```
4:    if s.visited == false then
5:        BFS(s)
6:    end if
7: end forreturn consistent
8:
9: BFS(v)
10: v.species = A
11: v.visited = true
12: L.enqueue(v)
13: while L.size ≠ 0 do
14:    c = L.dequeue()
15:    for each edge e connecting node c to some node d where e.traveled == false do
16:        if d.visited == true then
17:            if e.type == same && c.species ≠ d.species then return inconsistent
18:            else if e.type == different && c.species == d.species then return inconsistent
19:            end if
20:        else
21:            if e.type = same then
22:                d.species = c.species
23:            else if c.species == A then
24:                d.species = B
25:            else
26:                d.species = A
27:            end if
28:            d.visited = true
29:            L.enqueue(d)
30:        end if
31:        e.traveled = true
32:    end for
33: end while
34:
```

**Analysis**

*Proof of Run Time.* For the runtime of the algorithm, the for loop on Line 2 will execute n times, checking each vertex once to see if it is visited, and if so running a Breadth First Search using it as the root. This allows all nodes to be searched for inconsistencies even if there are separate groups of nodes in the graph. As the check is constant time, this adds $n$ operations to our run time.

For the runtime of the Breadth First Search, the while loop (Line 12) will only repeat while the queue is not empty, and it removes a node from the queue each iteration. Multiple nodes can potentially be added to the queue within one iteration but only unvisited nodes can be added and they are set to visited when they are added, ensuring a node can never be added twice. As a node can not be removed from the queue twice, the maximum number of times it can execute is equal to the number of nodes in the connected group. With this repeated for each connected group, the number of times this loop can execute is equal to the number of nodes in the graph. Not including the for loop, this adds 3n operations to our runtime.

The for loop (Line 4) does not repeat the same number of times for each iteration of the while loop. Instead, it is limited to only executing on untraveled edges, after which the edge is marked traveled before the next iteration of the for loop. This limits the for loop to executing a maximum of m times over the entire algorithm, where m is the number of edges in the graph. In the worst case where $d$ is not visited, the edge is of type "different" c.species is B, a single for loop will run 6 operations, giving us a total run time of 7m for the for loop.

This gives us a worst case of $n + 3n + 7m = 4n + 7m$ operations which is in $O(n + m)$ time. □