1. Argue that the worst-case time complexity of counting sort is $O(n + k)$, with $n$ the length of the input and $k$ indicating the range.

Illustrate the working of counting sort on input [0, 3, 2, 0, 3, 5] (see the book for example pictures).

2. Apply bucket sort on the following input using an array of length 10:

   $(7, a), (1, b), (1, c), (6, d), (3, e), (2, f), (8, g), (5, h), (7, i), (4, a)$

Here we don’t need the ‘floor’ part, and we use the usual alphabetic order on letters.

3. Apply radix sort to the following sequence:

   $22, 57, 7, 66, 100, 36, 44, 15, 43, 10, 3, 9, 13, 4, 29, 25$

Show step by step how the sequence is sorted.

4. Argue the correctness of radix sort.

5. Show how to sort $n$ integers in the range 0 to $n^3 - 1$ in $O(n)$.

6. Explain why the worst-case running time for bucket sort is in $\Theta(n^2)$.

   How can the algorithm be adapted to one with worst-case running time in $O(n \log n)$, while keeping linear running time in the average case?

7. Perform the following sequence of operations on an initially empty stack.

   Give the contents of the stack after every operation.

   push(S, 5), push(S, 3), pop(S), push(S, 2), push(S, 8),
   pop(S), pop(S), push(S, 9), push(S, 1)

8. Perform the following sequence of operations on an initially empty queue $Q$.

   Give the contents of the queue after every operation.

   enqueue(Q, 5), enqueue(Q, 3), dequeue(Q), enqueue(Q, 2),
   enqueue(Q, 8), dequeue(Q), dequeue(Q), enqueue(Q, 9),
   enqueue(Q, 1)

9. Provide pseudo-code descriptions for the operations size(Q), isEmpty(Q), and head(Q) of the abstract data type for queues. We assume $n = Q.length$ to be the size of the array.
10. Assume we have two stacks at our disposal, with the operations as specified in the abstract data type for stacks. Use them to implement the operations enque and deque of the abstract data type for queues.

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

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

Analyze the time complexity of push and pop in terms of $O$.

12. Explain how to implement two stacks in one array $A[1 \ldots n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is $n$. The operations push and pop should run in $O(1)$ time.

13. Adapt the procedures enque and deque from the book such that they also deal with overflow and underflow.

14. The Fibonacci numbers are given by the following recurrence: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ voor $i \geq 2$.

Give pseudo-code for the algorithm that naively computes the Fibonacci numbers. Give an analysis of the running time of your algorithm using both a recursion tree and a recurrence equation.

15. Read about the adaptation of quickSort to search for the kth smallest element in an array, for instance at [the wiki-page for quickSelect](#).

Write your own version of the pseudo-code for quickSelect that takes as input an array $A$, indices $l$ and $r$, and returns a smallest element of $A[l \ldots r]$. 
