Exercise Sheet 1

Exercise 1 Collecting Stickers

Let n be the amount of different Twilight stickers and X be the random variable representing the the required number of bags to achieve all n stickers. The stickers are distributed uniformly in the bags.
We would like to find the expected value $E[X]$.

(a)

Let X_i be the random variable that represents the length of round i, where a round i ends, if we achiev a sticker, different from the $i+1$ we achieved thus far.

Claim 1. *In the context* $E[X] = \sum_{i=1}^{n} E[X_i]$ *holds.*

Proof 1.
For $i \neq j$ the events X_i, X_j are independend, because the events take place striklty after another. And a later round does not depenend on the length of any previous round or the sticker taken.
We than now, that after round the first i rounds we have i different stickers. After n rounds we then have all stickers. The amount of bags needed is then the sum of the length of all rounds.

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] \overset{Lin.}{=} \sum_{i=1}^{n} E[X_i]$$

□

(b)

Claim 2. *In the context* $E[X_i] = \frac{n}{n-i+1}$ *holds.*

Proof 2.
First we show, that $E[X_i]$ is geometric distributed, meaning
$Pr(X_i = k) = (1-p_i)^{k-1} \cdot p_i$ holds, where p_i is the possibility to get a new card.

The event X_i is described as the first time, we achieve a new card. If $X_i = k$ the first $k-1$ bags may not contain a new card, which is the complementary event with possibility $(1-p_i)$.

In terms of possibily $Pr(X_i = k) = (1-p_i)^{k-1} \cdot p_i$ holds.

Because $E[X_i]$ is geometric distributed we now conclude $E[X_i] = \frac{1}{p_i}$.

In the last step we now, that we already have $i-1$ stickers in the i-th round. So the possibility in a uniform distribution to get a new sticker is $p_i = \frac{n-i+1}{n}$.
By the previous formula the claim $E[X_i] = \frac{n}{n-i+1}$ follows.

$\square$

(c)

Claim 3. *For the task $E[X] = O(n\,log\,n)$ holds.*

Proof 3.

$$E[X] \overset{(a)}{=} \sum_{i=1}^{n} E[X_i] \overset{(b)}{=} \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{n-i+1}$$
$$\overset{(*)}{=} n \sum_{i=1}^{n} \frac{1}{i} = n \cdot O(log\,n) = O(n\,log\,n)$$

In the step $(*)$ we reordered the sum, because in the first one we summed $\frac{1}{n} + \frac{1}{n-1} + ... + 1$ and in the second one we sum $1 + \frac{1}{2} + ... + \frac{1}{n}$. We can simply do this, because the sum is well defined (we only sum a finite amount real numbers).

$\square$

Exercise 2

In the knapsack problem, we are given n items. Each item has a *weight* g_i and a *value* w_i. Furthermore , we have a maximum weight G. All inputs are positive integers.

We would like to find a set $I \subseteq \{1, ..., n\}$ of items, such that the total value $\sum_{i \in I} w_i$ is maximum, subject to the constraint that the total weight is at most G, meaning $\sum_{i \in I} g_i \leq G$.

(a)

Define an appropriate decision version for the knapsack problem and show that it is NP-complete.

Solution:
We choose the canonical extension of an optimisation problem to a decision problem. Let the defined variables be as the the optimisation problem. Than take an additional positive Integer K.

The decision question is now, does there exist a set $I \subseteq \{1, ..., n\}$, such that $\sum_{i \in I} w_i \geq K$ and $\sum_{i \in I} g_i \leq G$.
Let $KNAP$ be this problem.

Claim 4. *The decision problem for the knapsack problem is NP-complete.*

Proof 4.

i) $KNAP \in NP$.
Let $I \subset [n]$ be the optimal solution. Obviously $|I| < n^c$ for some constant $c > 0$, because I is itsef part of the input.
Now construct TMV as follows.
Compute $\sum\limits_{i \in I} w_i = w'$ and
$\sum\limits_{i \in I} g_i = g'$.
Accept if and only if $w' \geq K$ and $g' \leq G$.

Both sums can be computed in polynomial time and the final comparison can be computed in logarithmic time in the length of the values. So V accepts (w, I) in $T(n) \in O(n^c)$ for some constant $c > 0$.

ii) SUBSET-SUM $\prec_p$ KNAP.
To reduce subset-sum to knapsack we choose the following transformation. Let S be the set of positive integers for subset-sum and T a value. The question for knapsack is now, does there exist a subset $O \subseteq S$ such that $\sum\limits_{i \in O} i = T$.

The reduction now sets $w_i = g_i = i$ for all $i \in S$ and $G = K = T$ for the goal values. One can easily see, that the reduction can be done in linear time because we only copy values, thus it can be computed in polynomial time.

Knapsack run on this input will now hold the following. If there exists a set I that satisfies the constraints $\sum\limits_{i \in I} w_i = \underset{i \in I}{i} \geq K$ and $\sum\limits_{i \in I} g_i = \sum\limits_{i \in I} i \leq K$. Thus if we set $O = I$ $\sum\limits_{i \in O} i = T$ holds.
If no solution existed in the original problem, than we cannot find a solution in the reduction, because both sums will sum up to exactly the same. So one of the constraints cannot be true.

(b)

Let $W := \sum\limits_{i=1}^{n} w_i$. Show that the knapsack problem can be solved in $O(nW)$ time. Why does this not contradict *(a)*?

Solution:
We proof this by giving a dynamic program, we used in approximation algorithms. We construct an Array A of sets of tupel. A tupel $(t, w) \in A(j)$ means. There exists a subset $I \subseteq \{1, .., j\}$, such that $\sum\limits_{i \in I} w_i = w$ and $\sum\limits_{i \in I} g_i = t$.

Furthermore we introduce a *domination* relation. A tupel (t, w) dominates another tupel (t', w'), iff $t \leq t'$ and $w \geq w'$. (Means we can get a bigger value with less weight). Observe, that we can have at most $G+1$ and $W+1$ tupels in $A(j)$ for any $j \leq n$, because otherwise there is a tupel $(t', w') \in A(j)$ that is dominated by another tupel. Next we assume, that we can put any item in the bag itself. If not, we can throw them away.
In each iteration we make a constant number of steps for each element of the previous set of tupels. We showed, that in any set $A(j)$ we can bound the number of tupels by $\min\{G + 1, W + 1\}$. We iterate in the outer loop for any element in our bag.

```
A(1) ← {(0,0),($g_1$,$w_1$)}
for j ← 2 to n do
    A(j) ← A(j-1)
    for each (t,w) ∈ A(j-1) do
        if t + $g_j$ ≤ G then
            A(j) ← A(j) ∪ {(t + $g_j$, w + $w_j$)}
    Remove dominated pairs
return $\max_{(t,w)\in A(n)}$ w
```

Therefore we can write, that the runtime of our algorithm lies in $O(nW)$. (If G is smaller than W our estimation is to big, but still holds).

This solution does not contradict with (a) because W is epxonantial in the inpu size (which is $log\, W$). The algorithm is pseudo-polynomial.

(c)

For the $(1-\varepsilon)$ - approximation we also use a trick learned in approximation algorithms, based on the algorithm of (b).

The trick is to round the values of w_i, so we do not have to look at each single value.

We assign new values by the following scheme

```
M ← $\max_{i\in I} w_i$
$\mu$ ← $\varepsilon\frac{M}{n}$
w_i'␣← $\lfloor w_i/\mu \rfloor$ for all $i \in S$
```

and run the algorithm from (b) on it. With the modified Values we now get

$$W' = \sum_{i=1}^{n} w'_i = \sum_{i=1}^{n} \left\lfloor \frac{w_i}{\varepsilon M/n} \right\rfloor = O(\frac{n^2}{\varepsilon}).$$

Because now W' is polynomial in n and ε it is strictly smaller than B which is exponantial in the input size. So the runtime of our algorithm is $O(n^3 \cdot \frac{1}{\varepsilon})$, which is in $poly(n, \frac{1}{\varepsilon})$.

Leaves us to proof, that the algorithm is a $(1-\varepsilon)$ - approximation. Let I be the set which gives us the optimal solution on W and O the set, that gives use the optimal solution on W' the rounded values.

First observe, that $M \leq OPT$, because we can always can take the most valueable item. The rounding of the values gives us the estimation $\mu w'_i \leq w_i \leq \mu(w'_i + 1)$ wich leads us to $\mu w'_i \geq w_i - \mu$.

$$\begin{aligned}
\sum_{i\in O} w_i &\geq \mu \sum_{i\in O} w'_i \\
&\geq \mu \sum_{i\in I} w'_i \\
&\geq (\sum_{i\in I} w_i) - |I|\mu \\
&\geq (\sum_{i\in I} w_i) - n\mu \\
&= (\sum_{i\in I} w_i) - \varepsilon M \\
&\geq OPT - \varepsilon OPT = (1-\varepsilon)OPT
\end{aligned}$$

We can conclude, that our algorithm is an FPTAS for the knapsack problem.

Exercise 3

True or False? For every $\varepsilon > 0$, there exists an NP - complete problem that can be solved deterministicly in $O(2^{n^\varepsilon})$ steps. Explain your answer.
Solution:

We proof this part by construction a language for each $\varepsilon > 0$, that is $NP-complete$ and can be solved in the given time.

Let $PAD-SAT_\varepsilon = \{(\Psi, 1^{|\Psi|^{\lceil \frac{k}{\varepsilon} \rceil}}) \mid \Psi \text{satisfyable in 3 cnf}\}$ where $k > 1$ is a arbitrary but fixed konstant.

Claim 5. *$PAD - SAT_\varepsilon$ is NP-complete.*

Proof 5.
This part is straight forward.
Given an variable assignment we can check, whether the formula Ψ is satisfied in linear time, by simply iteration over the clauses. Because the number of variables has to be strictly smaller than the formulas the assignment is polynomial bounded in the inputsize.

We can next reduce $PAD-SAT_\varepsilon$ to $3-SAT$. The reduction copys Ψ and throws away the rest. If the TM for $3-SAT$ accepts, the word is in the language, because Ψ is satisfiable. If it rejects, Ψ is not satisfiable, thus the word can't be in $PAD-SAT_\varepsilon$.

Claim 6. *There exists a DTM M such that $T_M(n) \in DTIME(2^{n^\varepsilon})$.*

Proof 5.

From the definition we can conclude, that $|\Psi| \leq n^{\frac{\varepsilon}{k}}$ holds, because the rest of the length of the word is filled with ones.

Ψ is a boolean formula and this formula contains each occurence of its variables. Therefore there can't be no more variables, than the length of Ψ.
So we can compute all Assignments, which are less than $2^{n^{\frac{\varepsilon}{k}}}$. As said before we can iterate over all clauses and literals and check whether there occures at least one true in each clause. Therefore we have linear costs in the length of Ψ.

We now construct a DTM M.
Given a word $(\Psi, 1^{|\Psi|^{\lceil \frac{k}{\varepsilon} \rceil}})$, we check first whether the number of ones is right in the end. This takes us at most n time. Next we test all assignments, if they satisfy the formula.

Testing takes $n^{\frac{\varepsilon}{k}}$ time and we have $2^{n^{\frac{\varepsilon}{k}}}$ possibility, so the runtime of M is $T_M(n) = n^{\frac{\varepsilon}{k}} 2^{n^{\frac{\varepsilon}{k}}}$. We ignore the linear Faktor, because we know that we can choose a faktor so that this function is strictly greate after a given n_0.

Now we have to show, that $T_M(n) \in O(2^{n^\varepsilon})$. We do this by checking the limiting value

$$\lim_{n\to\infty} \frac{T_M(n)}{2^{n^\varepsilon}} = \lim_{n\to\infty} n^{\frac{\varepsilon}{k}} 2^{n^{\frac{\varepsilon}{k}} - n^\varepsilon} = 0$$

Because $n^{\frac{\varepsilon}{k}} < n^\varepsilon$ holds for $k > 1$ and than we have a exponential function with a negative exponent and a polynomial faktor. By knowledge of Analysis I this sequence converges to 0.

□