Exercise Sheet 1

Exercise 1

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. Each bag contains a sticker that is chosen uniformly at random.
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 achieve a sticker, different from the $i+1$ we achieved thus far.

Claim 1. $E[X] = \sum_{i=1}^{n} E[X_i]$.

Proof 1.
By construction $X = \sum_{i=1}^{n} X_i$ holds, because in each round we obtain a new sticker and sum up all bags need in each round.
By linearity of the expected value it follows immediately that

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

holds. □

(b)

Claim 2. $E[X_i] = \frac{n}{n-i+1}$.

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

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

□

(c)

Claim 3. $E[X] = O(n \log n)$.

Proof 3.

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

$$= n \sum_{i=1}^{n} \frac{1}{i} = n \cdot O(\log\ n) = O(n\ \log\ n)$$

□

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 optimization problem to a decision problem. Let the defined variables be as the the optimization 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 itself part of the input.
The Verifier V as follows.
Compute $\sum_{i \in I} w_i = w'$ and $\sum_{i \in I} g_i = g'$.
Accept if and only if $w' \geq K$ and $g' \leq G$.

Both sums can be computed in polynomial time and the comparison can be computed in logarithmic time in the length of the values. 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 $\sum_{i \in O} i = T$.

The reduction function f sets $w_i := g_i := i$ for all $i \in S$ and $G := K := T$. One can easily see, that the reduction can be done in linear time because we only copy values.

$w \in$ SUBSET-SUM $\Rightarrow f(w) \in$ KNAP.
There exists a subset $O \subset S$ such that $\sum_{i \in O} i = T$. After the application of f we know that $O \subset I$ and $\underset{i \in O_i}{w} = K$ and $\underset{i \in O_i}{g} = G$. Therefore O is a solution for KNAP on this input.

$f(w) \in$ KNAP $\Rightarrow w \in$ SUBSET-SUM.
There exists a subset $O \subset I$ such that $\sum_{i \in O} g_i \leq G$ and $\sum_{i \in O} w_i \geq K$. From the reduction f we know that $\sum_{i \in O} g_i = \sum_{i \in O} w_i = \sum_{i \in O} i$ holds.
$\Rightarrow \sum_{i \in O} i \leq G \wedge \sum_{i \in O} i \geq K$.
Again from f we know that $G = K = T$ holds.
$\Rightarrow \sum_{i \in O} i = T$.

□

(b)

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

Solution:

To proof this, we show, that a dynamic program solves the knapsack problem and has the runtime of $O(nW)$.
Let $g[n, W]$ be a two dimensional array, that stores the in $g[i, w]$ the minimal weight, for the first i objects, with an value exactly w.
We can now give the recursive definition of to fill in w. Let the array be initialized with ∞.

$$\begin{array}{lcll} g[0,w] & := & 0 & \forall 0 \leq w \leq W \\ g[i,0] & := & 0 & \forall 1 \leq i \leq n \\ g[i,w] & := & g[i-1,w] & \text{if } w - w_i \leq 0 \\ g[i,w] & := & \min\{g[i-1,w], g[i-1,w-w_i] + g_i\} & \end{array}$$

After filling the array we can retrieve the maximal value by computing
$\underset{0 \leq w \leq W}{\text{argmax}}\{g[n,w] \mid g[n,w] \leq G\}$.

This algorithm can be easily implemented through a double for loop, comparing both cases. The result can be computed in a single for loop.
The algorithm computes the optimal solution, because it computes every possible solution and takes the greatest value.
The computation takes $O(nW)$ time, because the array has $n \times W$ cells and in each cell

we have an constant computation time $O(1)$. To find the maximal value we iterate over W cells and compute the maximum which can be done in time $O(W)$.
Therefore $T(n) = O(nW)$ holds.

This does not contradict with *(a)* because W can be exponential in the size of the input. The runtime of the algorithm is pseudo polynomial.

(c)

For the $(1-\varepsilon)$ - approximation we use an algorithm based on the algorithm of (b).
The trick is to round the values of w_i such that we do not have to look at each possible value of w.

We assign new values by the following scheme

```
M ← max_{i∈I} w_i
μ ← ε M/n
w_i' ␣← ⌊w_i/μ⌋ for all i ∈ S
```

and run the algorithm from (b) on it. With the modified values we can skip the array in μ steps.

$$\frac{W}{\mu} = \frac{n}{\varepsilon \cdot M}\sum_{i=1}^{n} w_i \overset{M \geq w_i}{\leq} \frac{n}{\varepsilon}\sum_{i=1}^{n} 1 = O(\frac{n^2}{\varepsilon})$$

The new grid has size $n \times O(\frac{n^2}{\varepsilon})$ therefore the runtime of the algorithm on the modified values 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 valuable item. The rounding of the values gives us the estimation $\mu w_i' \leq w_i \leq \mu(w_i' + 1)$ which leads us to $\mu w_i' \geq w_i - \mu$.

$$\begin{aligned}
\sum_{i \in O} w_i &\overset{\text{Def. } w_i'}{\geq} \mu \sum_{i \in O} w_i' \\
&\overset{O \text{ opt. on } w_i'}{\geq} \mu \sum_{i \in I} w_i' \\
&\geq (\sum_{i \in I} w_i) - |I|\mu \\
&\overset{|I|<n}{\geq} (\sum_{i \in I} w_i) - n\mu \\
&\overset{\text{Def. } \mu}{=} (\sum_{i \in I} w_i) - \varepsilon M \\
&\overset{I \text{ opt. on } w_i}{\geq} OPT - \varepsilon OPT = (1-\varepsilon)OPT
\end{aligned}$$

We can conclude that the 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 deterministically 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$_\varepsilon = \{(\Psi, 1^{|\Psi|^{\lceil \frac{2}{\varepsilon} \rceil}}) \mid \Psi \text{ satisfiable in 3 cnf}\}$

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

Proof 5.
PAD-SAT$_\varepsilon \in$ NP.
Given an variable assignment we can check whether the formula Ψ is satisfied in polynomial time. The number of variables has to be strictly smaller than the length of the formula. The assignment is polynomial bounded in the input size hence the witness is polynomial in the size of the input.

3-SAT $\succ_p$ PAD-SAT$_\varepsilon$.
The reduction function copies the formula Ψ, computes $|\Psi|^{\lceil \frac{2}{\varepsilon} \rceil}$ and appends that many ones to Ψ. The computation can be done in polynomial time. The reduction is valid because if Ψ is satisfiable it remains satisfiable after the reduction and vis-versa.

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

Proof 6.
We construct a DTM M that checks the satisfiability by brute-force. Given a word $w = (\Psi, 1^{|\Psi|^{\lceil \frac{2}{k} \rceil}})$ with $n = |w|$.
It holds that $|\Psi| \leq n^{\frac{\varepsilon}{2}}$. Otherwise $|w| = |\Psi| + |\Psi|^{\lceil \frac{2}{k} \rceil} > n^{\frac{\varepsilon}{2}} + (n^{\frac{\varepsilon}{2}})^{\frac{2}{k}} > n$ holds which is a contradiction.

$\Rightarrow$ There exists at most $n^{\frac{\varepsilon}{2}}$ possible assignments for Ψ. We can compute the result for a given assignment in n^c time for some $c > 0$.
Therefore $T_M(n) \leq n^c \cdot 2^{n^{\frac{\varepsilon}{2}}}$. At last we prove that $T_M(n) \in O(2^{n^\varepsilon})$.

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

By convergence criterion we know that there exist the DTM M that computes a NP complete problem in $O(2^{n^\varepsilon})$ time.

□