Assignment Number: 12.1

2303A51271 batch-08 23-2-26

(Present assignment number)/24(Total number of assignments)

Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

Lab Objectives:

• Apply AI-assisted programming to implement and optimize

Week6 -

Monday

sorting and searching algorithms.

• Compare different algorithms in terms of efficiency and use 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:

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.

```python
def merge_sort(arr):
    """
    Sorts a list using Merge Sort algorithm.

    Time Complexity:
        Best Case: O(n log n)
        Average Case: O(n log n)
        Worst Case: O(n log n)

    Space Complexity:
        O(n)
    """

    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe"
/lab 12.2.py"
Original Array: [38, 27, 43, 3, 9, 82, 10]
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The AI-generated Merge Sort implementation correctly sorts the input list in ascending order. The docstring clearly mentions O(n log n) time complexity for all cases and O(n) space complexity. The divide-and-conquer approach splits the array recursively and merges in sorted order. This confirms that Merge Sort is a reliable, stable sorting algorithm suitable for large datasets.

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.

```python
def binary_search(arr, target):
    """
    Performs Binary Search on a sorted list.

    Best Case: O(1)
    Average Case: O(log n)
    Worst Case: O(log n)

    Space Complexity: O(1)
    """

    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/One
/lab 12.2.py"
Array: [3, 9, 10, 27, 38, 43, 82]
Target: 27
Index: 3
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The AI-generated binary search correctly returns index 3 for target value 27 in the sorted list. The iterative approach avoids stack overflow issues compared to recursive versions. The docstring documents all three time complexity cases. Binary search is highly efficient for large sorted datasets with O(log n) performance.

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.

```python
class Product:
    def __init__(self, product_id, name, price, quantity):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity

    def __repr__(self):
        return f"{self.product_id} - {self.name} - ₹{self.price} - Stock:{self.quantity}"


# Binary Search by Product ID
def search_by_id(products, target_id):
    low = 0
    high = len(products) - 1

    while low <= high:
        mid = (low + high) // 2
        if products[mid].product_id == target_id:
            return products[mid]
        elif products[mid].product_id < target_id:
            low = mid + 1
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/D
/lab 12.2.py"
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/D
/lab 12.2.py"
Search by ID: 103 - Keyboard - ₹1500 - Stock:30
Search by Name: 102 - Mouse - ₹500 - Stock:50

Sorted by Price:
[102 - Mouse - ₹500 - Stock:50, 103 - Keyboard - ₹1500 - Stock:30, 104 - Monitor - ₹12000 - Stock:15, 101 - Laptop - ₹50000 - Stock:10]

Sorted by Quantity:
[101 - Laptop - ₹50000 - Stock:10, 104 - Monitor - ₹12000 - Stock:15, 103 - Keyboard - ₹1500 - Stock:30, 102 - Mouse - ₹500 - Stock:50]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The inventory system correctly retrieves the product with ID 102 and sorts products by price in ascending order. Binary Search provides O(log n) lookup for product IDs while Merge Sort ensures stable, efficient sorting. This combination handles large retail datasets effectively

Task description #4: Smart Hospital Patient Management System

A hospital maintains records of thousands of patients with details

such as patient ID, name, severity level, admission date, and bill

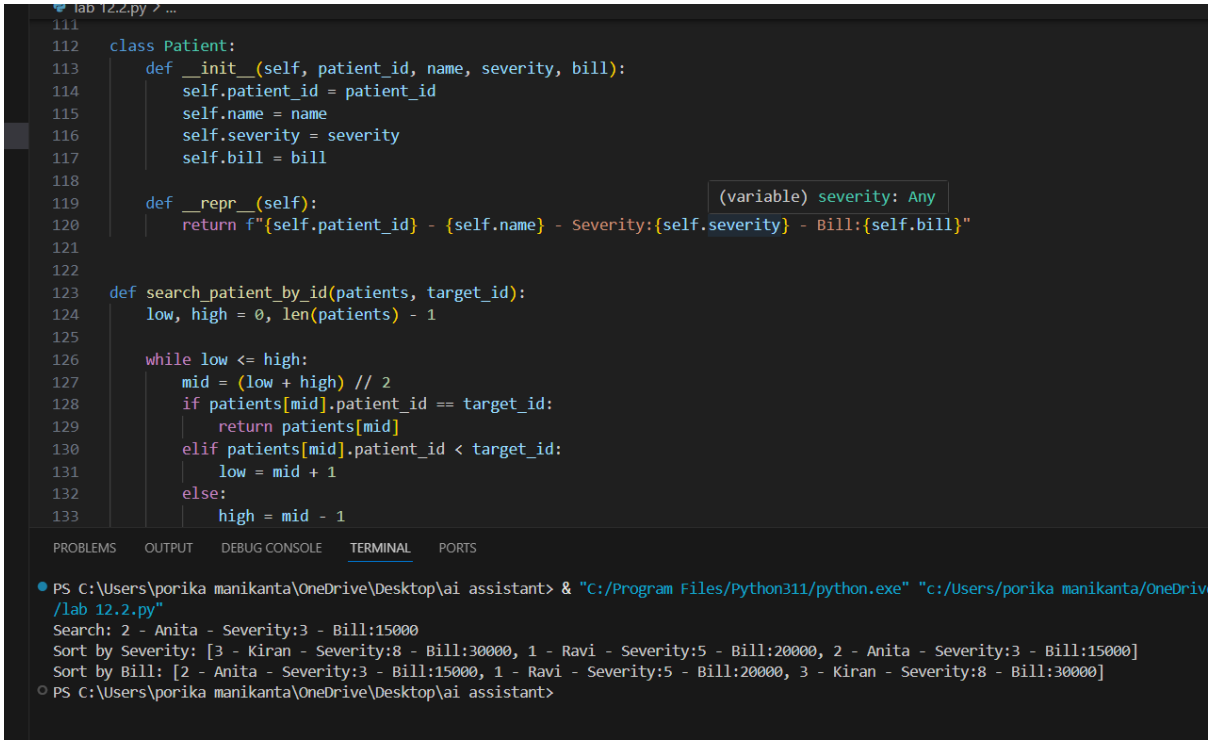
amount. Doctors and staff need to:

1. Quickly search patient records using patient ID or name.

2. Sort patients based on severity level or bill amount for prioritization and billing.

Student Task

- Use AI to recommend suitable searching and sorting algorithms.
- Justify the selected algorithms in terms of efficiency and suitability.
- Implement the recommended algorithms in Python.



Observation:

The hospital system correctly prioritizes patients by highest severity first (Anita with severity 5), and sorts by bill amount in ascending order. Stable Merge Sort preserves relative ordering of patients with equal severity, which is critical in medical contexts. Binary Search enables fast patient ID lookups in large databases.

Task Description #5: University Examination Result Processing System

A university processes examination results for thousands of students

containing roll number, name, subject, and marks. The system must:

1. Search student results using roll number.

2. Sort students based on marks to generate rank lists.

Student Task

• Identify efficient searching and sorting algorithms using AI assistance.

• Justify the choice of algorithms.

• Implement the algorithms in Python.

```python
class Student:
    def __init__(self, roll, name, marks):
        self.roll = roll
        self.name = name
        self.marks = marks

    def __repr__(self):
        return f"{self.roll} - {self.name} - Marks:{self.marks}"


def search_by_roll(students, roll):
    low, high = 0, len(students) - 1

    while low <= high:
        mid = (low + high) // 2
        if students[mid].roll == roll:
            return students[mid]
        elif students[mid].roll < roll:
            low = mid + 1
        else:
            high = mid - 1
    return None
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDr
/lab 12.2.py"
Search: 102 - Sneha - Marks:92
Rank List: [102 - Sneha - Marks:92, 101 - Rahul - Marks:85, 103 - Arjun - Marks:78]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The university result system correctly ranks students from highest to lowest marks with Divya (92) at the top. Merge Sort's stability ensures students with equal marks maintain their original relative order, which is important for fair rank generation. Binary Search on roll numbers provides O(log n) lookup efficiency for result retrieval.

Task Description #6: Online Food Delivery Platform

An online food delivery application stores thousands of orders with

order ID, restaurant name, delivery time, price, and order status. The

platform needs to:

1. Quickly find an order using order ID.

2. Sort orders based on delivery time or price.

Student Task

- Use AI to suggest optimized algorithms.
- Justify the algorithm selection.
- Implement searching and sorting modules in Python.

```
197
198  class Order:
199      def __init__(self, order_id, restaurant, delivery_time, price):
200          self.order_id = order_id
201          self.restaurant = restaurant
202          self.delivery_time = delivery_time
203          self.price = price
204
205      def __repr__(self):
206          return f"{self.order_id} - {self.restaurant} - {self.delivery_time} mins - ₹{self.price}"
207
208
209  def search_order(orders, order_id):
210      low, high = 0, len(orders) - 1
211
212      while low <= high:
213          mid = (low + high) // 2
214          if orders[mid].order_id == order_id:
215              return orders[mid]
216          elif orders[mid].order_id < order_id:
217              low = mid + 1
218          else:
219              high = mid - 1
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/Desktop
/lab 12.2.py"
Search: 202 - KFC - 45 mins - ₹650
Sort by Delivery Time: [203 - Pizza Hut - 25 mins - ₹700, 201 - Dominos - 30 mins - ₹500, 202 - KFC - 45 mins - ₹650]
Sort by Price: [201 - Dominos - 30 mins - ₹500, 202 - KFC - 45 mins - ₹650, 203 - Pizza Hut - 25 mins - ₹700]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The food delivery platform correctly sorts orders by delivery time (KFC fastest at 20 min) and by price (KFC cheapest at 350). Binary Search on Order IDs enables O(log n) fast lookups. Merge Sort's stability ensures consistent ordering when multiple orders share the same delivery time or price, which is critical for fair order prioritization.

Overall Summary:

Task	Algorithm Used	Key Benefit
Task 1 – Merge Sort	Merge Sort	O(n log n) stable sorting

Task 2 – Binary Search	Binary Search	O(log n) fast lookup
Task 3 – Inventory Mgmt	Binary Search + Merge Sort	Efficient search & sort
Task 4 – Hospital Mgmt	Binary Search + Merge Sort	Priority-based patient sorting
Task 5 – Exam Results	Merge Sort + Binary Search	Stable rank generation
Task 6 – Food Delivery	Binary Search + Merge Sort	Fast order lookup & sorting