Problem 1

Like any good party, U. K. Liptus’s party has a theme: the class NP. Rather than stick with the usual party tricks like juggling and tree trivia, her guests impress one another with their knowledge of the class NP. Even more exclusive is the class of NP-complete languages.

Juniper shows up to the party with a party trick in hand: Show that if P = NP, then every language $A \in P$, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete. Prove that $\emptyset$ and $\Sigma^*$ cannot be NP-complete.

Problem 2

Laurel also comes to the party, and history repeats itself with another showdown between Laurel and Juniper. Laurel’s problem is as follows: Let $A$ and $B$ be languages such that $A \in NP$ and $B \in NP$.

a. Prove that $AB = \{ x \cdot y \mid x \in A \text{ and } y \in B \} \in NP$, so that NP is closed under concatenation.

b. Prove that $A^* = \{ x_1x_2\ldots x_n \mid n \geq 0 \text{ and } x_i \in A \} \in NP$, so that NP is closed under the Kleene star operation.
Problem 3

Buck and Buck I. are handling the party’s guest list. Each guest’s name should be on its own line on the list, but all of the names have been concatenated together! Unfortunately, all of the whitespace keys on their computer broke yesterday. To verify a guest’s RSVP, they thus need to check if the guest’s name is a substring of the whole list. However, they can only find a Turing machine that computes palindromes, not substrings.

Define the following two languages:

- **SUBSTRING** = \{⟨x, s⟩ | x is a substring of s\}
- **PALINDROME** = \{⟨w⟩ | w = w^R, where w^R is the reverse of w\}

Help Buck and Buck I. come up with an algorithm to reduce **SUBSTRING** to **PALINDROME**. That is, assuming you have a machine M that can decide whether a string w is in **PALINDROME**, construct a machine that, on input ⟨x, s⟩, uses M as a subroutine to decide if x is a substring of s. Your algorithm must run in polynomial time.

**Hint:** If necessary, you may run M multiple times during your computation. Note that if your reduction is polynomial time, you can run M at most a polynomial number of times.
The following questions are lab problems.

Lab Problem 1

For this problem, only consider languages over the alphabet \{0, 1\}.

a. Let \( L_1 \) and \( L_2 \) be two languages in NP.
   (i) Must \( L_1 \cup L_2 \) be in NP?
   (ii) Must \( L_1 \cap L_2 \) be in NP?

b. Now, suppose \( L_1 \) and \( L_2 \) are both NP-complete.
   (i) Must \( L_1 \cup L_2 \) be NP-complete?
   (ii) Must \( L_1 \cap L_2 \) be NP-complete?

Lab Problem 2

In their quest to solve their guest list problem, Buck and Buck I. come across an extra eight cells of a Turing machine tape. They wonder if this will help increase their computational power at all.

a. Suppose you have an augmented TM which has an extra tape of length 8. Describe this augmented TM and explain why it is equivalent to a standard TM. That is, explain why a language can be recognized by this augmented TM if and only if it can be recognized by a standard TM.

   \textbf{Hint:} Show that each machine can be used to simulate the other.

b. Consider the language of all strings over \( \Sigma = \{0, 1\} \) that have some substring of length 8 that repeats, not necessarily consecutively. More formally, this language is the set of strings of the form \( wxyxz \), where \(|x| = 8\) and \( w, y, z \) are of arbitrary lengths (possibly zero).

   Describe a nondeterministic Turing machine that decides this language. Please try to use nondeterminism as much as possible (it will make your job a lot easier). Briefly analyze your TM and explain both why it is correct and why it halts on all inputs, on all branches.
Lab Problem 3

Will Owens comes to the party to finally meet his mysterious neighbor U. K. Liptus. He finds her in the shrubbery garden and introduces himself. U. K. Liptus is very interested in the fact that he’s been working on circuit problems for the Pinary Tree Company. She suggests that he start working on Turing machines, which is where most of the Discrete Logging Company’s research lies. In particular, she tells Will about a generalization of the halting problem to a more general class of computational models.

U. K. Liptus suggests a new kind of model called a SuperTM$_1$ that has access to the following magical black box: Given any pair $\langle M, x \rangle$ where $M$ is an encoding of a TM and $x$ is a string, this black box can correctly tell us whether $M$ will halt on input $x$.

a. Prove that every Turing-recognizable language $L$ can be decided by a SuperTM$_1$.

b. Consider the SuperTM$_1$ halting problem:

$$L_1 = \{ \langle M, x \rangle \mid M \text{ is a SuperTM}_1 \text{ that halts on input } x \}$$

Prove that there does not exist a SuperTM$_1$ that decides $L_1$.

c. Part (b) can be generalized to demonstrate that there is an infinite sequence of computational models: SuperTM$_1$’s, SuperTM$_2$’s, etc. For all $i$, SuperTM$_i$’s can magically solve the SuperTM$_{i-1}$ halting problem, but $L_i$ is not SuperTM$_i$-decidable.

We say that a language $A$ is $\omega$-decidable if there exists some $n$ for which some SuperTM$_n$ decides $A$. Give an example of a language that is not $\omega$-decidable (and provide a short proof that it is not $\omega$-decidable).