1.Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will be never be lower than the utility obtained playing against an optimal MIN.

Let a MIN node be terminal nodes. If MIN plays suboptimal, then the value of the node is greater than or equal to the value that I would be if MIN is optimal. Then the MIN node's value must be increased because that value that will be chosen is greater than the value that will be chosen when suboptimal and by doing this all the way up to the root, MAX node can be increased because the MIN nodes just right below MAX node are increased.

2. Prove that alpha-beta pruning takes time O(b^(m/2)) with optimal move ordering, where m is the maximum depth of the game tree and b is the branch factor.

Alpha beta pruning can help minimax algorithm being faster when ordering of nodes are optimal. It reduces the need to search large portions of the tree. The algorithm maintains two values, alpha and beta.
With a branch factor b and depth m, the maximum number of leaf node is O(b*b*b*b…),which is minimax search. But when the ordering move is optimal, for only even depth, it searches only the children of first node in the last level and the other nodes will not be visited and pruned because we don't need to visit once we have any circumstances that the other nodes except first node doesn't need to be searched. For example, from MIN level on the first branching factor, assume that we find that 3 and 2 is the minimum for first branch and the next branch respectively. When we also assume that the tree has an optimal ordering for the best result, the minimum value of the branch would be set to the first child. So we can say, since on the first branch, the minimum value is 3 and min value of next branch is 2 so no need to visit the other children because something less than 3 cannot change their parents value, which is MAX. Therefore, it will be only visit the first child and takes time of b*1*b*1*b*1…=>O(b^(m/2))

3. Programming assignment:
Create a program that can solve a cryptarithmetic problem. Each letter must be assigned a unique digit from zero to nine, and no term is allowed to have leading zeros. (For example, E and F cannot be zero in the following:)

```
E I G H T
- F O U R
-----------
```

F O U R

The program should allow a user to enter the subtrahend, the minuend, and the difference. For simplicity, you may limit the subtrahend term to five letters and the minuend and difference both to four letters. But the user can input any combination of letters (e.g., SIXTY - NINE = FONE). The program needs to output all valid assignments of the variables and digits, or output that no valid assignment exists.

Use the Most Constrained Variable, Most Constraining Variable, and Least Constraining Value heuristics in backtracking to solve the problem, and report the number of steps (i.e., the number of assignments) to solve the above (EIGHT-FOUR=FOUR) problem. Note for any other combinations of input letters from users, your program also needs to either find all valid assignments, or output no valid assignment exists.

(Extra 10% credits) Add "forward-checking" and/or "arc consistency" into above heuristics, and report the number of steps to solve the same problem. If you can show the improvement in the number of steps with either one or both heuristics, you will receive the extra credits.