Exercise 1

Exercise 1.a

Since all the deliveries takes one hour, the objective to minimize the time until the last elf has finished can be seen as minimize the maximum load (in terms of deliveries) that an elf will be tasked to do. So, the problem can be formulated in the following way:

ILP

$$\begin{aligned} \text{Minimize} \quad & T \\ \text{subject to} \quad & \sum_{e \in E} x_{eh} = 1, \qquad \forall h \in H \\ & \sum_{h \in H_e} x_{eh} \leq T, \qquad \forall e \in E \\ & x_{eh} \in \{0,1\}, \quad \forall e \in E,\ \forall h \in H \end{aligned}$$

LP

$$\begin{aligned} \text{Minimize} \quad & T \\ \text{subject to} \quad & \sum_{e \in E} x_{eh} = 1, \qquad \forall h \in H \\ & \sum_{h \in H_e} x_{eh} \leq T, \qquad \forall e \in E \\ & 0 \leq x_{eh} \leq 1, \quad \forall e \in E,\ \forall h \in H \end{aligned}$$

Where, for each pair (e, h), the variable x_{eh} determines whether the elf e is chosen to deliver in the house h or not. The variable T denotes the maximum load (the maximum finishing time). The first constraint tells that each house is assigned to only one elf, the second constraint tells that each elf e can be assigned to an house h only if $h \in H_e$ and that the number of houses assigned to an elf does not exceed the maximum load. The LP problem is obtained by replacing the constraint $x_{eh} \in \{0,1\}$ with $0 \leq x_{eh} \leq 1$, and this is the LP-relaxation of the ILP.

Exercise 1.b

Now, let OPT be the finishing time of the optimal LP solution. The following randomized algorithm gives a feasible integer solution $\tilde{x}$.

- Solve the LP and let the fractional solution be x^* of load OPT.
- For each house $h \in H$:
  - Select at random single elf capable of deliver in the house h so that the probability that a given pair (e, h) is chosen is x^*_{eh}.
  - Let $\tilde{x}_{eh}$ be equal 1.
  - For all other elves e' capable of deliver in the house h, let $\tilde{x}_{e'h}$ be equal to 0.

Since the algorithm will always choose one single elf capable of deliver in the house h, $\tilde{x}$ is a feasible solution to the original ILP problem. Now we need to show that this solution is, in expectation, just as good as the LP optimum.

But this is true because for any particular elf $e \in E$, the load on e is $\sum_{e \in E} \tilde{x}_{eh}$. For any particular pair (e, h) with $h \in H$, the probability that $\tilde{x}_{eh} = 1$ is x^*_{eh}. Thus the expected value of the load on an elf $e \in E$ is $\sum_{h \in H_e} x^*_{eh}$, which is at most OPT.

Exercise 1.c

In order to show that the algorithm is unlikely to deliver very bad results, the key point is to notice that for any particular elf e, the load on e is a sum of independent $\{0,1\}$ random variables and so we can use the formula: $P(X \geq (1+\varepsilon)\mu) < exp(-\mu\ min\{\varepsilon, \varepsilon^2\}/3)$, where $E[X] = \mu \leq OPT$. Now, let ε be just large enough so that $exp(-\mu\ min\{\varepsilon, \varepsilon^2\}/3) = \frac{1}{m^2}$. Then, we can say that:

$$P(\text{load on a single elf} \geq (1+\varepsilon)OPT) < \tfrac{1}{m^2}$$

Thus, by the union bound:

$$P(\text{load on any elf} \geq (1+\varepsilon)\ OPT) = P(\text{load on any elf} \geq (OPT + \varepsilon OPT)) \leq$$
$$\sum P(\text{load on a single elf} \geq (1+\varepsilon)OPT) \leq \frac{1}{m}$$

So it follows that:

$$P(\text{load on any elf} \leq (OPT + \varepsilon OPT)) = 1 - P(\text{load on any elf} \geq (OPT + \varepsilon OPT)) \geq 1 - \tfrac{1}{m}$$

Now we can prove that

$$\begin{aligned} & e^{(\frac{-OPT\ min\{\varepsilon,\varepsilon^2\}}{3})} = \tfrac{1}{m^2} \\ & e^{(\frac{OPT\ min\{\varepsilon,\varepsilon^2\}}{3})} = m^2 \\ & \tfrac{OPT\ min\{\varepsilon,\varepsilon^2\}}{3} = ln(m^2) \\ & min\{\varepsilon, \varepsilon^2\} = \tfrac{6ln(m)}{OPT} \end{aligned}$$

So $\varepsilon \cdot OPT$ is at most $max\{6ln(m), \sqrt{6ln(m)OPT}\}$, so we can say that, with high probability (i.e at least $1 - \frac{1}{m}$), the maximum finishing time is $O(OPT + \varepsilon \cdot OPT) = O(OPT + ln(m))$

Exercise 2

In order to prove that the Santa problem of finding whether it is possible to build 187 different sleighs with no two sleighs that have the exact same pieces is NP-Hard, we need to start from a known NP-Hard problem and reduce this known NP-Hard problem to an instance of the Santa problem by defining a polynomial time reduction which will allow us to say that if we can find a solution for the original NP-Hard problem, by applying the defined reduction to get an instance of the Santa problem this will result in having also a solution to the Santa problem, while whenever the NP-Hard problem that we reduce to an instance of the Santa problem will not have a solution, also the Santa problem will not have a solution.

Now, in order to reduce SAT to an instance of our Santa Problem we need to start from an instance of the SAT problem and find a way to make sure that if the instance of the SAT problem is satisfiable, this correspond to have that the Santa problem can be satisfied.

We start from an instance of the SAT problem (i.e a CNF formula ϕ) with $C_1, ..., C_k$ clauses and n literals. In our problem there will be arbitrary constraints that will preven us to put together some pairs of pieces in the same sleigh. We define an incompatible pair of pieces if it is represented by a literal x_i and it negation. So in order to build sleighs with some arbitrary constraints that will prevent some pieces to be put together in the same sleigh, we need to reduce out problem in such a way that we never pick a literal and its negation together. So, we define a graph G_ϕ in the following way: for each literal x_i in each clause we create a vertex and we connect with an edge this vertex with all the vertices from all the other clauses except if the vertex corresponds to the negation of x_i. Formally, we define the graph G_ϕ in with the following set of vertices defined as $V = \{\langle a, i\rangle \text{ s.t } a \in C_i\}$ and the set of edges defined as $E = \{(\langle a, i\rangle, \langle b, j\rangle) \text{ s.t } i \neq j \text{ and } b \neq \overline{a}\}$.

Of course this reduction can be done in polynomial time, so we need to prove that is correct. In order to prove the correctness of this reduction, let's now consider boolean assignment to the variables in of the SAT instance ϕ such that ϕ is satisfiable. If ϕ is satisfiable, this means that at least one literal from each clause is true. Then, if we consider the vertices corresponding to the literals that are set to true in the formula (and of course we will have that at least one literal per clause will be true if ϕ is satisfiable), because by construction we never connect a vertex with another vertex that corresponds to its negation in the CNF formula, the set of pieces that we select by picking the literals from different clauses that are true in the formula, will be a compatible configuration of pieces to build a sleigh.

On the other hand, if we pick a compatible configuration of pieces from the graph, by setting the literarls corresponding to the vertices chosen in the formula to true, this will result in having that the CNF formula is satisfied, because we pick at least one vertex from the vertices corresponding to a single clause and so, since at least one of them will be picked and we will set to true its corresponding term in the CNF formula, the CNF formula of the SAT instance will be satisfied.

So, since we proved that we can reduce SAT to our Santa Problem, and we proved that if we have an instance of SAT and this is satisfied, and if we apply the reduction we have an instance of the Santa Problem which is also satisfied, and also the viceversa, and we reduce the SAT instance into a Santa Problem instance in polynomial time, we have proved that Santa's Problem is at least as hard as SAT, and so since SAT is NP-Hard then also the Santa Problem is NP-Hard.

Exercise 3

Exercise 3.1

In order to design a cut game in which the Price of Anarchy is 2, let's consider a graph with 4 vertices where each vertex it's connected by an edge of unit weight with exactly 2 other vertices. In this configuration, we can have two possible situations in which we have a pure Nash equilibrium: the first is when, for each player i that chooses the strategy s_i, one of its neighbours has made the same choice and the other neighbour has made the opposite choice. This is of course a PNE, because we have that for each player, if it changes its strategy, will not improve its utility, because choosing the opposite strategy will however result in having one neighbour in its side and the other in the opposite side, so the utility of each player in this situation is $u_i = 1,\ \forall i = 1, 2, 3, 4$, and so the social optimal utility is equal to 4.

The second situation in which we have a PNE is when each player that chooses the strategy s_i has both neighbours that have made the opposite choice (i.e. they are in the opposite side of the cut). This is of course a PNE because we have that for each player, if it changes its strategy, can only worsen its utility, so it's not interested in changing its strategy. So the utility of each player in this situation is $u_i = 2,\ \forall i = 1, 2, 3, 4$, so the social utility is equal to 8, and this also correspond to the situation in which we have that the social utility is the optimal one (since each vertex has exactly two neighbours, each player i cannot have an utility > 2, so the social utility cannot be > 8).

So, since the worst possible PNE is the one in which the social utility is equal to 4, in this game the Price of Anarchy will be exactly:

$$\text{PoA} = \frac{\text{Best possible social utility}}{\text{social utility in the worst PNE}} = \frac{8}{4} = 2.$$

Exercise 3.2

In order to prove that the Price of Anarchy is at most 2 in the general case, the first step is to observe that in a pure Nash equilibrium, the total weight of the neighbouring vertices of of a vertex v_i in the cut is at least half the total weight of all the neighbouring vertices of the vertex. Let's assume the following notation:

- $W_{\text{neighbours in the cut}_{v_i}}$ = the total weight of the neighbours of a vertex v_i in the cut.
- $W_{\text{neighbours not in the cut}_{v_i}}$ = the total weight of the neighbours of a vertex v_i not in the cut.
- $W_{\text{all neighbours}_{v_i}}$ = the total weight of all the neighbours of a vertex v_i.

In order to prove the above statement, we can observe that:

$$W_{\text{neighbours in the cut}_{v_i}} + W_{\text{neighbours not in the cut}_{v_i}} = W_{\text{all neighbours}_{v_i}}$$

i.e. that the total weight of the neighbours of v_i in the cut plus the total weigth of the neighbours of v_i not in the cut is equal to the sum of all the weights of the neighbors of v_i.

Now, since we are assuming a PNE, we have that, for any vertex v_i:

$$W_{\text{neighbours in the cut}_{v_i}} \geq W_{\text{neighbours not in the cut}_{v_i}} \implies W_{\text{neighbours in the cut}_{v_i}} \geq \frac{1}{2}\ W_{\text{all neighbours}_{v_i}}$$

Now, if we apply the sum we have that

$$\sum_{v_i} W_{\text{neighbours in the cut}_{v_i}} \geq \frac{1}{2}\ \sum_{v_i} W_{\text{all neighbours}_{v_i}}$$

Now we can observe that $\sum_{v_i} W_{\text{neighbours in the cut}_{v_i}}$ is exactly the social utility in the cut game (because it's the sum of the weights of all the edges traversing the cut for every vertex v_i, so we are summing up all the utilities of every player).

On the other hand, we can also observe that $\sum_{v_i} W_{\text{all neighbours}_{v_i}}$ is for sure greater than the social optimal utility (that we will have when the size of the cut is maximum), and this follows from the definition of the cut, because if there is a cut (and in this particular case we are considering the cut of maximum size) then there needs to be at least one edge traversing the cut, so the sum over all the weights of the edges of the neighbouring vertices of every single vertex will be for sure greater than the social optimal utility. So, by multiplying by 2 the two members we get that:

$$2\sum_{v_i} W_{\text{neighbours in the cut}_{v_i}} = 2 \cdot \text{social utility} \geq \sum_{v_i} W_{\text{all neighbours}_{v_i}} \geq \text{social optimal utility}$$
$$\implies \frac{\text{social optimal utility}}{\text{social utility}} \leq 2$$

And since we proved this for a PNE, this is also valid for the worst PNE, so we can say that:

$$PoA = \frac{\text{social optimal utility}}{\text{social utility in the worst PNE}} \leq 2$$

Exercise 4

Exercise 4.2

In order to find a 3-approximation algorithm for this problem, the idea is to apply the k-center algorithm first to the set S of cities, and then once we have found the centers in S, we select for each of them its nearest antenna.

Formalizing the idea, the steps of the algorithm will be the following:

- For any $s_i \in S$ let $a_i \in A$ be its nearest antenna, if there are more nearest antennae then select any of these as a_i. Of course we must have that $dist(s_i, a_i) \leq OPT$ for any i.
- If $|S| \leq k$ then if we take $U = \{a_i | s_i \in S\}$, this gives an optimal solution and the algorithm can terminate.
- If otherwise $|S| > k$, then:
  - Apply the k-center algorithm to the set S of cities. This will give as a solution a set of k cities $S' = \{s_1, s_2, ..., s_k\} \subseteq S$.
  - Take $U = \{a_i | s_i \in S'\}$ as solution.

Proof that the above algorithm gives a 3-approximation for our problem: Let's consider an arbitrary city $c \in S$: if there is a city $s_i \in S'$ at distance at most $2OPT$ from c, then by the triangle inequality $dist(c, a_i) \leq dist(c, s_i) + dist(s_i, a_i) \leq 2OPT + OPT = 3OPT$. On the other hand, if we now assume that there is no city $s_i \in S'$ at distance at most $2OPT$ from c, then, we would have that for any two cities in $s_i, s_j \in S'$ we have that $dist(s_i, s_j) > 2OPT$. But then this means that the set $S' \cup \{c\}$ consists of $k+1$ cities such that any pair is at distance $> 2OPT$, but this is impossible because it would mean that in order to have a solution with value OPT we would need at least $k+1$ antennae. So the algorithm give a 3-approximation for our problem.

Exercise 4.3

In order to prove that finding an α approximation is NP-Hard, the idea is to reduce from the Dominating Set problem. So, let's consider an instance $G = (V, E)$ of the dominating set problem: we can define an instance of our problem in the follwing way:

- For each $v \in V$ we can consider a city s_v and an antenna a_v. With this construction we will have $|V|$ cities and $|V|$ antennae.
- Now let's define the distances in the following way. For each pair of vertices $(v, u) \in V$:
  - Let $dist(a_v, a_u) = dist(s_v, s_u) = 2$.
  - Let $dist(s_v, a_u) = 1$ if $v = u$ or if $(v, u) \in E$.
  - Let the distance be 3 otherwise.

With this configuration, the triangle inequality is satisfied. Now we can notice that any solution for this instance of our problem has value either 1 or 3 (because we have defined the distances equal to 2 only between pairs of both antennas or both cities). Assume that there is a dominating set U of size k. Now take as solution for our problem instance exactly the antennae that correspond with the vertices in U. Then, the value of this solution is 1. To prove the other direction, let's assume that there is a solution for our antennae problem instance of value 1. Then if we take the set U of the vertices that correspond to the k antennae, this is a dominating set.

So we have proved that there is a dominating set of size k if and only if the optimal value of the corresponding instance of our antennae problem is 1. But since the optimal value is either 1 or 3 this also gives us a lower bound of 3 on the approximation ratio. In fact, assume that we have an α-approximation algorithm for our problem with $\alpha < 3$. We need to prove that we can use such an algorithm to solve the dominating set problem, so let's assume the we are given an instance G, k of the dominating set problem and we want to compute if there exists a dominating set with at most k vertices in G. In order to do this, we can reduce it to an instance of our problem applying the reduction already defined above so that we can now apply the α-approximation algorithm. If there is a dominating set of value at most k then this means that the optimal value of our antennae problem instance is 1 and the answer given by the algorithm will be at most $\alpha \cdot 1 < 3$. If, on the other hand, there is no dominating set of value at most k, then the optimal value of the instance of our antennae problem is 3 and the answer given by the algorithm will be (at least) 3. So, by looking at the output of the algorithm we can say if there exists a dominating set of size at most k. But since we know that does not exists a polynomial time algorithm that solves the dominating set problem, assuming that $P \neq NP$, we can conclude that does not exists an α-approximation algorithm with $\alpha < 3$, assuming that $P \neq NP$.