Assignment -12.5(AI FOR CODING)

2303A51729

Batch-11

Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

Lab Objectives:`

• Apply AI-assisted programming to implement and optimize sorting and searching algorithms.

• Compare different algorithms in terms of efficiency and use

cases.

• Understand how AI tools can suggest optimized code and

complexity improvements.

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.

• Expected Output:

o A functional Python script implementing Merge Sort

with proper documentation.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input and then sor
def merge_sort(arr):
    """
    Sorts a list in ascending order using the Merge Sort algorithm.

    Time Complexity: O(n log n) - where n is the number of elements in the array.
    Space Complexity: O(n) - due to the temporary arrays used during merging.

    Parameters:
    arr (list): The list of elements to be sorted.

    Returns:
    list: A new list containing the sorted elements.
    """
    if len(arr) <= 1:
        return arr

    # Split the array into halves
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    # Merge the sorted halves
    return merge(left_half, right_half)
def merge(left, right):
    """Merges two sorted lists into a single sorted list.
    Parameters:
    left (list): The first sorted list.
    right (list): The second sorted list.
    Returns:
    list: A merged and sorted list containing all elements from left and right.
    """
    merged = []
    left_index = right_index = 0
```

```python
    merged = []
    left_index = right_index = 0

    # Merge the two lists while maintaining sorted order
    while left_index < len(left) and right_index < len(right):
        if left[left_index] < right[right_index]:
            merged.append(left[left_index])
            left_index += 1
        else:
            merged.append(right[right_index])
            right_index += 1

    # If there are remaining elements in left or right, add them to merged
    merged.extend(left[left_index:])
    merged.extend(right[right_index:])

    return merged
# Get the list of numbers from user input
user_input = input("Enter numbers to sort (separated by spaces): ")
# Convert the input string into a list of integers
arr = list(map(int, user_input.split()))
# Sort the array using merge_sort function
sorted_arr = merge_sort(arr)
# Print the sorted array
print("Sorted array:", sorted_arr)
```

Output:

```
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:\Program Files\Python39\python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab-12.5.py"
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:\Program Files\Python39\python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab-12.5.py"
Enter numbers to sort (separated by spaces): 8 5 6 4 7 9 1 2
Sorted array: [1, 2, 4, 5, 6, 7, 8, 9]
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
```

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.

• Expected Output:

o Python code implementing binary search with AI-

generated comments and docstrings.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input
def binary_search(arr, target):
    """
    Perform binary search on a sorted list to find the index of the target element.

    Parameters:
    arr (list): A sorted list of elements.
    target: The element to search for in the list.

    Returns:
    int: The index of the target element if found, otherwise -1.

    Complexity:
    - Best Case: O(1) when the target is at the middle of the list.
    - Average Case: O(log n) where n is the number of elements in the list.
    - Worst Case: O(log n) when the target is not found or is at one of the ends of the list.
    """
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2  # Calculate the middle index

        # Check if the target is present at mid
        if arr[mid] == target:
            return mid
        # If target is greater, ignore the left half
        elif arr[mid] < target:
            left = mid + 1
        # If target is smaller, ignore the right half
        else:
            right = mid - 1

    return -1  # Target was not found in the list
```

```
        # Check if the target is present at mid
        if arr[mid] == target:
            return mid
        # If target is greater, ignore the left half
        elif arr[mid] < target:
            left = mid + 1
        # If target is smaller, ignore the right half
        else:
            right = mid - 1

    return -1  # Target was not found in the list
# Get the sorted list and target from user input
arr_input = input("Enter a sorted list of elements (comma separated): ")
target_input = input("Enter the target element to search for: ")

# Convert the input string to a list of integers
arr = [int(x.strip()) for x in arr_input.split(",")]

# Convert the target input to an integer
target = int(target_input)
# Call the binary search function and print the result
result = binary_search(arr, target)
if result != -1:
    print(f"Target found at index: {result}")
else:
    print("Target not found in the list.")
```

```
BLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

er the target element to search for: 4
er a sorted list of elements (comma separated): 1,2,3,4,5
er the target element to search for: 4
er the target element to search for: 4
get found at index: 3
C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
get found at index: 3
get found at index: 3
C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
```

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.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input.
# Get the number of appointments
num_appointments = int(input("Enter the number of appointments: "))
# Initialize an empty list to store appointment records
appointments = []
# Get appointment details from the user
for i in range(num_appointments):
    appointment_id = input("Enter appointment ID: ")
    patient_name = input("Enter patient name: ")
    doctor_name = input("Enter doctor name: ")
    appointment_time = input("Enter appointment time (HH:MM): ")
    consultation_fee = float(input("Enter consultation fee: "))
    # Create a dictionary for each appointment and add it to the list
    appointment = {
        "appointment_id": appointment_id,
        "patient_name": patient_name,
        "doctor_name": doctor_name,
        "appointment_time": appointment_time,
        "consultation_fee": consultation_fee
    }
    appointments.append(appointment)
# Function to search for an appointment by ID using linear search
def search_appointment(appointment_id):
    for appointment in appointments:
        if appointment["appointment_id"] == appointment_id:
            return appointment
    return None
# Function to sort appointments by time using bubble sort
def sort_appointments_by_time():
    n = len(appointments)
    for i in range(n):
        for j in range(0, n-i-1):
            if appointments[j]["appointment_time"] > appointments[j+1]["appointment_time"]:
                appointments[j], appointments[j+1] = appointments[j+1], appointments[j] # Function to sort appointments by consultation fe
```

```python
def sort_appointments_by_fee():
    n = len(appointments)
    for i in range(n):
        for j in range(0, n-i-1):
            if appointments[j]["consultation_fee"] > appointments[j+1]["consultation_fee"]:
                appointments[j], appointments[j+1] = appointments[j+1], appointments[j] # Main program
# Get the appointment ID to search for
search_id = input("Enter appointment ID to search: ")
# Search for the appointment
result = search_appointment(search_id)
if result:
    print("Appointment found:")
    print(result)
else:
    print("Appointment not found.")
# Sort appointments by time and display
sort_appointments_by_time()
print("Appointments sorted by time:")
for appointment in appointments:
    print(appointment)
# Sort appointments by consultation fee and display
sort_appointments_by_fee()
print("Appointments sorted by consultation fee:")
for appointment in appointments:
    print(appointment)
```

Output:

```
Enter consultation fee: 23002
Enter appointment ID to search: 12
Appointment found:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
Enter appointment time (HH:MM): MM
Enter consultation fee: 23002
Enter appointment time (HH:MM): MM
Enter appointment time (HH:MM): MM
Enter consultation fee: 23002
Enter appointment ID to search: 12
Appointment found:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
Appointments sorted by time:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
Enter appointment time (HH:MM): MM
Enter consultation fee: 23002
Enter appointment ID to search: 12
Appointment found:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
Appointments sorted by time:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
Enter appointment ID to search: 12
Appointment found:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
Appointments sorted by time:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
Appointments sorted by time:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
Appointments sorted by time:
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
Appointments sorted by consultation fee:
Appointments sorted by consultation fee:
{'appointment_id': '14', 'patient_name': 'bdhwbehjf', 'doctor_name': 'vgwvfdew', 'appointment_time': 'MM', 'consultation_fee': 23002.0}
{'appointment_id': '12', 'patient_name': 'hwevdh', 'doctor_name': 'dgvdwed', 'appointment_time': 'HH', 'consultation_fee': 152000.0}
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
```

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.

```python
num_tickets = int(input("Enter the number of tickets to be booked: "))
# Create an empty list to store ticket details
tickets = []
# Get ticket details from the user
for i in range(num_tickets):
    ticket_id = input("Enter ticket ID: ")
    passenger_name = input("Enter passenger name: ")
    train_number = input("Enter train number: ")
    seat_number = input("Enter seat number: ")
    travel_date = input("Enter travel date (YYYY-MM-DD): ")
    # Append the ticket details as a dictionary to the tickets list
    tickets.append({
        "ticket_id": ticket_id,
        "passenger_name": passenger_name,
        "train_number": train_number,
        "seat_number": seat_number,
        "travel_date": travel_date
    })
# Function to search for a ticket by ticket ID
def search_ticket(ticket_id):
    for ticket in tickets:
        if ticket["ticket_id"] == ticket_id:
            return ticket
    return None
# Function to sort tickets by travel date
def sort_tickets_by_date():
    return sorted(tickets, key=lambda x: x["travel_date"])
# Function to sort tickets by seat number
def sort_tickets_by_seat():
    return sorted(tickets, key=lambda x: x["seat_number"])
# Get the ticket ID to search for
search_id = input("Enter ticket ID to search: ")
# Search for the ticket and print the result
result = search_ticket(search_id)
if result:
    print("Ticket found:", result)
else:
    print("Ticket not found.")
# Sort tickets by travel date and print the sorted list
sorted_by_date = sort_tickets_by_date()
print("Tickets sorted by travel date:")
for ticket in sorted_by_date:
    print(ticket)
# Sort tickets by seat number and print the sorted list
sorted_by_seat = sort_tickets_by_seat()
print("Tickets sorted by seat number:")
for ticket in sorted_by_seat:
    print(ticket)
```

Output:

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.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input.
# Get the number of records to be entered
num_records = int(input("Enter the number of records: "))
# Initialize an empty list to store the records
records = []
# Loop to get the records from user input
for _ in range(num_records):
    student_id = input("Enter student ID: ")
    room_number = input("Enter room number: ")
    floor = input("Enter floor: ")
    allocation_date = input("Enter allocation date (YYYY-MM-DD): ")
    # Append the record as a dictionary to the records list
    records.append({
        "student_id": student_id,
        "room_number": room_number,
        "floor": floor,
        "allocation_date": allocation_date
    })
# Function to search allocation details using student ID

def search_by_student_id(student_id):
    for record in records:
        if record["student_id"] == student_id:
            return record
    return None  # Return None if no record is found
# Function to sort records based on room number
def sort_by_room_number():
    return sorted(records, key=lambda x: x["room_number"])
# Function to sort records based on allocation date
def sort_by_allocation_date():
    return sorted(records, key=lambda x: x["allocation_date"])
# Example usage
# Search for a student ID
search_id = input("Enter student ID to search: ")
result = search_by_student_id(search_id)
```

```python
# Search for a student ID
search_id = input("Enter student ID to search: ")
result = search_by_student_id(search_id)
if result:
    print("Record found:", result)
else:
    print("No record found for student ID:", search_id)
# Sort records by room number
sorted_by_room = sort_by_room_number()
print("Records sorted by room number:")
for record in sorted_by_room:
    print(record)
# Sort records by allocation date
sorted_by_date = sort_by_allocation_date()
print("Records sorted by allocation date:")
for record in sorted_by_date:
    print(record)

```

```
ROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

nter student ID to search: 32
o record found for student ID: 32
ecords sorted by room number:
'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
ecords sorted by allocation date:
nter allocation date (YYYY-MM-DD): 2004-90
nter student ID to search: 32
o record found for student ID: 32
ecords sorted by room number:
'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
ecords sorted by allocation date:
ecords sorted by room number:
'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
ecords sorted by allocation date:
'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
ecords sorted by allocation date:
ecords sorted by allocation date:
'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
S C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
```

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.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input.
# Get the number of movies to be entered
num_movies = int(input("Enter the number of movies: "))
# Initialize an empty list to store movie records
movies = []
# Loop to get movie details from the user
for _ in range(num_movies):
    movie_id = input("Enter movie ID: ")
    title = input("Enter movie title: ")
    genre = input("Enter movie genre: ")
    rating = float(input("Enter movie rating (0-10): "))
    release_year = int(input("Enter release year: "))
    # Append the movie record as a dictionary to the movies list
    movies.append({
        "movie_id": movie_id,
        "title": title,
        "genre": genre,
        "rating": rating,
        "release_year": release_year
    })
# Function to search for a movie by movie ID
def search_movie_by_id(movie_id):
    for movie in movies:
        if movie["movie_id"] == movie_id:
            return movie
    return None  # Return None if movie is not found
# Function to sort movies by rating
def sort_movies_by_rating():
    return sorted(movies, key=lambda x: x["rating"], reverse=True)
# Function to sort movies by release year
def sort_movies_by_release_year():
```

```python
            return movie
    return None  # Return None if movie is not found
# Function to sort movies by rating
def sort_movies_by_rating():
    return sorted(movies, key=lambda x: x["rating"], reverse=True)
# Function to sort movies by release year
def sort_movies_by_release_year():
    return sorted(movies, key=lambda x: x["release_year"], reverse=True)
# Example usage
# Search for a movie by ID
search_id = input("Enter movie ID to search: ")
found_movie = search_movie_by_id(search_id)
if found_movie:
    print("Movie found:", found_movie)
else:
    print("Movie not found.")
# Sort movies by rating and display
sorted_by_rating = sort_movies_by_rating()
print("Movies sorted by rating:")
for movie in sorted_by_rating:
    print(movie)
# Sort movies by release year and display
sorted_by_release_year = sort_movies_by_release_year()
print("Movies sorted by release year:")
```

Output:

```
Enter the number of records: 1
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:\Program Files\Python39\python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab-12.5.py"
Enter the number of records: 1
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:\Program Files\Python39\python.exe" "c:/Users/Ajay Kumar/OneDrive/Desktop/Ai/lab-12.5.py"
Enter the number of records: 1
Enter student ID: 15
Enter room number: 15
Enter floor: 15
Enter allocation date (YYYY-MM-DD): 2004-90
Enter student ID to search: 32
No record found for student ID: 32
Records sorted by room number:
{'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
Records sorted by allocation date:
Enter allocation date (YYYY-MM-DD): 2004-90
Enter student ID to search: 32
No record found for student ID: 32
Records sorted by room number:
{'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
Records sorted by allocation date:
Records sorted by room number:
{'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
Records sorted by allocation date:
{'student_id': '15', 'room_number': '15', 'floor': '15', 'allocation_date': '2004-90'}
Records sorted by allocation date:
```

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.

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user input.
# Get the number of crops to be entered
num_crops = int(input("Enter the number of crops: "))
# Initialize an empty list to store crop data
crops = []
# Loop to get crop details from the user
for i in range(num_crops):
    crop_id = input("Enter crop ID: ")
    crop_name = input("Enter crop name: ")
    soil_moisture = float(input("Enter soil moisture level: "))
    temperature = float(input("Enter temperature: "))
    yield_estimate = float(input("Enter yield estimate: "))
    # Append the crop details as a dictionary to the crops list
    crops.append({
        "crop_id": crop_id,
        "crop_name": crop_name,
        "soil_moisture": soil_moisture,
        "temperature": temperature,
        "yield_estimate": yield_estimate
    })
# Function to search crop details by crop ID
def search_crop(crop_id):
    for crop in crops:
        if crop["crop_id"] == crop_id:
            return crop
    return None
# Function to sort crops by soil moisture level
def sort_by_moisture():
    return sorted(crops, key=lambda x: x["soil_moisture"])
# Function to sort crops by yield estimate
def sort_by_yield():
    return sorted(crops, key=lambda x: x["yield_estimate"])
# Get crop ID to search from the user
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
# Get crop ID to search from the user
search_id = input("Enter crop ID to search: ")
# Search for the crop and display details
crop_details = search_crop(search_id)
if crop_details:
    print("Crop Details:")
    print(f"Crop ID: {crop_details['crop_id']}")
    print(f"Crop Name: {crop_details['crop_name']}")
    print(f"Soil Moisture Level: {crop_details['soil_moisture']}")
    print(f"Temperature: {crop_details['temperature']}")
    print(f"Yield Estimate: {crop_details['yield_estimate']}")
else:
    print("Crop not found.")
# Get sorting preference from the user
sort_preference = input("Sort by (1) Soil Moisture or (2) Yield Estimate? Enter 1 or 2: ")
# Sort and display crops based on user preference
if sort_preference == "1":
    sorted_crops = sort_by_moisture()
    print("Crops sorted by Soil Moisture Level:")
    for crop in sorted_crops:
        print(f"{crop['crop_name']} - Soil Moisture: {crop['soil_moisture']}")
elif sort_preference == "2":
    sorted_crops = sort_by_yield()
    print("Crops sorted by Yield Estimate:")
    for crop in sorted_crops:
        print(f"{crop['crop_name']} - Yield Estimate: {crop['yield_estimate']}")
else:
    print("Invalid sorting preference.")
```

```
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai> & "C:\Program Files\Python39\py
Enter the number of crops: 1
Enter crop ID: 12
Enter crop name: fhdfvj
Enter soil moisture level: 5
Enter temperature: 15
Enter yield estimate: 25
Enter the number of crops: 1
Enter crop ID: 12
Enter crop name: fhdfvj
Enter soil moisture level: 5
Enter temperature: 15
Enter crop ID: 12
Enter crop name: fhdfvj
Enter soil moisture level: 5
Enter temperature: 15
Enter crop name: fhdfvj
Enter soil moisture level: 5
Enter temperature: 15
Enter soil moisture level: 5
Enter temperature: 15
Enter temperature: 15
Enter yield estimate: 25
Enter crop ID to search: 1
Crop not found.
Sort by (1) Soil Moisture or (2) Yield Estimate? Enter 1 or 2: 1
Crops sorted by Soil Moisture Level:
```

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

```python
#  give the python code for the above task with the well documented and well drafted and optimized Code: arr from user in
# Get the number of flights to be entered
num_flights = int(input("Enter the number of flights: "))
# Initialize an empty list to store flight information
flights = []
# Loop to get flight details from the user
for i in range(num_flights):
    flight_id = input("Enter flight ID: ")
    airline_name = input("Enter airline name: ")
    departure_time = input("Enter departure time (HH:MM): ")
    arrival_time = input("Enter arrival time (HH:MM): ")
    status = input("Enter flight status (On Time/Delayed): ")
    # Append the flight details as a dictionary to the flights list
    flights.append({
        "flight_id": flight_id,
        "airline_name": airline_name,
        "departure_time": departure_time,
        "arrival_time": arrival_time,
        "status": status
    })
# Function to search for flight details using flight ID
def search_flight(flight_id):
    for flight in flights:
        if flight["flight_id"] == flight_id:
            return flight
    return None
# Function to sort flights based on departure time
def sort_flights_by_departure():
```

```python
    return None
# Function to sort flights based on departure time
def sort_flights_by_departure():
    return sorted(flights, key=lambda x: x["departure_time"])
# Function to sort flights based on arrival time
def sort_flights_by_arrival():
    return sorted(flights, key=lambda x: x["arrival_time"])
# Example usage
# Search for a flight
search_id = input("Enter flight ID to search: ")
result = search_flight(search_id)
if result:
    print("Flight found:", result)
else:
    print("Flight not found.")
# Sort flights by departure time
sorted_by_departure = sort_flights_by_departure()
print("Flights sorted by departure time:")
for flight in sorted_by_departure:
    print(flight)
# Sort flights by arrival time
sorted_by_arrival = sort_flights_by_arrival()
print("Flights sorted by arrival time:")
for flight in sorted_by_arrival:
    print(flight)
```

```
Enter the number of flights: 2
Enter flight ID: 15
Enter airline name: ygeiusf
Enter departure time (HH:MM): HH
Enter flight ID: 15
Enter airline name: ygeiusf
Enter departure time (HH:MM): HH
Enter airline name: ygeiusf
Enter departure time (HH:MM): HH
Enter arrival time (HH:MM): 12
Enter departure time (HH:MM): HH
Enter arrival time (HH:MM): 12
Enter arrival time (HH:MM): 12
Enter flight status (On Time/Delayed):
Enter flight status (On Time/Delayed):
Enter flight ID: 56
Enter airline name: bhfrbhjgj
Enter departure time (HH:MM): MM
Enter arrival time (HH:MM): MM
Enter airline name: bhfrbhjgj
Enter departure time (HH:MM): MM
Enter arrival time (HH:MM): MM
Enter departure time (HH:MM): MM
Enter arrival time (HH:MM): MM
Enter arrival time (HH:MM): MM
Enter flight status (On Time/Delayed): On Time
Enter flight ID to search: 56
Flight found: {'flight_id': '56', 'airline_name': 'bhfrbhjgj', 'departure_time': 'MM', 'arrival_time': 'MM', 'status': 'On Time'}
Flights sorted by departure time:
{'flight_id': '15', 'airline_name': 'ygeiusf', 'departure_time': 'HH', 'arrival_time': '12', 'status': ''}
{'flight_id': '56', 'airline_name': 'bhfrbhjgj', 'departure_time': 'MM', 'arrival_time': 'MM', 'status': 'On Time'}
Flights sorted by arrival time:
{'flight_id': '15', 'airline_name': 'ygeiusf', 'departure_time': 'HH', 'arrival_time': '12', 'status': ''}
{'flight_id': '56', 'airline_name': 'bhfrbhjgj', 'departure_time': 'MM', 'arrival_time': 'MM', 'status': 'On Time'}
PS C:\Users\Ajay Kumar\OneDrive\Desktop\Ai>
```