Complexity Theory

1. Show that if $P = NP$, then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$ is *NP-Complete*.

We assume that $P = NP$. This means that all problems in NP have a polynomial time algorithm. Recall that our definition of *NP-Complete* requires polynomial time reductions.

So, take any arbitrary language $A \in P$ that is not $\emptyset$ or Σ^*. This means that there is at least one input that returns YES (let's call it i_y) and at least one input that returns NO (let's call it i_n). Now, to show A is NP-Complete, we need to show that every other problem in P can be reduced to it in polynomial time. The reductions are simple. Intuitively, we are going to simply solve the other problem as our "reduction" and then output a trivial input for A depending on the result. Given any other arbitrary problem $B \in P$, we reduce to A as follows. B has a polynomial time solver. Run this solver on the input to B. If the solver says YES, then our reduction will output i_y. If the solver says NO, then output i_n. Thus, the reduction is obviously valid because it is polynomial time, and it always outputs some string to A that will produce the same result.

We should address why this approach does NOT work with $\emptyset$ and Σ^*. This is because these languages do not have at least one example of each possible output ($\emptyset$ never says YES and Σ^* never says NO). So, the reduction above doesn't work because for an arbitrary problem B, there are two possible outputs but anything the reduction produces will be put into a solver for $\emptyset$ or Σ^* that can only output one possible answer. Thus, the reduction does not exist for these two trivial languages.

2. *PSPACE* is the complexity class containing all problems that can be solved using a *Deterministic Turing Machine* that uses at most a polynomial amount of additional space (on the tape). Prove that $P \subseteq$ *PSPACE*. (*Hint: Think about a generic problem in P, and the DTM that decides it. Try to use more than a polynomial space with this machine given the allotted time.*)

Suppose we have a problem in P. It must have a TM decider that runs in polynomial time (polynomial steps of computation). A Turing Machine can only move its head one tape cell per step of computation, so in a polynomial amount of steps, it can only reach a polynomal number of tape cells to interact with. Thus, any TM that runs in polynomial time must use a polynomial amount of space as well. Thus, $P \subseteq$ *PSPACE*.

3. Recall the *knapsack problem*: Given a list of item values $V = \{v_1, v_2, ..., v_n\}$, their respective weights $W = \{w_1, w_2, ..., w_n\}$, a target value $k \in \mathbb{N}$, and a knapsack capacity $W \in \mathbb{N}$, does there exist a subset of items such that the sum of the values of the items is at least k and the sum of the weight of the items is less than or equal to W. In other words, choose items I such that $\sum_{i\in I}(v_i) \geq k$ and $\sum_{i\in I}(w_i) \leq W$. Show that the *knapsack problem* is NP-Complete. You can use any other known NP-Complete problem for your reduction (you might want to look up other NP-Complete problems for this, but you don't have to!).

Verification is simple: Given the list of items you steal, simply sum the weight and value and make sure the two inequalities hold. This is trivially polynomial time (just a couple of loops and if statements).

For the reduction, the most natural choice is subset-sum. This problem states that given an array $A = \{a_1, a_2, a_3, ..., a_n\} \mid a_i \in \mathbb{N}$ and a target $t \in \mathbb{N}$, find a subset of values of A that sum to t exactly.

The reduction is as follows: Given A and t, inputs to the subset problem, we generate an instance of knapsack by:

- $V = W = \{a_1, a_2, ..., a_n\}$: The values of the items and the weights of the items are all just the original list A from our subset sum instance.
- $W = k = t$: The capacity is the target value is the subset sum target value.

This reduction works because setting $W = k$ means that the items that are stolen must exactly fill the knapsack capacity in order to reach the intended value k. This can only happen when that value and capacity is exactly t, providing a solution to the original subset sum instance.

4. You are working for *Shoogle* (A data company that definitely does not exist) and you are given the following task from your boss: *Shoogle* is interested in investing in their employees by offering a large amount of professional development. They have collected a series of *workshops* that each cover a subset of *topics*. For example, workshop 1 might cover machine learning and databases, workshop 2 might cover machine learning and data mining, and workshop 3 might cover advanced data structures (note that each workshop can cover multiple topics and there can be overlap across workshops). In addition, *Shoogle* wants to make sure that over the course of the professional development workshops, at least one topic in each sub-area of interest is represented. For example, they might want to make sure there is at least one AI workshop (machine learning or reinforcement learning or neural networks, etc.), at least one systems workshop (databases or cloud computing), etc.

   So...the problem is this: Given the availability of the professional development workshop, what topics each covers, and which areas need to be covered, can you devise a schedule over multiple weeks that ensures that at least one topic in every area is covered by at least one workshop?

   Let's formalize this a bit: The input contains multiple items:

   - T: A list of individual topics that can be covered (e.g., machine learning).
   - $W = \{w_1, w_2, ..., w_m | w_i \subseteq T\}$: A list of workshops, each of which is a subset of the topics T that will be covered in that workshop. There are m workshops total.
   - $S = \{s_1, s_2, ..., s_n | s_i \subseteq W\}$: The schedule availability for each week. Each s_i is the subset of workshops that can be scheduled in week i. There are n weeks.
   - $A = \{a_1, a_2, ..., a_p | a_i \subseteq T\}$: The subject areas that need to be covered. Each area is a list of Topics in that area.

   You want to output Yes if there is some schedule of workshops over the n weeks such that every area has at least one topic in T that is covered in at least one workshop.

   This problem is *NP-Complete*. Prove it! For your reduction, use *3-SAT*.

Verification is simple, so I won't bother explaining it in detail. Simply look at what workshops were scheduled and what topics they cover and loop to make sure every area has at least one topic covered.

For the reduction, we use 3-SAT. Given a formula θ in 3-CNF form, we convert it into a scheduling problem as follows:

- Create one topic for each variable in θ and its negation (e.g., $T = \{x_1, \bar{x_1}, x_2, \bar{x_2}, ...\}$)
- Create one topic covered per workshop. $W = \{w_1 = \{x_1\}, \bar{w_1} = \{\bar{x_1}\}, ...\}$ This effectively makes the list of topics and workshops the same, with a 1 to 1 correspondence.
- Create one area per clause in θ. For each clause, the three literals in that clause map to the three topics / workshops within that area.
- Create the schedule so that each variable and its negation workshop are available each week. For example, x_1 annd $\bar{x_1}$ are available during week 1, x_2 and $\bar{x_2}$ during week 2, etc.

This is a valid reduction because if the schedule can be done such that all areas are covered, then the formula is satisfiable. The workshops that get scheduled correspond to the variables that should be assigned to True, and since each area is associate with a clause, some variable from each clause in θ was selected.