AI ASSISTED CODING

LAB-12.4

NAME:swaranjith reddy thatipalli

ENROLLNO:2403A52049

BATCH:03

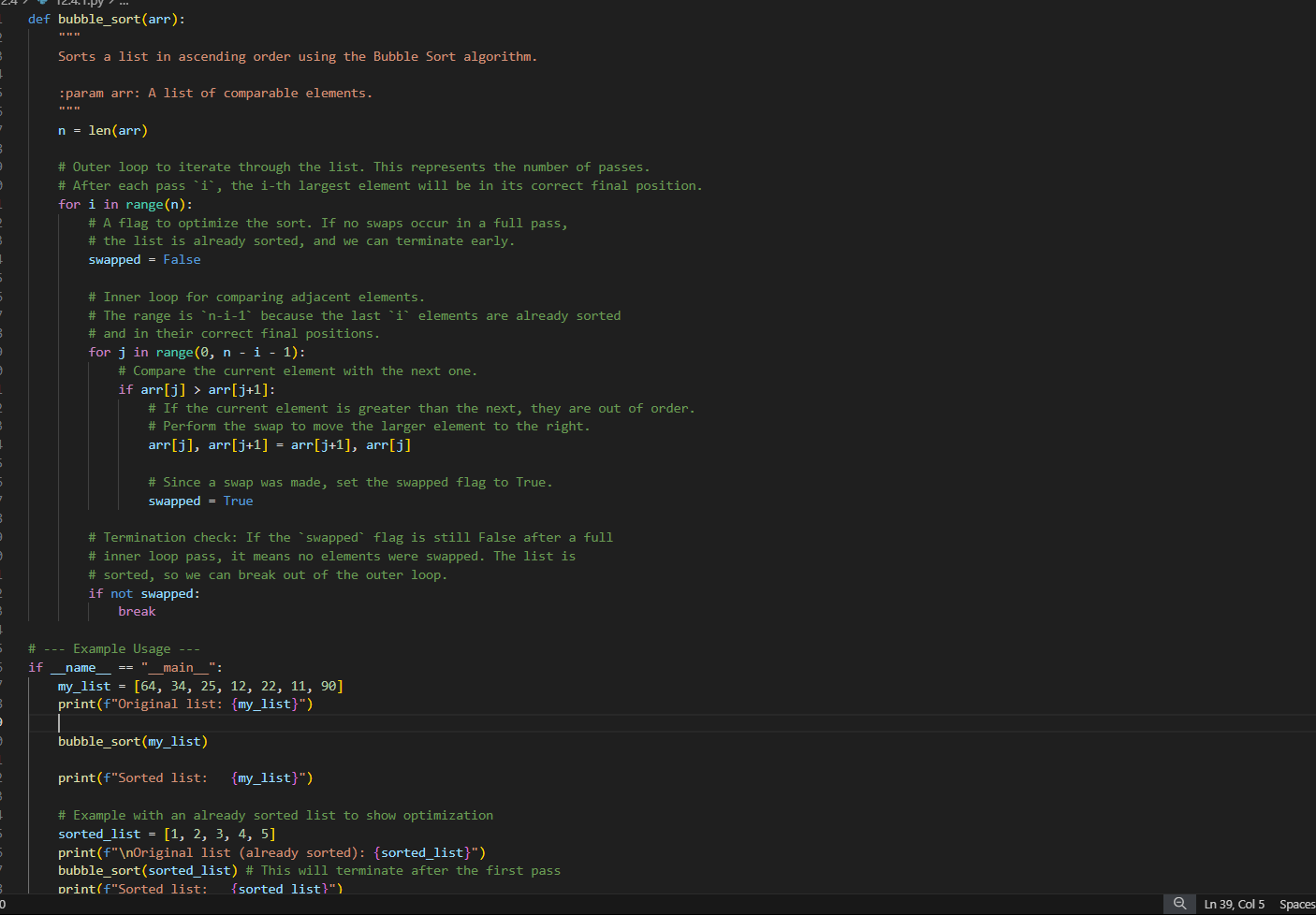
**TASK-01:**

Implementing Bubble Sort with AI Comments.

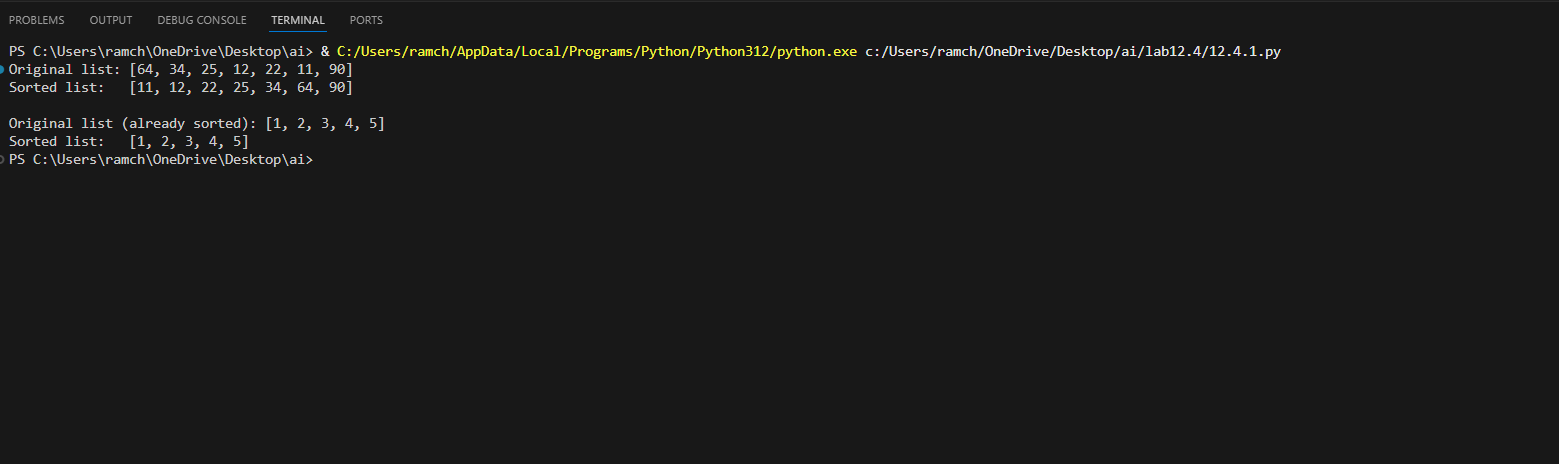
**PROMPT:**

Generate a python code to implement bubble sort.

**CODE:**



**OUTPUT:**



**OBSERVATION:**

**Bubble Sort** repeatedly compares and swaps adjacent elements if they are in the wrong order, moving the largest element to the end in each pass — like bubbles rising to the top.

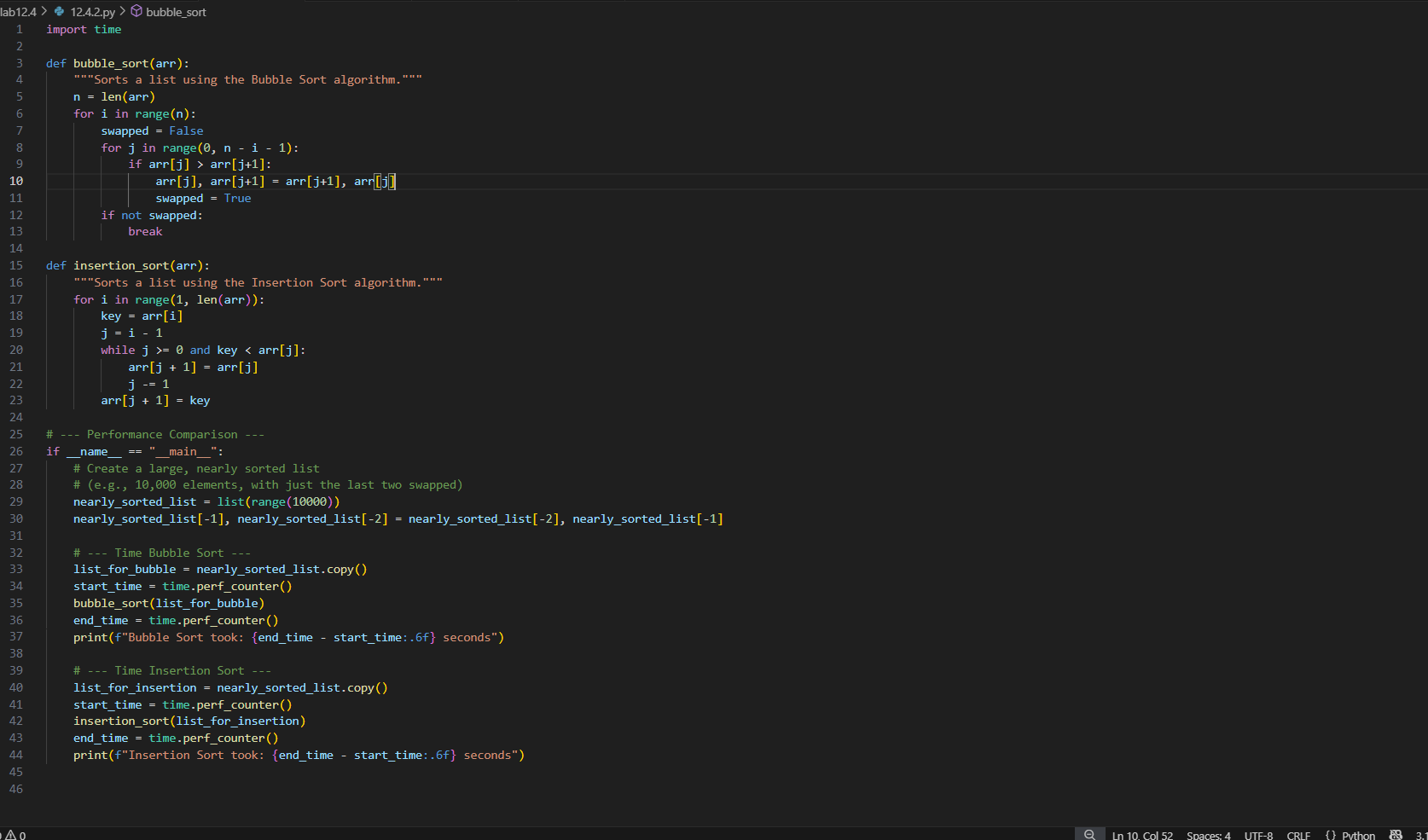
**TASK-02:**

Optimizing Bubble Sort → Insertion Sort

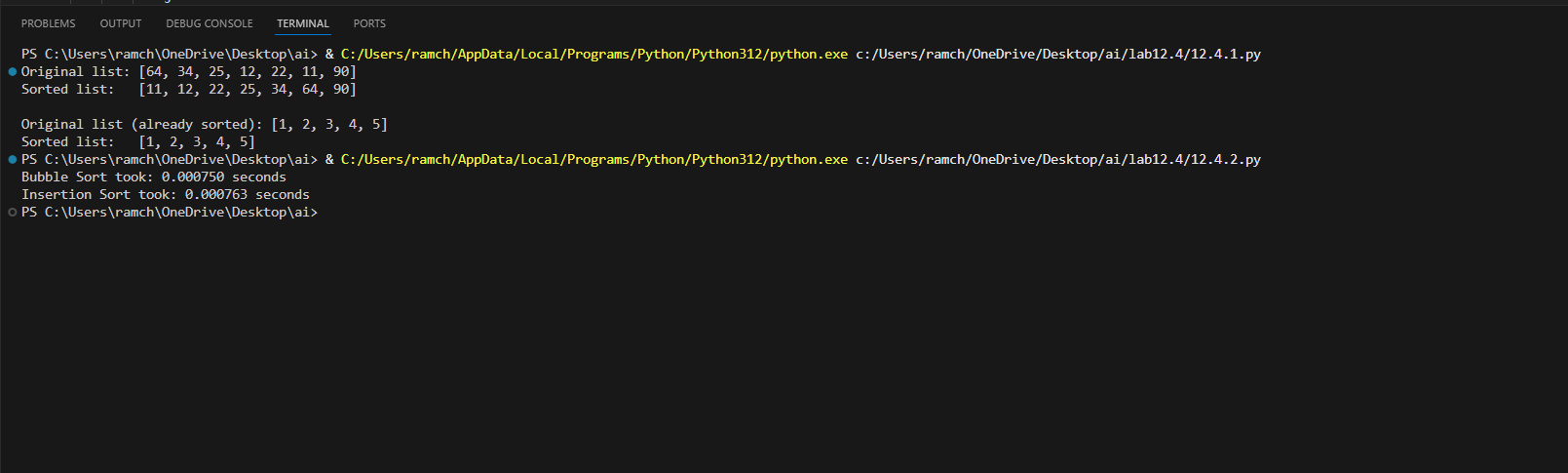
**PROMPT:**

Convert the following bubble sort code to the insertion sort.

**CODE:**



**OUTPUT:**



**OBSERVATION:**

**Optimization from Bubble Sort to Insertion Sort:**  
Instead of repeatedly swapping adjacent elements like in Bubble Sort, **Insertion Sort** shifts elements to insert each item directly into its correct position. This reduces unnecessary swaps and makes it faster, especially for nearly sorted data.

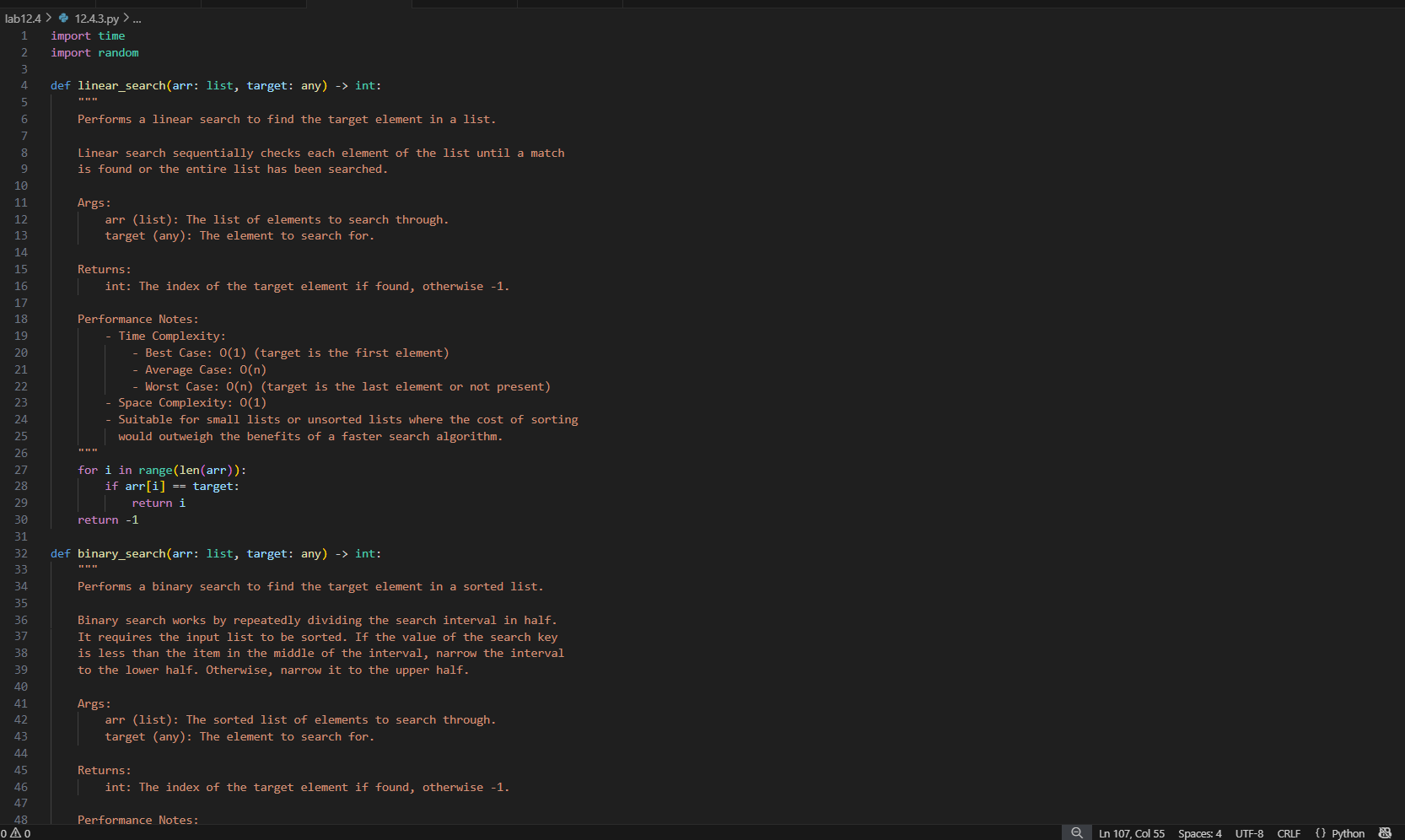
**TASK-03:**

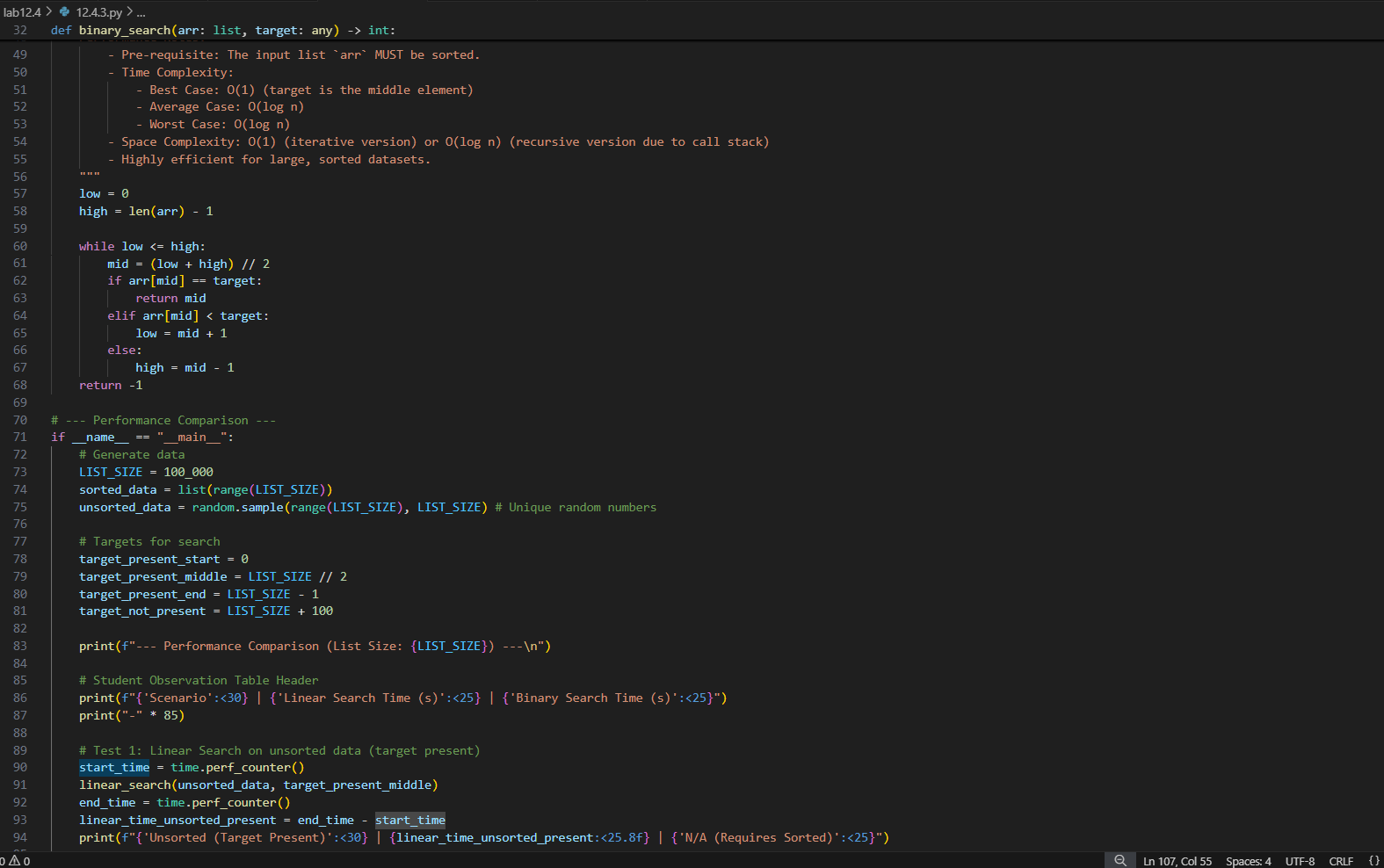
Binary Search vs Linear Search

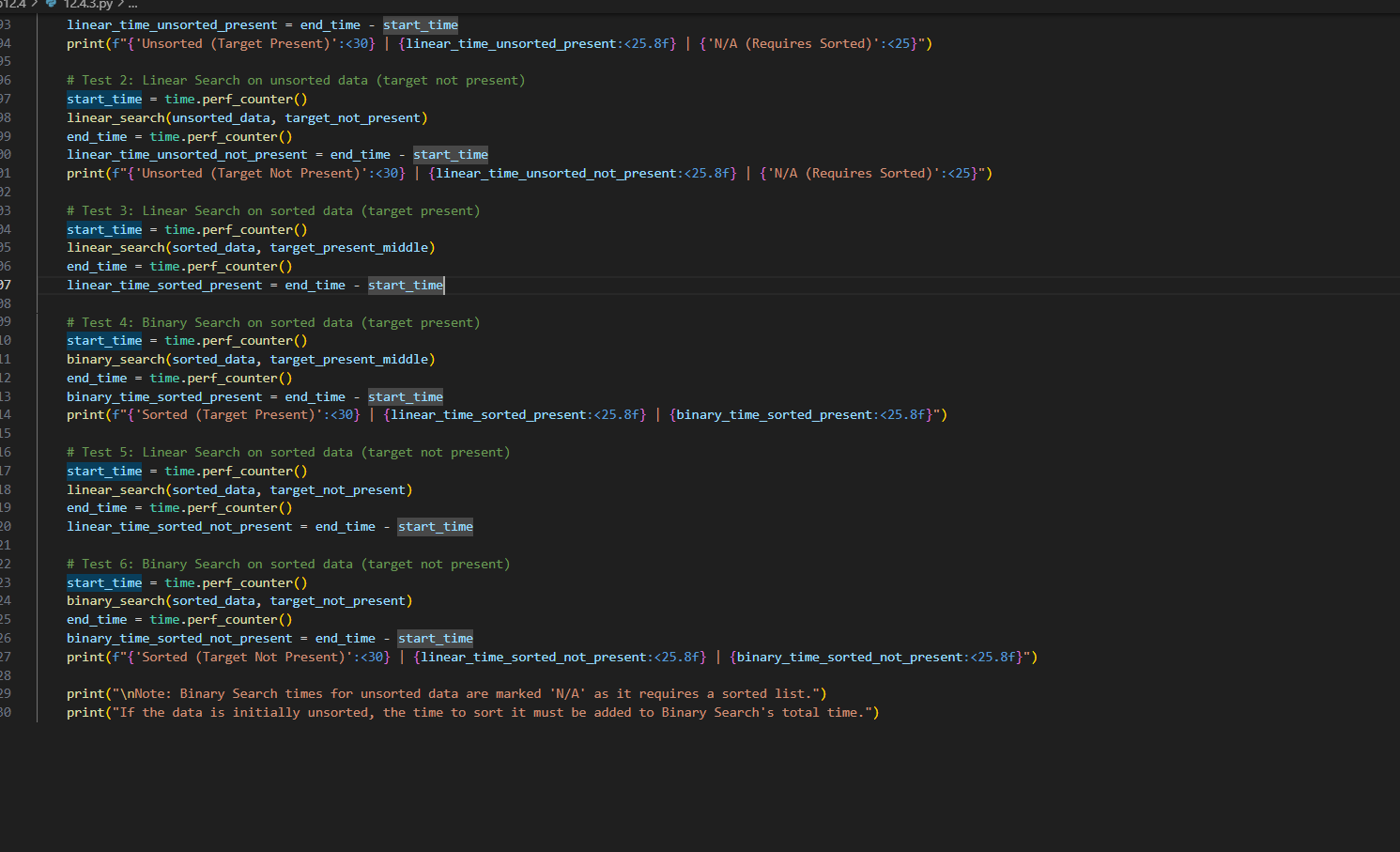
**PROMPT:**

Implement the linear search and binary search with comments.

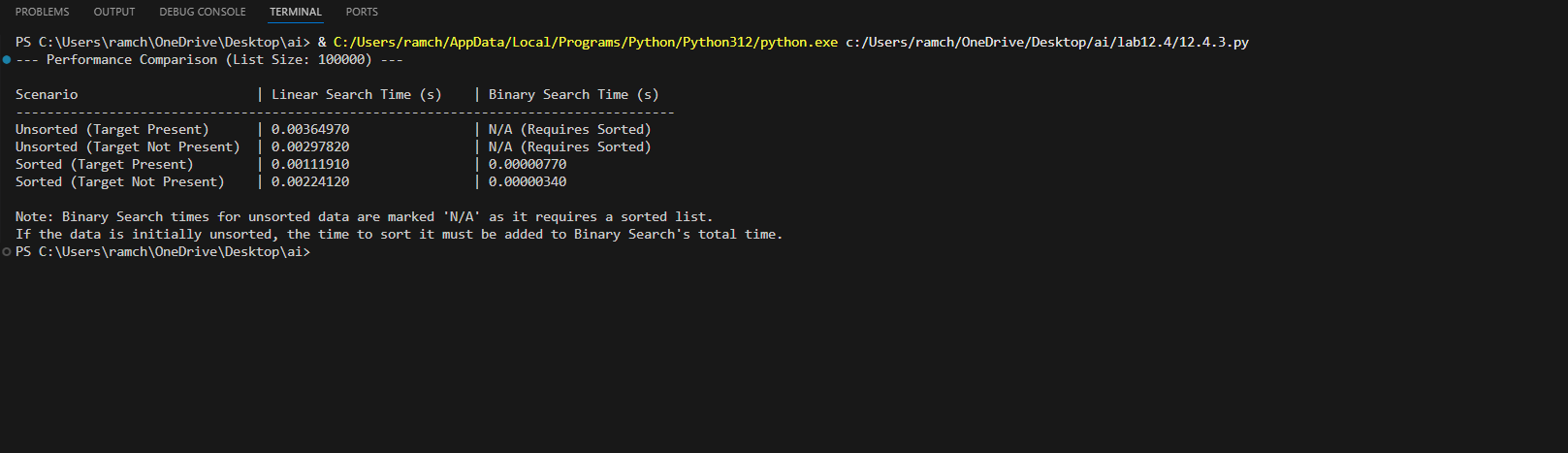
**CODE:**







**OUTPUT:**



**OBSERVATION:**

Linear Search**:** Checks each element one by one until the target is found or the list ends. Works on **unsorted** data but is **slow (O(n))**.

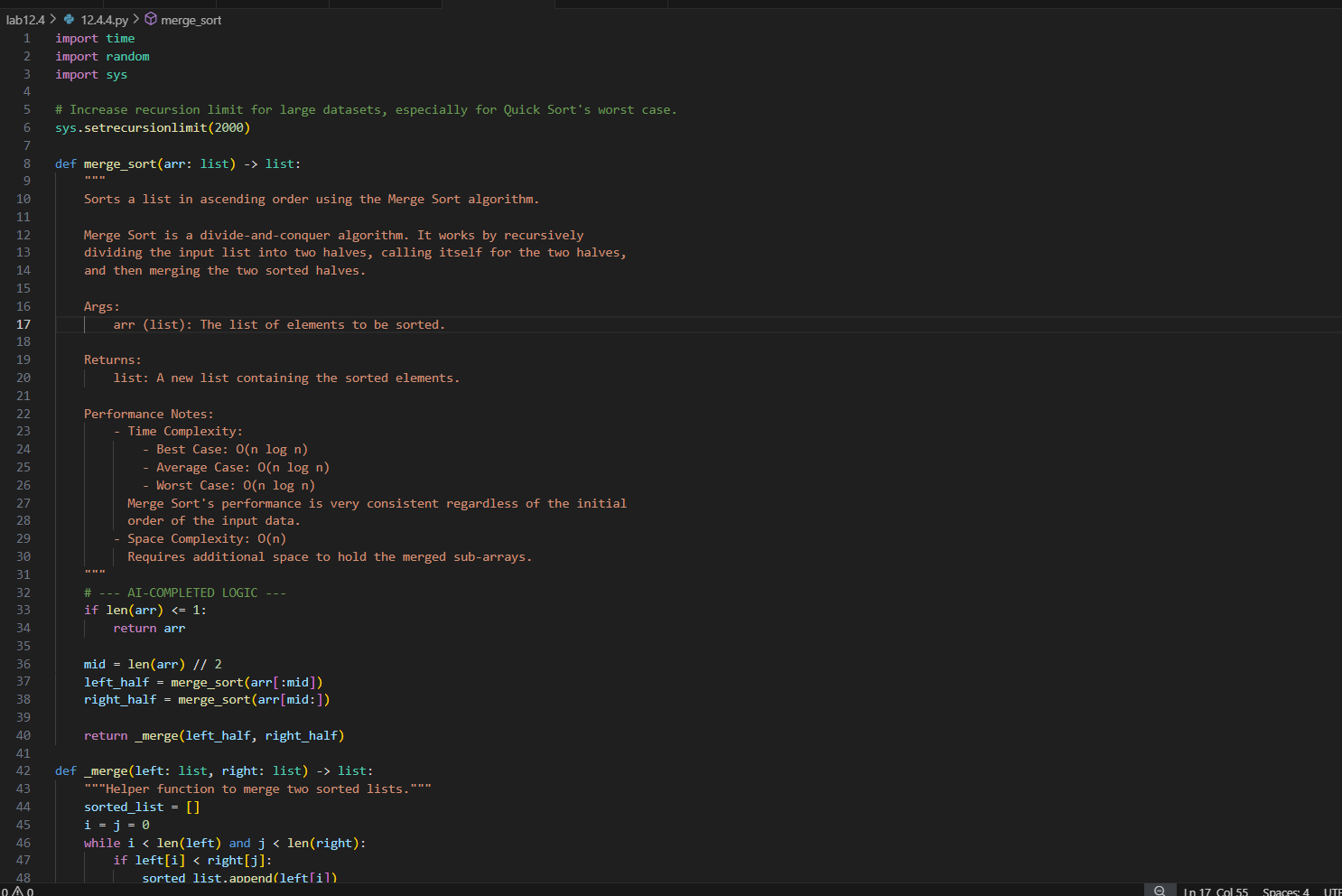
Binary Search**:** Repeatedly divides a **sorted** list in half to find the target. Much **faster (O(log n))**, but requires the data to be sorted.

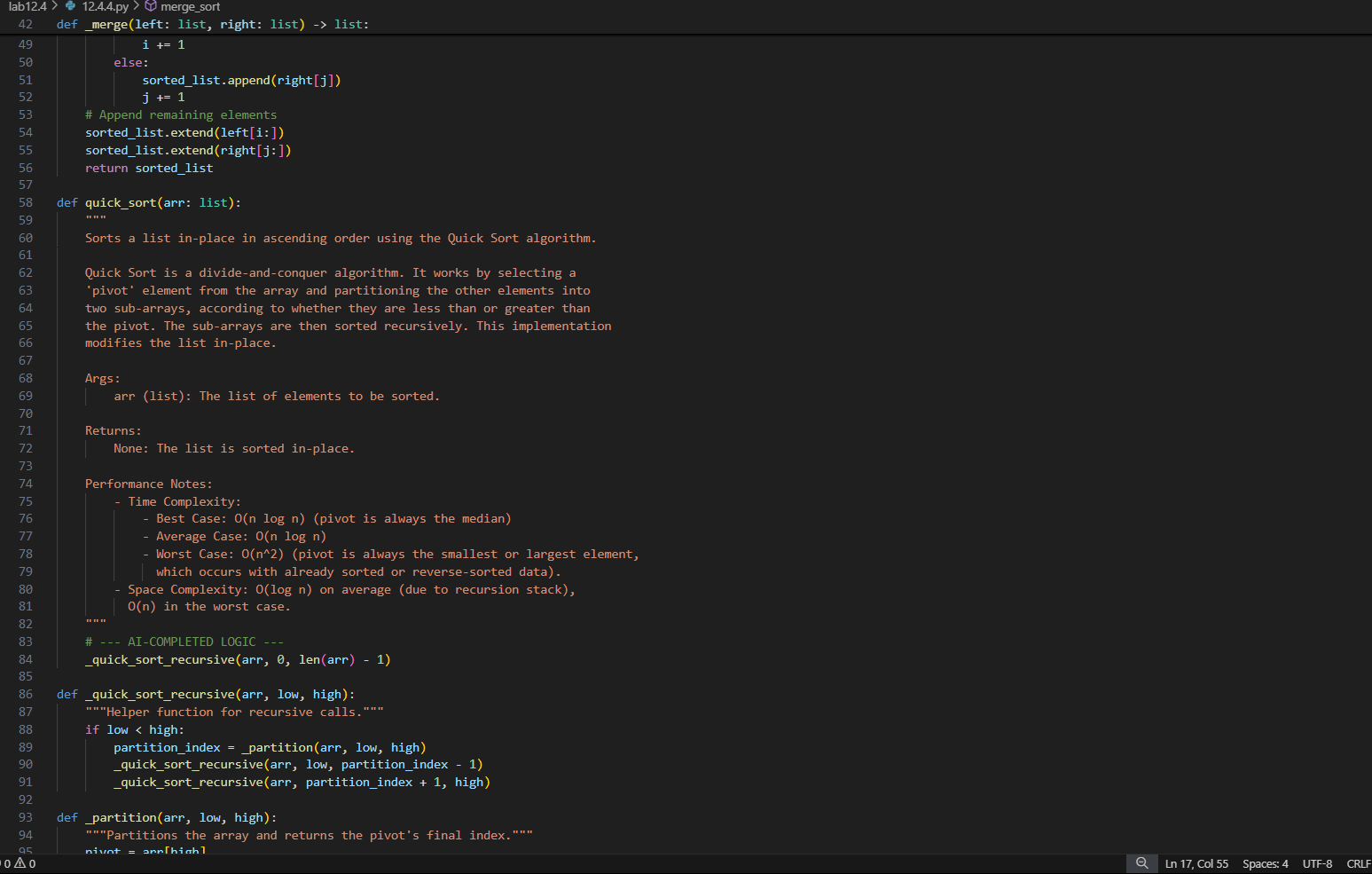
**TASK-04:**  
Quick Sort and Merge Sort Comparison

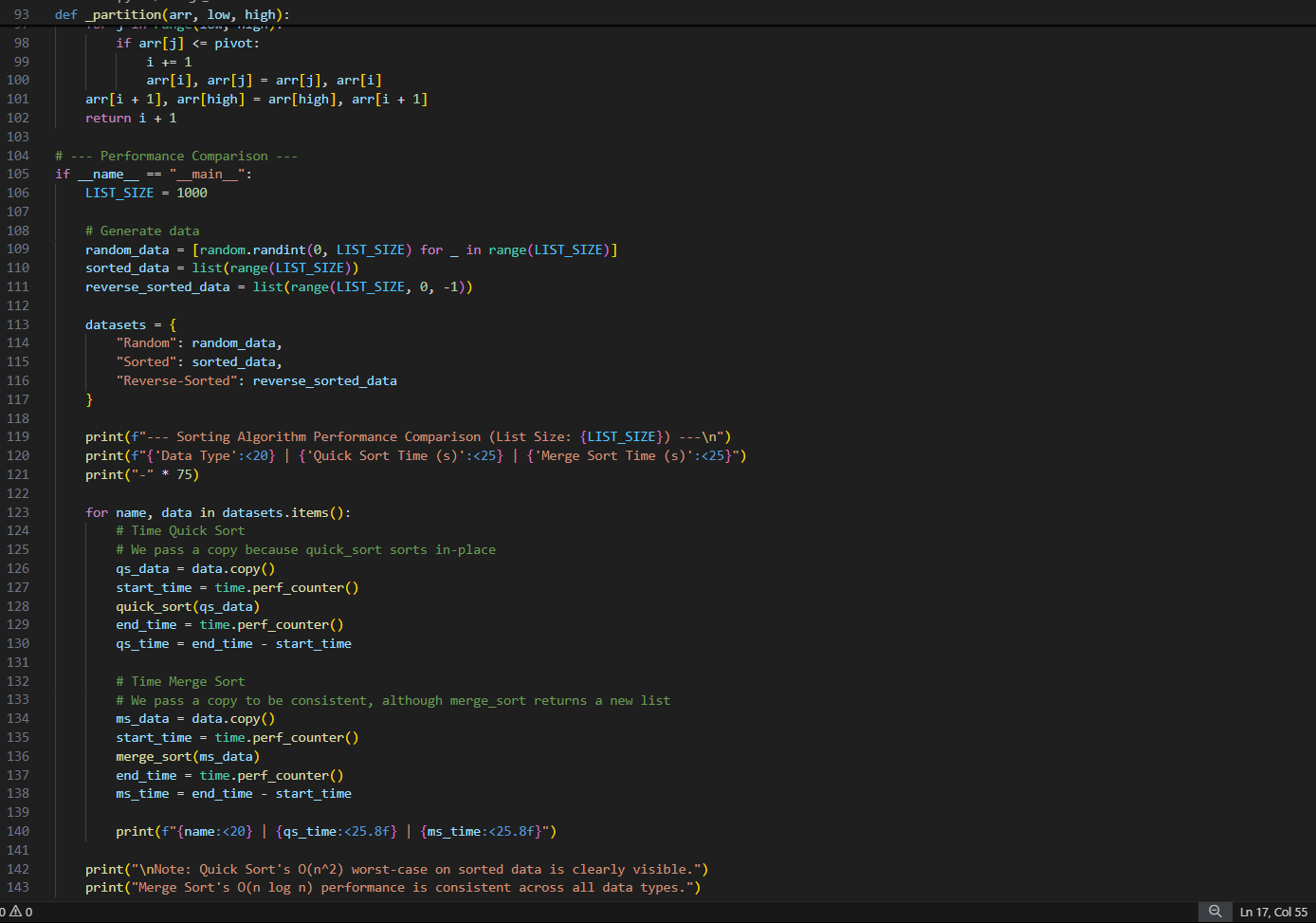
**PROMPT:**

Implement the quick sort and merge sort using recursion.

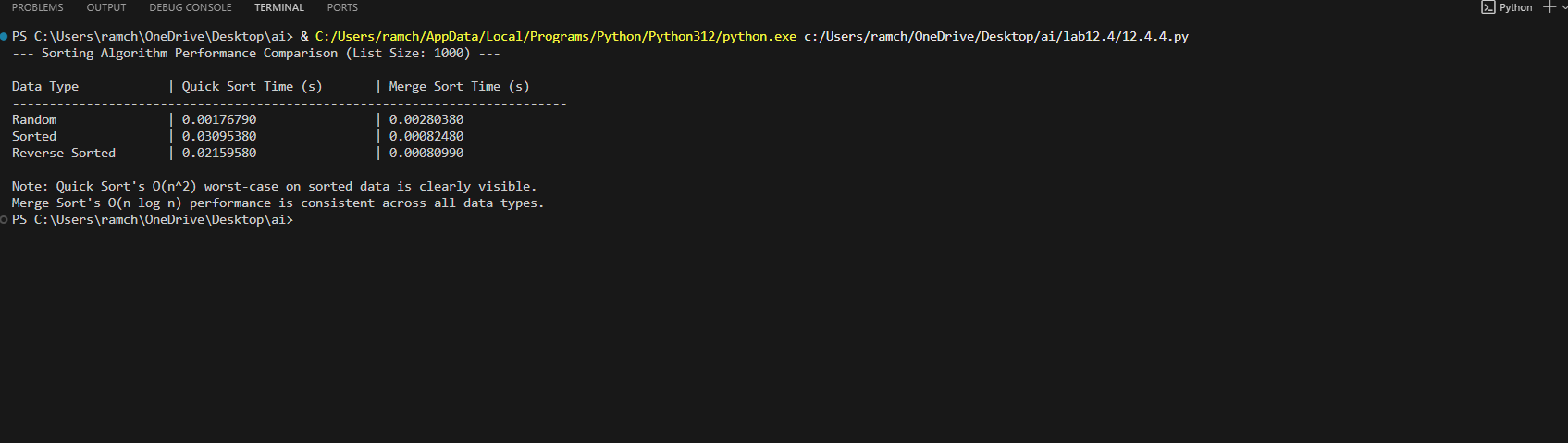
**CODE:**







**OUTPUT:**



**OBSERVATION:**

Quick Sort: Uses a **pivot** to partition the array into smaller and larger elements, then sorts each part recursively. It’s **faster on average** (O(n log n)) but may degrade to O(n²) in the worst case.

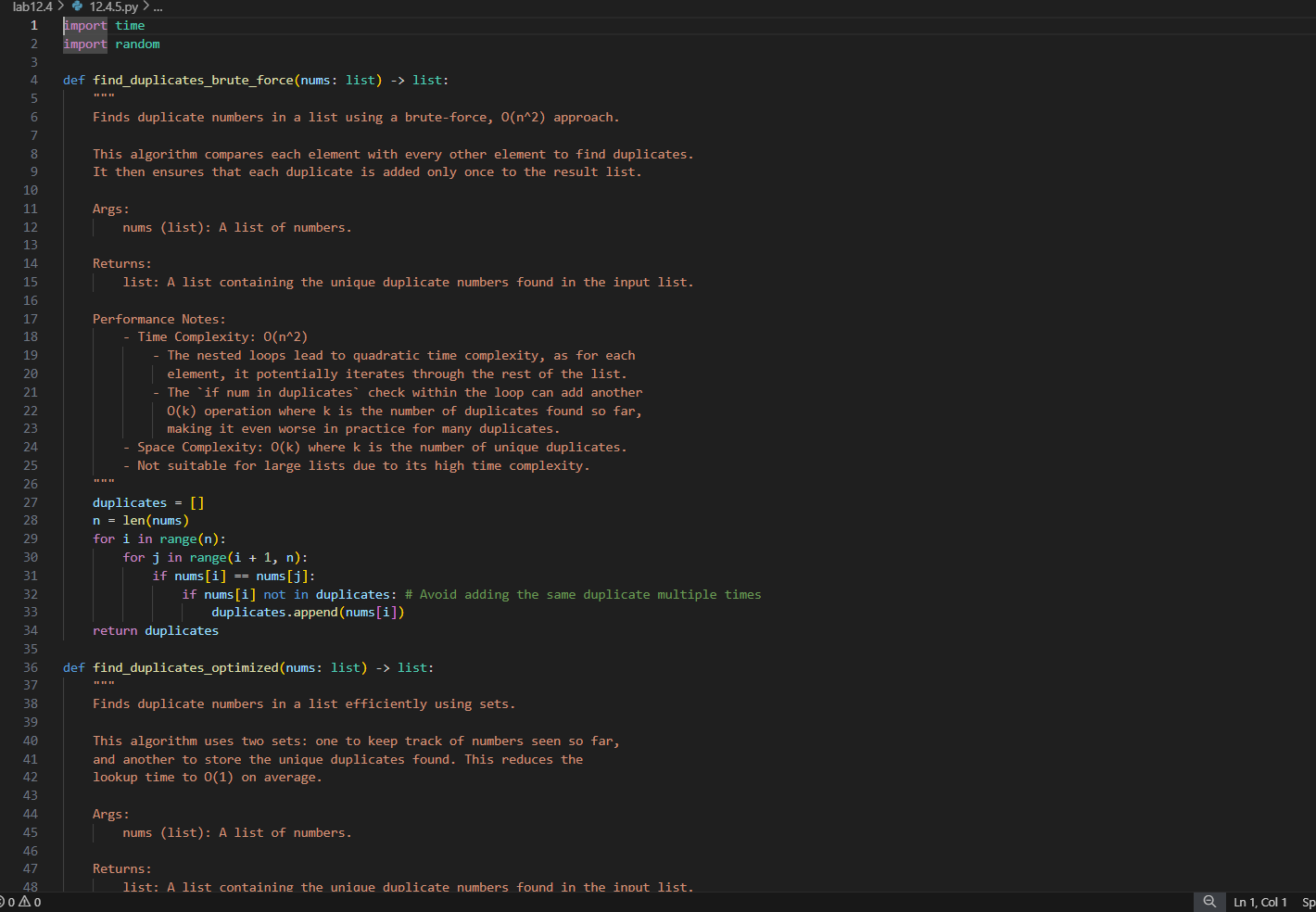
Merge Sort**:** Divides the array into halves, sorts them, and then **merges** them. It always runs in **O(n log n)** time but uses **extra memory** for merging.

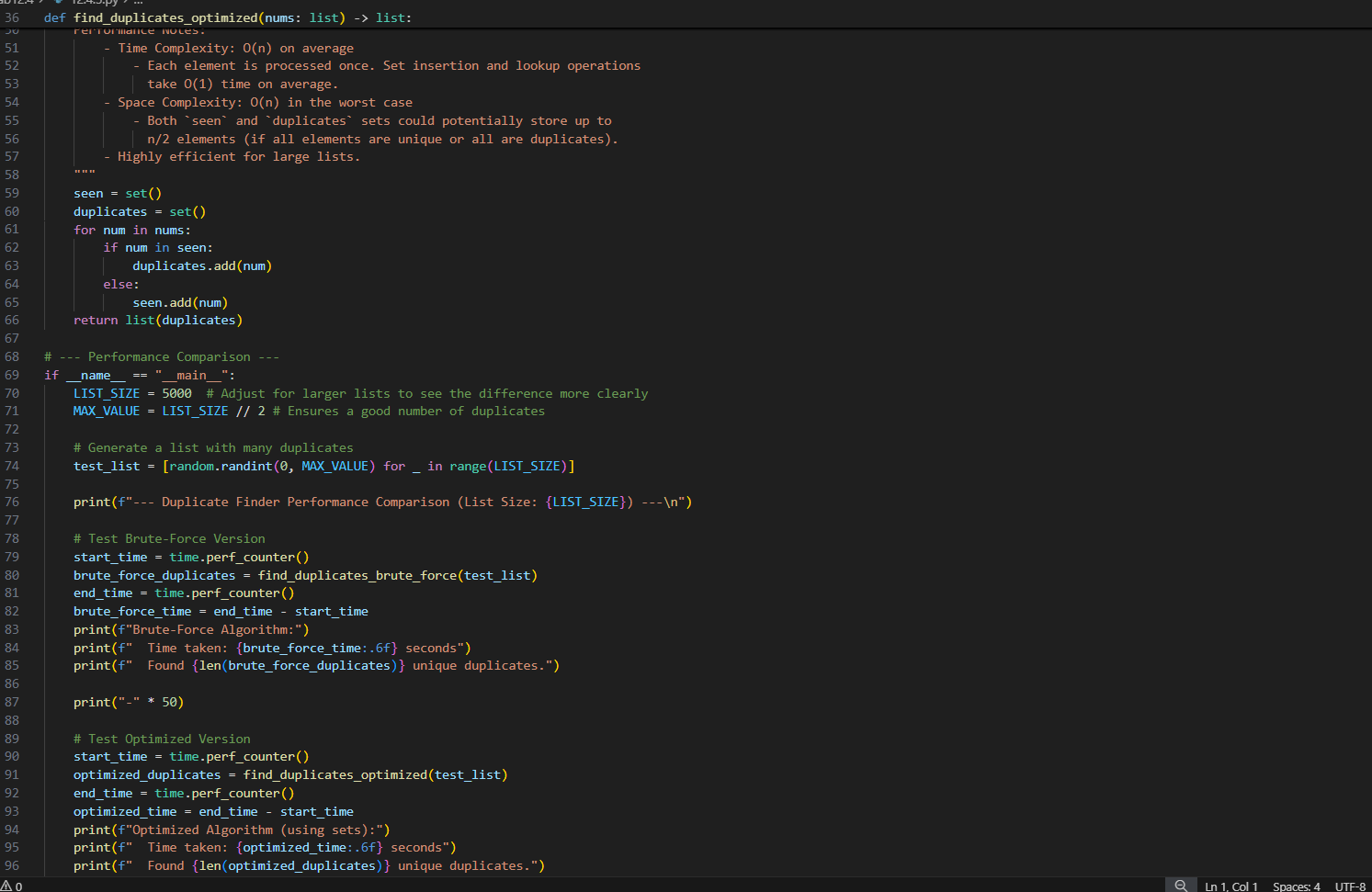
**TASK-05:**  
AI-Suggested Algorithm Optimization

**PROMPT:**

Generate the python code which implements the duplicate search.

**CODE:**





**OUTPUT:**



**OBSERVATION:**

The task involves first writing a naive duplicate-finding algorithm using nested loops, which has O(n²) complexity. Then, AI can optimize it by using a set or dictionary to track seen elements, reducing the complexity to O(n). Students compare execution times on large inputs and explain that the optimization improves efficiency by avoiding repeated comparisons.