

School of Computer Science and Artificial Intelligence**Lab Assignment # 12.5**

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

Submission Starts here**Screenshots:****Task Description #1**

(Sorting – Merge Sort Implementation)

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

• Instructions:

- o Prompt AI to create a function merge_sort(arr) that sorts a list in ascending order.
- o Ask AI to include time complexity and space complexity in the function docstring.
- o Verify the generated code with test cases.

```
ai_coding.py > ...
1  #(Sorting - Merge Sort Implementation)
2  #• Task: Use AI to generate a Python program that implements
3  #the Merge Sort algorithm.
4  #• Instructions:
5  #o Prompt AI to create a function merge_sort(arr) that
6  #sorts a list in ascending order.
7  #o Ask AI to include time complexity and space
8  #complexity in the function docstring.
9  #o Verify the generated code with test cases.
10 def merge_sort(arr):
11     """
12         Sorts a list in ascending order using the Merge Sort algorithm.
13
14         Time Complexity: O(n log n) - where n is the number of elements in the list.
15         Space Complexity: O(n) - due to the temporary arrays used for merging.
16
17         Parameters:
18             arr (list): The list to be sorted.
19
20         Returns:
21             list: A new sorted list in ascending order.
22             """
23
24     if len(arr) <= 1:
25         return arr
26
27     mid = len(arr) // 2
28     left_half = merge_sort(arr[:mid])
29     right_half = merge_sort(arr[mid:])
30
31     return merge(left_half, right_half)
32 def merge(left, right):
33     """Merges two sorted lists into a single sorted list.
34     Parameters:
35         left (list): The first sorted list.
36         right (list): The second sorted list.
37     Returns:
38         list: A merged sorted list.
```

```

ai_coding.py > ...
31  def merge(left, right):
32      """
33          merged = []
34          left_index = right_index = 0
35
36          while left_index < len(left) and right_index < len(right):
37              if left[left_index] < right[right_index]:
38                  merged.append(left[left_index])
39                  left_index += 1
40              else:
41                  merged.append(right[right_index])
42                  right_index += 1
43
44          # If there are remaining elements in left or right, add them to merged
45          merged.extend(left[left_index:])
46          merged.extend(right[right_index:])
47
48      return merged
49
50  # Test cases
51  if __name__ == "__main__":
52      test_cases = [
53          [38, 27, 43, 3, 9, 82, 10],
54          [5, 2, 9, 1, 5, 6],
55          [12, 11, 13, 5, 6, 7],
56          [],
57          [1],
58          [2, 1]
59      ]
60
61      for i, test in enumerate(test_cases):
62          sorted_list = merge_sort(test)
63          print(f"Test Case {i + 1}: {sorted_list}")
64
65
66
67
68
69
70

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

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

```

Task Description #2

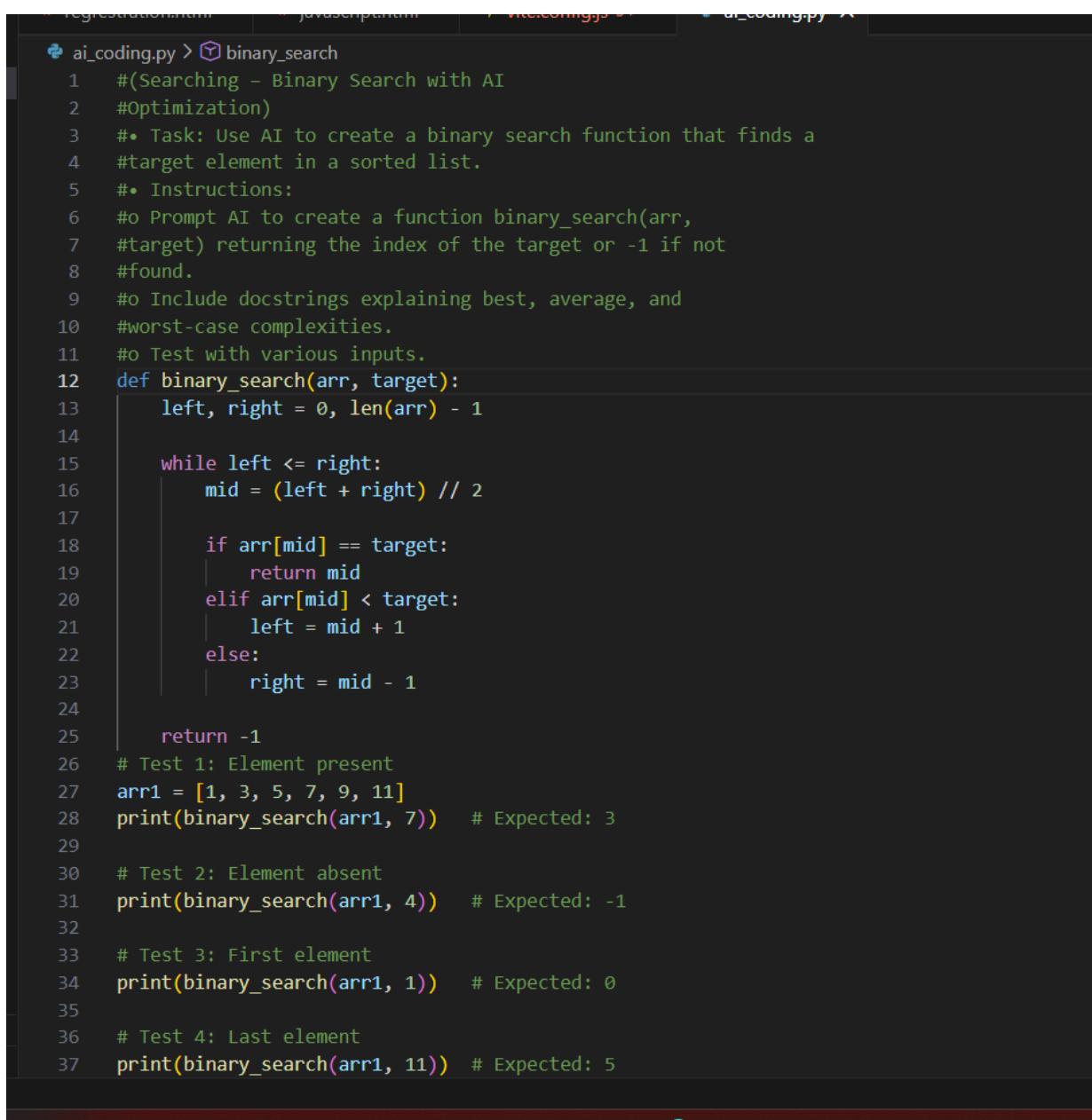
(Searching – Binary Search with AI Optimization)

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

- **Instructions:**

- o Prompt AI to create a function `binary_search(arr, target)` returning the index of the target or -1 if not found.
- o Include docstrings explaining best, average, and worst-case complexities.

- o Test with various inputs.



The screenshot shows a code editor window with a dark theme. The tab bar at the top has four tabs: 'registration.html', 'javascrip.html', 'vitelcoming.js', and 'ai_coding.py'. The 'ai_coding.py' tab is active, displaying the following Python code:

```
ai_coding.py > binary_search
1  #(Searching - Binary Search with AI
2  #Optimization)
3  #• Task: Use AI to create a binary search function that finds a
4  #target element in a sorted list.
5  #• Instructions:
6  #o Prompt AI to create a function binary_search(arr,
7  #target) returning the index of the target or -1 if not
8  #found.
9  #o Include docstrings explaining best, average, and
10 #worst-case complexities.
11 #o Test with various inputs.
12 def binary_search(arr, target):
13     left, right = 0, len(arr) - 1
14
15     while left <= right:
16         mid = (left + right) // 2
17
18         if arr[mid] == target:
19             return mid
20         elif arr[mid] < target:
21             left = mid + 1
22         else:
23             right = mid - 1
24
25     return -1
26 # Test 1: Element present
27 arr1 = [1, 3, 5, 7, 9, 11]
28 print(binary_search(arr1, 7))    # Expected: 3
29
30 # Test 2: Element absent
31 print(binary_search(arr1, 4))    # Expected: -1
32
33 # Test 3: First element
34 print(binary_search(arr1, 1))    # Expected: 0
35
36 # Test 4: Last element
37 print(binary_search(arr1, 11))   # Expected: 5
```

The screenshot shows a code editor window with a dark theme. At the top, there's a file navigation bar with icons for file operations like Open, Save, and New. Below the bar is a code editor area containing Python code for a binary search algorithm and its tests. The code is color-coded for syntax. At the bottom of the editor is a terminal window showing the execution of the script and its output.

```
ai_coding.py > binary_search
30 # Test 2: Element absent
31 print(binary_search(arr1, 4))    # Expected: -1
32
33 # Test 3: First element
34 print(binary_search(arr1, 1))    # Expected: 0
35
36 # Test 4: Last element
37 print(binary_search(arr1, 11))   # Expected: 5
38
39 # Test 5: Single element list
40 arr2 = [10]
41 print(binary_search(arr2, 10))   # Expected: 0
42
43 # Test 6: Empty list
44 arr3 = []
45 print(binary_search(arr3, 5))   # Expected: -1
```

TERMINAL

```
PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

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

Task Description #3: Smart Healthcare Appointment Scheduling System

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

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

Student Task

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

```

❷ ai_coding.py > ...
1  # Task Description #3: Smart Healthcare Appointment Scheduling
2  # System
3  # A healthcare platform maintains appointment records containing
4  # appointment ID, patient name, doctor name, appointment time, and
5  # consultation fee. The system needs to:
6  # 1. Search appointments using appointment ID.
7  # 2. Sort appointments based on time or consultation fee.
8  # Student Task
9  # • Use AI to recommend suitable searching and sorting
10 # algorithms.
11 # • Justify the selected algorithms.
12 # • Implement the algorithms in Python.
13 import datetime
14 class Appointment:
15     def __init__(self, appointment_id, patient_name, doctor_name, appointment_time, consultation_fee):
16         self.appointment_id = appointment_id
17         self.patient_name = patient_name
18         self.doctor_name = doctor_name
19         self.appointment_time = appointment_time
20         self.consultation_fee = consultation_fee
21 class HealthcarePlatform:
22     def __init__(self):
23         self.appointments = []
24     def add_appointment(self, appointment):
25         self.appointments.append(appointment)
26     def search_appointment_by_id(self, appointment_id):
27         # Using a hash map (dictionary) for O(1) average time complexity
28         appointment_dict = {appointment.appointment_id: appointment for appointment in self.appointments}
29         return appointment_dict.get(appointment_id, None)
30     def sort_appointments_by_time(self):
31         # Using Timsort (Python's built-in sort) which is efficient for real-world data
32         return sorted(self.appointments, key=lambda x: x.appointment_time)
33     def sort_appointments_by_fee(self):
34         # Using Timsort (Python's built-in sort) which is efficient for real-world data
35         return sorted(self.appointments, key=lambda x: x.consultation_fee)
36 # Example usage
37 if __name__ == "__main__":

```

In 61 Col 9 Spaces: 4 LITE: 8

```

❷ ai_coding.py > ...
38     platform = HealthcarePlatform()
39     platform.add_appointment(Appointment(1, "Alice", "Dr. Smith", datetime.datetime(2024, 6, 1, 10, 0), 100))
40     platform.add_appointment(Appointment(2, "Bob", "Dr. Jones", datetime.datetime(2024, 6, 1, 11, 0), 150))
41     platform.add_appointment(Appointment(3, "Charlie", "Dr. Brown", datetime.datetime(2024, 6, 1, 9, 0), 120))
42
43     # Search for an appointment by ID
44     appointment = platform.search_appointment_by_id(2)
45     if appointment:
46         print(f"Found appointment: {appointment.patient_name} with {appointment.doctor_name} at {appointment.appointment_time}")
47     else:
48         print("Appointment not found.")
49
50     # Sort appointments by time
51     sorted_by_time = platform.sort_appointments_by_time()
52     print("Appointments sorted by time:")
53     for aptt in sorted_by_time:
54         print(f"{appt.patient_name} with {appt.doctor_name} at {appt.appointment_time}")
55
56     # Sort appointments by consultation fee
57     sorted_by_fee = platform.sort_appointments_by_fee()
58     print("Appointments sorted by consultation fee:")
59     for aptt in sorted_by_fee:
60         print(f"{appt.patient_name} with {appt.doctor_name} at {appt.appointment_time} - Fee: {appt.consultation_fee}")

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

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

```

Task Description #4: Railway Ticket Reservation System

Scenario

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

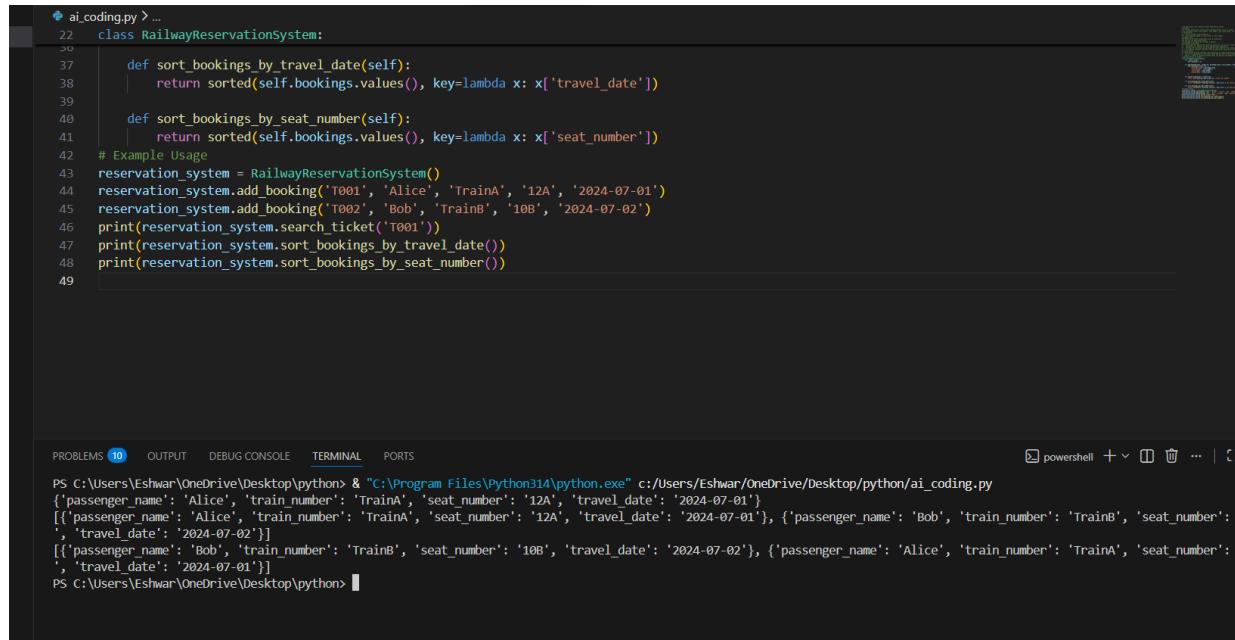
1. Search tickets using ticket ID.
2. Sort bookings based on travel date or seat number.

Student Task

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

```
ai_coding.py > ...
1  # ask Description #4: Railway Ticket Reservation System
2  # Scenario
3  # A railway reservation system stores booking details such as ticket
4  # ID, passenger name, train number, seat number, and travel date. The
5  # system must:
6  # 1. Search tickets using ticket ID.
7  # 2. Sort bookings based on travel date or seat number.
8  # Student Task
9  # • Identify efficient algorithms using AI assistance.
10 # • Justify the algorithm choices.
11 # • Implement searching and sorting in Python.
12 # Efficient Algorithms:
13 # 1. Searching: For searching tickets by ticket ID, we can use a hash table
14 #   (dictionary in Python) for O(1) average time complexity.
15 # 2. Sorting: For sorting bookings based on travel date or seat number, we can
16 #   use Timsort (Python's built-in sorting algorithm) which has O(n log n) time complexity.
17 # Justification:
18 # - Hash tables provide constant time complexity for search operations, making them ideal for quickly retrieving
19 # - Timsort is optimized for real-world data and performs well on partially sorted data,
20 #   which is common in booking systems where new entries are added frequently.
21 # Implementation in Python:
22 class RailwayReservationSystem:
23     def __init__(self):
24         self.bookings = {}
25
26     def add_booking(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
27         self.bookings[ticket_id] = {
28             'passenger_name': passenger_name,
29             'train_number': train_number,
30             'seat_number': seat_number,
31             'travel_date': travel_date
32         }
33
34     def search_ticket(self, ticket_id):
35         return self.bookings.get(ticket_id, "Ticket not found")
36
37     def sort_bookings_by_travel_date(self):
```

Ln 49 Col 1 Spaces: 4 UFT-8



```

ai_coding.py > ...
22  class RailwayReservationSystem:
30
37      def sort_bookings_by_travel_date(self):
38          return sorted(self.bookings.values(), key=lambda x: x['travel_date'])
39
40      def sort_bookings_by_seat_number(self):
41          return sorted(self.bookings.values(), key=lambda x: x['seat_number'])
42
43      # Example Usage
44      reservation_system = RailwayReservationSystem()
45      reservation_system.add_booking('T001', 'Alice', 'TrainA', '12A', '2024-07-01')
46      reservation_system.add_booking('T002', 'Bob', 'TrainB', '10B', '2024-07-02')
47      print(reservation_system.search_ticket('T001'))
48      print(reservation_system.sort_bookings_by_travel_date())
49      print(reservation_system.sort_bookings_by_seat_number())

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + ⌂ ⌂ ⌂ ⌂ ⌂

```

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

```

Task Description #5: Smart Hostel Room Allocation System

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

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

Student Task

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

```

ai_coding.py > ...
1  # Task Description #5: Smart Hostel Room Allocation System
2  # A hostel management system stores student room allocation details
3  # including student ID, room number, floor, and allocation date. The
4  # system needs to:
5  # 1. Search allocation details using student ID.
6  # 2. Sort records based on room number or allocation date.
7  # Student Task
8  # • Use AI to suggest optimized algorithms.
9  # • Justify the selections.
10 # • Implement the solution in Python
11 import datetime
12 class HostelManagementsystem:
13     def __init__(self):
14         self.allocations = []
15
16     def add_allocation(self, student_id, room_number, floor, allocation_date):
17         allocation = {
18             'student_id': student_id,
19             'room_number': room_number,
20             'floor': floor,
21             'allocation_date': allocation_date
22         }
23         self.allocations.append(allocation)
24
25     def search_by_student_id(self, student_id):
26         for allocation in self.allocations:
27             if allocation['student_id'] == student_id:
28                 return allocation
29         return None
30
31     def sort_by_room_number(self):
32         return sorted(self.allocations, key=lambda x: x['room_number'])
33
34     def sort_by_allocation_date(self):
35         return sorted(self.allocations, key=lambda x: x['allocation_date'])
36
37 # Example usage
38 hostel_system = HostelManagementsystem()

```

```

ai_coding.py > ...
36  # Example usage
37  hostel_system = HostelManagementsystem()
38  hostel_system.add_allocation('s001', '101A', 1, datetime.date(2023, 9, 1))
39  hostel_system.add_allocation('s002', '102B', 1, datetime.date(2023, 9, 2))
40  hostel_system.add_allocation('s003', '201A', 2, datetime.date(2023, 9, 3))
41  # Search for a student allocation
42  allocation = hostel_system.search_by_student_id('s002')
43  print(allocation)
44  # Sort by room number
45  sorted_by_room = hostel_system.sort_by_room_number()
46  print(sorted_by_room)
47  # Sort by allocation date
48  sorted_by_date = hostel_system.sort_by_allocation_date()
49  print(sorted_by_date)
50

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Eshwar\OneDrive\Desktop\python> & "c:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py

```

{'student_id': 's002', 'room_number': '102B', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 2)}
[{'student_id': 's001', 'room_number': '101A', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 1)}, {'student_id': 's002', 'room_number': '102B', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 2)}, {"student_id": "s003", "room_number": "201A", "floor": 2, "allocation_date": datetime.date(2023, 9, 3)}]
[{'student_id': 's001', 'room_number': '101A', 'floor': 1, 'allocation_date': datetime.date(2023, 9, 1)}, {"student_id": "s002", "room_number": "102B", "floor": 1, "allocation_date": datetime.date(2023, 9, 2)}, {"student_id": "s003", "room_number": "201A", "floor": 2, "allocation_date": datetime.date(2023, 9, 3)}]

```

PS C:\Users\Eshwar\OneDrive\Desktop\python>

Task Description #6: Online Movie Streaming Platform

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

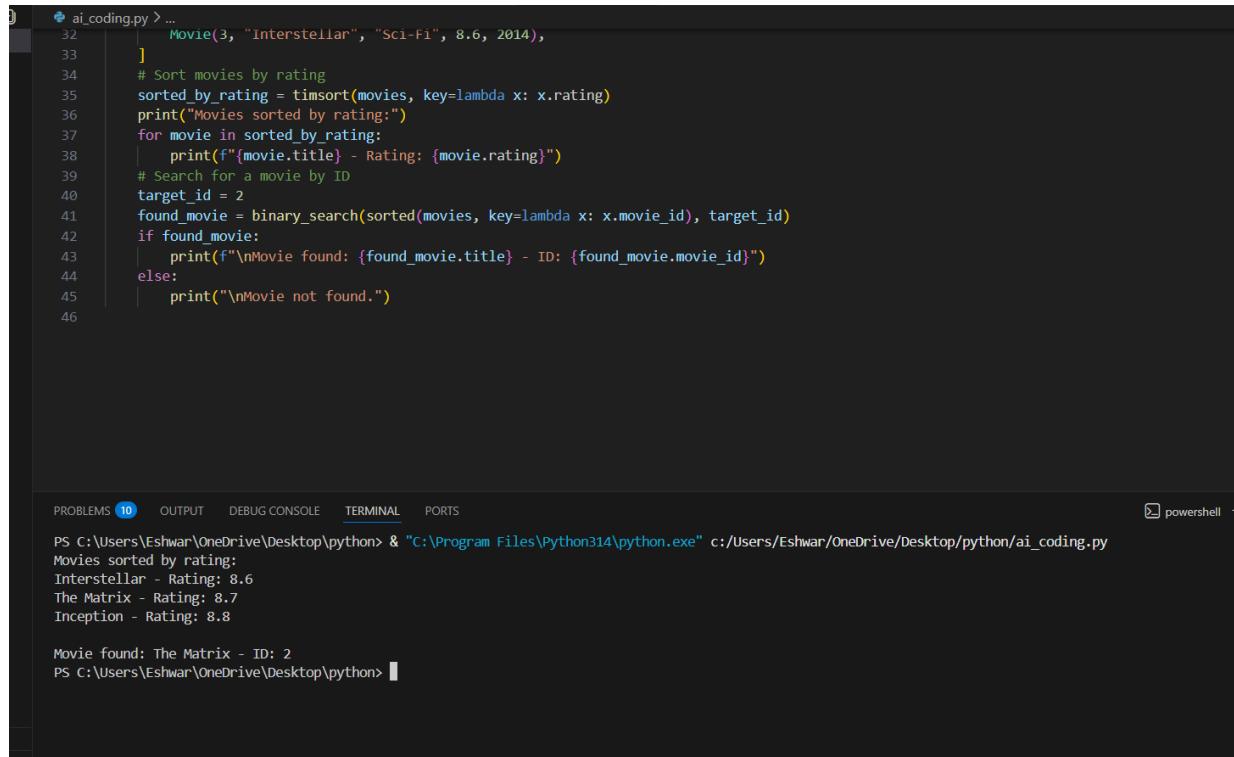
1. Search movies by movie ID.
2. Sort movies based on rating or release year.

Student Task

- Recommend searching and sorting algorithms using AI.

- Justify the chosen algorithms.
- Implement Python functions

```
ai_coding.py > ...
1  # Task Description #6: Online Movie Streaming Platform
2  # A streaming service maintains movie records with movie ID, title,
3  # genre, rating, and release year. The platform needs to:
4  # 1. Search movies by movie ID.
5  # 2. Sort movies based on rating or release year.
6  # Student Task:
7  class Movie:
8      def __init__(self, movie_id, title, genre, rating, release_year):
9          self.movie_id = movie_id
10         self.title = title
11         self.genre = genre
12         self.rating = rating
13         self.release_year = release_year
14     def binary_search(movies, target_id):
15         left, right = 0, len(movies) - 1
16         while left <= right:
17             mid = left + (right - left) // 2
18             if movies[mid].movie_id == target_id:
19                 return movies[mid]
20             elif movies[mid].movie_id < target_id:
21                 left = mid + 1
22             else:
23                 right = mid - 1
24         return None
25     def timsort(movies, key):
26         return sorted(movies, key=key)
27 # Example Usage
28 if __name__ == "__main__":
29     movies = [
30         Movie(1, "Inception", "Sci-Fi", 8.8, 2010),
31         Movie(2, "The Matrix", "Action", 8.7, 1999),
32         Movie(3, "Interstellar", "Sci-Fi", 8.6, 2014),
33     ]
34     # Sort movies by rating
35     sorted_by_rating = timsort(movies, key=lambda x: x.rating)
36     print("Movies sorted by rating:")
37     for movie in sorted_by_rating:
```



```

ai.coding.py > ...
32     Movie(3, "Interstellar", "Sci-Fi", 8.6, 2014),
33 ]
34 # Sort movies by rating
35 sorted_by_rating = timsort(movies, key=lambda x: x.rating)
36 print("Movies sorted by rating:")
37 for movie in sorted_by_rating:
38     print(f"{movie.title} - Rating: {movie.rating}")
39 # Search for a movie by ID
40 target_id = 2
41 found_movie = binary_search(sorted(movies, key=lambda x: x.movie_id), target_id)
42 if found_movie:
43     print(f"\nMovie found: {found_movie.title} - ID: {found_movie.movie_id}")
44 else:
45     print("\nMovie not found.")

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/OneDrive/Desktop/python/ai_coding.py
Movies sorted by rating:
Interstellar - Rating: 8.6
The Matrix - Rating: 8.7
Inception - Rating: 8.8

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

```

Task Description #7: Smart Agriculture Crop Monitoring System

An agriculture monitoring system stores crop data with crop ID, crop name, soil moisture level, temperature, and yield estimate. Farmers need to:

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

Student Task

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

```

ai_coding.py > ...
1  # Task Description #7: Smart Agriculture Crop Monitoring System
2  # An agriculture monitoring system stores crop data with crop ID, crop
3  # name, soil moisture level, temperature, and yield estimate. Farmers
4  # need to:
5  # 1. Search crop details using crop ID.
6  # 2. Sort crops based on moisture level or yield estimate.
7  # Student Task
8  # • Use AI-assisted reasoning to select algorithms.
9  # • Justify algorithm suitability.
10 # • Implement searching and sorting in Python.
11 # AI-assisted reasoning to select algorithms:
12 # For searching crop details using crop ID, a hash table (dictionary in Python) is suitable
13 # because it provides O(1) average time complexity for lookups, making it efficient for retrieving crop details based on unique identifiers.
14 # For sorting crops based on moisture level or yield estimate, the Timsort algorithm (used by Python's built-in sorted() function) is appropriate. T
15 # Justification of algorithm suitability:
16
17 # The hash table allows for fast retrieval of crop details using crop ID, which is essential for farmers who need quick access to specific crop info
18 # Implementation of searching and sorting in Python:
19 class Crop:
20     def __init__(self, crop_id, name, moisture_level, temperature, yield_estimate):
21         self.crop_id = crop_id
22         self.name = name
23         self.moisture_level = moisture_level
24         self.temperature = temperature
25         self.yield_estimate = yield_estimate
26
27     def __repr__(self):
28         return f"Crop(ID: {self.crop_id}, Name: {self.name}, Moisture: {self.moisture_level}, Temp: {self.temperature}, Yield: {self.yield_estimate})"
29
30 class CropMonitoringSystem:
31     def __init__(self):
32         self.crops = {}
33
34     def add_crop(self, crop):
35         self.crops[crop.crop_id] = crop
36
37     def search_crop_by_id(self, crop_id):

```

```

ai_coding.py > ...
19  class Crop:
20      def __init__(self, crop_id, name, moisture_level, temperature, yield_estimate):
21          self.moisture_level = moisture_level
22          self.temperature = temperature
23          self.yield_estimate = yield_estimate
24
25      def __repr__(self):
26          return f"Crop(ID: {self.crop_id}, Name: {self.name}, Moisture: {self.moisture_level}, Temp: {self.temperature}, Yield: {self.yield_estimate})"
27
28  class CropMonitoringSystem:
29      def __init__(self):
30          self.crops = {}
31
32      def add_crop(self, crop):
33          self.crops[crop.crop_id] = crop
34
35      def search_crop_by_id(self, crop_id):
36          return self.crops.get(crop_id, "Crop not found")
37
38      def sort_crops_by_moisture(self):
39          return sorted(self.crops.values(), key=lambda x: x.moisture_level)
40
41      def sort_crops_by_yield(self):
42          return sorted(self.crops.values(), key=lambda x: x.yield_estimate)
43
44
45      # Example usage
46      if __name__ == "__main__":
47          system = CropMonitoringSystem()
48          system.add_crop(Crop(1, "wheat", 30, 25, 100))
49          system.add_crop(Crop(2, "corn", 40, 22, 150))
50          system.add_crop(Crop(3, "rice", 20, 28, 120))
51
52          print(system.search_crop_by_id(2)) # Search for crop with ID 2
53          print(system.sort_crops_by_moisture()) # Sort crops by moisture level
54          print(system.sort_crops_by_yield()) # Sort crops by yield estimate
55
56

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

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

```

Task Description #8: Airport Flight Management System

An airport system stores flight information including flight ID, airline name, departure time, arrival time, and status. The system must:

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

Student Task

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

```
ai_coding.py > ...  
1  # Task Description #8: Airport Flight Management System  
2  # An airport system stores flight information including flight ID,  
3  # airline name, departure time, arrival time, and status. The system  
4  # must:  
5  # 1. Search flight details using flight ID.  
6  # 2. Sort flights based on departure time or arrival time.  
7  # Student Task  
8  # • Use AI to recommend algorithms.  
9  # • Justify the algorithm selection.  
10 # • Implement searching and sorting logic in Python.  
11  
12 # AI Recommendation:  
13 # For searching flight details using flight ID, a hash table (dictionary in Python) is recommended for O(1) average time complexity. This  
14 # For sorting flights based on departure time or arrival time, the Timsort algorithm (used by Python's built-in sorted() function) is re  
15  
16 class Flight:  
17     def __init__(self, flight_id, airline_name, departure_time, arrival_time, status):  
18         self.flight_id = flight_id  
19         self.airline_name = airline_name  
20         self.departure_time = departure_time  
21         self.arrival_time = arrival_time  
22         self.status = status  
23  
24 class AirportFlightManagementsystem:  
25     def __init__(self):  
26         self.flights = {}  
27  
28     def add_flight(self, flight):  
29         self.flights[flight.flight_id] = flight  
30  
31     def search_flight(self, flight_id):  
32         return self.flights.get(flight_id, None)  
33  
34     def sort_flights_by_departure_time(self):  
35         return sorted(self.flights.values(), key=lambda x: x.departure_time)  
36  
37     def sort_flights_by_arrival_time(self):
```

Ln 67, Col 9 Spaces: 4 UTF-8 CRLF { } Python Py

```

❸ ai.coding.py > ...
24     class AirportFlightManagementSystem:
25
26         def sort_flights_by_arrival_time(self):
27             |     return sorted(self.flights.values(), key=lambda x: x.arrival_time)
28
29     # Example Usage
30     if __name__ == "__main__":
31         system = AirportFlightManagementSystem()
32
33     flight1 = Flight("AA101", "American Airlines", "08:00", "10:00", "On Time")
34     flight2 = Flight("DL202", "Delta Airlines", "09:00", "11:00", "Delayed")
35     flight3 = Flight("UA303", "United Airlines", "07:30", "09:30", "On Time")
36
37     system.add_flight(flight1)
38     system.add_flight(flight2)
39     system.add_flight(flight3)
40
41     # Search for a flight
42     flight = system.search_flight("DL202")
43     if flight:
44         |     print(f"Flight ID: {flight.flight_id}, Airline: {flight.airline_name}, Status: {flight.status}")
45
46     # Sort flights by departure time
47     sorted_by_departure = system.sort_flights_by_departure_time()
48     print("Flights sorted by departure time:")
49     for f in sorted_by_departure:
50         |     print(f"{f.flight_id} - {f.departure_time}")
51
52     # Sort flights by arrival time
53     sorted_by_arrival = system.sort_flights_by_arrival_time()
54     print("Flights sorted by arrival time:")
55     for f in sorted_by_arrival:
56         |     print(f"{f.flight_id} - {f.arrival_time}")
57
58
59
60
61
62
63
64
65
66
67

```

PROBLEMS 10 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

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

```