Problem 1

Let n courses and r conflicting pairs be a directed graph, G, of n vertices and r edges.

1 = blue and -1 = white

```
Def bipartite(G):
    INITIALIZE colors  ← a dictionary to store colors of vertices
    INITIALIZE source node ← this will be cs 100 with no pre req
    INITIALIZE visited ← a dictionary to store visited status
    INITIALIZE queue q
    INITIALIZE fall ← list to contain fall semesters
    INITIALIZE spring ← list to contain spring semesters

    FOR every vertex, u, in G:
        colors[u] = 0  ← 0 means the node is uncolored
        visited[u] = -1  ← -1 means not visited
        IF u == “cs100”:
            Source node = u
            colors[u] = 1
    q.enqueue(source node)
    WHILE(q not empty):
        u = q.dequeue()
        IF visited[u] == -1:
            Visited[u] = 1
            FOR every edge, (u,v) in G:
                IF colors[u] == colors[v]:
                    return False
                IF visited[v] == -1:
```

```
                q.enqueue(v)
                colors[v] = -1*colors[u]
FOR every node, u, in G:
    IF colors[u] == 1:
        fall.append(u)
    ELSE:
        spring.append(u)
return (fall, spring)
```

Time Complexity:

This algorithm basically uses breadth first search at ground level, which is in O(V+E) if graph is in adjacency list form. Since V=n and E=r, this algorithm will take O(n+r) time to run