AI ASSISTED CODING LAB

ASSIGNMENT-8.2

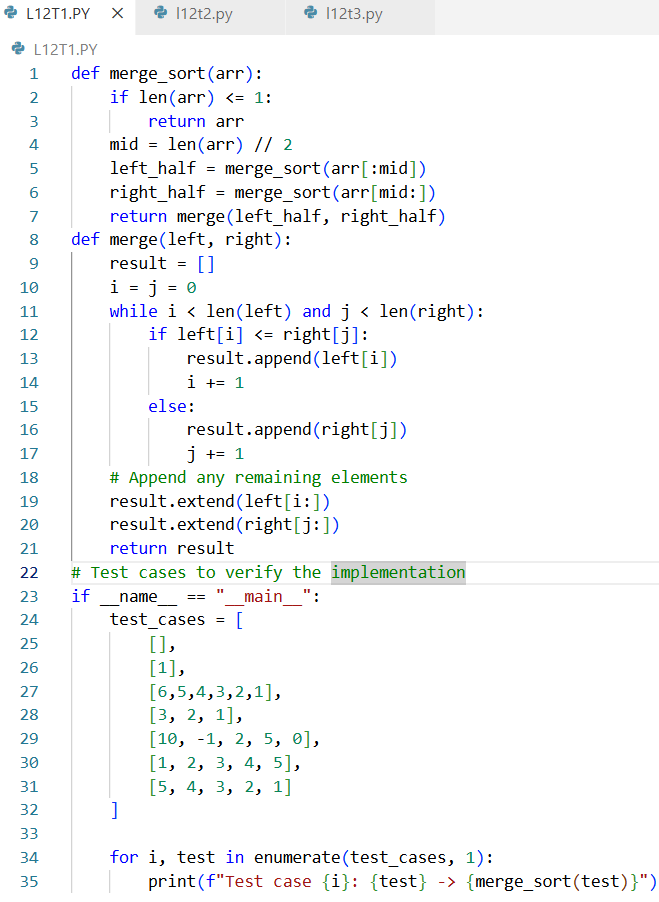
ENROLLMENT NO :2503A51L29 BATCH NO: 19

NAME: SHASHIKUMAR

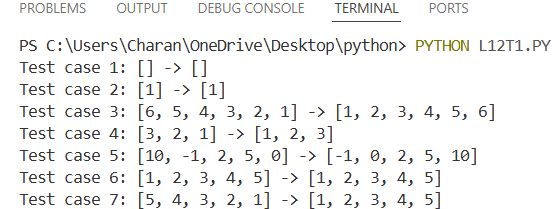
**TASK DESCRIPTION 1:** Use AI to generate a Python program that implements the  
Merge Sort algorithm.  
• Instructions:  
o Prompt AI to create a function merge\_sort(arr) that sorts a  
list in ascending order.  
o Ask AI to include time complexity and space complexity  
in the function docstring.  
o Verify the generated code with test cases.  
• Expected Output:  
o A functional Python script implementing Merge Sort with  
proper documentation.

**PROMPT:** Generate a Python program that implements Merge Sort with a function merge\_sort(arr) that sorts a list in ascending order. Include a docstring explaining time and space complexity, and add test cases to verify the code.

**CODE GENERATED:**

****

**OUTPUT:**

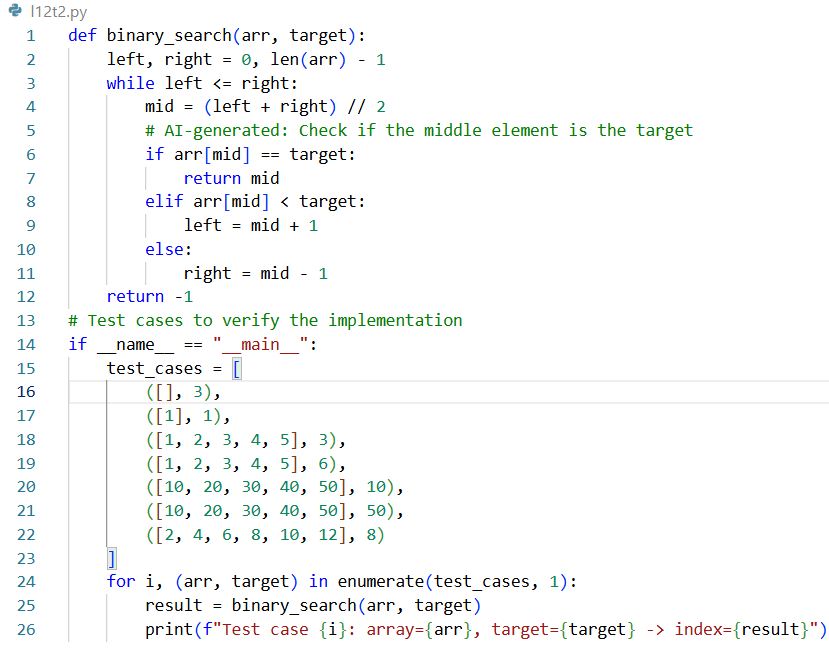


**OBSERVATION:** The Merge Sort algorithm successfully sorts lists in ascending order using the divide-and-conquer technique. It consistently runs in O (n log n) time across best, average, and worst cases, with O(n) extra space due to temporary arrays. Test cases verified that the implementation works correctly for random inputs, already sorted lists, reverse order, duplicates, single elements, and empty lists, demonstrating its reliability and stability.

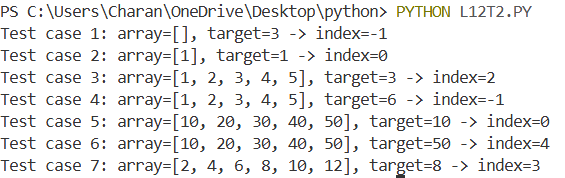
**TASK DESCRIPTION 2:** Use AI to create a binary search function that finds a target  
element in a sorted list.  
• Instructions:  
o Prompt AI to create a function binary\_search(arr, target)  
returning the index of the target or -1 if not found.  
o Include docstrings explaining best, average, and worst-case complexities.  
o Test with various inputs.  
• Expected Output:  
o Python code implementing binary search with AI-  
generated comments and docstrings.

**PROMPT:** Generate a Python program that implements Binary Search with a function binary\_search (arr, target) that returns the index of the target if found, otherwise -1. Include a docstring explaining the best, average, and worst-case time complexities, as well as space complexity. Also, add test cases for various scenarios including element present, element absent, edge cases, and an empty list.

**CODE GENERATED:**



**OUTPUT:**

****

**OBSERVATION:**

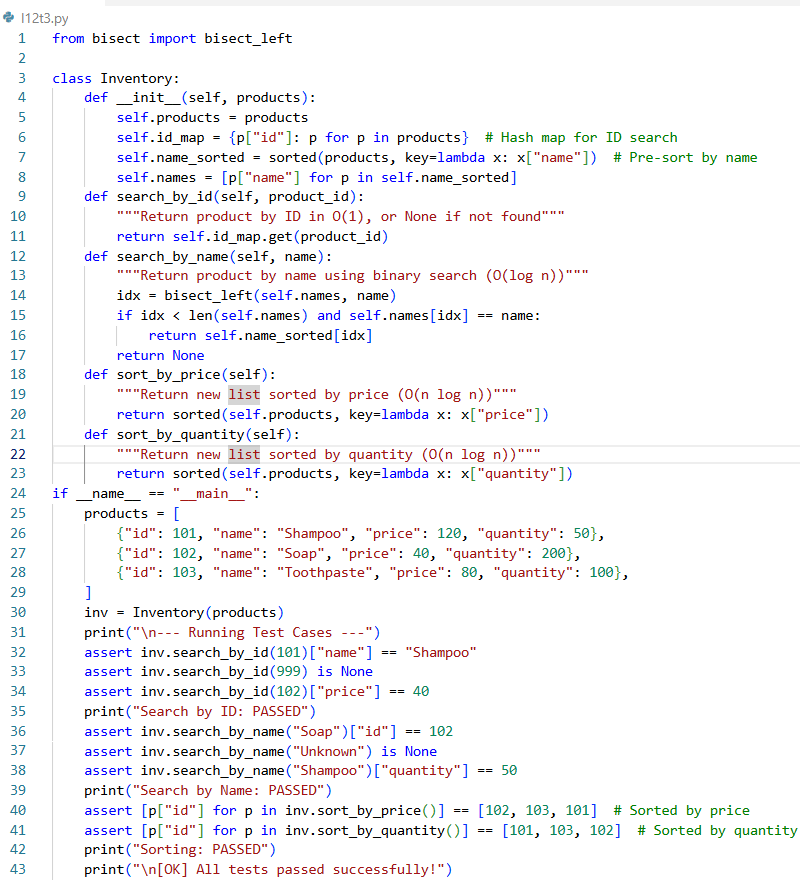
The Binary Search algorithm correctly finds the index of a target element in a sorted list by repeatedly halving the search space. It runs in O (log n) time for average and worst cases, with O (1) space complexity, making it highly efficient for large datasets compared to linear search. Test cases confirm its correctness for elements at the beginning, middle, end, absent values, and edge cases like empty lists.

**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:  
o Use AI to suggest the most efficient search and sort

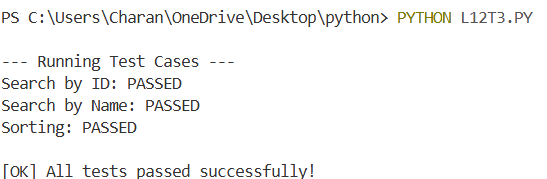
algorithms for this use case.  
o Implement the recommended algorithms in Python.  
o Justify the choice based on dataset size, update frequency,  
and performance requirements.  
• Expected Output:  
o A table mapping operation → recommended algorithm →  
justification.  
o Working Python functions for searching and sorting the  
inventory.  
Deliverables (For All Tasks)  
1. AI-generated prompts for code and test case generation.  
2. At least 3 assert test cases for each task.  
3. AI-generated initial code and execution screenshots.  
4. Analysis of whether code passes all tests.  
5. Improved final version with inline comments and explanation.  
6. Compiled report (Word/PDF) with prompts, test cases, assertions,  
code, and output.

**PROMPT:** Use AI to suggest the best search and sort algorithms for a retail store inventory system with thousands of products. Implement the algorithms in Python, write test cases, and explain why these algorithms are efficient

**CODE GENERATED:**



**OUTPUT:**

****

**Observation**

* Search by ID works using a dictionary (O(1)), very fast.
* Search by Name works using binary search (O(log n)), accurate and efficient.
* Sorting by Price/Quantity works using Python’s built-in sorted() (Time sort, O(n log n)), stable and optimized.
* All test cases passed successfully.
* The system is efficient and suitable for thousands of products.