University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Ahmadi, P. Schneider

# Algorithms Theory
# Sample Solution Exercise Sheet 1

**Due:** Monday, 5th of November, 2018, 14:15 pm

## Exercise 1: O-Notation *(3+4+5 Points)*

For a function $f(n)$, the set $\mathrm{O}\big(f(n)\big)$ contains all functions $g(n)$ that are *asymptotically* not growing faster than $f(n)$. The set $\Omega\big(f(n)\big)$ contains all functions $g(n)$ with $f(n) \in \mathrm{O}\big(g(n)\big)$. Finally, $\Theta\big(f(n)\big)$ contains all functions $g(n)$ for which $f(n) \in \mathrm{O}\big(g(n)\big)$ *and* $g(n) \in \mathrm{O}\big(f(n)\big)$. This is formalized as follows:

$$\begin{aligned}
\mathrm{O}\big(f(n)\big) &:= \{g(n) \mid \exists c > 0, n_0 \in \mathbb{N},\ \forall n \geq n_0 : g(n) \leq cf(n)\}\\
\Omega\big(f(n)\big) &:= \{g(n) \mid \exists c > 0, n_0 \in \mathbb{N}\ \forall n \geq n_0 : g(n) \geq cf(n)\}\\
\Theta\big(f(n)\big) &:= \{g(n) \mid \exists c_1, c_2 > 0, n_0 \in \mathbb{N}\ \forall n \geq n_0 : c_1 f(n) \leq g(n) \leq c_2 f(n)\}
\end{aligned}$$

State whether the following claims are correct or not. Prove or disprove with the definitions above.

(a) $n! \in \Omega\big(n^2\big)$

(b) $\sqrt{n^3} \in \mathrm{O}(n \log n)$ ***Hint:*** *For all* $\varepsilon > 0$ *there is an* $n_0 \in \mathbb{N}$ *such that for all* $n \geq n_0 :\ \log_2 n \leq n^{\varepsilon}$.

(c) $2^{\sqrt{\log_2 n}} \in \Theta(n)$

## Sample Solution

(a) The claim is true. We choose $n_0 = 4$ and $c = 1$. Then we have $n^2 \leq n \cdot (n-1) \cdot 2 \leq n!$ for all $n \geq 4$.

(b) The claim is false. Assume there exist $c > 0, n_0 \in \mathbb{N}$ such that for all $n \geq n_0 : \sqrt{n^3} \leq cn \log n$.

$$\begin{aligned}
& \sqrt{n^3} \leq cn \log n, \quad \forall n \geq n_0\\
\iff\ & n^{1/2} \leq c \log n, \quad \forall n \geq n_0\\
\iff\ & n^{1/4} \cdot n^{1/4} \leq c \log n, \quad \forall n \geq n_0\\
\implies\ & \big(n^{1/4} \leq c \quad \text{OR} \quad n^{1/4} \leq \log n\big), \quad \forall n \geq n_0
\end{aligned}$$

But $n^{1/4} \leq c$ is contradictory for all $n \geq c^4$. Additionally $n^{1/4} \leq \log n\ \forall n \geq n_0$ is a contradiction to the hint. Thus the assumption must have been false, and therefore $\sqrt{n^3} \notin \mathrm{O}(n \log n)$.

(c) The claim is false since $2^{\sqrt{\log_2 n}} \notin \Omega(n)\ (\supseteq \Theta(n))$. For a contradiction assume there exist $c > 0, n_0 \in \mathbb{N}$ such that

$$\begin{aligned}
& 2^{\sqrt{\log_2 n}} \geq cn, \quad \forall n \geq n_0\\
\iff\ & 2^{\sqrt{\log_2 n}\sqrt{\log_2 n}} \geq (cn)^{\sqrt{\log_2 n}}, \quad \forall n \geq n_0\\
\iff\ & n \ \geq\ (cn)^{\sqrt{\log_2 n}}, \quad \forall n \geq n_0\\
\overset{n \geq 16}{\implies}\ & n \ \geq\ (cn)^2, \quad \forall n \geq \max(n_0, 16)\\
\iff\ & \tfrac{1}{c^2} \ \geq\ n, \quad \forall n \geq \max(n_0, 16)
\end{aligned}$$

But this is contradictory for all $n \geq 1/c^2$.

## Exercise 2: Sort Functions by Asymptotic Growth *(5 Points)*

Use the definition of the O-notation to give a sequence of the functions below, which is ordered by asymptotic growth (ascending). Between two consecutive elements $g$ and $f$ in your sequence, insert either $\prec$ (in case $g \in O(f)$ *and* $f \notin O(g)$) or $\simeq$ (in case $g \in O(f)$ *and* $f \in O(g)$).

***Note***: *No formal proofs required, but you loose $\frac{1}{2}$ point for each error.*

| | | | |
|---|---|---|---|
| $n^2$ | $\sqrt{n}$ | $2^{\sqrt{n}}$ | $\log(n^2)$ |
| $2^{\sqrt{\log_2 n}}$ | $\log(n!)$ | $\log(\sqrt{n})$ | $(\log n)^2$ |
| $\log n$ | $10^{100}n$ | $n!$ | $n \log n$ |
| $2^n/n$ | $n^n$ | $\sqrt{\log n}$ | $n$ |

## Sample Solution

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\sqrt{\log n}$ | $\prec$ | $\log(\sqrt{n})$ | $\simeq$ | $\log n$ | $\simeq$ | $\log(n^2)$ |
| $\prec$ | $(\log n)^2$ | $\prec$ | $2^{\sqrt{\log_2 n}}$ | $\prec$ | $\sqrt{n}$ | $\prec$ | $n$ |
| $\simeq$ | $10^{100}n$ | $\prec$ | $n \log n$ | $\simeq$ | $\log(n!)$ | $\prec$ | $n^2$ |
| $\prec$ | $2^{\sqrt{n}}$ | $\prec$ | $2^n/n$ | $\prec$ | $n!$ | $\prec$ | $n^n$ |

## Exercise 3: Master Theorem for Recurrences *(5 Points)*

Use the *Master Theorem* for recurrences, to fill the following table. That is, in each cell write $\Theta\big(g(n)\big)$, such that $T(n) \in \Theta\big(g(n)\big)$ for the given parameters $a, b, f(n)$. Assume $T(1) \in \Theta(1)$. Additionally, in each cell note the case you used (1st, 2nd or 3rd by the order given in the lecture). We filled out one cell as an example.

***Note***: *You loose $\frac{1}{2}$ point if the complexity class is wrong and another $\frac{1}{2}$ if the case is wrong.*

| $T(n) = aT(\frac{n}{b}) + f(n)$ | $a = 16, b = 2$ | $a = 1, b = 2$ | $a = b = 3$ |
|---|---|---|---|
| $f(n) = 1$ | $\Theta(n^4)$, 1st | | |
| $f(n) = n^3$ | | | |
| $f(n) = n^4 \log n$ | | | |

## Sample Solution

| $T(n) = aT(\frac{n}{b}) + f(n)$ | $a = 16, b = 2$ | $a = 1, b = 2$ | $a = b = 3$ |
|---|---|---|---|
| $f(n) = 1$ | $\Theta(n^4)$, 1st | $\Theta(\log n)$, 3rd | $\Theta(n)$, 1st |
| $f(n) = n^3$ | $\Theta(n^4)$, 1st | $\Theta(n^3)$, 2nd | $\Theta(n^3)$, 2nd |
| $f(n) = n^4 \log n$ | $\Theta(n^4 \log^2 n)$, 3rd | $\Theta\big(n^4 \log n\big)$, 2nd | $\Theta(n^4 \log n)$, 2nd |

## Exercise 4: Peak Element *(5+4 Points)*

You are given an array $A[1 \dots n]$ of $n$ integers and the goal is to find a peak element, which is defined as an element in $A$ that is equal to or bigger than its direct neighbors in the array. Formally, $A[i]$ is a peak element if $A[i-1] \leq A[i] \geq A[i+1]$. To simplify the definition of peak elements on the rims of $A$, we introduce *sentinel-elements* $A[0] = A[n+1] = -\infty$.

(a) Give an algorithm with runtime $\mathrm{O}(\log n)$ (measured in the number of read operations on the array) which returns the position $i$ of a peak element.

(b) Prove that your algorithm always returns a peak element, give a recurrence relation for the runtime and use it to prove the runtime.

## Sample Solution

(a) **Algorithm 1** `Peak-Element(`$A, \ell, r$`)`

```
if ℓ = r then return A[ℓ]                                   ▷ base case
m ← ⌈(ℓ+r)/2⌉
if A[m] ≤ A[m+1] then
    return Peak-Element(A, m + 1, r)
else if A[m] ≤ A[m−1] then
    return Peak-Element(A, ℓ, m − 1)
else return A[m]                                            ▷ peak element found
```

A call of `Peak-Element(`$A, 1, n$`)` returns a peak element in $A$.

(b) We show the invariant that during each call of `Peak-Element(`$A, \ell, r$`)`, we have $A[\ell-1] \leq A[\ell]$ and $A[r] \geq A[r+1]$. Since $A[0], A[n+1] = -\infty$, this is obviously true for `Peak-Element(`$A, 1, n$`)`. During sub-calls of `Peak-Element(`$A, \ell, r$`)` this condition is maintained by the If-conditions and the recursive calls and the appropriate sub-array. This implies that we have found a peak element when $\ell = r$ (at the latest, but we may find one earlier).

During every recursive step, the considered sub-array is at most half the size of the previous one, thus the algorithm terminates eventually. Additionally, in each recurse step we make at most one recursive sub-call. Furthermore, in each recursive step we read at most 5 array entries. Thus we have $T(n) \leq T(n/2) + 5$ (reads), which solves to $T(n) \in \mathrm{O}(\log n)$ using the Master Theorem.

## Exercise 5: Frequent Numbers *(5+4 Points)*

You are given an Array $A[0 \ldots n-1]$ of $n$ integers and the goal is to determine frequent numbers which occur at least $n/3$ times in $A$. There can be at most three such numbers, if any exist at all.

(a) Give an algorithm with runtime $\mathrm{O}(n \log n)$ (measured in number of array entries that are read) based on the divide and conquer principle that outputs the frequent numbers (if any exist).

(b) Argue why your algorithm is correct, give a recurrence relation for the runtime and use it to prove the runtime.

## Sample Solution

(a) **Algorithm 2** `Frequent-Numbers(`$A, \ell, r$`)`

```
if ℓ = r then return {A[ℓ]}                                 ▷ base case
C ← Frequent-Numbers(A, ℓ, ⌈(ℓ+r)/2⌉−1)    ▷ candidates are the frequent numbers of left ...
C ← C ∪ Frequent-Numbers(A, ⌈(ℓ+r)/2⌉, r)                   ▷ ... and right sub-array
for c ∈ C do
    count the number of occurrences of c in A[ℓ...r]
    if c occurs less than (r−ℓ)/3 times in A[ℓ...r] then C ← C \ {c}
return C
```

A call of `Frequent-Numbers(`$A, 0, n-1$`)` solves the problem.

(b) We split the given array $A$ into two parts of (almost) equal size. A frequent number of that array must be a frequent number in the left half or the right half (or both). Thus it suffices to first find the frequent numbers of the left sub-array and then the ones of the right (if they exist). We do this by applying the procedure recursively and then check whether some of these are also frequent in $A$, by simply counting the number of occurrences of the candidates.

In each iteration we make recursive calls on two sub-arrays of half the size of $A$. Afterwards we count elements in the current array which takes at most $6n$ read operations if $n$ is the current size of the array (note that the set $C$ has size at most 6). We obtain the recurrence relation $T(n) \leq 2T(\lceil \frac{n}{2} \rceil) + 6n$ with base case $T(1) = 1$ (one read operation), which solves to $T(n) \in \mathrm{O}(n \log n)$ with the Master Theorem.