Question 7.3
Consider the problem of deciding whether a propositional logic sentence is true in a given model.

A. Write a recursive algorithm PL-TRUE?(s, m) that returns true if and only if the sentence s is true in the model m (where m assigns a truth value for every symbol in s). The algorithm should run in time linear in the size of the sentence. (Alternatively, use a version of this function from the online code repository.)
Model -> mod
Sentence -> s

```
def PL-TRUE (s, mod={}):
    if s in (True, False):
        return s
    op, args = s.op, s.args
    if is_prop_symbol(op):
        return mod.get(s)
    elif op == '~' :
        return not PL-TRUE(args[0], mod)

    //p and q are the arguments
    if op == '|' :
        return PL-TRUE(p, mod) or PL-TRUE(q, mod)
    elif op == '&' :
        return PL-TRUE(p, mod) and PL-TRUE(q, mod)
    if op == '==>' :
        return PL-TRUE(~p | q, mod)
    elif op == '<==' :
        return PL-TRUE(p | ~q, mod)
    if op == '<=>' :
        return PL-TRUE(p, mod) == PL-TRUE(q, mod)
```

Above is a simple version to solve the problem in linear time n, i.e. the size of the expression (O(n)).

B. Give three examples of sentences that can be determined to be true or false in a partial model that does not specify a truth value for some of the symbols.

Regardless of the value of some symbols, we can pick instances that will remain true or false. Three such examples are,
$\alpha \vee \neg \alpha \vee \beta$: This will remain true even if the partial model doesn't assign values to all.
$(\alpha \wedge \beta) \Rightarrow \alpha$: This will remain true even if the partial models doesn't assign values to all.
$\alpha \wedge \neg \beta \wedge \beta$: This will remain false even if the partial models doesn't assign values to all.

C. Show that the truth value (if any) of a sentence in a partial model cannot be determined efficiently in general.

As shown in (b), a partial model can in certain situations determine the truth value of a phrase by assessing whether it is valid or unsatisfiable, regardless of the value of unassigned symbols. As we learned in class, determining validity or unsatisfiability is an NP-complete problem, which means there is no generic algorithm that can solve it in polynomial time.

D. Modify your PL-TRUE? algorithm so that it can sometimes judge truth from partial models, while retaining its recursive structure and linear run time. Give three examples of sentences whose truth in a partial model is not detected by your algorithm.

We can sometimes judge truth from partial models using a method known as Short-circuit evaluation. It is the semantics of some Boolean operators in some programming languages in which the second argument is executed or evaluated only if the first argument does not suffice to determine the value of the expression. The algorithm in (A) already follows this evaluation while retaining its recursive structure and linear runtime.
if op == '|' : return PL-TRUE(p, model) or PL-TRUE(q, model)
This will return true if PL-TRUE(p, model) gives true even if model is not able to evaluate PL-TRUE(q, model) as Short-circuit is supported by python.

Three examples which truth is partial model is not detected by the suggested algorithm:

- $\alpha \vee \neg \beta \vee \beta$: if the partial model was unable to evaluate α it won't return true for this sentence.
- $\alpha \vee$ true: if the partial model was unable to evaluate α it won't return true for this sentence.
- $\alpha \wedge$ False: if the partial model was unable to evaluate α it won't return false for this sentence.

E. Investigate whether the modified algorithm makes TT-ENTAILS? more efficient.

The use of early termination in Boolean operators will result in a significant speedup. This is due to the fact that it is not necessary to enter values for all variables. Regardless of the other variable values, the value will be returned as soon as the result is true. Hence, the algorithm becomes more efficient.

Question 7.6
Prove, or find a counterexample to, each of the following assertions:

A. If $\alpha \models \gamma$ or $\beta \models \gamma$ (or both) then $(\alpha \wedge \beta) \models \gamma$

The following statement is true.
This follows the Monotonicity feature we know that $M(\alpha \wedge \beta) \subseteq M(a)$ and also $M(\alpha \wedge \beta) \subseteq M(\beta)$.

Let's prove it through truth table.

α	β	γ	α ∧ β
True	True	True	False
True	False	True	False
False	True	True	False
False	False	False	False

With reference to the table, we can see that whenever α and β are the same then even α ∧ β results in same. Hence, proving the above statement.

B. If α |= (β ∧ γ) then α |= β and α |= γ

The following statement is true.
From set theory, we know that M (β ∧ γ) ⊆ M (β) and M (β ∧ γ) ⊆ M (γ)
If M (a) ⊆ M (β ∧ γ) and using the thumb rule above, M (a) ⊆ M (β ∧ γ) ⊆ M (γ) and M (a) ⊆ M (β ∧ γ) ⊆ M (β).
If α holds for β and γ together then it should hold for β individually and γ individually too.

This can also be proved by a truth table,

β	γ	β ∧ γ
True	True	True
True	False	False
False	True	False
False	False	False

The first row satisfies the claim made in the statement.

C. If α |= (β ∨γ) then α |= β or α |= γ (or both).

The following statement is false.

Take Prop = {p, q}, Σ = {p}, α = (p ∧ q), β = (¬(p ∧ q)) Then:
models(Σ) = {{p}, {p, q}}
models(α) = {{p, q}}
models(β) = {∅, {p}, {q}}
Here, models(Σ) ⊆ models(α) ∪ models(β) iff Σ |= (α ∨ β)
but models(Σ) is not a ⊆ models(α) therefore Σ does not entail α
and models(Σ) is not a ⊆ models(β) therefore Σ does not entail β

Question 7.18
Consider the following sentence: [(Food ⇒ Party) ∨(Drinks ⇒ Party)] ⇒ [(Food ∧Drinks) ⇒ Party].

A. Determine, using enumeration, whether this sentence is valid, satisfiable (but not valid), or unsatisfiable.

A simple truth table has 8 rows and using that we can show that the above sentence is true for all models and hence it is valid.
F = food
P = party
D = drinks
f = false
t = true

F	**P**	**D**	**F=>P**	**D=>P**	**(F=>P) v (D=>P)**	**F ∧ D**	**F ∧ D => P**	**[(F=>P) v (D=>P)] => [(F^D)=>P]**
t	t	t	t	t	t	t	t	t
t	t	f	t	t	t	f	t	t
t	f	t	f	f	f	t	f	t
t	f	f	f	t	t	f	t	t
f	t	t	t	t	t	f	t	t
f	t	f	t	t	t	f	t	t
f	f	t	t	f	t	f	t	t
f	f	f	t	t	t	f	t	t

B. Convert the left-hand and right-hand sides of the main implication into CNF, showing each step, and explain how the results confirm your answer to (a).

Steps to convert to CNF:

1. [(F => P) v (D => P)] => [(F ^ D) => P]
2. [(!F v P) v (D => P)] => [(F ^ D) => P]
3. [(!F v P) v (!D v P)] => [(F ^ D) => P]
4. [(!F v P) v (!D v P)] => [!(F ^ D) v P]
5. [(!F v P) v (!D v P)] => [(!F v !D) v P]
6. ![(!F v P) v (!D v P)] v [(!F v !D) v P]
7. [!(!F v P) ^ !(!D v P)] v [(!F v !D) v P]
8. [(F ^ !P) ^ !(!D v P)] v [(!F v !D) v P]
9. [(F ^ !P) ^ (D ^ !P)] v [(!F v !D) v P]
10. (F ^ !P ^ D ^ !P) v (!F v !D v P)

11. (F ^ D ^ !P) v (!F v !D v P) -> CNF

In the 11th step, the sentence is in the form ¬ α ∨ α is also same as α => α, which is valid.

C. Prove your answer to (a) using resolution.

¬ α ∨ α is always true as true ∨ false and false ∨ true are both always true.
From the previous answer, we see that the CNF of the statement is equivalent to ¬ α ∨ α.
So, the CNF of the statement is also always true, hence the original sentence is valid.