in $O(n^2)$ time. If during the examination process, we find one triple that satisfies the requirement, we found a clique of size 3 accept G. Otherwise, when we finish examining all possible triple, we reject G since it does not contain a clique of size 3. The runtime of the above procedure is $\mathcal{O}(n^5)$, thus 3-CLIQUE $\in \mathcal{P}$.

(d) A dominating set of size 17 exists if and only if all nodes are covered by a subset of the nodes $D \subseteq V$ with $|D| = 17$. There are less than n^{17} sets with $|D| = 17$. We iterate through all of the sets (e.g., by using 17 nested for loops iterating over the identifiers of the node set if we number the nodes from 1 to n.). Then for each set we test whether it dominates the whole graph by testing whether every node not in the set has a neighbor in the set. If this is the case for some set we accept. We reject if it's not the case for all sets.

This test can be done in time $\mathcal{O}(n)$ for each other node by going through the respective part of the adjacency matrix. There are at most $|V \setminus D| \leq n$ nodes outside of D for a D with $|D| = 17$. So to check whether a specific D is a dominating set we only need $\mathcal{O}(n^2)$ time. In total the time complexity is $\mathcal{O}(n^{17} \cdot n^2) = \mathcal{O}(n^{19})$. Therefore 17-DOMINATINGSET $\in \mathcal{P}$

Exercise 4: Traveling Salesman Problem (TSP)

A *hamiltonian cycle* in a (complete) weighted graph is a cycle in a graph which contains every node exactly once and its length is the sum of the weights of its edges. The decision variant of the *traveling salesman problem* is given by

$$\text{D-TSP} := \{\langle G, k\rangle \quad | \text{undir., simple, weighted graph } G \text{ has a hamiltonian cycle of length at most } k\}.$$

You have seen in the lecture that *D-TSP* is $\mathcal{NP}$-complete. In the following we want to prove that a polynomial time algorithm for this decision problem would also imply a polynomial time algorithm for the corresponding optimization problem (this is in fact the case for most $\mathcal{NP}$-complete decision/optimization problems.)
The objective of the optimization variant of the *traveling salesman problem* is to find, for a (complete) given weighted graph G, a hamiltonian cycle with minimum length.

(a) Show how to use a polynomial number of repetitions of a *D-TSP* algorithm to determine the length of a shortest hamiltonian cycle of a given graph G.

(b) Use the previous result to show that if the decision variant of TSP is in $\mathcal{P}$ then the optimization variant of TSP is in $\mathcal{P}$ as well.

Hint: Go through the edges and try to find out whether an edge is needed for a shortest Hamiltonian cycle.

Sample Solution

(a) Let $G = (V, E)$ be the given graph. We know that no hamiltonian cycle is longer than the sum of the length of all edges which we denote with S. Now, for $k = 1, \ldots, S$ we invoke one instance of the *D-TSP* algorithm with $\langle G, k\rangle$. The smallest k that the algorithm accepts is the length of a shortest hamiltonian cycle of G.

(b) Let *OPT-TSP* applied to a graph G return the length of a shortest hamiltonian cycle of G (cf. the first part). If *D-TSP* can be solved in polynomial time then *OPT-TSP* runs in polynomial time as well.

To obtain a polynomial time algorithm for the optimization problem of TSP we go through the edges one by one and constantly remove edges from G. In the end the edges that remain in G form a hamiltonian cycle of shortest length. Let α be the length of a shortest hamiltonian cycle (we compute α by one invocation of *OPT-TSP* on G).

Now, we go through the edges one by one and constantly remove edges from the graph. Assume that we currently process edge e, we construct a new graph G' which is identical to the current graph G except for the weight of edge e, which is set to ∞ (or a sufficiently large number, e.g., a number larger than the sum of all weights). Now we run *OPT-TSP* on G' and compare the result with α. If the result is larger than α we know that e is contained in every shortest hamiltonian cycle of the current graph. We add it to the final hamiltonian cycle and keep the edge in G. If the result equals α (note that it cannot be smaller than α) then we know that there is a hamiltonian cycle of length α which does not use e. We delete e from G and continue with the next edge.

The runtime of this process is polynomial as there are polynomially many edges and each invocation of *OPT-TSP* takes polynomial time.