Algorithm 1: BFS (Breadth-First Search) for Cycle Detection

Pseudocode:

```
ALGORITHM bfs_detect_cycle(M, V)
//Breadth-First Search traversal of matrix
//Input: List vertices = [vertex], matrix Matrix = {vertices, edges}
//Output: List Cycles filled with lists of vertices Cycle in order of each cycle
//For asymptotic analysis comments, will refer to LENGTH(vertices) as V, and LENGTH(edges) as E

num_vertices ← LENGTH(vertices)
visited ← [FALSE * num_vertices]
queue ← [EMPTY QUEUE]
cycles ← [EMPTY LIST]

FOR start IN range(num_vertices) DO                                   // O(V)
        IF visited[start] == FALSE DO
                path ← [EMPTY LIST]
                APPEND (start, [start]) to queue
                WHILE queue is not empty DO
                        current, path == POP[queue]
                        visited[current] = TRUE
```

```
        FOR neighbor IN range(num_vertices) DO                    // O(V)
            IF matrix[current][neighbor] exists DO
                IF neighbor IN path DO                                              //O(E)
                    cycle ← path[index of neighbor in path to end] + [neighbor]     //O(LENGTH(cycle))
                    APPEND cycle TO cycles                                          //since LENGTH(cycle) == V in
                ELSE IF visited[neighbor] == FALSE DO                               //worst case, O(V)
                    APPEND (neighbor, path + [neighbor]) TO queue

IF LENGTH(cycles) > 0 DO
    FOR cycle in cycles DO
        PRINT cycle
```

Time Complexity:

1. **Traversal of Matrix**: O(V^2) (nested loop for adjacency matrix traversal).
2. **Cycle Detection**: O(E·V) (worst case where all edges are processed, and path comparison takes O(V).
3. **Total Complexity**: O(V^2+E·V).

Algorithm 2: DFS (Depth-First Search) for Cycle DetectionPseudocode:

```
ALGORITHM dfs_detect_cycle(M, V)
// Depth-First Search traversal of matrix
// Input: List vertices = [vertex], matrix Matrix = {vertices, edges}
// Output: List Cycles filled with lists of vertices Cycle in order of each cycle
// For asymptotic analysis comments, refer to LENGTH(vertices) as V, and LENGTH(edges) as E
num_vertices ← LENGTH(vertices)
visited ← [FALSE * num_vertices]
cycles ← [EMPTY LIST]
unique_cycles ← [EMPTY LIST]

FUNCTION dfs(node, stack, visited)
    visited[node] = TRUE
    APPEND node TO stack

    FOR neighbor IN range(num_vertices) DO
        IF matrix[node][neighbor] exists DO
            IF neighbor IN stack DO
                cycle_start ← INDEX(neighbor IN stack)
```

```
            cycle ← stack[cycle_start:] + [neighbor]
            sorted_cycle ← SORT(cycle)
            IF sorted_cycle NOT IN unique_cycles DO
                APPEND sorted_cycle TO unique_cycles
                APPEND [vertices[i] FOR i IN cycle] TO cycles
        ELSE IF visited[neighbor] == FALSE DO
            CALL dfs(neighbor, stack, visited)

    POP stack
END FUNCTION

FOR node IN range(num_vertices) DO
    IF visited[node] == FALSE DO
        CALL dfs(node, [], visited)

IF LENGTH(cycles) > 0 DO
    FOR cycle IN cycles DO
        PRINT cycle
ELSE
    PRINT "No Cycles Detected"
```

Time Complexity:

1. **Traversal of Matrix**: $O(V^2)$ (nested loop for adjacency matrix traversal).
2. **Cycle Detection**: $O(E \cdot V)$ (worst case where all edges are processed, and path comparison takes $O(V)$.
3. **Total Complexity**: $O(V^2+E \cdot V)$.