SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week6 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber:12.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms** **Lab Objectives:** • Apply AI-assisted programming to implement and optimize sorting and searching algorithms. • Compare different algorithms in terms of efficiency and use	Week6 - Monday

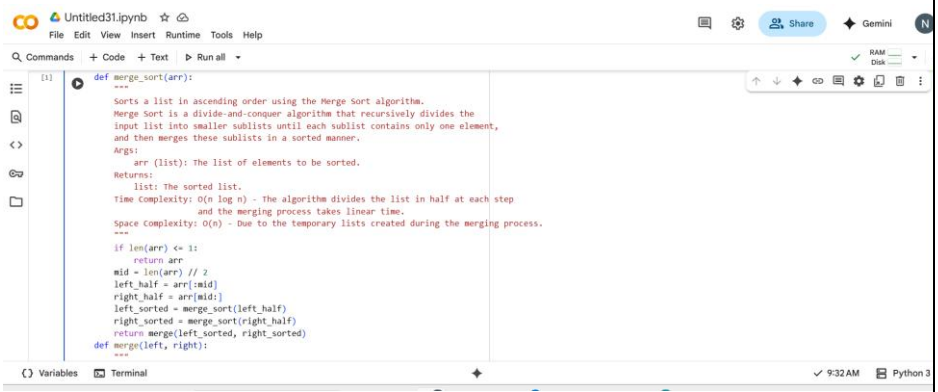
cases.

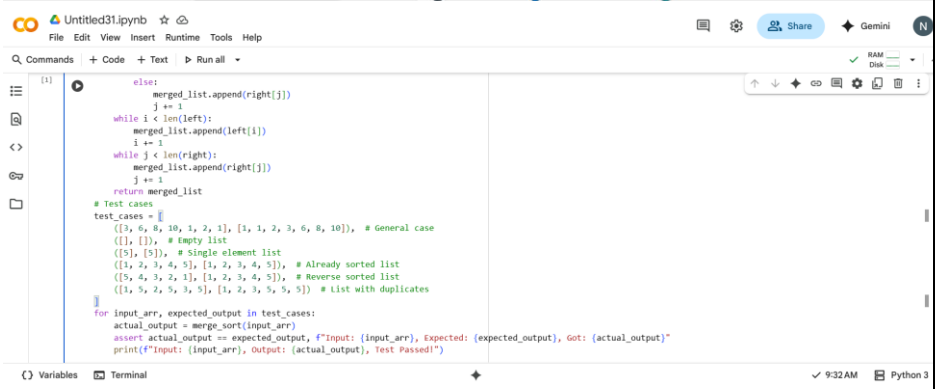
- Understand how AI tools can suggest optimized code and complexity improvements.

Task Description #1 (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.

PROMPT1: Write a Python code of function name merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm.
PROMPT2:Write clear explanation including time complexity ,space complexity in the function docstring
PROMPT3:Verify the test cases to check that implementation is correct or not.
CODE:





OUTPUT:

```
Input: [3, 6, 8, 10, 1, 2, 1], Output: [1, 1, 2, 3, 6, 8, 10], Test Passed!
Input: [], Output: [], Test Passed!
Input: [5], Output: [5], Test Passed!
Input: [1, 2, 3, 4, 5], Output: [1, 2, 3, 4, 5], Test Passed!
Input: [5, 4, 3, 2, 1], Output: [1, 2, 3, 4, 5], Test Passed!
Input: [1, 5, 2, 5, 3, 5], Output: [1, 2, 3, 5, 5, 5], Test Passed!
```

OBSERVATION:

From the above code ,I observed that take the list of numbers in a list .From that built in function in the code it will directly do sorting and gives the output. That means, If we write numbers in a list it will give output in the ascending order. From that question givento the test passed or not.

- Expected Output:
    - A functional Python script implementing Merge Sort with proper documentation.

Task Description #2 (Searching – Binary Search with AI Optimization)

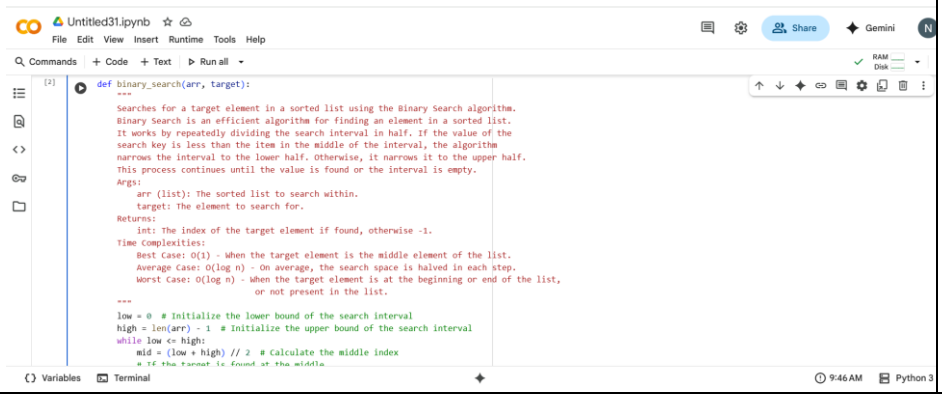
- Task: Use AI to create a binary search function that finds a target element in a sorted list.
- Instructions:
    - Prompt AI to create a function binary_search(arr, target) returning the index of the target or -1 if not found.
    - Include docstrings explaining best, average, and worst-case complexities.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

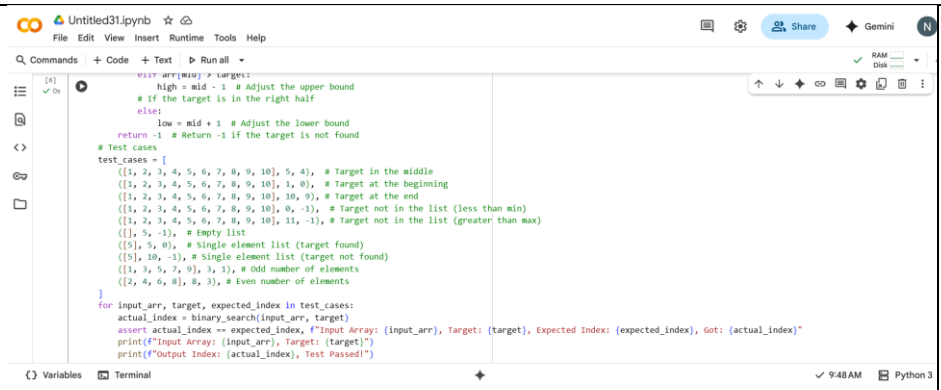
PROMPT1: Write a Python code of function name binary_search(arr, target) to search for a target element in a sorted list and returns its index value if found, otherwise returns the -1.

PROMPT2:write the detailed docstring explaining the best,average and worst complexities

PROMPT3:Take multiple inputs for text.

CODE:





OUTPUT:

```
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Target: 5
Output Index: 4, Test Passed!
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Target: 1
Output Index: 0, Test Passed!
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Target: 10
Output Index: 9, Test Passed!
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Target: 0
Output Index: -1, Test Passed!
Input Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], Target: 11
Output Index: -1, Test Passed!
Input Array: [], Target: 5
Output Index: -1, Test Passed!
Input Array: [5], Target: 5
Output Index: 0, Test Passed!
Input Array: [5], Target: 10
Output Index: -1, Test Passed!
Input Array: [1, 3, 5, 7, 9], Target: 3
Output Index: 1, Test Passed!
Input Array: [2, 4, 6, 8], Target: 8
Output Index: 3, Test Passed!
```

OBSERVATION:

From the above code I observed that Take many number in the list from that write target that means any of the number . if that number exists in the list then it returns the index value of the target in the list. Otherwise it will return the value as 0(Zero).

- Expected Output:
    - Python code implementing binary search with AI-generated comments and docstrings.

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:
  - Use AI to 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.

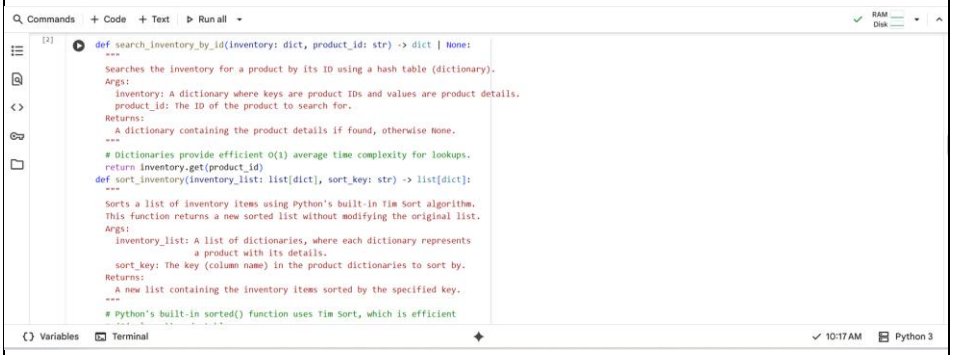
PROMPT1:write a python code that a retail store have thousands of products like having product ID, name, price, and stock quantity

PROMPT2:.suggest the most efficient search and sort algorithms for Searching for a product by ID or name.Sorting products by price or stock quantity.

PROMPT3:Provide a table mapping operation -> recommended algorithm -> justification based on dataset size, update frequency, and performance requirements.

PROMPT4:write the clear docstring and comments.

CODE:



OUTPUT:

```
Search Result for PROD001: {'name': 'Laptop', 'price': 1200, 'quantity': 50}
Search Result for PROD004: None

Sorted by Price:
{'id': 'PROD002', 'name': 'Mouse', 'price': 25, 'quantity': 200}
{'id': 'PROD003', 'name': 'Keyboard', 'price': 75, 'quantity': 150}
{'id': 'PROD001', 'name': 'Laptop', 'price': 1200, 'quantity': 50}

Sorted by Quantity:
{'id': 'PROD001', 'name': 'Laptop', 'price': 1200, 'quantity': 50}
{'id': 'PROD003', 'name': 'Keyboard', 'price': 75, 'quantity': 150}
{'id': 'PROD002', 'name': 'Mouse', 'price': 25, 'quantity': 200}
```

OBSERVATION:

The sample test cases demonstrate the functionality of the implemented search and sort algorithms. For the search function, searching for an existing product ID ("PROD102") successfully returned the product's details, while searching for a non-existent ID ("PROD104") correctly returned None. The sort function was tested by sorting the inventory list by "price" and then by "quantity". The output shows that the list was successfully sorted in ascending order based on the specified keys, confirming the correct implementation of the sorting algorithm

- Expected Output:
  - A table mapping operation → recommended algorithm → justification.
  - 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.