ASSIGNMENT -12.5

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

and Optimizing Algorithms

NAME:M.NISHITHA

HTNO: 2303A51540

BTNO-19

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.

CODE:

Users > nishithamarepally > 12.5_1.py > merge

```python
def merge_sort(arr):

    if len(arr) <= 1:
        return arr

    # Split the array into two 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):

    rged = []
    i = j = 0

    # Compare elements from both lists and append the smallest
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
```

```
Users > nishithamarepally > 12.5_1.py > merge
15    def merge(left, right):
25            else:
26                merged.append(right[j])
27                j += 1
28
29        # Append any remaining elements from the left list
30        while i < len(left):
31            merged.append(left[i])
32            i += 1
33
34        # Append any remaining elements from the right list
35        while j < len(right):
36            merged.append(right[j])
37            j += 1
38
39        return merged
40
41
42    # Test cases to verify the implementation
43    if __name__ == "__main__":
44        test_array = [38, 27, 43, 3, 9, 82, 10]
45        print("Original array:", test_array)
46        sorted_array = merge_sort(test_array)
47        print("Sorted array:", sorted_array)
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/12.5_1.py
Original array: [38, 27, 43, 3, 9, 82, 10]
Sorted array: [3, 9, 10, 27, 38, 43, 82]
nishithamarepally@nishithas-MacBook-Air ~ %

⊗ 0 ⚠ 0   0
```

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.

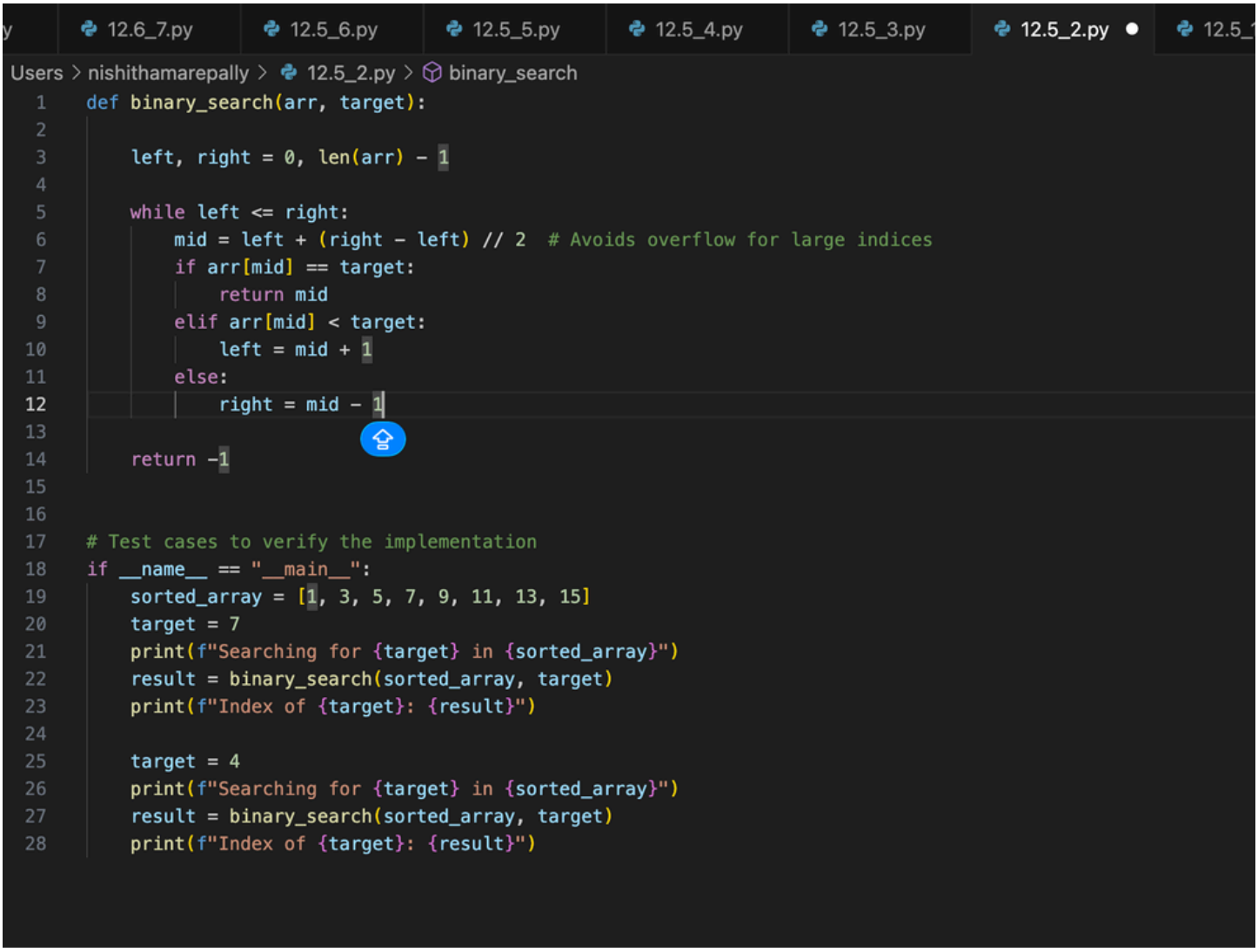
o Include docstrings explaining best, average, and worst-case complexities.

o Test with various inputs.

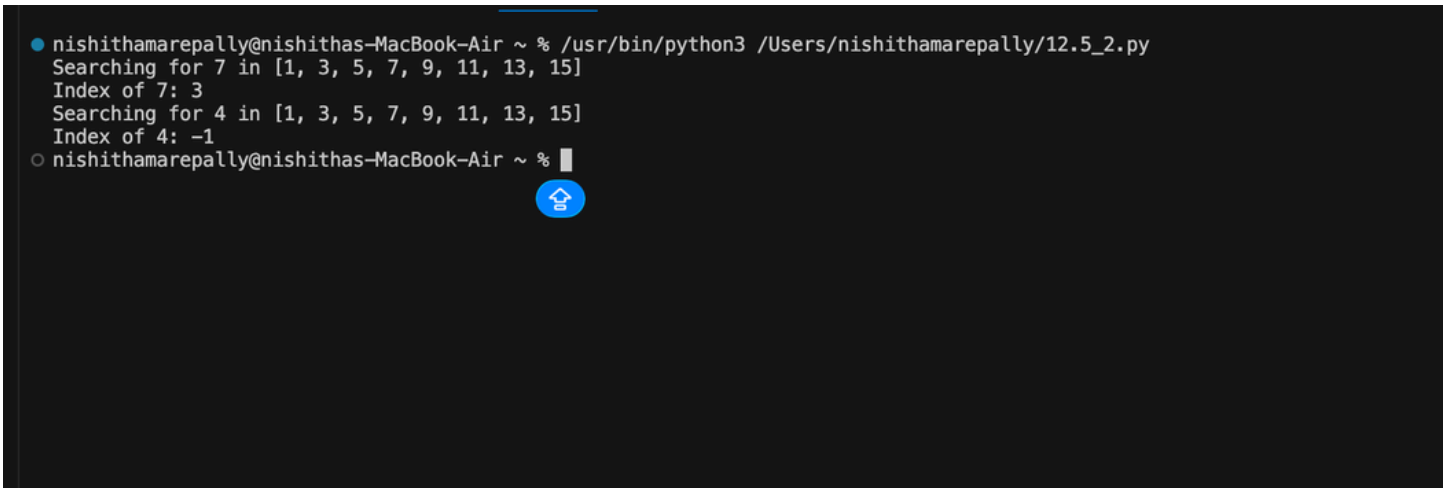
• Expected Output:

o Python code implementing binary search with AI-generated comments and docstrings.

CODE:



OUTPUT:



Task Description #3: Smart Healthcare Appointment SchedulingSystem

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.

CODE :

Users > nishithamarepally > Desktop > 12.5_3.py > merge_sort_appointments

```python
class Appointment:
    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

    def __repr__(self):
        return f"Appointment({self.appointment_id}, {self.patient_name}, {self.doctor_name}, {self.appointment


def binary_search_appointments(appointments, target_id):

    left, right = 0, len(appointments) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if appointments[mid].appointment_id == target_id:
            return appointments[mid]
        elif appointments[mid].appointment_id < target_id:
            left = mid + 1
        else:
            right = mid - 1

    return None


def merge_sort_appointments(appointments, key):

    if len(appointments) <= 1:
        return appointments

    mid = len(appointments) // 2
    left_half = merge_sort_appointments(appointments[:mid], key)
    right_half = merge_sort_appointments(appointments[mid:], key)

    return merge(left_half, right_half, key)
```

⚠ 0 0

12.5_8.py | 12.6_7.py | 12.5_6.py | 12.5_5.py | 12.5_4.py | 12.5_3.py

Users > nishithamarepally > Desktop > 12.5_3.py > merge_sort_appointments

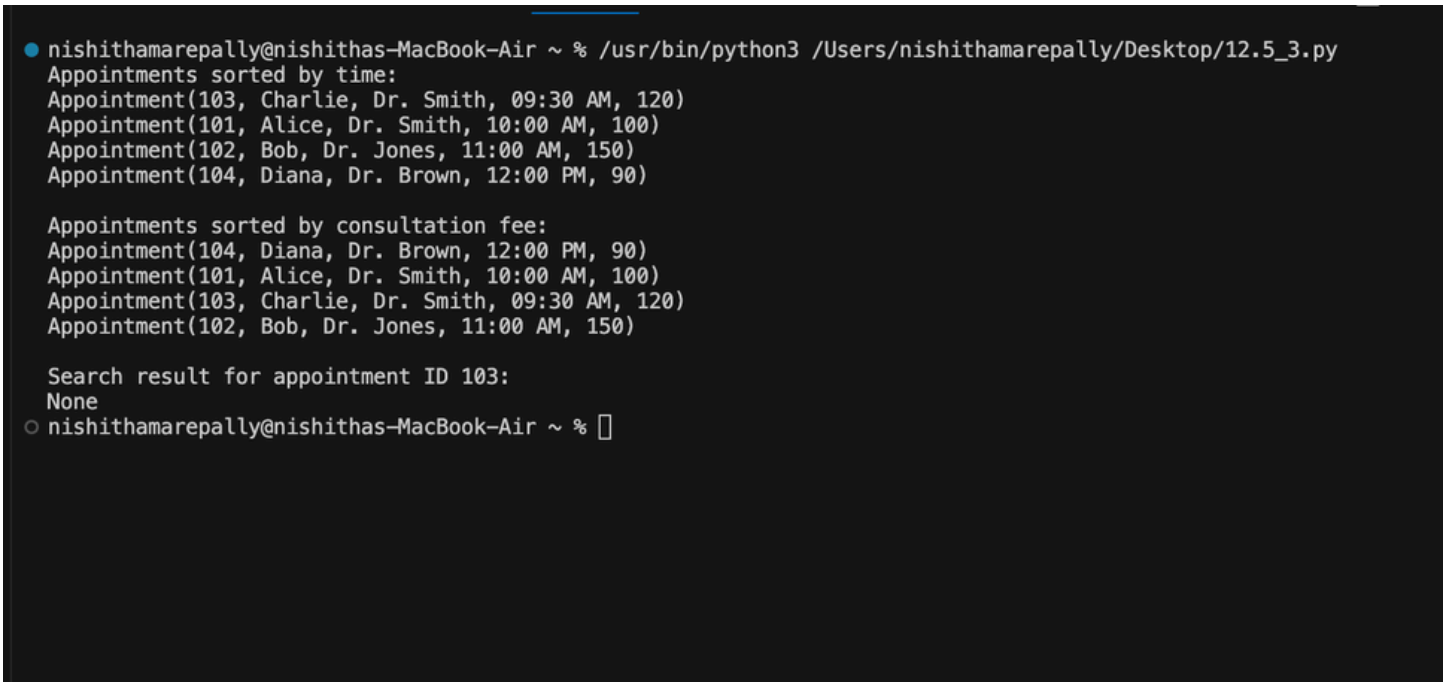
```python
def merge_sort_appointments(appointments, key):
    return merge(left_half, right_half, key)


def merge(left, right, key):
    merged = []
    i = j = 0

    while i < len(left) and j < len(right):
        if getattr(left[i], key) <= getattr(right[j], key):
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1

    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged


# Example Usage
if __name__ == "__main__":
    appointments = [
        Appointment(101, "Alice", "Dr. Smith", "10:00 AM", 100),
        Appointment(102, "Bob", "Dr. Jones", "11:00 AM", 150),
        Appointment(103, "Charlie", "Dr. Smith", "09:30 AM", 120),
        Appointment(104, "Diana", "Dr. Brown", "12:00 PM", 90),
    ]

    # Sort appointments by time
    sorted_by_time = merge_sort_appointments(appointments, key="appointment_time")
    print("Appointments sorted by time:")
    for appointment in sorted_by_time:
        print(appointment)

```

12.5_8.py | 12.6_7.py | 12.5_6.py | 12.5_5.py | 12.5_4.py | 12.5_3.py

sers > nishithamarepally > Desktop > 12.5_3.py > merge_sort_appointments

```python
    for appointment in sorted_by_time:
        print(appointment)

    # Sort appointments by consultation fee
    sorted_by_fee = merge_sort_appointments(appointments, key="consultation_fee")
    print("\nAppointments sorted by consultation fee:")
    for appointment in sorted_by_fee:
        print(appointment)

    # Search for an appointment by ID
    target_id = 103
    result = binary_search_appointments(sorted_by_time, target_id)
    print(f"\nSearch result for appointment ID {target_id}:")
    print(result)
```

OUTPUT:



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.

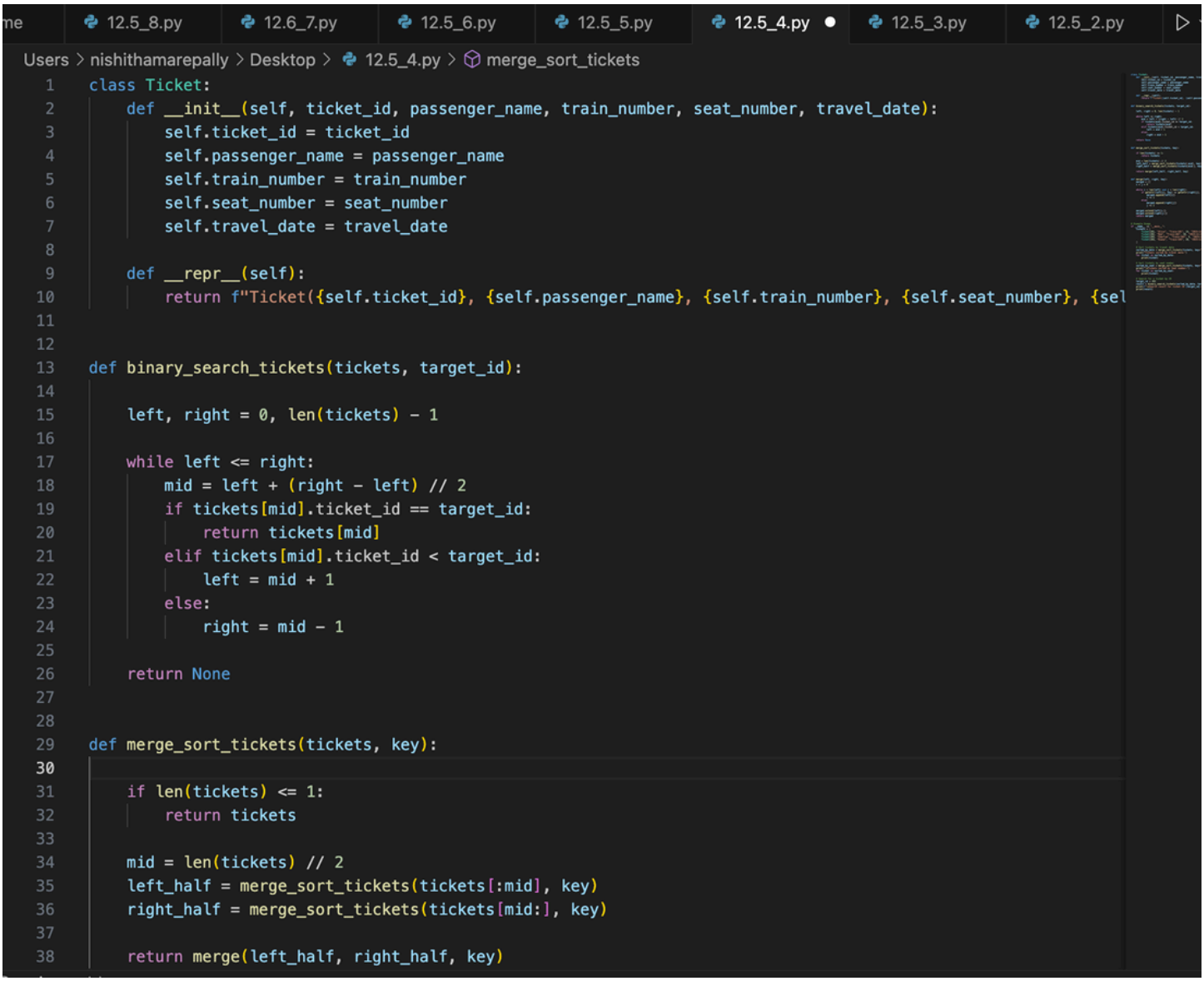
Student Task

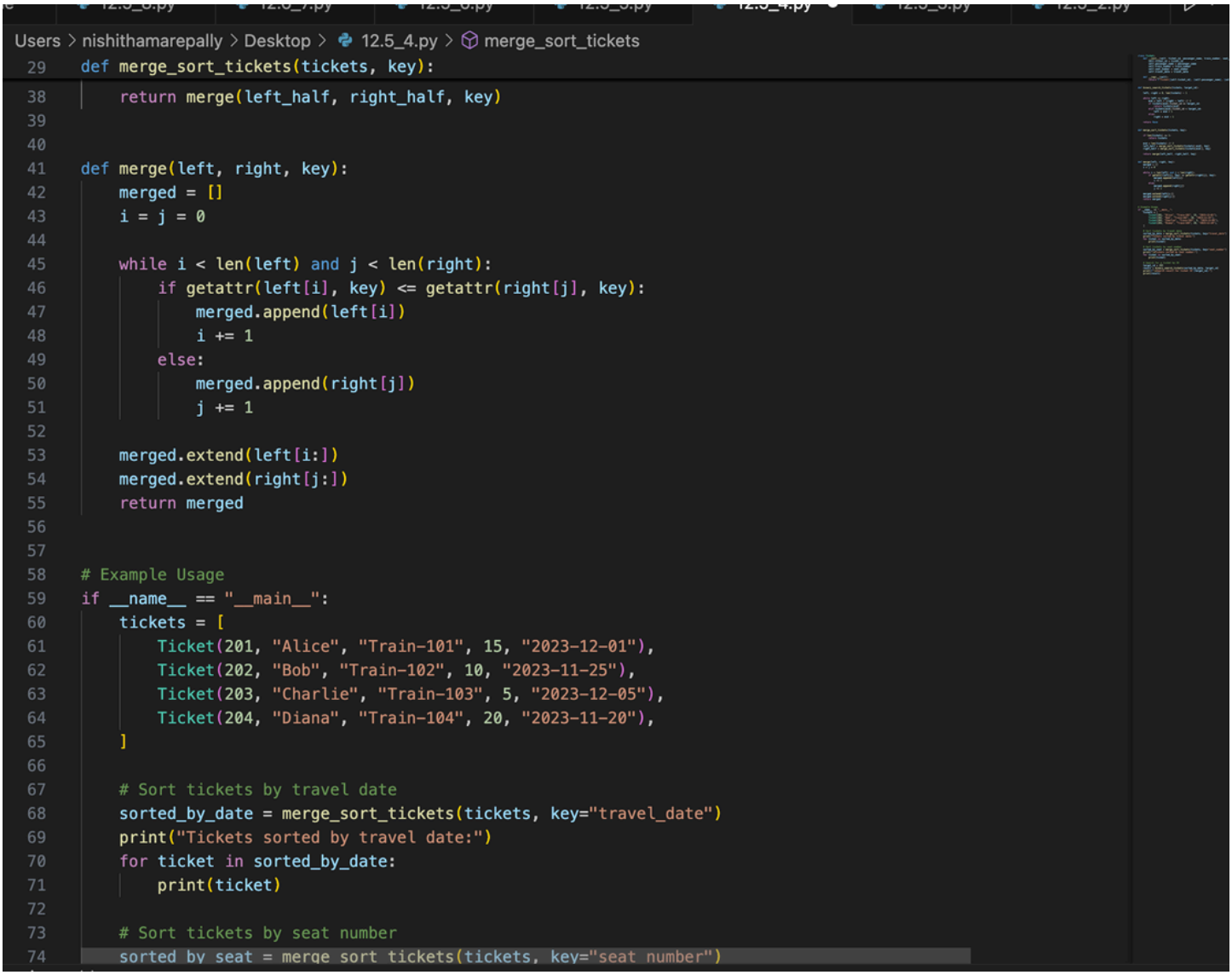
• Identify efficient algorithms using AI assistance.

• Justify the algorithm choices.

• Implement searching and sorting in Python.

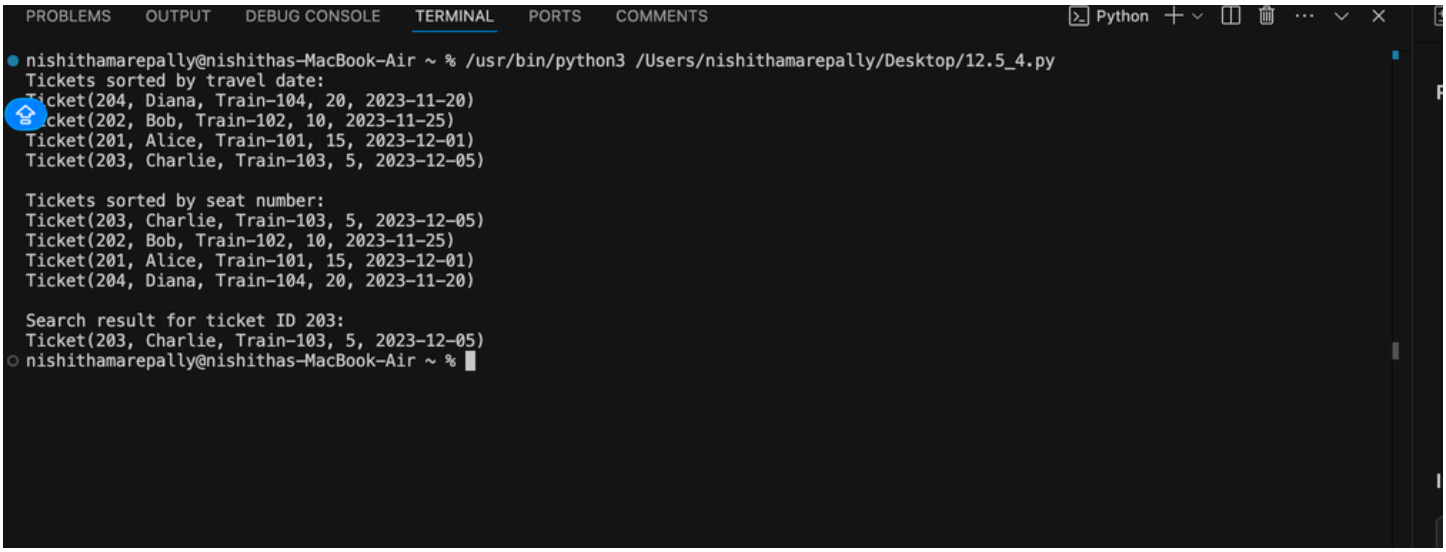
CODE:


12.5_8.py
12.6_7.py
12.5_6.py
12.5_5.py
12.5_4.py
12.5_3.py
12.5_2.py
Users > nishithamarepally > Desktop > 12.5_4.py > merge_sort_tickets
class Ticket:
def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
self.ticket_id = ticket_id
self.passenger_name = passenger_name
self.train_number = train_number
self.seat_number = seat_number
self.travel_date = travel_date
def __repr__(self):
return f"Ticket({self.ticket_id}, {self.passenger_name}, {self.train_number}, {self.seat_number}, {sel
def binary_search_tickets(tickets, target_id):
left, right = 0, len(tickets) - 1
while left <= right:
mid = left + (right - left) // 2
if tickets[mid].ticket_id == target_id:
return tickets[mid]
elif tickets[mid].ticket_id < target_id:
left = mid + 1
else:
right = mid - 1
return None
def merge_sort_tickets(tickets, key):
if len(tickets) <= 1:
return tickets
mid = len(tickets) // 2
left_half = merge_sort_tickets(tickets[:mid], key)
right_half = merge_sort_tickets(tickets[mid:], key)
return merge(left_half, right_half, key)


Users > nishithamarepally > Desktop > 12.5_4.py > merge_sort_tickets
def merge_sort_tickets(tickets, key):
return merge(left_half, right_half, key)
def merge(left, right, key):
merged = []
i = j = 0
while i < len(left) and j < len(right):
if getattr(left[i], key) <= getattr(right[j], key):
merged.append(left[i])
i += 1
else:
merged.append(right[j])
j += 1
merged.extend(left[i:])
merged.extend(right[j:])
return merged
# Example Usage
if __name__ == "__main__":
tickets = [
Ticket(201, "Alice", "Train-101", 15, "2023-12-01"),
Ticket(202, "Bob", "Train-102", 10, "2023-11-25"),
Ticket(203, "Charlie", "Train-103", 5, "2023-12-05"),
Ticket(204, "Diana", "Train-104", 20, "2023-11-20"),
]
# Sort tickets by travel date
sorted_by_date = merge_sort_tickets(tickets, key="travel_date")
print("Tickets sorted by travel date:")
for ticket in sorted_by_date:
print(ticket)
# Sort tickets by seat number
sorted_by_seat = merge_sort_tickets(tickets, key="seat_number")


12.5_8.py
12.6_7.py
12.5_6.py
12.5_5.py
12.5_4.py
12.5_3.py
12.5_2.py
Users > nishithamarepally > Desktop > 12.5_4.py > merge_sort_tickets
print(ticket)
# Sort tickets by seat number
sorted_by_seat = merge_sort_tickets(tickets, key="seat_number")
print("\nTickets sorted by seat number:")
for ticket in sorted_by_seat:
print(ticket)
# Search for a ticket by ID
target_id = 203
result = binary_search_tickets(sorted_by_date, target_id)
print(f"\nSearch result for ticket ID {target_id}:")
print(result)

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.

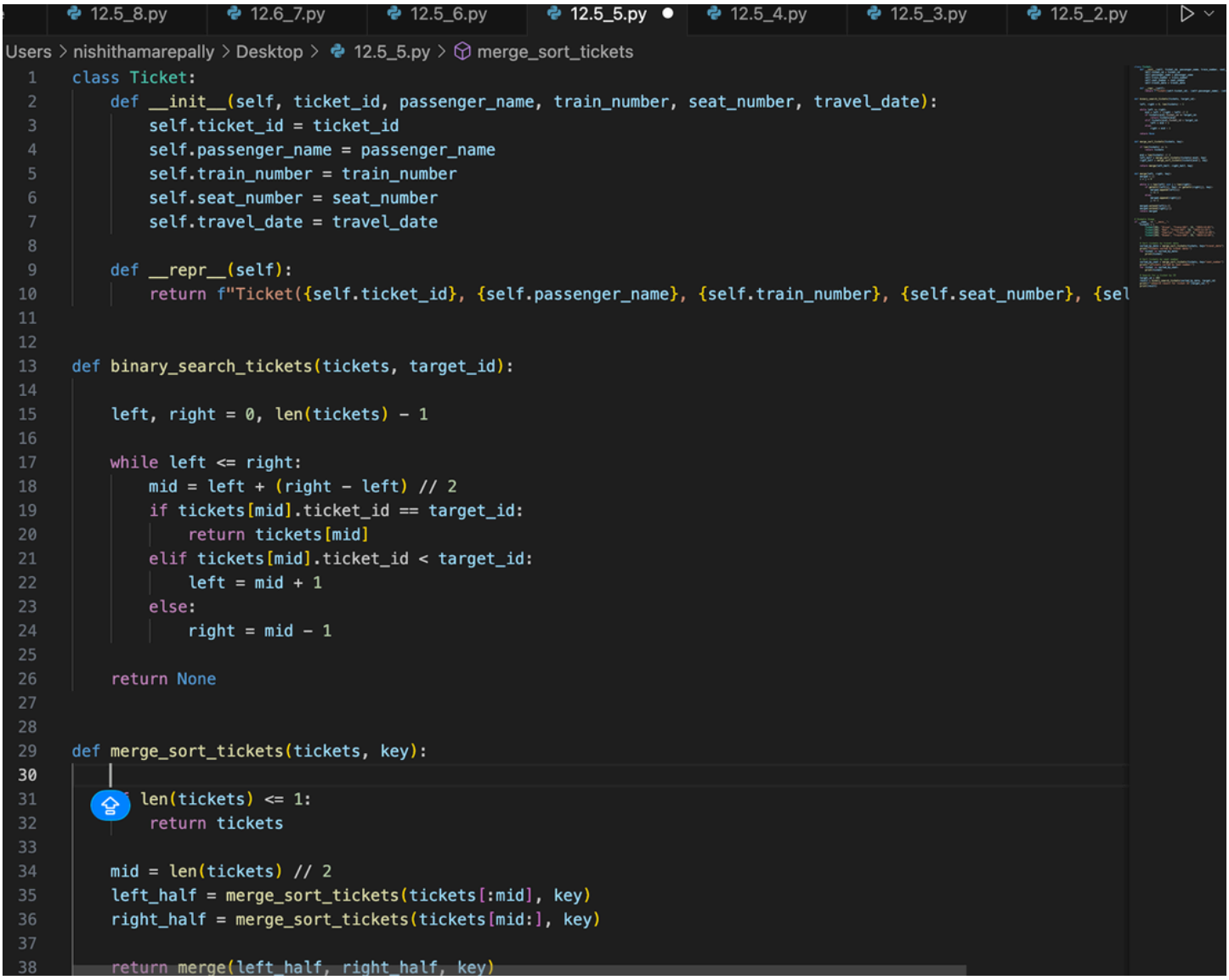
Student Task

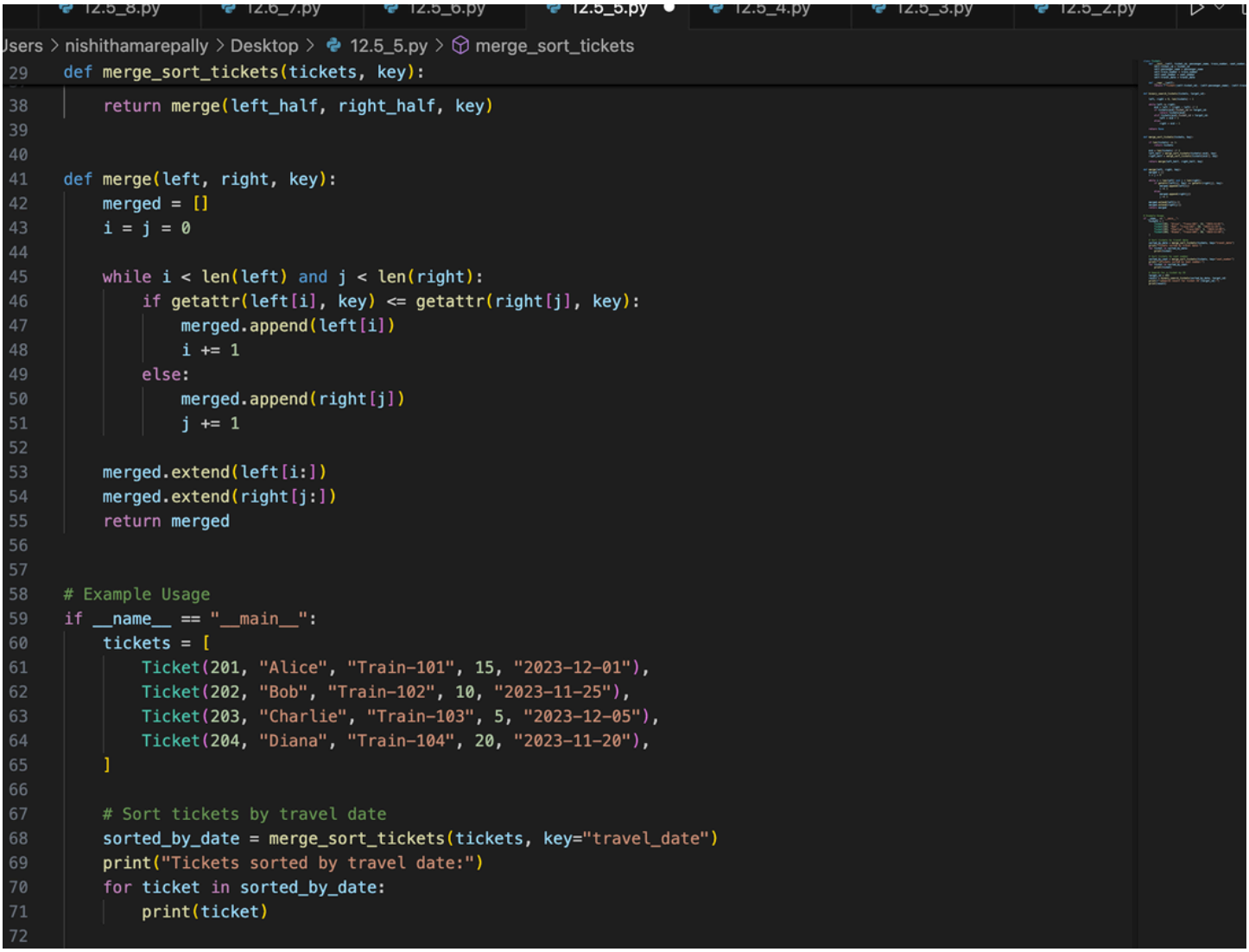
• Use AI to suggest optimized algorithms.

• Justify the selections.

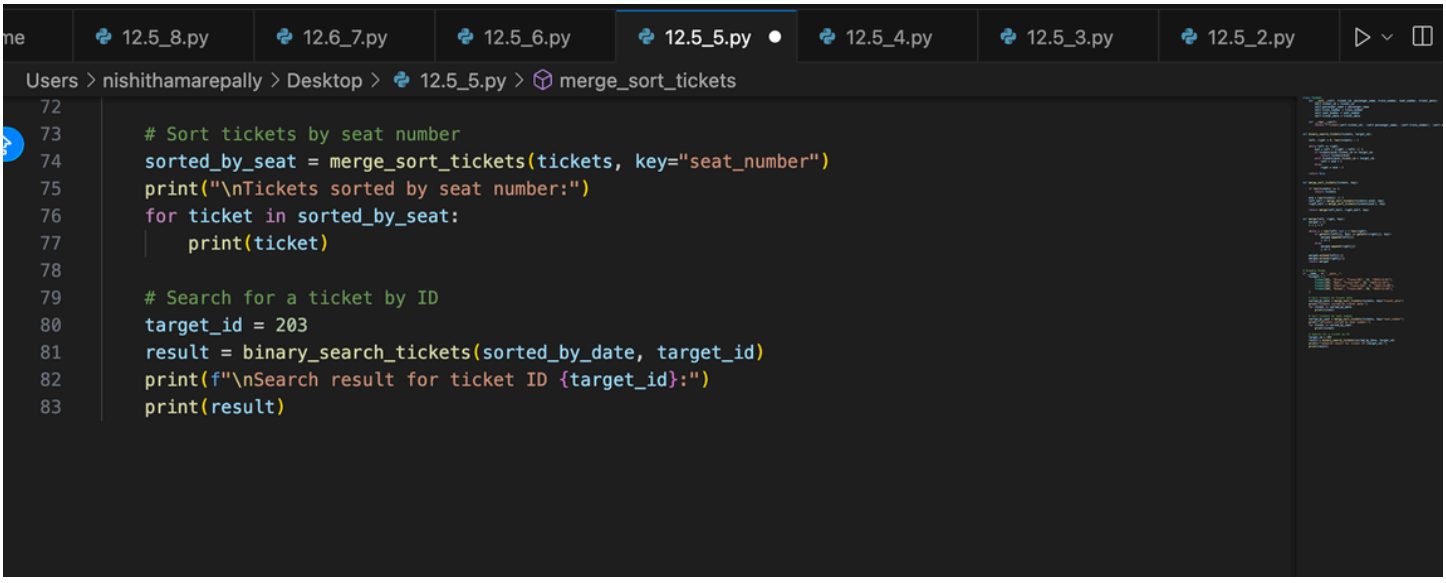
• Implement the solution in Python.

CODE:


12.5_8.py
12.6_7.py
12.5_6.py
12.5_5.py
12.5_4.py
12.5_3.py
12.5_2.py
Users > nishithamarepally > Desktop > 12.5_5.py > merge_sort_tickets
class Ticket:
def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
self.ticket_id = ticket_id
self.passenger_name = passenger_name
self.train_number = train_number
self.seat_number = seat_number
self.travel_date = travel_date
def __repr__(self):
return f"Ticket({self.ticket_id}, {self.passenger_name}, {self.train_number}, {self.seat_number}, {sel
def binary_search_tickets(tickets, target_id):
left, right = 0, len(tickets) - 1
while left <= right:
mid = left + (right - left) // 2
if tickets[mid].ticket_id == target_id:
return tickets[mid]
elif tickets[mid].ticket_id < target_id:
left = mid + 1
else:
right = mid - 1
return None
def merge_sort_tickets(tickets, key):
len(tickets) <= 1:
return tickets
mid = len(tickets) // 2
left_half = merge_sort_tickets(tickets[:mid], key)
right_half = merge_sort_tickets(tickets[mid:], key)
return merge(left_half, right_half, key)

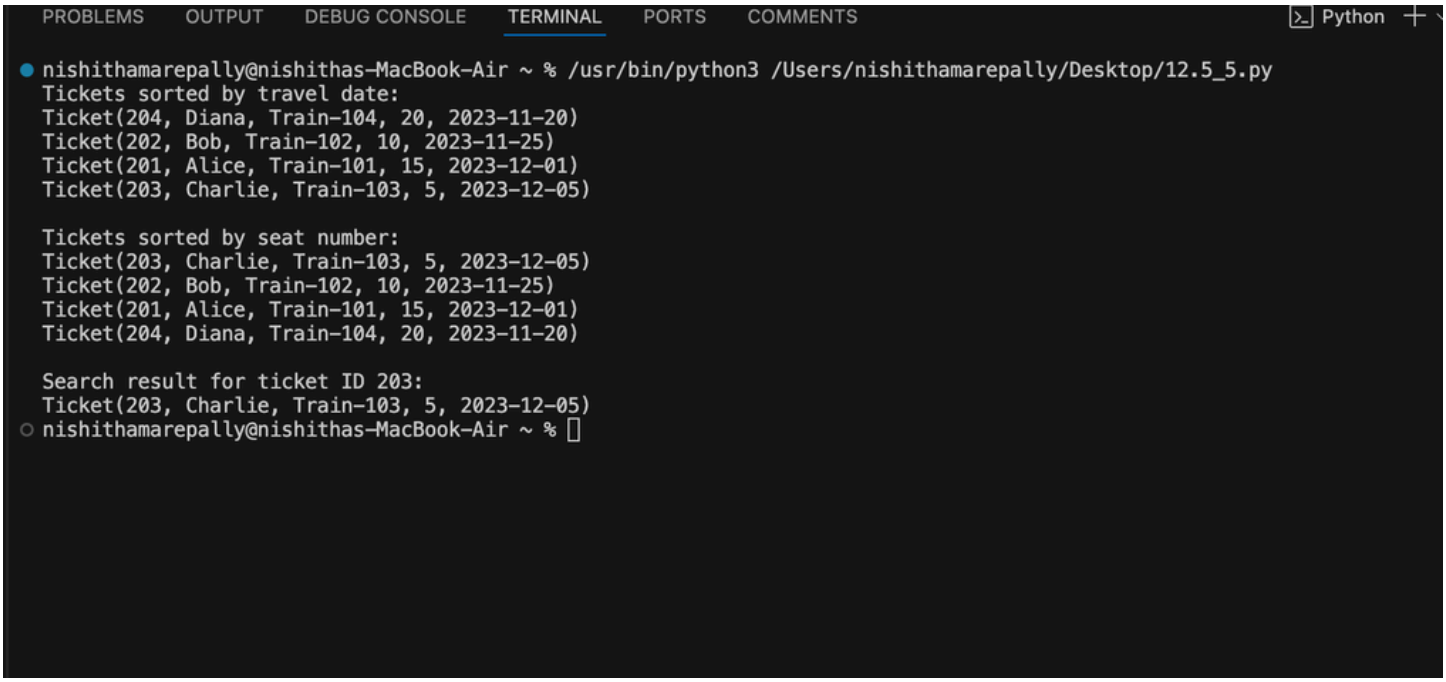


```
Users > nishithamarepally > Desktop > 12.5_5.py > merge_sort_tickets
29  def merge_sort_tickets(tickets, key):
38      return merge(left_half, right_half, key)
39
40
41  def merge(left, right, key):
42      merged = []
43      i = j = 0
44
45      while i < len(left) and j < len(right):
46          if getattr(left[i], key) <= getattr(right[j], key):
47              merged.append(left[i])
48              i += 1
49          else:
50              merged.append(right[j])
51              j += 1
52
53      merged.extend(left[i:])
54      merged.extend(right[j:])
55      return merged
56
57
58  # Example Usage
59  if __name__ == "__main__":
60      tickets = [
61          Ticket(201, "Alice", "Train-101", 15, "2023-12-01"),
62          Ticket(202, "Bob", "Train-102", 10, "2023-11-25"),
63          Ticket(203, "Charlie", "Train-103", 5, "2023-12-05"),
64          Ticket(204, "Diana", "Train-104", 20, "2023-11-20"),
65      ]
66
67      # Sort tickets by travel date
68      sorted_by_date = merge_sort_tickets(tickets, key="travel_date")
69      print("Tickets sorted by travel date:")
70      for ticket in sorted_by_date:
71          print(ticket)
72
```



```
12.5_8.py | 12.6_7.py | 12.5_6.py | 12.5_5.py | 12.5_4.py | 12.5_3.py | 12.5_2.py
Users > nishithamarepally > Desktop > 12.5_5.py > merge_sort_tickets
72
73      # Sort tickets by seat number
74      sorted_by_seat = merge_sort_tickets(tickets, key="seat_number")
75      print("\nTickets sorted by seat number:")
76      for ticket in sorted_by_seat:
77          print(ticket)
78
79      # Search for a ticket by ID
80      target_id = 203
81      result = binary_search_tickets(sorted_by_date, target_id)
82      print(f"\nSearch result for ticket ID {target_id}:")
83      print(result)
```

OUTPUT:



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

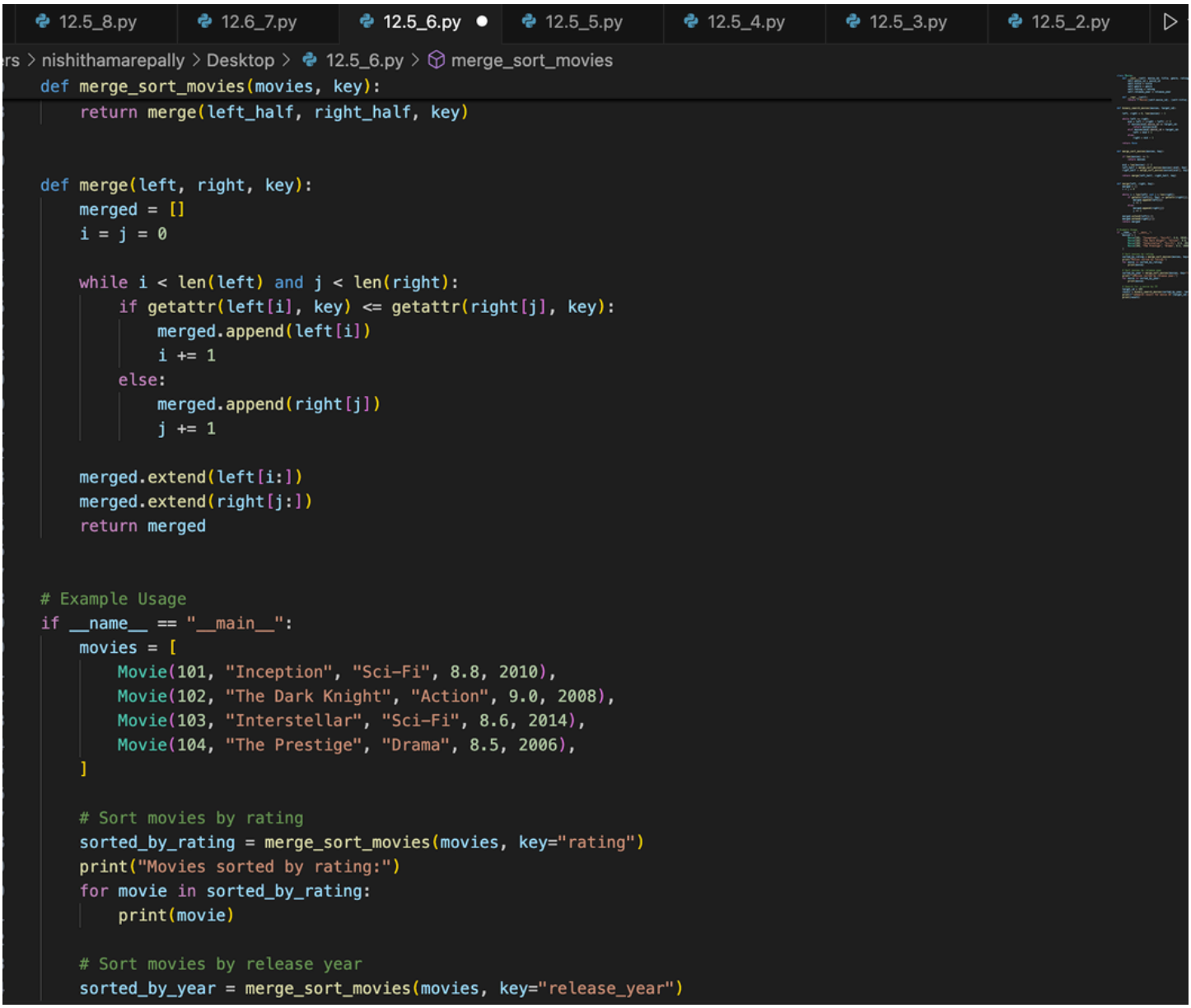
• Recommend searching and sorting algorithms using AI.

• Justify the chosen algorithms.

• Implement Python functions.

CODE:

```python
class Movie:
    def __init__(self, movie_id, title, genre, rating, release_year):
        self.movie_id = movie_id
        self.title = title
        self.genre = genre
        self.rating = rating
        self.release_year = release_year

    def __repr__(self):
        return f"Movie({self.movie_id}, {self.title}, {self.genre}, {self.rating}, {self.release_year})"


def binary_search_movies(movies, target_id):

    left, right = 0, len(movies) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if movies[mid].movie_id == target_id:
            return movies[mid]
        elif movies[mid].movie_id < target_id:
            left = mid + 1
        else:
            right = mid - 1

    return None


def merge_sort_movies(movies, key):

    if len(movies) <= 1:
        return movies

    mid = len(movies) // 2
    left_half = merge_sort_movies(movies[:mid], key)
    right_half = merge_sort_movies(movies[mid:], key)

    return merge(left_half, right_half, key)
```


12.5_8.py
12.6_7.py
12.5_6.py
12.5_5.py
12.5_4.py
12.5_3.py
12.5_2.py
rs > nishithamarepally > Desktop > 12.5_6.py > merge_sort_movies
def merge_sort_movies(movies, key):
return merge(left_half, right_half, key)
def merge(left, right, key):
merged = []
i = j = 0
while i < len(left) and j < len(right):
if getattr(left[i], key) <= getattr(right[j], key):
merged.append(left[i])
i += 1
else:
merged.append(right[j])
j += 1
merged.extend(left[i:])
merged.extend(right[j:])
return merged
# Example Usage
if __name__ == "__main__":
movies = [
Movie(101, "Inception", "Sci-Fi", 8.8, 2010),
Movie(102, "The Dark Knight", "Action", 9.0, 2008),
Movie(103, "Interstellar", "Sci-Fi", 8.6, 2014),
Movie(104, "The Prestige", "Drama", 8.5, 2006),
]
# Sort movies by rating
sorted_by_rating = merge_sort_movies(movies, key="rating")
print("Movies sorted by rating:")
for movie in sorted_by_rating:
print(movie)
# Sort movies by release year
sorted_by_year = merge_sort_movies(movies, key="release_year")

12.5_8.py | 12.6_7.py | 12.5_6.py | 12.5_5.py | 12.5_4.py

sers > nishithamarepally > Desktop > 12.5_6.py > merge_sort_movies

```
72
73      # Sort movies by release year
74      sorted_by_year = merge_sort_movies(movies, key="release_year")
75      print("\nMovies sorted by release year:")
76      for movie in sorted_by_year:
77          print(movie)
78
79      # Search for a movie by ID
80      target_id = 103
81      result = binary_search_movies(sorted_by_year, target_id)
82      print(f"\nSearch result for movie ID {target_id}:")
83      print(result)
```

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Python

```
nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.5_6.py
Movies sorted by rating:
Movie(104, The Prestige, Drama, 8.5, 2006)
Movie(103, Interstellar, Sci-Fi, 8.6, 2014)
Movie(101, Inception, Sci-Fi, 8.8, 2010)
Movie(102, The Dark Knight, Action, 9.0, 2008)

Movies sorted by release year:
Movie(104, The Prestige, Drama, 8.5, 2006)
Movie(102, The Dark Knight, Action, 9.0, 2008)
Movie(101, Inception, Sci-Fi, 8.8, 2010)
Movie(103, Interstellar, Sci-Fi, 8.6, 2014)

Search result for movie ID 103:
Movie(103, Interstellar, Sci-Fi, 8.6, 2014)
nishithamarepally@nishithas-MacBook-Air ~ %
```

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.

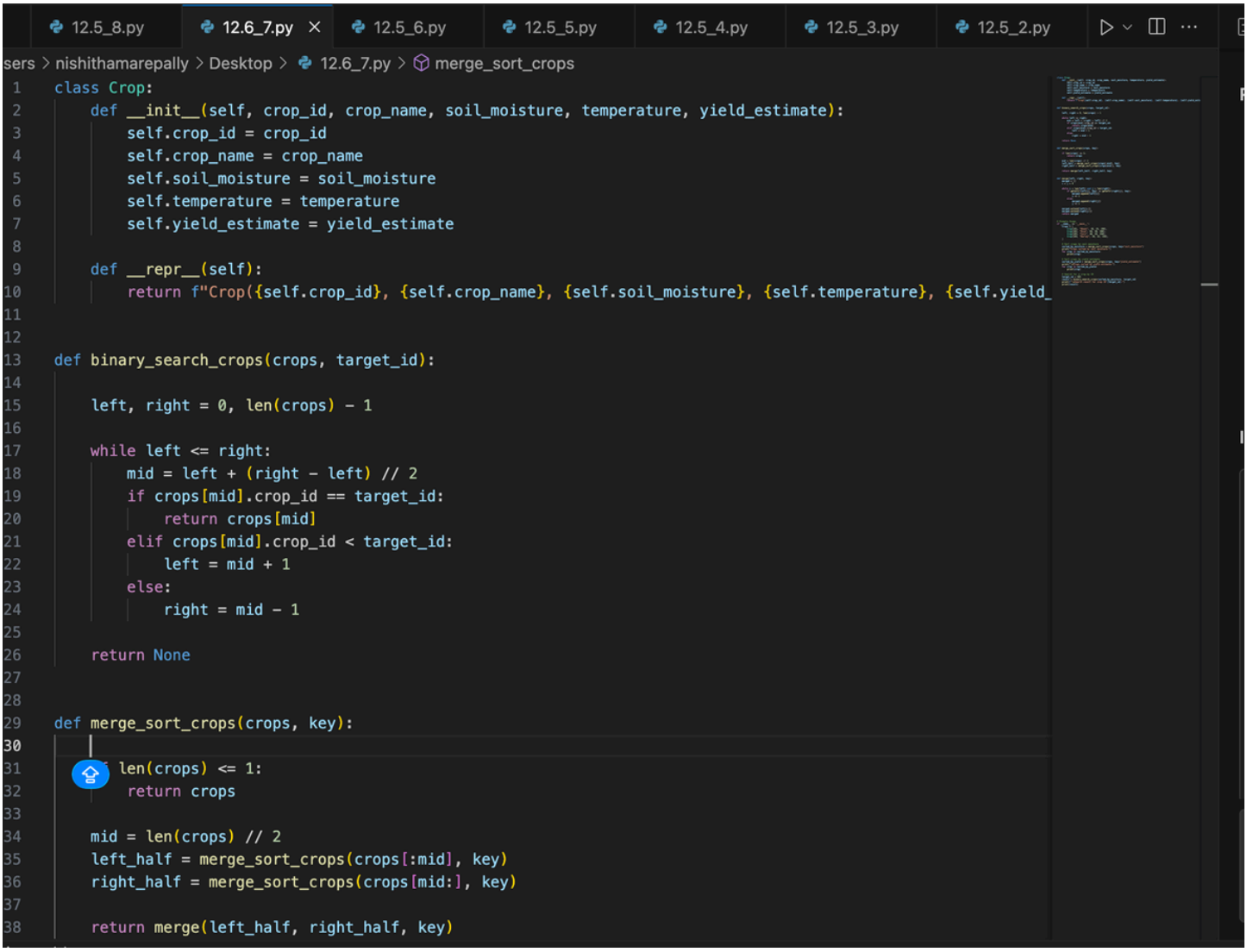
Student Task

• Use AI-assisted reasoning to select algorithms.

• Justify algorithm suitability.

• Implement searching and sorting in Python.

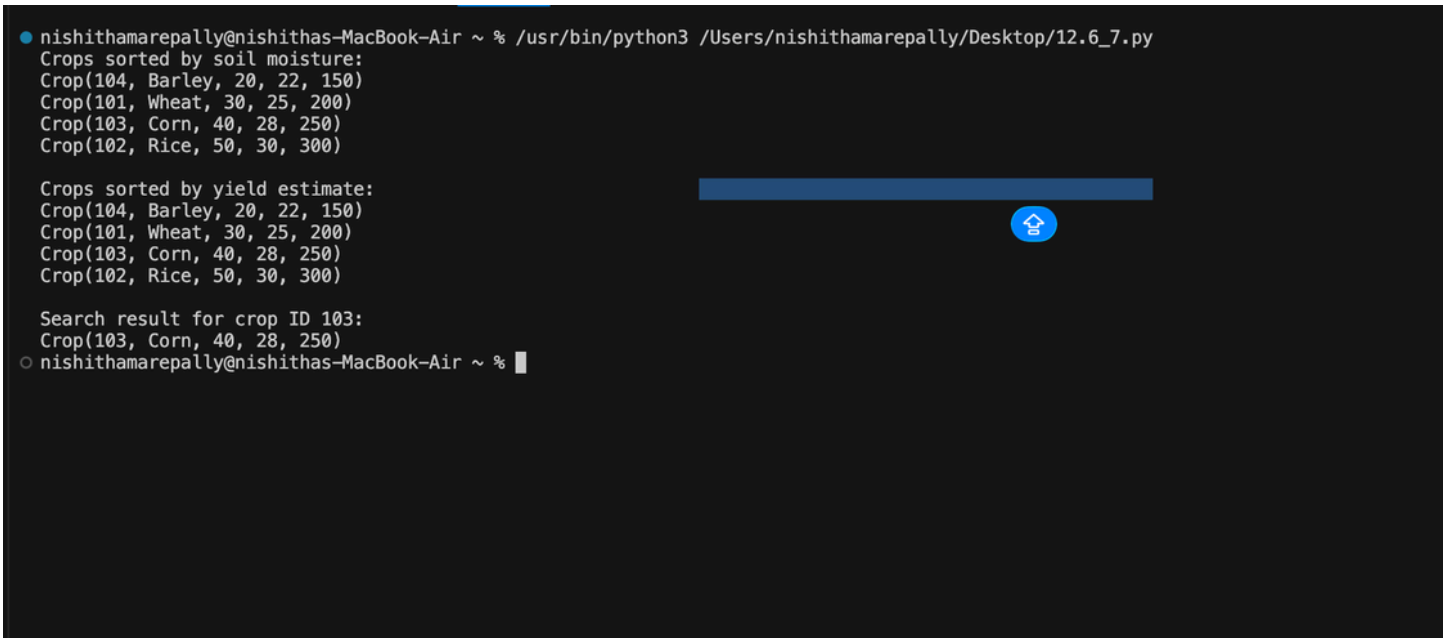
CODE:



```
12.5_8.py  12.6_7.py  12.5_6.py  12.5_5.py  12.5_4.py  12.5_3.py  12.5_2.py
Users > nishithamarepally > Desktop > 12.6_7.py > merge_sort_crops
        return merge(left_half, right_half, key)


def merge(left, right, key):
    merged = []
    i = j = 0

    while i < len(left) and j < len(right):
        if getattr(left[i], key) <= getattr(right[j], key):
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1

    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged


# Example Usage
if __name__ == "__main__":
    crops = [
        Crop(101, "Wheat", 30, 25, 200),
        Crop(102, "Rice", 50, 30, 300),
        Crop(103, "Corn", 40, 28, 250),
        Crop(104, "Barley", 20, 22, 150),
    ]

    # Sort crops by soil moisture
    sorted_by_moisture = merge_sort_crops(crops, key="soil_moisture")
    print("Crops sorted by soil moisture:")
    for crop in sorted_by_moisture:
        print(crop)

```

```
12.5_8.py  12.6_7.py  12.5_6.py  12.5_5.py  12.5_4.py  12.5_3.py  12.5_2.py
sers > nishithamarepally > Desktop > 12.6_7.py > merge_sort_crops
    sorted_by_moisture = merge_sort_crops(crops, key="soil_moisture")
    print("Crops sorted by soil moisture:")
    for crop in sorted_by_moisture:
        print(crop)

    # Sort crops by yield estimate
    sorted_by_yield = merge_sort_crops(crops, key="yield_estimate")
    print("\nCrops sorted by yield estimate:")
    for crop in sorted_by_yield:
        print(crop)

    # Search for a crop by ID
    target_id = 103
    result = binary_search_crops(sorted_by_moisture, target_id)
    print(f"\nSearch result for crop ID {target_id}:")
    print(result)
```

OUTPUT:



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.

CODE:

```python
class Flight:
    def __init__(self, flight_id, airline_name, departure_time, arrival_time, status):
        self.flight_id = flight_id
        self.airline_name = airline_name
        self.departure_time = departure_time
        self.arrival_time = arrival_time
        self.status = status

    def __repr__(self):
        return f"Flight({self.flight_id}, {self.airline_name}, {self.departure_time}, {self.arrival_time}, {se


def binary_search_flights(flights, target_id):

    left, right = 0, len(flights) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if flights[mid].flight_id == target_id:
            return flights[mid]
        elif flights[mid].flight_id < target_id:
            left = mid + 1
        else:
            right = mid - 1

    return None


def merge_sort_flights(flights, key):

    if len(flights) <= 1:
        return flights

    mid = len(flights) // 2
    left_half = merge_sort_flights(flights[:mid], key)
    right_half = merge_sort_flights(flights[mid:], key)

    return merge(left_half, right_half, key)
```

12.5_8.py ● 12.6_7.py 12.5_6.py 12.5_5.py 12.5_4.py 12.5_3.py 12.5_2.py

Users > nishithamarepally > Desktop > 12.5_8.py > ...

```
29  def merge_sort_flights(flights, key):
38      return merge(left_half, right_half, key)
39
40
41  def merge(left, right, key):
42      merged = []
43      i = j = 0
44
45      while i < len(left) and j < len(right):
46          if getattr(left[i], key) <= getattr(right[j], key):
47              merged.append(left[i])
48              i += 1
49          else:
50              merged.append(right[j])
51              j += 1
52
53      merged.extend(left[i:])
54      merged.extend(right[j:])
55      return merged
56
57
58  # Example Usage
59  if __name__ == "__main__":
60      flights = [
61          Flight(101, "Airline A", "08:00", "10:00", "On Time"),
62          Flight(102, "Airline B", "09:30", "11:30", "Delayed"),
63          Flight(103, "Airline C", "07:45", "09:45", "On Time"),
64          Flight(104, "Airline D", "10:15", "12:15", "Cancelled"),
65      ]
66
67      # Sort flights by departure time
68      sorted_by_departure = merge_sort_flights(flights, key="departure_time")
69      print("Flights sorted by departure time:")
70      for flight in sorted_by_departure:
71          print(flight)
72
73      # Sort flights by arrival time
74      sorted_by_arrival = merge_sort_flights(flights, key="arrival_time")
```

Users > nishithamarepally > Desktop > 12.5_8.py > ...

```
73      # Sort flights by arrival time
74      sorted_by_arrival = merge_sort_flights(flights, key="arrival_time")
75      print("\nFlights sorted by arrival time:")
76      for flight in sorted_by_arrival:
77          print(flight)
78
79      # Search for a flight by ID
80      target_id = 103
81      result = binary_search_flights(sorted_by_departure, target_id)
82      print(f"\nSearch result for flight ID {target_id}:")
83      print(result)
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
Python
nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.5_8.py
Flights sorted by departure time:
Flight(103, Airline C, 07:45, 09:45, On Time)
Flight(101, Airline A, 08:00, 10:00, On Time)
Flight(102, Airline B, 09:30, 11:30, Delayed)
Flight(104, Airline D, 10:15, 12:15, Cancelled)
Flights sorted by arrival time:
Flight(103, Airline C, 07:45, 09:45, On Time)
Flight(101, Airline A, 08:00, 10:00, On Time)
Flight(102, Airline B, 09:30, 11:30, Delayed)
Flight(104, Airline D, 10:15, 12:15, Cancelled)
Search result for flight ID 103:
None
nishithamarepally@nishithas-MacBook-Air ~ %