SR UNIVERSITY

## School of Computer Science and Artificial Intelligence

## Lab Assignment # 12.5

| | |
|---|---|
| **Program** | **: B. Tech (CSE)** |
| **Specialization** | **: -** |
| **Course Title** | **: AI Assisted Coding** |
| **Course Code** | **: 23CS002PC304** |
| **Semester** | **II** |
| **Academic Session** | **: 2025-2026** |
| **Name of Student** | **: P.Eshwar** |
| **Enrollment No.** | **: 2403A51L26** |
| **Batch No.** | **51** |
| **Date** | **: 20/02/26** |

## Submission Starts here

### Screenshots:

### Task Description #1

(Sorting – Merge Sort Implementation)

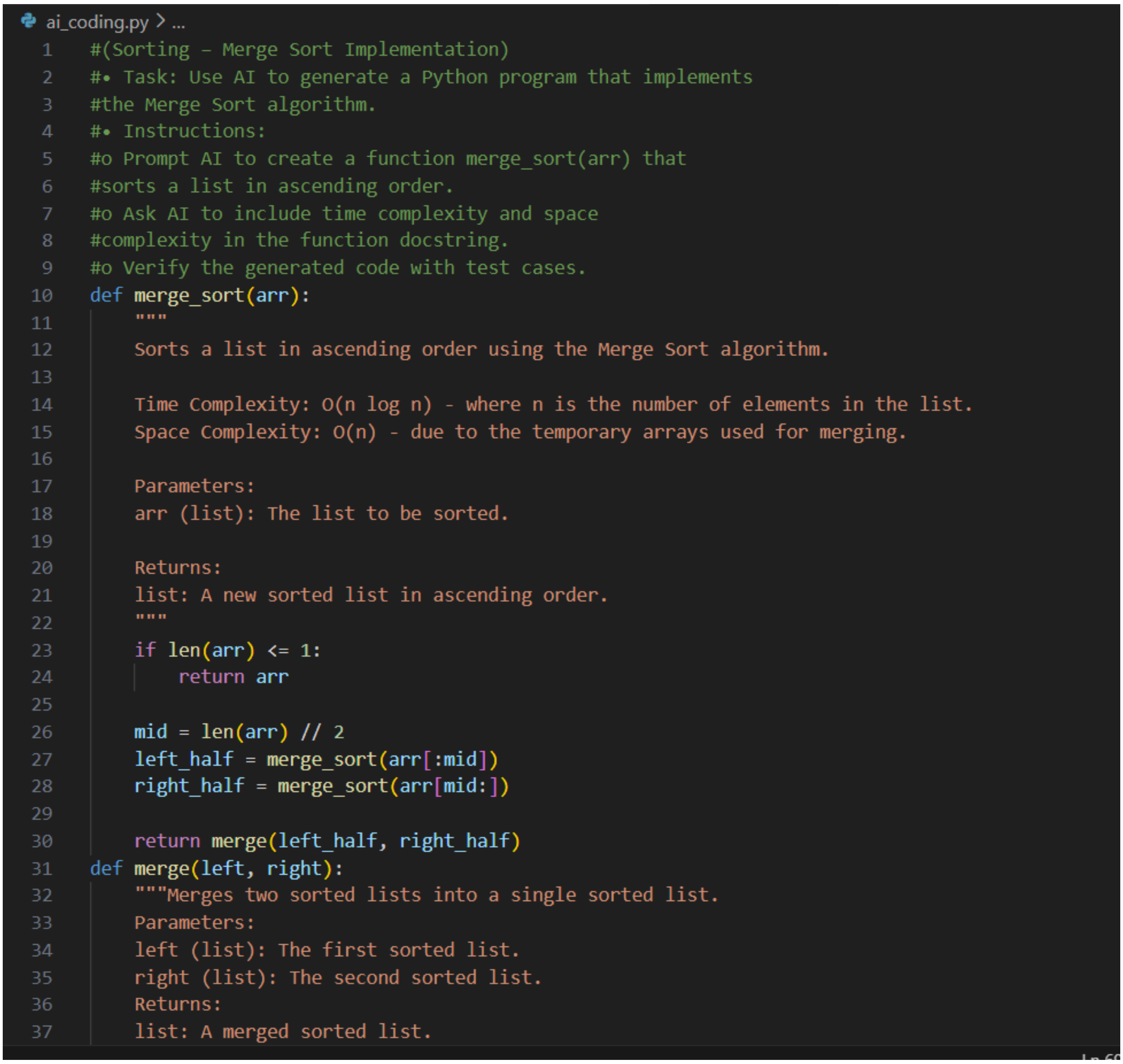
• **Task:** Use AI to generate a Python program that implements the Merge Sort algorithm.

• **Instructions:**

o Prompt AI to create a function merge_sort(arr) that sorts a list in ascending order.

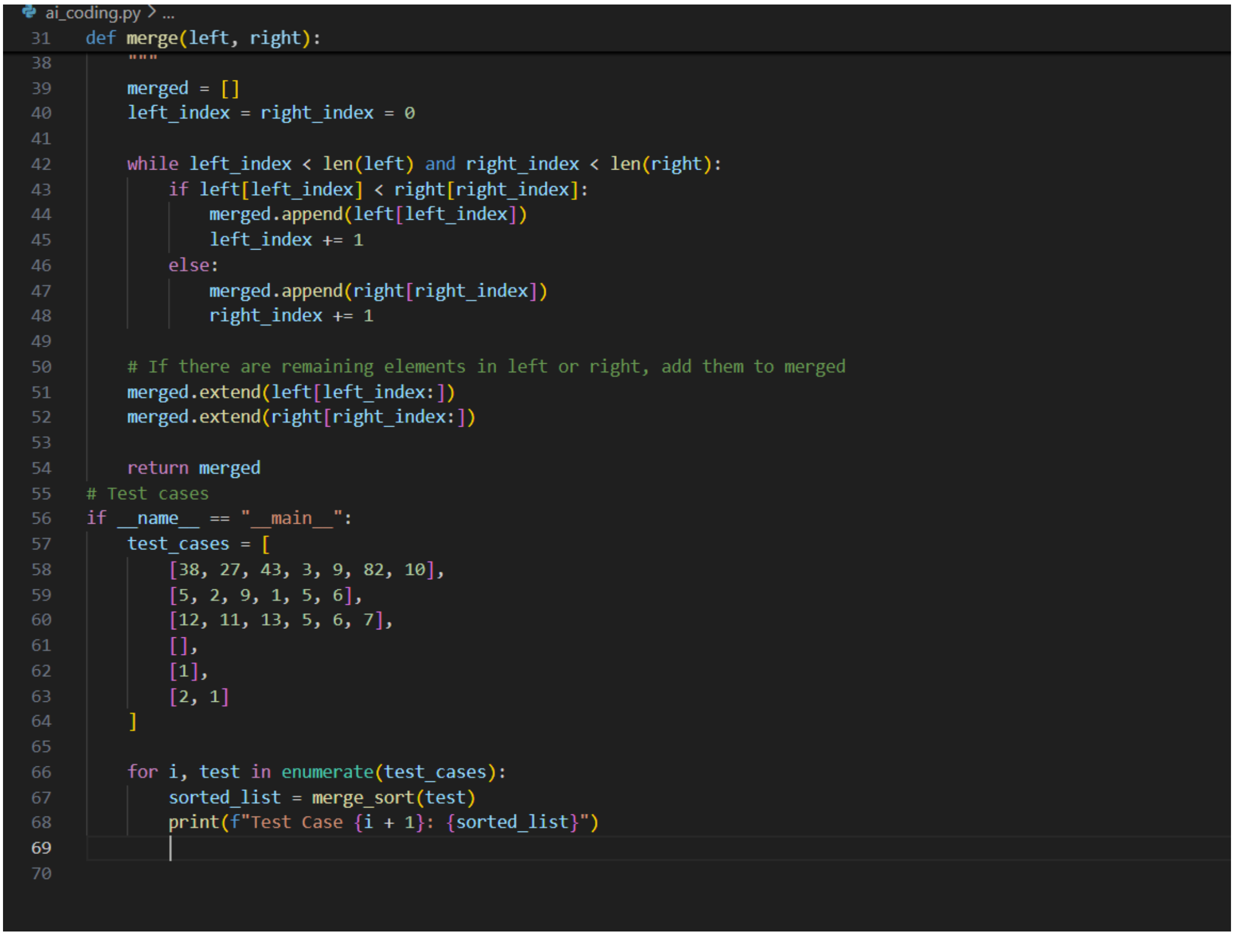
o Ask AI to include time complexity and space complexity in the function docstring.

o Verify the generated code with test cases.

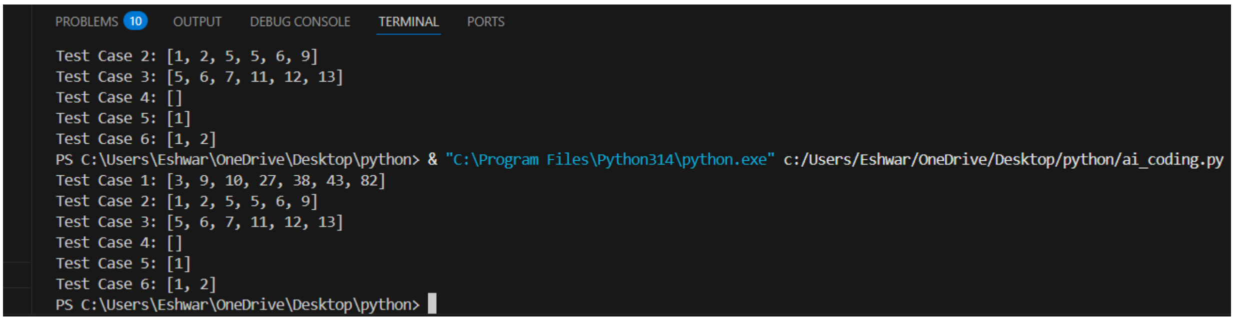


ai_coding.py > ...

```python
#(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.
def merge_sort(arr):
    """
    Sorts a list in ascending order using the Merge Sort algorithm.

    Time Complexity: O(n log n) - where n is the number of elements in the list.
    Space Complexity: O(n) - due to the temporary arrays used for merging.

    Parameters:
    arr (list): The list to be sorted.

    Returns:
    list: A new sorted list in ascending order.
    """
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    return merge(left_half, right_half)
def merge(left, right):
    """Merges two sorted lists into a single sorted list.
    Parameters:
    left (list): The first sorted list.
    right (list): The second sorted list.
    Returns:
    list: A merged sorted list.
```



```python
def merge(left, right):
    """
    merged = []
    left_index = right_index = 0

    while left_index < len(left) and right_index < len(right):
        if left[left_index] < right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1

    # If there are remaining elements in left or right, add them to merged
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])

    return merged
# Test cases
if __name__ == "__main__":
    test_cases = [
        [38, 27, 43, 3, 9, 82, 10],
        [5, 2, 9, 1, 5, 6],
        [12, 11, 13, 5, 6, 7],
        [],
        [1],
        [2, 1]
    ]

    for i, test in enumerate(test_cases):
        sorted_list = merge_sort(test)
        print(f"Test Case {i + 1}: {sorted_list}")
        
```



```
Test Case 2: [1, 2, 5, 5, 6, 9]
Test Case 3: [5, 6, 7, 11, 12, 13]
Test Case 4: []
Test Case 5: [1]
Test Case 6: [1, 2]
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
Test Case 1: [3, 9, 10, 27, 38, 43, 82]
Test Case 2: [1, 2, 5, 5, 6, 9]
Test Case 3: [5, 6, 7, 11, 12, 13]
Test Case 4: []
Test Case 5: [1]
Test Case 6: [1, 2]
PS C:\Users\Eshwar\OneDrive\Desktop\python> 
```

**Task Description #2**

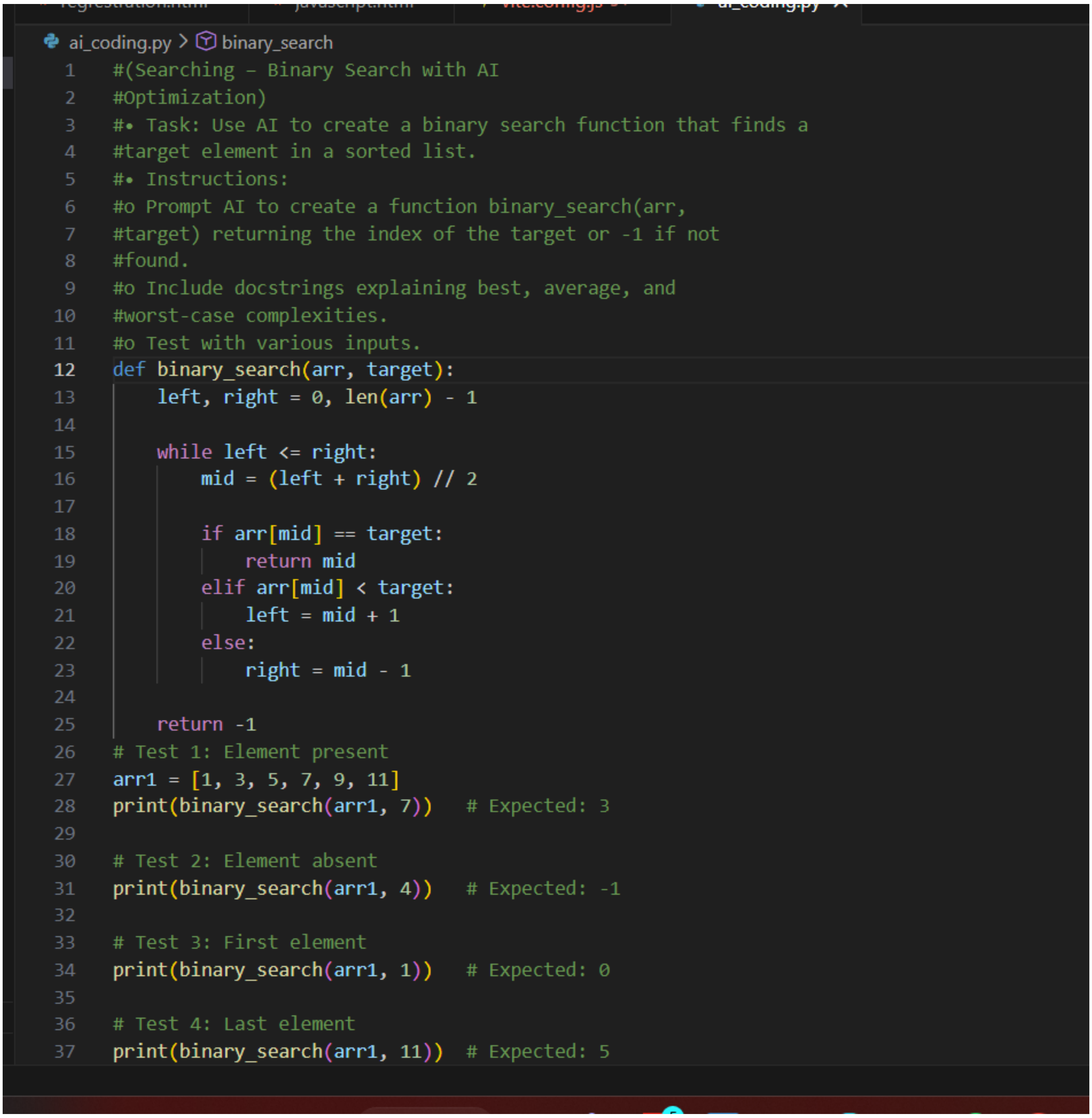
(Searching – Binary Search with AI Optimization)

- **Task:** Use AI to create a binary search function that finds a target element in a sorted list.
- **Instructions:**

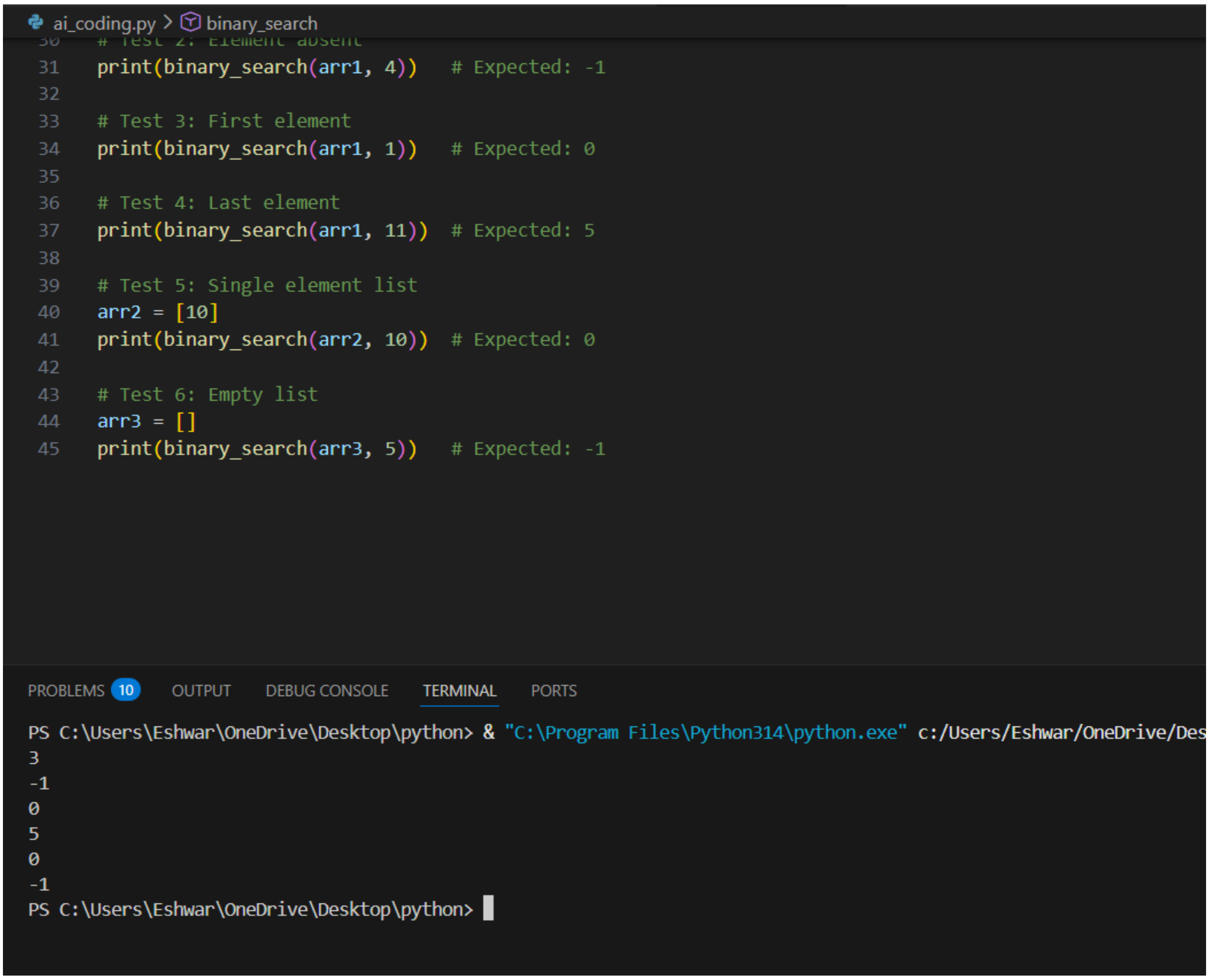
o Prompt AI to create a function binary_search(arr, target) returning the index of the target or -1 if not found.

o Include docstrings explaining best, average, and worst-case complexities.

o Test with various inputs.



```python
#(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.
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
# Test 1: Element present
arr1 = [1, 3, 5, 7, 9, 11]
print(binary_search(arr1, 7))   # Expected: 3

# Test 2: Element absent
print(binary_search(arr1, 4))   # Expected: -1

# Test 3: First element
print(binary_search(arr1, 1))   # Expected: 0

# Test 4: Last element
print(binary_search(arr1, 11))  # Expected: 5
```



```python
# Test 2: Element absent
print(binary_search(arr1, 4))   # Expected: -1

# Test 3: First element
print(binary_search(arr1, 1))   # Expected: 0

# Test 4: Last element
print(binary_search(arr1, 11))  # Expected: 5

# Test 5: Single element list
arr2 = [10]
print(binary_search(arr2, 10))  # Expected: 0

# Test 6: Empty list
arr3 = []
print(binary_search(arr3, 5))   # Expected: -1
```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Des
3
-1
0
5
0
-1
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

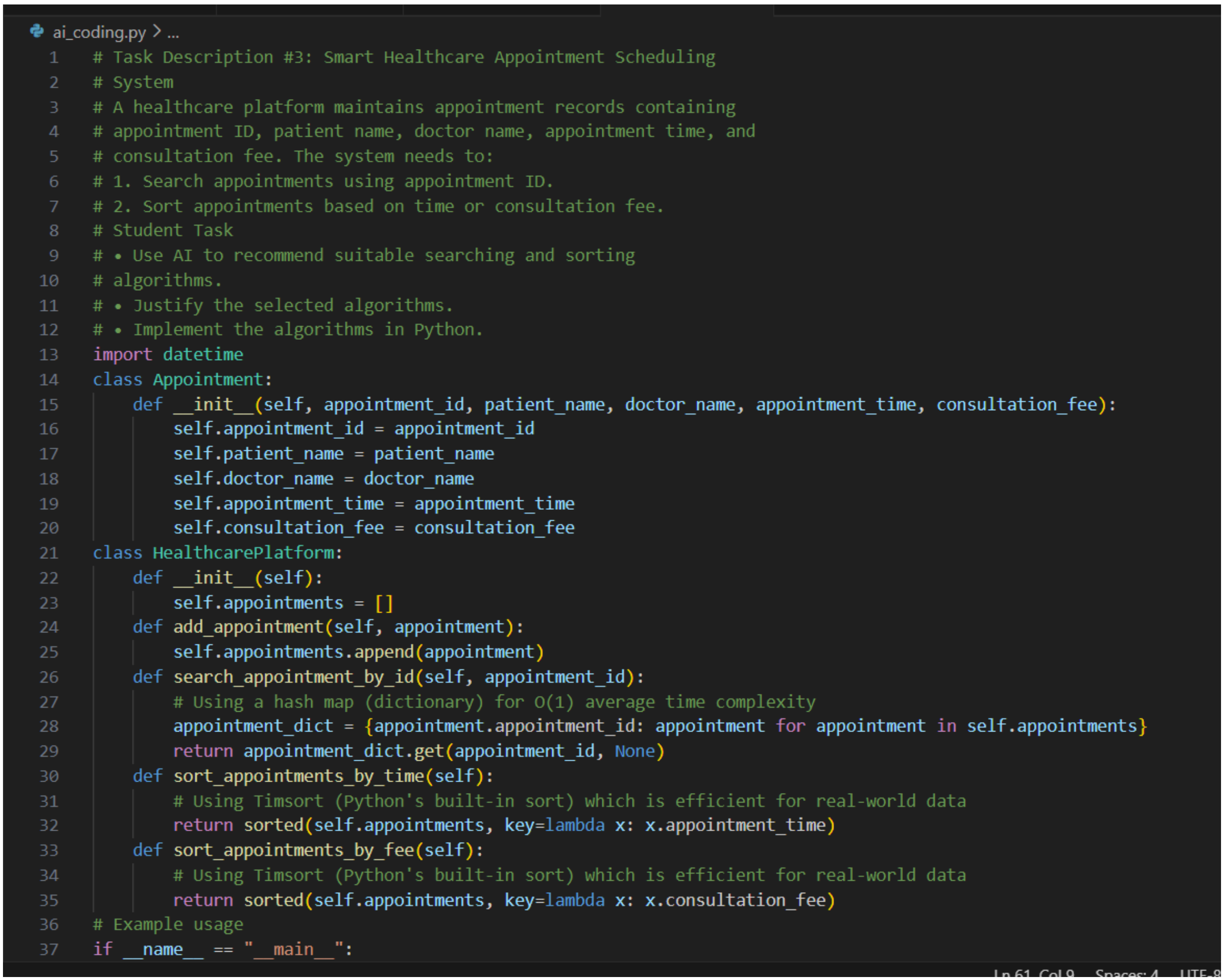
**Task Description #3:** Smart Healthcare Appointment Scheduling System

A healthcare platform maintains appointment records containing appointment ID, patient name, doctor name, appointment time, and consultation fee. The system needs to:

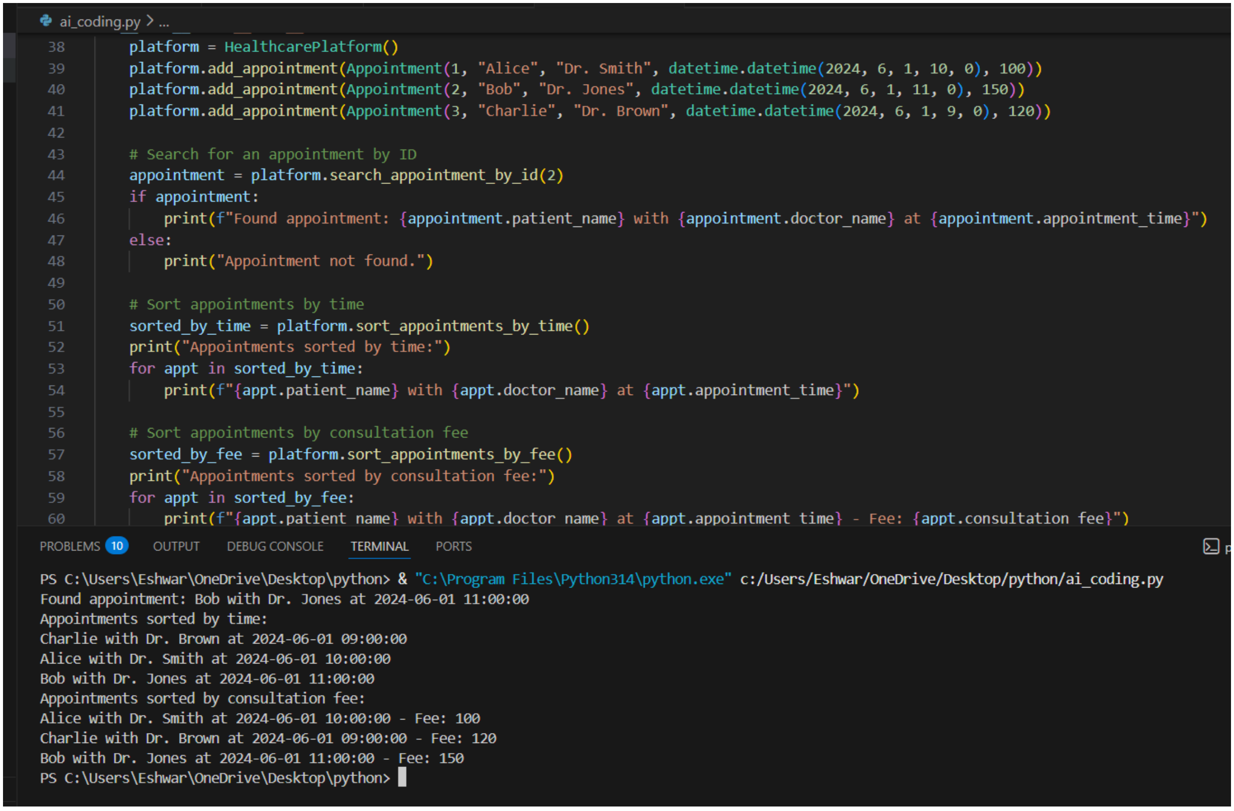
1. Search appointments using appointment ID.
2. Sort appointments based on time or consultation fee.

Student Task

- Use AI to recommend suitable searching and sorting algorithms.
- Justify the selected algorithms.
- Implement the algorithms in Python.



```python
# Task Description #3: Smart Healthcare Appointment Scheduling
# System
# A healthcare platform maintains appointment records containing
# appointment ID, patient name, doctor name, appointment time, and
# consultation fee. The system needs to:
# 1. Search appointments using appointment ID.
# 2. Sort appointments based on time or consultation fee.
# Student Task
# • Use AI to recommend suitable searching and sorting
# algorithms.
# • Justify the selected algorithms.
# • Implement the algorithms in Python.
import datetime
class Appointment:
    def __init__(self, appointment_id, patient_name, doctor_name, appointment_time, consultation_fee):
        self.appointment_id = appointment_id
        self.patient_name = patient_name
        self.doctor_name = doctor_name
        self.appointment_time = appointment_time
        self.consultation_fee = consultation_fee
class HealthcarePlatform:
    def __init__(self):
        self.appointments = []
    def add_appointment(self, appointment):
        self.appointments.append(appointment)
    def search_appointment_by_id(self, appointment_id):
        # Using a hash map (dictionary) for O(1) average time complexity
        appointment_dict = {appointment.appointment_id: appointment for appointment in self.appointments}
        return appointment_dict.get(appointment_id, None)
    def sort_appointments_by_time(self):
        # Using Timsort (Python's built-in sort) which is efficient for real-world data
        return sorted(self.appointments, key=lambda x: x.appointment_time)
    def sort_appointments_by_fee(self):
        # Using Timsort (Python's built-in sort) which is efficient for real-world data
        return sorted(self.appointments, key=lambda x: x.consultation_fee)
# Example usage
if __name__ == "__main__":
```



```python
    platform = HealthcarePlatform()
    platform.add_appointment(Appointment(1, "Alice", "Dr. Smith", datetime.datetime(2024, 6, 1, 10, 0), 100))
    platform.add_appointment(Appointment(2, "Bob", "Dr. Jones", datetime.datetime(2024, 6, 1, 11, 0), 150))
    platform.add_appointment(Appointment(3, "Charlie", "Dr. Brown", datetime.datetime(2024, 6, 1, 9, 0), 120))

    # Search for an appointment by ID
    appointment = platform.search_appointment_by_id(2)
    if appointment:
        print(f"Found appointment: {appointment.patient_name} with {appointment.doctor_name} at {appointment.appointment_time}")
    else:
        print("Appointment not found.")

    # Sort appointments by time
    sorted_by_time = platform.sort_appointments_by_time()
    print("Appointments sorted by time:")
    for appt in sorted_by_time:
        print(f"{appt.patient_name} with {appt.doctor_name} at {appt.appointment_time}")

    # Sort appointments by consultation fee
    sorted_by_fee = platform.sort_appointments_by_fee()
    print("Appointments sorted by consultation fee:")
    for appt in sorted_by_fee:
        print(f"{appt.patient_name} with {appt.doctor_name} at {appt.appointment_time} - Fee: {appt.consultation_fee}")
```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
Found appointment: Bob with Dr. Jones at 2024-06-01 11:00:00
Appointments sorted by time:
Charlie with Dr. Brown at 2024-06-01 09:00:00
Alice with Dr. Smith at 2024-06-01 10:00:00
Bob with Dr. Jones at 2024-06-01 11:00:00
Appointments sorted by consultation fee:
Alice with Dr. Smith at 2024-06-01 10:00:00 - Fee: 100
Charlie with Dr. Brown at 2024-06-01 09:00:00 - Fee: 120
Bob with Dr. Jones at 2024-06-01 11:00:00 - Fee: 150
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

**Task Description #4:** Railway Ticket Reservation System

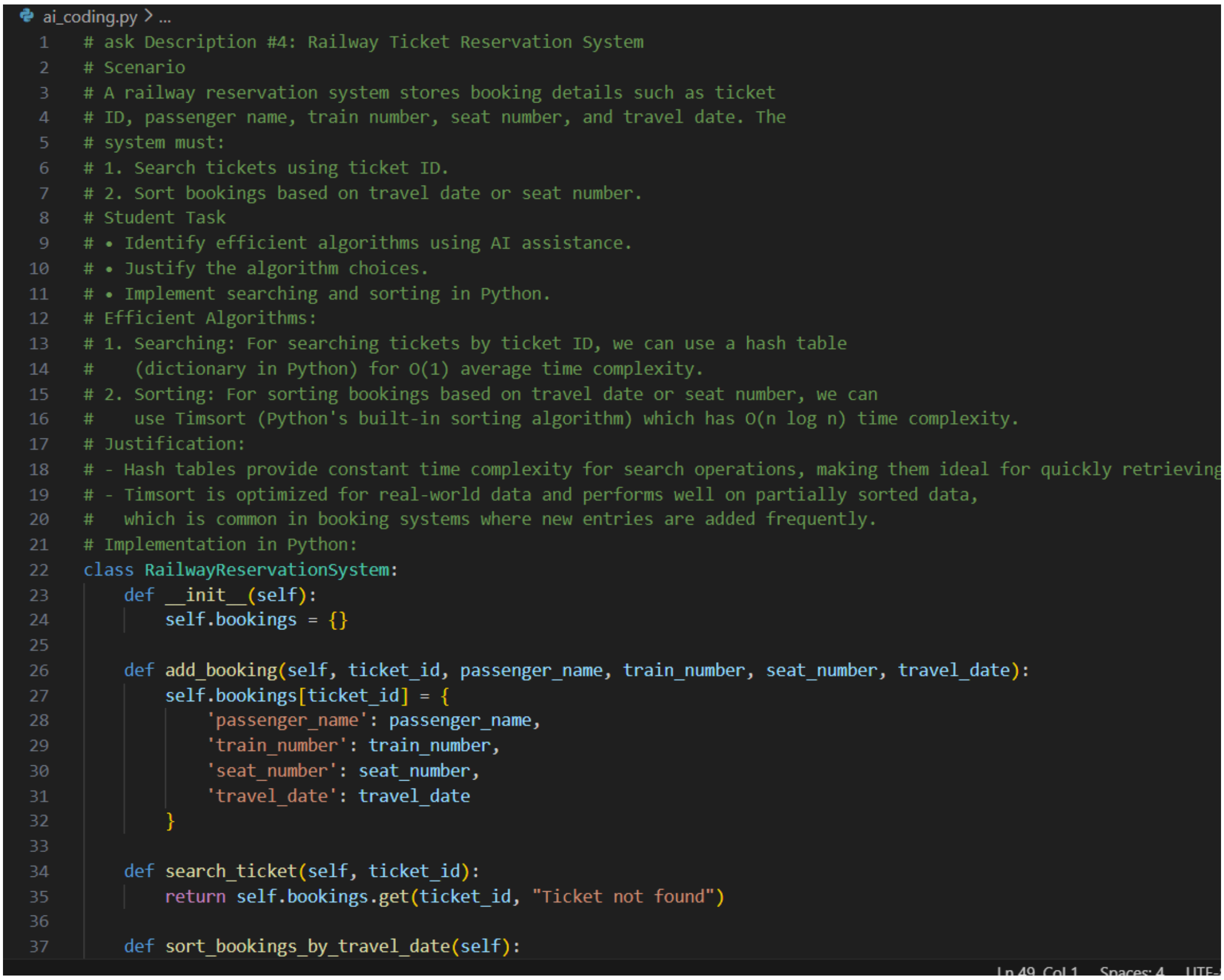
Scenario

A railway reservation system stores booking details such as ticket ID, passenger name, train number, seat number, and travel date. The system must:

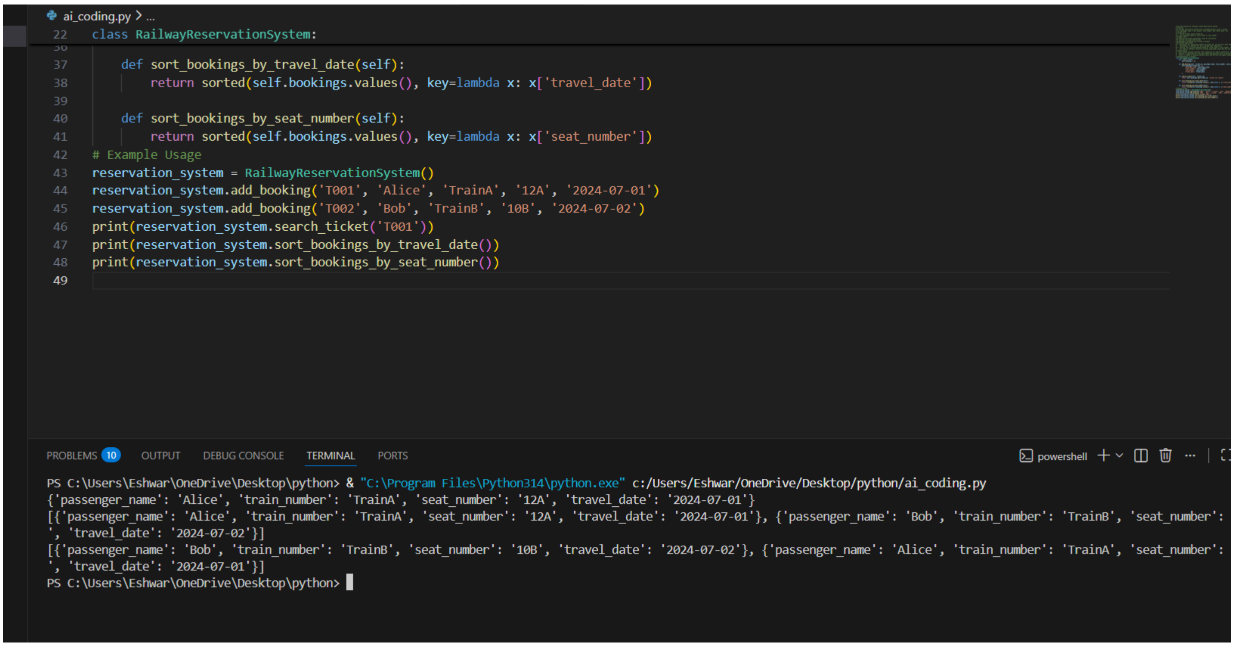
1. Search tickets using ticket ID.
2. Sort bookings based on travel date or seat number.

Student Task

- Identify efficient algorithms using AI assistance.
- Justify the algorithm choices.
- Implement searching and sorting in Python.



```python
# ask Description #4: Railway Ticket Reservation System
# Scenario
# A railway reservation system stores booking details such as ticket
# ID, passenger name, train number, seat number, and travel date. The
# system must:
# 1. Search tickets using ticket ID.
# 2. Sort bookings based on travel date or seat number.
# Student Task
# • Identify efficient algorithms using AI assistance.
# • Justify the algorithm choices.
# • Implement searching and sorting in Python.
# Efficient Algorithms:
# 1. Searching: For searching tickets by ticket ID, we can use a hash table
#    (dictionary in Python) for O(1) average time complexity.
# 2. Sorting: For sorting bookings based on travel date or seat number, we can
#    use Timsort (Python's built-in sorting algorithm) which has O(n log n) time complexity.
# Justification:
# - Hash tables provide constant time complexity for search operations, making them ideal for quickly retrieving
# - Timsort is optimized for real-world data and performs well on partially sorted data,
#   which is common in booking systems where new entries are added frequently.
# Implementation in Python:
class RailwayReservationSystem:
    def __init__(self):
        self.bookings = {}

    def add_booking(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
        self.bookings[ticket_id] = {
            'passenger_name': passenger_name,
            'train_number': train_number,
            'seat_number': seat_number,
            'travel_date': travel_date
        }

    def search_ticket(self, ticket_id):
        return self.bookings.get(ticket_id, "Ticket not found")

    def sort_bookings_by_travel_date(self):
```



```python
class RailwayReservationSystem:

    def sort_bookings_by_travel_date(self):
        return sorted(self.bookings.values(), key=lambda x: x['travel_date'])

    def sort_bookings_by_seat_number(self):
        return sorted(self.bookings.values(), key=lambda x: x['seat_number'])
# Example Usage
reservation_system = RailwayReservationSystem()
reservation_system.add_booking('T001', 'Alice', 'TrainA', '12A', '2024-07-01')
reservation_system.add_booking('T002', 'Bob', 'TrainB', '10B', '2024-07-02')
print(reservation_system.search_ticket('T001'))
print(reservation_system.sort_bookings_by_travel_date())
print(reservation_system.sort_bookings_by_seat_number())
```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
{'passenger_name': 'Alice', 'train_number': 'TrainA', 'seat_number': '12A', 'travel_date': '2024-07-01'}
[{'passenger_name': 'Alice', 'train_number': 'TrainA', 'seat_number': '12A', 'travel_date': '2024-07-01'}, {'passenger_name': 'Bob', 'train_number': 'TrainB', 'seat_number':
', 'travel_date': '2024-07-02'}]
[{'passenger_name': 'Bob', 'train_number': 'TrainB', 'seat_number': '10B', 'travel_date': '2024-07-02'}, {'passenger_name': 'Alice', 'train_number': 'TrainA', 'seat_number':
', 'travel_date': '2024-07-01'}]
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

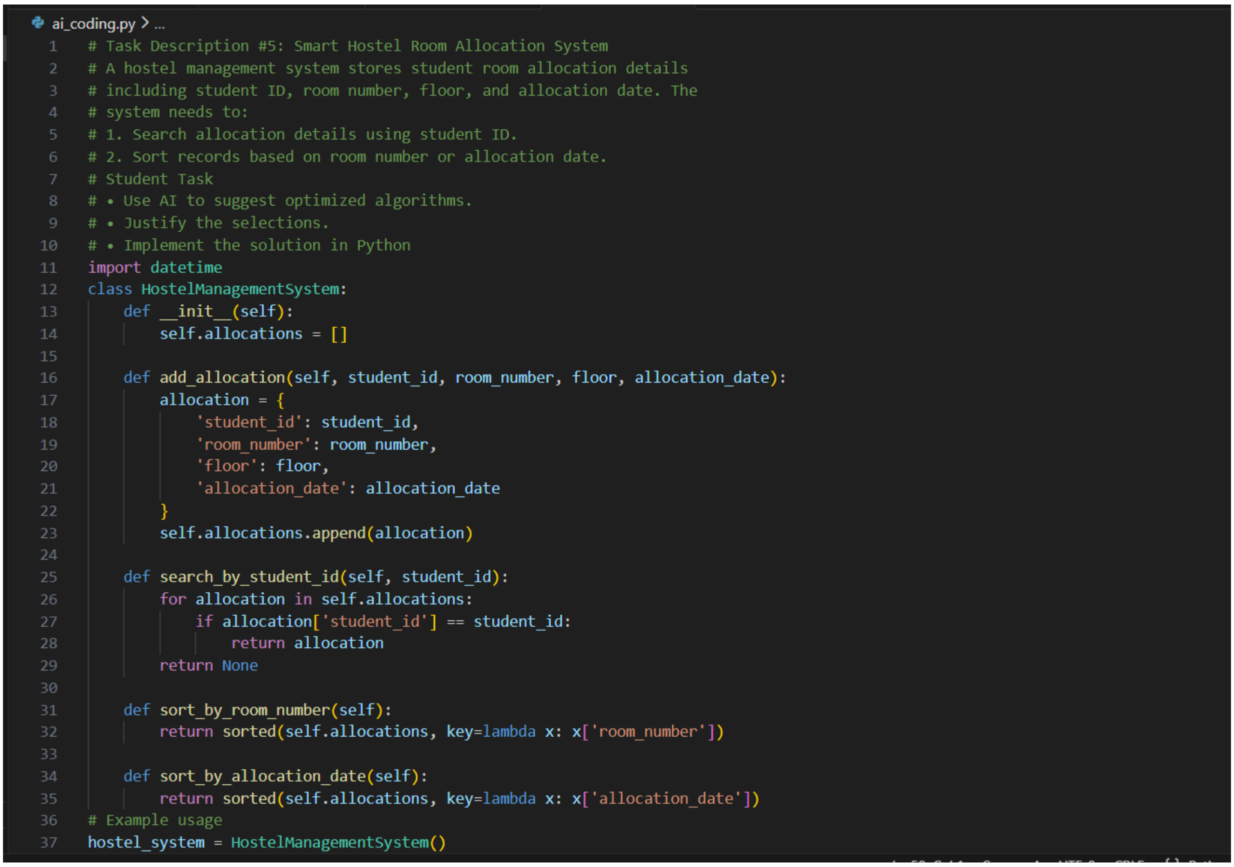
**Task Description #5:** Smart Hostel Room Allocation System

A hostel management system stores student room allocation details including student ID, room number, floor, and allocation date. The system needs to:

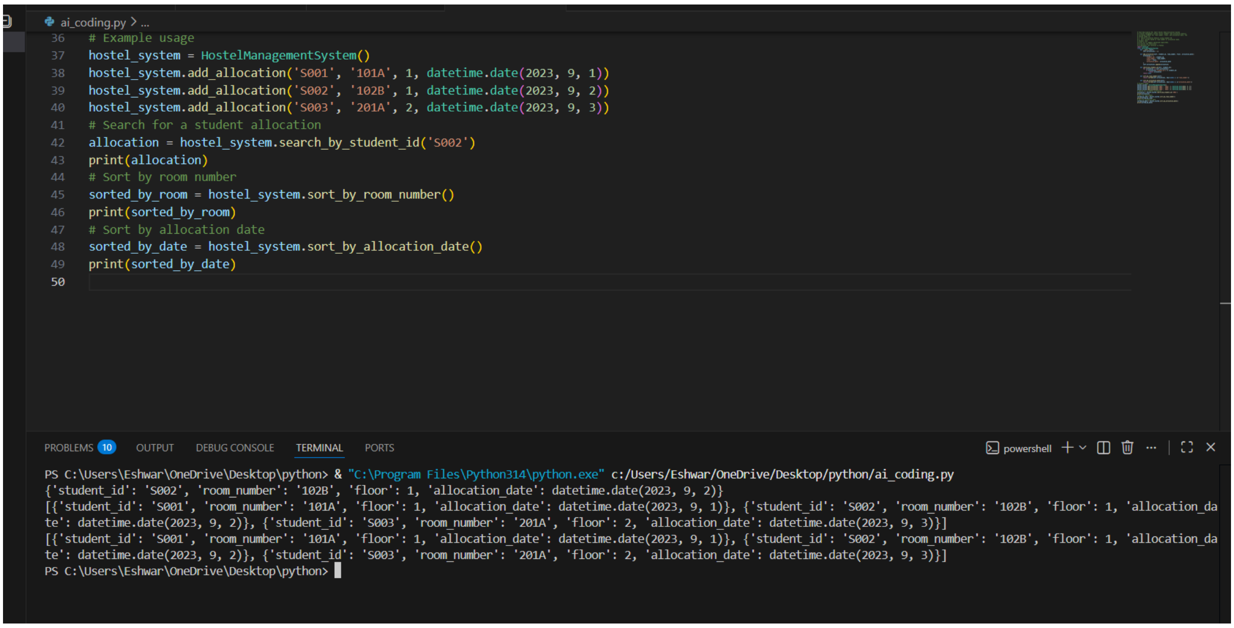
1. Search allocation details using student ID.
2. Sort records based on room number or allocation date.

Student Task

- Use AI to suggest optimized algorithms.
- Justify the selections.
- Implement the solution in Python



```python
# Task Description #5: Smart Hostel Room Allocation System
# A hostel management system stores student room allocation details
# including student ID, room number, floor, and allocation date. The
# system needs to:
# 1. Search allocation details using student ID.
# 2. Sort records based on room number or allocation date.
# Student Task
# • Use AI to suggest optimized algorithms.
# • Justify the selections.
# • Implement the solution in Python
import datetime
class HostelManagementSystem:
    def __init__(self):
        self.allocations = []

    def add_allocation(self, student_id, room_number, floor, allocation_date):
        allocation = {
            'student_id': student_id,
            'room_number': room_number,
            'floor': floor,
            'allocation_date': allocation_date
        }
        self.allocations.append(allocation)

    def search_by_student_id(self, student_id):
        for allocation in self.allocations:
            if allocation['student_id'] == student_id:
                return allocation
        return None

    def sort_by_room_number(self):
        return sorted(self.allocations, key=lambda x: x['room_number'])

    def sort_by_allocation_date(self):
        return sorted(self.allocations, key=lambda x: x['allocation_date'])
# Example usage
hostel_system = HostelManagementSystem()
```



```python
# Example usage
hostel_system = HostelManagementSystem()
hostel_system.add_allocation('S001', '101A', 1, datetime.date(2023, 9, 1))
hostel_system.add_allocation('S002', '102B', 1, datetime.date(2023, 9, 2))
hostel_system.add_allocation('S003', '201A', 2, datetime.date(2023, 9, 3))
# Search for a student allocation
allocation = hostel_system.search_by_student_id('S002')
print(allocation)
# Sort by room number
sorted_by_room = hostel_system.sort_by_room_number()
print(sorted_by_room)
# Sort by allocation date
sorted_by_date = hostel_system.sort_by_allocation_date()
print(sorted_by_date)
```

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
{'student_id': 'S002', 'room_number': '102B', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 2)}
[{'student_id': 'S001', 'room_number': '101A', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 1)}, {'student_id': 'S002', 'room_number': '102B', 'floor': 1, 'allocation_da
te': datetime.date(2023, 9, 2)}, {'student_id': 'S003', 'room_number': '201A', 'floor': 2, 'allocation_date': datetime.date(2023, 9, 3)}]
[{'student_id': 'S001', 'room_number': '101A', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 1)}, {'student_id': 'S002', 'room_number': '102B', 'floor': 1, 'allocation_da
te': datetime.date(2023, 9, 2)}, {'student_id': 'S003', 'room_number': '201A', 'floor': 2, 'allocation_date': datetime.date(2023, 9, 3)}]
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

**Task Description #6:** Online Movie Streaming Platform

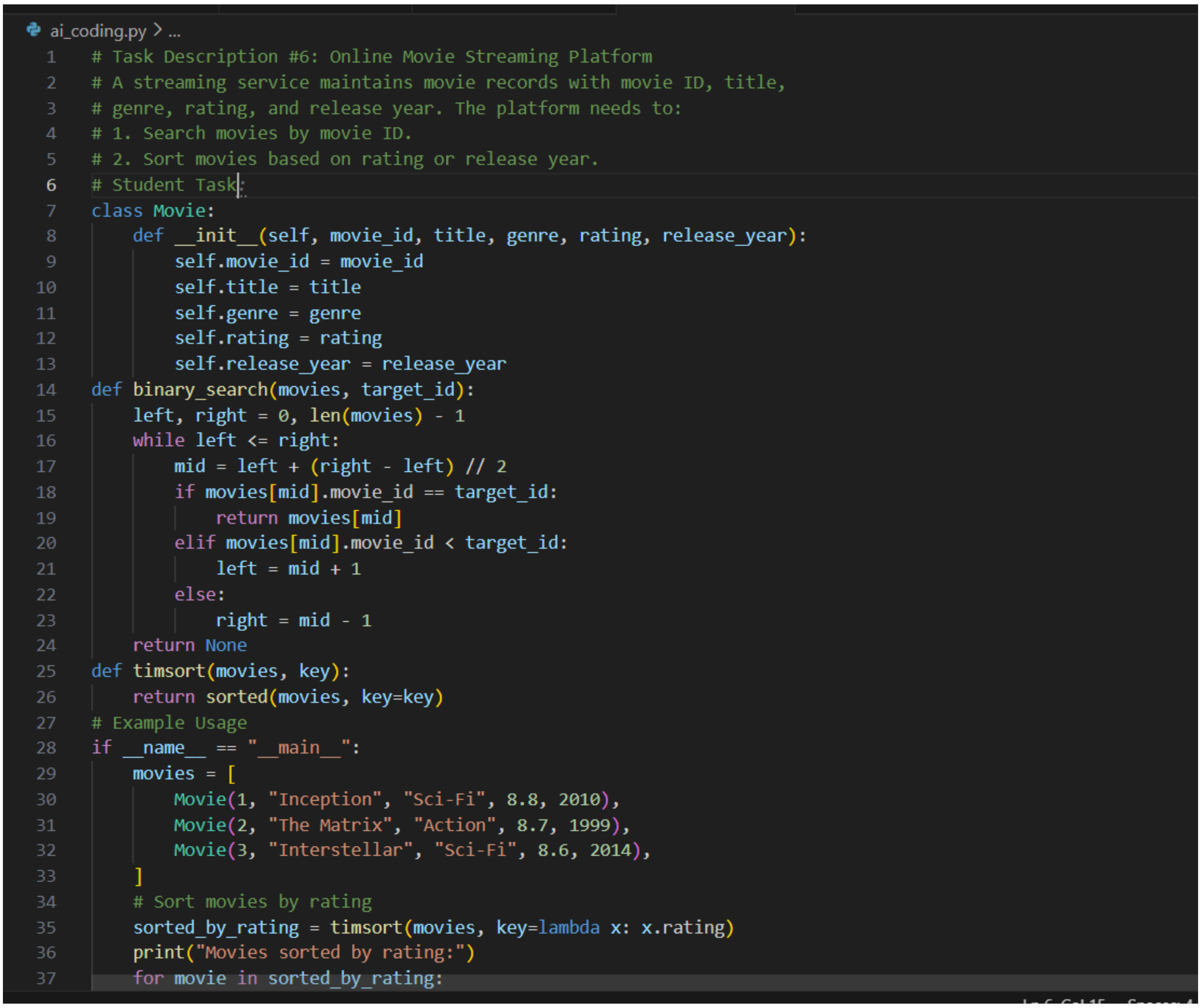
A streaming service maintains movie records with movie ID, title, genre, rating, and release year. The platform needs to:

1. Search movies by movie ID.
2. Sort movies based on rating or release year.

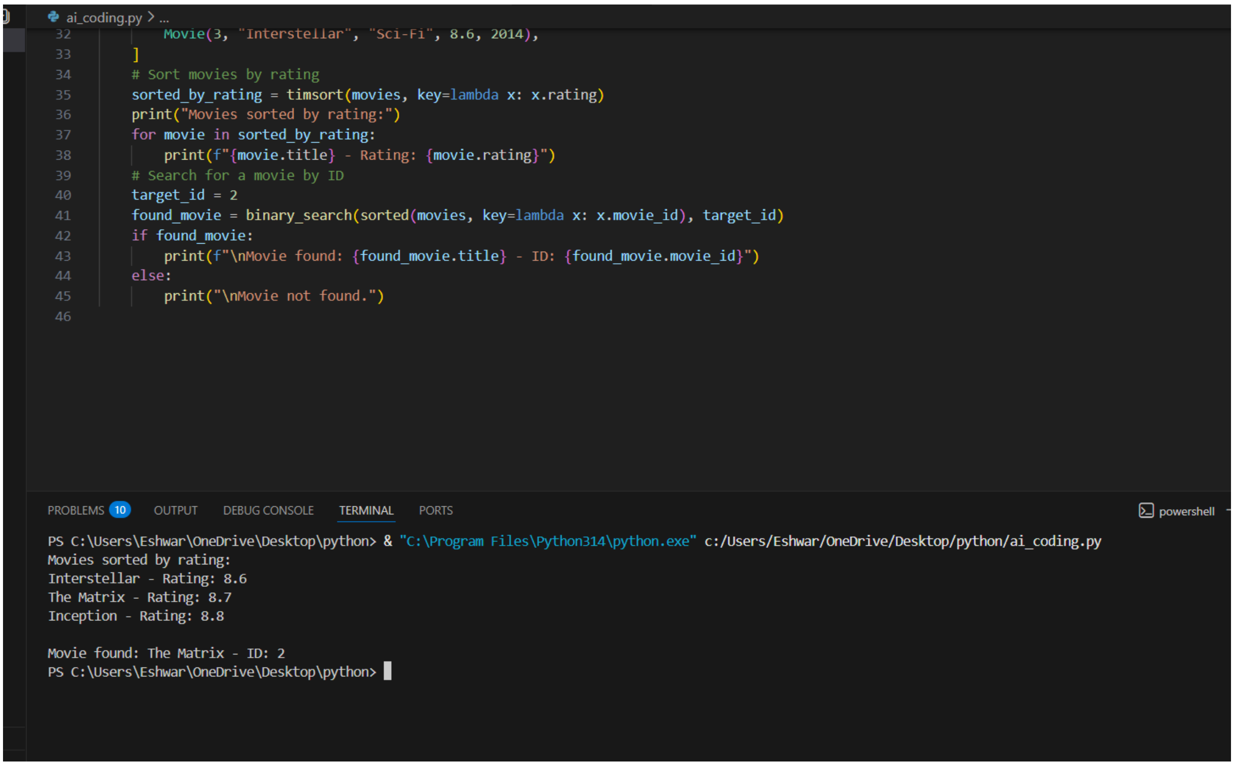
Student Task

• Recommend searching and sorting algorithms using AI.

- Justify the chosen algorithms.
- Implement Python functions



```python
# Task Description #6: Online Movie Streaming Platform
# A streaming service maintains movie records with movie ID, title,
# genre, rating, and release year. The platform needs to:
# 1. Search movies by movie ID.
# 2. Sort movies based on rating or release year.
# Student Task:
class Movie:
    def __init__(self, movie_id, title, genre, rating, release_year):
        self.movie_id = movie_id
        self.title = title
        self.genre = genre
        self.rating = rating
        self.release_year = release_year
def binary_search(movies, target_id):
    left, right = 0, len(movies) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if movies[mid].movie_id == target_id:
            return movies[mid]
        elif movies[mid].movie_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def timsort(movies, key):
    return sorted(movies, key=key)
# Example Usage
if __name__ == "__main__":
    movies = [
        Movie(1, "Inception", "Sci-Fi", 8.8, 2010),
        Movie(2, "The Matrix", "Action", 8.7, 1999),
        Movie(3, "Interstellar", "Sci-Fi", 8.6, 2014),
    ]
    # Sort movies by rating
    sorted_by_rating = timsort(movies, key=lambda x: x.rating)
    print("Movies sorted by rating:")
    for movie in sorted_by_rating:
```



```python
        Movie(3, "Interstellar", "Sci-Fi", 8.6, 2014),
    ]
    # Sort movies by rating
    sorted_by_rating = timsort(movies, key=lambda x: x.rating)
    print("Movies sorted by rating:")
    for movie in sorted_by_rating:
        print(f"{movie.title} - Rating: {movie.rating}")
    # Search for a movie by ID
    target_id = 2
    found_movie = binary_search(sorted(movies, key=lambda x: x.movie_id), target_id)
    if found_movie:
        print(f"\nMovie found: {found_movie.title} - ID: {found_movie.movie_id}")
    else:
        print("\nMovie not found.")
```

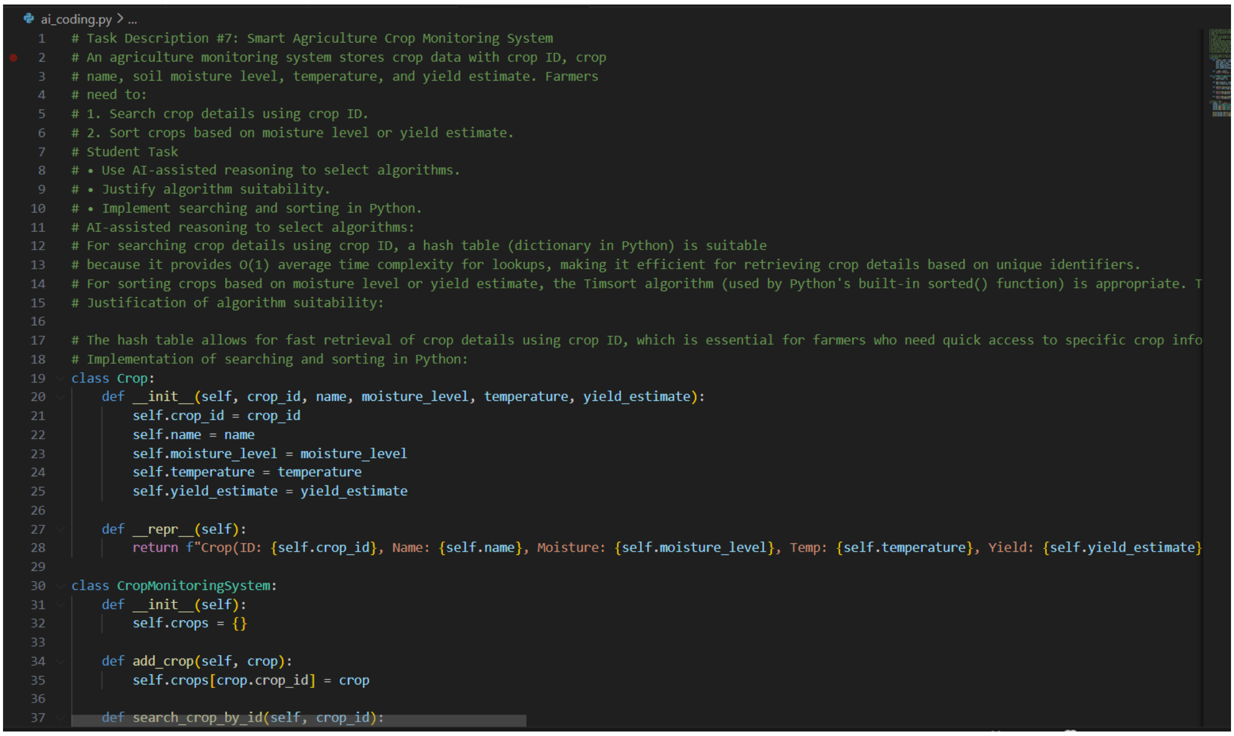
```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
Movies sorted by rating:
Interstellar - Rating: 8.6
The Matrix - Rating: 8.7
Inception - Rating: 8.8

Movie found: The Matrix - ID: 2
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

**Task Description #7:** Smart Agriculture Crop Monitoring System
An agriculture monitoring system stores crop data with crop ID, crop name, soil moisture level, temperature, and yield estimate. Farmers need to:

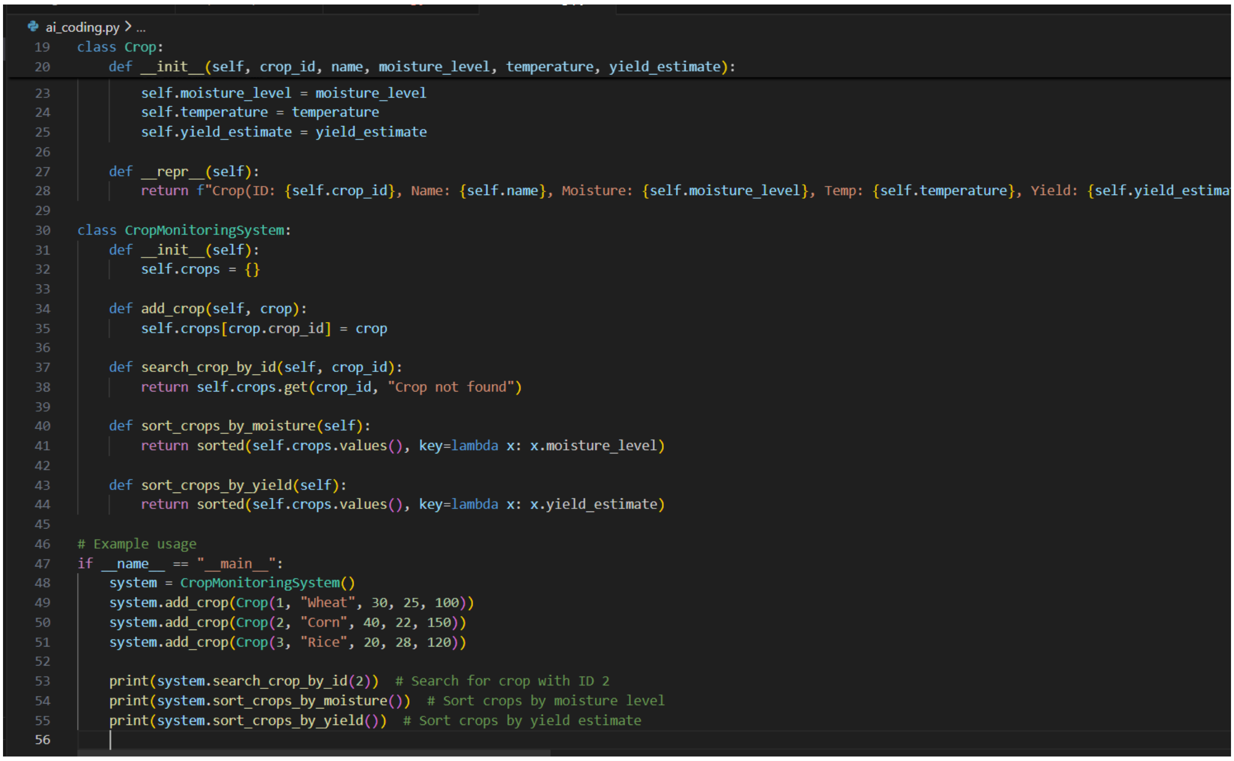
1. Search crop details using crop ID.
2. Sort crops based on moisture level or yield estimate.

Student Task

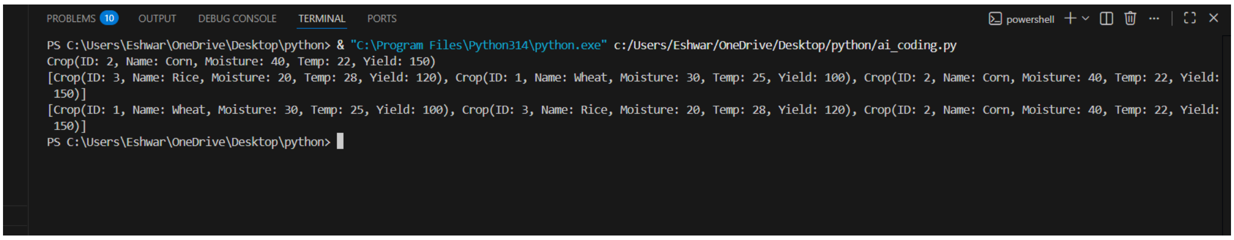
- Use AI-assisted reasoning to select algorithms.
- Justify algorithm suitability.
- Implement searching and sorting in Python.



```python
# Task Description #7: Smart Agriculture Crop Monitoring System
# An agriculture monitoring system stores crop data with crop ID, crop
# name, soil moisture level, temperature, and yield estimate. Farmers
# need to:
# 1. Search crop details using crop ID.
# 2. Sort crops based on moisture level or yield estimate.
# Student Task
# • Use AI-assisted reasoning to select algorithms.
# • Justify algorithm suitability.
# • Implement searching and sorting in Python.
# AI-assisted reasoning to select algorithms:
# For searching crop details using crop ID, a hash table (dictionary in Python) is suitable
# because it provides O(1) average time complexity for lookups, making it efficient for retrieving crop details based on unique identifiers.
# For sorting crops based on moisture level or yield estimate, the Timsort algorithm (used by Python's built-in sorted() function) is appropriate. T
# Justification of algorithm suitability:

# The hash table allows for fast retrieval of crop details using crop ID, which is essential for farmers who need quick access to specific crop info
# Implementation of searching and sorting in Python:
class Crop:
    def __init__(self, crop_id, name, moisture_level, temperature, yield_estimate):
        self.crop_id = crop_id
        self.name = name
        self.moisture_level = moisture_level
        self.temperature = temperature
        self.yield_estimate = yield_estimate

    def __repr__(self):
        return f"Crop(ID: {self.crop_id}, Name: {self.name}, Moisture: {self.moisture_level}, Temp: {self.temperature}, Yield: {self.yield_estimate}

class CropMonitoringSystem:
    def __init__(self):
        self.crops = {}

    def add_crop(self, crop):
        self.crops[crop.crop_id] = crop

    def search_crop_by_id(self, crop_id):
```



```python
class Crop:
    def __init__(self, crop_id, name, moisture_level, temperature, yield_estimate):
        self.moisture_level = moisture_level
        self.temperature = temperature
        self.yield_estimate = yield_estimate

    def __repr__(self):
        return f"Crop(ID: {self.crop_id}, Name: {self.name}, Moisture: {self.moisture_level}, Temp: {self.temperature}, Yield: {self.yield_estima

class CropMonitoringSystem:
    def __init__(self):
        self.crops = {}

    def add_crop(self, crop):
        self.crops[crop.crop_id] = crop

    def search_crop_by_id(self, crop_id):
        return self.crops.get(crop_id, "Crop not found")

    def sort_crops_by_moisture(self):
        return sorted(self.crops.values(), key=lambda x: x.moisture_level)

    def sort_crops_by_yield(self):
        return sorted(self.crops.values(), key=lambda x: x.yield_estimate)

# Example usage
if __name__ == "__main__":
    system = CropMonitoringSystem()
    system.add_crop(Crop(1, "Wheat", 30, 25, 100))
    system.add_crop(Crop(2, "Corn", 40, 22, 150))
    system.add_crop(Crop(3, "Rice", 20, 28, 120))

    print(system.search_crop_by_id(2))  # Search for crop with ID 2
    print(system.sort_crops_by_moisture())  # Sort crops by moisture level
    print(system.sort_crops_by_yield())  # Sort crops by yield estimate
    
```



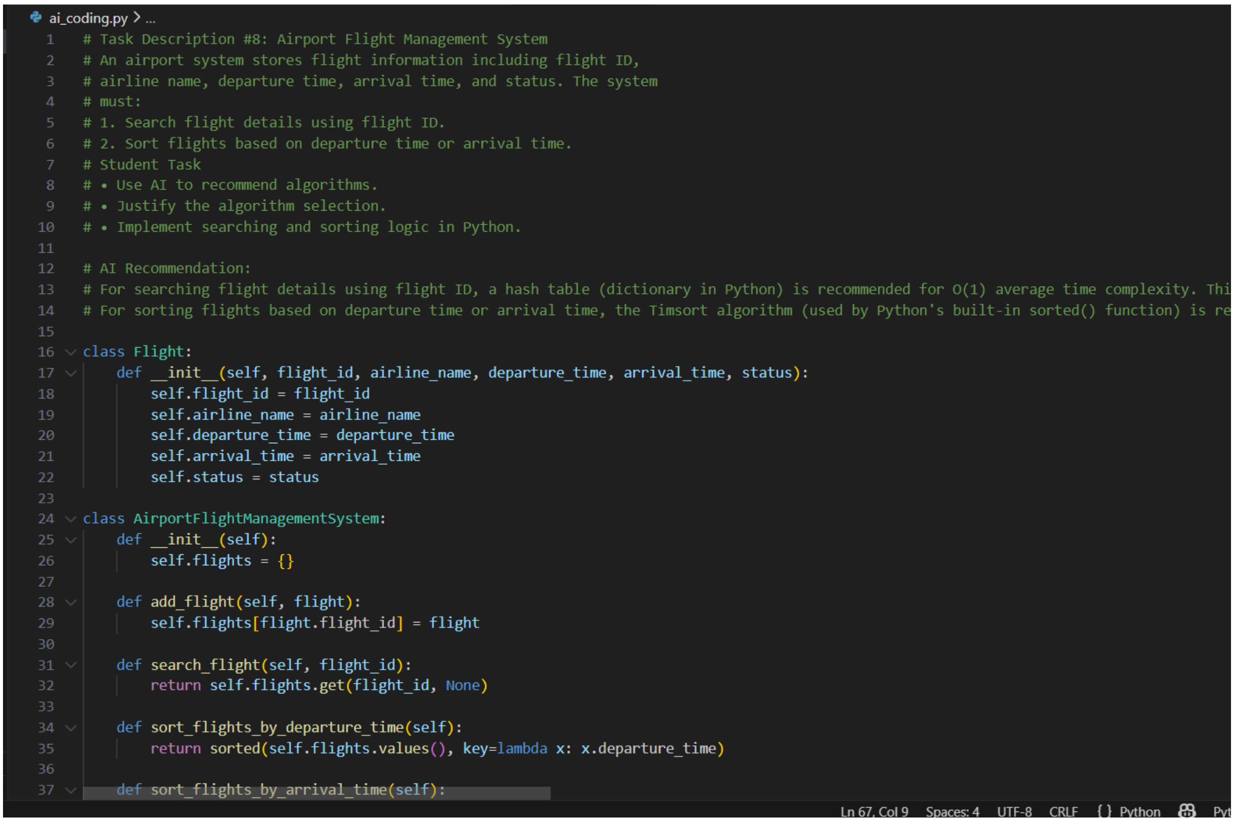
```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
Crop(ID: 2, Name: Corn, Moisture: 40, Temp: 22, Yield: 150)
[Crop(ID: 3, Name: Rice, Moisture: 20, Temp: 28, Yield: 120), Crop(ID: 1, Name: Wheat, Moisture: 30, Temp: 25, Yield: 100), Crop(ID: 2, Name: Corn, Moisture: 40, Temp: 22, Yield:
 150)]
[Crop(ID: 1, Name: Wheat, Moisture: 30, Temp: 25, Yield: 100), Crop(ID: 3, Name: Rice, Moisture: 20, Temp: 28, Yield: 120), Crop(ID: 2, Name: Corn, Moisture: 40, Temp: 22, Yield:
 150)]
PS C:\Users\Eshwar\OneDrive\Desktop\python> 
```

**Task Description #8:** Airport Flight Management System
An airport system stores flight information including flight ID, airline name, departure time, arrival time, and status. The system must:

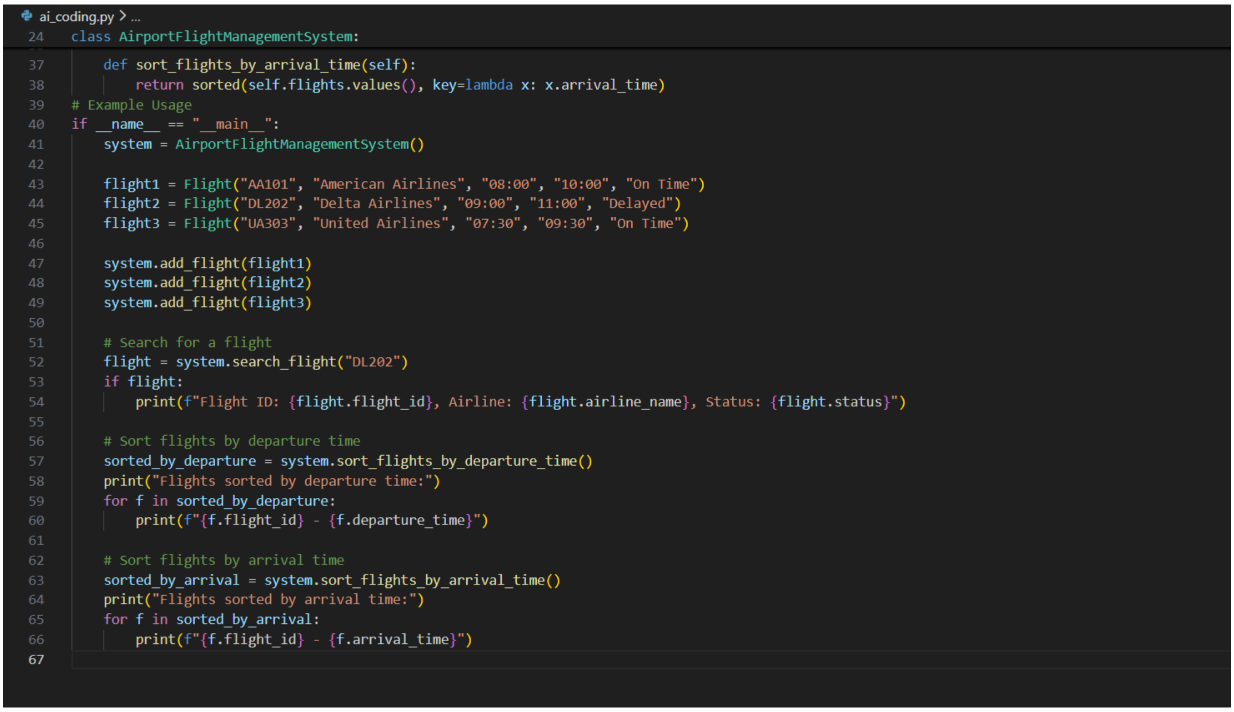
1. Search flight details using flight ID.
2. Sort flights based on departure time or arrival time.

Student Task

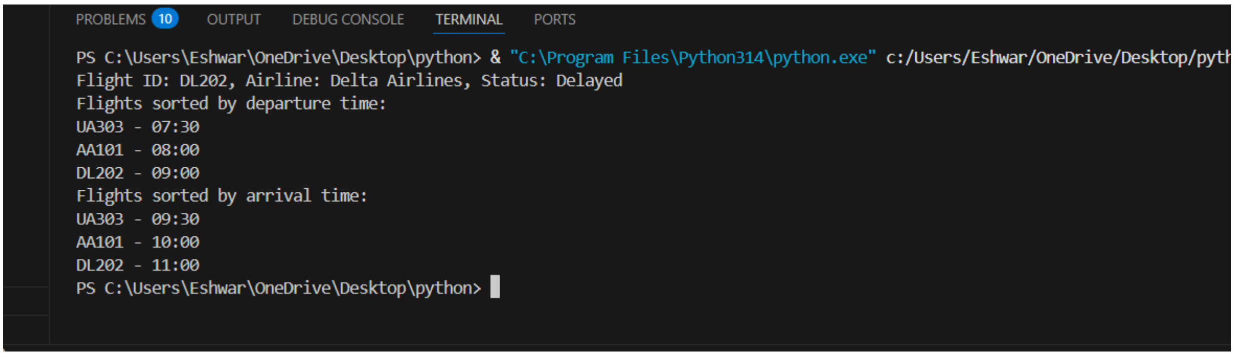
- Use AI to recommend algorithms.
- Justify the algorithm selection.
- Implement searching and sorting logic in Python.



```python
# Task Description #8: Airport Flight Management System
# An airport system stores flight information including flight ID,
# airline name, departure time, arrival time, and status. The system
# must:
# 1. Search flight details using flight ID.
# 2. Sort flights based on departure time or arrival time.
# Student Task
# • Use AI to recommend algorithms.
# • Justify the algorithm selection.
# • Implement searching and sorting logic in Python.

# AI Recommendation:
# For searching flight details using flight ID, a hash table (dictionary in Python) is recommended for O(1) average time complexity. Thi
# For sorting flights based on departure time or arrival time, the Timsort algorithm (used by Python's built-in sorted() function) is re

class Flight:
    def __init__(self, flight_id, airline_name, departure_time, arrival_time, status):
        self.flight_id = flight_id
        self.airline_name = airline_name
        self.departure_time = departure_time
        self.arrival_time = arrival_time
        self.status = status

class AirportFlightManagementSystem:
    def __init__(self):
        self.flights = {}

    def add_flight(self, flight):
        self.flights[flight.flight_id] = flight

    def search_flight(self, flight_id):
        return self.flights.get(flight_id, None)

    def sort_flights_by_departure_time(self):
        return sorted(self.flights.values(), key=lambda x: x.departure_time)

    def sort_flights_by_arrival_time(self):
```



```python
class AirportFlightManagementSystem:
    def sort_flights_by_arrival_time(self):
        return sorted(self.flights.values(), key=lambda x: x.arrival_time)
# Example Usage
if __name__ == "__main__":
    system = AirportFlightManagementSystem()

    flight1 = Flight("AA101", "American Airlines", "08:00", "10:00", "On Time")
    flight2 = Flight("DL202", "Delta Airlines", "09:00", "11:00", "Delayed")
    flight3 = Flight("UA303", "United Airlines", "07:30", "09:30", "On Time")

    system.add_flight(flight1)
    system.add_flight(flight2)
    system.add_flight(flight3)

    # Search for a flight
    flight = system.search_flight("DL202")
    if flight:
        print(f"Flight ID: {flight.flight_id}, Airline: {flight.airline_name}, Status: {flight.status}")

    # Sort flights by departure time
    sorted_by_departure = system.sort_flights_by_departure_time()
    print("Flights sorted by departure time:")
    for f in sorted_by_departure:
        print(f"{f.flight_id} - {f.departure_time}")

    # Sort flights by arrival time
    sorted_by_arrival = system.sort_flights_by_arrival_time()
    print("Flights sorted by arrival time:")
    for f in sorted_by_arrival:
        print(f"{f.flight_id} - {f.arrival_time}")
```



```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/pyth
Flight ID: DL202, Airline: Delta Airlines, Status: Delayed
Flights sorted by departure time:
UA303 - 07:30
AA101 - 08:00
DL202 - 09:00
Flights sorted by arrival time:
UA303 - 09:30
AA101 - 10:00
DL202 - 11:00
PS C:\Users\Eshwar\OneDrive\Desktop\python> 
```