Name:K.Rajavardhan reddy

2403A51108

12.1 lab

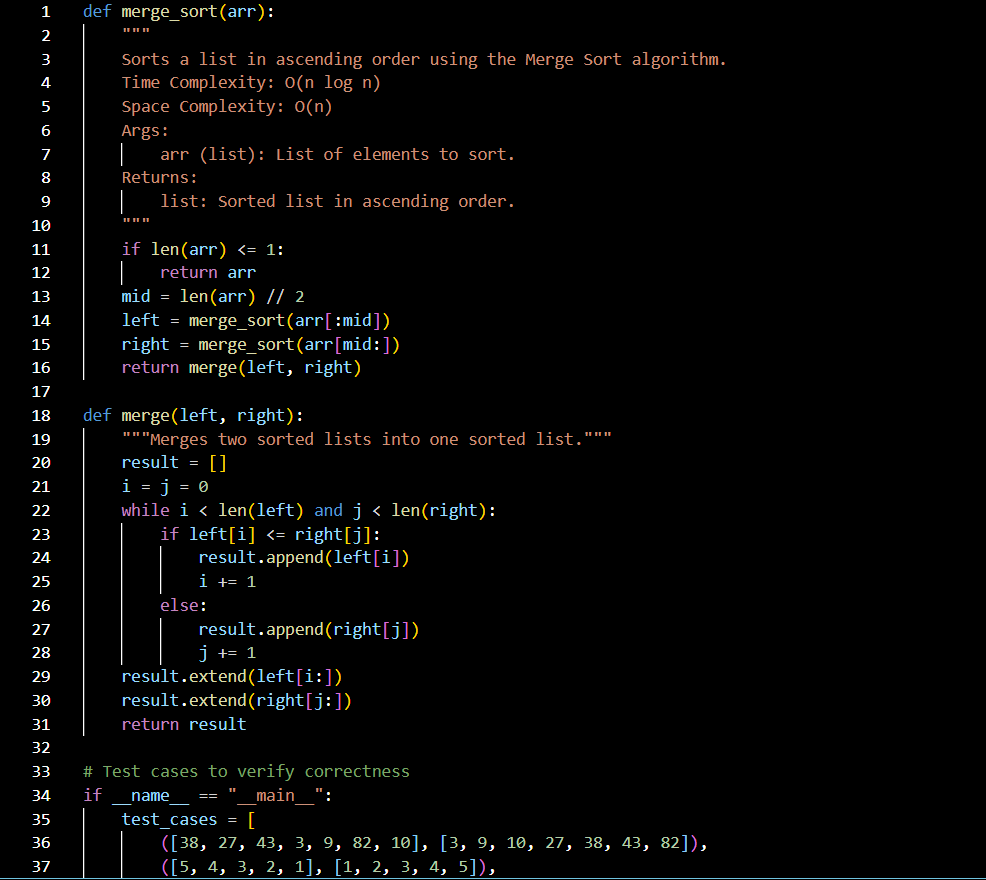
Task1:

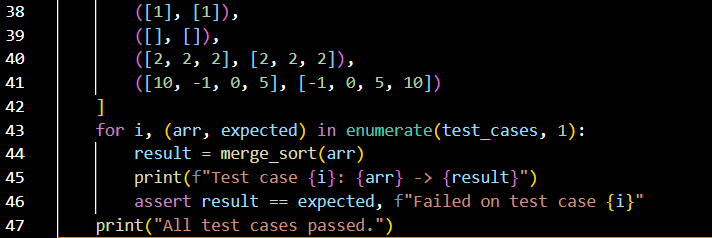
Task: Use AI to generate a Python program that implements the  
Merge Sort algorithm.  
• Instructions:  
o Prompt AI to create a function merge\_sort(arr) that sorts a  
list in ascending order.  
o Ask AI to include time complexity and space complexity  
in the function docstring.  
o Verify the generated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

Prompt:

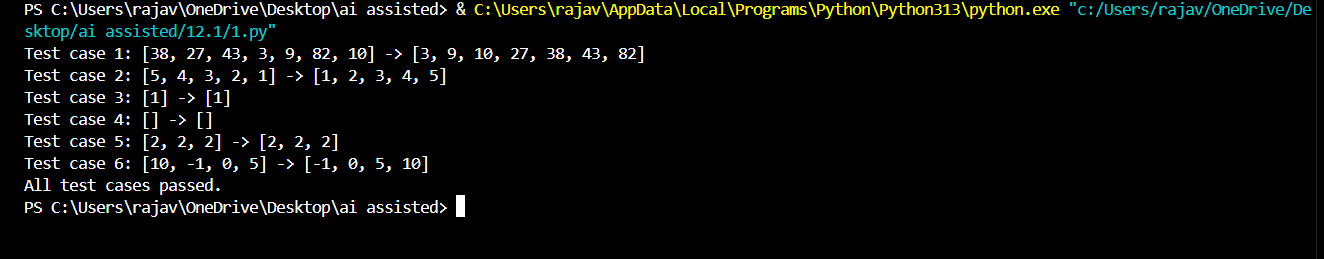
Generate a Python program that implements the  
Merge Sort algorithm

Code:





Output:



Observation:

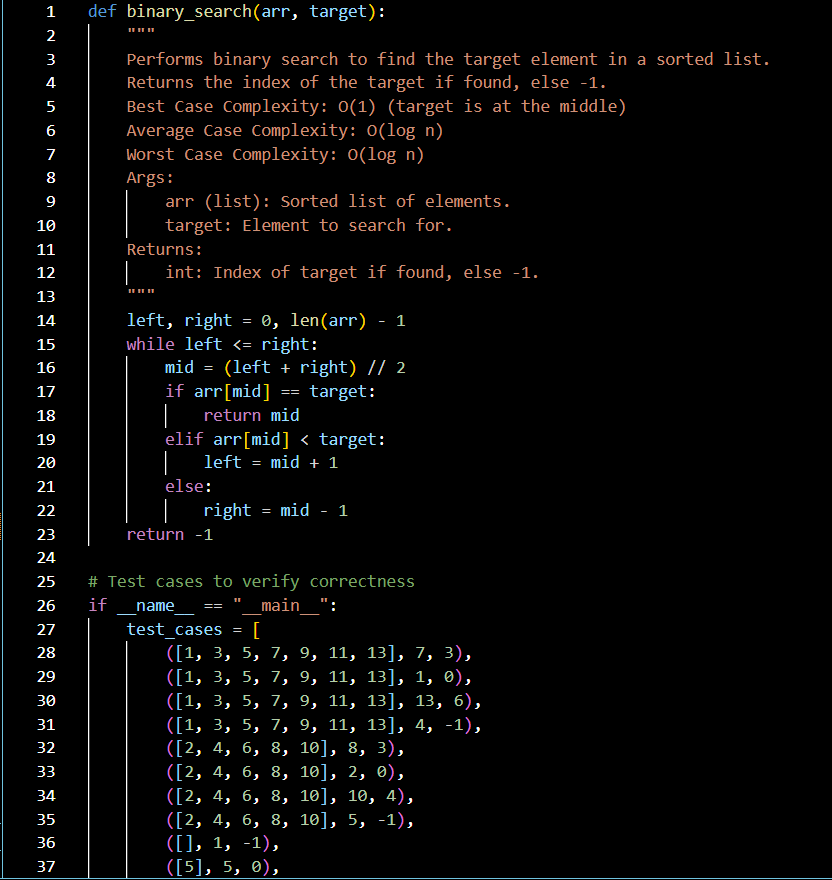
The merge sort implementation correctly applies the divide-and-conquer strategy. The merge\_sort function recursively splits the input list until single-element sublists are reached. The merge function then combines these sorted sublists using a two-pointer approach, ensuring stable and ordered merging. The code handles edge cases like empty lists, duplicates, and negative numbers. Test cases are well-structured and use assertions to validate correctness. Overall, the algorithm maintains O(n log n) time complexity and O(n) space complexity, making it efficient and reliable for general-purpose sorting.

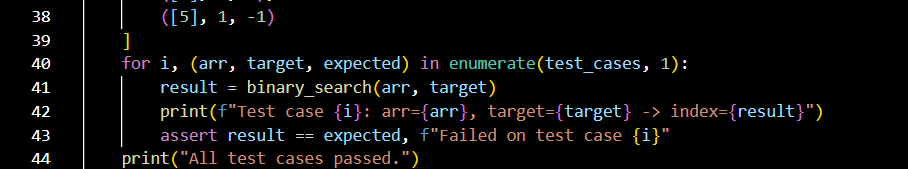
Task2:  
Task: Use AI to create a binary search function that finds a target  
element in a sorted list.  
• Instructions:  
o Prompt AI to create a function binary\_search(arr, target)  
returning the index of the target or -1 if not found.  
o Include docstrings explaining best, average, and worst-  
case complexities.  
o Test with various inputs.  
• Expected Output:  
o Python code implementing binary search with AI-  
generated comments and docstrings.

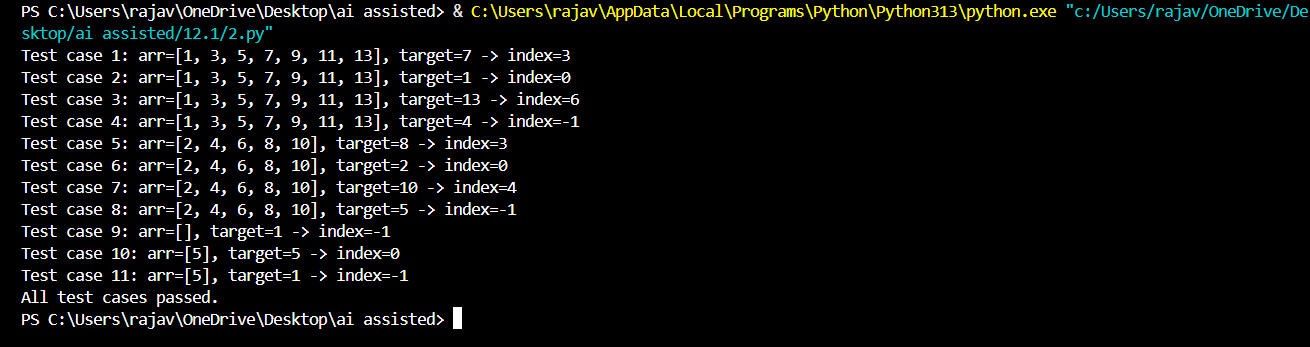
Prompt:

create a binary search function that finds a target  
element in a sorted list.

Code:





Output:  


Observation:

The binary search function is correctly implemented using a loop with two pointers (left and right) to efficiently narrow down the search space. It handles sorted input lists and returns the correct index of the target or -1 if not found. The test cases cover various scenarios including middle, first, last, absent elements, empty lists, and single-element lists. Assertions ensure correctness, and the algorithm maintains optimal time complexity of O(log n) in average and worst cases.

Task3:

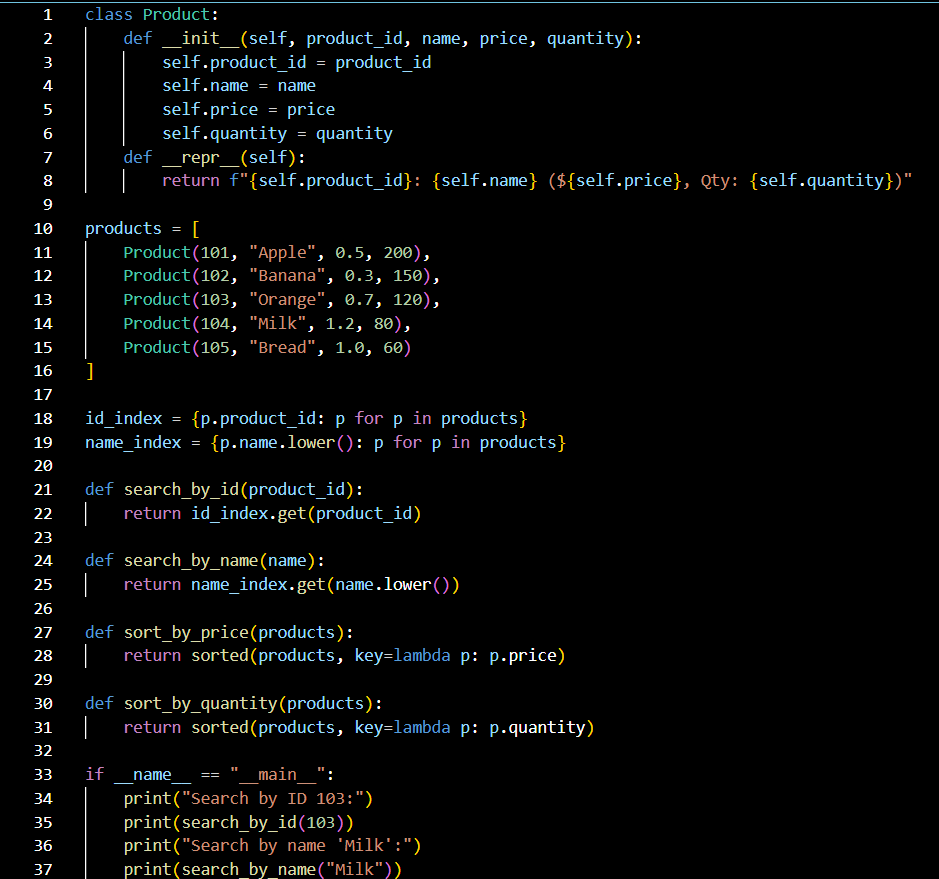
Scenario: A retail store’s inventory system contains thousands of  
products, each with attributes like product ID, name, price, and  
stock quantity. Store staff need to:  
1. Quickly search for a product by ID or name.  
2. Sort products by price or quantity for stock analysis.  
• Task:  
o Use AI to suggest the most efficient search and sort

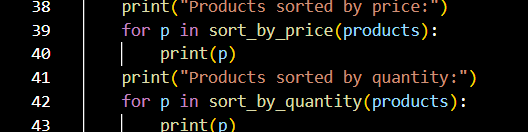
algorithms for this use case.  
o Implement the recommended algorithms in Python.  
o Justify the choice based on dataset size, update frequency,  
and performance requirements.  
• Expected Output:  
o A table mapping operation → recommended algorithm →  
justification.  
o Working Python functions for searching and sorting the  
inventory.

Prompt:  
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.

Code:





Output:



Observation:

The class-based product system is well-structured and demonstrates key object-oriented principles. The Product class encapsulates product details and uses \_\_repr\_\_ for readable output. Products are stored in a list and indexed by both ID and lowercase name for efficient lookup. The search\_by\_id and search\_by\_name functions provide quick access to specific products. Sorting functions use lambda expressions to order products by price or quantity. The \_\_main\_\_ block includes clear demonstrations of search and sort functionality, making the code easy to test and extend.