# Algorithmic
# Discrete Mathematics
# 2. Exercise Sheet

TECHNISCHE UNIVERSITÄT DARMSTADT

**Department of Mathematics**
**PD Dr. Ulf Lorenz**
Dipl.-Math. David Meffert

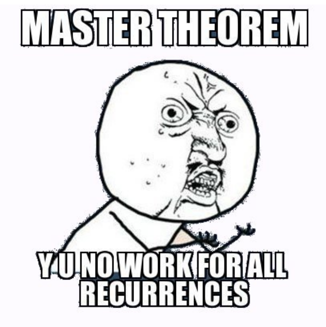
**SS 2012**
**2. and 3. May 2012**
Version of June 21, 2012

## Groupwork

**Exercise G1** (Master-Theorem)
Determine, if possible, fixed bounds for the complexities of the recurrences

(a) $T(n) = 4T(\frac{n}{2}) + n^3$,
(b) $T(n) = 4T(\frac{n}{2}) + n$,
(c) $T(n) = 4T(\frac{n}{2}) + n^2 \log n$,
(d) $T(n) = 4T(\frac{n}{2}) + n^2$.

*Hint:*




**Solution:** Throughout the whole exercise we have $\log_b a = \log_2 4 = 2$.

(a) We have $f(n) = n^3$. So $f(n) \in \Omega(n^{2+\varepsilon})$, because of $0 \le 1 \cdot n^3 \le f(n)$. Furthermore $4 \cdot f(\frac{n}{2}) = 4\frac{n^3}{8} = \frac{1}{2}n^3 \le c \cdot f(n)$, holds for $c = \frac{1}{2}$. So by the third case of the Master-Theorem we conclude $T(n) \in \Theta(n^3)$.

(b) We have $f(n) = n$. So $f(n) \in O(n^{2-\varepsilon})$ for $\varepsilon = 1$ because of $f(n) \le 1 \cdot n$. By the first case of the Master-Theorem we conclude $T(n) \in \Theta(n^2)$.

(c) We have $f(n) = n^2 \log n$. We immediately see $f(n) \notin O(n^{2-\varepsilon})$ and $f(n) \notin \Theta(n^2)$. We want to show that third case of the Master-Theorem can't be used either, because the second condition can't be fulfilled. Le $c \in (0,1)$. We get

$$\begin{aligned}
& 4 \cdot f(\frac{n}{2}) \le c \cdot f(n) \\
\Leftrightarrow\quad & 4 \cdot \frac{n^2}{4} \log(\frac{n}{2}) \le c \cdot n^2 \log(n) \\
\Leftrightarrow\quad & n^2 \log(\frac{n}{2}) \le c \cdot n^2 \log(n) \\
\Leftrightarrow\quad & \log(n) - \log(2) \le c \cdot \log(n) \\
\Leftrightarrow\quad & \log(n) - 1 \le c \cdot \log(n) \\
\Leftrightarrow\quad & -1 \le \log(n)(c-1) \\
\Leftrightarrow\quad & \frac{-1}{c-1} \ge \log(n)
\end{aligned}$$

This can't hold for all $n \in \mathbb{N}$ because $\{\log(n) \mid n \in \mathbb{N}\}$ is not bounded. So $c \in (0,1)$ ist not a possible choice. By this example we can see that although the Master-Theorem is quite powerful it can't be used for alle types of recurrences.

(d) We have $f(n) = n^2$. So $f(n) \in \Theta(n^2)$ because of $0 \leq 1 \cdot n^2 \leq f(n) \leq 1 \cdot n^2$. By the second case of the Master-Theorem we conclude $T(n) \in \Theta(n^2 \log n)$.

**Exercise G2** (Complexity)

(a) Let $f, t\colon \mathbb{N} \to \mathbb{R}$ be functions with $f \in O(t)$. Prove $O(f) + O(t) \subseteq O(t)$ and $O(f) + O(f) \subseteq O(t)$.

(b) Does $3^{3+n} \in O(3^n)$ hold?

(c) Does $3^{3n} \in O(3^n)$ hold?

(d) Show that $O(f) \cdot O(g) = O(f \cdot g)$ holds for $f, g\colon \mathbb{N} \to \mathbb{R}_+$.

*Remark:* For real valued functions $f, g\colon \mathbb{N} \to \mathbb{R}$ one just substitutes $f(n)$, $g(n)$ with $|f(n)|$, $|g(n)|$ in the definition of $O(g)$.

**Solution:**

(a) Let $g, h\colon \mathbb{N} \to \mathbb{R}$ with $g \in O(f)$ and $h \in O(t)$. By definition we get $n_g, n_h \in \mathbb{N}$, $c_g, c_h \in \mathbb{R}$ with

$$|g(n)| \leq c_g |f(n)| \quad \text{and} \quad |h(n)| \leq c_h |t(n)|$$

for all $n \geq n_g, n_h$. Furthermore by assumption we get $n_f \in \mathbb{N}$ and $c_f \in \mathbb{R}$ with $|f(n)| \leq c_f |t(n)|$ for all $n \geq n_f$. Putting things together we conclude

$$\begin{aligned} |g(n) + h(n)| &\leq c_g |f(n)| + c_h \cdot |t(n)| \leq c_g c_f |t(n)| + c_h \cdot |t(n)| \\ &= (c_g c_f + c_h)|t(n)| \end{aligned}$$

for all $n \geq \max\{n_g, n_h, n_f\}$. So we get $g + h \in O(t)$.

The second inclusion can be proved the same way or alternatively by showing $O(f) \subseteq O(t)$.

(b) We have $3^{3+n} \in O(3^n)$ because of $3^{3+n} = 27 \cdot 3^n \leq 27 \cdot 3^n$ for all $n \in \mathbb{N}$.

(c) The term $3^{3n} = 27^n$ is obviously not in $O(3^n)$.

(d) By definition we have

$$h \in O(f) \Longleftrightarrow \exists c_h, n_h \qquad h(n) \leq c_h f(n) \qquad \forall n \geq n_h$$

and

$$k \in O(g) \Longleftrightarrow \exists c_k, n_k \qquad k(n) \leq c_k g(n) \qquad \forall n \geq n_k.$$

So for $h \in O(f)$ and $k \in O(g)$ we have

$$(h \cdot k)(n) \leq c_k c_h (f \cdot g)(n) \qquad \forall n \geq \max\{n_h, n_k\}.$$

and therefore $h \cdot k \in O(f \cdot g)$. This proves the first inclusion.

For the second one let $l \in O(f \cdot g)$. This means there exist $n_l \in \mathbb{N}$ and $c_l \in \mathbb{R}$ with $l(n) \leq c_l (f \cdot g)(n)$ for all $n \geq n_l$. Now set $l = f \cdot \frac{l}{f}$. Obviously $f \in O(f)$ and by dividing the last inequality by $f(n)$ we get $\left(\frac{l}{f}\right)(n) \leq c_l g(n)$ for all $n \geq n_l$. So we have $\frac{l}{f} \in O(g)$.

**Exercise G3** (Algorithms)

(a) Given two algorithms $A$ and $B$:

- Algorithm $A$ has complexity $O(f)$.
- Algorithm $B$ has complexity $O(g)$.

We want to look at two new algorithms using $A$ and $B$.

**Algorithm 1**

```
INPUT : n ∈ ℕ
for i = 1,...,100 do
  run algorithm A
end for
for i= 1,..., n/2 do
  run algorithm B
end for
```

**Algorithm 2**

```
if n ≥ 30 then
  run algorithm A
else
  run algorithmus B
end if
```

We already know $f \in \Omega(g)$. Determine the best possible estimates for the runtime of both algorithms.

(b) Take a look at algorithm 3 and determine the best possible estimate for its runtime. Justify you answer.

**Algorithm 3**

```
INPUT : n ∈ ℕ
m = n
while m > 1 do
  for j = 1,...,n/2 do
    a=3 · b
    c = a +b
  end for
  m = 1/2 · m
end while
```

**Solution:**

(a) For the runtime of algorithm 1 we get $100 \cdot O(f) + \frac{n}{2} \cdot O(g)$. Because we already know $f \in \Omega(g)$ we can summarize that to $O(f \cdot h)$ with $h(n) = n$.

For algorithm 2 we notice that only the runtime for $n \to \infty$ is important. So only the second case of the **if**-part is relevant. Hence algorithm 2 has runtime $O(f)$.

(b) We go through the outer loop $\log n$ times. The inner loop we go through $\frac{n}{2}$ times. Ignoring any constant factors we get $O(n \log n)$ for the runtime of the algorithm.

**Exercise G4** (Sets)
Order the functions

$$n^2, \sqrt{n}, n!, n^n, n$$

by their complexity. Start with lowest complexity and use the $o$-notation. Determine $n_0$ dependend on $c > 0$ in every of those cases, too.

*Remark:*

$$f \in o(g) :\Longleftrightarrow \forall c > 0 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : 0 \leq f(n) < cg(n)$$

**Solution:**

$$\begin{aligned} \sqrt{n} &\in o(n) & n_0 &= \left\lceil \frac{1}{c^2} \right\rceil \\ n &\in o(n^2) & n_0 &= \left\lceil \frac{1}{c} \right\rceil \\ n^2 &\in o(n!) & n_0 &= \max\left\{6, \left\lceil \frac{1}{c} \right\rceil \right\} \\ n! &\in o(n^n) & n_0 &= \max\left\{3, \left\lceil \frac{1}{c} \right\rceil \right\} \end{aligned}$$

For explanation: In last two cases we have chosen $n_0 \geq 6$, $n_0 \geq 3$ because $n! > n^3$ holds for $n \geq 6$ and $n^n > n \cdot n!$ holds for $n \geq 3$. Should be a easy exercise to proof this.

## Homework

**Exercise H4** (Asymptotics) (14 points)

(a) Prove that for $r_1, r_2 \in \mathbb{R}_+$ we have $n^{r_1} \in O(n^{r_2})$ and $r_1^n \in O(r_2^n)$ iff $r_1 \leq r_2$.

(b) Prove the following statements for functions $f, t\colon \mathbb{N} \to \mathbb{R}$:

i. $O(f) + O(f) \subseteq O(f)$.

ii. $O(f) \cdot O(t) \subseteq O(f \cdot t)$.

iii. $\max\{f, t\} \in \Theta(f + t)$ for $f, t \geq 0$.

**Solution:**

(a) For all $n \in \mathbb{N}$ the statement $n^{r_1} \leq cn^{r_2}$ is equivalent to $n^{r_1 - r_2} \leq c$. The function $n^x$ is bounded iff $x \leq 0$, which means $r_1 \leq r_2$.

The second statement can be proved the same way. For all $n \in \mathbb{N}$ the statement $r_1^n \leq cr_2^n$ is equivalent to $\left(\frac{r_1}{r_2}\right)^n \leq c$. The function $x^n$ is bounded iff $x \leq 1$, which means $r_1 \leq r_2$.

(b) The proofs all work the same way in general by playing around with the definitions.

i. Let $g, h\colon \mathbb{N} \to \mathbb{R}$ with $g, h \in O(f)$. By definition we have $c_g, c_h \in \mathbb{R}$ and $n_g, n_h \in \mathbb{N}$ with

$$|g(n)| \leq c_g |f(n)| \quad \text{and} \quad |h(n)| \leq c_h |f(n)|$$

for all $n \geq n_g, n_h$. We conclude

$$|g(n) + h(n)| \leq |g(n)| + |h(n)| \leq (c_g + c_h)|f(n)$$

for all $n \geq \max\{n_g, n_h\}$, which means $g + h \in O(f)$.

ii. Let $g, h\colon \mathbb{N} \to \mathbb{R}$ with $g \in O(f)$ and $h \in O(t)$. By definition we have $c_g, c_h \in \mathbb{R}$ and $n_g, n_h \in \mathbb{N}$ with

$$|g(n)| \leq c_g |f(n)| \quad \text{and} \quad |h(n)| \leq c_h |t(n)|$$

for all $n \geq n_g, n_h$. We conclude

$$|g(n) \cdot h(n)| = |g(n)| \cdot |h(n)| \leq c_g |f(n)| \cdot c_h |t(n)| = (c_g c_h)|(f \cdot t)(n)|.$$

for all $n \geq \max\{n_g, n_h\}$, which means $g \cdot h \in O(f \cdot t)$.

iii. We want to proof the inequality

$$\max\{f, t\}(n) \geq \frac{1}{2}(f(n) + t(n)) \tag{1}$$

pointwise for all $n \in \mathbb{N}$ and therefore distinguish two cases. For every $n \in \mathbb{N}$ with $f(n) \geq t(n)$ we get

$$\max\{f, t\}(n) = f(n) = \frac{1}{2}(f(n) + f(n)) \geq \frac{1}{2}(f(n) + t(n)).$$

For all other $n \in \mathbb{N}$ with $t(n) \geq f(n)$ we get

$$\max\{f, t\}(n) = t(n) = \frac{1}{2}(t(n) + t(n)) \geq \frac{1}{2}(f(n) + t(n))$$

the same way. So by equation (1) we conclude $\max\{f, t\} \in \Omega(f + t)$. Because of the obvious inequality $\max\{f, g\}(n) \leq (f + g)(n)$ we get $\max\{f, g\} \in O(f + g)$. Hence we have $\max\{f, g\} \in \Theta(f + g)$.

**Exercise H5** (A sorting algorithm) (10 points)

The algorithm *SortList* sorts a sequence of numbers in ascending order.

**Algorithm 4** SortList(*list*)

INPUT: sequence of numbers, *list* $= a_1, \ldots, a_n$, $a_i \in \mathbb{N}$
**if** n <=1 **then**
  return *list*
**else**
  *leftlist* $= a_1, \ldots, a_{\lceil \frac{n}{2} \rceil}$
  *rightlist* $= a_{\lceil \frac{n}{2} \rceil + 1}, \ldots, a_n$
  return Sort(SortList(*lelftlist*),SortList(*rightlist*))
**end if**

---

**Algorithm 5** Sort(*rightlist*, *leftlist*)

INPUT: two sequences of numbers:
*rightlist* $= a_1, ..., a_l$, *leftlist* $= b_1, ..., b_k$, $a_i, b_i \in \mathbb{N}$
*newlist*
**while** *rightlist* and *leftlist* not empty **do**
  **if** first element of *leftlist* $<=$ first element of *rightlist* **then**
    append first element of *leftlist* to *newlist* and delete it from *leftlist*
  **else**
    append first element of *rightlist* to *newlist* and delete it from *rightlist*
  **end if**
**end while**
**while** *leftlist* not empty **do**
  append first element of *leftlist* to *newlist* and delete it from *leftlist*
**end while**
**while** *rightlist* not empty **do**
  append first element of *rightlist* to *newlist* and delete it from *leftlist*
**end while**
return *newlist*

---

(a) Sort the sequence $9, 10, 7, 3, 1, 2, 12, 9, 23$ in ascending order by using the algorithm *SortList*. Make sure to include detailed steps for the algorithm in your solution to indicate that you understand how it works.

(b) What is the runtime of the algorithm *SortList*?

**Solution:**

(a) We use the short term $S$ for the algorithm *Sort* and write $S$(rightlist; leftlist). For the algorithm *SortList* we use the short term $SL$. So we have

$$\begin{aligned}
SL(9,10,7,3,1,2,12,9,23) &\rightsquigarrow S(SL(9,10,7,3,1); SL(2,12,9,23)) \\
&\rightsquigarrow S(S(SL(9,10,7); SL(3,1)); S(SL(2,12); SL(9,23))) \\
&\rightsquigarrow S(S(S(SL(9,10); SL(7)); S(SL(3); SL(1))); S(S(SL(2); SL(12)); S(SL(9), SL(23)))) \\
&\rightsquigarrow S(S(S(S(SL(9), SL(10)); 7); S(3; 1)); S(S(2; 12); S(9,23))) \\
&\rightsquigarrow S(S(S(S(9,10); 7); S(3; 1)); S(S(2; 12); S(9,23))) \\
&\rightsquigarrow S(S(S(9,10; 7); S(3; 1)); S(S(2; 12); S(9,23))) \\
&\rightsquigarrow S(S(7,9,10; 1,3); S(2,12; 9,23)) \\
&\rightsquigarrow S(1,3,7,9,10; 2,9,12,23) \\
&\rightsquigarrow 1,2,3,7,9,9,10,12,23.
\end{aligned}$$

Now we want to show the the last step in detail and thereby demonstrate how the *Sort* algorithm works. We use $S$(rightlist; leftlist; newlist) to indicate all the steps and get

$$\begin{aligned}
S(1,3,7,9,10; 2,9,12,23; \emptyset) &\rightsquigarrow S(3,7,9,10; 2,9,12,23; 1) \rightsquigarrow S(3,7,9,10; 9,12,23; 1,2) \\
&\rightsquigarrow S(7,9,10; 9,12,23; 1,2,3) \rightsquigarrow S(9,10; 9,12,23; 1,2,3,7) \\
&\rightsquigarrow S(10; 9,12,23; 1,2,3,7,9) \rightsquigarrow S(10; 12,23; 1,2,3,7,9,9) \\
&\rightsquigarrow S(\emptyset; 12,23; 1,2,3,7,9,9,10) \rightsquigarrow S(\emptyset; 23; 1,2,3,7,9,9,10,12) \\
&\rightsquigarrow S(\emptyset; \emptyset; 1,2,3,7,9,9,10,12,23) \rightsquigarrow 1,2,3,7,9,9,10,12,23.
\end{aligned}$$

(b) The given algorithm is called *MergeSort* and is a recursive algorithm. We have $T(1) = 1$ and get the recurrence

$$\begin{aligned}
T(n) &= \underbrace{T(\frac{n}{2})}_{SortList(leftlist)} + \underbrace{T(\frac{n}{2})}_{SortList(rightlist)} + \underbrace{n}_{Sort} \\
&= 2T(\frac{n}{2}) + n.
\end{aligned}$$

By the Master-Theorem(second case) we conclude $T(n) \in \Theta(n \log n)$.

**Exercise H6** (6 points)

Given algorithm 6. What does the algorithm? Determine its runtime.

**Algorithm 6**

INPUT : $n \in \mathbb{N}$
K1 = 2;
K2 = n;
**while** K2 > K1 **do**
  K2 = n/K1
  **if** $\lceil K2 \rceil == K2$ **then**
    **return** K1
  **else**
    K1=K1+1
  **end if**
**end while**
**return** 0

**Solution:** The algorithm tests if a given number $n \in \mathbb{N}$ is prime. This is done by checking all possible divisors from $2, \ldots, \lfloor\sqrt{n}\rfloor$. The checking part works by dividing $n$ by $i \in \{2, \ldots, \lfloor\sqrt{n}\rfloor\}$ and looking if this fraction is a natural number. If a divisor is found the algorithm returns this divisor and otherwise it returns 0. In the case of output 0 the number $n$ is prime. The relevant part for the runtime (**while**-condition) is used $\sqrt{n}$ times, so the runtime is $\Theta(\sqrt{n})$).