Depth-First Search (DFS) Algorithms with Time Complexity Analysis

Recursive DFS Algorithm

Algorithm 1 Recursive DFS(G, u)

```
Input: Graph G = (V, E), starting vertex u
Output: Visited vertices in depth-first order
function DFS-Recursive(G, u, visited)
    visited[u] ← TRUE                              ▷ Mark vertex u as visited
    for all v ∈ G.Adj[u] do                        ▷ Iterate over all neighbors of u
        if visited[v] = FALSE then                 ▷ If v is not visited
            DFS-Recursive(G, v, visited)           ▷ Recursive call for v
        end if
    end for
end function
```

Time Complexity Analysis of Recursive DFS

- **Initialization**: Marking all vertices as unvisited takes $\mathcal{O}(V)$.
- **Traversal**:
  - Each vertex is visited exactly once: $\mathcal{O}(V)$.
  - Each edge is explored exactly once in the adjacency list: $\mathcal{O}(E)$.
- **Overall Time Complexity**: $\mathcal{O}(V + E)$.

—

Iterative DFS Algorithm

Algorithm 2 Iterative DFS(G, s)

```
Input: Graph G = (V, E), starting vertex s
Output: Visited vertices in depth-first order
function DFS-Iterative(G, s)
    Initialize:
    visited[v] ← FALSE for all v ∈ V               ▷ Mark all vertices as unvisited
    stack ← Empty Stack
    Push(stack, s)                                 ▷ Push the starting vertex onto the stack
    while stack ≠ Empty do                         ▷ Process stack until it is empty
        u ← Pop(stack)                             ▷ Remove and process the top vertex
        if visited[u] = FALSE then                 ▷ If u is not visited
            visited[u] ← TRUE                      ▷ Mark u as visited
            for all v ∈ G.Adj[u] do                ▷ Iterate over all neighbors of u
                if visited[v] = FALSE then         ▷ If v is not visited
                    Push(stack, v)                 ▷ Add v to the stack
                end if
            end for
        end if
    end while
end function
```

Time Complexity Analysis of Iterative DFS

- **Initialization**: Marking all vertices as unvisited takes $\mathcal{O}(V)$.

- **Traversal**:
  - Each vertex is pushed and popped from the stack exactly once: $\mathcal{O}(V)$.
  - Each edge is explored exactly once in the adjacency list: $\mathcal{O}(E)$.
- **Overall Time Complexity**: $\mathcal{O}(V + E)$.

—

Key Differences Between Recursive and Iterative DFS

- **Recursive DFS**:
  - Uses function call stack for maintaining the DFS traversal order.
  - Simpler to implement but limited by system recursion depth.
- **Iterative DFS**:
  - Uses an explicit stack to simulate the recursion.
  - More memory-efficient for very deep graphs, as it avoids stack overflow.