Datenstrukturen & Algorithmen  
Solution of Sheet 4    FS 13

Solution 4.1  
Statements about $O$-notation.

a) The statement is false, which we prove by contradiction. Assume that $\log^2(n) \in \Omega(\sqrt{n})$. Then there exist constants $c \in \mathbb{R}^+$ and $n_0 \in \mathbb{N}$, such that for every $n \geq n_0$ the following holds.

$$\log^2(n) \geq c\sqrt{n} \iff \frac{\log^2(n)}{\sqrt{n}} \geq c$$  \hspace{1cm} (1)

On the other hand, we have

$$\lim_{n \to \infty} \frac{\log^2(n)}{\sqrt{n}} \overset{(*)}{=} \lim_{n \to \infty} \frac{2\log(n) \cdot \frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n \to \infty} \frac{4\log(n)}{\sqrt{n}} \overset{(*)}{=} \lim_{n \to \infty} \frac{4 \cdot \frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n \to \infty} \frac{8}{\sqrt{n}} = 0$$  \hspace{1cm} (2)

(where $(*$) is obtained using L’Hôpital’s rule). Thus, $\frac{\log^2(n)}{\sqrt{n}}$ converges to 0 and there is no constant $c > 0$ that is a lower bound for the sequence.

b) The statement is correct. We use again the rule of L’Hospital and observe

$$\lim_{n \to \infty} \frac{n^2 + n}{1.1^n} = \lim_{n \to \infty} \frac{2n + 1}{(\ln 1.1)1.1^n} = \lim_{n \to \infty} \frac{2}{(\ln 1.1)^21.1^n} = 0,$$  \hspace{1cm} (3)

so even $n^2 + n \in o(1.1^n)$ holds. In exercise 1.4a) it was shown that this especially implies $n^2 + n \in O(1.1^n)$.

c) This statement is also correct, since we have

$$n! = \prod_{i=1}^{n} i \leq \prod_{i=1}^{n} n = n^n,$$  \hspace{1cm} (4)

and for the constants $c = 1$ and $n_0 = 1$ we obtain that $n! \leq cn^n$ for every $n \geq n_0$.

d) This statement is false. For contradiction we assume that it was correct. Then in particular we obtain that $n \log n \in \Omega(n^2)$.

Thus, there exist constants $c \in \mathbb{R}^+$ and $n_0 \in \mathbb{N}$, such that for every $n \geq n_0$ the following holds.

$$n \log(n) \geq cn^2 \iff \frac{\log(n)}{n} \geq c$$  \hspace{1cm} (5)

However, using L’Hospital’s rule we also obtain

$$\lim_{n \to \infty} \frac{\log n}{n} = \lim_{n \to \infty} \frac{\frac{1}{n}}{1} = \lim_{n \to \infty} \frac{1}{n} = 0.$$  \hspace{1cm} (6)

Thus, $\frac{\log n}{n}$ converges to 0 and there is no constant $c > 0$ that is a lower bound for the sequence, which is a contradiction.
Solution 4.2  Questions about sorting algorithms.

a) A sequence $f_1, f_2, \ldots, f_n$ is called a Min-Heap, if $f_i \leq f_{2i}$ for every $i$ with $2i \leq n$ and $f_i \leq f_{2i+1}$ for every $i$ with $2i + 1 \leq n$. For the sorted sequence, we have $f_i \leq f_j$ for every $j \geq i$, therefore the sorted sequence is indeed a Min-Heap.

b) A sequence $f_1, f_2, \ldots, f_n$ is called a Max-Heap if $f_i \geq f_{2i}$ for every $i$ with $2i \leq n$ and $f_i \geq f_{2i+1}$ for every $i$ with $2i + 1 \leq n$. Then, the smallest element will be in a position $i$ where $2i > n$, i.e. it is located in the second half of the array.

c) Even naive implementations of insertion sort and bubble sort are already stable. Merge sort can easily be made stable, if we remember to take the leftmost element when a tie is encountered while merging. There is no easy way to make selection sort, quicksort and heapsort stable.

d) Selection sort, insertion sort, bubble sort and heapsort work directly on the array to be sorted, and are therefore in-situ. Quicksort requires between $\Omega(\log n)$ and $O(\log n)$ additional space for storing the recursive function calls. This additional space is not used for elements in the sequence, therefore Quicksort is also in-situ. For merge sort, parts of the array must be copied for the merging. There are (complicated) methods to perform the merging in-situ, but no such methods can be implemented as simple modifications of the standard algorithm.

e) Even though Quicksort has a worst-case running time of $\Theta(n^2)$, with a random selection of the pivot the probability to get the quadratic running time is extremely small. The expected running time of quicksort is $O(n \log n)$. In addition, we know from the previous part of this exercise that quicksort works in-situ (in contrast to merge sort). Furthermore, a much smaller constant is “hidden” in the expected running time of quicksort than in the one of merge sort.

Solution 4.3  Heapsort.

a) In the picture below on the left, the array $[A,L,G,O,R,I,T,H,M,U,S]$ is visualized in a tree structure. However, this does not yet represent a max-heap, as the heap condition does not hold already for the root ($A$ is lexicographically smaller than $L$ and $G$).

The method presented in the lecture considers the vertices $R$, $O$, $G$, $L$ and $A$ in exactly this order, and at each step it restores the heap property of the heap below the considered vertex. This gives the heap shown in the picture below on the right.

```
        A
       / \  \
      L   G
     /\   /\ \\
    O  R  I  T
   /\  /\  /\  /\ \\
  H  M U S
```


b) For $i = 1, 2, \ldots, n - 2$ the following is executed. In $i$-th step, we swap $A[1]$ with $A[n + 1 - i]$,
and for $A[1..n-i]$ we restore the heap property. This results in the following execution of the algorithm. The double line indicates the end of current heap and the beginning of the already sorted sequence.

\[
\begin{align*}
&\text{USTORIGHMAL} \\
&TSLORIGHM\underline{A}U \\
&SRLOAI\underline{G}HMTU \\
&R\underline{O}LMAIGH\underline{S}TU \\
&OMLHAIGR\underline{S}TU \\
&MHLGAIOR\underline{S}TU \\
&LHIGAMOR\underline{S}TU \\
&IHAGL\underline{M}OR\underline{S}TU \\
&HGAILMOR\underline{S}TU \\
&GAHILMOR\underline{S}TU \\
&A\underline{G}HILMOR\underline{S}TU \\
\end{align*}
\]

**Solution 4.4**  
*Search trees.*

a) Inserting the keys in the specified order results in the following search tree:

![Search tree diagram](image)
In order to delete 9, we can replace its node either by its predecessor (figure on the left) or its successor (figure on the right).

b) Inserting the keys in the specified order results in the following AVL tree:

Again, there are two possible solutions that result from deleting 9. Replacing the node by its predecessor gives the solution on the left, replacing by the successor gives the solution on the right.