

AI ASSISTED CODING

LAB-12.5

Katta Lasya

2303A51733

Batch-11

Task Description #1 (Sorting – Merge Sort Implementation)

- Task: Use AI to generate a Python program that implements the Merge Sort algorithm.
- Instructions:
 - Prompt AI to create a function `merge_sort(arr)` that sorts a list in ascending order.
 - Ask AI to include time complexity and space complexity in the function docstring.
 - Verify the generated code with test cases.
- Expected Output:
 - A functional Python script implementing Merge Sort with proper documentation.

PROMPT:

```
#Generate code for merge_sort(arr) function to sort an array in ascending order, include  
time and space complexity in docstring.
```

GIVEN CODE AND OUTPUT:

```

1  #Generate code for merge_sort(arr) function to sort an array in ascending order, include time and space complexity in docstring.
2  def merge(left, right):
3      """Helper function to merge two sorted arrays."""
4      result = []
5      i = j = 0
6      while i < len(left) and j < len(right):
7          if left[i] <= right[j]:
8              result.append(left[i])
9              i += 1
10         else:
11             result.append(right[j])
12             j += 1
13     result.extend(left[i:])
14     result.extend(right[j:])
15
16
17 def merge_sort(arr):
18     """Sort an array in ascending order using the merge sort algorithm.
19
20     Merge sort is a divide-and-conquer algorithm that recursively divides the array into halves, sorts each half, and then merges the sorted halves back together.
21     The time complexity of merge sort is O(n log n) in all cases (best, average, and worst), making it an efficient sorting algorithm for large datasets.
22     The space complexity is O(n) due to the temporary arrays used during the merging process.
23
24     Args:
25         arr (list): The list of elements to be sorted."""
26     if len(arr) <= 1:
27         return arr
28
29     mid = len(arr) // 2
30     left_half = merge_sort(arr[:mid])
31     right_half = merge_sort(arr[mid:])
32
33     return merge(left_half, right_half)
34
#Example usage:
35 arr = list(map(int, input("Enter the array elements separated by space: ").split()))
36 sorted_arr = merge_sort(arr)
37 print("Sorted array:", sorted_arr)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/Desktop/AIAC/lab12.py
Enter the array elements separated by space: 67 43 01 54 93
Sorted array: [1, 43, 54, 67, 93]
PS C:\Users\katta\OneDrive\Desktop\AIAC>

```

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:**
 - Prompt AI to create a function **binary_search(arr, target)** returning the index of the target or -1 if not found.
 - Include docstrings explaining best, average, and worst-case complexities.
 - Test with various inputs.
- **Expected Output:**
 - Python code implementing binary search with AI-generated comments and docstrings.

PROMPT:

#Generate code for **binary_search(arr, target)** function to return the index of the target element and -1 if not found, include best, average, and worst-case complexities in docstring.

GIVEN CODE AND OUTPUT:

```

1  #Generate code for binary_search(arr, target) function to return the index of the target element and -1 if not found, :
2  def binary_search(arr, target):
3      """Perform binary search on a sorted array to find the index of the target element.
4
5      Args:
6          arr (list): A sorted list of elements.
7          target: The element to search for in the array.
8
9      Returns:
10         int: The index of the target element if found, otherwise -1.
11         Complexity:
12             Best Case: O(1) - when the target is at the middle of the array.
13             Average Case: O(log n) - when the target is found after several iterations.
14             Worst Case: O(log n) - when the target is not found or is at the end of the array.
15
16     """
17     left, right = 0, len(arr) - 1
18
19     while left <= right:
20         mid = left + (right - left) // 2
21
22         # Check if the target is present at mid
23         if arr[mid] == target:
24             return mid
25         # If target is greater, ignore the left half
26         elif arr[mid] < target:
27             left = mid + 1
28         # If target is smaller, ignore the right half
29         else:
30             right = mid - 1
31
32     # Target was not found in the array
33     return -1
34
35     # Example usage:
36     sorted_array = [1, 2, 3, 4, 5, 6, 7, 8, 9]
37     target = 5
38     result = binary_search(sorted_array, target)
39     print(f"Target {target} found at index: {result}")
40     target = 10
41     result = binary_search(sorted_array, target)
42     print(f"Target {target} found at index: {result}") # Output: -1

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\katta\OneDrive\Desktop\AIAC> & C:/Users/katta/anaconda3/python.exe c:/Users/katta/OneDrive/Desktop/AIAC/lab12.py

Target 5 found at index: 4
Target 10 found at index: -1

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.

PROMPT:

""""Generate python code for a healthcare management system that maintains appointment records containing appointment ID, patient name, doctor name, appointment time, and

consultation fee. Give code to search appointments using appointment ID, and sort based on time or consultation fee by using suitable searching and sorting algorithms."""

GIVEN CODE:

```
"""Generate python code for a healthcare management system that maintains appointment records containing appointment ID, patient name, doctor name, appointment time, and consultation fee. Give code to search appointments using appointment ID, and sort based on time or consultation fee by using suitable searching and sorting algorithms."""
class Appointment:
    """A class to represent an appointment in a healthcare management system."""

    def __init__(self, appointment_id, patient_name, doctor_name, appointment_time, consultation_fee):
        self.appointment_id = appointment_id
        self.patient_name = patient_name
        self.doctor_name = doctor_name
        self.appointment_time = appointment_time
        self.consultation_fee = consultation_fee

class AppointmentManager:
    """A class to manage appointment records in a healthcare management system."""

    def __init__(self):
        self.appointments = []

    def add_appointment(self, appointment):
        """Add an appointment to the manager."""
        self.appointments.append(appointment)

    def search_appointment_by_id(self, appointment_id):
        """Search for an appointment by its ID using linear search."""
        for appointment in self.appointments:
            if appointment.appointment_id == appointment_id:
                return appointment
        return None

    def sort_appointments_by_time(self):
        """Sort appointments based on appointment time using the built-in sorted function."""
        self.appointments.sort(key=lambda x: x.appointment_time)

    def sort_appointments_by_fee(self):
        """Sort appointments based on consultation fee using the built-in sorted function."""
        self.appointments.sort(key=lambda x: x.consultation_fee)
```

```
# Example usage:
appointment_manager = AppointmentManager()
appointment_manager.add_appointment(Appointment(1, "Alice", "Dr. Smith", "2024-07-01 10:00", 100))
appointment_manager.add_appointment(Appointment(2, "Bob", "Dr. Jones", "2024-07-01 11:00", 150))
appointment_manager.add_appointment(Appointment(3, "Charlie", "Dr. Brown", "2024-07-01 09:00", 120))
# Search for an appointment by ID
appointment = appointment_manager.search_appointment_by_id(2)
if appointment:
    print(f"Appointment found: {appointment.patient_name} with {appointment.doctor_name} at {appointment.appointment_time} for ${appointment.consultation_fee}")
else:
    print("Appointment not found.")
# Sort appointments by time
appointment_manager.sort_appointments_by_time()
print("Appointments sorted by time:")
for appointment in appointment_manager.appointments:
    print(f"{appointment.patient_name} with {appointment.doctor_name} at {appointment.appointment_time} for ${appointment.consultation_fee}")
# Sort appointments by consultation fee
appointment_manager.sort_appointments_by_fee()
print("Appointments sorted by consultation fee:")
for appointment in appointment_manager.appointments:
    print(f"{appointment.patient_name} with {appointment.doctor_name} at {appointment.appointment_time} for ${appointment.consultation_fee}")
```

OUTPUT:

```
PS C:\Users\katta\OneDrive\Desktop\AIAC> & c:/Users/katta/anaconda3/p
Appointment found: Bob with Dr. Jones at 2024-07-01 11:00 for $150
Appointments sorted by time:
Charlie with Dr. Brown at 2024-07-01 09:00 for $120
Alice with Dr. Smith at 2024-07-01 10:00 for $100
Bob with Dr. Jones at 2024-07-01 11:00 for $150
Appointments sorted by consultation fee:
Alice with Dr. Smith at 2024-07-01 10:00 for $100
Charlie with Dr. Brown at 2024-07-01 09:00 for $120
Bob with Dr. Jones at 2024-07-01 11:00 for $150
```

Task Description #4: Railway Ticket Reservation System Scenario

A railway reservation system stores booking details such as ticketID, 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.

PROMPT:

#write a python program to create a railway reservation system stores booking details such as passenger name,ticket id,train number,seat number, and travel date.you have to search the tickets using ticket id and sort bookings based on travel date or set number. identify the best algorithms for searching and sortingand implement them in your program.

GIVEN CODE AND OUTPUT:

```
1 #write a python program to create a railway reservation system stores booking details such as passenger name,ticket id,train n
2 class RailwayReservationSystem:
3     """A simple railway reservation system to manage booking details."""
4     def __init__(self):
5         """Initialize the reservation system with an empty list of bookings."""
6         self.bookings = []
7     def book_ticket(self, passenger_name, ticket_id, train_number, seat_number, travel_date):
8         """Book a ticket and store the booking details."""
9         booking = {
10             'passenger_name': passenger_name,
11             'ticket_id': ticket_id,
12             'train_number': train_number,
13             'seat_number': seat_number,
14             'travel_date': travel_date
15         }
16         self.bookings.append(booking)
17     def search_ticket(self, ticket_id):
18         """Search for a ticket using its ticket ID."""
19         for booking in self.bookings:
20             if booking['ticket_id'] == ticket_id:
21                 return booking
22         return None
23     def sort_bookings_by_travel_date(self):
24         """Sort bookings based on travel date using the built-in sorted function."""
25         self.bookings.sort(key=lambda x: x['travel_date'])
26     def sort_bookings_by_seat_number(self):
27         """Sort bookings based on seat number using the built-in sorted function."""
28         self.bookings.sort(key=lambda x: x['seat_number'])
29     def display_bookings(self):
30         """Display all bookings in the reservation system."""
31         for booking in self.bookings:
32             print(booking)
33
34 # Example usage
35 reservation_system = RailwayReservationSystem()
36 reservation_system.book_ticket("Alice", "T123", "Train1", "A1", "2024-07-01")
37 reservation_system.book_ticket("Bob", "T124", "Train2", "B1", "2024-07-02")
38 reservation_system.book_ticket("Charlie", "T125", "Train1", "A2", "2024-07-01")
39 print("All Bookings:")
40 reservation_system.display_bookings()
41 print("\nSearch for Ticket ID T124:")
42 ticket = reservation_system.search_ticket("T124")
43 if ticket:
44     print(ticket)
45 else:
46     print("Ticket not found.")
47 print("\nBookings sorted by Travel Date:")
48 reservation_system.sort_bookings_by_travel_date()
49 reservation_system.display_bookings()
50 print("\nBookings sorted by Seat Number:")
51 reservation_system.sort_bookings_by_seat_number()
52 reservation_system.display_bookings()
```

The screenshot shows a terminal window with the following content:

```
34 reservation_system = RailwayReservationSystem()
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> c:; cd 'c:\Users\thota\OneDrive\Desktop\AIAC'; & 'C:\Users\thota\anaconda3\python.exe' 'c:\Users\thota\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61625' '--' 'c:\Users\thota\OneDrive\Desktop\AIAC\lab5.py'
{'passenger_name': 'Alice', 'ticket_id': 'T123', 'train_number': 'Train1', 'seat_number': 'A1', 'travel_date': '2024-07-01'}
{'passenger_name': 'Charlie', 'ticket_id': 'T125', 'train_number': 'Train1', 'seat_number': 'A2', 'travel_date': '2024-07-01'}
○ Search for Ticket ID T124:
{'passenger_name': 'Bob', 'ticket_id': 'T124', 'train_number': 'Train2', 'seat_number': 'B1', 'travel_date': '2024-07-02'}

Bookings sorted by Travel Date:
{'passenger_name': 'Alice', 'ticket_id': 'T123', 'train_number': 'Train1', 'seat_number': 'A1', 'travel_date': '2024-07-01'}
{'passenger_name': 'charlie', 'ticket_id': 'T125', 'train_number': 'Train1', 'seat_number': 'A2', 'travel_date': '2024-07-01'}
{'passenger_name': 'Bob', 'ticket_id': 'T124', 'train_number': 'Train2', 'seat_number': 'B1', 'travel_date': '2024-07-02'}

Bookings sorted by Seat Number:
{'passenger_name': 'Alice', 'ticket_id': 'T123', 'train_number': 'Train1', 'seat_number': 'A1', 'travel_date': '2024-07-01'}
{'passenger_name': 'Charlie', 'ticket_id': 'T125', 'train_number': 'Train1', 'seat_number': 'A2', 'travel_date': '2024-07-01'}
{'passenger_name': 'Bob', 'ticket_id': 'T124', 'train_number': 'Train2', 'seat_number': 'B1', 'travel_date': '2024-07-02'}
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

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.

PROMPT:

#write a python program to create a hostel management system it stores student room allocation details including student id,room number,floor,and allocation date. you have to search allocation details using student id and sort records based on room number or allocation date.you have to sugest optimized algorhms for searching and sorting the records

CODE AND OUTPUT:

```
❷ lab5.py > ...
1  #write a python program to create a hostel management system it stores student room allocation details in
2  class HostelManagementSystem:
3      """A simple hostel management system to store student room allocation details."""
4      def __init__(self):
5          """Initialize the hostel management system."""
6          self.allocations = []
7      def add_allocation(self, student_id, room_number, floor, allocation_date):
8          """Add a new room allocation record."""
9          self.allocations.append({
10              'student_id': student_id,
11              'room_number': room_number,
12              'floor': floor,
13              'allocation_date': allocation_date
14          })
15      def search_by_student_id(self, student_id):
16          """Search for allocation details using student id."""
17          for allocation in self.allocations:
18              if allocation['student_id'] == student_id:
19                  return allocation
20          return None
21      def sort_by_room_number(self):
22          """Sort records based on room number using an optimized sorting algorithm (Timsort)."""
23          self.allocations.sort(key=lambda x: x['room_number'])
24      def sort_by_allocation_date(self):
25          """Sort records based on allocation date using an optimized sorting algorithm (Timsort)."""
26          self.allocations.sort(key=lambda x: x['allocation_date'])
27      def display_allocations(self):
28          """Display all room allocation records."""
29          for allocation in self.allocations:
30              print(allocation)
31  # Example usage
32  hostel_system = HostelManagementSystem()
```

```

31 # Example usage
32 hostel_system = HostelManagementSystem()
33 hostel_system.add_allocation('S001', '101', '1st Floor', '2024-01-15')
34 hostel_system.add_allocation('S002', '102', '1st Floor', '2024-01-16')
35 hostel_system.add_allocation('S003', '201', '2nd Floor', '2024-01-17')
36 print("Search for student S002:")
37 print(hostel_system.search_by_student_id('S002'))
38 print("\nAllocations sorted by room number:")
39 hostel_system.sort_by_room_number()
40 hostel_system.display_allocations()
41 print("\nAllocations sorted by allocation date:")
42 hostel_system.sort_by_allocation_date()
43 hostel_system.display_allocations()
44
45

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> c::; cd 'c:\Users\thota\OneDrive\Desktop\AIAC'; & 'C:\Users\thota\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51977' '--' 'c:\AIAC\lab5.py'
{'student_id': 'S002', 'room_number': '102', 'floor': '1st Floor', 'allocation_date': '2024-01-16'}

Allocations sorted by room number:
{'student_id': 'S001', 'room_number': '101', 'floor': '1st Floor', 'allocation_date': '2024-01-15'}
{'student_id': 'S002', 'room_number': '102', 'floor': '1st Floor', 'allocation_date': '2024-01-16'}
{'student_id': 'S003', 'room_number': '201', 'floor': '2nd Floor', 'allocation_date': '2024-01-17'}

Allocations sorted by allocation date:
{'student_id': 'S001', 'room_number': '101', 'floor': '1st Floor', 'allocation_date': '2024-01-15'}
{'student_id': 'S002', 'room_number': '102', 'floor': '1st Floor', 'allocation_date': '2024-01-16'}
{'student_id': 'S003', 'room_number': '201', 'floor': '2nd Floor', 'allocation_date': '2024-01-17'}
○ (base) PS C:\Users\thota\OneDrive\Desktop\AIAC>

Allocations sorted by allocation date:
{'student_id': 'S001', 'room_number': '101', 'floor': '1st Floor', 'allocation_date': '2024-01-15'}
{'student_id': 'S002', 'room_number': '102', 'floor': '1st Floor', 'allocation_date': '2024-01-16'}
{'student_id': 'S003', 'room_number': '201', 'floor': '2nd Floor', 'allocation_date': '2024-01-17'}
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>

```

In 45 Col 1 Spaces: 4

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.

PROMPT:

#write a python program to create a streaming service maintains movie records with movie id,title,genre,rating, and release year. you have to search movies by movie id and sort movies based on rating or release year. you have to recommend searching and sorting algorithms for

this application and justify your choices with comments and docstrings

CODE AND OUTPUT:

```
lab5.py > ⌂ StreamingService > ⌂ search_by_id
1  #write a python program to create a streaming service maintains movie records with movie id,title,genre,rating, and release year
2  class Movie:
3      """A class to represent a movie record."""
4      def __init__(self, movie_id, title, genre, rating, release_year):
5          """Initialize the movie record with its attributes."""
6          self.movie_id = movie_id
7          self.title = title
8          self.genre = genre
9          self.rating = rating
10         self.release_year = release_year
11     def __str__():
12         """Return a string representation of the movie record."""
13         return f"{self.movie_id}: {self.title} ({self.genre}, {self.rating}, {self.release_year})"
14 class StreamingService:
15     """A class to represent a streaming service that maintains movie records."""
16     def __init__(self):
17         """Initialize the streaming service with an empty list of movies."""
18         self.movies = []
19     def add_movie(self, movie):
20         """Add a movie record to the streaming service."""
21         self.movies.append(movie)
22     def search_by_id(self, movie_id):
23         """Search for a movie by its ID using linear search."""
24         for movie in self.movies:
25             if movie.movie_id == movie_id:
26                 return movie
27         return None
lab5.py > ⌂ StreamingService > ⌂ search_by_id
4  class StreamingService:
5      def search_by_id(self, movie_id):
6          return None
7      def sort_by_rating(self):
8          """Sort movies based on rating using the built-in sorted function."""
9          return sorted(self.movies, key=lambda x: x.rating, reverse=True)
10     def sort_by_release_year(self):
11         """Sort movies based on release year using the built-in sorted function."""
12         return sorted(self.movies, key=lambda x: x.release_year)
13 # I have chosen to implement linear search for searching movies by movie ID because it is a simple and straightforward algorithm
14 # For sorting movies based on rating and release year, I have chosen to use the built-in sorted function in Python, which implements
15 # Example usage:
16 if __name__ == "__main__":
17     service = StreamingService()
18     service.add_movie(Movie(1, "Inception", "Sci-Fi", 8.8, 2010))
19     service.add_movie(Movie(2, "The Matrix", "Action", 8.7, 1999))
20     service.add_movie(Movie(3, "Interstellar", "Sci-Fi", 8.6, 2014))
21     print("Search for movie with ID 2:")
22     print(service.search_by_id(2))
23     print("\nMovies sorted by rating:")
24     for movie in service.sort_by_rating():
25         print(movie)
26     print("\nMovies sorted by release year:")
27     for movie in service.sort_by_release_year():
28         print(movie)
```

```
C:\Users\thota\OneDrive\Desktop\AIAC> python C:\Users\thota\OneDrive\Desktop\AIAC\lab5.py

Movies sorted by rating:
Search for movie with ID 2:
2: The Matrix (Action, 8.7, 1999)

Movies sorted by rating:
1: Inception (Sci-Fi, 8.8, 2010)
2: The Matrix (Action, 8.7, 1999)
2: The Matrix (Action, 8.7, 1999)
3: Interstellar (Sci-Fi, 8.6, 2014)

Movies sorted by release year:
2: The Matrix (Action, 8.7, 1999)
2: The Matrix (Action, 8.7, 1999)
3: Interstellar (Sci-Fi, 8.6, 2014)

2: The Matrix (Action, 8.7, 1999)
3: Interstellar (Sci-Fi, 8.6, 2014)
2: The Matrix (Action, 8.7, 1999)
3: Interstellar (Sci-Fi, 8.6, 2014)

Movies sorted by release year:
2: The Matrix (Action, 8.7, 1999)
1: Inception (Sci-Fi, 8.8, 2010)
3: Interstellar (Sci-Fi, 8.6, 2014)
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

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.

PROMPT:

```
#write a python program to create an agriculture monitoring system it stores crop data with crop id,crop name,soil moisture level,temperature,and yeild estimate. you have to search crop details using crop id and sort crops based on moisture level or yield estimate .you have to select reasoning algorithm and jusity alorithm and implement searching and sorting
```

CODE AND OUTPUT:

```

1  #write a python program to create an agriculture monitoring system it stores crop data with crop id,crop name,soil moisture level,t
2  class Crop:
3      """A class to represent a crop in the agriculture monitoring system."""
4      def __init__(self, crop_id, crop_name, soil_moisture, temperature, yield_estimate):
5          """Initialize the crop with its details."""
6          self.crop_id = crop_id
7          self.crop_name = crop_name
8          self.soil_moisture = soil_moisture
9          self.temperature = temperature
10         self.yield_estimate = yield_estimate
11 class AgricultureMonitoringSystem:
12     """A class to represent the agriculture monitoring system."""
13     def __init__(self):
14         """Initialize the monitoring system with an empty list of crops."""
15         self.crops = []
16     def add_crop(self, crop):
17         """Add a crop to the monitoring system."""
18         self.crops.append(crop)
19     def search_crop_by_id(self, crop_id):
20         """Search for a crop by its ID using linear search algorithm."""
21         for crop in self.crops:
22             if crop.crop_id == crop_id:
23                 return crop
24         return None
25     def sort_crops_by_moisture(self):
26         """Sort crops based on soil moisture level using bubble sort algorithm."""
27         n = len(self.crops)
28         for i in range(n):
29             for j in range(0, n-i-1):
30                 if self.crops[j].soil_moisture > self.crops[j+1].soil_moisture:
31                     self.crops[j], self.crops[j+1] = self.crops[j+1], self.crops[j]
32     def sort_crops_by_yield(self):
33         """Sort crops based on yield estimate using bubble sort algorithm."""
34         n = len(self.crops)
35         for i in range(n):
36             for j in range(0, n-i-1):
37                 if self.crops[j].yield_estimate > self.crops[j+1].yield_estimate:
38                     self.crops[j], self.crops[j+1] = self.crops[j+1], self.crops[j]

```

In 7 Col 35 Spaces: 4 LITE-B CRLF { } Python base (3.12)

Help lab2 lab3.py Launch

lab5.py > Crop > __init__

```

11  class AgricultureMonitoringSystem:
12      def __init__(self):
13          self.crops = []
14      def add_crop(self, crop):
15          self.crops.append(crop)
16      def search_crop_by_id(self, crop_id):
17          for crop in self.crops:
18              if crop.crop_id == crop_id:
19                  return crop
20          return None
21      def sort_crops_by_moisture(self):
22          n = len(self.crops)
23          for i in range(n):
24              for j in range(0, n-i-1):
25                  if self.crops[j].soil_moisture > self.crops[j+1].soil_moisture:
26                      self.crops[j], self.crops[j+1] = self.crops[j+1], self.crops[j]
27      def sort_crops_by_yield(self):
28          n = len(self.crops)
29          for i in range(n):
30              for j in range(0, n-i-1):
31                  if self.crops[j].yield_estimate > self.crops[j+1].yield_estimate:
32                      self.crops[j], self.crops[j+1] = self.crops[j+1], self.crops[j]
33
# Example usage
34 if __name__ == "__main__":
35     system = AgricultureMonitoringSystem()
36     system.add_crop(Crop(1, "Wheat", 30, 25, 100))
37     system.add_crop(Crop(2, "Corn", 40, 28, 150))
38     system.add_crop(Crop(3, "Rice", 20, 22, 120))
39
40     print("search for crop with ID 2:")
41     crop = system.search_crop_by_id(2)
42     if crop:
43         print(f"Crop found: {crop.crop_name}, Soil Moisture: {crop.soil_moisture}, Temperature: {crop.temperature}, Yield Estimate: {crop.yield_estimate}")
44     else:
45         print("Crop not found.")
46
47     print("\nCrops sorted by soil moisture level:")
48     system.sort_crops_by_moisture()
49     for crop in system.crops:
50         print(f"{crop.crop_name}: Soil Moisture: {crop.soil_moisture}")
51
52     print("\nCrops sorted by yield estimate:")
53     system.sort_crops_by_yield()
54     for crop in system.crops:
55         print(f"{crop.crop_name}: Yield Estimate: {crop.yield_estimate}")
56
57
58
59
60
61
62
63
64
65

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> c:; cd 'c:\Users\thota\OneDrive\Desktop\AIAC'; & 'C:\Users\thota\anaconda\thota\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60072' '--' 'c:\Users\thota\OneDrive\Desktop\AIAC\lab5.py'
● (base) PS C:\Users\thota\OneDrive\Desktop\AIAC> c:; cd 'c:\Users\thota\OneDrive\Desktop\AIAC'; & 'C:\Users\thota\anaconda\thota\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57787' '--' 'c:\Users\thota\OneDrive\Desktop\AIAC\lab5.py'
Search for crop with ID 2:
Crop found: Corn, Soil Moisture: 40, Temperature: 28, Yield Estimate: 150

Crops sorted by soil moisture level:
Rice: Soil Moisture: 20
Wheat: Soil Moisture: 30
Corn: Soil Moisture: 40
Corn: Soil Moisture: 40

Crops sorted by yield estimate:
Wheat: Yield Estimate: 100
Rice: Yield Estimate: 120
○ Corn: Yield Estimate: 150
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC>

Corn: Soil Moisture: 40

Crops sorted by yield estimate:
Wheat: Yield Estimate: 100
Rice: Yield Estimate: 120
Corn: Soil Moisture: 40

Crops sorted by yield estimate:
Corn: Soil Moisture: 40

Crops sorted by yield estimate:
Wheat: Yield Estimate: 100
Rice: Yield Estimate: 120
```

In 7 Col 25 Spans 4 LITE 9

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

PROMPT:

#write a python program to create an airport system store flight information including flight id,airline name,departure time,arrival time,and status. you have to search flight details using flight id and sort based on departure time or arrival time..you have to recommend algorithms and justify the algorithm and implement searching and sorting algorithms with comments

CODE AND OUTPUT:

```

ab5.py > ⚙ Flight > ⚡ _init_
    #write a python program to create an airport system store flight information including flight id,airline name,departure time,arrival time, status
    class Flight:
        """A class to represent a flight in the airport system."""
        def __init__(self, flight_id, airline_name, departure_time, arrival_time, status):
            """Initialize the flight with its details."""
            self.flight_id = flight_id
            self.airline_name = airline_name
            self.departure_time = departure_time
            self.arrival_time = arrival_time
            self.status = status
    class AirportSystem:
        """A class to represent the airport system that manages flight information."""
        def __init__(self):
            """Initialize the airport system with an empty list of flights."""
            self.flights = []
        def add_flight(self, flight):
            """Add a flight to the airport system."""
            self.flights.append(flight)
        def search_flight(self, flight_id):
            """Search for a flight by its flight ID using linear search algorithm."""
            for flight in self.flights:
                if flight.flight_id == flight_id:
                    return flight
            return None
        def sort_flights_by_departure_time(self):
            """Sort flights based on departure time using the built-in sorted function."""
            self.flights.sort(key=lambda x: x.departure_time)
        def sort_flights_by_arrival_time(self):
            """Sort flights based on arrival time using the built-in sorted function."""
            self.flights.sort(key=lambda x: x.arrival_time)
    # Justification for algorithms:
lab2   lab3.py  || ? ↴ ↑ ⏪ Launch
lab5.py > ⚙ Flight > ⚡ _init_
    self.flights.sort(key=lambda x: x.arrival_time)
    # Justification for algorithms:
    # 1. Linear Search: I have chosen the linear search algorithm for searching flight details by flight ID because it is simple and efficient for small datasets.
    # 2. Built-in Sort: I have chosen to use the built-in sorted function for sorting flights based on departure and arrival times.
    # Example usage:
    if __name__ == "__main__":
        airport_system = AirportSystem()
        flight1 = Flight("AA101", "American Airlines", "08:00", "10:00", "On Time")
        flight2 = Flight("DL202", "Delta Airlines", "09:00", "11:00", "Delayed")
        flight3 = Flight("UA303", "United Airlines", "07:30", "09:30", "On Time")
        airport_system.add_flight(flight1)
        airport_system.add_flight(flight2)
        airport_system.add_flight(flight3)

        # Search for a flight
        search_result = airport_system.search_flight("DL202")
        if search_result:
            print(f"Flight found: {search_result.airline_name} departing at {search_result.departure_time}")
        else:
            print("Flight not found.")

        # Sort flights by departure time
        airport_system.sort_flights_by_departure_time()
        print("\nFlights sorted by departure time:")
        for flight in airport_system.flights:
            print(f"{flight.flight_id}: {flight.departure_time}")

        # Sort flights by arrival time
        airport_system.sort_flights_by_arrival_time()
        print("\nFlights sorted by arrival time:")

```

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files `lab2`, `lab3.py`, and `lab5.py`. `lab5.py` is the active file.
- Code Editor:** Displays the contents of `lab5.py`. The code sorts flights by departure and arrival times and prints them. Lines 51-57 show sorting by departure time, and lines 58-64 show sorting by arrival time.
- Terminal:** Shows the command `python lab5.py` being run. The output includes:
 - Flight found: Delta Airlines departing at 09:00
 - Flights sorted by departure time:
 - UA303: 07:30
 - AA101: 08:00
 - DL202: 09:00
 - Flights sorted by arrival time:
 - UA303: 07:30
 - AA101: 08:00
 - DL202: 09:00
 - AA101: 08:00
 - DL202: 09:00
 - AA101: 08:00
 - DL202: 09:00
 - DL202: 09:00

```
ta\OneDrive\Desktop\AIAC\Lab5.py'
```

```
Flight found: Delta Airlines departing at 09:00
```

```
Flights sorted by departure time:
```

```
○ UA303: 07:30
```

```
AA101: 08:00
```

```
DL202: 09:00
```

```
Flights sorted by departure time:
```

```
UA303: 07:30
```

```
AA101: 08:00
```

```
DL202: 09:00
```

```
UA303: 07:30
```

```
AA101: 08:00
```

```
DL202: 09:00
```

```
AA101: 08:00
```

```
DL202: 09:00
```

```
DL202: 09:00
```

```
Flights sorted by arrival time:
```

```
UA303: 09:30
```

```
AA101: 10:00
```

```
UA303: 09:30
```

```
AA101: 10:00
```

```
DL202: 11:00
```

```
(base) PS C:\Users\thota\OneDrive\Desktop\AIAC> []
```