This C code is a combination of two algorithms: Selection Sort and Parallel Depth-First Search (DFS) using the Message Passing Interface (MPI) for parallel processing. Let's break down the code:

Selection Sort Section:

1. **Function Definitions:**
   - `swap`: A simple function to swap two integer values.
   - `selection Sort`: Implements the selection sort algorithm to sort an array in ascending order.
2. **Main Function (Selection Sort):**
   - Generates an array (`arr`) of size `ARRAY_SIZE` filled with random values.
   - Prints the unsorted array.
   - Measures the time it takes to sort the array using the `selection Sort` function.
   - Prints the sorted array.
   - Displays the time taken for the selection sort operation.

Parallel DFS Section:

1. **Graph Representation:**
   - An adjacency matrix `adjacency_matrix` representing a graph with `GRAPH_SIZE` vertices.

- `visited` array to keep track of visited vertices during DFS.

2. **DFS Function:**
   - `DFS`: Performs depth-first search on the graph starting from a given vertex. It recursively explores the adjacent vertices and marks them as visited.
3. **Main Function (Parallel DFS using MPI):**
   - Initializes MPI and retrieves the rank of the current process.
   - Synchronizes processes before starting DFS using `MPI_Barrier`.
   - Process with rank 0 starts a timer before DFS.
   - Calls DFS from a specified starting vertex (`start_vertex`).
   - Process with rank 0 stops the timer after DFS completion.
   - Displays the time taken for the parallel DFS operation.

Explanation:

- The code first performs a selection sort on a randomly generated array and prints the unsorted and sorted arrays along with the time taken for the sorting operation.

- Then, it performs a parallel DFS on a graph represented by an adjacency matrix. The DFS is executed in parallel using MPI, and the time taken for the DFS operation is displayed by the process with rank 0.

Note:

- The parallelism in DFS is not explicitly utilized in this code, as each process independently executes DFS on the entire graph. For better parallelization, you might want to divide the graph into subgraphs and distribute them among processes.