

ASSIGNMENT 12.5

Algorithms with AI Assistance – Sorting, Searching & Optimizing Algorithms

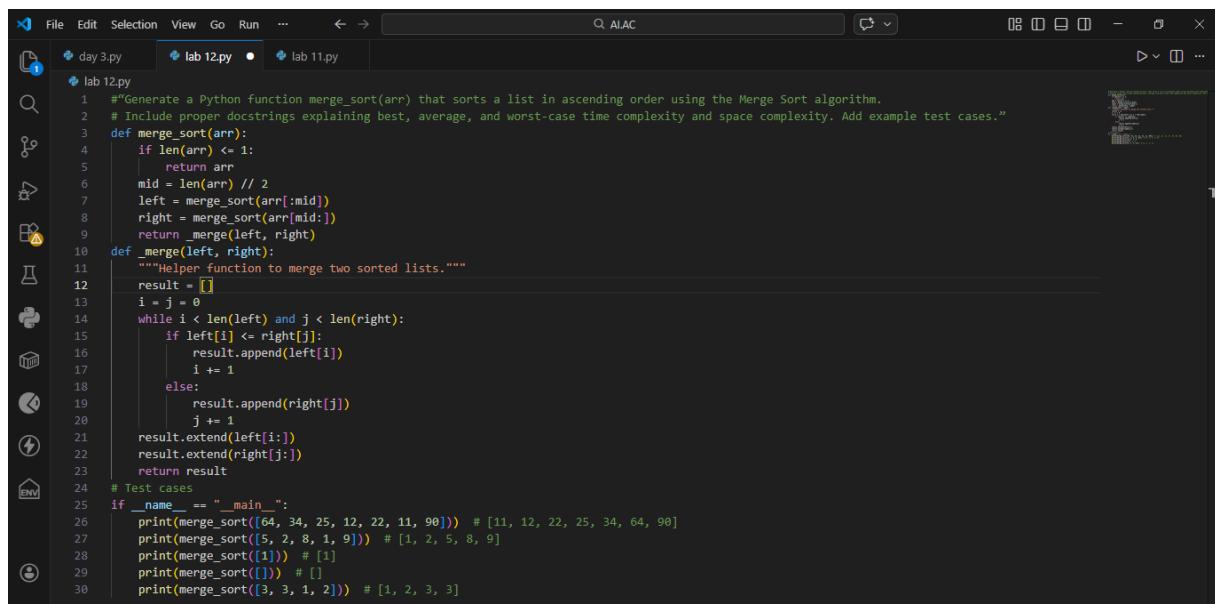
Atla Sreeja

2403A51L02

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.



```
#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:



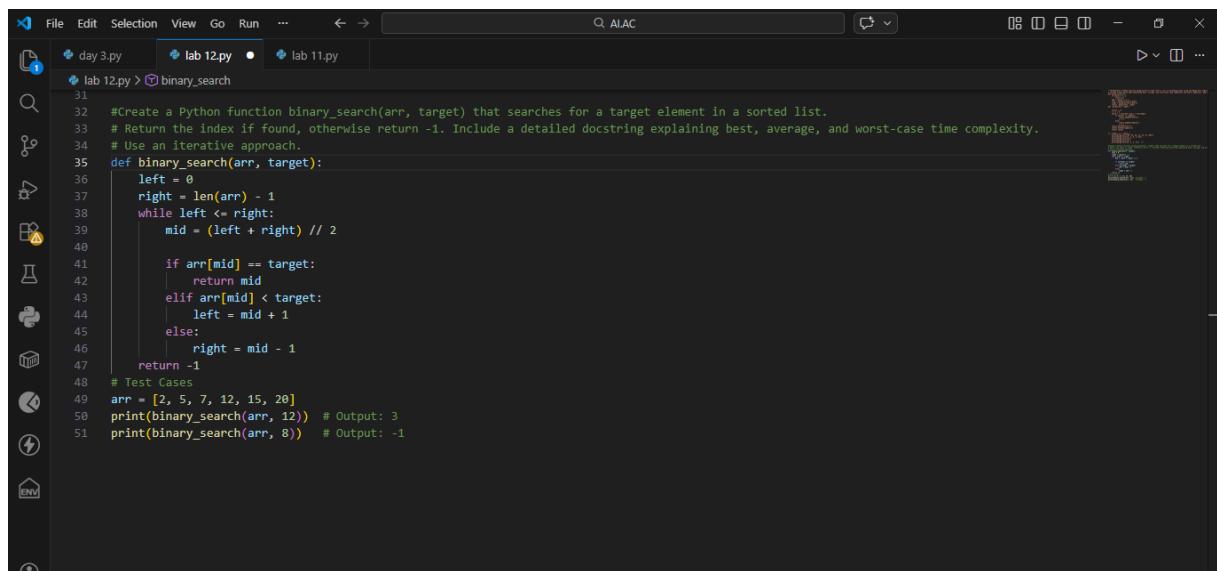
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:/Users/Love/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab_12.py"
[11, 12, 22, 25, 34, 64, 90]
[1, 2, 5, 8, 9]
[1]
[]
[1, 2, 3, 3]
```

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.



A screenshot of a code editor showing a Python file named `lab 12.py`. The code defines an iterative binary search function. It initializes `left` to 0 and `right` to the length of the array minus one. It then enters a loop where it calculates the middle index `mid` as the average of `left` and `right`. If the element at `arr[mid]` matches the target, it returns `mid`. If it's less than the target, it sets `left` to `mid + 1`. If it's greater, it sets `right` to `mid - 1`. If no match is found after the loop, it returns -1. Test cases are provided for targets 12 and 8.

```
31 #Create a Python function binary_search(arr, target) that searches for a target element in a sorted list.
32 # Return the index if found, otherwise return -1. Include a detailed docstring explaining best, average, and worst-case time complexity.
33 # Use an iterative approach.
34
35 def binary_search(arr, target):
36     left = 0
37     right = len(arr) - 1
38     while left <= right:
39         mid = (left + right) // 2
40
41         if arr[mid] == target:
42             return mid
43         elif arr[mid] < target:
44             left = mid + 1
45         else:
46             right = mid - 1
47     return -1
48 # Test Cases
49 arr = [2, 5, 7, 12, 15, 20]
50 print(binary_search(arr, 12)) # Output: 3
51 print(binary_search(arr, 8)) # Output: -1
```

OUTPUT:



A screenshot of a terminal window showing the execution of the `lab 12.py` script. The terminal shows the command being run and the resulting output. The output includes the sorted array [1, 2, 5, 8, 9] and the results of two search operations: finding 12 at index 3 and failing to find 8 at index -1.

```
[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.

```
#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.

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

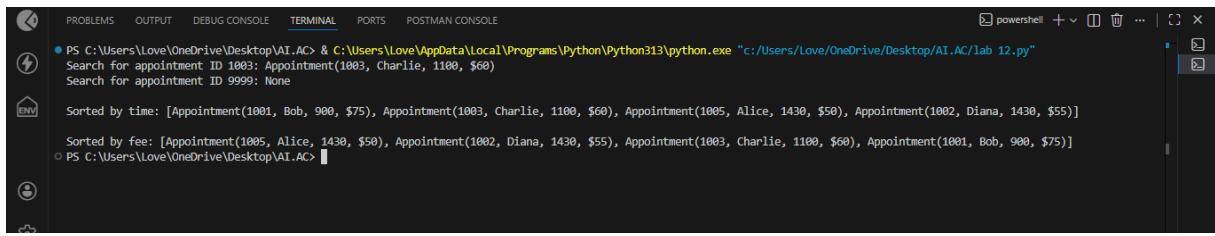
def sort_appointments_by_time(appointments):
    if len(appointments) <= 1:
        return appointments
    mid = len(appointments) // 2
    left = sort_appointments_by_time(appointments[:mid])
    right = sort_appointments_by_time(appointments[mid:])
    return _merge_appointments(left, right, key=lambda 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):
    if len(appointments) <= 1:
        return appoitments[0]
    left = _merge_sort_helper(appointments[:mid], key)
    right = _merge_sort_helper(appointments[mid:], key)
    return _merge_appointments(left, right, key)
```

```
# 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:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
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)]

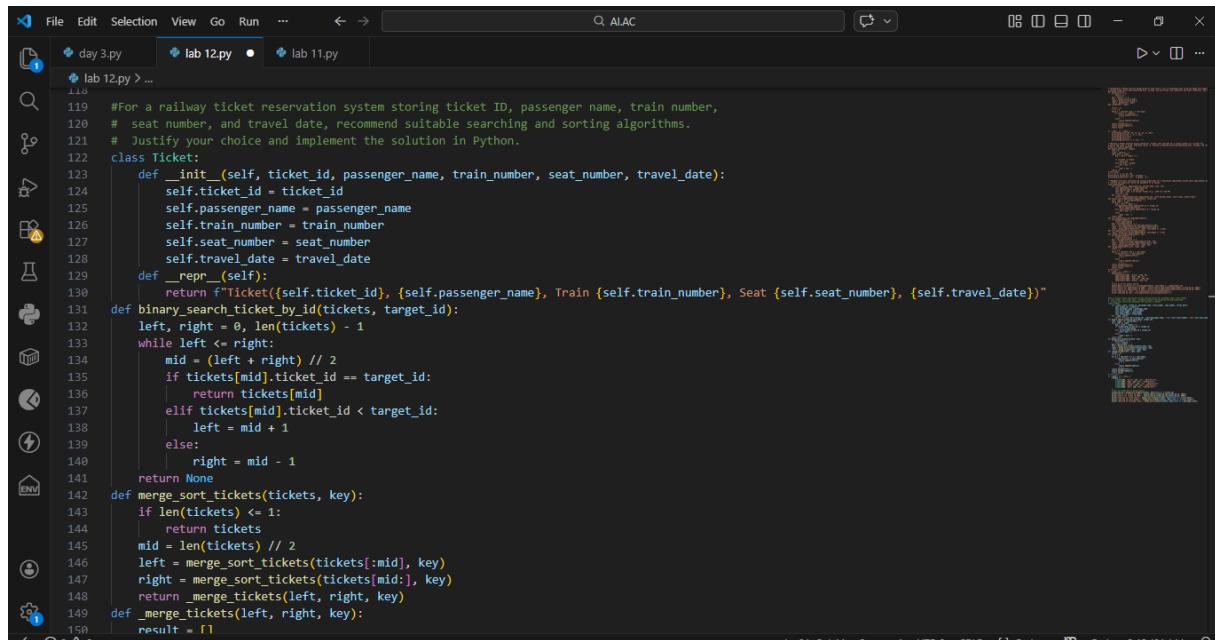
Sorted by fee: [Appointment(1005, Alice, 1430, $50), Appointment(1002, Diana, 1430, $55), Appointment(1003, Charlie, 1100, $60), Appointment(1001, Bob, 900, $75)]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

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.



```
File Edit Selection View Go Run ... ← → Q AI.AC
day 3.py lab 12.py lab 11.py
118 #For a railway ticket reservation system storing ticket ID, passenger name, train number,
119 # seat number, and travel date, recommend suitable searching and sorting algorithms.
120 # Justify your choice and implement the solution in Python.
121
122 class Ticket:
123     def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
124         self.ticket_id = ticket_id
125         self.passenger_name = passenger_name
126         self.train_number = train_number
127         self.seat_number = seat_number
128         self.travel_date = travel_date
129     def __repr__(self):
130         return f"Ticket({self.ticket_id}, {self.passenger_name}, Train {self.train_number}, Seat {self.seat_number}, {self.travel_date})"
131     def binary_search_ticket_by_id(tickets, target_id):
132         left, right = 0, len(tickets) - 1
133         while left <= right:
134             mid = (left + right) // 2
135             if tickets[mid].ticket_id == target_id:
136                 return tickets[mid]
137             elif tickets[mid].ticket_id < target_id:
138                 left = mid + 1
139             else:
140                 right = mid - 1
141         return None
142     def merge_sort_tickets(tickets, key):
143         if len(tickets) <= 1:
144             return tickets
145         mid = len(tickets) // 2
146         left = merge_sort_tickets(tickets[:mid], key)
147         right = merge_sort_tickets(tickets[mid:], key)
148         return merge_tickets(left, right, key)
149     def merge_tickets(left, right, key):
150         result = []
151         i, j = 0, 0
```

```

day 3.py    lab 12.py  lab 11.py
lab 12.py > ...
142     def merge_sort_tickets(tickets, key):
143         return _merge_tickets(left, right, key)
144     def _merge_tickets(left, right, key):
145         result = []
146         i = j = 0
147         while i < len(left) and j < len(right):
148             if key(left[i]) <= key(right[j]):
149                 result.append(left[i])
150                 i += 1
151             else:
152                 result.append(right[j])
153                 j += 1
154             result.extend(left[i:])
155             result.extend(right[j:])
156         return result
157     # Test Cases
158     if __name__ == "__main__":
159         tickets = [
160             Ticket(1005, "Alice", 101, "A1", "2024-01-15"),
161             Ticket(1001, "Bob", 102, "B2", "2024-01-10"),
162             Ticket(1003, "Charlie", 101, "C3", "2024-01-15"),
163             Ticket(1002, "Diana", 103, "D4", "2024-01-12"),
164             Ticket(1004, "Eve", 102, "E5", "2024-01-10"),
165         ]
166         # Sort by ticket ID for binary search
167         tickets_by_id = merge_sort_tickets(tickets, key=lambda x: x.ticket_id)
168         print("Search for Ticket ID 1003:", binary_search_ticket_by_id(tickets_by_id, 1003))
169         print("Search for Ticket ID 9999:", binary_search_ticket_by_id(tickets_by_id, 9999))
170         print("\nSorted by Travel Date:", merge_sort_tickets(tickets, key=lambda x: x.travel_date))
171         print("\nSorted by Train Number:", merge_sort_tickets(tickets, key=lambda x: x.train_number))
172         print("\nSorted by Passenger Name:", merge_sort_tickets(tickets, key=lambda x: x.passenger_name))
173
174
175
176
177

```

OUTPUT:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
powershell + - x
● 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, 2024-01-15), 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.

```
File Edit Selection View Go Run ... < > Q AIAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > ...
178 #Design searching and sorting algorithms for a hostel room allocation system that searches by student ID and sorts by room number or allocation date
179 # Justify algorithm selection and provide Python implementation.
180
181 class HostelAllocation:
182     def __init__(self, student_id, student_name, room_number, allocation_date):
183         self.student_id = student_id
184         self.student_name = student_name
185         self.room_number = room_number
186         self.allocation_date = allocation_date
187     def __repr__(self):
188         return f"HostelAllocation({self.student_id}, {self.student_name}, Room {self.room_number}, {self.allocation_date})"
189     def binary_search_by_student_id(allocations, target_id):
190         left, right = 0, len(allocations) - 1
191         while left <= right:
192             mid = (left + right) // 2
193             if allocations[mid].student_id == target_id:
194                 return allocations[mid]
195             elif allocations[mid].student_id < target_id:
196                 left = mid + 1
197             else:
198                 right = mid - 1
199         return None
200     def merge_sort_allocations(allocations, key):
201         if len(allocations) <= 1:
202             return allocations
203         mid = len(allocations) // 2
204         left = merge_sort_allocations(allocations[:mid], key)
205         right = merge_sort_allocations(allocations[mid:], key)
206         return _merge_allocations(left, right, key)
207     def _merge_allocations(left, right, key):
208         result = []
209         i = j = 0
210         while i < len(left) and j < len(right):
211             if key(left[i]) <= key(right[j]):
212                 result.append(left[i])
213                 i += 1
214             else:
215                 result.append(right[j])
216                 j += 1
217         result.extend(left[i:])
218         result.extend(right[j:])
219         return result
220
221 # Test Cases
222 if __name__ == "__main__":
223     allocations = [
224         HostelAllocation(1005, "Alice", 201, "2024-01-10"),
225         HostelAllocation(1001, "Bob", 105, "2024-01-05"),
226         HostelAllocation(1003, "Charlie", 302, "2024-01-15"),
227         HostelAllocation(1002, "Diana", 203, "2024-01-08"),
228         HostelAllocation(1004, "Eve", 101, "2024-01-12"),
229     ]
230     allocations_by_id = merge_sort_allocations(allocations, key=lambda x: x.student_id)
231     print("Search for Student ID 1003:", binary_search_by_student_id(allocations_by_id, 1003))
232     print("Search for Student ID 9999:", binary_search_by_student_id(allocations_by_id, 9999))
233     print("\nSorted by Room Number:", merge_sort_allocations(allocations, key=lambda x: x.room_number))
234     print("\nSorted by Allocation Date:", merge_sort_allocations(allocations, key=lambda x: x.allocation_date))
```

```
File Edit Selection View Go Run ... < > Q AIAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > ...
200     def merge_sort_allocations(allocations, key):
201         left = merge_sort_allocations(allocations[:mid], key)
202         right = merge_sort_allocations(allocations[mid:], key)
203         return _merge_allocations(left, right, key)
204     def _merge_allocations(left, right, key):
205         result = []
206         i = j = 0
207         while i < len(left) and j < len(right):
208             if key(left[i]) <= key(right[j]):
209                 result.append(left[i])
210                 i += 1
211             else:
212                 result.append(right[j])
213                 j += 1
214         result.extend(left[i:])
215         result.extend(right[j:])
216         return result
217
218 # Test Cases
219 if __name__ == "__main__":
220     allocations = [
221         HostelAllocation(1005, "Alice", 201, "2024-01-10"),
222         HostelAllocation(1001, "Bob", 105, "2024-01-05"),
223         HostelAllocation(1003, "Charlie", 302, "2024-01-15"),
224         HostelAllocation(1002, "Diana", 203, "2024-01-08"),
225         HostelAllocation(1004, "Eve", 101, "2024-01-12"),
226     ]
227     allocations_by_id = merge_sort_allocations(allocations, key=lambda x: x.student_id)
228     print("Search for Student ID 1003:", binary_search_by_student_id(allocations_by_id, 1003))
229     print("Search for Student ID 9999:", binary_search_by_student_id(allocations_by_id, 9999))
230     print("\nSorted by Room Number:", merge_sort_allocations(allocations, key=lambda x: x.room_number))
231     print("\nSorted by Allocation Date:", merge_sort_allocations(allocations, key=lambda x: x.allocation_date))
```

OUTPUT:

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.

The screenshot shows a code editor interface with a dark theme. The top bar includes standard file operations like File, Edit, Selection, View, Go, Run, and a search bar labeled 'ALAC'. Below the bar, there are tabs for 'day 3.py', 'lab 12.py' (which is the active tab), and 'lab 11.py'. The main area contains the following Python code:

```
235 #Design searching and sorting algorithms for a movie streaming platform that searches by movie ID and sorts by rating or release year.
236 # Justify algorithm selection and provide Python implementation.
237 class Movie:
238     def __init__(self, movie_id, title, rating, release_year):
239         self.movie_id = movie_id
240         self.title = title
241         self.rating = rating # IMdb rating (0-10)
242         self.release_year = release_year
243     def __repr__(self):
244         return f'Movie({self.movie_id}, {self.title}, Rating: {self.rating}, Year: {self.release_year})'
245 def binary_search_by_movie_id(movies, target_id):
246     left, right = 0, len(movies) - 1
247     while left <= right:
248         mid = (left + right) // 2
249         if movies[mid].movie_id == target_id:
250             return movies[mid]
251         elif movies[mid].movie_id < target_id:
252             left = mid + 1
253         else:
254             right = mid - 1
255     return None
256 def merge_sort_movies(movies, key):
257     if len(movies) <= 1:
258         return movies
259     mid = len(movies) // 2
260     left = merge_sort_movies(movies[:mid], key)
261     right = merge_sort_movies(movies[mid:], key)
262     return _merge_movies(left, right, key)
263 def _merge_movies(left, right, key):
264     result = []
265     i = j = 0
266     while i < len(left) and j < len(right):
```

The screenshot shows a code editor with multiple tabs open. The active tab contains Python code for implementing a merge sort algorithm on a list of movies based on their rating. The code includes a helper function for merging two sorted lists and the main merge sort function. It also includes several test cases using the `Movie` class to verify the correctness of the sorting logic.

```
File Edit Selection View Go Run ... < > Search AIAC

day 3.py lab 12.py ● lab 11.py

lab 12.py > [e] movies_by_id
256 def merge_sort_movies(movies, key):
257     if len(movies) <= 1:
258         return movies
259     mid = len(movies) // 2
260     left = merge_sort_movies(movies[:mid], key)
261     right = merge_sort_movies(movies[mid:], key)
262     return _merge_movies(left, right, key)
263
264 def _merge_movies(left, right, key):
265     result = []
266     i = j = 0
267     while i < len(left) and j < len(right):
268         if key(left[i]) <= key(right[j]):
269             result.append(left[i])
270             i += 1
271         else:
272             result.append(right[j])
273             j += 1
274     result.extend(left[i:])
275     result.extend(right[j:])
276     return result
277
278 # Test Cases
279 if __name__ == "__main__":
280     movies = [
281         Movie(1005, "Inception", 8.8, 2010),
282         Movie(1001, "The Shawshank Redemption", 9.3, 1994),
283         Movie(1003, "The Dark Knight", 9.0, 2008),
284         Movie(1002, "Pulp Fiction", 8.9, 1994),
285         Movie(1004, "Forrest Gump", 8.8, 1994),
286     ]
287     movies_by_id = merge_sort_movies(movies, key=lambda x: x.movie_id)
288     print("Search for Movie ID 1003:", binary_search_by_movie_id(movies_by_id, 1003))
289     print("Search for Movie ID 9999:", binary_search_by_movie_id(movies_by_id, 9999))
290     print("\nSorted by Rating (Descending):", sorted(merge_sort_movies(movies, key=lambda x: x.rating), key=lambda x: x.rating, reverse=True))
291     print("\nSorted by Release Year:", merge_sort_movies(movies, key=lambda x: x.release_year))
```

Output:

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.

```

291     #Design searching and sorting algorithms for an agriculture crop monitoring system that searches by crop ID and sorts by soil moisture or yield estimate
292     # Justify algorithm selection and provide Python implementation.
293     class CropMonitoring:
294         def __init__(self, crop_id, crop_name, soil_moisture, yield_estimate):
295             self.crop_id = crop_id
296             self.crop_name = crop_name
297             self.soil_moisture = soil_moisture # percentage (0-100)
298             self.yield_estimate = yield_estimate # kg/hectare
299         def __repr__(self):
300             return f"CropMonitoring({self.crop_id}, {self.crop_name}, Moisture: {self.soil_moisture}%, Yield: {self.yield_estimate} kg/ha)"
301     def binary_search_by_crop_id(crops, target_id):
302         """Binary search for crop by ID. Time: O(log n), Space: O(1)"""
303         left, right = 0, len(crops) - 1
304         while left <= right:
305             mid = (left + right) // 2
306             if crops[mid].crop_id == target_id:
307                 return crops[mid]
308             elif crops[mid].crop_id < target_id:
309                 left = mid + 1
310             else:
311                 right = mid - 1
312         return None
313     def merge_sort_crops(crops, key):
314         """Merge sort for crops. Time: O(n log n), Space: O(n)"""
315         if len(crops) <= 1:
316             return crops
317         mid = len(crops) // 2
318         left = merge_sort_crops(crops[:mid], key)
319         right = merge_sort_crops(crops[mid:], key)
320         return merge_crops(left, right, key)
321     def _merge_crops(left, right, key):
322         result = []
323         i = j = 0
324         while i < len(left) and j < len(right):
325             if key(left[i]) <= key(right[j]):
326                 result.append(left[i])
327                 i += 1
328             else:
329                 result.append(right[j])
330                 j += 1
331         result.extend(left[i:])
332         result.extend(right[j:])
333         return result
334     # Test Cases
335     if __name__ == "__main__":
336         crops = [
337             CropMonitoring(1005, "Wheat", 45, 5200),
338             CropMonitoring(1001, "Rice", 60, 4800),
339             CropMonitoring(1003, "Corn", 50, 6100),
340             CropMonitoring(1002, "Barley", 55, 4500),
341             CropMonitoring(1004, "Soybean", 48, 3900),
342         ]
343         crops_by_id = merge_sort_crops(crops, key=lambda x: x.crop_id)
344         print("Search for Crop ID 1003:", binary_search_by_crop_id(crops_by_id, 1003))
345         print("Search for Crop ID 9999:", binary_search_by_crop_id(crops_by_id, 9999))
346         print("\nSorted by Soil Moisture:", merge_sort_crops(crops, key=lambda x: x.soil_moisture))
347         print("\nSorted by Yield Estimate:", merge_sort_crops(crops, key=lambda x: x.yield_estimate))

```

```

313     def merge_sort_crops(crops, key):
314         left = merge_sort_crops(crops[:mid], key)
315         right = merge_sort_crops(crops[mid:], key)
316         return _merge_crops(left, right, key)
317     def _merge_crops(left, right, key):
318         result = []
319         i = j = 0
320         while i < len(left) and j < len(right):
321             if key(left[i]) <= key(right[j]):
322                 result.append(left[i])
323                 i += 1
324             else:
325                 result.append(right[j])
326                 j += 1
327         result.extend(left[i:])
328         result.extend(right[j:])
329         return result
330     # Test Cases
331     if __name__ == "__main__":
332         crops = [
333             CropMonitoring(1005, "Wheat", 45, 5200),
334             CropMonitoring(1001, "Rice", 60, 4800),
335             CropMonitoring(1003, "Corn", 50, 6100),
336             CropMonitoring(1002, "Barley", 55, 4500),
337             CropMonitoring(1004, "Soybean", 48, 3900),
338         ]
339         crops_by_id = merge_sort_crops(crops, key=lambda x: x.crop_id)
340         print("Search for Crop ID 1003:", binary_search_by_crop_id(crops_by_id, 1003))
341         print("Search for Crop ID 9999:", binary_search_by_crop_id(crops_by_id, 9999))
342         print("\nSorted by Soil Moisture:", merge_sort_crops(crops, key=lambda x: x.soil_moisture))
343         print("\nSorted by Yield Estimate:", merge_sort_crops(crops, key=lambda x: x.yield_estimate))

```

Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
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, Corn, 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>

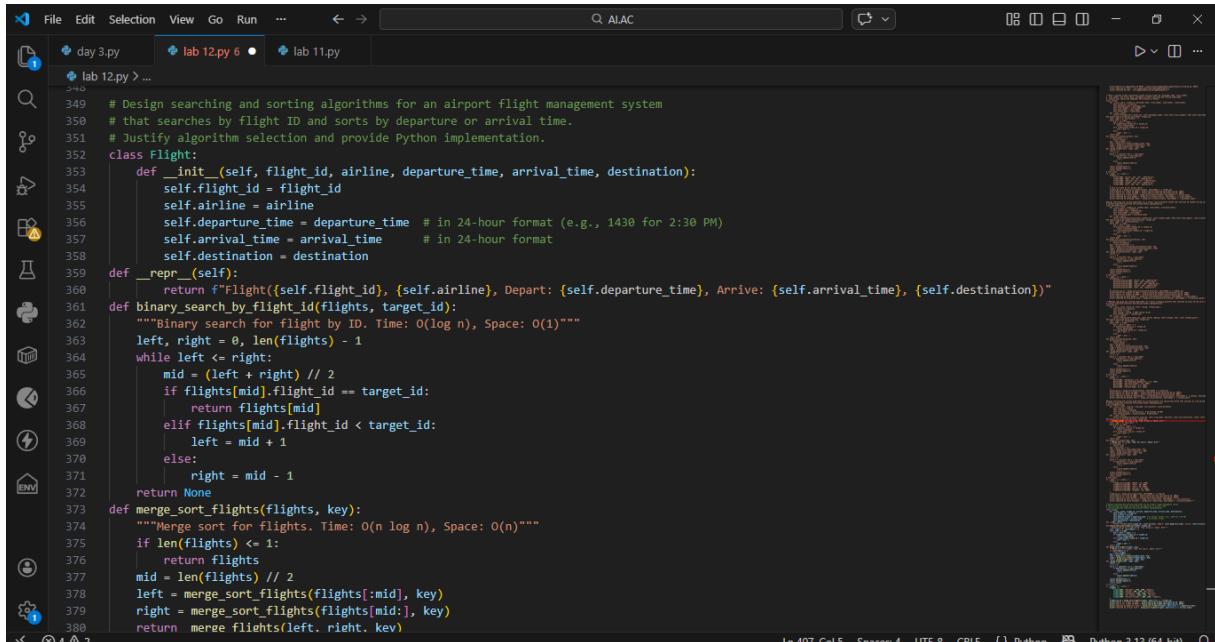
```

Explanation:

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

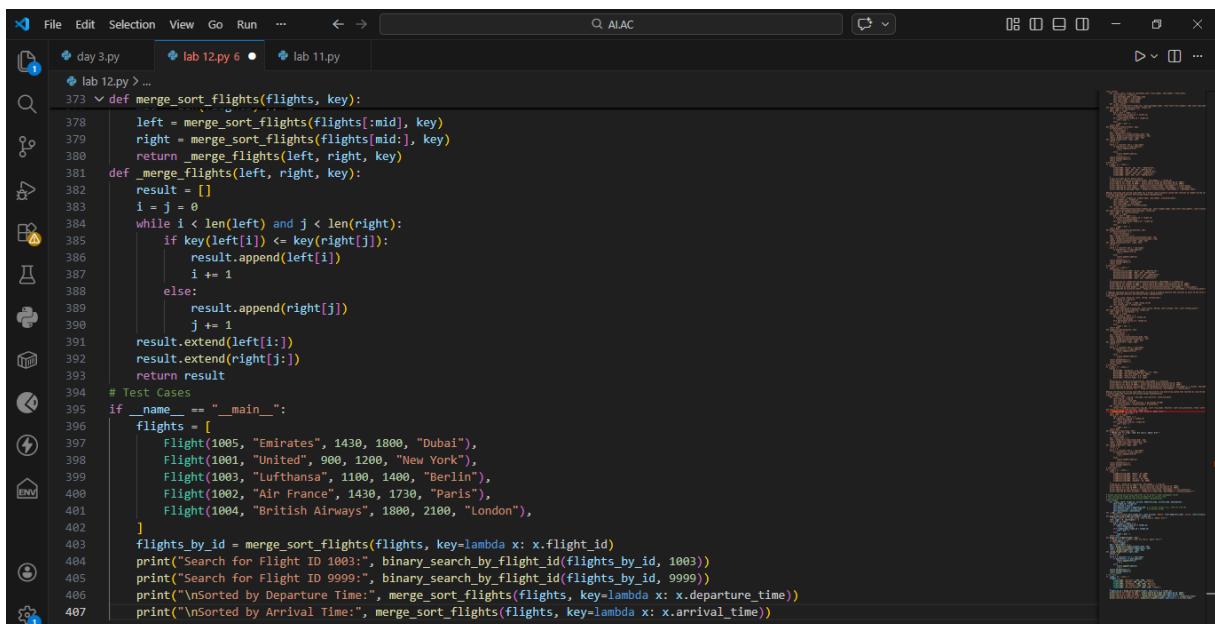
Task 8: Airport Flight Management System

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.



The screenshot shows a code editor window with the following Python code:

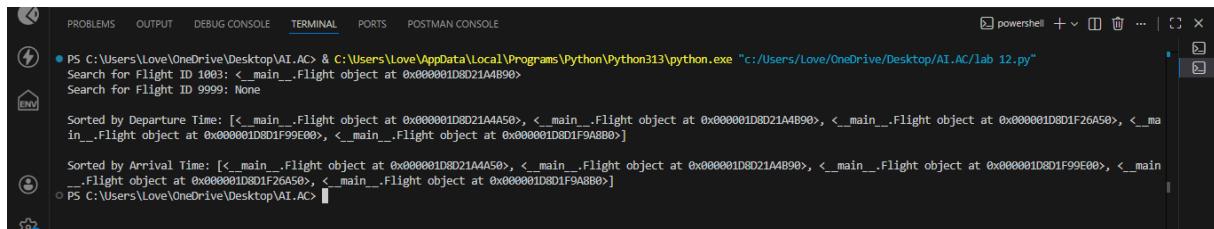
```
File Edit Selection View Go Run ... ← → Q AIAC
day 3.py lab 12.py 6 lab 11.py
346 # Design searching and sorting algorithms for an airport flight management system
347 # that searches by flight ID and sorts by departure or arrival time.
348 # Justify algorithm selection and provide Python implementation.
349
350 class Flight:
351     def __init__(self, flight_id, airline, departure_time, arrival_time, destination):
352         self.flight_id = flight_id
353         self.airline = airline
354         self.departure_time = departure_time # in 24-hour format (e.g., 1430 for 2:30 PM)
355         self.arrival_time = arrival_time # in 24-hour format
356         self.destination = destination
357
358     def __repr__(self):
359         return f"Flight({self.flight_id}, {self.airline}, Depart: {self.departure_time}, Arrive: {self.arrival_time}, {self.destination})"
360
361 def binary_search_by_flight_id(flights, target_id):
362     """Binary search for flight by ID. Time: O(log n), Space: O(1)"""
363     left, right = 0, len(flights) - 1
364     while left <= right:
365         mid = (left + right) // 2
366         if flights[mid].flight_id == target_id:
367             return flights[mid]
368         elif flights[mid].flight_id < target_id:
369             left = mid + 1
370         else:
371             right = mid - 1
372     return None
373
374 def merge_sort_flights(flights, key):
375     """Merge sort for flights. Time: O(n log n), Space: O(n)"""
376     if len(flights) <= 1:
377         return flights
378     mid = len(flights) // 2
379     left = merge_sort_flights(flights[:mid], key)
380     right = merge_sort_flights(flights[mid:], key)
381     return merge(left, right, key)
382
383 def merge(left, right, key):
384     result = []
385     i = j = 0
386     while i < len(left) and j < len(right):
387         if key(left[i]) <= key(right[j]):
388             result.append(left[i])
389             i += 1
390         else:
391             result.append(right[j])
392             j += 1
393     result.extend(left[i:])
394     result.extend(right[j:])
395     return result
396
397 # Test Cases
398 if __name__ == "__main__":
399     flights = [
400         Flight(1005, "Emirates", 1430, 1800, "Dubai"),
401         Flight(1001, "United", 900, 1200, "New York"),
402         Flight(1003, "Lufthansa", 1100, 1400, "Berlin"),
403         Flight(1002, "Air France", 1430, 1730, "Paris"),
404         Flight(1004, "British Airways", 1800, 2100, "London"),
405     ]
406
407     flights_by_id = merge_sort_flights(flights, key=lambda x: x.flight_id)
408     print("Search for Flight ID 1003:", binary_search_by_flight_id(flights_by_id, 1003))
409     print("Search for Flight ID 9999:", binary_search_by_flight_id(flights_by_id, 9999))
410
411     print("\nSorted by Departure Time:", merge_sort_flights(flights, key=lambda x: x.departure_time))
412     print("\nSorted by Arrival Time:", merge_sort_flights(flights, key=lambda x: x.arrival_time))
```



The screenshot shows the continuation of the Python code from the previous editor window:

```
File Edit Selection View Go Run ... ← → Q AIAC
day 3.py lab 12.py 6 lab 11.py
373 def merge_sort_flights(flights, key):
374     left = merge_sort_flights(flights[:mid], key)
375     right = merge_sort_flights(flights[mid:], key)
376     return _merge(left, right, key)
377
378 def _merge(left, right, key):
379     result = []
380     i = j = 0
381     while i < len(left) and j < len(right):
382         if key(left[i]) <= key(right[j]):
383             result.append(left[i])
384             i += 1
385         else:
386             result.append(right[j])
387             j += 1
388     result.extend(left[i:])
389     result.extend(right[j:])
390     return result
391
392 # Test Cases
393 if __name__ == "__main__":
394     flights = [
395         Flight(1005, "Emirates", 1430, 1800, "Dubai"),
396         Flight(1001, "United", 900, 1200, "New York"),
397         Flight(1003, "Lufthansa", 1100, 1400, "Berlin"),
398         Flight(1002, "Air France", 1430, 1730, "Paris"),
399         Flight(1004, "British Airways", 1800, 2100, "London"),
400     ]
401
402     flights_by_id = merge_sort_flights(flights, key=lambda x: x.flight_id)
403     print("Search for Flight ID 1003:", binary_search_by_flight_id(flights_by_id, 1003))
404     print("Search for Flight ID 9999:", binary_search_by_flight_id(flights_by_id, 9999))
405
406     print("\nSorted by Departure Time:", merge_sort_flights(flights, key=lambda x: x.departure_time))
407     print("\nSorted by Arrival Time:", merge_sort_flights(flights, key=lambda x: x.arrival_time))
```

Output:



A screenshot of a terminal window titled "powershell". The window shows the following command and its 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>, <__main__.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 0x000001D8D1F26A50>, <__main__.Flight object at 0x000001D8D1F99E00>, <__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.