Be careful to show each step in a chain of equivalences — don't combine multiple steps in one, even if the multiple steps are applications of the same rule.

The commutative law allows changing the order of operands for $\cup$ and $\cap$ ($A \cup B \equiv B \cup A$ and $A \cap B \equiv B \cap A$), but it is a rule application and thus needs to be cited as a justification (and be its own step). No hidden commutative steps!

The associative law only applies across the same operations — $A \cup (B \cap C) \not\equiv (A \cup B) \cap C$. (To understand this, note that everything in A is in $A \cup (B \cap C)$ but only those things in A which are also in C are in $(A \cup B) \cap C$).) As a result, use parens when there's a mix of $\cup$ and $\cap$ as the precedence rules are not clear in this situation.

By this token, #1e should have parens. Both interpretations $\overline{(A \cup B) \cap C}$ and $\overline{A \cup (B \cap C)}$ were accepted as correct.

The basic statement of DeMorgan's Law applies to two operands, though the book showed that it can be extended to multiple instances of the same operator so you can write $\overline{A \cup B \cup C} \equiv \overline{A} \cap \overline{B} \cap \overline{C}$ with the justification "DeMorgan's Law". However, the book did not establish that it also applies across a mix of $\cup$ and $\cap$ so you need to show that chain of equivalences.

Double complement requires that the two complement operations be applied to the same thing. It does not directly apply to $\overline{A \cap \overline{B \cap \overline{\overline{C}}}}$.

#2 states that a, b, and c are values of type `int` in Java, which means that each have 32 bits. If not all 32 bits are written, fill in the leftmost bits with 0s. That means that c can also be written as `0x0000FFFF`.

Line up operands for bitwise operations starting from the rightmost bit (and remember to fill in the leftmost bits with 0s if they aren't specified).

The 32 bit size also means that left-shifting (`<<`) loses bits from the left end.

#6 asked about both one-to-one and onto; your answer should address both and explain why or why not for each. Don't just cite the definition as your explanation — explain how the particular function does (or doesn't) satisfy the definition. "There's no possibility of having the same output for different inputs" is a only claim, and is pretty much just the definition of one-to-one. *Why* is there no possibility of having the same output for different inputs?

Just showing $f(1)$, $f(2)$, $f(3)$, etc for several values of n is not a proof of onto or one-to-one. (A single value can serve as a counterexample to show that something is false, but to show that something is true one must show it for all values.) Writing out some examples can be helpful for figuring out whether you are trying to show that the function is or isn't, but there either needs to be enough values written out for there to be a clear pattern visible or (better!) a general argument made that applies to any number.