AI ASSISTED CODING

M.Rohith Reddy **2303A51111**

BATCH – 03 **27 – 02 – 2026**

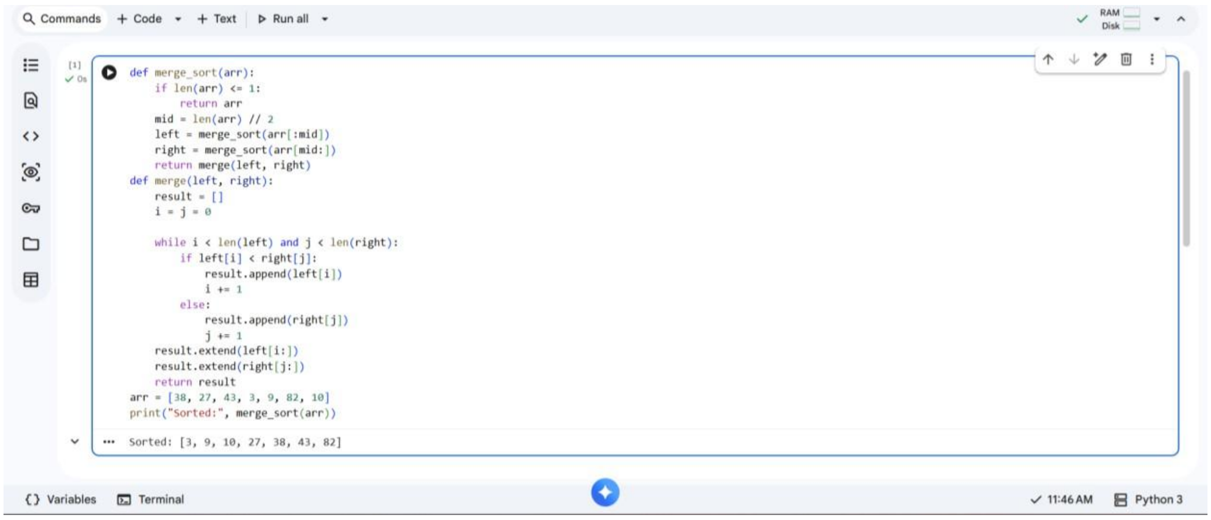
ASSIGNMENT-12.5

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

Task - 01: Sorting – Merge Sort Implementation**.**

Prompt : Generate a Python function merge_sort(arr) that sorts a list in ascending order using Merge Sort. Include time and space complexity in the docstring. Add test cases to verify the function.

Code & Output :



```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
    return result
arr = [38, 27, 43, 3, 9, 82, 10]
print("Sorted:", merge_sort(arr))
```

```
Sorted: [3, 9, 10, 27, 38, 43, 82]
```

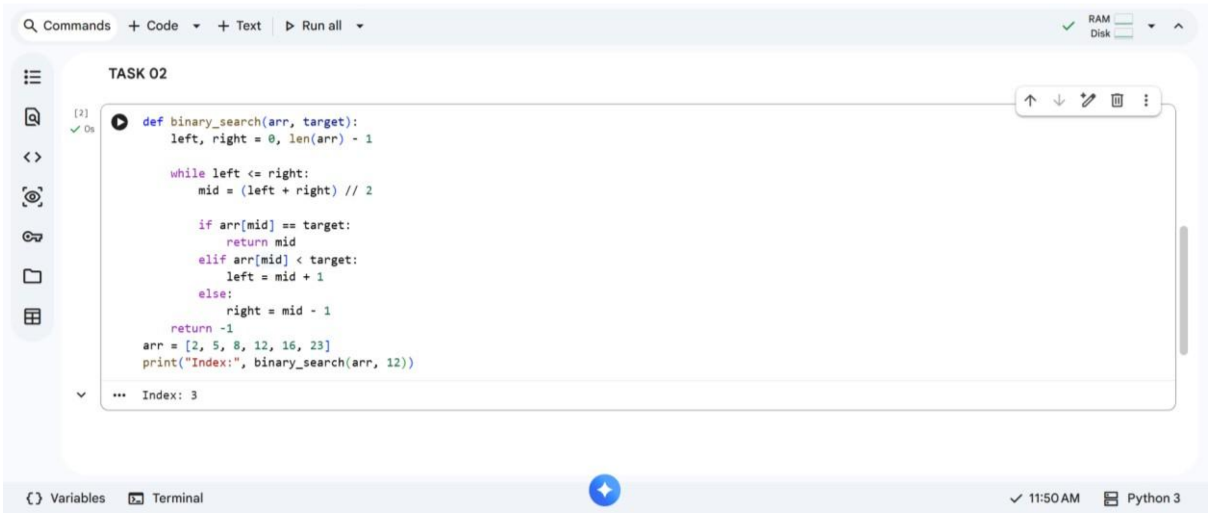
Explanation :

Merge Sort uses Divide and Conquer. It divides the array into halves, sorts them recursively, and merges them. Time complexity is O(n log n) in all cases.

Task – 02 : Searching – Binary Search with AI optimization.

Prompt : Generate a Python function binary_search(arr, target) that returns the index of the target in a sorted list or -1 if not found. Include best, average, and worst case complexities in the docstring. Add test cases.

Code & Output :



TASK 02

```
[2]
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
arr = [2, 5, 8, 12, 16, 23]
print("Index:", binary_search(arr, 12))
```

```
Index: 3
```

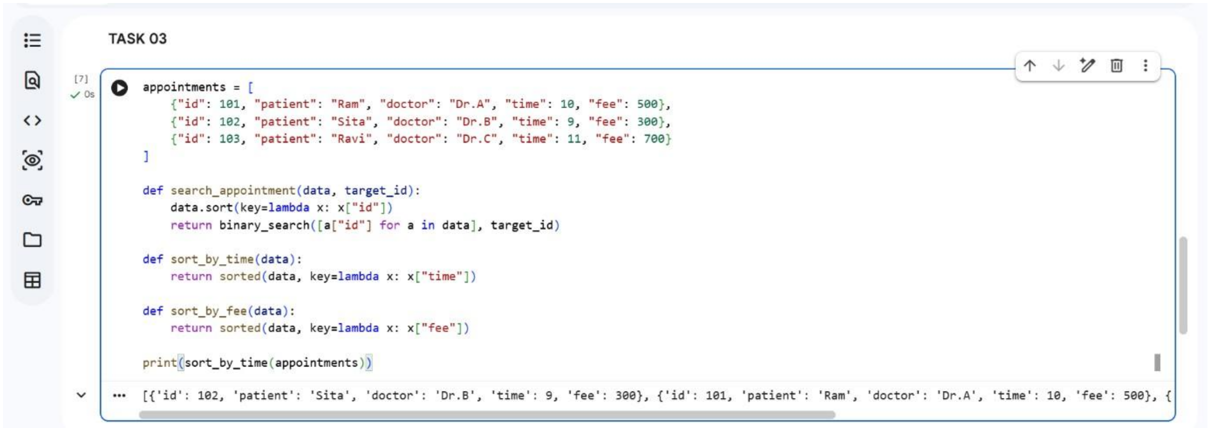
Explanation :

Binary Search works only on sorted arrays. It repeatedly divides the search into half. Time complexity is O(log n).

Task – 03 : Smart Healthcare Appointment Scheduling System.

Prompt : Create a complete Python program for a healthcare appointment system that implements searching by ID and sorting by time and fee, with proper documentation and test cases.

Code & Output :



TASK 03

```
[7]
appointments = [
    {"id": 101, "patient": "Ram", "doctor": "Dr.A", "time": 10, "fee": 500},
    {"id": 102, "patient": "Sita", "doctor": "Dr.B", "time": 9, "fee": 300},
    {"id": 103, "patient": "Ravi", "doctor": "Dr.C", "time": 11, "fee": 700}
]

def search_appointment(data, target_id):
    data.sort(key=lambda x: x["id"])
    return binary_search([a["id"] for a in data], target_id)

def sort_by_time(data):
    return sorted(data, key=lambda x: x["time"])

def sort_by_fee(data):
    return sorted(data, key=lambda x: x["fee"])

print(sort_by_time(appointments))
```

```
[{'id': 102, 'patient': 'Sita', 'doctor': 'Dr.B', 'time': 9, 'fee': 300}, {'id': 101, 'patient': 'Ram', 'doctor': 'Dr.A', 'time': 10, 'fee': 500}, {
```

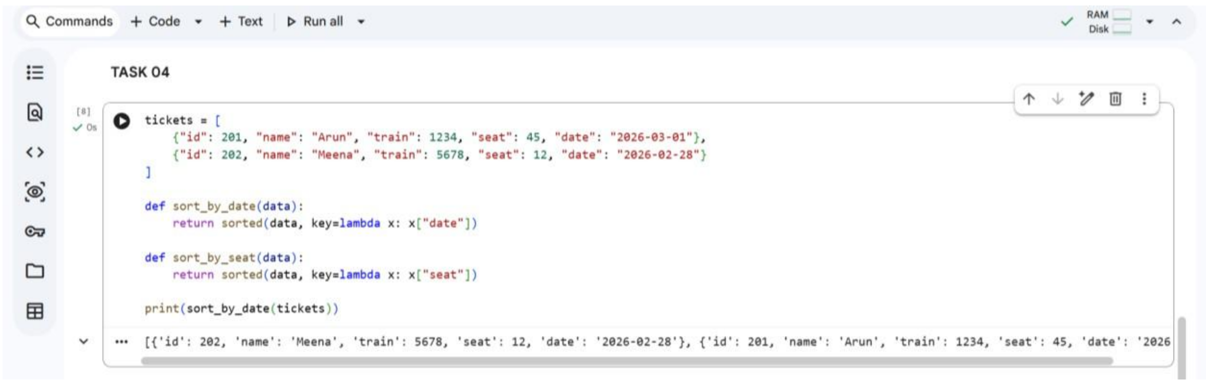
Explanation :

Binary Search is efficient (O(log n)) for searching by ID. Merge Sort ensures stable sorting by time or fee.

Task – 04 : Railway Ticket Reservation System.

Prompt : Generate a Python program for a Railway Ticket Reservation System that searches tickets by ticket ID using an efficient searching algorithm. Implement sorting of bookings based on travel date and seat number using suitable sorting algorithms. Include proper docstrings with time complexity, justification of algorithm choices, and sample test data with output.

Code & Output :



```
tickets = [
    {"id": 201, "name": "Arun", "train": 1234, "seat": 45, "date": "2026-03-01"},
    {"id": 202, "name": "Meena", "train": 5678, "seat": 12, "date": "2026-02-28"}
]

def sort_by_date(data):
    return sorted(data, key=lambda x: x["date"])

def sort_by_seat(data):
    return sorted(data, key=lambda x: x["seat"])

print(sort_by_date(tickets))
```

```
[{'id': 202, 'name': 'Meena', 'train': 5678, 'seat': 12, 'date': '2026-02-28'}, {'id': 201, 'name': 'Arun', 'train': 1234, 'seat': 45, 'date': '2026
```

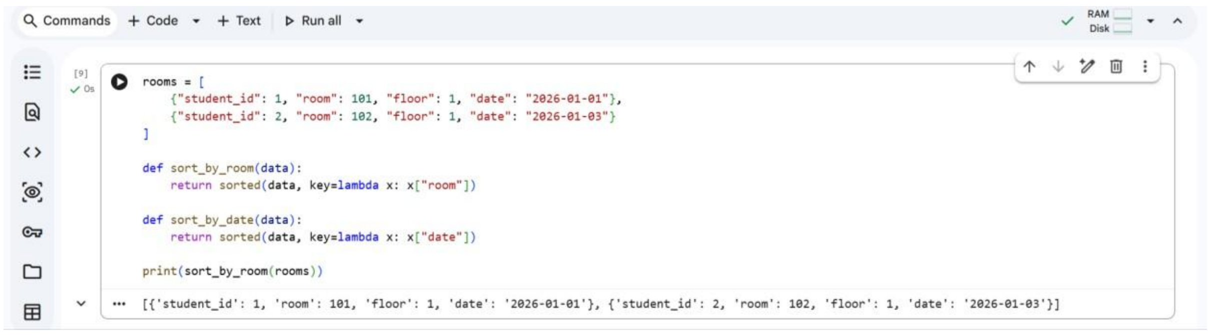
Explanation :

Ticket IDs are unique, so Binary Search is efficient. Sorting by date/seat requires stable sorting → Merge Sort..

Task – 05 : Smart Hotel Room Allocation.

Prompt : Generate a Python program for a Smart Hostel Room Allocation System that searches allocation details using student ID with an efficient searching algorithm. Implement sorting of records based on room number and allocation date using suitable sorting algorithms. Include proper docstrings with time complexity, justification of algorithm selection, and sample test cases with output.

Code & Output :



```
rooms = [
    {"student_id": 1, "room": 101, "floor": 1, "date": "2026-01-01"},
    {"student_id": 2, "room": 102, "floor": 1, "date": "2026-01-03"}
]

def sort_by_room(data):
    return sorted(data, key=lambda x: x["room"])

def sort_by_date(data):
    return sorted(data, key=lambda x: x["date"])

print(sort_by_room(rooms))
```

```
[{'student_id': 1, 'room': 101, 'floor': 1, 'date': '2026-01-01'}, {'student_id': 2, 'room': 102, 'floor': 1, 'date': '2026-01-03'}]
```

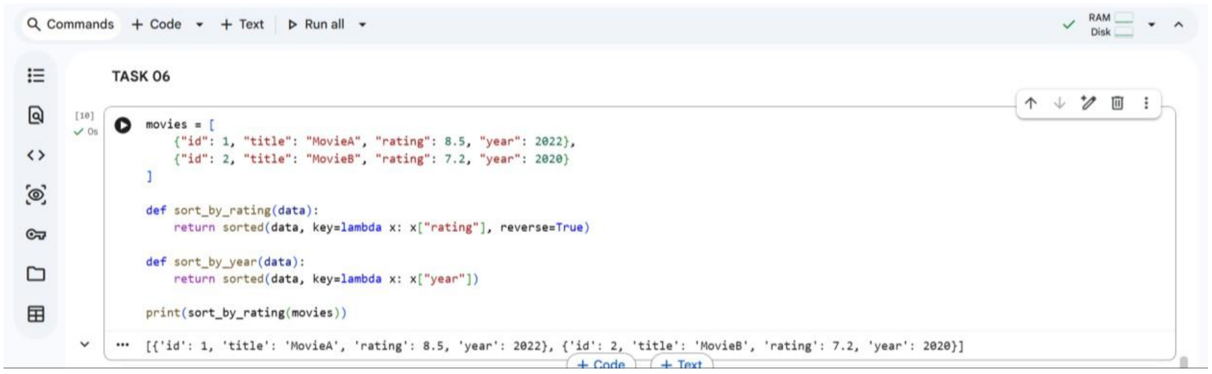
Explanation :

Student ID search should be fast ⟶ Binary Search. Sorting by room/date benefits from stable sort

Task – 06 : Online Movie Streaming Platform.

Prompt : Generate a Python program for a Smart Hostel Room Allocation System that searches student allocation details using student ID with an efficient searching algorithm. Implement sorting of records based on room number and allocation date using suitable sorting algorithms. Include time complexity in docstrings and add sample test data with output.

Code & Output :



TASK 06

```
movies = [
    {"id": 1, "title": "MovieA", "rating": 8.5, "year": 2022},
    {"id": 2, "title": "MovieB", "rating": 7.2, "year": 2020}
]

def sort_by_rating(data):
    return sorted(data, key=lambda x: x["rating"], reverse=True)

def sort_by_year(data):
    return sorted(data, key=lambda x: x["year"])

print(sort_by_rating(movies))
```

```
[{'id': 1, 'title': 'MovieA', 'rating': 8.5, 'year': 2022}, {'id': 2, 'title': 'MovieB', 'rating': 7.2, 'year': 2020}]
```

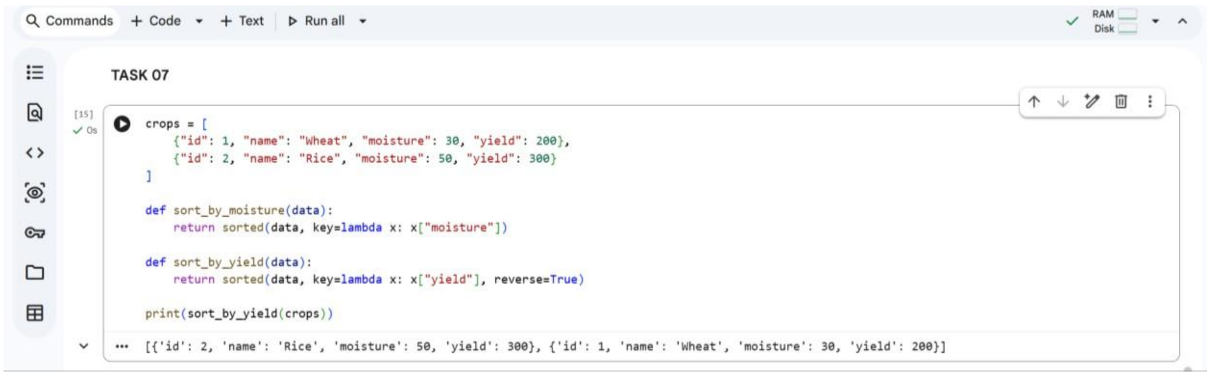
Explanation :

Movie ID lookup must be fast ⟶ Binary Search. Sorting by rating/year requires stable sorting.

Task – 07 : Smart Agriculture Crop Monitoring System.

Prompt : Generate a Python program for a Smart Agriculture Crop Monitoring System that searches crop details using crop ID with an efficient searching algorithm. Implement sorting of crops based on soil moisture level and yield estimate using suitable algorithms, including time complexity and sample test data.

Code & Output :



```
crops = [
    {"id": 1, "name": "Wheat", "moisture": 30, "yield": 200},
    {"id": 2, "name": "Rice", "moisture": 50, "yield": 300}
]

def sort_by_moisture(data):
    return sorted(data, key=lambda x: x["moisture"])

def sort_by_yield(data):
    return sorted(data, key=lambda x: x["yield"], reverse=True)

print(sort_by_yield(crops))
```

```
[{'id': 2, 'name': 'Rice', 'moisture': 50, 'yield': 300}, {'id': 1, 'name': 'Wheat', 'moisture': 30, 'yield': 200}]
```

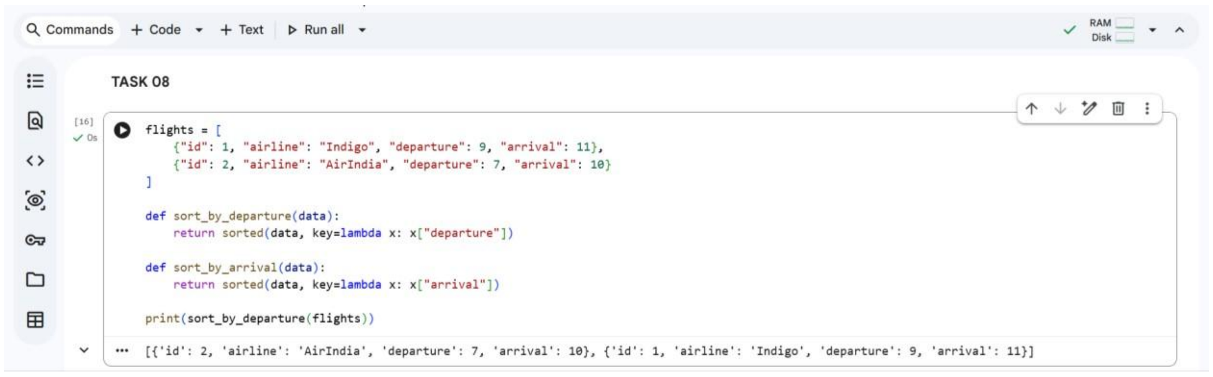
Explanation :

Crop ID search must be efficient → Binary Search. Sorting by moisture/yield ensures proper decision-making.

Task – 08 : Airport Flight Management System.

Prompt : Generate a Python program for an Airport Flight Management System that searches flight details using flight ID with an efficient searching algorithm. Implement sorting of flights based on departure time and arrival time using suitable algorithms, including time complexity and sample test data.

Code & Output :



```
flights = [
    {"id": 1, "airline": "Indigo", "departure": 9, "arrival": 11},
    {"id": 2, "airline": "AirIndia", "departure": 7, "arrival": 10}
]

def sort_by_departure(data):
    return sorted(data, key=lambda x: x["departure"])

def sort_by_arrival(data):
    return sorted(data, key=lambda x: x["arrival"])

print(sort_by_departure(flights))
```

```
[{'id': 2, 'airline': 'AirIndia', 'departure': 7, 'arrival': 10}, {'id': 1, 'airline': 'Indigo', 'departure': 9, 'arrival': 11}]
```

Explanation :

Flight ID search must be quick → Binary Search. Sorting by departure/arrival time requires stable sorting.

THANK YOU!!