1 Problem 1

Prove whether the following languages are decidable, where $M$ and $N$ are descriptions of Turing Machines.

a. $\{\langle M, w \rangle | M$‘s head never moves left on input $w.\}$

b. $\{\langle M \rangle | M$ accepts all strings starting with 010.$\}$

c. $\{\langle M, N \rangle |$ Every string in $L(M) \cap L(N)$ starts with 101.$\}$

2 Problem 2

Determine and prove whether each of the following languages is decidable, undecidable but Turing-recognizable, or not Turing-recognizable. $M$ is interpreted as a Turing machine, $L(M)$ denotes the language of $M$, and $k$ is
interpreted as a positive integer.

\[ A = \{ (M, k) \mid \text{On all inputs } w \in \Sigma^* \text{, } M \text{ runs for at most } k \text{ steps} \} \]

\[ B = \{ (M, k) \mid L(M) \text{ is decidable by a Turing machine that, on all inputs } w \in \Sigma^* \text{, runs for at most } k \text{ steps} \} \]

\[ C = \{ (M) \mid \text{There exists a positive integer } k \text{ such that, on all inputs } w \in \Sigma^* \text{, } M \text{ runs for at most } k \text{ steps} \} \]

**Hint:** if a Turing machine runs for at most \( k \) steps, how much of its input can it read?

### 3 Problem 3

The beloved Qwop, after taking home the gold to his small, but resilient, country, has won himself international acclaim. Qwop merch sales have skyrocketed, and industry cannot keep up with the demand. In order to combat this, they have devised a Qwop game to keep the public busy until more Qwop merch can be produced. This game consists of playing a series of Qwop games on a keyboard that extends infinitely to the right, where each game uses keys that are adjacent on the board, and a game’s keys occur in chronological succession. A valid generalized Qwop game starts with the first game’s keys near the left side of the keyboard, and each sequential game’s keys are further right on the keyboard than the game before it, but may include keys of previous games so long as they are adjacent. This game is formally defined below.

Given a finite alphabet \( \Sigma \), let \( C = \{ S_1, \ldots, S_k \} \) denote a collection of subsets of \( \Sigma \), i.e. \( S_i \subset \Sigma \) for \( 1 \leq i \leq k \). Consider the following language:

\[ \text{GENERALIZEDQWOP} = \{ (\Sigma, C, n) \mid \text{there exists a string } w \in \Sigma^* \text{ such that } |w| \leq n, \text{ and for each } i, \text{ all the elements of } S_i \text{ occur together in some order as a substring of } w \} \]

Recall that a substring of a word \( w = w_1 \ldots w_m \) where \( w_i \in \Sigma \) is a string of the form \( w_jw_{j+1} \ldots w_l \) for some \( 1 \leq j \leq l \leq m \). In particular, a substring is not the same as a subsequence.

Prove that the language \( \text{GENERALIZEDQWOP} \) is NP-complete.
4 Problem 4

We define the language \textsc{SetSplit} as follows.

\[
\text{SetSplit} = \{ \langle S, F \rangle : \text{S is a set, and } F \text{ is a set of subsets of } S \\
\text{and there is a partition of } S \text{ into two subsets } S_1 \text{ and } S_2 \text{ such } \\
\text{that no subset in } F \text{ is entirely contained in either } S_1 \text{ or } S_2. \}
\]

For example, if \( S = \{1, 2, 3, 4\} \) and \( F = \{\{1, 2\}, \{3, 4\}\} \), then \( F \in \text{SetSplit} \), since we can choose \( S_1 = \{1, 3\} \) and \( S_2 = \{2, 4\} \).

Recall the variation of 3-SAT called 3-\textsc{Not-All-Equal-SAT} (3-\textsc{NAE-SAT}).

\[
\text{3-NAE-SAT} = \{ \langle C, n \rangle : C \text{ is a collection of triples } C_1, C_2, \ldots, C_m \\
\text{of literals over } n \text{ boolean variables } x_1, x_2, \ldots, x_n \text{ such that } \exists a_1, a_2, \ldots, a_n \\
\text{such that every triple } C_i \text{ has a true and false literal } \}
\]

Given \( \langle C, n \rangle \), each triple consists of three literals \((x_i, x_j, x_k)\). If each variable can be assigned such that for all triples, not all literals have the same truth value, then \( \langle C, n \rangle \) is an instance of 3-\textsc{NAE-SAT}.

Using a reduction from 3-\textsc{NAE-SAT}, prove that \textsc{SetSplit} is NP-complete.

5 Problem 5

The year is 1997, and IBM’s Deep Blue is all the rage! It just beat Garry Kasparov, the world chess champion. How did it do it? One trick is that it has a clever heuristic that can tell whether there is a move to checkmate one’s opponent.

Suppose that Deep Blue is given a Boolean formula \( \phi \) over two sets of variables, \( x_1, \ldots, x_n \) and \( y_1, \ldots, y_m \). (The formula \( \phi \) somehow captures the current chess board.) Deep Blue gets to decide how to set the truth values of \( x_i \)'s while Garry Kasparov gets to assign the values to \( y_1, \ldots, y_m \). Deep Blue can win if there exists a setting to \( x_1, \ldots, x_n \) (i.e. a move) such that no matter how Kasparov sets \( y_1, \ldots, y_m \), \( \phi \) evaluates to True. Formally, Deep Blue’s success hinges on its ability to decide the language \textsc{CheckMate}, defined as follows:
CheckMate = \{\langle \phi(x_1, \ldots, x_n, y_1, \ldots, y_m) \rangle \mid \exists a_1, \ldots, a_n \text{ such that } \forall b_1, \ldots, b_m \phi(a_1, \ldots, a_n, b_1, \ldots, b_m) = T\}

a. Recall the language ChromaticNumber we defined in class:

ChromaticNumber = \{\langle G, k \rangle \mid G \text{ is a } k\text{-colorable graph that cannot be colored with } k-1 \text{ colors}\}

Show that ChromaticNumber \leq_p CheckMate.

**Hint:** the Cook-Levin theorem gives us a procedure that computes a Boolean formula \( \varphi \) from the values \( \langle G, k \rangle \) such that \( \varphi \) is satisfiable if and only if \( G \) is \( k \)-colorable. You can use this procedure (without worrying about how it works) in your reduction.

b. Explain why part (a) implies that CheckMate is both NP- and coNP-hard.

c. Suppose that researchers at IBM have developed a top secret algorithm \( A \) that decides SAT in polynomial time. Give an algorithm that decides CheckMate in polynomial time using \( A \) as a subroutine.

**Hint:** it will actually help if, as an intermediate step, you first come up with a non-deterministic polynomial-time algorithm.