Lab:

Assignment: Implement QuickSort with GitHub Copilot Assistance

Objective:

Developers will use GitHub Copilot to implement, enhance, test, and explain the QuickSort algorithm in their preferred programming language (C#, Python, Java, JavaScript, etc.). They will also leverage Copilot for debugging, documentation, and generating a simple UI for user interaction.

Instructions:

1. Set Up GitHub Copilot

Ensure GitHub Copilot and GitHub Copilot Chat extensions are installed in your IDE (VS Code, JetBrains, etc.).

Follow the official GitHub Copilot documentation if needed.

2. Implement QuickSort Using Copilot

Use Copilot to generate a QuickSort function in your preferred language.

Modify and enhance the generated implementation.

3. Use GitHub Copilot Chat for Explanation

Ask Copilot Chat to explain the generated QuickSort implementation, including how it works and its key components.

Document the explanation as comments or a Markdown file.

4. Enhance & Optimize the Algorithm

Use Copilot to suggest performance improvements and optimize recursion or memory usage.

Consider implementing both recursive and iterative versions and compare them.

5. Compare QuickSort with Other Sorting Algorithms

Use Copilot Chat to analyze the time and space complexity of QuickSort.

Compare QuickSort with MergeSort, HeapSort, and built-in sorting methods in your chosen language.

6. Write Unit Tests

Use Copilot to generate unit tests (e.g., using xUnit, NUnit, Pytest, Jest, etc.).

Ensure the tests cover various scenarios, including empty arrays, sorted arrays, duplicates, and large datasets.

7. Create a Simple Web Interface

Use GitHub Copilot’s code edit feature to generate a simple web page (HTML/CSS/JavaScript or a minimal Flask/.NET MVC page).

The page should:

Accept user input for an array of numbers.

Run the QuickSort algorithm.

Display the sorted result on the page.

8. Debug and Refine the Code

Introduce intentional bugs and use Copilot Chat to find and fix them.

Improve error handling and ensure the algorithm handles edge cases properly.

9. Benchmark and Analyze Performance

Implement performance testing using appropriate benchmarking tools (e.g., timeit in Python, Stopwatch in C#).

Compare execution time between QuickSort and built-in sorting functions.

10. Document the Process

Use Copilot to generate code documentation and a Markdown file summarizing:

How Copilot assisted in the development process.

Performance comparisons and key learnings.

Bonus Tasks (Optional)

- ☐ Add more sorting algorithms, and add option for the user to select the sorting algorithm
- ☐ Parallelize QuickSort: Implement a multi-threaded or parallelized version using Copilot's suggestions.
- ☐ Add Visual Representation: Use a simple UI animation (e.g., canvas-based sorting visualization).
- ☐ Generate API Endpoint: Create a basic REST API to expose the QuickSort function.