

ASSIGNMENT 12.5

Algorithms with AI Assistance – Sorting, Searching & Optimizing Algorithms

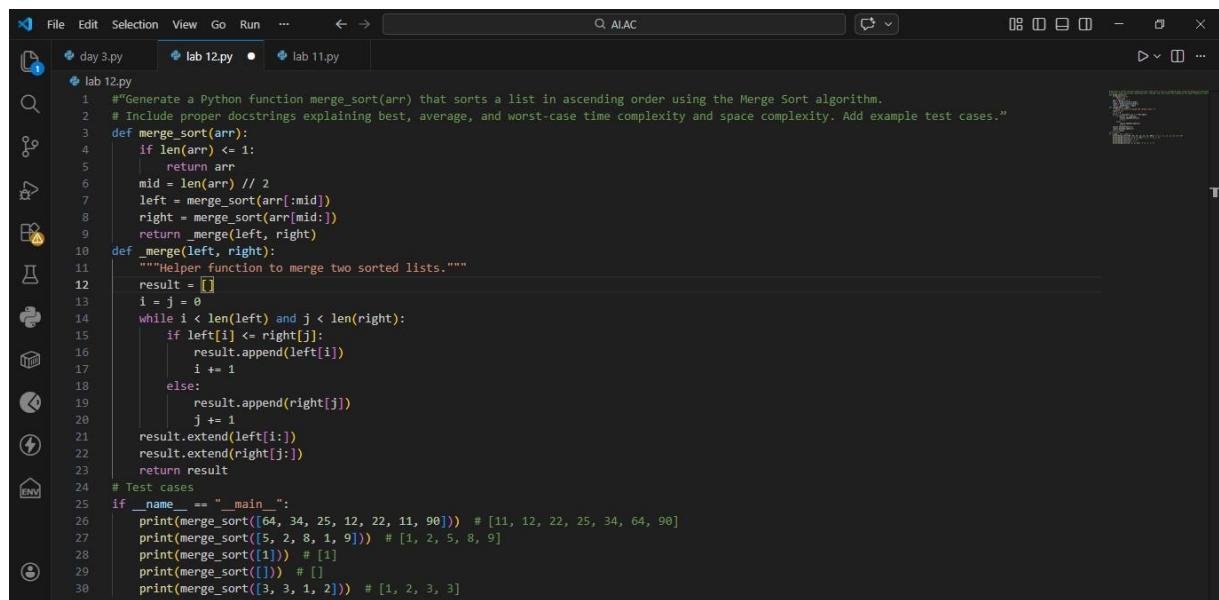
AmruthSagar Vemuganti

2403A51L44

B-52

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.



The screenshot shows a code editor with a dark theme. The file `lab 12.py` is open, containing the following Python code:

```
1  #“Generate a Python function merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm.
2  # Include proper docstrings explaining best, average, and worst-case time complexity and space complexity. Add example test cases.”
3  def merge_sort(arr):
4      if len(arr) <= 1:
5          return arr
6      mid = len(arr) // 2
7      left = merge_sort(arr[:mid])
8      right = merge_sort(arr[mid:])
9      return _merge(left, right)
10 def _merge(left, right):
11     """Helper function to merge two sorted lists."""
12     result = []
13     i = j = 0
14     while i < len(left) and j < len(right):
15         if left[i] < right[j]:
16             result.append(left[i])
17             i += 1
18         else:
19             result.append(right[j])
20             j += 1
21     result.extend(left[i:])
22     result.extend(right[j:])
23     return result
24 # Test cases
25 if __name__ == "__main__":
26     print(merge_sort([64, 34, 25, 12, 22, 11, 90])) # [11, 12, 22, 25, 34, 64, 90]
27     print(merge_sort([5, 2, 8, 1, 9])) # [1, 2, 5, 8, 9]
28     print(merge_sort([1])) # [1]
29     print(merge_sort([])) # []
30     print(merge_sort([3, 3, 1, 2])) # [1, 2, 3, 3]
```

OUTPUT:



The screenshot shows a terminal window with the following output:

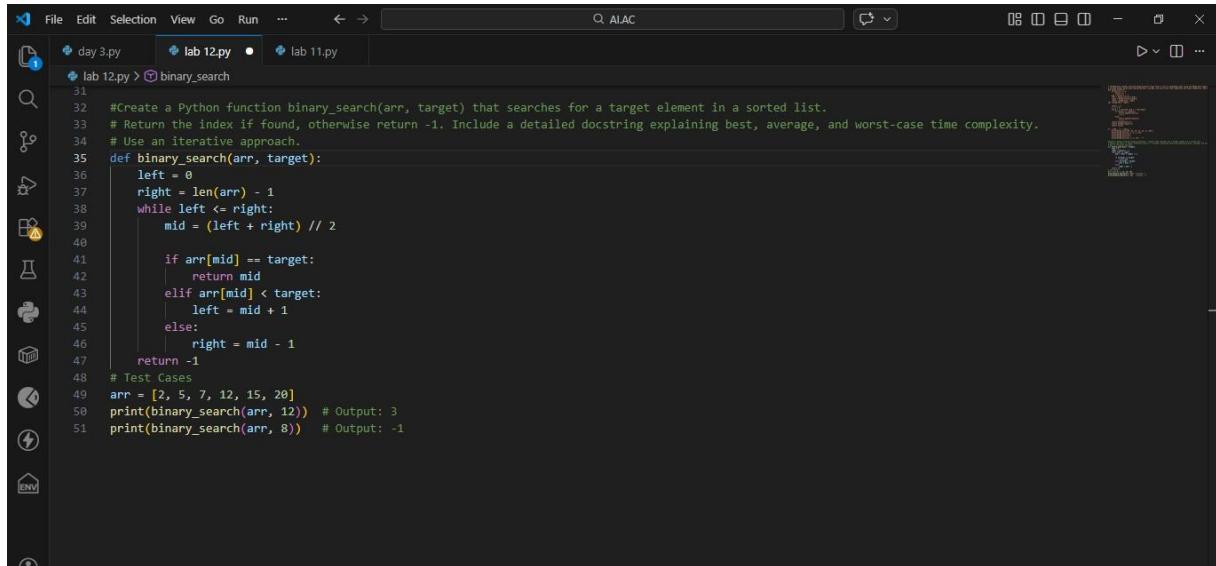
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 12.py"
[11, 12, 22, 25, 34, 64, 90]
[1, 2, 5, 8, 9]
[1]
[]
[1, 2, 3, 3]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

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.



The screenshot shows a code editor window with several tabs at the top: "day 3.py", "lab 12.py" (which is the active tab), and "lab 11.py". The code in "lab 12.py" is as follows:

```
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
49 # Test Cases
50 arr = [2, 5, 7, 12, 15, 20]
51 print(binary_search(arr, 12)) # Output: 3
52 print(binary_search(arr, 8)) # Output: -1
```

OUTPUT:



The screenshot shows a terminal window with the following session:

```
[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.

```

File Edit Selection View Go Run ... ← → Q AIAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > _merge_appointments
53 #Suggest efficient searching and sorting algorithms for a healthcare appointment system where appointments must be searched by appointment ID and
54 # Justify the algorithm choice and implement it in Python.
55 class Appointment:
56     def __init__(self, appointment_id, patient_name, time, fee):
57         self.appointment_id = appointment_id
58         self.patient_name = patient_name
59         self.time = time # in 24-hour format (e.g., 1430 for 2:30 PM)
60         self.fee = fee
61     def __repr__(self):
62         return f'Appointment({self.appointment_id}, {self.patient_name}, {self.time}, ${self.fee})'
63 def binary_search_appointment(appointments, target_id):
64     left, right = 0, len(appointments) - 1
65     while left <= right:
66         mid = (left + right) // 2
67         if appointments[mid].appointment_id == target_id:
68             return appointments[mid]
69         elif appointments[mid].appointment_id < target_id:
70             left = mid + 1
71         else:
72             right = mid - 1
73     return None
74 def sort_appointments_by_time(appointments):
75     if len(appointments) <= 1:
76         return appointments
77     mid = len(appointments) // 2
78     left = sort_appointments_by_time(appointments[:mid])
79     right = sort_appointments_by_time(appointments[mid:])
80     return _merge_appointments(left, right, key=lambda x: x.time)
81 def sort_appointments_by_fee(appointments):
82     """Sort appointments by consultation fee."""
83     return _merge_sort_helper(appointments, key=lambda x: x.fee)
84 def _merge_sort_helper(appointments, key):

```

```

File Edit Selection View Go Run ... ← → Q AIAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > ...
84 def _merge_sort_helper(appointments, key):
85     left = _merge_sort_helper(appointments[:mid], key)
86     right = _merge_sort_helper(appointments[mid:], key)
87     return _merge_appointments(left, right, key)
88 def _merge_appointments(left, right, key):
89     result = []
90     i = j = 0
91     while i < len(left) and j < len(right):
92         if key(left[i]) <= key(right[j]):
93             result.append(left[i])
94             i += 1
95         else:
96             result.append(right[j])
97             j += 1
98     result.extend(left[i:])
99     result.extend(right[j:])
100    return result
101
102 # Test Cases
103 if __name__ == "__main__":
104     appointments = [
105         Appointment(1005, "Alice", 1430, 50),
106         Appointment(1001, "Bob", 0900, 75),
107         Appointment(1003, "Charlie", 1100, 60),
108         Appointment(1002, "Diana", 1430, 55),
109     ]
110
111     # Sort by ID for binary search
112     appointments_sorted_by_id = sorted(appointments, key=lambda x: x.appointment_id)
113     print("Search for appointment ID 1003:", binary_search_appointment(appointments_sorted_by_id, 1003))
114     print("Search for appointment ID 9999:", binary_search_appointment(appointments_sorted_by_id, 9999))
115     print("\nSorted by time:", sort_appointments_by_time(appointments))
116     print("\nSorted by fee:", sort_appointments_by_fee(appointments))
117

```

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 ALAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > ...
119 #For a railway ticket reservation system storing ticket ID, passenger name, train number,
120 # seat number, and travel date, recommend suitable searching and sorting algorithms.
121 # Justify your choice and implement the solution in Python.
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 = []

```

```
File Edit Selection View Go Run ... < > Q AIAC 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
```

The code defines a `merge_sort_tickets` function that takes a list of `Ticket` objects and a key function. It uses a recursive merge sort algorithm to sort the tickets based on the provided key. The function `_merge_tickets` is used to merge two sorted lists into one. Test cases are provided for sorting by ticket ID, passenger name, travel date, and train number.

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

powershell +v ⌂ ⌂ ... | 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.

```

179 #Design searching and sorting algorithms for a hostel room allocation system that searches by student ID and sorts by room number or allocation date.
180 # Justify algorithm selection and provide Python implementation.
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     # Test Cases
221     if __name__ == "__main__":
222         allocations = [
223             HostelAllocation(1005, "Alice", 201, "2024-01-10"),
224             HostelAllocation(1001, "Bob", 105, "2024-01-05"),
225             HostelAllocation(1003, "Charlie", 302, "2024-01-15"),
226             HostelAllocation(1002, "Diana", 203, "2024-01-08"),
227             HostelAllocation(1004, "Eve", 101, "2024-01-12"),
228         ]
229         allocations_by_id = merge_sort_allocations(allocations, key=lambda x: x.student_id)
230         print("Search for Student ID 1003:", binary_search_by_student_id(allocations_by_id, 1003))
231         print("Search for Student ID 9999:", binary_search_by_student_id(allocations_by_id, 9999))
232         print("\nSorted by Room Number:", merge_sort_allocations(allocations, key=lambda x: x.room_number))
233         print("\nSorted by Allocation Date:", merge_sort_allocations(allocations, key=lambda x: x.allocation_date))

```

```

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     # Test Cases
218     if __name__ == "__main__":
219         allocations = [
220             HostelAllocation(1005, "Alice", 201, "2024-01-10"),
221             HostelAllocation(1001, "Bob", 105, "2024-01-05"),
222             HostelAllocation(1003, "Charlie", 302, "2024-01-15"),
223             HostelAllocation(1002, "Diana", 203, "2024-01-08"),
224             HostelAllocation(1004, "Eve", 101, "2024-01-12"),
225         ]
226         allocations_by_id = merge_sort_allocations(allocations, key=lambda x: x.student_id)
227         print("Search for Student ID 1003:", binary_search_by_student_id(allocations_by_id, 1003))
228         print("Search for Student ID 9999:", binary_search_by_student_id(allocations_by_id, 9999))
229         print("\nSorted by Room Number:", merge_sort_allocations(allocations, key=lambda x: x.room_number))
230         print("\nSorted by Allocation Date:", merge_sort_allocations(allocations, key=lambda x: x.allocation_date))

```

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 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.

```

File Edit Selection View Go Run ... < > Q AIAC
day 3.py lab 12.py ● lab 11.py
lab 12.py > movies_by_id
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 the Visual Studio Code interface with the following details:

- File Explorer:** Shows files: day 3.py, lab 12.py (selected), movies_by_id, and lab 11.py.
- Code Editor:** Displays Python code for a merge sort algorithm on a movie database. The code includes functions for merging sorted lists and sorting a list of movies by rating or release year. It also includes test cases for searching by movie ID and sorting by rating or year.

```

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

```

Output:

The terminal window shows the following 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 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.

```

291 #Design searching and sorting algorithms for an agriculture crop monitoring system that searches by crop ID and sorts by soil moisture or yield e
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 = []

```

```

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 powerhell + ×
● 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.

```

File Edit Selection View Go Run ... ↵ → Q ALAC
day 3.py lab 12.py 6 lab 11.py
lab 12.py > ...
349 # Design searching and sorting algorithms for an airport flight management system
350 # that searches by flight ID and sorts by departure or arrival time.
351 # Justify algorithm selection and provide Python implementation.
352 class Flight:
353     def __init__(self, flight_id, airline, departure_time, arrival_time, destination):
354         self.flight_id = flight_id
355         self.airline = airline
356         self.departure_time = departure_time # in 24-hour format (e.g., 1430 for 2:30 PM)
357         self.arrival_time = arrival_time # in 24-hour format
358         self.destination = destination
359     def __repr__(self):
360         return f"Flight({self.flight_id}, {self.airline}, Depart: {self.departure_time}, Arrive: {self.arrival_time}, {self.destination})"
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     def merge_sort_flights(flights, key):
374         """Merge sort for flights. Time: O(n log n), Space: O(n)"""
375         if len(flights) <= 1:
376             return flights
377         mid = len(flights) // 2
378         left = merge_sort_flights(flights[:mid], key)
379         right = merge_sort_flights(flights[mid:], key)
380         return merge(left, right, key)
381     def merge(left, right, key):
382         result = []
383         i = j = 0
384         while i < len(left) and j < len(right):
385             if key(left[i]) <= key(right[j]):
386                 result.append(left[i])
387                 i += 1
388             else:
389                 result.append(right[j])
390                 j += 1
391         result.extend(left[i:])
392         result.extend(right[j:])
393         return result
394     # Test Cases
395     if __name__ == "__main__":
396         flights = [
397             Flight(1005, "Emirates", 1430, 1800, "Dubai"),
398             Flight(1001, "United", 900, 1200, "New York"),
399             Flight(1003, "Lufthansa", 1100, 1400, "Berlin"),
400             Flight(1002, "Air France", 1430, 1730, "Paris"),
401             Flight(1004, "British Airways", 1800, 2100, "London"),
402         ]
403         flights_by_id = merge_sort_flights(flights, key=lambda x: x.flight_id)
404         print("Search for Flight ID 1003:", binary_search_by_flight_id(flights_by_id, 1003))
405         print("Search for Flight ID 9999:", binary_search_by_flight_id(flights_by_id, 9999))
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))

```

```

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

```

Output:

The screenshot shows a terminal window in Visual Studio Code. The title bar says "powershell". The command "python.exe" was run, and the output shows a list of flight objects sorted by departure time and arrival time. The code includes imports for "Flight" and "FlightManager", and defines a "Flight" class with attributes like ID, name, and arrival/departure times.

```
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 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.