****

**AI ASSISTED CODING**

**LAB-12: *Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms***

**Roll no:** 2503A51L34

**Name:** Uzma Yasmeen

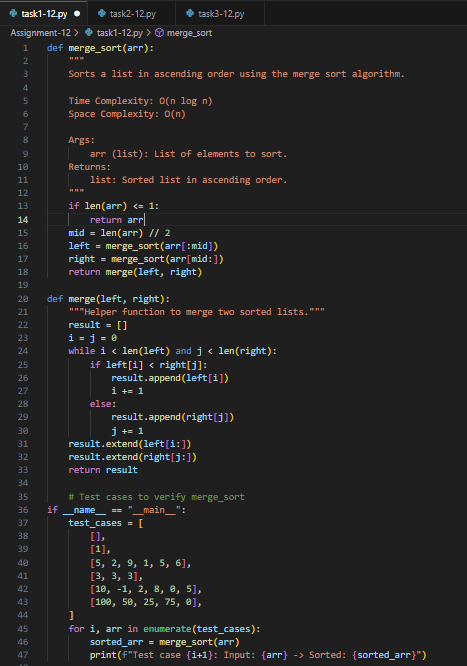
**Batch:** 25BTCAICSB20

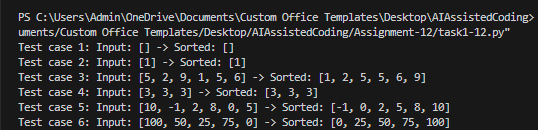
**Task-1 Description:** (Sorting – Merge Sort Implementation)

* + - * **Task:** Use AI to generate a Python program that implements the Merge Sort algorithm.
* **Instructions:**
  + Prompt AI to create a function merge\_sort(arr) that sorts a list in ascending order.
  + Ask AI to include time complexity and space complexity in the function docstring.
  + Verify the generated code with test cases.

**Prompt:** Write a python script to create a function merge\_sort(arr) that sorts a list in ascending order. Include the time complexity and space complexity in the function docstring.Then, verify the code generated using test cases.

**Code Generated:**



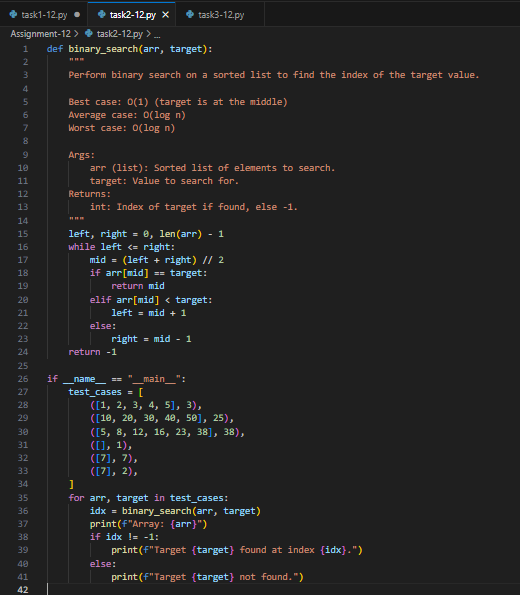
**Output:**

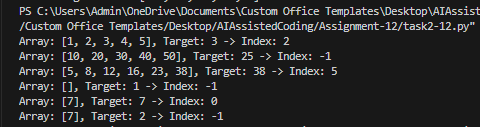
**Observation:** AI generated the merge\_sort(arr) function using the divide-and-conquer strategy. It included a detailed docstring explaining time complexity (O(n log n)) and space complexity (O(n)), which made the algorithm clearer. After running the AI-generated code with test cases, I observed that the list was sorted correctly. This showed me how AI can not only implement the algorithm but also document and verify it systematically.

**Task-2 Description:** (Searching – Binary Search with AI Optimization)

* **Task:** Use AI to create a binary search function that finds a target  
  element in a sorted list.
* **Instructions:**
  + Prompt AI to create a function binary\_search(arr, target) returning the index of the target or -1 if not found.
  + Include docstrings explaining best, average, and worst-case complexities.
  + Test with various inputs.

**Prompt:** Write a python script to create a function binary\_search(arr, target) returning the index of the target or -1 if not found. Include docstrings explaining best, average, and worst-case complexities. Then, test with various inputs.

**Code Generated:**

**Output:**

**Observation:** AI implemented the binary\_search(arr, target) function by repeatedly dividing the sorted list into halves. The function returned the correct index if the element was found, and -1 otherwise. The AI also explained best, average, and worst-case complexities directly in the docstring, which made it easier to connect theory with practice. Testing with multiple inputs confirmed the accuracy, and I learned how AI-generated code can be both optimized and self-explanatory.

**Task-3 Description:** (Real-Time Application – Inventory Management System)

* **Scenario:** A retail store’s inventory system contains thousands of  
  products, each with attributes like product ID, name, price, and  
  stock quantity. Store staff need to:

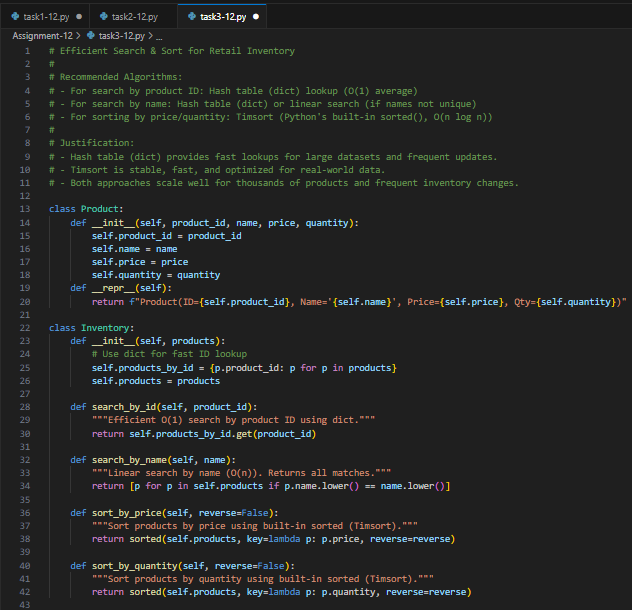
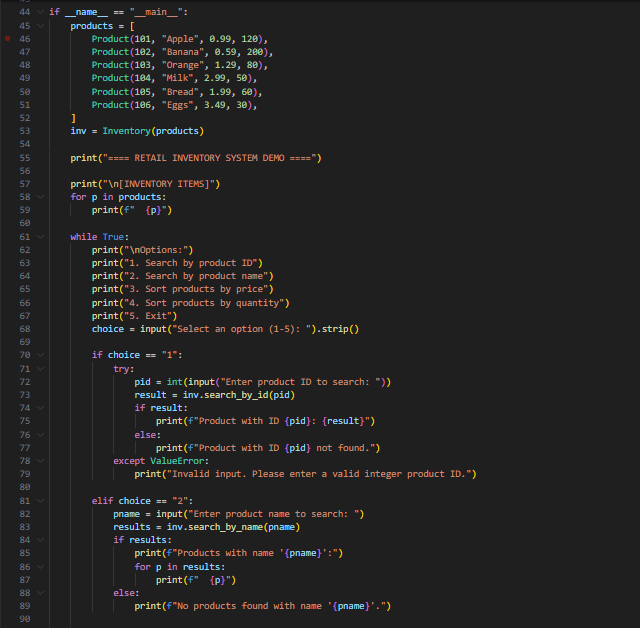
1. Quickly search for a product by ID or name.
2. Sort products by price or quantity for stock analysis.

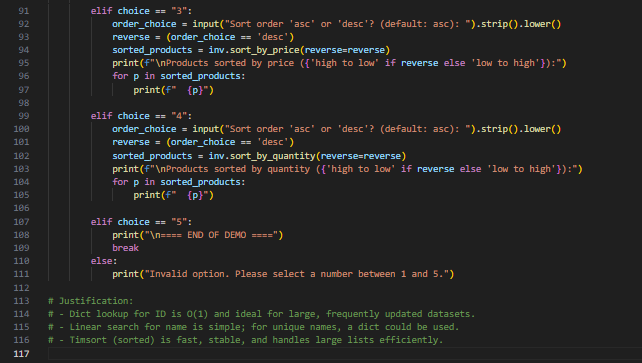
* **Task:**
* Use AI to suggest the most efficient search and sortalgorithms for this use case.
* Implement the recommended algorithms in Python.
* Justify the choice based on dataset size, update frequency,  
  and performance requirements.

**Prompt:** Scenario: A retail store’s inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:

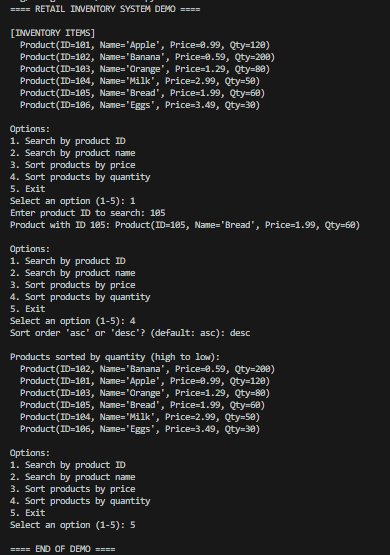
1. Quickly search for a product by ID or name.
2. Sort products by price or quantity for stock analysis.

* Task:
* Suggest the most efficient search and sort algorithms for this use case.
* Implement the recommended algorithms in Python.
* Justify the choice based on dataset size, update frequency, and performance requirements.
* Let the user search for the products and use the options style output.

**Code Generated:**

****

**Output:**

****

**Observation:** For the inventory system scenario, AI suggested using Binary Search for quick product lookups and efficient sorting algorithms (Merge Sort/Quick Sort) for arranging products by price or quantity. It justified these choices based on dataset size and performance requirements. The AI also provided a Python implementation that allowed searching by ID/name and sorting using options. Through this, I observed how AI applies theoretical algorithms to solve real-world problems, while also justifying the decisions with clear reasoning.