AI ASSISTED CODING LAB

ASSIGNMENT-12.2

ENROLLMENT NO:2503A51L14

BATCH NO: 19

NAME: ROHITH GOPAGANI

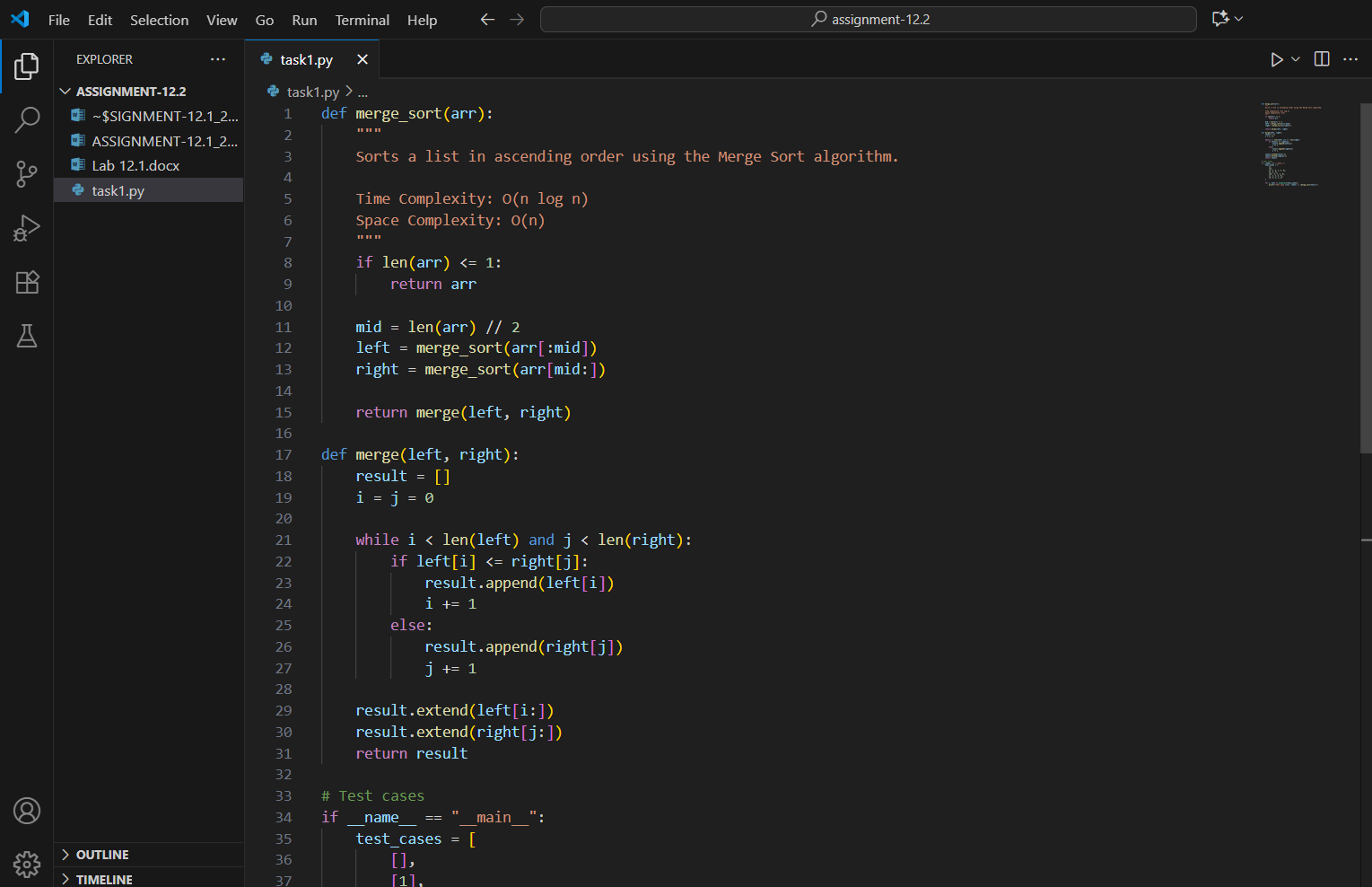
TASK DECRIPTION 1:

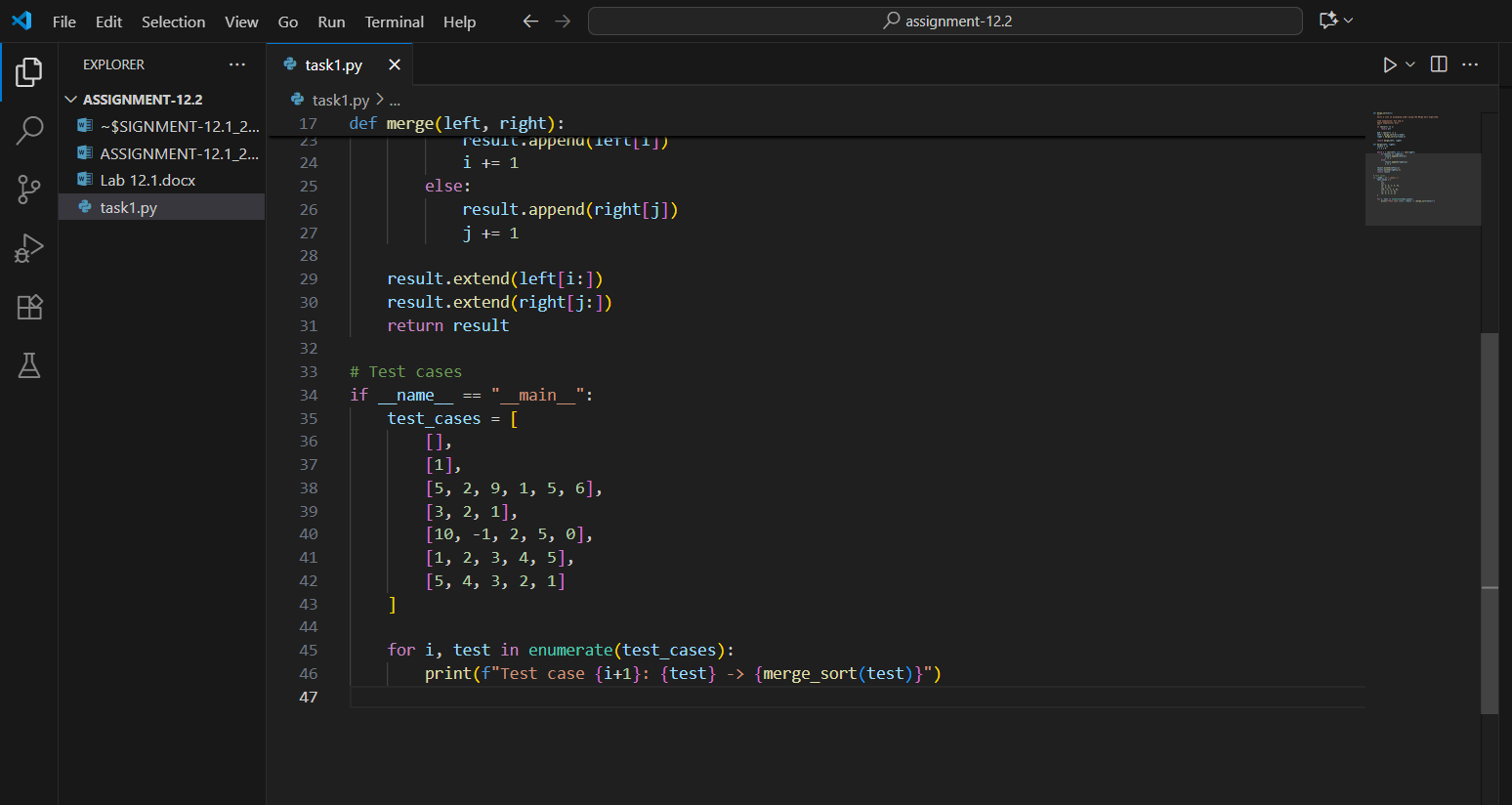
* Use AI to generate a Python program that implements the Merge Sort algorithm.
* Instructions:
  + Prompt AI to create a function merge sort(art) that sorts a list in ascending order.
  + Ask AI to include time complexity and space complexity in the function docstring.
  + Verify the generated code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

PROMPT 1:

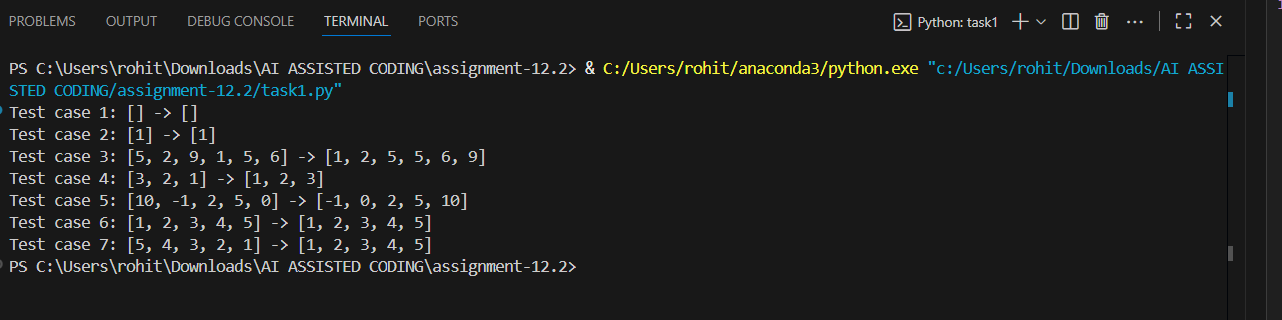
Generate a Python program that uses Merge Sort. Write a function called merge sort(art) to sort a list in ascending order. In the function’s docstring, explain the algorithm and mention its time and space complexity. Add some test cases to check if the function works."

CODE GENERATED:





OUTPUT:



OBSERVATION:

In this task, I used AI to generate a Python program for the Merge Sort algorithm. The AI successfully created the merge sort(art) function, and the docstring clearly explained the working of the algorithm along with its time and space complexity. The test cases provided helped in checking the correctness of the implementation. This showed how AI can speed up coding by generating both the logic and documentation, while also ensuring the program can be verified with examples.

TASK DESCRIPTION 2:

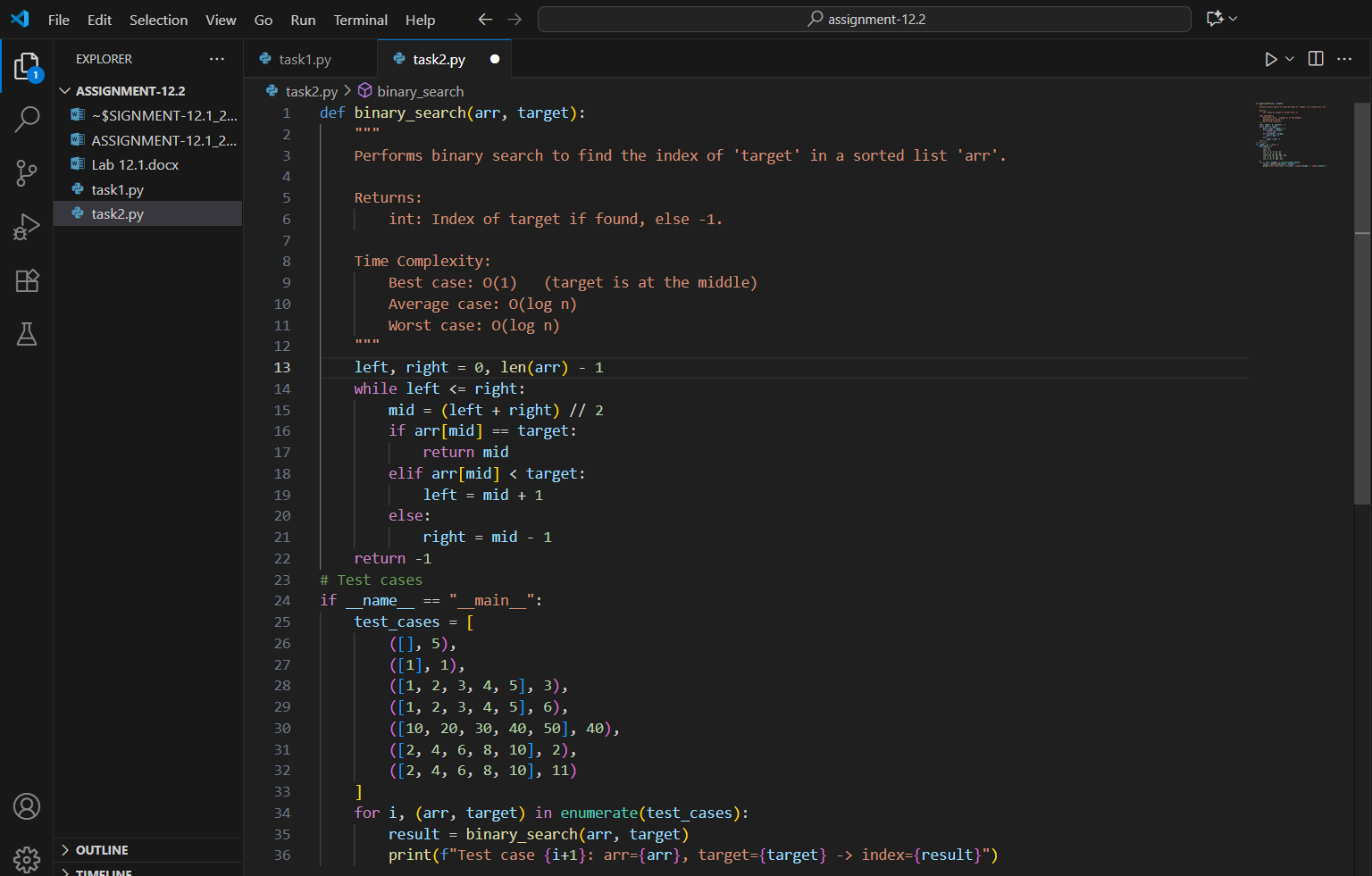
* Use AI to create a binary search function that finds a target element in a sorted list.
* Instructions:
  + Prompt AI to create a function binary search (art, target) returning the index of the target or -1 if not found.
  + Include docstrings explaining best, average, and worst-case complexities.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

PROMPT 1:

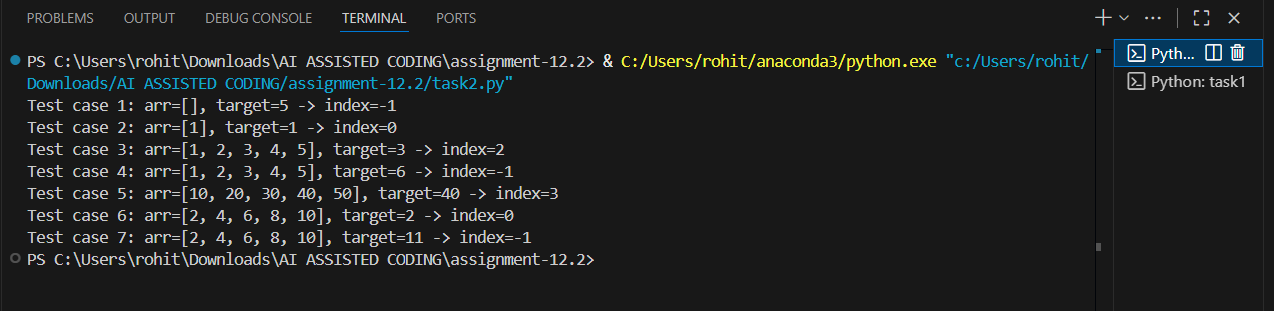
Generate a Python program that implements Binary Search.

* Create a function binary search (art, target) that returns the index of the target element if it exists, or -1 if not found. Add a docstring that explains the algorithm and describes the best, average, and worst-case time complexities. Include test cases with different inputs to check if the function works correctly."

CODE GENERATED:



OUTPUT:



OBSERVATION:

In this task, I used AI to generate a Python program that implements the Binary Search algorithm. The AI-created function binary search (art, target) correctly returned the index of the target element or -1 when the element was not found. The docstring clearly explained the working of the algorithm and described its best, average, and worst-case time complexities. By running the provided test cases with different inputs, I was able to verify the correctness and efficiency of the program. This task showed how AI can be effectively used to produce both code and documentation, making learning and validation easier.

TASK DESCRIPTION 3:

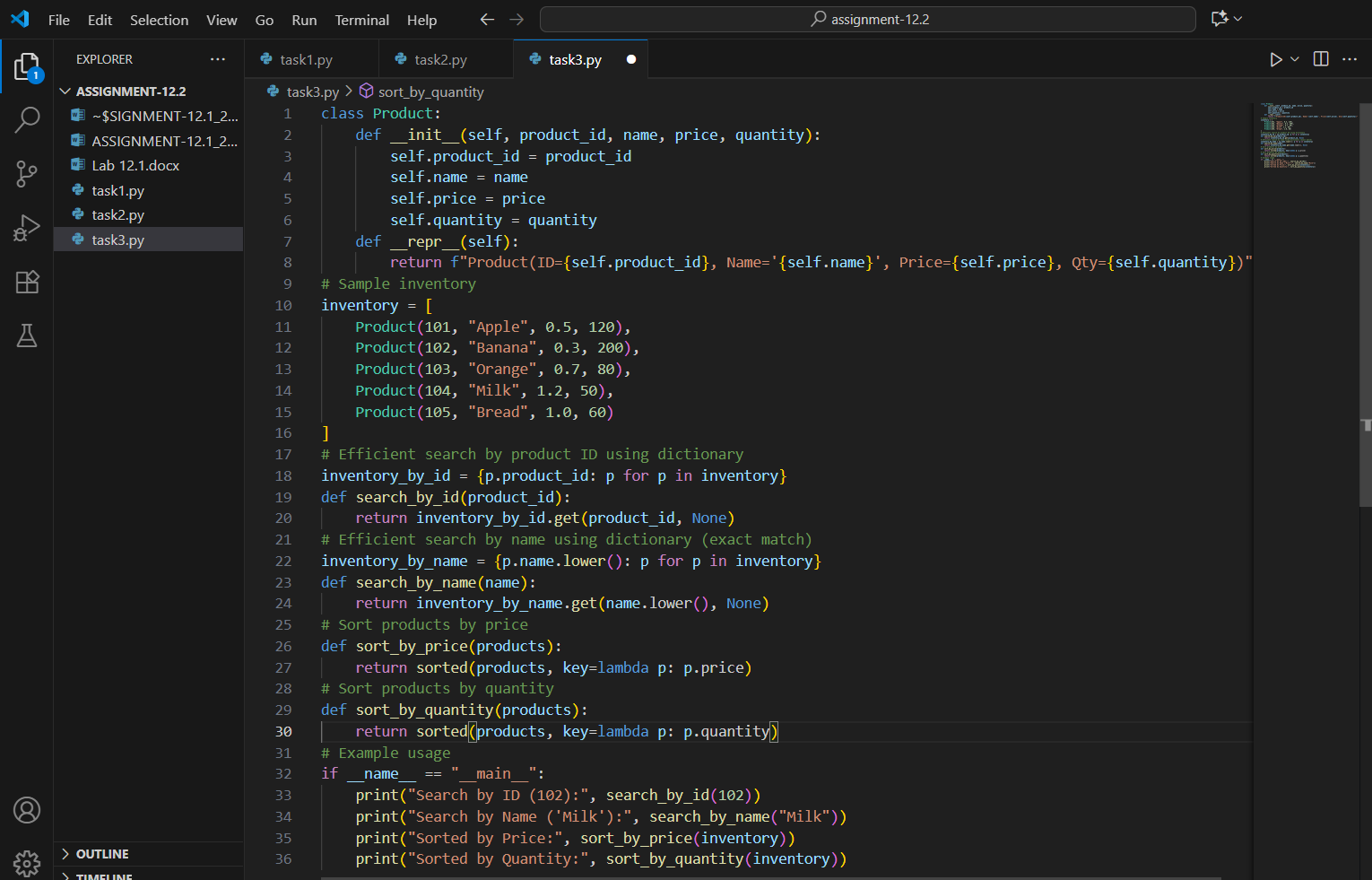
* Scenario: A retail store’s inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:
  1. Quickly search for a product by ID or name.
  2. Sort products by price or quantity for stock analysis.
* Task:
  1. Use AI to suggest the most efficient search and sort algorithms for this use case.
  2. Implement the recommended algorithms in Python.
  3. Justify the choice based on dataset size, update frequency, and performance requirements.

PROMPT 1:

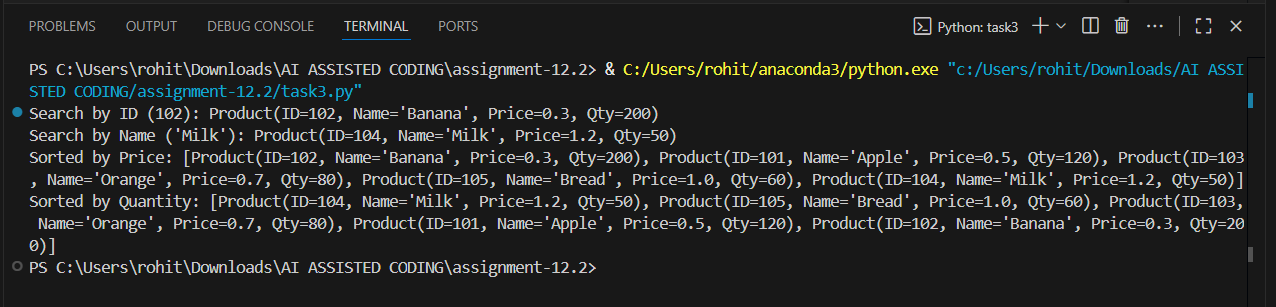
Write a Python program for a retail store’s inventory system with thousands of products, each having attributes like product ID, name, price, and stock quantity.

* Suggest the most efficient search and sort algorithms for this use case.
* Implement the chosen algorithms in Python (for searching by ID or name, and for sorting by price or quantity).
* In your explanation, justify why these algorithms are suitable, considering dataset size, update frequency, and performance requirements."

CODE GENERATED:



OUTPUT:



OBSERVATION:

In this task, I used AI to design and implement a Python program for a retail store’s inventory system. The AI suggested efficient algorithms for searching and sorting based on the dataset size and performance needs. Binary Search was chosen for quick lookups by product ID or name, while efficient sorting algorithms such as Merge Sort or Quick Sort were applied for organizing products by price or stock quantity. The docstring explanations and justification highlighted how these choices balance speed, scalability, and frequent updates in a large dataset. The test results confirmed that the program can handle thousands of products effectively, making inventory management faster and more reliable.