Q5.5) The problem of determining whether an arbitrary graph has chromatic number *k*, where *k*≥3 is a very hard problem (this problems falls under *NP−Complete* and means there does not currently exists an efficient algorithm for solving the problem). However, determining whether an arbitrary graph has chromatic number 2 is much easier (there does exist efficient algorithms to do so, we say that they fall under P). Given a graph *G* on *n* vertices, create an algorithm that will return TRUE if $\chi(G) = 2$ and FALSE if $\chi(G) \neq 2$. Clearly explain how your algorithm works, why it guarantees the correct output, and determine the running time of your algorithm.

Graph Coloring is a process of assigning colors to the vertices of a graph. It ensures that no two adjacent vertices of the graph are colored with the same color. The problem of determining whether an arbitrary graph has chromatic number k, where k≥3 is a very hard problem (this problems falls under NP–Complete and means there does not currently exists an efficient algorithm for solving the problem).
However, we can still determine whether an arbitrary graph has chromatic number 2 using the following algorithm/pseudocode as say that the problem falls under P.

Algorithm/Pseudocode
We will use the adjacency list representation for the graph. In this representation, each vertex v is associated with a linked list Adj[v] that contains all the neighbors of v. We will use an array Color, where Color[v] is the color of vertex v. Initially Color[v] = null for all vertices v.

```
function 2-COLORABLE():
//BFS traversal while there are unvisited vertices
    for v in V do
        // vertex v has not been visited
        if Color[v] = null do
            // now do BFS at v
            initialize an empty queue Q
            Color[v] ← Red
            Enqueue(Q,v)
            while Q is not empty do
```

```
                u ← Dequeue(Q)
                for each w in Adj[u] do
                    if Color[w] = null do
                        if Color[u] = Red
                            Color[w] ← Blue
                        else
                            Color[w] ← Red
                        end if
                        Enqueue(Q,w)
                    else if Color[w] = Color[u]
                        print FALSE
                    end if
                end for
            end while
        end if
    end for
    print TRUE
end function
```

Explanation - Working of Algorithm

Assumption for Color 1 is Red and Color 2 is Blue so that if the graph is 2-colorable, all the vertices will assigned either Red or Blue color and no adjacent vertices which share an edge will have same color.
The idea of the algorithm is as follows. We start with an arbitrary vertex s, color it Red, and run BFS from there in order to color all vertices in the same connected component with s. It is necessary that all neighbors of s are colored Blue, and all neighbors of these neighbors are colored Red, etc. If at some point we detect a vertex that is colored by two different colors, then the graph is not 2-colorable. After doing BFS at s, if there are uncolored vertices in G we choose one of them and repeat the same process.

Runtime Analysis of the Algorithm

The total time for all "for" loops that are executed is O(|E|), because each loop corresponds to an edge of G. Thus, the total time complexity is **O(|V | + |E|)**.

Proof of Correctness

We will check the correctness of the algorithm for two parts.

Part I
First we show that if the algorithm outputs FALSE, then the graph is not 2-colorable. Because every two BFS trees are not connected, the coloring for each tree is independent of each other. Suppose, that the algorithm outputs FALSE during the BFS tree T that starts at a vertex v, we will show that this tree is not 2-colorable, and hence the whole graph is also not 2-colorable. Suppose for a contradiction that T is 2-colorable, then any 2-coloring of T is completely determined by the color of v, and we can assume without loss of generality that v has color Red. We prove the following Claim using Induction on the distance from a vertex u to v.
Claim: Let u be a vertex in T. Any color that u gets from the program is the color that u must get in a 2-coloring of T where v is colored Red.
Proof of Claim
For the base case, the distance from u to v is 0, i.e., u is v itself. The Claim holds in this case because v is colored Red. For the induction step, suppose that the distance from u to v is d+ 1 for some d ≥ 0. Therefore there is a neighbor u′ of u such that the distance from u′ to v is d. By the induction hypothesis the color assigned to u′ by the program is the color u′ must get in a 2-coloring of T where v is Red. Given the color of u′, u has only once choice and it’s clear that this is the choice chosen by the program. This completes the induction step, and hence the proof of the Claim. We reach a contradiction because some vertex u in T is colored both Red and Blue by the program.

Part II
Now we show that if the program outputs TRUE, then indeed the graph is 2-colorable. This is so because the program actually provides a 2-coloring of the graph. This is because the BFS algorithm colors every vertex of the graph, and for any edge (u, v) the two endpoints must have different colors, for otherwise the program would outputs FALSE.