# ASSIGNMENT 12.5

Algorithms with AI Assistance – Sorting, Searching & Optimizing Algorithms

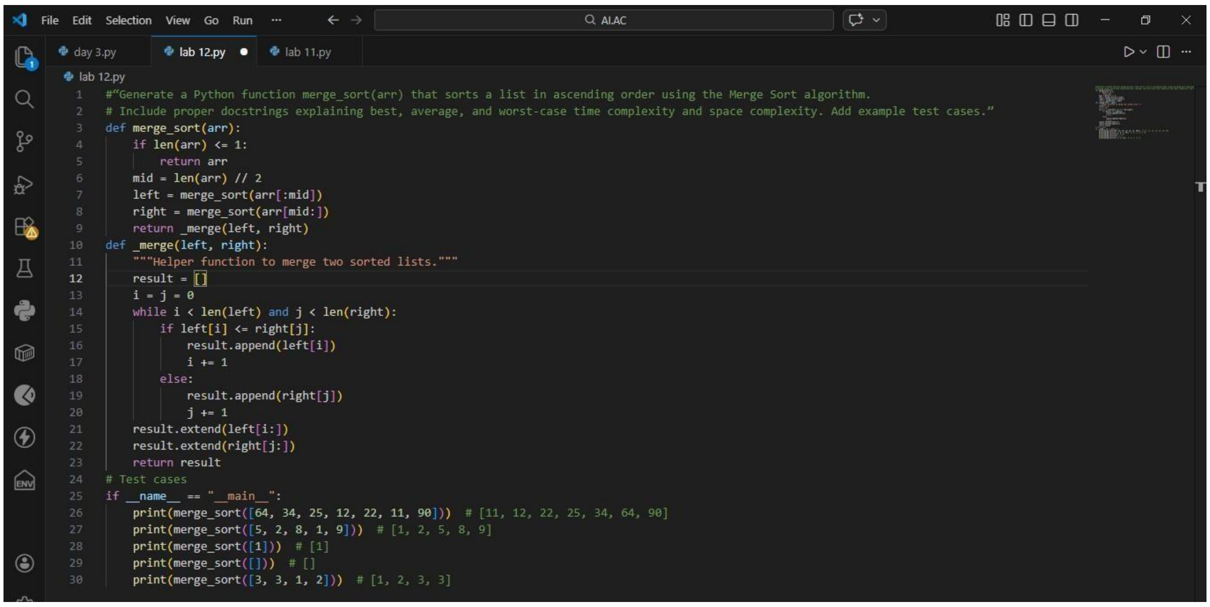
Katakam Santhosh Kumar

2403A51L21

B-51

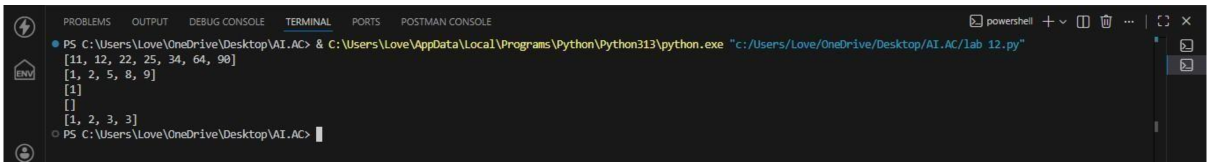
## Task 1: Merge Sort Implementation

**Prompt**: Generate a Python function merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm. Include proper docstrings explaining best, average, and worst-case time complexity and space complexity. Add example test cases.



```python
#"Generate a Python function merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm.
# Include proper docstrings explaining best, average, and worst-case time complexity and space complexity. Add example test cases."
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):
    """Helper function to merge two sorted lists."""
    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
# Test cases
if __name__ == "__main__":
    print(merge_sort([64, 34, 25, 12, 22, 11, 90]))  # [11, 12, 22, 25, 34, 64, 90]
    print(merge_sort([5, 2, 8, 1, 9]))  # [1, 2, 5, 8, 9]
    print(merge_sort([1]))  # [1]
    print(merge_sort([]))  # []
    print(merge_sort([3, 3, 1, 2]))  # [1, 2, 3, 3]
```

**OUTPUT:**



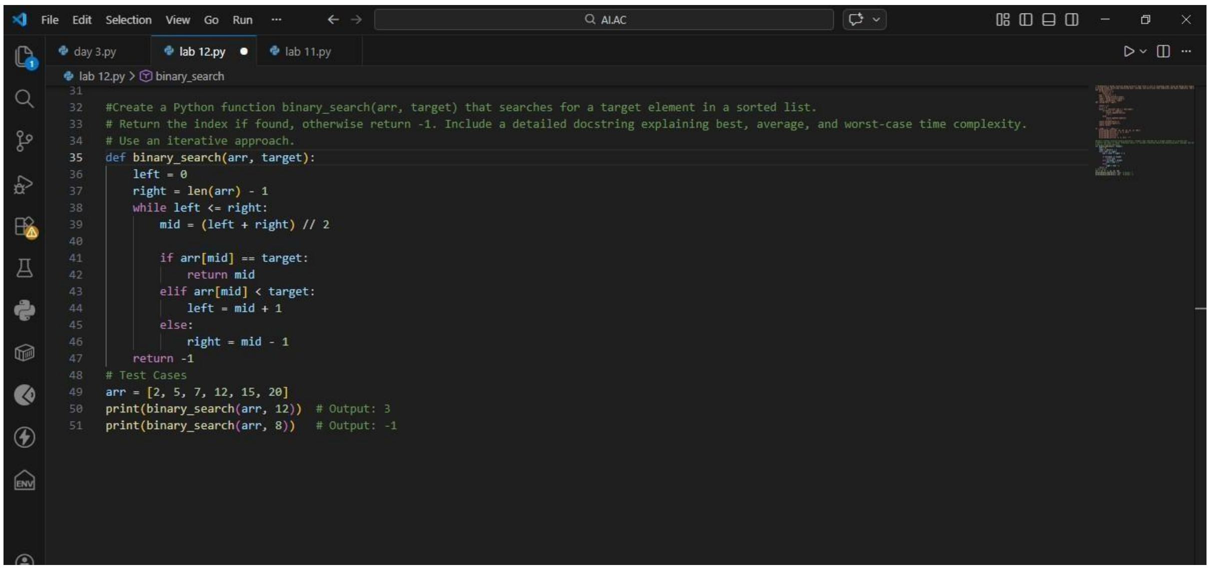
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
[11, 12, 22, 25, 34, 64, 90]
[1, 2, 5, 8, 9]
[1]
[]
[1, 2, 3, 3]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**Explanation:**

- Correct recursive implementation.
- Proper merge logic generation.
- Accurate complexity explanation.

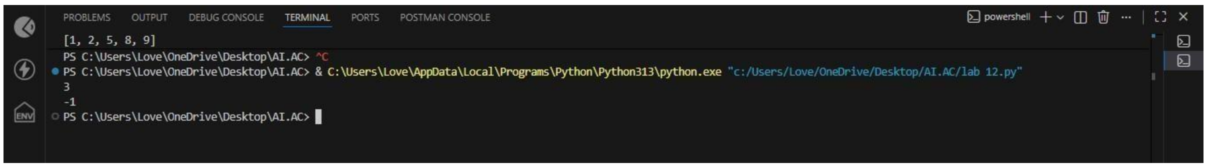
## Task 2: Binary Search Implementation

**Prompt:** Create a Python function binary_search(arr, target) that searches for a target element in a sorted list. Return the index if found, otherwise return -1. Include a detailed docstring explaining best, average, and worst-case time complexity. Use an iterative approach.



```python
#Create a Python function binary_search(arr, target) that searches for a target element in a sorted list.
# Return the index if found, otherwise return -1. Include a detailed docstring explaining best, average, and worst-case time complexity.
# Use an iterative approach.
def binary_search(arr, target):
    left = 0
    right = 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 Cases
arr = [2, 5, 7, 12, 15, 20]
print(binary_search(arr, 12))  # Output: 3
print(binary_search(arr, 8))   # Output: -1
```

## OUTPUT:



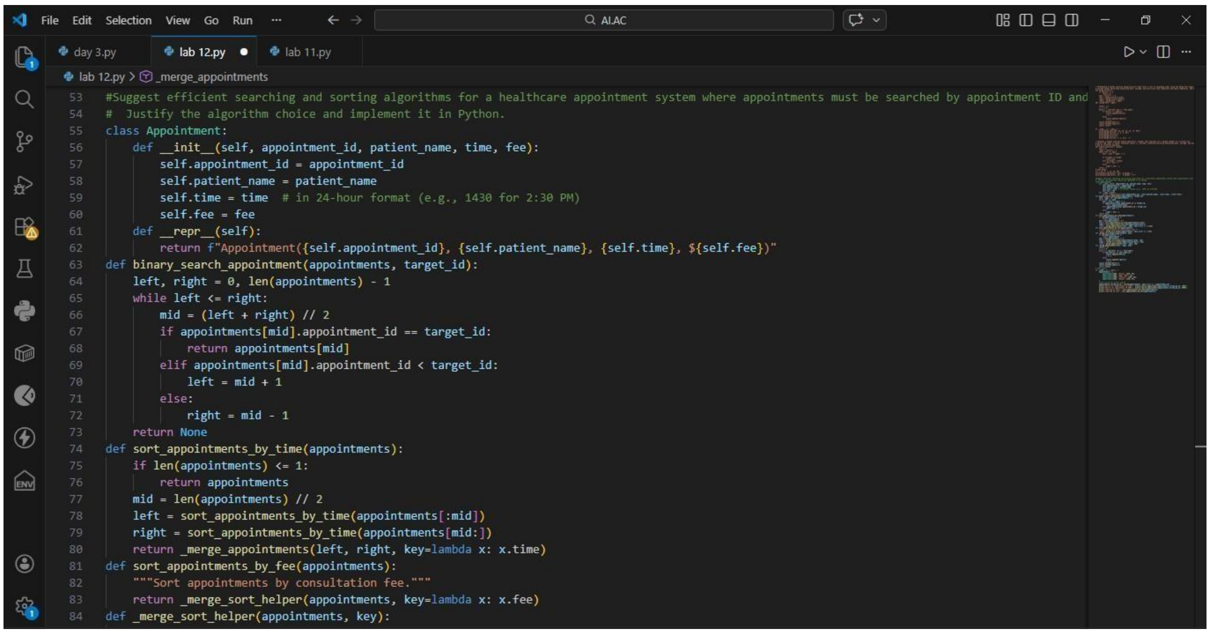
```
[1, 2, 5, 8, 9]
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
3
-1
PS C:\Users\Love\OneDrive\Desktop\AI.AC> 
```

## Explanation:

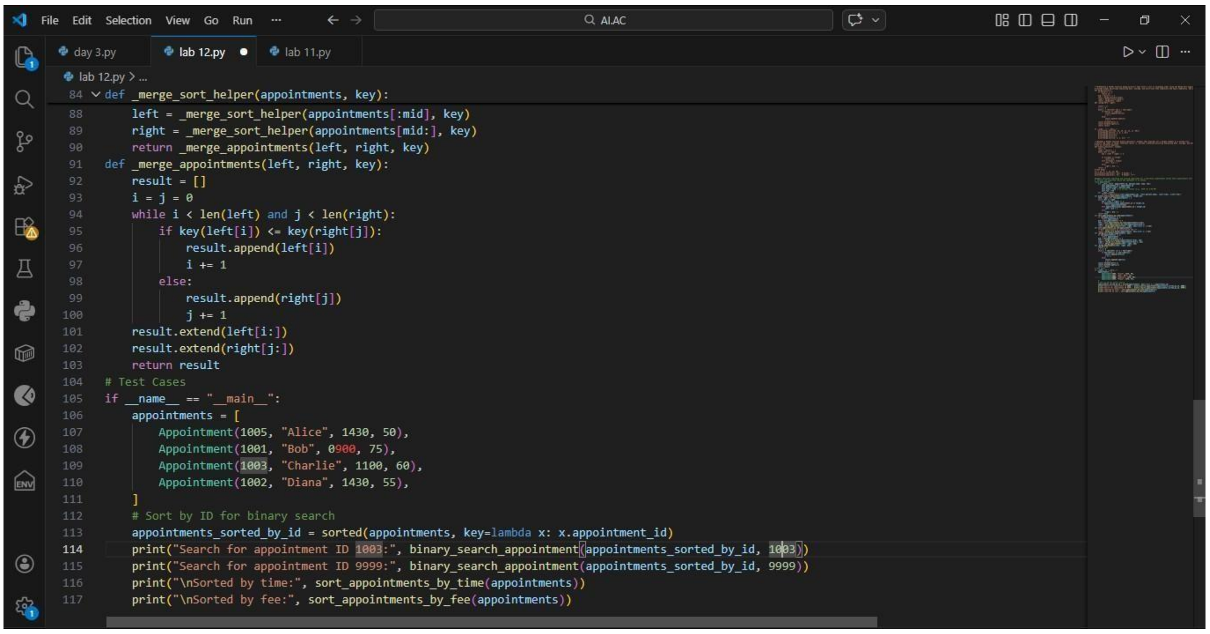
- Correct mid calculation.
- Prevents boundary errors.
- Provides optimized iterative version.

## Task 3: Smart Healthcare Appointment Scheduling System

**Prompt** : Suggest efficient searching and sorting algorithms for a healthcare appointment system where appointments must be searched by appointment ID and sorted by time or consultation fee. Justify the algorithm choice and implement it in Python.

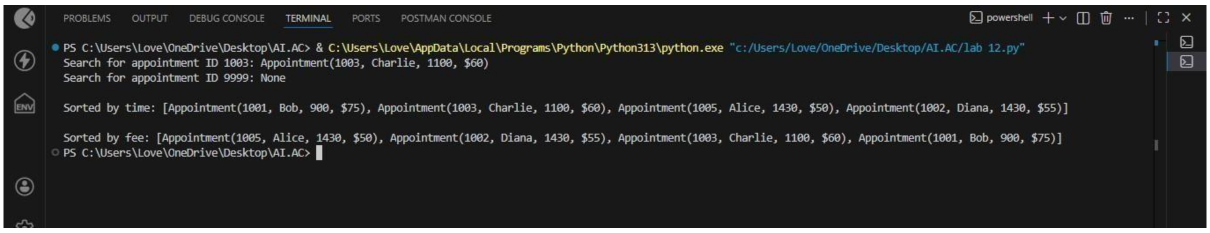


```python
#Suggest efficient searching and sorting algorithms for a healthcare appointment system where appointments must be searched by appointment ID and
#  Justify the algorithm choice and implement it in Python.
class Appointment:
    def __init__(self, appointment_id, patient_name, time, fee):
        self.appointment_id = appointment_id
        self.patient_name = patient_name
        self.time = time  # in 24-hour format (e.g., 1430 for 2:30 PM)
        self.fee = fee
    def __repr__(self):
        return f"Appointment({self.appointment_id}, {self.patient_name}, {self.time}, ${self.fee})"
def binary_search_appointment(appointments, target_id):
    left, right = 0, len(appointments) - 1
    while left <= right:
        mid = (left + right) // 2
        if appointments[mid].appointment_id == target_id:
            return appointments[mid]
        elif appointments[mid].appointment_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def sort_appointments_by_time(appointments):
    if len(appointments) <= 1:
        return appointments
    mid = len(appointments) // 2
    left = sort_appointments_by_time(appointments[:mid])
    right = sort_appointments_by_time(appointments[mid:])
    return _merge_appointments(left, right, key=lambda x: x.time)
def sort_appointments_by_fee(appointments):
    """Sort appointments by consultation fee."""
    return _merge_sort_helper(appointments, key=lambda x: x.fee)
def _merge_sort_helper(appointments, key):
```



```python
def _merge_sort_helper(appointments, key):
    left = _merge_sort_helper(appointments[:mid], key)
    right = _merge_sort_helper(appointments[mid:], key)
    return _merge_appointments(left, right, key)
def _merge_appointments(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    appointments = [
        Appointment(1005, "Alice", 1430, 50),
        Appointment(1001, "Bob", 0900, 75),
        Appointment(1003, "Charlie", 1100, 60),
        Appointment(1002, "Diana", 1430, 55),
    ]
    # Sort by ID for binary search
    appointments_sorted_by_id = sorted(appointments, key=lambda x: x.appointment_id)
    print("Search for appointment ID 1003:", binary_search_appointment(appointments_sorted_by_id, 1003))
    print("Search for appointment ID 9999:", binary_search_appointment(appointments_sorted_by_id, 9999))
    print("\nSorted by time:", sort_appointments_by_time(appointments))
    print("\nSorted by fee:", sort_appointments_by_fee(appointments))
```

**OUTPUT:**



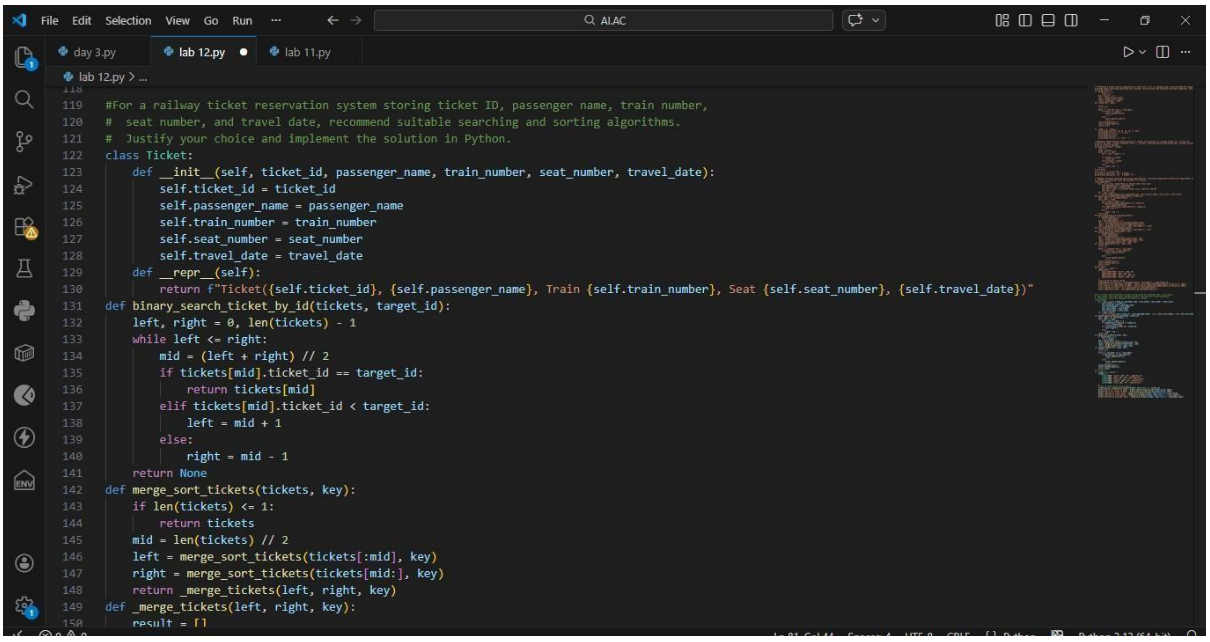
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for appointment ID 1003: Appointment(1003, Charlie, 1100, $60)
Search for appointment ID 9999: None

Sorted by time: [Appointment(1001, Bob, 900, $75), Appointment(1003, Charlie, 1100, $60), Appointment(1005, Alice, 1430, $50), Appointment(1002, Diana, 1430, $55)]

Sorted by fee: [Appointment(1005, Alice, 1430, $50), Appointment(1002, Diana, 1430, $55), Appointment(1003, Charlie, 1100, $60), Appointment(1001, Bob, 900, $75)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

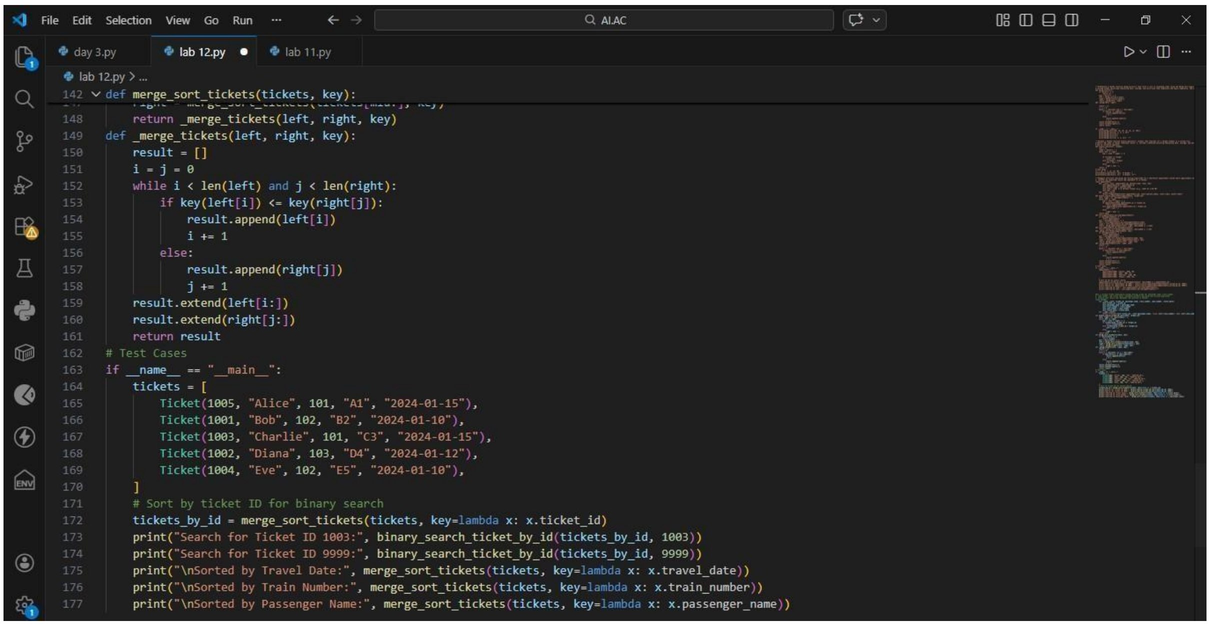
- Hospital systems manage large records.
- Fast ID lookup is critical.
- Stable sorting preserves appointment order.

## Task 4: Railway Ticket Reservation System

**Prompt:** For a railway ticket reservation system storing ticket ID, passenger name, train number, seat number, and travel date, recommend suitable searching and sorting algorithms. Justify your choice and implement the solution in Python.

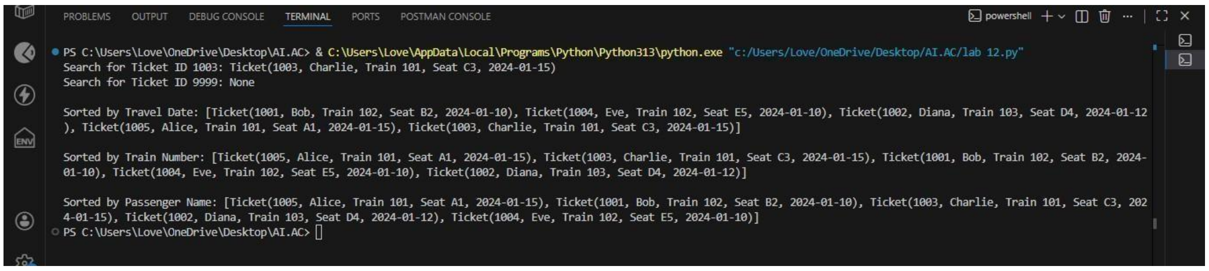


```python
#For a railway ticket reservation system storing ticket ID, passenger name, train number,
#  seat number, and travel date, recommend suitable searching and sorting algorithms.
#  Justify your choice and implement the solution in Python.
class Ticket:
    def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
        self.ticket_id = ticket_id
        self.passenger_name = passenger_name
        self.train_number = train_number
        self.seat_number = seat_number
        self.travel_date = travel_date
    def __repr__(self):
        return f"Ticket({self.ticket_id}, {self.passenger_name}, Train {self.train_number}, Seat {self.seat_number}, {self.travel_date})"
def binary_search_ticket_by_id(tickets, target_id):
    left, right = 0, len(tickets) - 1
    while left <= right:
        mid = (left + right) // 2
        if tickets[mid].ticket_id == target_id:
            return tickets[mid]
        elif tickets[mid].ticket_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def merge_sort_tickets(tickets, key):
    if len(tickets) <= 1:
        return tickets
    mid = len(tickets) // 2
    left = merge_sort_tickets(tickets[:mid], key)
    right = merge_sort_tickets(tickets[mid:], key)
    return _merge_tickets(left, right, key)
def _merge_tickets(left, right, key):
    result = []
```



```python
def merge_sort_tickets(tickets, key):
    return _merge_tickets(left, right, key)
def _merge_tickets(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    tickets = [
        Ticket(1005, "Alice", 101, "A1", "2024-01-15"),
        Ticket(1001, "Bob", 102, "B2", "2024-01-10"),
        Ticket(1003, "Charlie", 101, "C3", "2024-01-15"),
        Ticket(1002, "Diana", 103, "D4", "2024-01-12"),
        Ticket(1004, "Eve", 102, "E5", "2024-01-10"),
    ]
    # Sort by ticket ID for binary search
    tickets_by_id = merge_sort_tickets(tickets, key=lambda x: x.ticket_id)
    print("Search for Ticket ID 1003:", binary_search_ticket_by_id(tickets_by_id, 1003))
    print("Search for Ticket ID 9999:", binary_search_ticket_by_id(tickets_by_id, 9999))
    print("\nSorted by Travel Date:", merge_sort_tickets(tickets, key=lambda x: x.travel_date))
    print("\nSorted by Train Number:", merge_sort_tickets(tickets, key=lambda x: x.train_number))
    print("\nSorted by Passenger Name:", merge_sort_tickets(tickets, key=lambda x: x.passenger_name))
```

## OUTPUT:



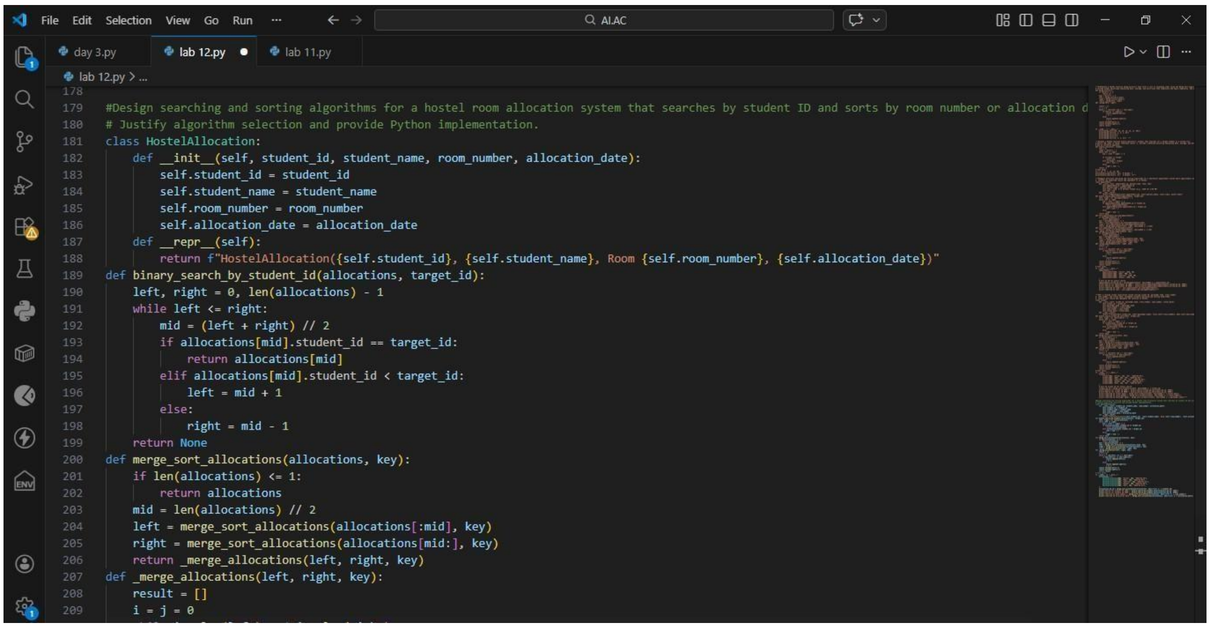
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for Ticket ID 1003: Ticket(1003, Charlie, Train 101, Seat C3, 2024-01-15)
Search for Ticket ID 9999: None

Sorted by Travel Date: [Ticket(1001, Bob, Train 102, Seat B2, 2024-01-10), Ticket(1004, Eve, Train 102, Seat E5, 2024-01-10), Ticket(1002, Diana, Train 103, Seat D4, 2024-01-12
), Ticket(1005, Alice, Train 101, Seat A1, 2024-01-15), Ticket(1003, Charlie, Train 101, Seat C3, 2024-01-15)]

Sorted by Train Number: [Ticket(1005, Alice, Train 101, Seat A1, 2024-01-15), Ticket(1003, Charlie, Train 101, Seat C3, 2024-01-15), Ticket(1001, Bob, Train 102, Seat B2, 2024-
01-10), Ticket(1004, Eve, Train 102, Seat E5, 2024-01-10), Ticket(1002, Diana, Train 103, Seat D4, 2024-01-12)]

Sorted by Passenger Name: [Ticket(1005, Alice, Train 101, Seat A1, 2024-01-15), Ticket(1001, Bob, Train 102, Seat B2, 2024-01-10), Ticket(1003, Charlie, Train 101, Seat C3, 202
4-01-15), Ticket(1002, Diana, Train 103, Seat D4, 2024-01-12), Ticket(1004, Eve, Train 102, Seat E5, 2024-01-10)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

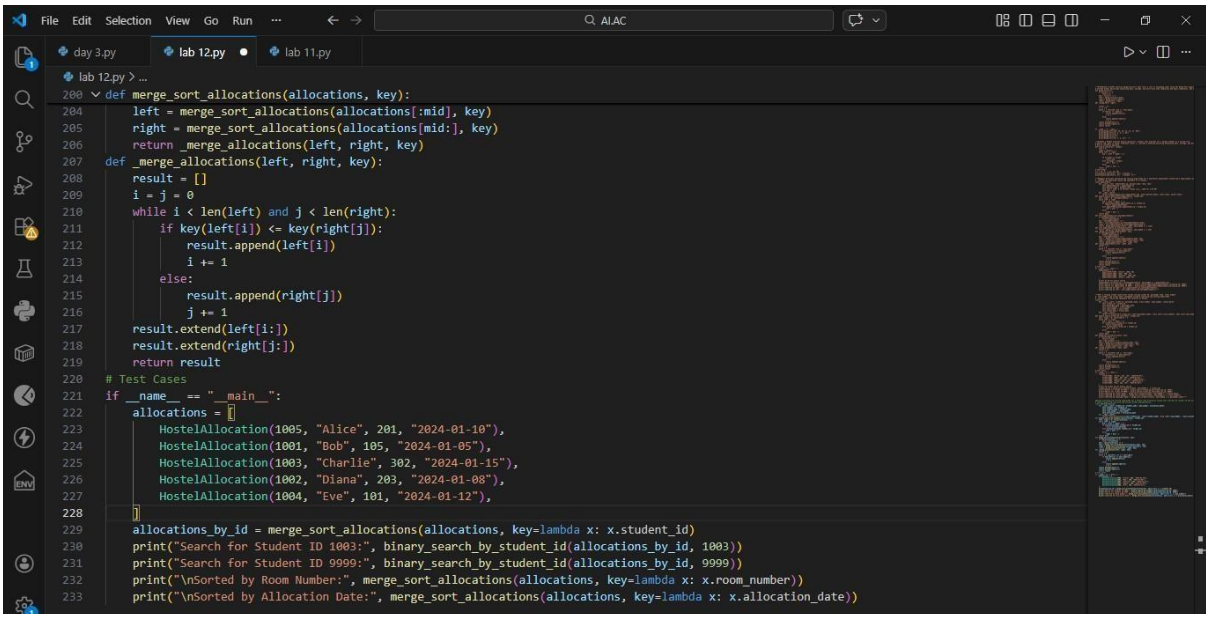
- Railway systems process thousands of bookings daily.
- O(log n) search improves efficiency.
- Sorting by date ensures chronological order.

## Task 5: Smart Hostel Room Allocation System

**Prompt:** Design searching and sorting algorithms for a hostel room allocation system that searches by student ID and sorts by room number or allocation date. Justify algorithm selection and provide Python implementation.

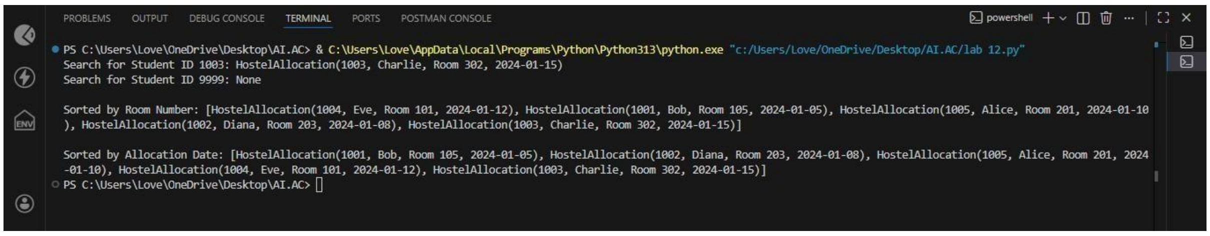


```python
#Design searching and sorting algorithms for a hostel room allocation system that searches by student ID and sorts by room number or allocation d
# Justify algorithm selection and provide Python implementation.
class HostelAllocation:
    def __init__(self, student_id, student_name, room_number, allocation_date):
        self.student_id = student_id
        self.student_name = student_name
        self.room_number = room_number
        self.allocation_date = allocation_date
    def __repr__(self):
        return f"HostelAllocation({self.student_id}, {self.student_name}, Room {self.room_number}, {self.allocation_date})"
def binary_search_by_student_id(allocations, target_id):
    left, right = 0, len(allocations) - 1
    while left <= right:
        mid = (left + right) // 2
        if allocations[mid].student_id == target_id:
            return allocations[mid]
        elif allocations[mid].student_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def merge_sort_allocations(allocations, key):
    if len(allocations) <= 1:
        return allocations
    mid = len(allocations) // 2
    left = merge_sort_allocations(allocations[:mid], key)
    right = merge_sort_allocations(allocations[mid:], key)
    return _merge_allocations(left, right, key)
def _merge_allocations(left, right, key):
    result = []
    i = j = 0
```



```python
def merge_sort_allocations(allocations, key):
    left = merge_sort_allocations(allocations[:mid], key)
    right = merge_sort_allocations(allocations[mid:], key)
    return _merge_allocations(left, right, key)
def _merge_allocations(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    allocations = [
        HostelAllocation(1005, "Alice", 201, "2024-01-10"),
        HostelAllocation(1001, "Bob", 105, "2024-01-05"),
        HostelAllocation(1003, "Charlie", 302, "2024-01-15"),
        HostelAllocation(1002, "Diana", 203, "2024-01-08"),
        HostelAllocation(1004, "Eve", 101, "2024-01-12"),
    ]
    allocations_by_id = merge_sort_allocations(allocations, key=lambda x: x.student_id)
    print("Search for Student ID 1003:", binary_search_by_student_id(allocations_by_id, 1003))
    print("Search for Student ID 9999:", binary_search_by_student_id(allocations_by_id, 9999))
    print("\nSorted by Room Number:", merge_sort_allocations(allocations, key=lambda x: x.room_number))
    print("\nSorted by Allocation Date:", merge_sort_allocations(allocations, key=lambda x: x.allocation_date))
```

## OUTPUT:



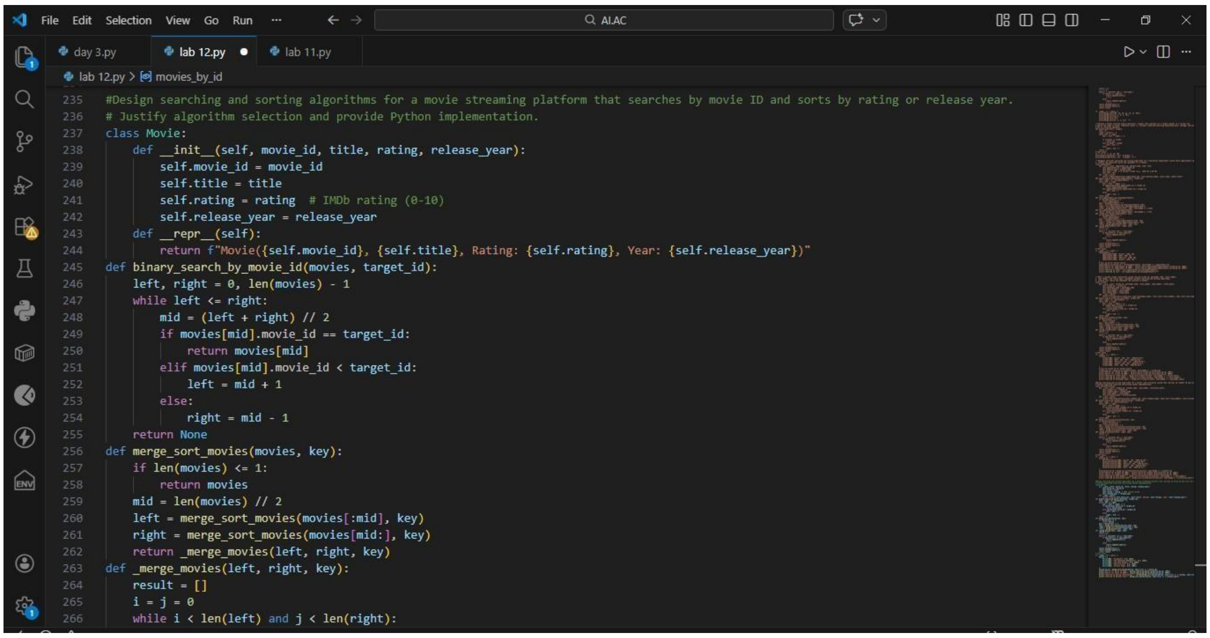
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for Student ID 1003: HostelAllocation(1003, Charlie, Room 302, 2024-01-15)
Search for Student ID 9999: None

Sorted by Room Number: [HostelAllocation(1004, Eve, Room 101, 2024-01-12), HostelAllocation(1001, Bob, Room 105, 2024-01-05), HostelAllocation(1005, Alice, Room 201, 2024-01-10
), HostelAllocation(1002, Diana, Room 203, 2024-01-08), HostelAllocation(1003, Charlie, Room 302, 2024-01-15)]

Sorted by Allocation Date: [HostelAllocation(1001, Bob, Room 105, 2024-01-05), HostelAllocation(1002, Diana, Room 203, 2024-01-08), HostelAllocation(1005, Alice, Room 201, 2024
-01-10), HostelAllocation(1004, Eve, Room 101, 2024-01-12), HostelAllocation(1003, Charlie, Room 302, 2024-01-15)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

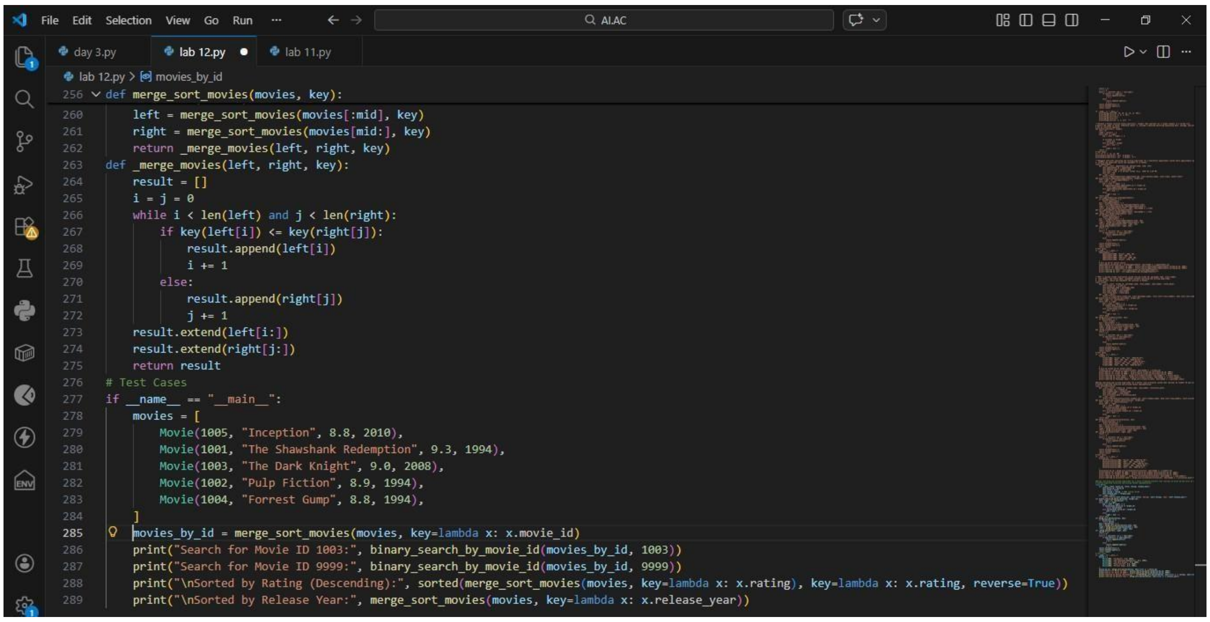
- Student IDs are unique.
- Efficient lookup required for management.
- Stable sorting ensures consistent record ordering.

**Task 6:** Online Movie Streaming Platform

**Prompt:** Recommend optimized searching and sorting algorithms for a movie streaming platform that searches by movie ID and sorts by rating or release year. Justify and implement in Python.

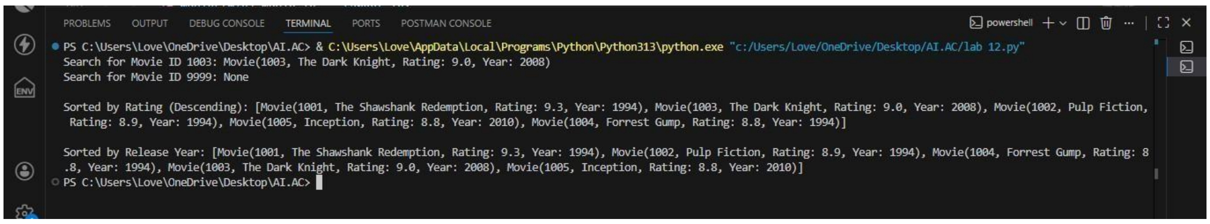


```python
#Design searching and sorting algorithms for a movie streaming platform that searches by movie ID and sorts by rating or release year.
# Justify algorithm selection and provide Python implementation.
class Movie:
    def __init__(self, movie_id, title, rating, release_year):
        self.movie_id = movie_id
        self.title = title
        self.rating = rating  # IMDb rating (0-10)
        self.release_year = release_year
    def __repr__(self):
        return f"Movie({self.movie_id}, {self.title}, Rating: {self.rating}, Year: {self.release_year})"
def binary_search_by_movie_id(movies, target_id):
    left, right = 0, len(movies) - 1
    while left <= right:
        mid = (left + right) // 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 merge_sort_movies(movies, key):
    if len(movies) <= 1:
        return movies
    mid = len(movies) // 2
    left = merge_sort_movies(movies[:mid], key)
    right = merge_sort_movies(movies[mid:], key)
    return _merge_movies(left, right, key)
def _merge_movies(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
```



```python
def merge_sort_movies(movies, key):
        left = merge_sort_movies(movies[:mid], key)
        right = merge_sort_movies(movies[mid:], key)
        return _merge_movies(left, right, key)
def _merge_movies(left, right, key):
        result = []
        i = j = 0
        while i < len(left) and j < len(right):
            if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    movies = [
        Movie(1005, "Inception", 8.8, 2010),
        Movie(1001, "The Shawshank Redemption", 9.3, 1994),
        Movie(1003, "The Dark Knight", 9.0, 2008),
        Movie(1002, "Pulp Fiction", 8.9, 1994),
        Movie(1004, "Forrest Gump", 8.8, 1994),
    ]
    movies_by_id = merge_sort_movies(movies, key=lambda x: x.movie_id)
    print("Search for Movie ID 1003:", binary_search_by_movie_id(movies_by_id, 1003))
    print("Search for Movie ID 9999:", binary_search_by_movie_id(movies_by_id, 9999))
    print("\nSorted by Rating (Descending):", sorted(merge_sort_movies(movies, key=lambda x: x.rating), key=lambda x: x.rating, reverse=True))
    print("\nSorted by Release Year:", merge_sort_movies(movies, key=lambda x: x.release_year))
```

## Output:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for Movie ID 1003: Movie(1003, The Dark Knight, Rating: 9.0, Year: 2008)
Search for Movie ID 9999: None

Sorted by Rating (Descending): [Movie(1001, The Shawshank Redemption, Rating: 9.3, Year: 1994), Movie(1003, The Dark Knight, Rating: 9.0, Year: 2008), Movie(1002, Pulp Fiction,
 Rating: 8.9, Year: 1994), Movie(1005, Inception, Rating: 8.8, Year: 2010), Movie(1004, Forrest Gump, Rating: 8.8, Year: 1994)]

Sorted by Release Year: [Movie(1001, The Shawshank Redemption, Rating: 9.3, Year: 1994), Movie(1002, Pulp Fiction, Rating: 8.9, Year: 1994), Movie(1004, Forrest Gump, Rating: 8
.8, Year: 1994), Movie(1003, The Dark Knight, Rating: 9.0, Year: 2008), Movie(1005, Inception, Rating: 8.8, Year: 2010)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

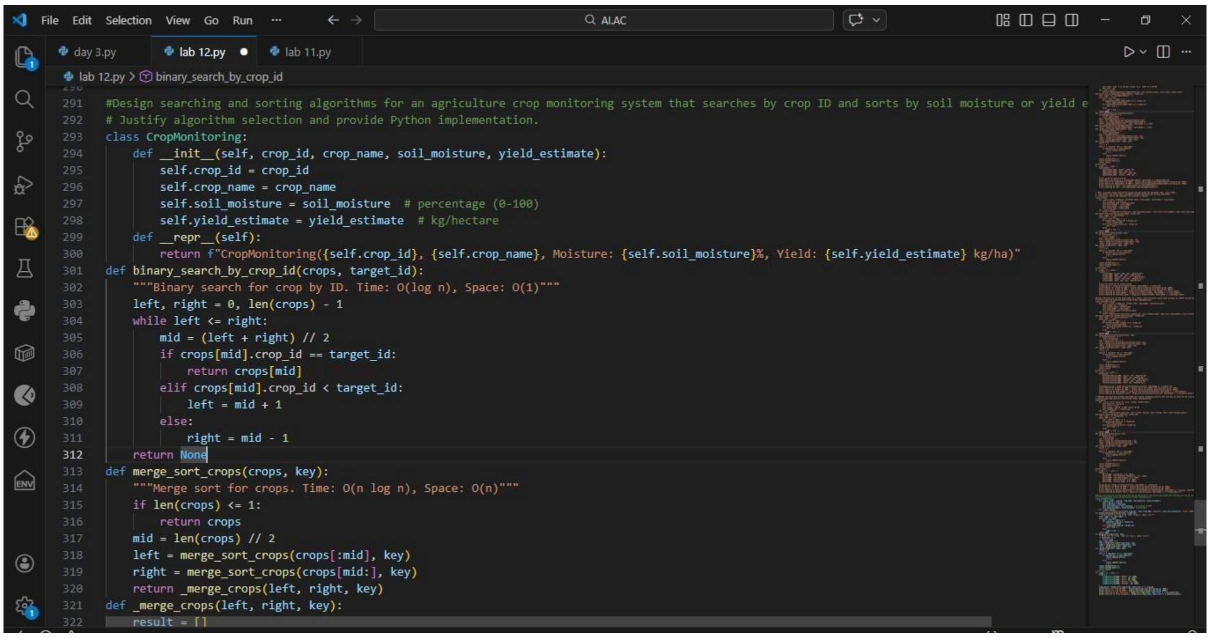
## Explanation:

- Large movie databases.
- Fast search improves user experience.

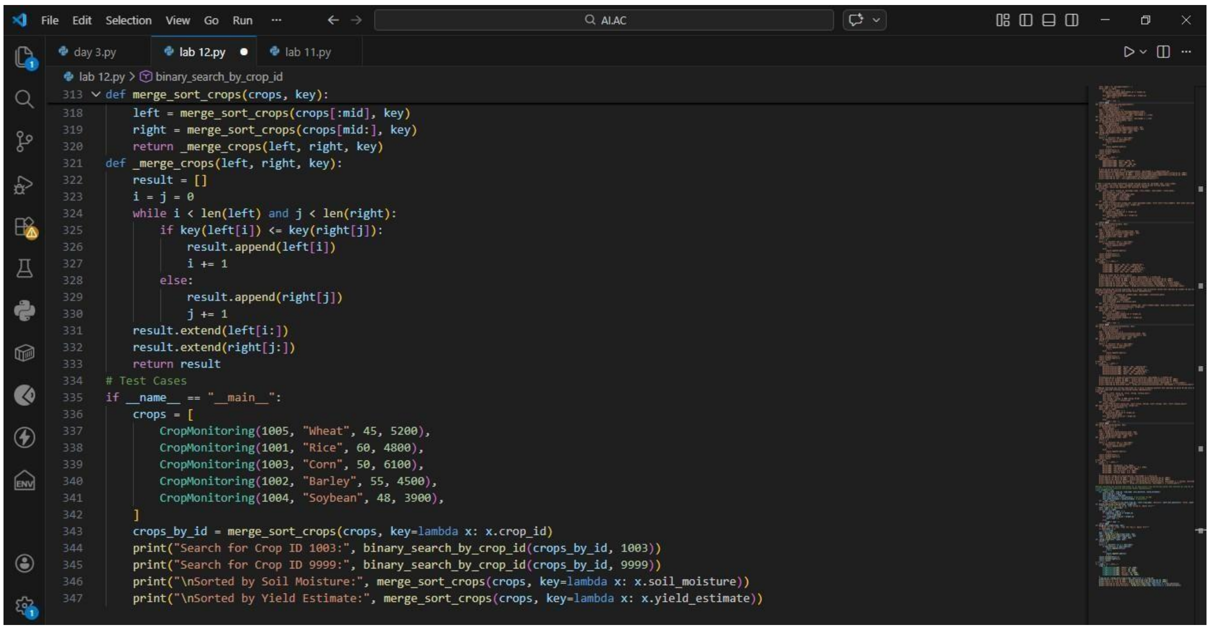
- Sorting by rating supports recommendation systems.

## Task 7: Smart Agriculture Crop Monitoring System

**Prompt :** Suggest suitable searching and sorting algorithms for an agriculture crop monitoring system that searches crops by crop ID and sorts by soil moisture or yield estimate. Justify and implement in Python.

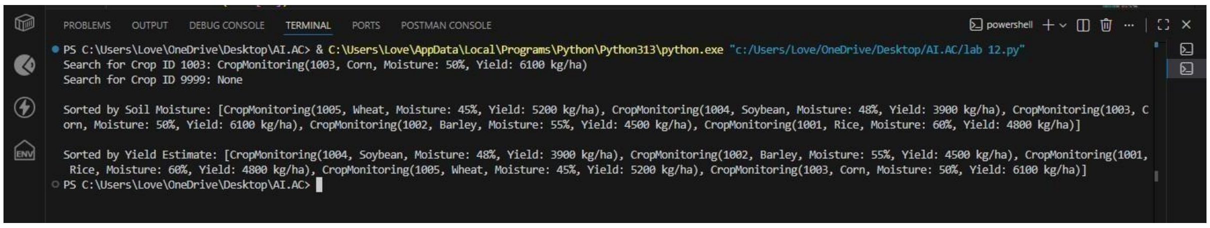


```python
#Design searching and sorting algorithms for an agriculture crop monitoring system that searches by crop ID and sorts by soil moisture or yield e
# Justify algorithm selection and provide Python implementation.
class CropMonitoring:
    def __init__(self, crop_id, crop_name, soil_moisture, yield_estimate):
        self.crop_id = crop_id
        self.crop_name = crop_name
        self.soil_moisture = soil_moisture  # percentage (0-100)
        self.yield_estimate = yield_estimate  # kg/hectare
    def __repr__(self):
        return f"CropMonitoring({self.crop_id}, {self.crop_name}, Moisture: {self.soil_moisture}%, Yield: {self.yield_estimate} kg/ha)"
def binary_search_by_crop_id(crops, target_id):
    """Binary search for crop by ID. Time: O(log n), Space: O(1)"""
    left, right = 0, len(crops) - 1
    while left <= right:
        mid = (left + right) // 2
        if crops[mid].crop_id == target_id:
            return crops[mid]
        elif crops[mid].crop_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def merge_sort_crops(crops, key):
    """Merge sort for crops. Time: O(n log n), Space: O(n)"""
    if len(crops) <= 1:
        return crops
    mid = len(crops) // 2
    left = merge_sort_crops(crops[:mid], key)
    right = merge_sort_crops(crops[mid:], key)
    return _merge_crops(left, right, key)
def _merge_crops(left, right, key):
    result = []
```



```python
def merge_sort_crops(crops, key):
    left = merge_sort_crops(crops[:mid], key)
    right = merge_sort_crops(crops[mid:], key)
    return _merge_crops(left, right, key)
def _merge_crops(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    crops = [
        CropMonitoring(1005, "Wheat", 45, 5200),
        CropMonitoring(1001, "Rice", 60, 4800),
        CropMonitoring(1003, "Corn", 50, 6100),
        CropMonitoring(1002, "Barley", 55, 4500),
        CropMonitoring(1004, "Soybean", 48, 3900),
    ]
    crops_by_id = merge_sort_crops(crops, key=lambda x: x.crop_id)
    print("Search for Crop ID 1003:", binary_search_by_crop_id(crops_by_id, 1003))
    print("Search for Crop ID 9999:", binary_search_by_crop_id(crops_by_id, 9999))
    print("\nSorted by Soil Moisture:", merge_sort_crops(crops, key=lambda x: x.soil_moisture))
    print("\nSorted by Yield Estimate:", merge_sort_crops(crops, key=lambda x: x.yield_estimate))
```

## Output:



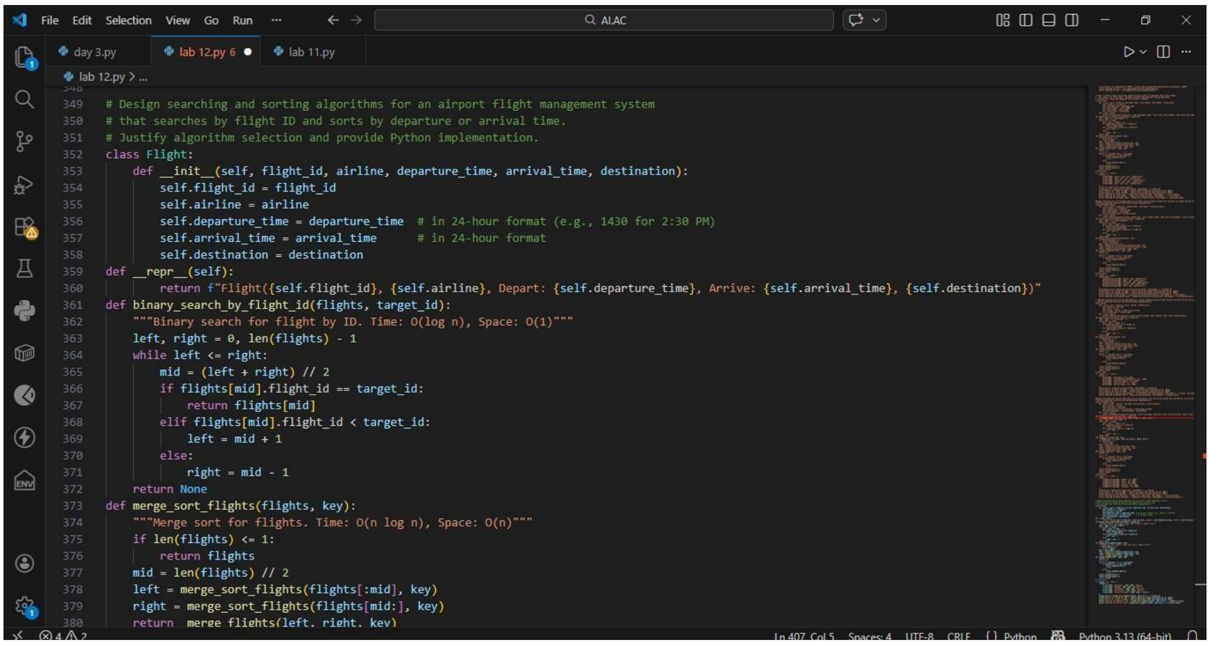
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for Crop ID 1003: CropMonitoring(1003, Corn, Moisture: 50%, Yield: 6100 kg/ha)
Search for Crop ID 9999: None

Sorted by Soil Moisture: [CropMonitoring(1005, Wheat, Moisture: 45%, Yield: 5200 kg/ha), CropMonitoring(1004, Soybean, Moisture: 48%, Yield: 3900 kg/ha), CropMonitoring(1003, C
orn, Moisture: 50%, Yield: 6100 kg/ha), CropMonitoring(1002, Barley, Moisture: 55%, Yield: 4500 kg/ha), CropMonitoring(1001, Rice, Moisture: 60%, Yield: 4800 kg/ha)]

Sorted by Yield Estimate: [CropMonitoring(1004, Soybean, Moisture: 48%, Yield: 3900 kg/ha), CropMonitoring(1002, Barley, Moisture: 55%, Yield: 4500 kg/ha), CropMonitoring(1001,
 Rice, Moisture: 60%, Yield: 4800 kg/ha), CropMonitoring(1005, Wheat, Moisture: 45%, Yield: 5200 kg/ha), CropMonitoring(1003, Corn, Moisture: 50%, Yield: 6100 kg/ha)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

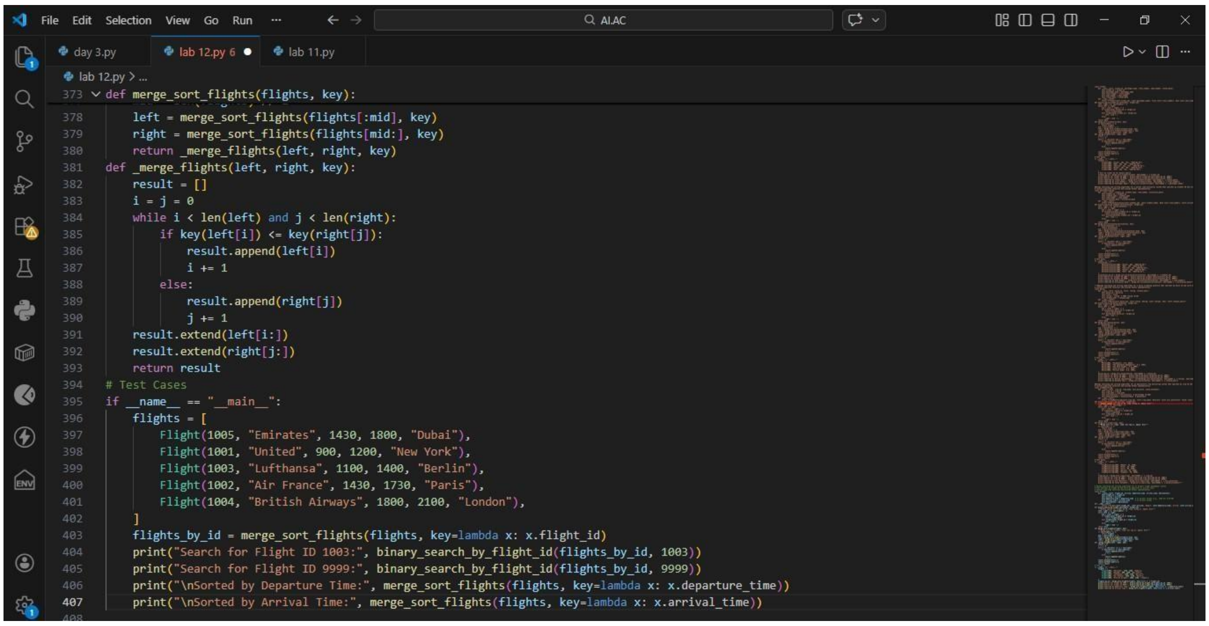
- Farmers need quick data access.
- Sorting helps decision-making.
- Efficient for large monitoring datasets.

## Task 8: Airport Flight Management System

**Prompt:** Design searching and sorting algorithms for an airport flight management system that searches by flight ID and sorts by departure or arrival time. Provide justification and Python implementation.

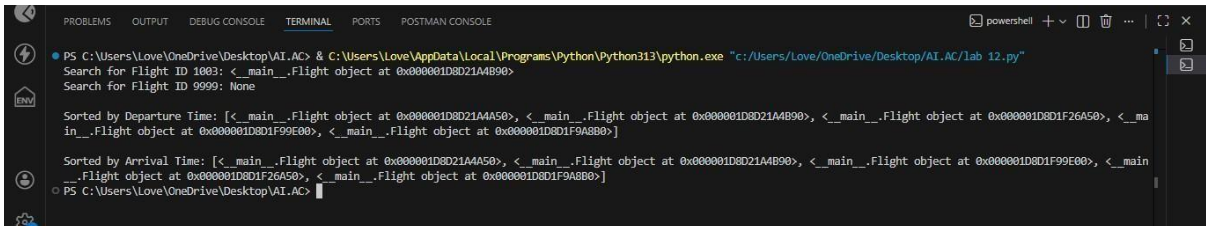


```python
# Design searching and sorting algorithms for an airport flight management system
# that searches by flight ID and sorts by departure or arrival time.
# Justify algorithm selection and provide Python implementation.
class Flight:
    def __init__(self, flight_id, airline, departure_time, arrival_time, destination):
        self.flight_id = flight_id
        self.airline = airline
        self.departure_time = departure_time  # in 24-hour format (e.g., 1430 for 2:30 PM)
        self.arrival_time = arrival_time      # in 24-hour format
        self.destination = destination
def __repr__(self):
        return f"Flight({self.flight_id}, {self.airline}, Depart: {self.departure_time}, Arrive: {self.arrival_time}, {self.destination})"
def binary_search_by_flight_id(flights, target_id):
    """Binary search for flight by ID. Time: O(log n), Space: O(1)"""
    left, right = 0, len(flights) - 1
    while left <= right:
        mid = (left + right) // 2
        if flights[mid].flight_id == target_id:
            return flights[mid]
        elif flights[mid].flight_id < target_id:
            left = mid + 1
        else:
            right = mid - 1
    return None
def merge_sort_flights(flights, key):
    """Merge sort for flights. Time: O(n log n), Space: O(n)"""
    if len(flights) <= 1:
        return flights
    mid = len(flights) // 2
    left = merge_sort_flights(flights[:mid], key)
    right = merge_sort_flights(flights[mid:], key)
    return  merge_flights(left, right, key)
```



```python
def merge_sort_flights(flights, key):
    left = merge_sort_flights(flights[:mid], key)
    right = merge_sort_flights(flights[mid:], key)
    return _merge_flights(left, right, key)
def _merge_flights(left, right, key):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if key(left[i]) <= key(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
# Test Cases
if __name__ == "__main__":
    flights = [
        Flight(1005, "Emirates", 1430, 1800, "Dubai"),
        Flight(1001, "United", 900, 1200, "New York"),
        Flight(1003, "Lufthansa", 1100, 1400, "Berlin"),
        Flight(1002, "Air France", 1430, 1730, "Paris"),
        Flight(1004, "British Airways", 1800, 2100, "London"),
    ]
    flights_by_id = merge_sort_flights(flights, key=lambda x: x.flight_id)
    print("Search for Flight ID 1003:", binary_search_by_flight_id(flights_by_id, 1003))
    print("Search for Flight ID 9999:", binary_search_by_flight_id(flights_by_id, 9999))
    print("\nSorted by Departure Time:", merge_sort_flights(flights, key=lambda x: x.departure_time))
    print("\nSorted by Arrival Time:", merge_sort_flights(flights, key=lambda x: x.arrival_time))
```

## Output:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
Search for Flight ID 1003: <__main__.Flight object at 0x000001D8D21A4B90>
Search for Flight ID 9999: None

Sorted by Departure Time: [<__main__.Flight object at 0x000001D8D21A4A50>, <__main__.Flight object at 0x000001D8D21A4B90>, <__main__.Flight object at 0x000001D8D1F26A50>, <__ma
in__.Flight object at 0x000001D8D1F99E00>, <__main__.Flight object at 0x000001D8D1F9A8B0>]

Sorted by Arrival Time: [<__main__.Flight object at 0x000001D8D21A4A50>, <__main__.Flight object at 0x000001D8D21A4B90>, <__main__.Flight object at 0x000001D8D1F99E00>, <__main
__.Flight object at 0x000001D8D1F26A50>, <__main__.Flight object at 0x000001D8D1F9A8B0>]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

## Explanation:

- Airports manage thousands of flights. Fast lookup is critical.
- Time-based sorting must be accurate.