1. Apply insertion sort, selection sort, bubble sort, mergesort and quicksort to the following sequence:

\[ 22, 15, 36, 44, 10, 3, 9, 13, 29, 25 \]

In each case, show step by step how the sequence is sorted.

2. Give a worst-case and a best-case input sequence of length \( n \) for insertion sort and selection sort, for any \( n \geq 1 \) (with regard to the pseudocode of these sorting algorithms on the slides).

Argue for both algorithms that sorting your worst-case sequence takes \( O(n^2) \) time. How long does it take to sort your best-case sequence?

3. Consider the following input to the 0-1 knapsack problem: items \((3, 1)\) and \((5, 2)\) and \((6, 3)\), where the 1st and 2nd index are the value and weight, respectively. The total weight mustn’t exceed 5.

Show how the dynamic programming approach to the 0-1 knapsack problem computes a solution.

4. Adapt quicksort so that it sorts sequences in non-increasing (instead of non-decreasing) order.

5. Argue that any sorting algorithm based on pairwise comparison of numbers in the sequence has a worst-case time complexity of at least \( O(n \log n) \).

6. Suppose that it is known beforehand that the numbers in a sequence of length \( n \) are from a fixed range \( \{1, \ldots, m\} \). Counting sort simply counts the number of occurrences of each \( i \in \{1, \ldots, m\} \) in the sequence. These counts are maintained in a separate array of length \( m \). At the end the sorted list can be simply built using these counts.

(a) Argue that the worst-case time complexity of counting sort is \( O(n + m) \).

(b) How is it possible that counting sort does not adhere to the lower bound determined in the previous exercise?