**AI ASSISTED CODING **

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

Name:Bathini Sahasra

Roll No:2503A51L07

Batch:25BTCAICSB19

**Task Description #1**:

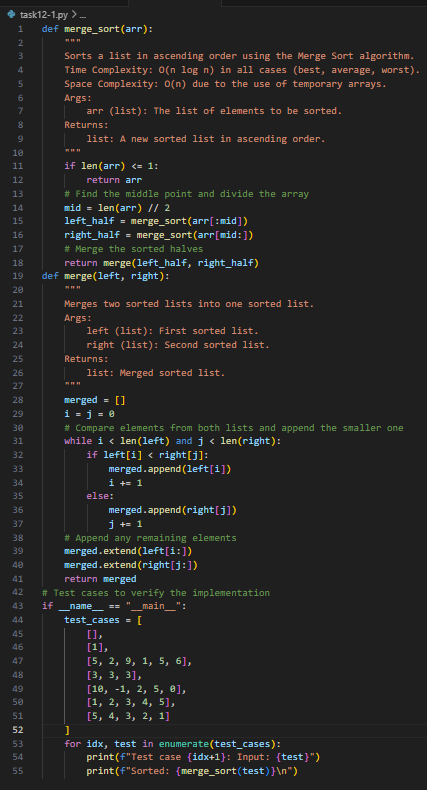
(Sorting – Merge Sort Implementation)  
• Task: Use AI to generate a Python program that implements the  
Merge Sort algorithm.  
• Instructions:  
o Prompt AI to create a function merge\_sort(arr) that sorts a  
list in ascending order.  
o Ask AI to include time complexity and space complexity  
in 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*

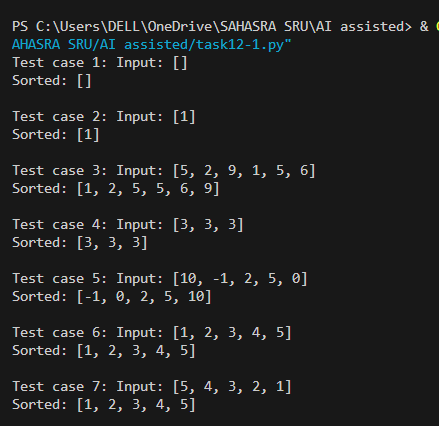
*Merge Sort algorithm create a function merge\_sort(arr) that sorts a*

*list in ascending order include time complexity and space complexity*

*in the function docstring.Verify the generated code with test cases.*

**

**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 Description #2**

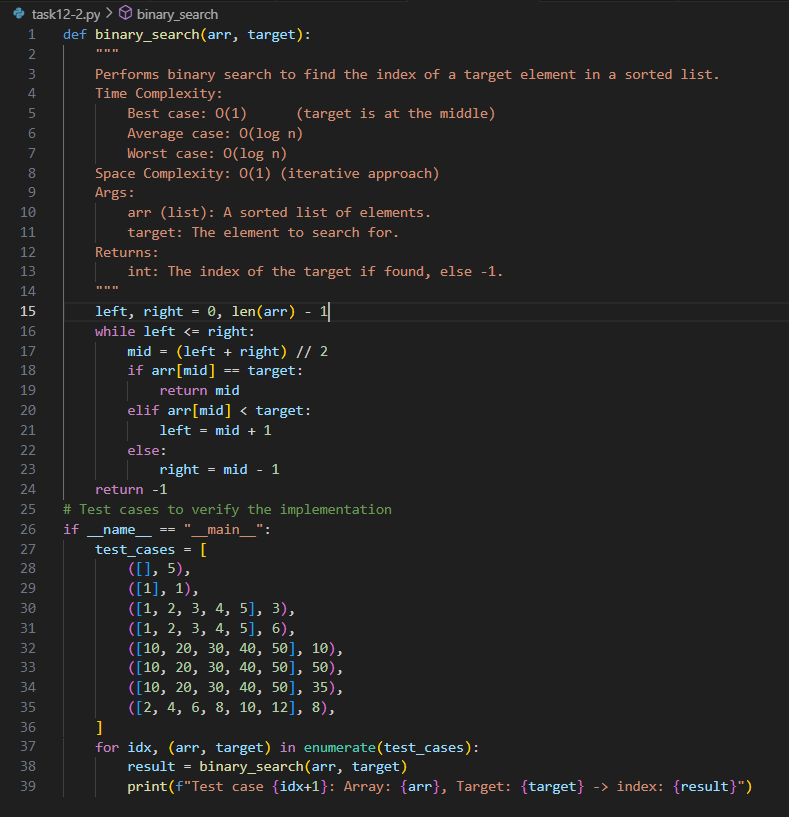
(Searching – Binary Search with AI  
Optimization)  
• Task: Use AI to create a binary search function that finds a target  
element in a sorted list.  
• Instructions:  
o Prompt AI to create a function binary\_search(arr, target)  
returning the index of the target or -1 if not found.  
o Include docstrings 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*

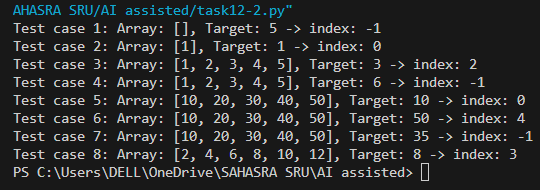
*element in a sorted list.create a function binary\_search(arr, target)*

*returning the index of the target or -1 if not found.Include docstrings explaining*

*best, average, and worst-case complexities.Test with various inputs.*

**

**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 Description #3**

(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:  
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**: *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.*

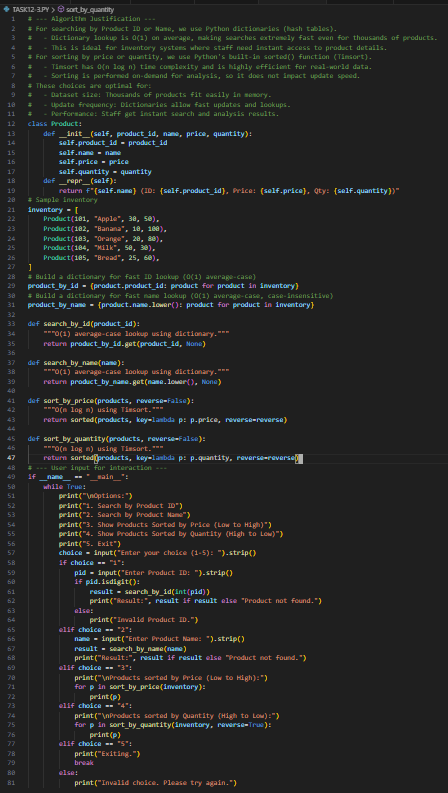
*2. Sort products by price or quantity for stock analysis.*

*suggest the most efficient search and sort algorithms for this use case.*

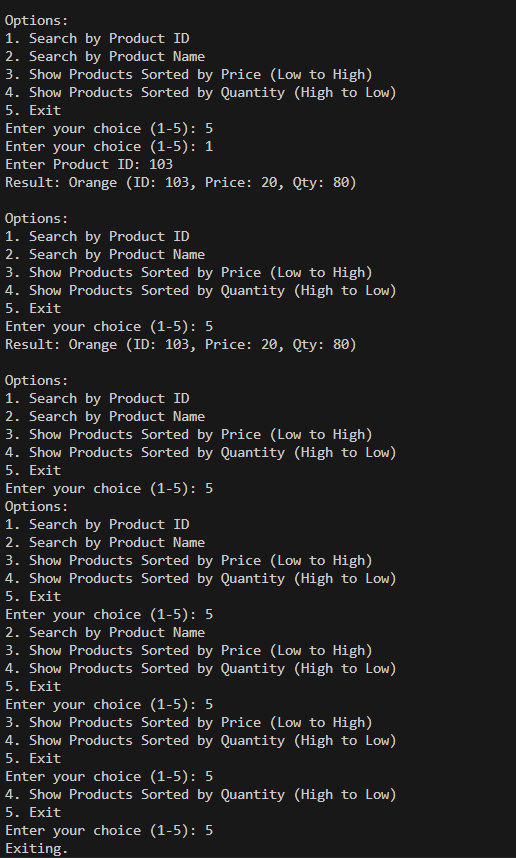
*o Implement the recommended algorithms in Python.*

*o Justify the choice based on dataset size, update frequency,*

*and performance requirements*

******

**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.