Exercises

1. (a) 56312
   (b) 12356

2. (a) We sort the tuples first and take the tastiest food greedily until the capacity Q is filled.

```
def greed(tuples, Q):
  # tuples is a list of (fi, ti, qi)
  tuples = sorted(tuples, key=lambda tup: tup[1], reverse=True)
  sumq = 0
  out = []
  idx = 0
  while sumq < Q:
    f, t, q = tuples[idx]
    if sumq+q <= Q:
      out.append((f, q))
      sumq += q
      idx += 1
    else:
      out.append((f, Q-sumq))
      sumq = Q
  return out
```

   (b) Consider `tuples` after sorting. We relabel `tuples` in our discussion, such that `tuples[0]` has index 0 and t_0 is the largest among t_i.
   Inductive hypothesis: At step t-1, there exists one optimal solution that includes the choices (f_0, q_0),... , (f_{t-1}, q_{t-1}).
   Inductive step: Case 1: Step t is not the final step. Suppose toward a contradiction that there exists no optimal solution that includes all of (f_0, q_0),... , (f_t, q_t). Then all the optimal solutions available for step t-1 assign to f_t a strictly smaller quantity than q_t. Since all f_i for which $t_i > t_t$ have been greedily chosen before step t, all subsequent choices have $t_i \leq t_t$. Therefore, if we increase the quantity of f_t to q_t, and reduce the quantity of any subsequent choice by the same amount, the capacity constraint still holds, while $\sum_i x_i T_i$ cannot decrease, leading to a contradiction. Therefore, there exists one optimal solution that includes the choices up to step t.
   Case 2: Step t is the final step. The last execution of the while loop goes into the `else` block. Since the algorithm picks the largest remaining t_i, the choices remain optimal.

Problems

1. (a) No. For n=40, the optimal solution is {10, 10, 10, 10}, but the algorithm gives {25, 10, 1, 1, 1, 1, 1}.

(b) Yes. Let the output of Algorithm 1 be $\{q_0, q_1, ..., q_s\}$, where q_i is the number of coin 2^i chosen. Let an optimal solution be $\{a_0, a_1, ..., a_s\}$. Algorithm 1 always finishes because the smallest denomination is 1. We can write

$$n = \sum_{i=0}^{s} q_i 2^i = \sum_{i=0}^{s} a_i 2^i$$

Lemma 1: If a solution is optimal, for all $i = 0, 1, ..., s-1$, a_i can only be 0 or 1.

Proof: Suppose toward a contradiction that an optimal solution has $a_i > 1$ for some $i < s$. Replacing two 2^i coins with one 2^{i+1} coin results in a better solution, which contradicts optimality.

Lemma 2: If a solution is optimal, $a_s = \lfloor n/2^s \rfloor$.

Proof: Case 1: If $n < 2^s$, then $a_s = \lfloor n/2^s \rfloor = 0$ which is correct. Case 2: $n \geq 2^s$. We eliminate the other two possibilities of trichotomy. Suppose toward a contradiction that $a_s > \lfloor n/2^s \rfloor$, we have $a_s > n/2^s$ since a_s is an integer, and therefore $n \geq a_s 2^s > n$, a contradiction. Suppose toward a contradiction that $a_s < \lfloor n/2^s \rfloor$. We have $a_s \leq \lfloor n/2^s \rfloor - 1$ since a_s is an integer. By Lemma 1, a_i is 0 or 1 for all $i = 0, 1, ..., s-1$, so we have the following contradiction:

$$\begin{aligned} n = \sum_{i=0}^{s} a_i 2^i &\leq \sum_{i=0}^{s-1} 2^i + a_s 2^s = 2^s - 1 + a_s 2^s \\ &\leq 2^s - 1 + (\lfloor n/2^s \rfloor - 1) 2^s \leq -1 + \lfloor n/2^s \rfloor 2^s < n \end{aligned}$$

Lemma 3: The output of Algorithm 1 satisfies Lemma 1 and 2.

Proof: Since p^* can only be 2^s when $n > 2^s$, the number of times that coin 2^s is appended to `coins` is $\lfloor n/2^s \rfloor$. I.e. $q_s = a_s$ for Lemma 2. In each subsequent loop of `while`, we can write the output of `max` as $p^* = 2^k < n$, $k < s$. We always have $n < 2 \cdot 2^k$, because `max` would output 2^{k+1} if otherwise. (Note part (a) doesn't have this property.) Therefore we have $n - p^* = n - 2^k < 2^k$. So for $i < s$, each coin 2^i can only be appended to `coins` at most once. I.e. Lemma 1 is satisfied.

Lemma 3 establishes that $q_s = a_s$. Next we prove that $q_i = a_i$ for $i = s-1, s-2, ..., 1, 0$ such that Algorithm 1 is correct. From the algorithm, we have

$$\begin{aligned} &q_s = \lfloor n/2^s \rfloor = a_s \\ &m_s = n \mod 2^s \\ &q_{s-1} = \lfloor m_s/2^{s-1} \rfloor \\ &m_{s-1} = m_s \mod 2^{s-1} \\ &... \\ &q_0 = \lfloor m_1/2^0 \rfloor \\ &m_0 = m_1 \mod 2^0 \end{aligned}$$

To prove $a_{s-1} = q_{s-1}$, we eliminate the possibilities that they differ. By Lemma 1, both can be either 0 or 1. Suppose toward a contradiction that $a_{s-1} = 1$ and $q_{s-1} = 0$. The former gives $n - a_s 2^s = n \mod 2^s = \sum_{i=0}^{s-1} a_i 2^i \geq 2^{s-1}$ but the latter gives $m_s = n \mod 2^s < 2^{s-1}$. Suppose toward another contradiction that $a_{s-1} = 0$ and $q_{s-1} = 1$. The former gives $n - a_s 2^s = n \mod 2^s = \sum_{i=0}^{s-2} a_i 2^i < 2^{s-1}$ but the latter gives $m_s = n \mod 2^s \geq 2^{s-1}$. Therefore we must have $a_{s-1} = q_{s-1}$.

Similar arguments can be applied to $i = s-2, s-3, ..., 0$. Therefore $q_i = a_i$ for all i. I.e. the algorithm is correct.

2. (a) Sort turkeys in non-increasing b. Give a message to a turkey at the end of its awake time, if it hasn't heard a message. Running time is sorting plus one pass so O(n log n).

```
def turkeys(turs):
  # turs is a list of turkeys [(a1, b1), (a2, b2),...]. 1-based numbering
  turs = sorted(turs, key=lambda tup: tup[1])
  t = [-float('inf')]  # message times. 1-based numbering
  for i in range(len(turs)):
    if turs[i][0] > t[-1]:
      t.append(turs[i][1])
  return t
```

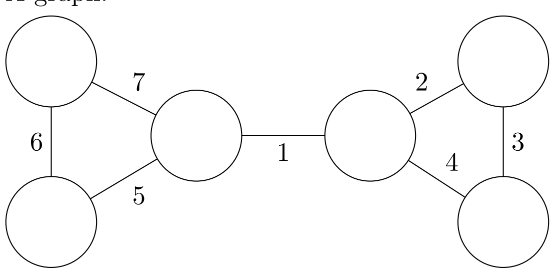
(b) Base case: Before the `for` loop, `t` is empty and therefore optimal for `turs[1:1]`$=\phi$.

Hypothesis: At the end of step i-1 of the `for` loop, `t` is optimal for `turs[1:i]`.

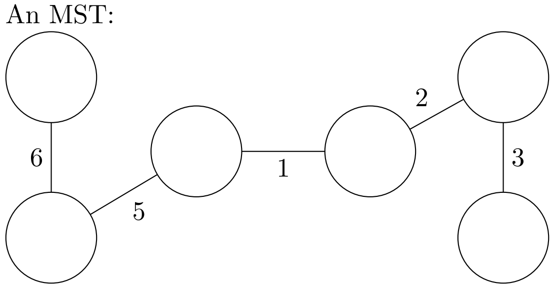
Induction: For step i, there are two cases. Case 1: If `turs[i][0]` $\leq$ `t[-1]`, we do not append anything to `t`. Because (1) the optimal length of `t` cannot decrease if we append a new turkey to `turs`, and (2) `turs[i]` is covered by `t[-1]`, `t` is optimal at step i. Case 2: When `turs[i][0]>t[-1]`, we add one message time to `t`. Suppose toward a contradiction that `t` is non-optimal at step i. Because the optimal length of `t` cannot decrease if we append a new turkey, the optimal length of `t` remains the same, whereas our algorithm adds one to it. Since `turs[i][0]>t[-1]`, the newest message is not heard by any turkeys in `turs[1:i]`. So one less messages can be used for `turs[1:i]`, which contradicts the hypothesis that the length of `t` is optimal at step i-1.

3. (a) Suppose toward a contradiction that T is an MST but not a minimum-maximum spanning tree (mMST), and say that T' is an mMST. Let (u, v) be the heaviest edge in T. Then deleting (u, v) from T will disconnect it into two trees, A and B, which form a cut on the set of vertices V. Since T' and T share the same V, A and B also form a cut on T'. Since T' is an mMST, all its edges have weights strictly less than (u, v), including the edge (w, x) that connects A and B in T'. Replacing (u, v) with (w, x) in T forms a spanning tree with a smaller total weight than T, leading to a contradiction that T is an MST.

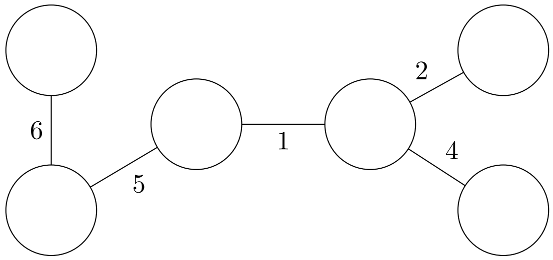
(b) A graph:



An MST:



A minimum-maximum spanning tree that is not an MST:



6
5
1
2
4