# **Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms**

**Name:** Suhana Rehan

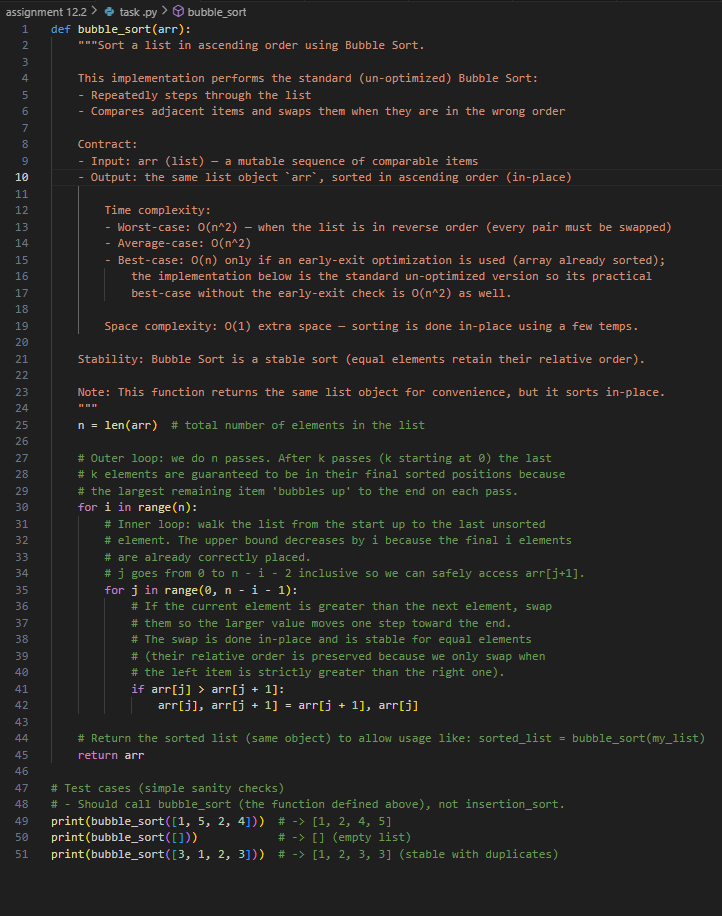
**Enrollment Number**: 2503A51L36

**Assignment Number:**12.2

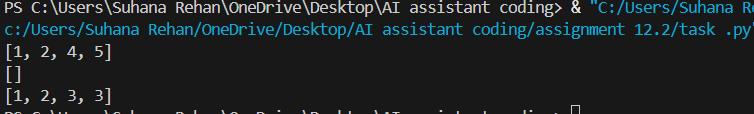
**Task 1: Bubble Sort with AI Comments**

**Prompt:**   
Add inline comments and time complexity analysis to this Bubble Sort implementation.

**Code:**

****

**Output:**

****

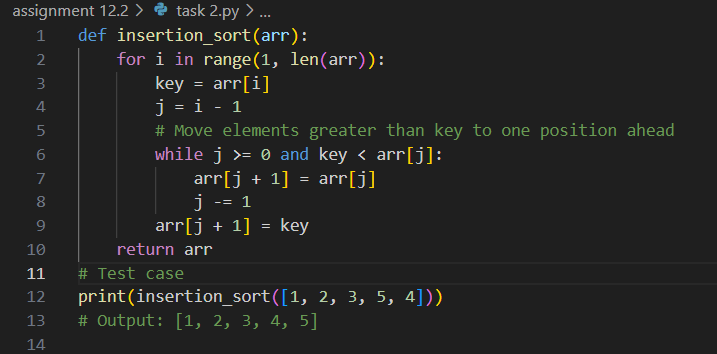
**Observations:**

* Bubble Sort repeatedly swaps adjacent elements until the list is sorted.
* It performs (O(n^2)) comparisons in the worst case, making it inefficient for large datasets.
* AI comments clarified the role of each pass and how early termination can improve performance slightly.

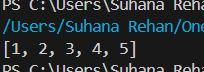
**Task 2: Optimizing Bubble Sort → Insertion Sort**

**Prompt:**   
Suggest a more efficient sorting algorithm for nearly sorted arrays and explain why.

**Code:**

****

**Output:**

****

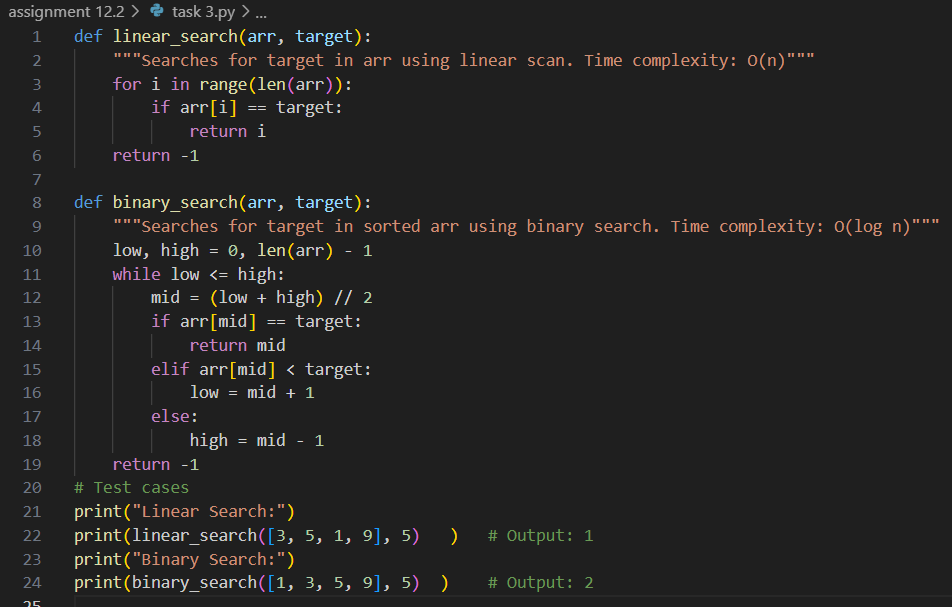
**Observations:**

* Insertion Sort is faster than Bubble Sort on nearly sorted data due to fewer shifts.
* It has a best-case time complexity of (O(n)) when the array is already sorted.
* AI highlighted that Insertion Sort minimizes unnecessary swaps, making it ideal for incremental sorting.

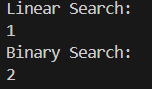
**Task 3: Binary Search vs Linear Search**

**Prompt:**   
Generate docstrings and performance notes for Linear and Binary Search, and explain when Binary Search is preferable.

**Code:**

****

**Output:**



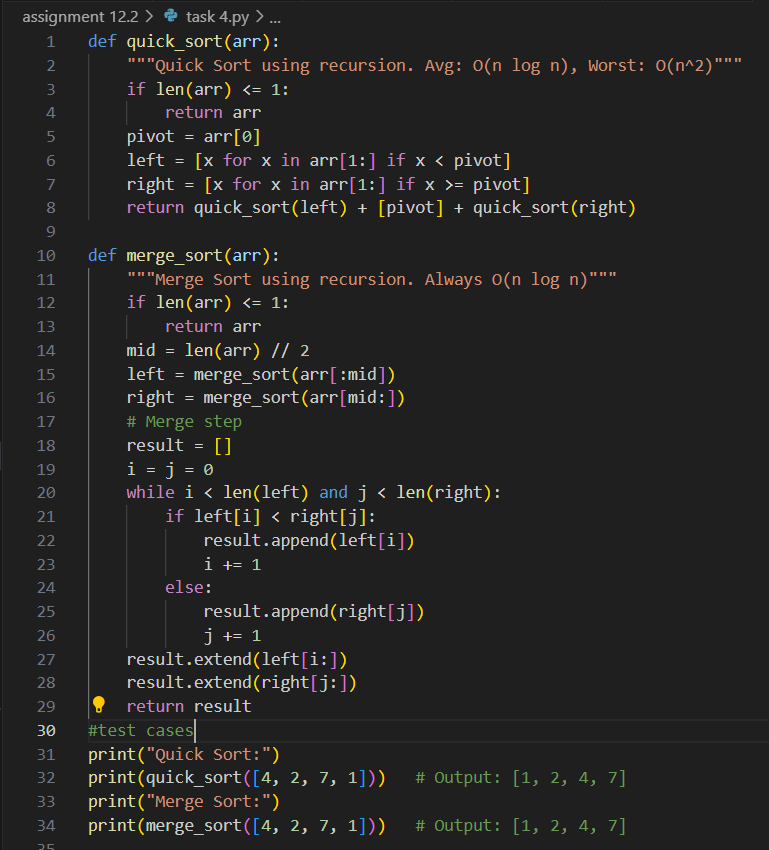
**Observations:**

* Linear Search works on any list but is slower for large datasets ((O(n))).
* Binary Search requires sorted input and performs in (O(\log n)) time.
* AI emphasized Binary Search’s efficiency for large, sorted datasets and its limitations on unsorted data.

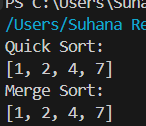
**Task 4: Quick Sort and Merge Sort Comparison**

**Prompt:**   
Complete recursive Quick Sort and Merge Sort functions with docstrings and explain their time complexities.

**Code:**

****

**Output:**



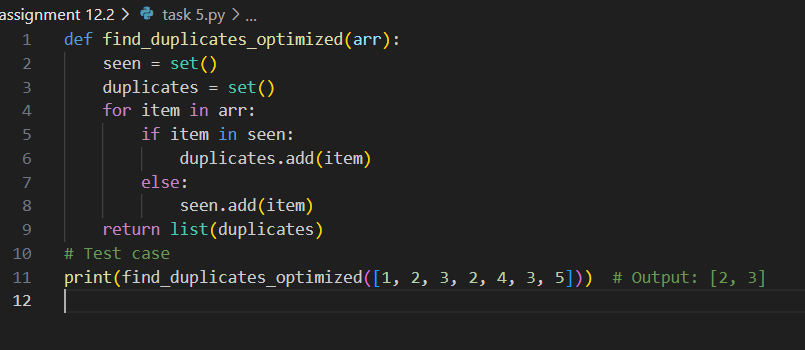
**Observations:**

* Quick Sort is faster on average but suffers in worst-case scenarios ((O(n^2))).
* Merge Sort guarantees (O(n \log n)) performance regardless of input order.
* AI explained that Merge Sort is stable and better for linked lists, while Quick Sort is faster in-place.

**Task 5: AI-Suggested Algorithm Optimization**

**Prompt:**   
Optimize this naive duplicate-finder algorithm and explain how the time complexity improves.

**Code:**

****

**Output:**



**Observations:**

* The brute-force method checks all pairs, resulting in (O(n^2)) time.
* AI replaced it with a set-based approach, reducing complexity to (O(n)).
* Execution time dropped significantly on large inputs, validating the AI’s optimization strategy.