Problems Solved:

36	37	38	39	40

Name:

Matrikel-Nr.:

Problem 36. Is there a Turing machine M over the alphabet $\Sigma = \{0, 1\}$ that can multiply two 128 bit integers in $O(1)$ steps? The Turing machine would get the binary representations of two integers, i. e., two words of length 128, as input and has to produce the product in binary form as output. Justify your answer. If the answer is *yes*, describe how the multiplication is done and find an upper bound for the number of steps, if the answer is *no*, explain why it is not possible.

Solution of Problem 36:

Yes. This is a finite problem. So we need 2^{256} states in order to read the 256 input letters from the tape. Each of these states basically represents the product. Then for each of these 2^{256} states we need 256 states to write out the answer to the tape. So basically we go over the input (256 steps) then back to the beginning of the tape and eventually write out the result. All this is done in roughly $3 \cdot 256$ steps. But this number is a constant. So the Turing machine returns a result in less than 2000 steps, that's clearly $O(1)$ no matter how the input length is taken into account.

Problem 37. True or false?

1. $5n^2 + 7 = O(n^2)$
2. $5n^2 = O(n^3)$
3. $4n + n \log n = O(n)$
4. $(n \log n + 1024 \log n)^2 = O(n^2 (\log n)^3)$
5. $3^n = O(9^n)$
6. $9^n = O(3^n)$

Prove your answers based on the formal definition of $O(f(n))$, i. e., for all functions $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ we have

$$g(n) = O(f(n)) \iff \exists c \in \mathbb{R}_{>0} : \exists N \in \mathbb{N} : \forall n \geq N : g(n) \leq c \cdot f(n).$$

Solution of Problem 37:

For each of the cases we prove or disprove according to the definition of $g(n) = O(f(n))$, i. e., we have to show the existence/non-existence of $c \in \mathbb{R}_{>0}$ and $N \in \mathbb{N}$ such that for all $n \geq N$ the inequality $g(n) \leq c \cdot f(n)$ holds.

1. $5n^2 + 7 = O(n^2)$ is true.
   Take $c = 12$ and $N = 2$, then for all natural $n \geq N$ it holds:

$$5n^2 + 7 < 5n^2 + 7n^2 = 12n^2.$$

2. $5n^2 = O(n^3)$ is true.
   Take $c = 1$ and $N = 6$, then for all natural $n \geq N$ it holds:
   $$5n^2 < n^3.$$
3. $4n + n \log n = O(n)$ is false.
   The log function is a monotonically increasing function with the property that for any $y \in \mathbb{R}$ there exists an $x \in \mathbb{R}$ such that $\log(x) > y$. Therefore, there is no $c \in \mathbb{R}_{>0}$ and no $N \in \mathbb{N}$ such that for all $n \geq N$ the relation $(4 + \log n)n \leq cn$ holds.
4. $(n \log n + 1024 \log n)^2 = O(n^2 (\log n)^3)$ is true
   Since the base of the log function is not explicitly given, we must prove it for any base $b \in \mathbb{R}_{>0}$, i. e. $\log(n) = \log_b(n)$. Take $c = 2048$ and $N = \max(1024, b)$, then for all natural $n \geq N$ it holds $1 \leq \log(n)$ and
   $$(n \log n + 1024 \log n)^2 \leq n^2 (\log n)^2 \leq n^2 (\log n)^3.$$
5. $3^n = O(9^n)$ is true.
   Take $c = 1$ and $N = 1$, then for all natural $n \geq N$ it holds:
   $$3^n \leq 9^n.$$
6. $9^n = O(3^n)$ is false.
   For all $c \in \mathbb{R}_{>0}$ we can find $N \in \mathbb{N}$ such that for all $n \geq N$ it holds $9^n = 3^n \cdot 3^n > c3^n$.

Problem 38. Let $f, g, h : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Prove or disprove based on Definition 45 from the lecture notes.

1. $f(n) = O(f(n))$
2. $f(n) = O(g(n)) \implies g(n) = O(f(n))$
3. $f(n) = O(g(n)) \wedge g(n) = O(h(n)) \implies f(n) = O(h(n))$

Solution of Problem 38:

For each of the cases we prove or disprove according to the definition of $g(n) = O(f(n))$, i. e., we have to show the existence/non-existence of $c \in \mathbb{R}_{>0}$ and $N \in \mathbb{N}$ such that for all $n \geq N$ the inequality $g(n) \leq c \cdot f(n)$ holds.

1. $f(n) = O(f(n))$ is true.
   Take $c = 1$ and $N = 1$, then for all natural $n \geq N$ it holds $f(n) \leq cf(n)$.
2. $f(n) = O(g(n)) \implies g(n) = O(f(n))$ is false.
   Counter example: $f(n) = 3^n$, $g(n) = 9^n$. See proof in Problem 37 (5) and (6).
3. $f(n) = O(g(n)) \wedge g(n) = O(h(n)) \implies f(n) = O(h(n))$ is true.
   Assume $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Then by definition of the big-O notation, there exist $c_1, c_2 \in \mathbb{R}_{>0}$ and $N_1, N_2 \in \mathbb{N}$ such that for all $n \geq N_1$ $f(n) \leq c_1 g(n)$ and for all $n \geq N_2$ $g(n) \leq c_2 h(n)$. We can take $c_3 := c_1 C_2$ and $N_3 := \max(N_1, N_2)$ then we have $f(n) \leq c_1 g(n) \leq c_1 c_2 h(n)$ for all $n \geq N_3$. This is exactly what we have to show for the above implication to be true.

Problem 39. Write a LOOP program in the core syntax (variables may be only incremented/decremented by 1) that computes the function $f : \mathbb{N} \to \mathbb{N}$, $f(n) = 2^n$.

1. Count the number of variable assignments (depending on n) during the execution of your LOOP program with input n.
2. What is the time complexity (the asymptotic complexity of the number of variable assignments) of your program (depending on n)?
3. Is it possible to write a LOOP program with time complexity better than $O(2^n)$? Give an informal reasoning of your answer.
4. Optional. Let $l(k)$ denote the bit length of a number $k \in \mathbb{N}$. Let $b = l(n)$, i.e., b denotes the bit length of the input. What is the time complexity of your program depending on b, if every variable assignment $x_i := x_j + 1$ costs time $O(l(x_j))$?

   Hint: You must determine an O-notation for $s(n) = \sum_{k=0}^{2^n-1} l(k)$. Split this sum into $s(n) = \sum_{k=0}^{2^{n-1}-1} l(k) + \sum_{k=0}^{2^{n-1}-1} l(2^{n-1}+k)$. The number of bits of each term of the second sum is easy to determine. Compare the first sum with $s(n-1)$. Then continue by expanding $s(n-1)$ in the same way.

Solution of Problem 39:

```
x0 := 0;
x0 := x0 + 1;
LOOP x1 DO
    LOOP x0 DO
        x0 := x0 + 1;
    END;
END;
```

1. At the end of the program we have $x_0 = 2^n$. Since we start with $x_0 := 0$ there are exactly 2^n additional assignments, i.e. $2^n + 1$ assignments.
2. From the previous reasoning, we get a time complexity of $O(2^n)$.
3. Since the result must be 2^n and the only way in a loop program to reach this value is by adding 1 to a previous value, we must have at least 2^n assignments. So the complexity (depending on n) cannot be better. There is no way to design a divide and conquer algorithm with the primitive operations from Definition 23 (lecture notes).
4. We (roughly) have to determine the sum

$$s(n) = \sum_{k=0}^{2^n-1} l(k)$$

since the operation $x_0 := x_0 + 1$ is executed 2^n times. (We ignore decrements of the loop counters.)

Note that the above sum can be split in two halves. So we get

$$\begin{aligned}
s(n) &= \sum_{k=0}^{2^{n-1}-1} l(k) + \sum_{k=0}^{2^{n-1}-1} l(2^{n-1}+k) \\
&= \sum_{k=0}^{2^{n-1}-1} l(k) + \sum_{k=0}^{2^{n-1}-1} (n-1) \\
&= s(n-1) + (n-1)2^{n-1} \\
&= s(n-2) + (n-2)2^{n-2} + (n-1)2^{n-1} \\
&= \dots \\
&= s(0) + 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \dots + (n-2)2^{n-2} + (n-1)2^{n-1} \\
&= 1 + \sum_{k=1}^{n-1} k \cdot 2^k \\
&= 3 + (n-2)2^n \\
&= O(n2^n)
\end{aligned}$$

Since $n \approx 2^b$ we get: $O(2^b \cdot 2^{2^b})$ as the time complexity depending on $b = l(n)$.

Problem 40. Define *concrete* languages L_i $(i = 1, \dots, 4)$ over the alphabet $\Sigma = \{0, 1\}$ such that L_i has infinitely many words and $L_i \neq \Sigma^*$. The following properties must be fulfilled.

(i) There exists a (deterministic) Turing machine M_1 with $L_1 = L(M_1)$ such that there is a constant $K \in \mathbb{N}$ and every word $w \in L_1$ is accepted in less then K steps.

(ii) Every (deterministic) Turing machine M_2 with $L_2 = L(M_2)$ needs at least n steps to accept a word $w \in L_2$ with $|w| = n \in \mathbb{N}$.

(iii) Every (deterministic) Turing machine M_3 with $L_3 = L(M_3)$ needs at least n^2 steps to accept a word $w \in L_3$ with $|w| = n \in \mathbb{N}$.

(iv) Every (deterministic) Turing machine M_4 with $L_4 = L(M_4)$ needs at least 2^n steps to accept a word $w \in L_4$ with $|w| = n \in \mathbb{N}$.

By *concrete* language it is meant that your definition defines an explicit set of words (preferably of the form $L_i = \{w \in \Sigma^* \mid \dots\}$) and not simply a class from which to choose. In other words,

> Let $L_1 \neq \Sigma^*$ be an infinite language such that (i) holds.

does not count as a *concrete* language.
In each case (informally) argue why your language fulfills the respective conditions.
Note that the exercise asks about acceptance of a word, not the computation of a result.

Solution of Problem 40:

(i) $L_1 = \{0w \mid w \in \Sigma^*\}$ Check whether the first letter is 0 and stop.

(ii) $L_2 = \{1^n \mid n \in \mathbb{N}\}$

(iii) $L_3 = \left\{ww^{-1} \,\middle|\, w \in \Sigma^*\right\}$ Palindromes. A TM would have to check letter by letter. That amounts to roughly $O(n)$ steps for each of the n letters of w.

(iv) $L_4 = \{w \in \Sigma^* \mid \exists n \in \mathbb{N}, \exists M \text{ TM} : |w| = n \wedge w = \langle M \rangle \wedge M \text{ accepts } \varepsilon \text{ in } 2^n \text{ steps}\}$ A "proof" relies on the assumption that there is no other way to check whether M accepts ε in 2^n steps than by simulating M.