

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

```
1  def merge_sort(arr):
2
3      if len(arr) <= 1:
4          return arr
5
6      # Split the array into two halves
7      mid = len(arr) // 2
8      left_half = merge_sort(arr[:mid])
9      right_half = merge_sort(arr[mid:])
10
11     # Merge the sorted halves
12     return merge(left_half, right_half)
13
14
15 def merge(left, right):
16
17     merged = []
18     i = j = 0
19
20     # Compare elements from both lists and append the smallest
21     while i < len(left) and j < len(right):
22         if left[i] < right[j]:
23             merged.append(left[i])
24             i += 1
25         else:
26             merged.append(right[j])
27             j += 1
28
```

```

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 ~ %

```

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:

```
y | 12.6_7.py | 12.5_6.py | 12.5_5.py | 12.5_4.py | 12.5_3.py | 12.5_2.py • | 12.5_1
Users > nishithamarepally > 12.5_2.py > binary_search
1 def binary_search(arr, target):
2
3     left, right = 0, len(arr) - 1
4
5     while left <= right:
6         mid = left + (right - left) // 2 # Avoids overflow for large indices
7         if arr[mid] == target:
8             return mid
9         elif arr[mid] < target:
10            left = mid + 1
11        else:
12            right = mid - 1
13
14    return -1
15
16
17 # Test cases to verify the implementation
18 if __name__ == "__main__":
19     sorted_array = [1, 3, 5, 7, 9, 11, 13, 15]
20     target = 7
21     print(f"Searching for {target} in {sorted_array}")
22     result = binary_search(sorted_array, target)
23     print(f"Index of {target}: {result}")
24
25     target = 4
26     print(f"Searching for {target} in {sorted_array}")
27     result = binary_search(sorted_array, target)
28     print(f"Index of {target}: {result}")
```

OUTPUT:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/12.5_2.py
  Searching for 7 in [1, 3, 5, 7, 9, 11, 13, 15]
  Index of 7: 3
  Searching for 4 in [1, 3, 5, 7, 9, 11, 13, 15]
  Index of 4: -1
○ nishithamarepally@nishithas-MacBook-Air ~ %
```



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.

CODE :

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

```
1  class Appointment:
2      def __init__(self, appointment_id, patient_name, doctor_name, appointment_time, consultation_fee):
3          self.appointment_id = appointment_id
4          self.patient_name = patient_name
5          self.doctor_name = doctor_name
6          self.appointment_time = appointment_time
7          self.consultation_fee = consultation_fee
8
9      def __repr__(self):
10         return f"Appointment({self.appointment_id}, {self.patient_name}, {self.doctor_name}, {self.appointment_
11
12
13 def binary_search_appointments(appointments, target_id):
14
15     left, right = 0, len(appointments) - 1
16
17     while left <= right:
18         mid = left + (right - left) // 2
19         if appointments[mid].appointment_id == target_id:
20             return appointments[mid]
21         elif appointments[mid].appointment_id < target_id:
22             left = mid + 1
23         else:
24             right = mid - 1
25
26     return None
27
28
29 def merge_sort_appointments(appointments, key):
30
31     if len(appointments) <= 1:
32         return appointments
33
34     mid = len(appointments) // 2
35     left_half = merge_sort_appointments(appointments[:mid], key)
36     right_half = merge_sort_appointments(appointments[mid:], key)
37
38     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_3.py

Users > nishithamarepally > Desktop > 12.5_3.py > merge_sort_appointments
29 def merge_sort_appointments(appointments, key):
30     return merge(left_half, right_half, key)
31
32
33
34
35
36
37
38     def merge(left, right, key):
39         merged = []
40         i = j = 0
41
42         while i < len(left) and j < len(right):
43             if getattr(left[i], key) <= getattr(right[j], key):
44                 merged.append(left[i])
45                 i += 1
46             else:
47                 merged.append(right[j])
48                 j += 1
49
50         merged.extend(left[i:])
51         merged.extend(right[j:])
52
53     return merged
54
55
56
57
58 # Example Usage
59 if __name__ == "__main__":
60     appointments = [
61         Appointment(101, "Alice", "Dr. Smith", "10:00 AM", 100),
62         Appointment(102, "Bob", "Dr. Jones", "11:00 AM", 150),
63         Appointment(103, "Charlie", "Dr. Smith", "09:30 AM", 120),
64         Appointment(104, "Diana", "Dr. Brown", "12:00 PM", 90),
65     ]
66
67     # Sort appointments by time
68     sorted_by_time = merge_sort_appointments(appointments, key="appointment_time")
69     print("Appointments sorted by time:")
70     for appointment in sorted_by_time:
71         print(appointment)
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_3.py

Users > nishithamarepally > Desktop > 12.5_3.py > merge_sort_appointments
70     for appointment in sorted_by_time:
71         print(appointment)
72
73     # Sort appointments by consultation fee
74     sorted_by_fee = merge_sort_appointments(appointments, key="consultation_fee")
75     print("\nAppointments sorted by consultation fee:")
76     for appointment in sorted_by_fee:
77         print(appointment)
78
79     # Search for an appointment by ID
80     target_id = 103
81     result = binary_search_appointments(sorted_by_time, target_id)
82     print(f"\nSearch result for appointment ID {target_id}:")
83     print(result)
```

OUTPUT:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.5_3.py
Appointments sorted by time:
Appointment(103, Charlie, Dr. Smith, 09:30 AM, 120)
Appointment(101, Alice, Dr. Smith, 10:00 AM, 100)
Appointment(102, Bob, Dr. Jones, 11:00 AM, 150)
Appointment(104, Diana, Dr. Brown, 12:00 PM, 90)

Appointments sorted by consultation fee:
Appointment(104, Diana, Dr. Brown, 12:00 PM, 90)
Appointment(101, Alice, Dr. Smith, 10:00 AM, 100)
Appointment(103, Charlie, Dr. Smith, 09:30 AM, 120)
Appointment(102, Bob, Dr. Jones, 11:00 AM, 150)

Search result for appointment ID 103:
None
○ nishithamarepally@nishithas-MacBook-Air ~ %
```

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:

```
me | 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

```
1 class Ticket:
2     def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
3         self.ticket_id = ticket_id
4         self.passenger_name = passenger_name
5         self.train_number = train_number
6         self.seat_number = seat_number
7         self.travel_date = travel_date
8
9     def __repr__(self):
10        return f"Ticket({self.ticket_id}, {self.passenger_name}, {self.train_number}, {self.seat_number}, {self.travel_date})"
11
12
13 def binary_search_tickets(tickets, target_id):
14
15     left, right = 0, len(tickets) - 1
16
17     while left <= right:
18         mid = left + (right - left) // 2
19         if tickets[mid].ticket_id == target_id:
20             return tickets[mid]
21         elif tickets[mid].ticket_id < target_id:
22             left = mid + 1
23         else:
24             right = mid - 1
25
26     return None
27
28
29 def merge_sort_tickets(tickets, key):
30
31     if len(tickets) <= 1:
32         return tickets
33
34     mid = len(tickets) // 2
35     left_half = merge_sort_tickets(tickets[:mid], key)
36     right_half = merge_sort_tickets(tickets[mid:], key)
37
38     return merge(left_half, right_half, key)
```

```
Users > nishithamarepally > Desktop > 12.5_4.py > merge_sort_tickets
29 def merge_sort_tickets(tickets, key):
30     return merge(left_half, right_half, key)
31
32
33 def merge(left, right, key):
34     merged = []
35     i = j = 0
36
37     while i < len(left) and j < len(right):
38         if getattr(left[i], key) <= getattr(right[j], key):
39             merged.append(left[i])
40             i += 1
41         else:
42             merged.append(right[j])
43             j += 1
44
45     merged.extend(left[i:])
46     merged.extend(right[j:])
47
48     return merged
49
50
51 # Example Usage
52 if __name__ == "__main__":
53     tickets = [
54         Ticket(201, "Alice", "Train-101", 15, "2023-12-01"),
55         Ticket(202, "Bob", "Train-102", 10, "2023-11-25"),
56         Ticket(203, "Charlie", "Train-103", 5, "2023-12-05"),
57         Ticket(204, "Diana", "Train-104", 20, "2023-11-20"),
58     ]
59
60
61     # Sort tickets by travel date
62     sorted_by_date = merge_sort_tickets(tickets, key="travel_date")
63     print("Tickets sorted by travel date:")
64     for ticket in sorted_by_date:
65         print(ticket)
66
67
68     # Sort tickets by seat number
69     sorted_by_seat = merge_sort_tickets(tickets, key="seat_number")
70
71
72
73
74
```

```
me 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
70     for ticket in sorted_by_date:
71         print(ticket)
72
73
74     # Sort tickets by seat number
75     sorted_by_seat = merge_sort_tickets(tickets, key="seat_number")
76     print("\nTickets sorted by seat number:")
77     for ticket in sorted_by_seat:
78         print(ticket)
79
80
81     # Search for a ticket by ID
82     target_id = 203
83     result = binary_search_tickets(sorted_by_date, target_id)
84     print(f"\nSearch result for ticket ID {target_id}:")
85     print(result)
```

OUTPUT:

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Python + ×
```

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.5_4.py
Tickets sorted by travel date:
Ticket(204, Diana, Train-104, 20, 2023-11-20)
Ticket(202, Bob, Train-102, 10, 2023-11-25)
Ticket(201, Alice, Train-101, 15, 2023-12-01)
Ticket(203, Charlie, Train-103, 5, 2023-12-05)

Tickets sorted by seat number:
Ticket(203, Charlie, Train-103, 5, 2023-12-05)
Ticket(202, Bob, Train-102, 10, 2023-11-25)
Ticket(201, Alice, Train-101, 15, 2023-12-01)
Ticket(204, Diana, Train-104, 20, 2023-11-20)

Search result for ticket ID 203:
Ticket(203, Charlie, Train-103, 5, 2023-12-05)
○ nishithamarepally@nishithas-MacBook-Air ~ %
```

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

```
1 class Ticket:
2     def __init__(self, ticket_id, passenger_name, train_number, seat_number, travel_date):
3         self.ticket_id = ticket_id
4         self.passenger_name = passenger_name
5         self.train_number = train_number
6         self.seat_number = seat_number
7         self.travel_date = travel_date
8
9     def __repr__(self):
10        return f"Ticket({self.ticket_id}, {self.passenger_name}, {self.train_number}, {self.seat_number}, {self.travel_date})"
11
12
13 def binary_search_tickets(tickets, target_id):
14
15     left, right = 0, len(tickets) - 1
16
17     while left <= right:
18         mid = left + (right - left) // 2
19         if tickets[mid].ticket_id == target_id:
20             return tickets[mid]
21         elif tickets[mid].ticket_id < target_id:
22             left = mid + 1
23         else:
24             right = mid - 1
25
26     return None
27
28
29 def merge_sort_tickets(tickets, key):
30
31     if len(tickets) <= 1:
32         return tickets
33
34     mid = len(tickets) // 2
35     left_half = merge_sort_tickets(tickets[:mid], key)
36     right_half = merge_sort_tickets(tickets[mid:], key)
37
38     return merge(left_half, right_half, key)
```

```
Users > nishithamarepally > Desktop > 12.5_5.py > merge_sort_tickets
29     def merge_sort_tickets(tickets, key):
30         return merge(left_half, right_half, key)
31
32
33
34
35
36
37
38
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
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
```

```
ne 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:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.5_5.py
Tickets sorted by travel date:
Ticket(204, Diana, Train-104, 20, 2023-11-20)
Ticket(202, Bob, Train-102, 10, 2023-11-25)
Ticket(201, Alice, Train-101, 15, 2023-12-01)
Ticket(203, Charlie, Train-103, 5, 2023-12-05)

Tickets sorted by seat number:
Ticket(203, Charlie, Train-103, 5, 2023-12-05)
Ticket(202, Bob, Train-102, 10, 2023-11-25)
Ticket(201, Alice, Train-101, 15, 2023-12-01)
Ticket(204, Diana, Train-104, 20, 2023-11-20)

Search result for ticket ID 203:
Ticket(203, Charlie, Train-103, 5, 2023-12-05)
○ nishithamarepally@nishithas-MacBook-Air ~ %
```

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:

```
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
> nishithamarepally > Desktop > 12.5_6.py > merge_sort_movies
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
users > nishithamarepally > Desktop > 12.5_6.py > merge_sort_movies
/2
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:

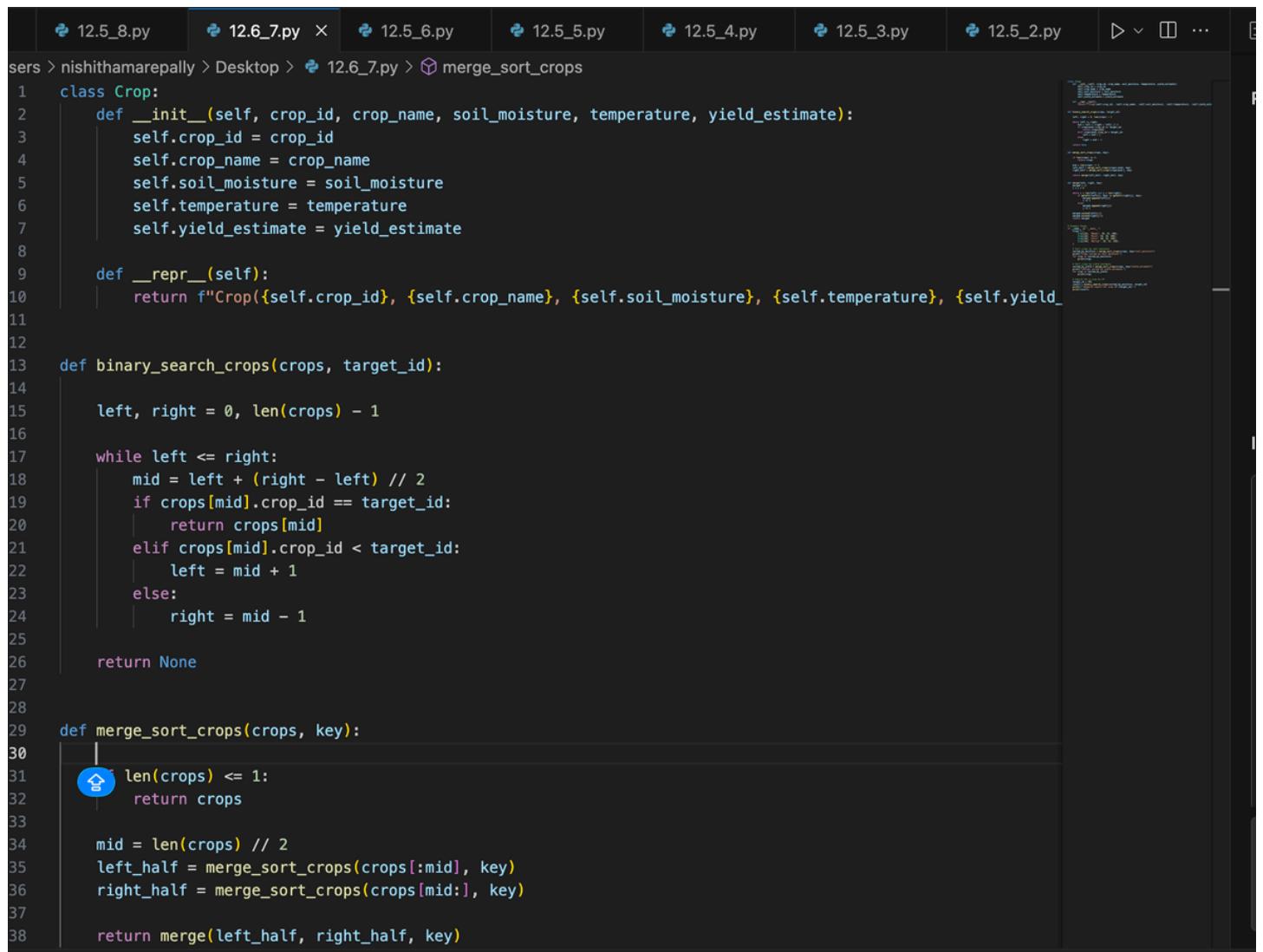
1. Search crop details using crop ID.

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:



The screenshot shows a code editor window with multiple tabs at the top, all labeled '12.5_x.py'. The active tab is '12.6_7.py'. The code in the editor is as follows:

```
1  class Crop:
2      def __init__(self, crop_id, crop_name, soil_moisture, temperature, yield_estimate):
3          self.crop_id = crop_id
4          self.crop_name = crop_name
5          self.soil_moisture = soil_moisture
6          self.temperature = temperature
7          self.yield_estimate = yield_estimate
8
9      def __repr__(self):
10         return f"Crop({self.crop_id}, {self.crop_name}, {self.soil_moisture}, {self.temperature}, {self.yield_
11
12
13 def binary_search_crops(crops, target_id):
14
15     left, right = 0, len(crops) - 1
16
17     while left <= right:
18         mid = left + (right - left) // 2
19         if crops[mid].crop_id == target_id:
20             return crops[mid]
21         elif crops[mid].crop_id < target_id:
22             left = mid + 1
23         else:
24             right = mid - 1
25
26     return None
27
28
29 def merge_sort_crops(crops, key):
30
31     if len(crops) <= 1:
32         return crops
33
34     mid = len(crops) // 2
35     left_half = merge_sort_crops(crops[:mid], key)
36     right_half = merge_sort_crops(crops[mid:], key)
37
38     return merge(left_half, right_half, key)
```

```
12.5_8.py 12.6_7.py X 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
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     crops = [
61         Crop(101, "Wheat", 30, 25, 200),
62         Crop(102, "Rice", 50, 30, 300),
63         Crop(103, "Corn", 40, 28, 250),
64         Crop(104, "Barley", 20, 22, 150),
65     ]
66
67     # Sort crops by soil moisture
68     sorted_by_moisture = merge_sort_crops(crops, key="soil_moisture")
69     print("Crops sorted by soil moisture:")
70     for crop in sorted_by_moisture:
71         print(crop)
72
```

```
12.5_8.py 12.6_7.py X 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
68     sorted_by_moisture = merge_sort_crops(crops, key="soil_moisture")
69     print("Crops sorted by soil moisture:")
70     for crop in sorted_by_moisture:
71         print(crop)
72
73     # Sort crops by yield estimate
74     sorted_by_yield = merge_sort_crops(crops, key="yield_estimate")
75     print("\nCrops sorted by yield estimate:")
76     for crop in sorted_by_yield:
77         print(crop)
78
79     # Search for a crop by ID
80     target_id = 103
81     result = binary_search_crops(sorted_by_moisture, target_id)
82     print(f"\nSearch result for crop ID {target_id}:")
83     print(result)
```

OUTPUT:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/12.6_7.py
Crops sorted by soil moisture:
Crop(104, Barley, 20, 22, 150)
Crop(101, Wheat, 30, 25, 200)
Crop(103, Corn, 40, 28, 250)
Crop(102, Rice, 50, 30, 300)

Crops sorted by yield estimate:
Crop(104, Barley, 20, 22, 150)
Crop(101, Wheat, 30, 25, 200)
Crop(103, Corn, 40, 28, 250)
Crop(102, Rice, 50, 30, 300)

Search result for crop ID 103:
Crop(103, Corn, 40, 28, 250)
○ nishithamarepally@nishithas-MacBook-Air ~ %
```

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:

```
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 > ...
1 class Flight:
2     def __init__(self, flight_id, airline_name, departure_time, arrival_time, status):
3         self.flight_id = flight_id
4         self.airline_name = airline_name
5         self.departure_time = departure_time
6         self.arrival_time = arrival_time
7         self.status = status
8
9     def __repr__(self):
10        return f"Flight({self.flight_id}, {self.airline_name}, {self.departure_time}, {self.arrival_time}, {self.status})"
11
12
13 def binary_search_flights(flights, target_id):
14
15     left, right = 0, len(flights) - 1
16
17     while left <= right:
18         mid = left + (right - left) // 2
19         if flights[mid].flight_id == target_id:
20             return flights[mid]
21         elif flights[mid].flight_id < target_id:
22             left = mid + 1
23         else:
24             right = mid - 1
25
26     return None
27
28
29 def merge_sort_flights(flights, key):
30
31     if len(flights) <= 1:
32         return flights
33
34     mid = len(flights) // 2
35     left_half = merge_sort_flights(flights[:mid], key)
36     right_half = merge_sort_flights(flights[mid:], key)
37
38     return merge(left_half, right_half, key)
39
```

```
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):
30         return merge(left_half, right_half, key)
31
32
33
34
35     def merge(left, right, key):
36         merged = []
37         i = j = 0
38
39         while i < len(left) and j < len(right):
40             if getattr(left[i], key) <= getattr(right[j], key):
41                 merged.append(left[i])
42                 i += 1
43             else:
44                 merged.append(right[j])
45                 j += 1
46
47         merged.extend(left[i:])
48         merged.extend(right[j:])
49
50     return merged
51
52
53
54
55 # Example Usage
56 if __name__ == "__main__":
57     flights = [
58         Flight(101, "Airline A", "08:00", "10:00", "On Time"),
59         Flight(102, "Airline B", "09:30", "11:30", "Delayed"),
60         Flight(103, "Airline C", "07:45", "09:45", "On Time"),
61         Flight(104, "Airline D", "10:15", "12:15", "Cancelled"),
62     ]
63
64
65     # Sort flights by departure time
66     sorted_by_departure = merge_sort_flights(flights, key="departure_time")
67     print("Flights sorted by departure time:")
68     for flight in sorted_by_departure:
69         print(flight)
70
71
72     # Sort flights by arrival time
73     sorted_by_arrival = merge_sort_flights(flights, key="arrival_time")
74
```

```
Users > nishithamarepally > Desktop > 12.5_8.py > ...
72
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 ~ %