Exercise 1. (6+4+5 points)
This exercise is concerned with stacks and queues.

(a) Assume we have two stacks at our disposal, with the operations push and pop as specified in the abstract data type for stacks. Use them to implement the operations enqueue and dequeue of the abstract data type for queues.

Do it in such a way that the operation enqueue is in $O(1)$, and the operation dequeue is on average in $O(1)$.

(b) Consider the operations enqueue and dequeue from your answer to the previous exercise. Explain why the time complexity is as required there, that is: explain why your enqueue is in $O(1)$, and why your dequeue is on average in $O(1)$.

(c) Consider a stack with in addition to the usual operations push and pop also the following operation multipop:

```
Algorithm multipop(S, k):
    while not isEmpty(S) and k > 0 do
        pop()
        k := k - 1
```

Use amortized analysis (either aggregate analysis or the accounting method) to show that the amortized complexity of one operation is in $O(1)$.

Exercise 2. (6 points)
This exercise is concerned with singly linked lists.

(a) We consider singly linked lists, where for every node $v$ there is $v.element$ and $v.next$ with the suggested meaning. For the list $L$ there is $L.first$ giving the first node, and $L.last$ giving the last node.

Give pseudo-code for the algorithm insertFirst that takes as input a singly linked list $L$ and data $d$, and that updates $L$ by inserting a node with data $d$ as first node of the list.
Exercise 3. (4+4+5 points)
This exercise is concerned with max-heaps and heapsort.
(a) Apply (‘on the fly’) the algorithm for bottom-up max-heap construction to the following array: [5, 1, 2, 4, 3].
   Give your answer in the form of pictures, and indicate with arrows where swaps take place.
(b) Apply (‘on the fly’) heapsort to the array [5, 1, 2, 4, 3];
   start from the max-heap obtained in the previous exercise.
   You may use either the array-notation for the heaps, or pictures.
(c) Give the worst-case time complexity of heapsort in terms of \(O\); briefly motivate your answer.

Exercise 4. (5 points)
This exercise is concerned with AVL-trees.
(a) Construct an AVL-tree by inserting the numbers

1 5 4 3 2 6

one by one, starting from the empty tree. After each insertion, rebalance the tree if needed. Give your answer in pictures (with comments if needed).

Exercise 5. (5+6+5 points)
This exercise is concerned with binary trees and graphs.
(a) The preorder traversal of a binary tree \(B\) yields the following sequence of keys: 7, 6, 5, 4, 3, 2, 1. Give three different possibilities for \(B\).
(b) We consider binary trees implemented in a linked structure, where \(T.root\) gives the root of the tree \(T\). In a node \(v\), we have \(v.parent\) pointing to the parent, \(v.left\) pointing to the left child, and \(v.right\) pointing to the right child.
   Give pseudo-code of an algorithm \texttt{levelOrder} that takes as input a binary tree \(T\) and that performs a level order traversal of the input tree.
   You may use an operation \texttt{visit} that visits a node, and a queue with operations as specified in its abstract data type.
(c) Explain (informally) how the algorithm for level order traversal can be generalized to an algorithm for breadth-first search (BFS) that takes as input a directed graph and a start node.
Exercise 6. (5+4+5 points)
This exercise is concerned with dynamic programming.
For (a) and (b) we consider the algorithm for \texttt{knapsack01}:

\textbf{Algorithm knapsack01}(S, W):

\begin{verbatim}
new B[0...n,0...W]
for w := 0 to W do
    B[0, w] := 0
for k := 1 to n do
    B[k, 0] := 0
for w := 1 to W do
    if \(w_k \leq w\) then
        \(B[k, w] := \max(B[k-1, w], B[k-1, w-w_k] + b_k)\)
    else
        \(B[k, w] := B[k-1, w]\)
\end{verbatim}

(a) Apply the algorithm to maximal weight \(W = 5\) and the following set \(S\) with items with benefit and weight:

\begin{center}
\begin{tabular}{c|cc}
  & \(b\) & \(w\) \\
\hline
\(s_1\) & 2 & 1 \\
\(s_2\) & 5 & 2 \\
\(s_3\) & 4 & 3 \\
\end{tabular}
\end{center}

Give your answer in the form of a table

\begin{center}
\begin{tabular}{c|cc}
\(k\) \& \(w\) & \ldots \\
\hline
\vdots
\end{tabular}
\end{center}

(b) Give the worst-case time complexity of the algorithm for \texttt{knapsack01} in terms of \(O\); briefly motivate your answer.

(c) The Fibonacci numbers are given by the following recurrence: \(F_0 = 0\), \(F_1 = 1\), en \(F_i = F_{i-1} + F_{i-2}\) voor \(i \geq 2\). Give a dynamic programmic algorithm \texttt{fib} for computing the Fibonacci numbers such that computing \texttt{fib}(n) is in \(O(n)\).
Exercise 7. (5+5+5 points)
This exercise is concerned with greedy choice.

(a) Consider the shortest path problem: given a weighted directed graph and vertices $s$ and $d$ in that graph, determine a shortest path from $s$ to $d$.

Give an example showing that the greedy choice for an edge with minimal weight does not necessarily yield an optimal solution.

(b) Consider the activity selection problem: given a set of activities all with start time and finish time, select a maximum-size subset of mutually compatible activities.

Give an example showing that the greedy choice for an activity with minimal start time does not necessarily yield an optimal solution.

(c) Give step by step (in pictures) the construction of the tree for a Hufmann encoding of the following characters with frequency table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Exercise 8. (6 points)
This exercise is concerned with string matching.

(a) Apply (‘on the fly’) the Boyer–Moore pattern matching algorithm to the following pattern $P$ and text $T$. Give and number all steps. You do not have to give the function last explicitly:

$T = a d a c a a d c b a d a b a c$
$P = a b a c$

The mark for the exam is (the total number of points plus 10) divided by 10.