LAB-12.5: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

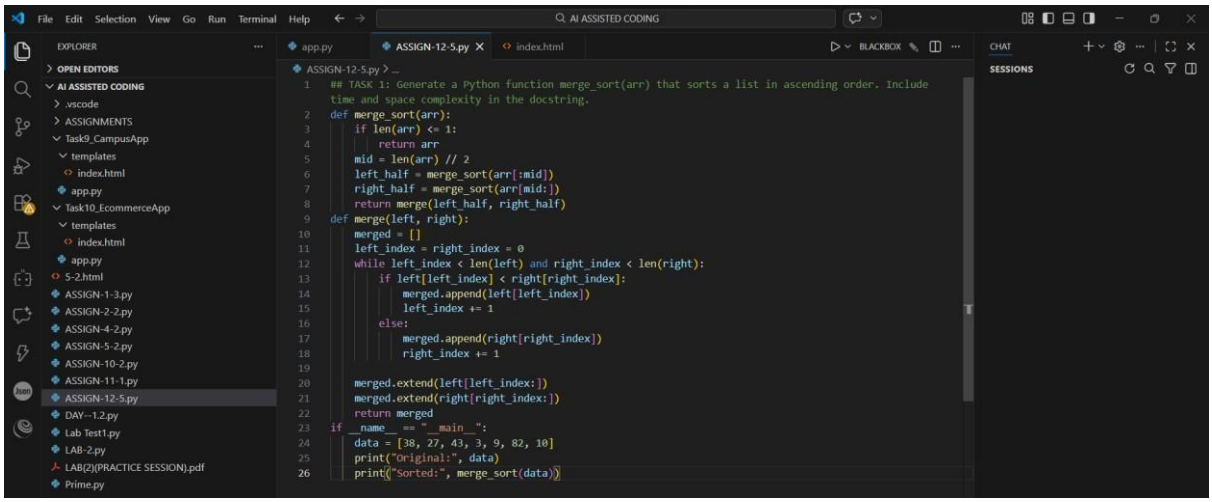
Vyshnavi Parisha

2403A51L34

B-52

Task 1 – Merge Sort:

Prompt: Generate a Python function merge_sort(arr) that sorts a list in ascending order. Include time and space complexity in the docstring.

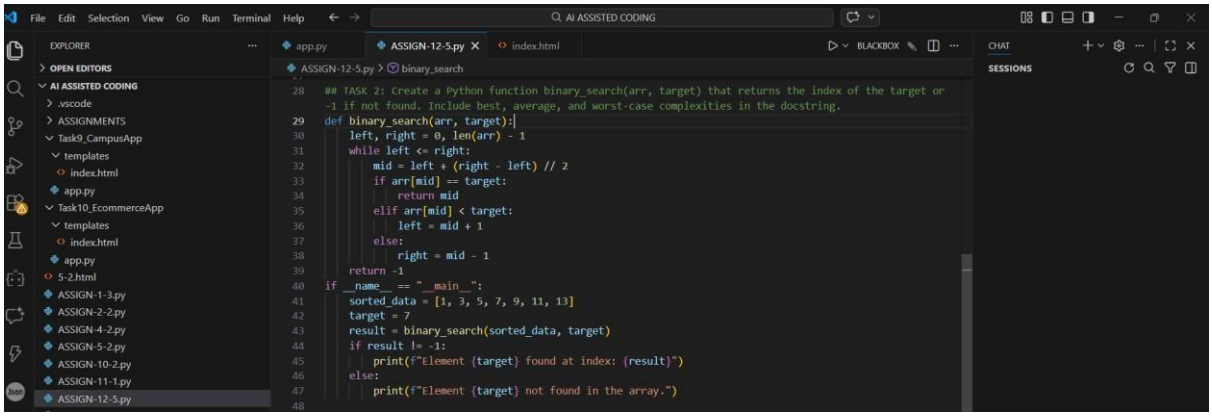


OUTPUT:

Explanation: Merge Sort is chosen because it guarantees O(n log n) time complexity in all cases. It is stable and efficient for large datasets.

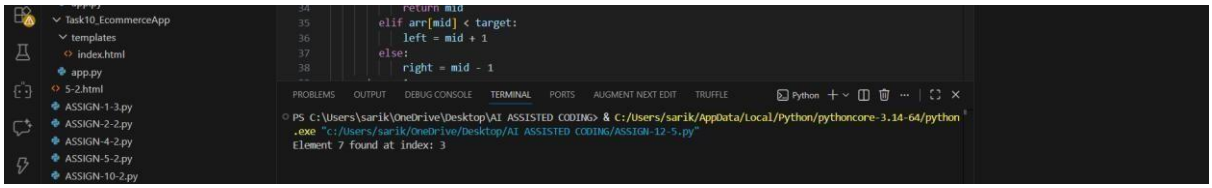
Task 2 – Binary Search:

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



```python
## TASK 2: Create a Python function binary_search(arr, target) that returns the index of the target or
-1 if not found. Include best, average, and worst-case complexities in the docstring.
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    sorted_data = [1, 3, 5, 7, 9, 11, 13]
    target = 7
    result = binary_search(sorted_data, target)
    if result != -1:
        print(f"Element {target} found at index: {result}")
    else:
        print(f"Element {target} not found in the array.")
```

OUTPUT:

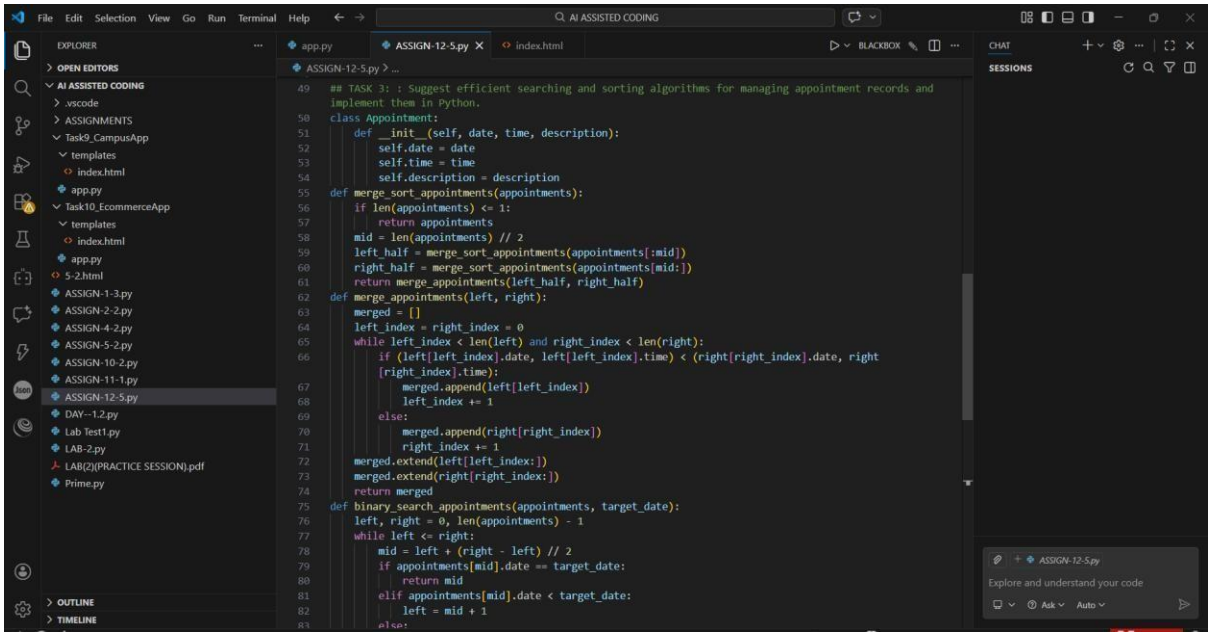


```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python
.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-12-5.py"
Element 7 found at index: 3
```

Explanation: Binary Search works on sorted lists and reduces the search space by half each step. It provides O(log n) efficiency, making it faster than linear search.

Task 3 – Healthcare Appointment System:

Prompt: Suggest efficient searching and sorting algorithms for managing appointment records and implement them in Python.



```python
## TASK 3: : Suggest efficient searching and sorting algorithms for managing appointment records and
implement them in Python.
class Appointment:
    def __init__(self, date, time, description):
        self.date = date
        self.time = time
        self.description = description
def merge_sort_appointments(appointments):
    if len(appointments) <= 1:
        return appointments
    mid = len(appointments) // 2
    left_half = merge_sort_appointments(appointments[:mid])
    right_half = merge_sort_appointments(appointments[mid:])
    return merge_appointments(left_half, right_half)
def merge_appointments(left, right):
    merged = []
    left_index = right_index = 0
    while left_index < len(left) and right_index < len(right):
        if (left[left_index].date, left[left_index].time) < (right[right_index].date, right
        [right_index].time):
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged
def binary_search_appointments(appointments, target_date):
    left, right = 0, len(appointments) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if appointments[mid].date == target_date:
            return mid
        elif appointments[mid].date < target_date:
            left = mid + 1
        else:
```

```python
def binary_search_appointments(appointments, target_date):
    left, right = 0, len(appointments) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if appointments[mid].date == target_date:
            return mid
        elif appointments[mid].date < target_date:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    appointments = [
        Appointment("2024-07-01", "10:00", "Doctor's appointment"),
        Appointment("2024-07-02", "14:00", "Meeting with client"),
        Appointment("2024-07-01", "12:00", "Lunch with friend"),
    ]
    sorted_appointments = merge_sort_appointments(appointments)
    target_date = "2024-07-01"
    index = binary_search_appointments(sorted_appointments, target_date)
    if index != -1:
        print(f"Appointment found: {sorted_appointments[index].description} at {sorted_appointments
        [index].time}")
    else:
        print("No appointment found on the target date.")
```

OUTPUT:

```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python
.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-12-5.py"
Appointment found: Lunch with friend at 12:00
```

Explanation: Binary Search is used for fast appointment ID lookup. Merge Sort is used for stable and efficient sorting by time or consultation fee.

Task 4 – Railway Reservation System:

Prompt: Recommend suitable algorithms to search tickets by ticket ID and sort bookings by travel date or seat number.

```python
## TASK 4: Recommend suitable algorithms to search tickets by ticket ID and sort bookings by travel
date or seat number.
class Ticket:
    def __init__(self, ticket_id, travel_date, seat_number):
        self.ticket_id = ticket_id
        self.travel_date = travel_date
        self.seat_number = seat_number
def merge_sort_tickets(tickets, key):
    if len(tickets) <= 1:
        return tickets
    mid = len(tickets) // 2
    left_half = merge_sort_tickets(tickets[:mid], key)
    right_half = merge_sort_tickets(tickets[mid:], key)
    return merge_tickets(left_half, right_half, key)
def merge_tickets(left, right, key):
    merged = []
    left_index = right_index = 0
    while left_index < len(left) and right_index < len(right):
        if getattr(left[left_index], key) < getattr(right[right_index], key):
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged
def binary_search_tickets(tickets, target_id):
    left, right = 0, len(tickets) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if tickets[mid].ticket_id == target_id:
            return mid
        elif tickets[mid].ticket_id < target_id:
            left = mid + 1
        else:
```

```
def binary_search_tickets(tickets, target_id):
        if tickets[mid].ticket_id == target_id:
            return mid
        elif tickets[mid].ticket_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    tickets = [
        Ticket("T001", "2024-08-01", "A1"),
        Ticket("T002", "2024-08-02", "B1"),
        Ticket("T003", "2024-08-01", "A2"),
    ]
    sorted_tickets_by_date = merge_sort_tickets(tickets, key="travel_date")
    sorted_tickets_by_seat = merge_sort_tickets(tickets, key="seat_number")
    target_id = "T002"
    index = binary_search_tickets(sorted_tickets_by_date, target_id)
    if index != -1:
        print(f"Ticket found: {sorted_tickets_by_date[index].ticket_id} for travel date
        {sorted_tickets_by_date[index].travel_date}")
    else:
        print("No ticket found with the target ID.")
```

OUTPUT:

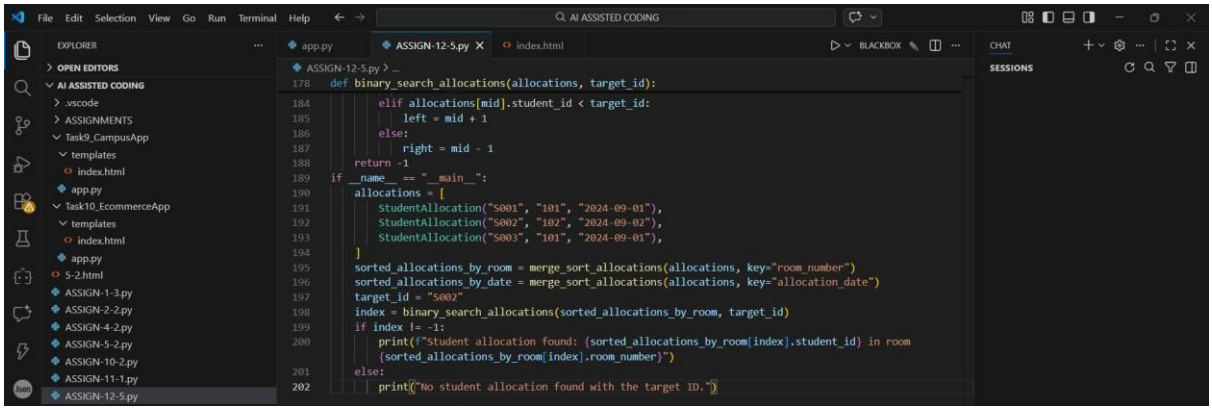
```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python
.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-12-5.py"
Ticket found: T002 for travel date 2024-08-02
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING>
```

Explanation: Binary Search ensures fast ticket lookup in sorted records. Merge Sort guarantees consistent O(n log n) sorting performance.

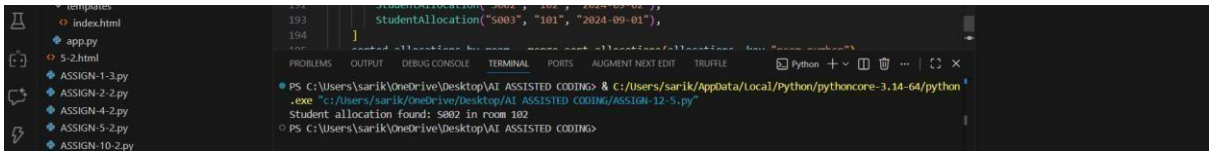
Task 5 – Hostel Room Allocation:

Prompt: Suggest optimized algorithms to search student allocations by ID and sort records by room number or allocation date.

```
## TASK 5: Suggest optimized algorithms to search student allocations by ID and sort records by room
number or allocation date.
class StudentAllocation:
    def __init__(self, student_id, room_number, allocation_date):
        self.student_id = student_id
        self.room_number = room_number
        self.allocation_date = allocation_date
def merge_sort_allocations(allocations, key):
    if len(allocations) <= 1:
        return allocations
    mid = len(allocations) // 2
    left_half = merge_sort_allocations(allocations[:mid], key)
    right_half = merge_sort_allocations(allocations[mid:], key)
    return merge_allocations(left_half, right_half, key)
def merge_allocations(left, right, key):
    merged = []
    left_index = right_index = 0
    while left_index < len(left) and right_index < len(right):
        if getattr(left[left_index], key) < getattr(right[right_index], key):
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged
def binary_search_allocations(allocations, target_id):
    left, right = 0, len(allocations) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if allocations[mid].student_id == target_id:
            return mid
        elif allocations[mid].student_id < target_id:
            left = mid + 1
        else:
```



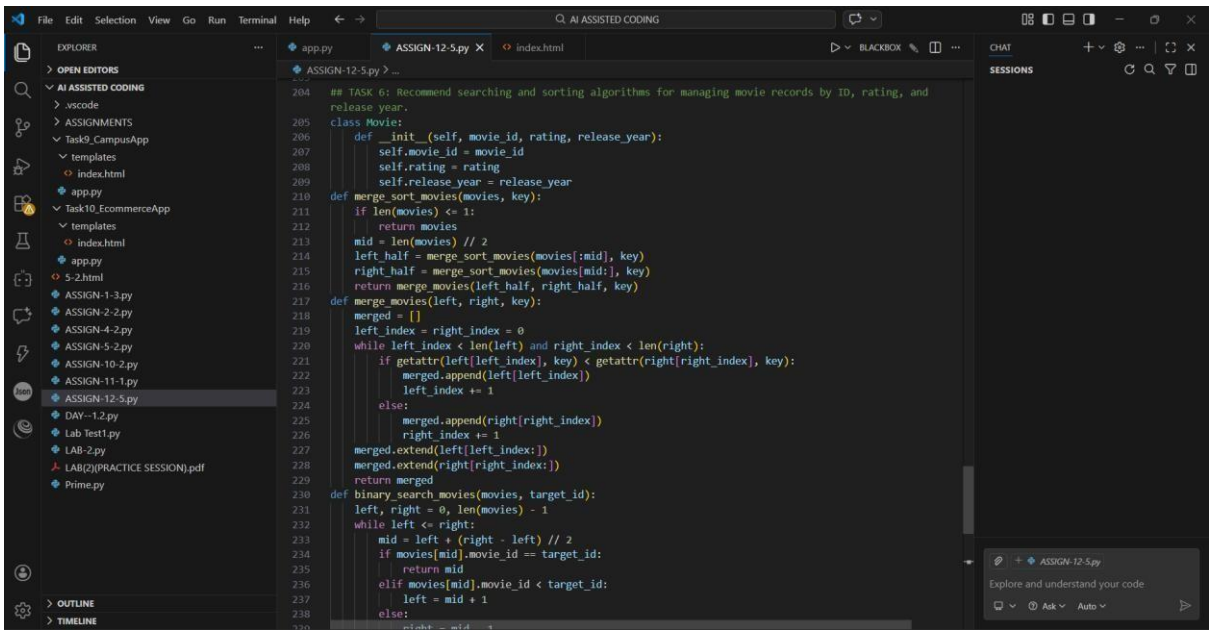
OUTPUT:

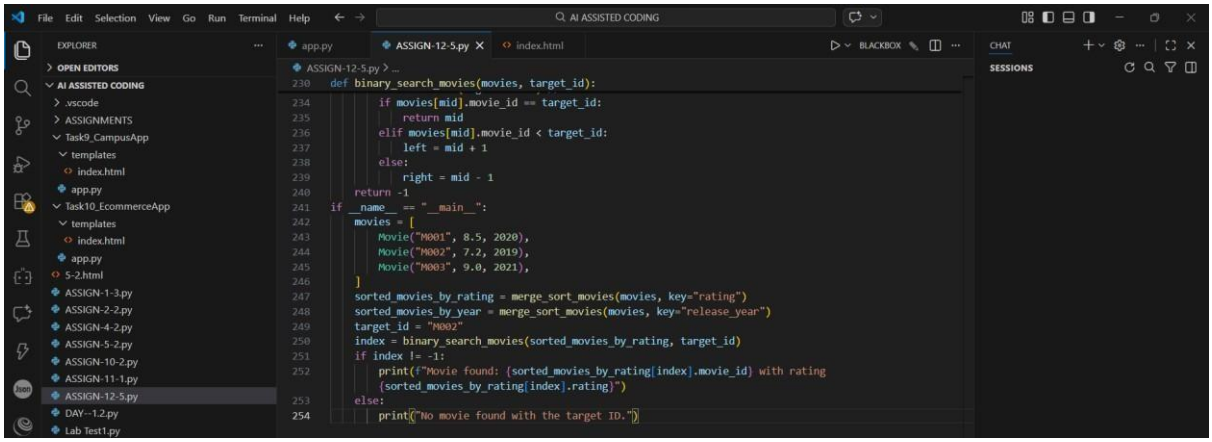


Explanation: Binary Search enables efficient student ID searching. Merge Sort handles large room allocation records reliably.

Task 6 – Movie Streaming Platform:

Prompt: Recommend searching and sorting algorithms for managing movie records by ID, rating, and release year.





```
def binary_search_movies(movies, target_id):
        if movies[mid].movie_id == target_id:
            return mid
        elif movies[mid].movie_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    movies = [
        Movie("M001", 8.5, 2020),
        Movie("M002", 7.2, 2019),
        Movie("M003", 9.0, 2021),
    ]
    sorted_movies_by_rating = merge_sort_movies(movies, key="rating")
    sorted_movies_by_year = merge_sort_movies(movies, key="release_year")
    target_id = "M002"
    index = binary_search_movies(sorted_movies_by_rating, target_id)
    if index != -1:
        print(f"Movie found: {sorted_movies_by_rating[index].movie_id} with rating
        {sorted_movies_by_rating[index].rating}")
    else:
        print("No movie found with the target ID.")
```

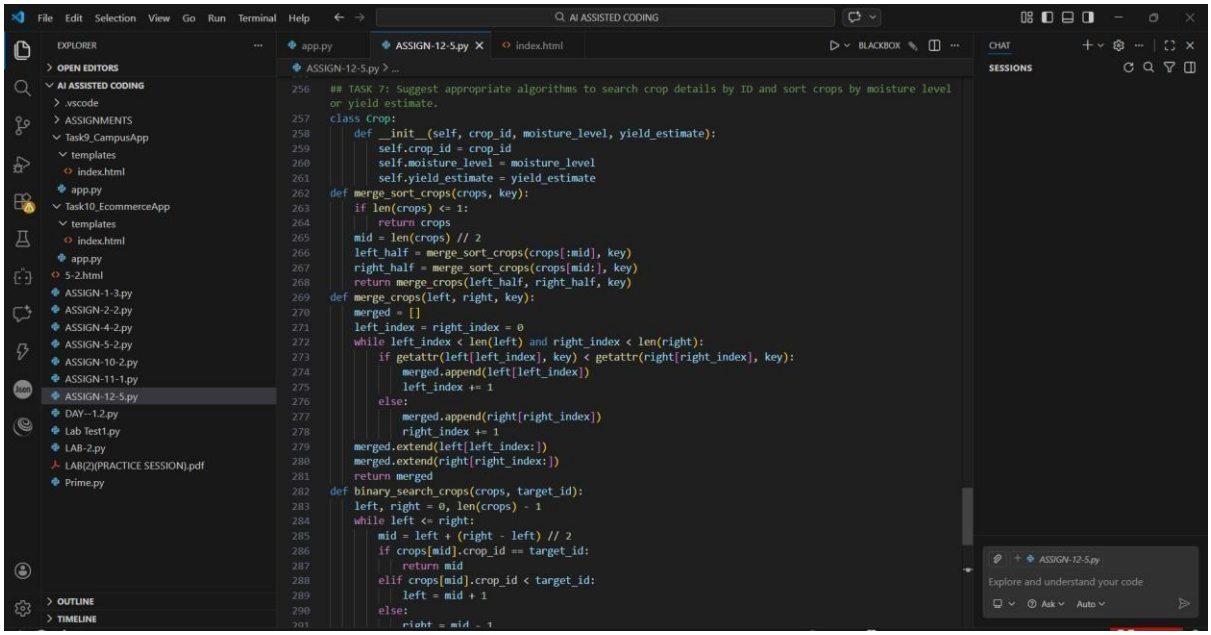
OUTPUT:

```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python
.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-12-5.py"
No movie found with the target ID.
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING>
```

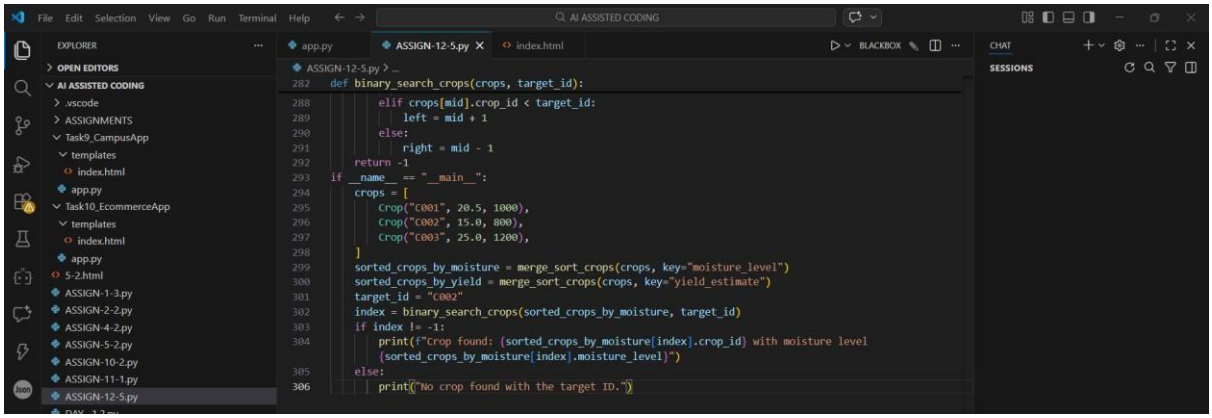
Explanation: Binary Search quickly locates movies by ID. Merge Sort efficiently sorts movies by rating or release year.

Task 7 – Smart Agriculture System:

Prompt: Suggest appropriate algorithms to search crop details by ID and sort crops by moisture level or yield estimate.



```
## TASK 7: Suggest appropriate algorithms to search crop details by ID and sort crops by moisture level
or yield estimate.
class Crop:
    def __init__(self, crop_id, moisture_level, yield_estimate):
        self.crop_id = crop_id
        self.moisture_level = moisture_level
        self.yield_estimate = yield_estimate
def merge_sort_crops(crops, key):
    if len(crops) <= 1:
        return crops
    mid = len(crops) // 2
    left_half = merge_sort_crops(crops[:mid], key)
    right_half = merge_sort_crops(crops[mid:], key)
    return merge_crops(left_half, right_half, key)
def merge_crops(left, right, key):
    merged = []
    left_index = right_index = 0
    while left_index < len(left) and right_index < len(right):
        if getattr(left[left_index], key) < getattr(right[right_index], key):
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged
def binary_search_crops(crops, target_id):
    left, right = 0, len(crops) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if crops[mid].crop_id == target_id:
            return mid
        elif crops[mid].crop_id < target_id:
            left = mid + 1
        else:
```



```
def binary_search_crops(crops, target_id):
        elif crops[mid].crop_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return -1
if __name__ == "__main__":
    crops = [
        Crop("C001", 20.5, 1000),
        Crop("C002", 15.0, 800),
        Crop("C003", 25.0, 1200),
    ]
    sorted_crops_by_moisture = merge_sort_crops(crops, key="moisture_level")
    sorted_crops_by_yield = merge_sort_crops(crops, key="yield_estimate")
    target_id = "C002"
    index = binary_search_crops(sorted_crops_by_moisture, target_id)
    if index != -1:
        print(f"Crop found: {sorted_crops_by_moisture[index].crop_id} with moisture level
        {sorted_crops_by_moisture[index].moisture_level}")
    else:
        print("No crop found with the target ID.")
```

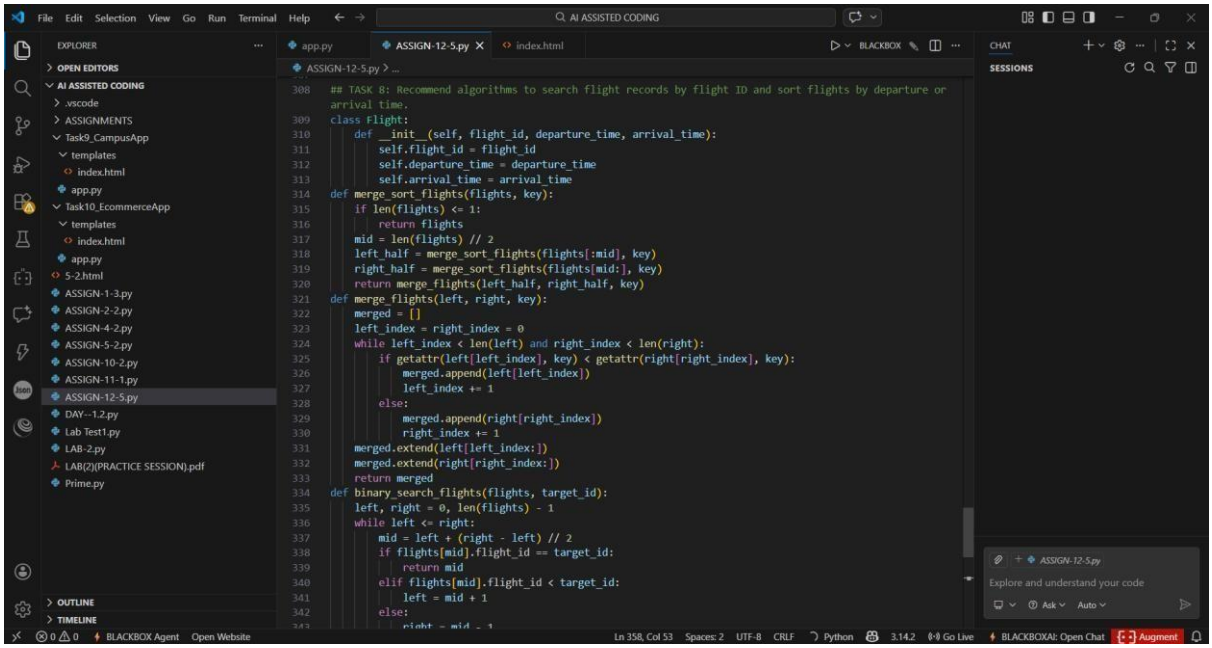
OUTPUT:

```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python
.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-12-5.py"
No crop found with the target ID.
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING>
```

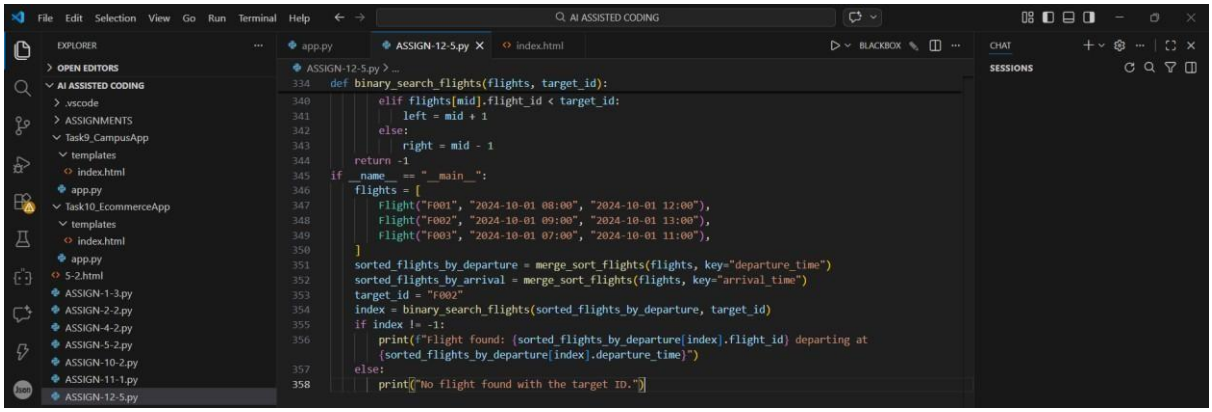
Explanation: Binary Search improves search efficiency in sorted crop data. Merge Sort ensures accurate sorting of agricultural metrics.

Task 8 – Airport Flight Management:

Prompt: Recommend algorithms to search flight records by flight ID and sort flights by departure or arrival time.



```
## TASK 8: Recommend algorithms to search flight records by flight ID and sort flights by departure or
arrival time.
class Flight:
    def __init__(self, flight_id, departure_time, arrival_time):
        self.flight_id = flight_id
        self.departure_time = departure_time
        self.arrival_time = arrival_time
def merge_sort_flights(flights, key):
    if len(flights) <= 1:
        return flights
    mid = len(flights) // 2
    left_half = merge_sort_flights(flights[:mid], key)
    right_half = merge_sort_flights(flights[mid:], key)
    return merge_flights(left_half, right_half, key)
def merge_flights(left, right, key):
    merged = []
    left_index = right_index = 0
    while left_index < len(left) and right_index < len(right):
        if getattr(left[left_index], key) < getattr(right[right_index], key):
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])
    return merged
def binary_search_flights(flights, target_id):
    left, right = 0, len(flights) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if flights[mid].flight_id == target_id:
            return mid
        elif flights[mid].flight_id < target_id:
            left = mid + 1
        else:
```



OUTPUT:

Explanation: Binary Search provides fast flight ID lookup. Merge Sort ensures stable sorting of time-based flight records.