1. Describe a state space in which iterative deepening search performs much worse than depth-first search (e.g. $O(n^2)$ vs. $O(n)$). Explain why. (Do not collaborate on this problem beyond discussing the working of the search algorithms.)

2. A horn clause is a clause with at most one positive (non-negated) literal. How can you modify the DPLL algorithm given in figure 7.16 of R & N to answer a query $KB \models p$ in linear time, if $KB$ is a collection of horn clauses and $p$ is a propositional symbol?

**Ans.** First we negate $p$ to get $\neg p$ and then add it as a clause to the KB. If $KB \land \neg p$ is unsatisfiable then $KB \models p$ and vice versa. Observe that if $KB$ is horn then so is $KB \land p$ (A CNF formula is horn if every clause is).

Our algorithm is as follows:

(a) if every clause in $clauses$ is true in $model$ then return $true$
(b) if some clause in $clauses$ is false in $model$ then return $false$
(c) $P, value \rightarrow FIND\_UNIT\_CLAUSE(clauses, model)$
(d) if $P$ is non-null, return $DPLL(clauses, symbols - P, EXTEND(P, value, model))$
(e) else return $true$

The major modification made above to DPLL is that we do not branch recursively at the end (We have also removed the pure symbol heuristic because it doesn’t affect the worst-case complexity). The procedure thus takes polynomial time since the number of calls made to DPLL is at most the number of variables in the formula. The correctness of the procedure follows from two observations:

(a) Every horn CNF formula without a unit clause is always satisfiable. This is because a satisfying assignment can be found by setting every variable to false. Every clause contains at least one negated literal and is therefore satisfied.

(b) If a horn CNF formula $KB$ has a unit clause $p$ (or $\neg p$), then replacing the value of variable $p$ with $true$ (correspondingly $false$) in every clause in which it appears leaves that clause in horn form, and therefore the entire KB remains in horn form.

Thus, calling procedure DPLL on a Horn KB will keep propagating unit clauses, until we are either left with an unsatisfiable KB or a horn KB without unit clauses, in which case the KB is satisfiable.

A more delicate analysis is needed to show that this procedure can be implemented in linear time. At the beginning, we create for each variable $p$ a list of pointers to each clause that $p$ appears in. When we do unit propagation for $p$, we go down this list and visit each clause in it, decrementing the count of variables in that clause by one. If that brings the number of variables in the clause down to 1, we push it onto the stack of unit clauses to
process. Each list is thus iterated over at most once (when it first appears in a unit clause), and the length of each list is equal to the number of clauses the corresponding variable appears in. The total number of iterations made by the algorithm is therefore equal to the number of occurrences of every variable in all the clauses, which is equal to the size of the KB. Since all the other operations take a constant number of steps per variable, the entire procedure is linear in the size of the KB.