Problem 1
Consider the Boolean formula \( f \):

\[
(x_1 \lor x_2 \lor x_3) \land (\neg x_2 \lor x_4) \land (\neg x_3 \lor x_4)
\]

Part 1. One possible variable ordering for \( f \) is \( x_1 < x_2 < x_3 < x_4 \). How many possible variable orderings are there for \( f \)?

Part 2. Consider two different variable orderings for \( f \), \( x_1 < x_2 < x_3 < x_4 \) and \( x_4 < x_3 < x_2 < x_1 \). Draw the BDDs for \( f \) for each of the selected variable orderings. For each BDD node \( v \), label its outgoing edge to low(\( v \)) with “0” and its outgoing edge to high(\( v \)) with “1”.

BDD Pseudo-Code Primitives
The following is the pseudo-code for the \texttt{AND} operation on BDDs presented in the class. Recall that we assume a fixed variable ordering that all BDDs must follow.

\[
\text{Bdd} \text{ AND(Bdd} f, \text{Bdd} g)\
\{
\text{ if (} f \text{ == ZERO()} \text{ ||} g \text{ == ONE()} \text{)} \\
\quad \text{ return } f; \\
\text{ if (} f \text{ == ONE()} \text{ ||} g \text{ == ZERO()} \text{)} \\
\quad \text{ return } g; \\
\text{ if (} \text{VAR}(f) \text{ == VAR}(g) \text{)} \\
\quad \text{ return ITE(VAR}(f), \text{AND(LOW}(f),\text{LOW}(g)), \text{AND(HIGH}(f),\text{HIGH}(g))); \\
\text{ if (} \text{VAR}(f) \text{ < VAR}(g) \text{)} \\
\quad \text{ return ITE(VAR}(f), \text{AND(LOW}(f),g), \text{AND(HIGH}(f),g)); \\
\text{ return ITE(VAR}(g), \text{AND(LOW}(g),f), \text{AND(HIGH}(g),f)); \\
\}\n\]

The above pseudo-code introduces the following primitives:

- \texttt{==} checks equality between two BDDs.
- \texttt{ZERO()} returns the constant “0” BDD. \texttt{ONE()} returns the constant “1” BDD.
- \texttt{VAR(f)} returns the variable labeling the root of the BDD \( f \).
• LOW(f) and HIGH(f) return the “low” and “high” sub-BDDs of f, respectively.

• For two variables v1 and v2, v1 < v2 iff v1 appears before v2 in the variable ordering.

• ITE(v,f,g) returns the BDD h such that \( \text{VAR}(h) = v \), LOW(h) = f, and HIGH(h) = g. In the special case when f = g, we have ITE(v,f,g) = f.

We will use these primitives in the next problem.

**Problem 2**

For a BDD f, we write Formula(f) to denote the Boolean formula that f represents. For a Boolean formula \( \Phi \) and a variable v, and a Boolean value b, we write \( \Phi[v = b] \) to mean the Boolean formula obtained by replacing all occurrences of v in \( \Phi \) with b. For example, suppose \( \Phi = (\neg x_1 \lor x_2) \). Then \( \Phi[x_1 = 1] \) is \( x_2 \).

**Part 1.** Using the primitives introduced earlier, write the pseudo-code for the function SUB1 that:

1. takes three arguments – a BDD f, a variable v, and a Boolean value b, and
2. returns the BDD h such that Formula(h) = Formula(f)[v = b].

In other words, your pseudo-code should look like the following:

```plaintext
Bdd SUB1(Bdd f, Var v, Bool b)
{
    ...
}
```

Let \( \Phi \) be a Boolean formula, and \( \Sigma \) be a conjunction of literals. Then \( \Phi \diamond \Sigma \) denotes the formula obtained from \( \Phi \) as follows:

- for each literal \( x_i \) appearing in \( \Sigma \), replace all occurrences of \( x_i \) in \( \Phi \) with “true”.
- for each literal \( \neg x_i \) appearing in \( \Sigma \), replace all occurrences of \( x_i \) in \( \Phi \) with “false”.

**Part 2.** Using the primitives introduced earlier, write the pseudo-code for the function SUB2 that:

1. takes two arguments – a BDD f, and a BDD g such that Formula(g) is a conjunction of literals, and
2. returns the BDD h such that Formula(h) = Formula(f) \( \diamond \) Formula(g).

In other words, your pseudo-code should look like the following:

```plaintext
Bdd SUB2(Bdd f, Bdd g)
{
    ...
}
```
Part 3. Using the primitives introduced earlier, write the pseudo-code for the function \texttt{IMPL} that:

1. takes two arguments – a BDD $f$, and a variable $v$, and
2. returns “0” if $\text{Formula}(f) \Rightarrow \neg v$
3. returns “1” if $\text{Formula}(f) \Rightarrow v$
4. returns “-1” otherwise.

In other words, your pseudo-code should look like the following:

```c
int IMPL(Bdd f,Var v) {
    ...
}
```

Assume that $f$ is not the “0” BDD. It is OK to add extra helper functions and global variables to your pseudo-code.