1. To prove that integer programming is NP-hard, we need to prove that SAT is in NP and SAT can reduce to integer programming.
   Prove SAT belongs to NP. Once we have a solution to the SAT problem, we can verify it in polynomial time. Assume we have m clauses(equations) and n literals(variables) , we can go through every literal and clause to verify the correctness, the time is O(mn) which is within the polynomial bound.

   Prove SAT reduces to integer programming. Any SAT can reduce to 3SAT, so we use 3SAT instead. Assume we are given a 3 SAT with n variables x_1, x_2, x_3, … , x_n. We can construct a integer linear programming with this 3 SAT: for each x_i, there is a corresponding z_i, representing one variable in the integer linear programming, for each z_i, we force it be to either 0 or 1 (false or true), and for each clause, we can transform it into a integer linear programming constraint, for example , (x_1 U negative x_2 U x_5) is true when z_1 + (1-z_2) + z_5 > 0 with z_i being 0 or 1. So every 3-SAT can be constructed to an integer linear programming problem. And if variables in integer linear programming are greater than 1, we can use multiple x_i and z_i to represent 1 variable in the integer linear programming. So any integer linear programming has an integer solution if the corresponding 3-SAT has a solution.

   We proved that SAT is in NP, and 3-SAT which is a reduction from SAT problem can reduce to integer linear programming problem (sat <= 3-SAT <= ILP). so ILP is in NP hard.

2. We can prove half-SAT is in NP-complete by reducing SAT to HALF-SAT.
   HALF-SAT is in NP, if we are given all the values to the variables, we can check if the HALF-SAT is satisfiable or not in polynomial time by checking whether there are half true and half false variables and whether they are satisfiable inside each clause.

   Assume F has variables x_1, x_2, x_3,..., x_n, we construct a new F' with variable from F, and y_1, y_2, … y_n, where x_1 = ~y_1, x_2 = ~y_2. And for every clause in F, we add one more conjunction such as if one clause in F is (x_1 U x_2) we construct it into (x_1 U x_2) ^ (~y_1 U ~y_2). y_1 takes the opposite value of x_1.

   Claim: SAT with exactly half variables are true is satisfiable if and only if HALF-SAT is satisfiable.
   Forward proof, if SAT with half true variables is satisfiable meaning that some of the y_i are false and some of the y_i are true. By construction, if half of the variables from F are true, and F is satisfiable, then half of the y_i must be true, then the newly added clauses must be all true so the HALF-SAT is satisfiable.
   Backward proof, HALF-SAT belongs to SAT, so if HALF-SAT is satisfiable, then SAT must be satisfiable.

3. First we prove that the class schedule problem is in NP. If we are given the class schedule with K classes, we can check whether they are overlapping or not in polynomial time O(n) time by going through these K courses and checking for overlapping.

   Prove NPC by reduction from the Independent Set.
   We construct a graph, each vertex represents a class schedule. If we can find an Independent set that has at least K elements in it, we construct a class schedule that has at least K courses without overlapping.
   Claim: Independent set with at least K vertices is satisfiable if and only if we can find a class schedule with at least K courses without overlapping.
   Forward proof: If we can find an independent set with at least K vertices in the set, each vertex in the set can be assigned a class that is not overlapping with any other class. Then we can find a class schedule with at least K courses, we can take at least K courses.
   Backward proof, if we are given a set of classes, and we can take at least K courses, then we can construct these courses as vertices in an independent set.

   The problem is NP and it can be reduced from the independent set problem which is in NPC. So the problem is in NPC.

4. Algorithm: we choose vertices with two colors, add them to the vertex cover, we should have N/2 vertices. According to the fact that planar graph is 4-colorable, meaning that the optimal solution will only have N/4 vertices in the vertex cover, so the alpha is 2.

5. A. Connected graph is a graph that you can reach any vertex in the graph from any vertex in the graph, So one depth first search can visit all the vertices. Now we add all the non-leaf vertices which are all the vertices except the vertex in the last level of dfs. The set of vertices is all the vertices except the root and the last vertex, but they are connected to the vertex in the set, so the set is a vertex cover.

   B. Assume in DFS there are N nodes in the vertex cover, since we choose every endpoint in every edge, there are N/2 edges. And we can choose one endpoint for every edge since the other endpoint is clearly connected to it. So the minimum vertex cover number would be N/2. Alpha value is 2, this is a 2-approximation.