~ means the complement of the value.
v = or

1. To find the second satisfying assignment, take the output that your algorithm provides and have each bit changed to its complement pass it to the oracle until it returns true then return the second satisfying assignment.

   Do not return(b1,...,bn), instead use it as input that we will iterate through:
   For index i = 0 to n:
   If W(~bi, bi+1,...,bn) == True: #oracle returns true
   Return (~bi, bi+1,...,bn)
   Else: revert the complemented value back to it's original

4.
   a. To show CNF2 is in P, we create a decider M for CNF2:
      M = " On input FORMULA:
      1. If the first clause is in the form of any combination of variables and literals (i.e. (y or A)) and another clause is the complement of the first clause (i.e. (~y or A)) , REJECT
      2. Else if, if the first clause is in the form of (x or A):
         a. If the complement of x doesn't appear in the rest of the FORMULA: remove all (x or B) that appear, ACCEPT
      3. Else if, if a clause in the form of (x or A) and there exists a (~x v B), remove both clauses, and replace with (A or B)
      4. Return to 1

   b. To show CNF3 is in NP: give CNF3 a FORMULA, count each variable in the FORMULA, and if any variable occurs more than 3 times, replace until variables that occur more than 3 times occurs at most 3 times. If this can not be done, reject; else, accept

      Reduce 3SAT <=P CNF3:
      Given a FORMULA:
      1. Check if a variable occurs more than 3 times in the FORMULA.
         a. If so, create an equivalent amount of variables and the complement of that variable
         b. You then "or" every variable with a complement of a different variable that we created.
         c. You then and each clause to each other
         d. You then replace the variable that occurred >3 times in the FORMULA with the variables and attached the and clause to the FORMULA

      In example, given the FORMULA = (x v a v b) and (x v c v d) and (x v e v f) and (x v g v h). I create a1, …, a4. I then generate the clauses (a1 v ~a4) and (a2 v ~a3) and (a3 v ~a2) and (a4 v ~a1). I replace the x's with a1, … a4, and I attach the newly generated classes:

(a1 v a v b) and (a2 v c v d) and (a3 v e v f) and (a4 v g v h) and (a1 v ~a4) and (a2 v ~a3) and (a3 v ~a2) and (a4 v ~a1)

Since 3SAT is reducible to CNF3 and CNF3 is in NP, CNF3 is in NP Complete.

6. To prove SCHEDULE is in NP, we can assume a schedule has r slots and that it can be verified in polynomial time that students are not taking more than one exam in the same slot. This can be done by setting the slot amount r to a specific number and iterating through each index of S and seeing if that index value is a subset of F, if so, and if the size of the index value of S is at most r.

We can then prove that SCHEDULE is NP-hard through showing that 3color <=P SCHEDULE. Given a graph G <Nodes N, Edges E>, we create an instance of <F,S,h> where the nodes are equal to the exams in F, and the edges are the instances in S. We then assign each node a slot from 0 to r, where no adjacent node is the same slot. Thus when a value in S is selected, the values in that value, displayed on the graph, will not be in the same slot. Thus F = N, S = E, and h = 3.

So, given a 3-coloring for G, we can get a schedule for <V,E, 3> by assigning finals that correlate to the nodes in the graph. Then, associate a slot to each node, but adjacent nodes do not have the same slot.
So with <V,E,3>, a 3-coloring for the original graph by correlating a color to the slot.

Since SCHEDULE is in NP and can be reduced to 3color, SCHEDULE is NP Complete.

To show SCHEDULE <=P 3color, we generate a set of exams that equate to the number of nodes, and the set of S will be the edges between each node.
There must exist a graph that can accept a SCHEDULE and not have two nodes that share the same slot.