Exercise 1: CNF and satisfiability

Part a

If n is some reasonable measure of the "size" of the formula ($n =$ the number of logical connectives would work in our case), then the complexity of this method will be $O(2^n)$. This is because the "and introduction" and "or elimination" rules could cause the number of nodes at any given level in the tree to double, so in general, the tree will have an exponential number of nodes in it.

Part b

This method will always work. First, we need to check that we can always produce a proof tree whose leaves do not have any logical connectives. Recall that we can use "not introduction" and "not elimination". For any given logical connective that occurs to either side of the turnstyle, there is a rule that can be applied to the given sequent to reduce it to sequents in which that particular connective does not occur and where no new connectives were added. Since the original formula had only finitely many connectives, we can always produces a proof tree whose leaves do not have any logical connectives.

Secondly, we have to verify that the CNF produced is equivalent to the original formula. We can prove this by induction on the size of the proof tree. This essentially comes down to showing that for each rule, the sequents above the line are satisfiable if and only if the sequent below the line is satisfiable. This is a somewhat tedious, but straighforward, case analysis.

We'll show the case for "implies introduction" only. So we need to show that $\Gamma, \alpha \vdash \Delta_1, \beta, \Delta_2$ is true if and only if $\Gamma \vdash \Delta_1, \alpha \Rightarrow \beta, \Delta_2$ is true. If something in Γ is false, or if something in one of Δ_1 and Δ_2 is true, this fact trivially holds (check what it means for a sequent to be true). So we can assume that everything in Γ is true and nothing in Δ_1 and Δ_2 is true. If α and β are both false, then $\alpha \Rightarrow \beta$ is true, so both sequents are true. The analysis for the other combinations of valuations for α and β is similar.

Exercise 2: NP Problems

Part 1

Let n be the number of vertices in the graph.

1. First, nondeterministically assign each vertex in the graph one of the k colors. This takes $O(n)$ time.
2. Now we (deterministically) check if the first step produced a k-coloring. For each vertex v, we have to check that all adjacent vertices have a different color from v. If the edges and color assignments are stored in unsorted lists, it will take $O(n^2)$ time to check each vertex, so it takes $O(n^3)$ time to check the entire graph.

Thus, we have a polynomial time, nondeterministic algorithm for solving the k-coloing problem, i.e. we have an algorithm in NP.