1. Show the Traveling Salesman Problem does not have the greedy choice property.

The Traveling Salesman Problem has as input an undirected weighted graph with a starting node. The goal is to find a path of minimal weight that visits every node exactly once.

Solution:

The following example shows that the greedy method does not work for the Travelling Salesman Problem: a greedy choice for an edge with minimal weight means that we as a start travel via the edge with weight 1. Then we can no longer avoid the edge with weight 100.

![Graph Example](image)

2. This question is concerned with rod cutting.

Define the *density* of a rod of length $i$ as $\frac{p_i}{i}$, that is, the value per decimeter.

A greedy algorithm for rod cutting starts by cutting a rod of length $i$ with a highest density, and then continues in this way. Give an example showing that this approach does not necessarily lead to an optimal solution.

Solution:

Assume the following list of prices:

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

| Density $\frac{p_i}{i}$ | 1 | 5/2 | 7/3 |

A greedy choice for the highest density yields in case of a rod of length 3: first a rod of length 2 with value 5, then a rod of length 1 with value 1, so in total value 6.
The best choice, however, is a rod of length 3 with value 7.
Hence a greedy choice for a rod with highest density does not yield an optimal solution.

3. This exercise is concerned with fractional knapsack.

Apply the algorithm for fractional knapsack to the following set of items with benefit-weight values, using in addition total weight $W = 18$:

$\{(12, 4), (10, 6), (8, 5), (11, 7), (14, 3), (7, 1), (9, 6)\}$

Solution:
We consider the set $S = \{s_1, \ldots, s_7\}$ with the following benefit, weight, and quotient benefit per weight:

<table>
<thead>
<tr>
<th>$s$</th>
<th>$b$</th>
<th>$w$</th>
<th>$\frac{b}{w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>12</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$s_2$</td>
<td>10</td>
<td>6</td>
<td>$\frac{5}{3}$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>8</td>
<td>5</td>
<td>$\frac{4}{3}$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>11</td>
<td>7</td>
<td>$\frac{14}{7}$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>14</td>
<td>3</td>
<td>$\frac{4}{1}$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$s_7$</td>
<td>9</td>
<td>6</td>
<td>$\frac{3}{2}$</td>
</tr>
</tbody>
</table>

The greedy order for selection is: $s_6, s_5, s_1, s_2, s_3, s_4, s_7$. Choosing according to this greedy order without exceeding the given maximal total weight $W = 18$ yields the following total benefit:

$7 \cdot \frac{1}{1} + 14 \cdot \frac{3}{3} + 12 \cdot \frac{4}{4} + 10 \cdot \frac{6}{6} + 8 \cdot \frac{4}{5} = 49 \frac{2}{5}$

4. This exercise is concerned with fractional knapsack.

Give a (small) example showing that a greedy choice for an item with the largest benefit does not necessarily yield an optimal solution.

Solution:
For the fractional knapsack problem with the set of items given as

<table>
<thead>
<tr>
<th>$s$</th>
<th>$b$</th>
<th>$w$</th>
<th>$\frac{b}{w}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$s_2$</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$s_3$</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
and maximal total weight $W = 3$ the greedy choice for an item with the largest benefit means choosing $s_1$ with benefit 3, whereas the optimal solution is $\{s_2, s_3\}$ with benefit 4.

5. This exercise is concerned with fractional knapsack.

Give a (small) example showing that a greedy choice for an item with the lowest weight does not necessarily yield an optimal solution.

Solution:

For the fractional knapsack problem with the set of items given as

\[
\begin{array}{ccc}
 s_1 & b & w \\
 s_2 & 1 & 1 & 1 \\
 s_3 & 4 & 2 & 2 \\
\end{array}
\]

and maximal total weight $W = 2$ the greedy choice for an item with lowest weight means choosing $s_1$ with benefit 1, whereas the optimal solution is $\{s_2\}$ with benefit 4.

6. This exercise is concerned with fractional knapsack.

Explain why the algorithm for fractional knapsack is in $O(n \log n)$, with $n$ the number of elements of the set $S$ from which we choose (parts of) items.

Solution:

We represent the set $S$ by a priority queue, where the key of an item is its benefit-weight ratio $\frac{b}{w}$, and a higher key means a higher priority. We assume an implementation of the priority queue using a max-heap. Then, removing an item with highest priority or adding an item is in $O(\log n)$ with $n$ the number of elements of $S$.

The initialization part of the algorithm in fact corresponds to adding every element of $S$ one by one to the priority queue. (This is a bit implicit in the algorithm as given on the slides.) This part of the algorithm has time complexity in $O(n \log n)$. The while-loop is carried out at most $n$ times. In each iteration an item is removed from the priority queue, and some updates in elementary time are done. So the time complexity of the while-loop is in $O(n \log n)$. This means that the algorithm is in $O(n \log n)$.

Remark: if we implement the priority queue using an unsorted (or sorted) list instead of by a heap, we get a time complexity of $O(n^2)$, because then removing from (adding to) the priority queue is in $O(n)$.

Another remark: the book mentions in exercise 16.2-6 that fractional knapsack can be solved in linear time. This is using a weighted median selection; a method that is beyond the scope of the course.
7. This exercise is concerned with the activity-selection problem.
   Apply the greedy algorithm for activity-selection to the following set of
   activities, with given start- and finish-time:

   \{(1, 2), (1, 3), (1, 4), (2, 5), (3, 7), (4, 9), (5, 6), (6, 8), (7, 9)\}

   Solution:
   We order the activities with their finish time in increasing order:

   \{(1, 2), (1, 3), (1, 4), (2, 5), (5, 6), (3, 7), (6, 8), (4, 9), (7, 9)\}

   We repeatedly take the greedy choice of an activity with earliest finish
   time and compatible with the selection so far. This yields the following
   list of subproblem: chosen activity:

   | (0, 9) | (1, 2) |
   | (2, 9) | (2, 5) |
   | (5, 9) | (5, 6) |
   | (6, 9) | (6, 8) |

8. This exercise is concerned with the activity-selection problem.
   Show that a greedy choice for an activity with the least duration and which
   is moreover compatible with the choice so far does not yield an optimal
   solution.
   Solution:
   A greedy choice for an activity with the least duration and which is mo-
   reover compatible with the choice for far yields for the following activity
   selection problem:

   \{(0, 3), (3, 6), (2, 4)\}

   the selection fo (2, 4). This is not the optimal solution (which is selection
   (0, 3) and (3, 6).

9. Consider the activity selection problem. Does a greedy choice for an acti-
   vity with the largest (or last) start-time (and which is moreover compatible
   with the activities selected so far) yield an optimal solution?
   If so, adapt the algorithm for activity-selection to this greedy choice.
   Solution:
The following is analogous to the proof on the slides.
Assume an activity selection problem $S$ consisting of $n$ activities, and a solution $A$. Let $a_k$ be the activity with the largest start time, so $s_k \geq s_i$ for all $i \in \{1, \ldots, n\}$. We show that a solution containing $a_k$ exists. If $a_k \in A$ then we are done. If $a_k \notin A$, then we consider an activity $a_p \in A$ with largest start time. Because $s_k \geq s_p$ the activity $a_k$ is compatible with all activities in $A - \{a_p\}$. The set $A - \{a_p\} \cup \{a_k\}$ has the same cardinality as $A$ and hence is also a solution for $S$. This is a solution containing $a_k$.

An adaptation of the algorithm. It takes as input an array $s$ of start times and an array $f$ of finish times. We assume the activities (on those two arrays) to be ordered on decreasing start time.

**Algorithm** activitySelector$(s, f)$:

1. $n := s.length$
2. $A := \{a_1\}$
3. $k := 1$
4. **for** $m = 2$ **to** $n$ **do**
   - **if** $f[m] \leq s[k]$ **then**
     - $A := A \cup \{a_m\}$
     - $k := m$

10. Given a set of $n$ costumers at a post office. Every costumer $i$ needs $t_i$ time for being served. Give a greedy algorithm that determines an order of the costumers that minimalizes the total amount of time spent at the post office by all costumers together. (We assume a costumer leaves once being served.)

   **Solution:**

   Example: $n = 3$ and $t_1 = 5$, $t_2 = 10$, $t_3 = 3$. There are $3! = 6$ possible orders. A possible order is 123 with total time $5+(5+10)+(5+10+3) = 38$. Another possibility is the order 321 with total time $3+(3+10)+(3+10+5) = 34$.

   We order the costumers according to non-decreasing time $t_i$. In the example: 321 with total time 29.

11. (Huffman)

   Give an example of a coding that is not prefix-free, but nevertheless non-ambiguous.

   **Solution:**

   $a = 0$ and $b = 01$.

12. (Huffman)

   Give the frequency-table and the Huffman-codingtree for the following string (do not count spaces):
how much wood would a woodchuck chuck

Solution:
The frequency table is as follows:

<table>
<thead>
<tr>
<th>h</th>
<th>o</th>
<th>w</th>
<th>m</th>
<th>u</th>
<th>c</th>
<th>d</th>
<th>l</th>
<th>a</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

A Huffman encoding is given by the following tree:

```
13
/  \
18  3
/ \ / \  
  7 d3 3 3
/ / / /     
  k 7 3 3
```

13. (Huffman)

Explain why the time complexity of Huffman’s algorithm is in \( O(n \log n) \).

Solution:

We represent the input set \( C \) as a priority queue. We assume that we implement the priority queue by a min-heap.

To start with, in the algorithm we build a min-heap from the elements of the input \( C \). This is with the bottom-up heap construction in \( O(n) \) with \( n \) the number of elements in \( C \).

The for-loop is iterated \( n - 1 \) times. In each iteration we remove (twice) from the priority queue, and we insert (once) in the priority queue. For the implementation using a min-heap, both the operation for removal and the one for insertion are in \( \log n \). Therefore the time complexity of the for-loop is in \( O(n \log n) \).

The initialization together with the for-loop is in \( O(n + n \log n) = O(n \log n) \).
14. (Huffman)

Adapt Huffman’s algorithm to deal with ternary code words using the symbols 0, 1, 2.

Solution:

Instead of joining two binary tree with minimal frequencies together to one binary tree, we joint three ternary trees with minimal frequencies together to one ternary tree. If we apply this idea to the example of a previous exercise (11) we indeed get an encoding, however not an optimal one.

We should take care to start with the right number of leaves, so $3 + 2k$ for certain $k$. If we do not have sufficiently many leaves, we should add a dummy leave with frequency 0.

We assume an input $C$ consisting of at least 3 objects. The idea of the algorithm:

```
Algorithm HuffmanCodeTernary(C):
    n := |C|
    Q := C
    if n even then
        new node n
        n.freq := 0
        insert(Q, n)
        n := n + 1
    for i = 1 to (n - 1)/2 do
        new node z
        z.left := x := removeMin(Q)
        z.middle := y := removeMin(Q)
        z.right := u := removeMin(Q)
        z.freq := x.freq + y.freq + u.freq
        insert(Q, z)
    return removeMin(Q)
```

15. Apply Dijkstra’s algorithm to the following graph (take for the direction from lower to higher in the alphabet), taking $A$ as source node. Give updates step by step by annotating the graph or by giving a table.
Solution:
The relevant data in a table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>{A}</td>
<td>0</td>
<td>13</td>
<td>6</td>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>{A, D}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>∞</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>{A, C, D}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>∞</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>{A, C, D, F}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>∞</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>{A, B, C, D, F}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>{A, B, C, D, E, F}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>{A, B, C, D, E, F, G}</td>
<td>0</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>18</td>
</tr>
</tbody>
</table>

Dijkstra’s algorithm:

**Algorithm** Dijkstra($G, w, s$):

  initialize($G, s$)
  $S := ∅$
  $Q := G.V$
  while $Q ≠ ∅$ do
    $u := extractMin(Q)$
    $S := S ∪ \{u\}$
    for each $v ∈ G.Adj[u]$ do
      relax($u, v, w$)