1. We will start at any vertex, and save this starting vertex in a variable. Then follow the outgoing edge to the next vertex, and its outgoing edge, and so on; while keeping track of the vertices visited. If we found a vertex with zero outgoing edge or more than 1 outgoing edges; or if we arrived at a visited vertex other than the starting vertex, the graph is not a circle. If we arrived at the starting vertex, and the number of vertices visited is equal to the total number of vertices in the graph, the graph is a circle.
Pseudocode:

```
bool is_circle(graph) {
    int total_vertices = total number of vertices in graph;
    Vertex start = any vertex in graph;
    Vertex current = start;
    set<Vertex> visited;
    do {
        if (current.number_of_outgoing_edges() != 1 or visited.contains(current)) {
            return false;
        }
        visited.add(current);
        current = vertex that the outgoing edge lead to;
    } while (current != start);
    return (visited.size() == total_vertices);
}
```

2. The best case for this algorithm will be $\Omega(1)$ time. If the first vertex does not satisfy the definition of a circle and the number of outgoing edges is not equal to 1, we can directly conclude that the graph is not a circle.

3. Assume the number of vertices in the graph is n.
The worst case will need $\Theta(n)$ time. If the graph is actually a circle, we will have to loop through every vertex in the graph and come back to the starting vertex and conclude that the graph is a circle.