Prob. 1	Prob. 2

Team members: Küng, Pirelli, Schubert, Dousse, Vu

Problem 1.

Let us prove that *TWICE_SAT* is NP-complete by reducing 3SAT to it. Given a 3CNF formula ϕ with k variables, we create the formula $\phi' := \phi$ *and* $(x_{(k+1)}$ *or not* $(x_{(k+1)})$. Clearly, if ϕ has at least one satisfying assignment, ϕ' must have at least two satisfying assignment, one where $x_{(k+1)}$ is true and one where it is false. Therefore, *TWICE_SAT* can be reduced to 3SAT, which means *TWICE_SAT* is NP-complete.

Problem 2.

Let us define our algorithm this way :

loop for each variable x:
remove all clauses containing variables that only appear once

remove all clauses containing literals that appear twice (in the entire formula)

remove all clauses containing variables that appear twice in the clause

find clauses c_1 and c_2 such that $c_1 = X \text{ or } Y$ *and* $c_2 = \text{ not}(x) \text{ or } Z$ for some Y, Z (clauses)

if c_1 and c_2 exist, remove c_1 and c_2, add $c_3 := Y \text{ or } Z$

if any clause is empty and the loop isn't finished, the formula is not satisfiable.
if the loop finishes, the formula is satisfiable.

We always remove clauses that can be trivially satisfied: the ones with unique variables (just set that variable to true or false), the ones with the same variable twice (which means it doesn't appear elsewhere), and the ones with literals that appear twice (which means the opposite literal doesn't appear, so we can set the variable to true or false).

Then we reduce the formula step by step. The principle is simple: if there are clauses of the form "$x \text{ or } Y$" and "$\text{not}(x) \text{ or } Z$", they are equivalent to one bigger clause "$Y \text{ or } Z$" since either x is false and Y needs to be true, or not(x) is false and Z needs to be true.

If we find an empty clause before the end of the loop, we abort because empty clauses are not satisfiable.