AI-ASSISTED CODING

Assignment-12.1

Name: R. Akshitha

HT NO. : 2303A52464

Batch No. :36

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.

ai lab .py

C: > Users > rajam > ai lab .py > merge_sort

```python
def merge_sort(arr):
    """
    Merge sort implementation
    Time Complexity: O(n log n)
    Space Complexity: O(n)
    """
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)


def merge(left, right):
    """
    Merge two sorted arrays
    """
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
```

C: > Users > rajam > ai lab .py > merge_sort

```python
    def merge(left, right):
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                result.append(left[i])
                i += 1
            else:
                result.append(right[j])
                j += 1

        result.extend(left[i:])
        result.extend(right[j:])

        return result


# Test
if __name__ == "__main__":
    arr = [64, 34, 25, 12, 22, 11, 90]
    print(f"Original array: {arr}")
    print(f"Sorted array: {merge_sort(arr)}")
```

OUTPUT:

```
Original array: [64, 34, 25, 12, 22, 11, 90]
Sorted array: [11, 12, 22, 25, 34, 64, 90]
```

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
def binary_search(arr, target):
    """
    Performs a binary search to find the target element in a sorted array.
    Returns the index of the target if found, otherwise returns -1.
    """
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

# Example usage with the 'sorted_list' from your kernel state:
sorted_list = [11, 12, 22, 25, 34, 64, 90] # Using the sorted_list variable from kernel state

target_element_1 = 25
target_element_2 = 100

index_1 = binary_search(sorted_list, target_element_1)
```

```
        else:
            high = mid - 1
    return -1

# Example usage with the 'sorted_list' from your kernel state:
sorted_list = [11, 12, 22, 25, 34, 64, 90] # Using the sorted_list variable from kernel state

target_element_1 = 25
target_element_2 = 100

index_1 = binary_search(sorted_list, target_element_1)
index_2 = binary_search(sorted_list, target_element_2)

if index_1 != -1:
    print(f"Element {target_element_1} found at index {index_1} in the sorted list.")
else:
    print(f"Element {target_element_1} not found in the sorted list.")

if index_2 != -1:
    print(f"Element {target_element_2} not found in the sorted list.")
else:
    print(f"Element {target_element_2} not found in the sorted list.")
```

OUTPUT:

```
Element 25 found at index 3 in the sorted list.
Element 100 not found in the sorted list.
```

Task Description #3 (Real-Time Application – Inventory Management System)

• Scenario: A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:

1. Quickly search for a product by ID or name.

2. Sort products by price or quantity for stock analysis.

• Task:

o Use AI to suggest the most efficient search and sort algorithms for this use case.

o Implement the recommended algorithms in Python.

o Justify the choice based on dataset size, update

frequency, and performance requirements.

• Expected Output:

o A table mapping operation → recommended

algorithm → justification.

o Working Python functions for searching and sorting

the inventory.

```
import collections

# 1. Sample product dataset
products_data = [
    {"product_id": 101, "product_name": "Laptop Pro", "price": 1200.00, "quantity": 15},
    {"product_id": 102, "product_name": "Gaming Mouse", "price": 75.50, "quantity": 50},
    {"product_id": 103, "product_name": "USB-C Hub", "price": 30.00, "quantity": 100},
    {"product_id": 104, "product_name": "External SSD 1TB", "price": 150.00, "quantity": 25
    {"product_id": 105, "product_name": "Wireless Keyboard", "price": 90.00, "quantity": 30
    {"product_id": 106, "product_name": "Monitor 27-inch", "price": 300.00, "quantity": 10}
    {"product_id": 107, "product_name": "Webcam HD", "price": 45.00, "quantity": 40},
    {"product_id": 108, "product_name": "Laptop Sleeve", "price": 20.00, "quantity": 60}
]
print("Sample products_data created successfully.")

# 2. Hash Map for Product ID Search
def build_id_index(products):
    """
    Builds a hash-based index for products using product_id as the key.
    """
    id_index = {}
    for product in products:
        id_index[product["product_id"]] = product
    return id_index
```

```
id_index = build_id_index(products_data)
print("ID index built successfully.")

def search_by_id(id_index, product_id):
    """
    Searches for a product by its ID in the id_index.
    Returns the product dictionary if found, otherwise None.
    """
    return id_index.get(product_id, None)
print("search_by_id function defined.")

# 3. Trie for Product Name Search
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False
        self.product_ids = [] # Store a list of product IDs associated with this word
print("TrieNode class defined.")

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word, product_id):
```

```
    def insert(self, word, product_id):
        node = self.root
        for char in word.lower(): # Convert to lowercase for case-insensitive search
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True
        if product_id not in node.product_ids:
            node.product_ids.append(product_id)

    def _find_all_product_ids_in_subtree(self, node, results):
        if node.is_end_of_word:
            results.extend(node.product_ids)
        for char in node.children:
            self._find_all_product_ids_in_subtree(node.children[char], results)

    def search_prefix(self, prefix):
        node = self.root
        for char in prefix.lower(): # Convert to lowercase for case-insensitive search
            if char not in node.children:
                return [] # No words start with this prefix
            node = node.children[char]

        # Found the node corresponding to the end of the prefix
        results = []
```

```
        results = []
        self._find_all_product_ids_in_subtree(node, results)
        return sorted(list(set(results))) # Return unique and sorted product IDs
print("Trie class defined with insert and search_prefix methods.")

product_trie = Trie()
for product in products_data:
    product_trie.insert(product["product_name"], product["product_id"])
print("Trie populated with product names and IDs.")

# 4. Sorting Functions
def sort_products_by_price(products, reverse=False):
    """
    Sorts a list of product dictionaries by their 'price'.
    """
    return sorted(products, key=lambda product: product['price'], reverse=reverse)
print("sort_products_by_price function defined.")

def sort_products_by_quantity(products, reverse=False):
    """
    Sorts a list of product dictionaries by their 'quantity'.
    """
    return sorted(products, key=lambda product: product['quantity'], reverse=reverse)
print("sort_products_by_quantity function defined.")
```

```python
print("\n--- Testing search_prefix function ---")
prefix_1 = "lap"
product_ids_1 = product_trie.search_prefix(prefix_1)
print(f"Products with prefix '{prefix_1}': {product_ids_1}")
if product_ids_1:
    print("Details:")
    for pid in product_ids_1:
        product = search_by_id(id_index, pid)
        if product: print(f"  - {product['product_name']} (ID: {product['product_id']})")

prefix_4 = "xyz"
product_ids_4 = product_trie.search_prefix(prefix_4)
print(f"Products with prefix '{prefix_4}': {product_ids_4}")

# Testing Sorting by Price
print("\n--- Products sorted by price (ascending) ---")
sorted_by_price_asc = sort_products_by_price(products_data)
for product in sorted_by_price_asc:
    print(f"Product: {product['product_name']}, Price: ${product['price']:.2f}")

# Testing Sorting by Quantity
print("\n--- Products sorted by quantity (descending) ---")
sorted_by_quantity_desc = sort_products_by_quantity(products_data, reverse=True)
for product in sorted_by_quantity_desc:
    print(f"Product: {product['product_name']}, Quantity: {product['quantity']}")
```

OUTPUT:

```
Sample products_data created successfully.
ID index built successfully.
search_by_id function defined.
TrieNode class defined.
Trie class defined with insert and search_prefix methods.
Trie populated with product names and IDs.
sort_products_by_price function defined.
sort_products_by_quantity function defined.

--- Testing search_by_id function ---
Found product by ID 101: Laptop Pro
Product with ID 999 not found.

--- Testing search_prefix function ---
Products with prefix 'lap': [101, 108]
Details:
  - Laptop Pro (ID: 101)
  - Laptop Sleeve (ID: 108)
Products with prefix 'xyz': []

--- Products sorted by price (ascending) ---
Product: Laptop Sleeve, Price: $20.00
Product: USB-C Hub, Price: $30.00
Product: Webcam HD, Price: $45.00
Product: Gaming Mouse, Price: $75.50
Product: Wireless Keyboard, Price: $90.00
Product: External SSD 1TB, Price: $150.00
Product: Monitor 27 inch, Price: $300.00
```

```
--- Products sorted by price (ascending) ---
Product: Laptop Sleeve, Price: $20.00
Product: USB-C Hub, Price: $30.00
Product: Webcam HD, Price: $45.00
Product: Gaming Mouse, Price: $75.50
Product: Wireless Keyboard, Price: $90.00
Product: External SSD 1TB, Price: $150.00
Product: Monitor 27-inch, Price: $300.00
Product: Laptop Pro, Price: $1200.00

--- Products sorted by quantity (descending) ---
Product: USB-C Hub, Quantity: 100
Product: Laptop Sleeve, Quantity: 60
Product: Gaming Mouse, Quantity: 50
Product: Webcam HD, Quantity: 40
Product: Wireless Keyboard, Quantity: 30
Product: External SSD 1TB, Quantity: 25
Product: Laptop Pro, Quantity: 15
Product: Monitor 27-inch, Quantity: 10
```

Task description #4: Smart Hospital Patient Management System

A hospital maintains records of thousands of patients with details such as patient ID, name, severity level, admission date, and bill amount. Doctors and staff need to:

1. Quickly search patient records using patient ID or name.
2. Sort patients based on severity level or bill amount for prioritization and billing.

Student Task

• Use AI to recommend suitable searching and sorting algorithms.

• Justify the selected algorithms in terms of efficiency and suitability.

• Implement the recommended algorithms in Python.

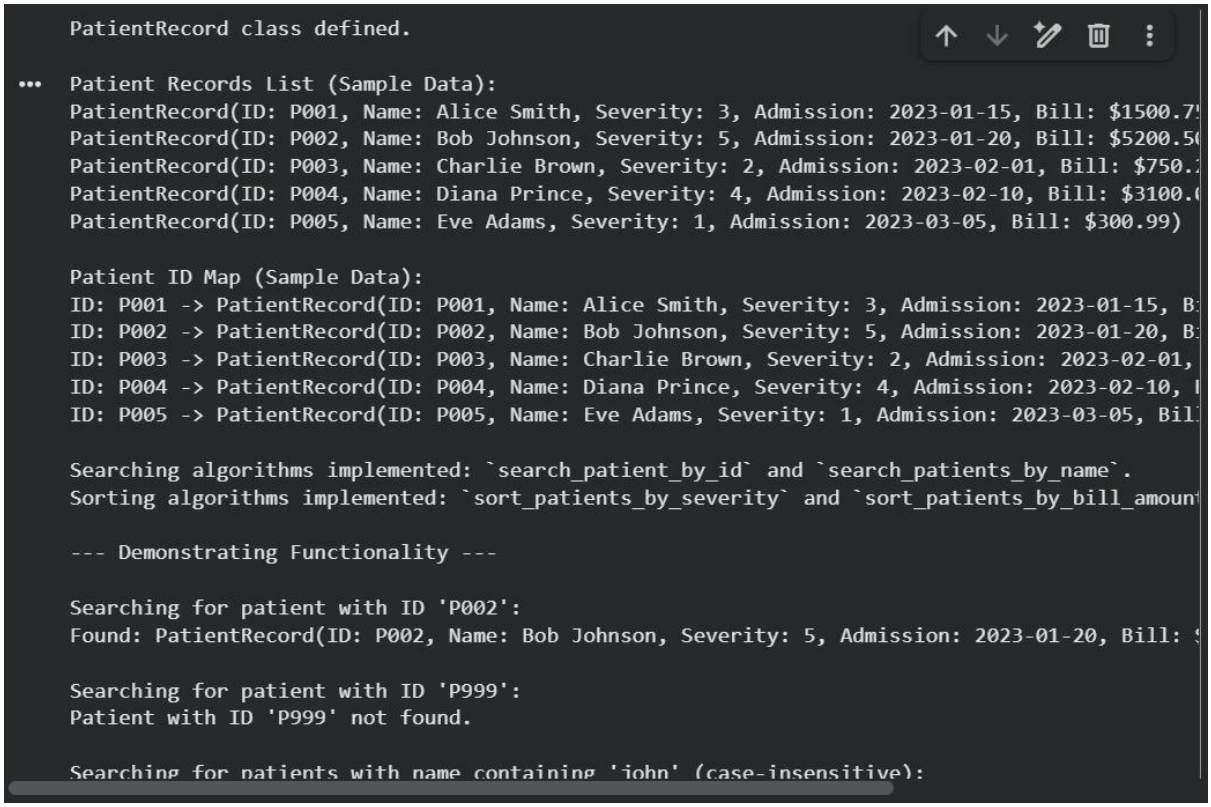
```python
import datetime

# Define Patient Data Structure
class PatientRecord:
    def __init__(self, patient_id, name, severity, admission_date, bill_amount):
        self.patient_id = patient_id
        self.name = name
        self.severity = severity
        self.admission_date = admission_date
        self.bill_amount = bill_amount

    def __repr__(self):
        return (f"PatientRecord(ID: {self.patient_id}, Name: {self.name}, "
                f"Severity: {self.severity}, Admission: {self.admission_date}, "
                f"Bill: ${self.bill_amount:.2f})")

print("PatientRecord class defined.")

# Implement Data Structure and Populate with Sample Data
patient_records = []
patient_id_map = {}

patient1 = PatientRecord("P001", "Alice Smith", 3, datetime.date(2023, 1, 15), 1500.75)
patient2 = PatientRecord("P002", "Bob Johnson", 5, datetime.date(2023, 1, 20), 5200.50)
patient3 = PatientRecord("P003", "Charlie Brown", 2, datetime.date(2023, 2, 1), 750.20)
```

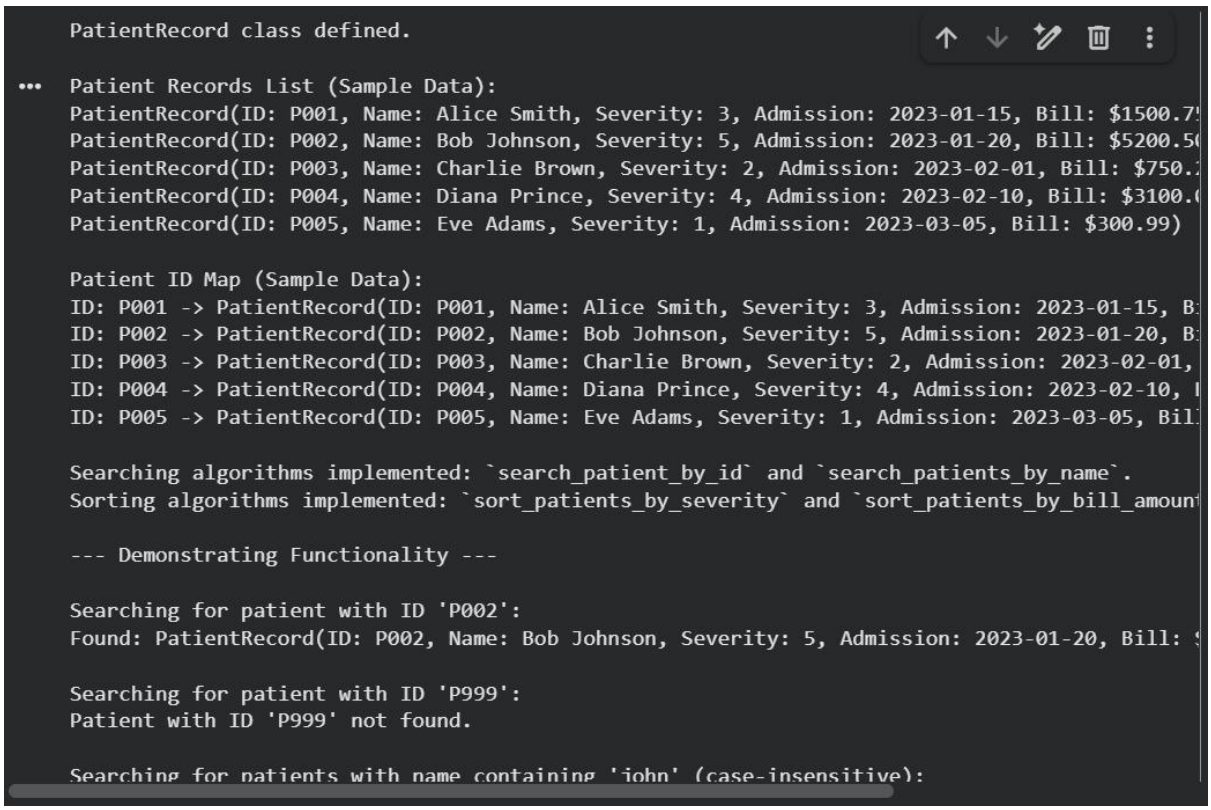
```python
patient_records.append(patient1)
patient_records.append(patient2)
patient_records.append(patient3)
patient_records.append(patient4)
patient_records.append(patient5)

patient_id_map[patient1.patient_id] = patient1
patient_id_map[patient2.patient_id] = patient2
patient_id_map[patient3.patient_id] = patient3
patient_id_map[patient4.patient_id] = patient4
patient_id_map[patient5.patient_id] = patient5

print("\nPatient Records List (Sample Data):")
for record in patient_records:
    print(record)

print("\nPatient ID Map (Sample Data):")
for patient_id, record in patient_id_map.items():
    print(f"ID: {patient_id} -> {record}")

# Implement Searching Algorithms
def search_patient_by_id(patient_id_map, target_id):
    """
    Searches for a patient record by ID using the patient_id_map (hash map).
    Returns the PatientRecord object if found, otherwise None.
```

```
    return patient_id_map.get(target_id)

def search_patients_by_name(patient_records, target_name):
    """
    Searches for patient records by name (case-insensitive) using a linear search.
    Returns a list of matching PatientRecord objects.
    """
    matching_patients = []
    target_name_lower = target_name.lower()
    for patient in patient_records:
        if target_name_lower in patient.name.lower():
            matching_patients.append(patient)
    return matching_patients

print("\nSearching algorithms implemented: `search_patient_by_id` and `search_patients_by_n

# Implement Sorting Algorithms
def sort_patients_by_severity(patient_records):
    """
    Sorts patient records by severity in ascending order.
    """
    return sorted(patient_records, key=lambda patient: patient.severity)

def sort_patients_by_bill_amount(patient_records):
    """
```

```
if found_patients_name:
    for patient in found_patients_name:
        print(f"Found: {patient}")
else:
    print(f"No patients found with name containing '{search_name}'.")

search_name_partial = "li"
found_patients_partial = search_patients_by_name(patient_records, search_name_partial)
print(f"\nSearching for patients with name containing '{search_name_partial}' (case-insensi
if found_patients_partial:
    for patient in found_patients_partial:
        print(f"Found: {patient}")
else:
    print(f"No patients found with name containing '{search_name_partial}'.")

# Test Sort by Severity
sorted_by_severity = sort_patients_by_severity(patient_records)
print("\nPatients sorted by Severity (Ascending):")
for patient in sorted_by_severity:
    print(patient)

# Test Sort by Bill Amount
sorted_by_bill = sort_patients_by_bill_amount(patient_records)
print("\nPatients sorted by Bill Amount (Descending):")
for patient in sorted_by_bill:
    print(patient)
```

OUTPUT:





Task Description #5: University Examination Result Processing System

A university processes examination results for thousands of students

containing roll number, name, subject, and marks. The system must:

1. Search student results using roll number.

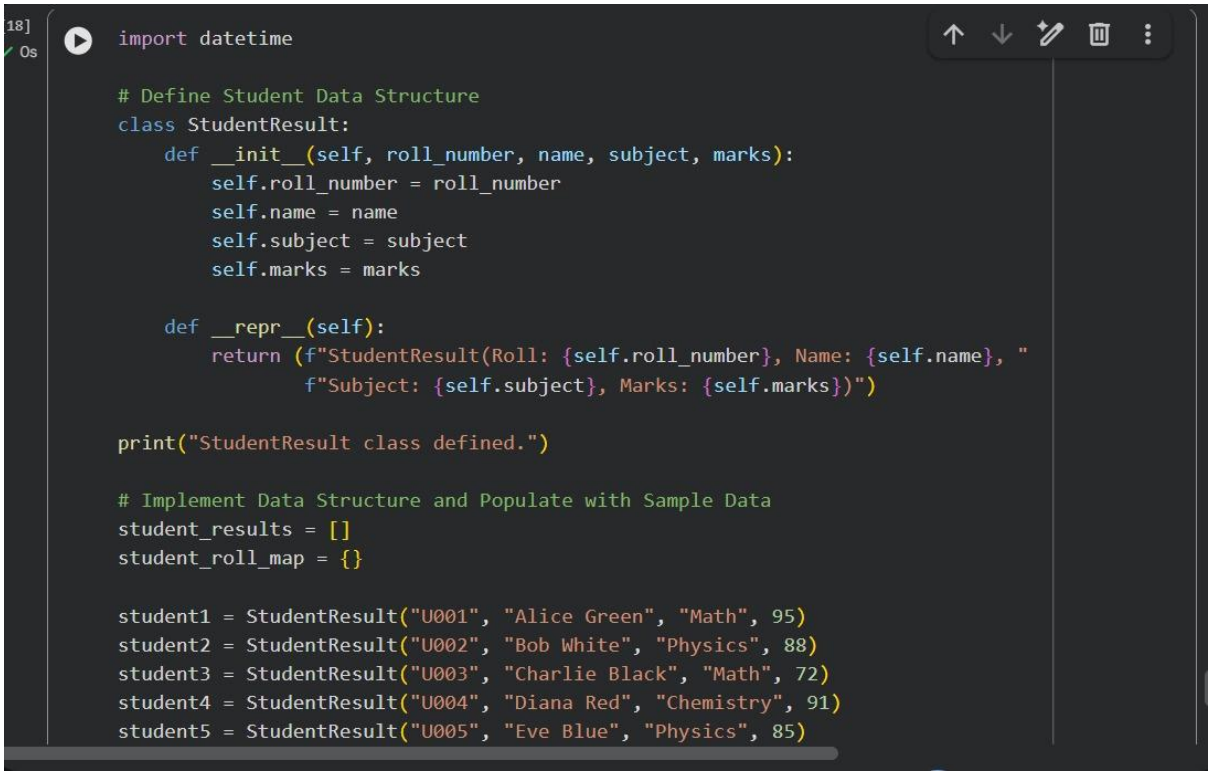
2. Sort students based on marks to generate rank lists.

Student Task

• Identify efficient searching and sorting algorithms using AI assistance.

• Justify the choice of algorithms.

• Implement the algorithms in Python.



```python
student6 = StudentResult("U006", "Frank Grey", "Math", 95)

student_results.extend([student1, student2, student3, student4, student5, student6])

for student in student_results:
    student_roll_map[student.roll_number] = student

print("\nStudent Results List (Sample Data):")
for result in student_results:
    print(result)

print("\nStudent Roll Number Map (Sample Data):")
for roll, result in student_roll_map.items():
    print(f"Roll: {roll} -> {result}")

# Implement Searching Algorithm for Roll Number
def search_student_by_roll_number(roll_map, target_roll):
    """
    Searches for a student result by roll number using the roll_map (hash map).
    Returns the StudentResult object if found, otherwise None.
    Justification: A hash map provides O(1) average time complexity for lookups,
    making it extremely efficient for unique identifiers like roll numbers.
    """
    return roll_map.get(target_roll)

print("\nSearching algorithm implemented: `search_student_by_roll_number`.")
```


```python
student6 = StudentResult("U006", "Frank Grey", "Math", 95)

student_results.extend([student1, student2, student3, student4, student5, student6])

for student in student_results:
    student_roll_map[student.roll_number] = student

print("\nStudent Results List (Sample Data):")
for result in student_results:
    print(result)

print("\nStudent Roll Number Map (Sample Data):")
for roll, result in student_roll_map.items():
    print(f"Roll: {roll} -> {result}")

# Implement Searching Algorithm for Roll Number
def search_student_by_roll_number(roll_map, target_roll):
    """
    Searches for a student result by roll number using the roll_map (hash map).
    Returns the StudentResult object if found, otherwise None.
    Justification: A hash map provides O(1) average time complexity for lookups,
    making it extremely efficient for unique identifiers like roll numbers.
    """
    return roll_map.get(target_roll)

print("\nSearching algorithm implemented: `search_student_by_roll_number`.")
```

```
print("Sorting algorithm implemented: `sort_students_by_marks`.")

# Demonstrate and Test
print("\n--- Demonstrating Functionality ---")

# Test Search by Roll Number
search_roll = "U004"
found_student_roll = search_student_by_roll_number(student_roll_map, search_roll)
print(f"\nSearching for student with Roll Number '{search_roll}':")
if found_student_roll:
    print(f"Found: {found_student_roll}")
else:
    print(f"Student with Roll Number '{search_roll}' not found.")

search_roll_not_found = "U999"
found_student_roll_not_found = search_student_by_roll_number(student_roll_map, search_roll_not_found)
print(f"\nSearching for student with Roll Number '{search_roll_not_found}':")
if found_student_roll_not_found:
    print(f"Found: {found_student_roll_not_found}")
else:
    print(f"Student with Roll Number '{search_roll_not_found}' not found.")

# Test Sort by Marks (Rank List)
sorted_by_marks = sort_students_by_marks(student_results)
print("\nStudents sorted by Marks (Rank List - Descending):")
for i, student in enumerate(sorted by marks):
```

OUTPUT:

```
StudentResult class defined.

Student Results List (Sample Data):
StudentResult(Roll: U001, Name: Alice Green, Subject: Math, Marks: 95)
StudentResult(Roll: U002, Name: Bob White, Subject: Physics, Marks: 88)
StudentResult(Roll: U003, Name: Charlie Black, Subject: Math, Marks: 72)
StudentResult(Roll: U004, Name: Diana Red, Subject: Chemistry, Marks: 91)
StudentResult(Roll: U005, Name: Eve Blue, Subject: Physics, Marks: 85)
StudentResult(Roll: U006, Name: Frank Grey, Subject: Math, Marks: 95)

Student Roll Number Map (Sample Data):
Roll: U001 -> StudentResult(Roll: U001, Name: Alice Green, Subject: Math, Marks: 95)
Roll: U002 -> StudentResult(Roll: U002, Name: Bob White, Subject: Physics, Marks: 88)
Roll: U003 -> StudentResult(Roll: U003, Name: Charlie Black, Subject: Math, Marks: 72)
Roll: U004 -> StudentResult(Roll: U004, Name: Diana Red, Subject: Chemistry, Marks: 91)
Roll: U005 -> StudentResult(Roll: U005, Name: Eve Blue, Subject: Physics, Marks: 85)
Roll: U006 -> StudentResult(Roll: U006, Name: Frank Grey, Subject: Math, Marks: 95)

Searching algorithm implemented: `search_student_by_roll_number`.
Sorting algorithm implemented: `sort_students_by_marks`.

--- Demonstrating Functionality ---

Searching for student with Roll Number 'U004':
Found: StudentResult(Roll: U004, Name: Diana Red, Subject: Chemistry, Marks: 91)

Searching for student with Roll Number 'U999':
Student with Roll Number 'U999' not found.
```

```
Searching algorithm implemented: `search_student_by_roll_number`.
Sorting algorithm implemented: `sort_students_by_marks`.

--- Demonstrating Functionality ---

Searching for student with Roll Number 'U004':
Found: StudentResult(Roll: U004, Name: Diana Red, Subject: Chemistry, Marks: 91)

Searching for student with Roll Number 'U999':
Student with Roll Number 'U999' not found.

Students sorted by Marks (Rank List - Descending):
Rank 1: StudentResult(Roll: U001, Name: Alice Green, Subject: Math, Marks: 95)
Rank 2: StudentResult(Roll: U006, Name: Frank Grey, Subject: Math, Marks: 95)
Rank 3: StudentResult(Roll: U004, Name: Diana Red, Subject: Chemistry, Marks: 91)
Rank 4: StudentResult(Roll: U002, Name: Bob White, Subject: Physics, Marks: 88)
Rank 5: StudentResult(Roll: U005, Name: Eve Blue, Subject: Physics, Marks: 85)
Rank 6: StudentResult(Roll: U003, Name: Charlie Black, Subject: Math, Marks: 72)
```

Task Description #6: Online Food Delivery Platform

An online food delivery application stores thousands of orders with order ID, restaurant name, delivery time, price, and order status. The platform needs to:

1. Quickly find an order using order ID.

2. Sort orders based on delivery time or price.

Student Task

• Use AI to suggest optimized algorithms.

• Justify the algorithm selection.

• Implement searching and sorting modules in Python.

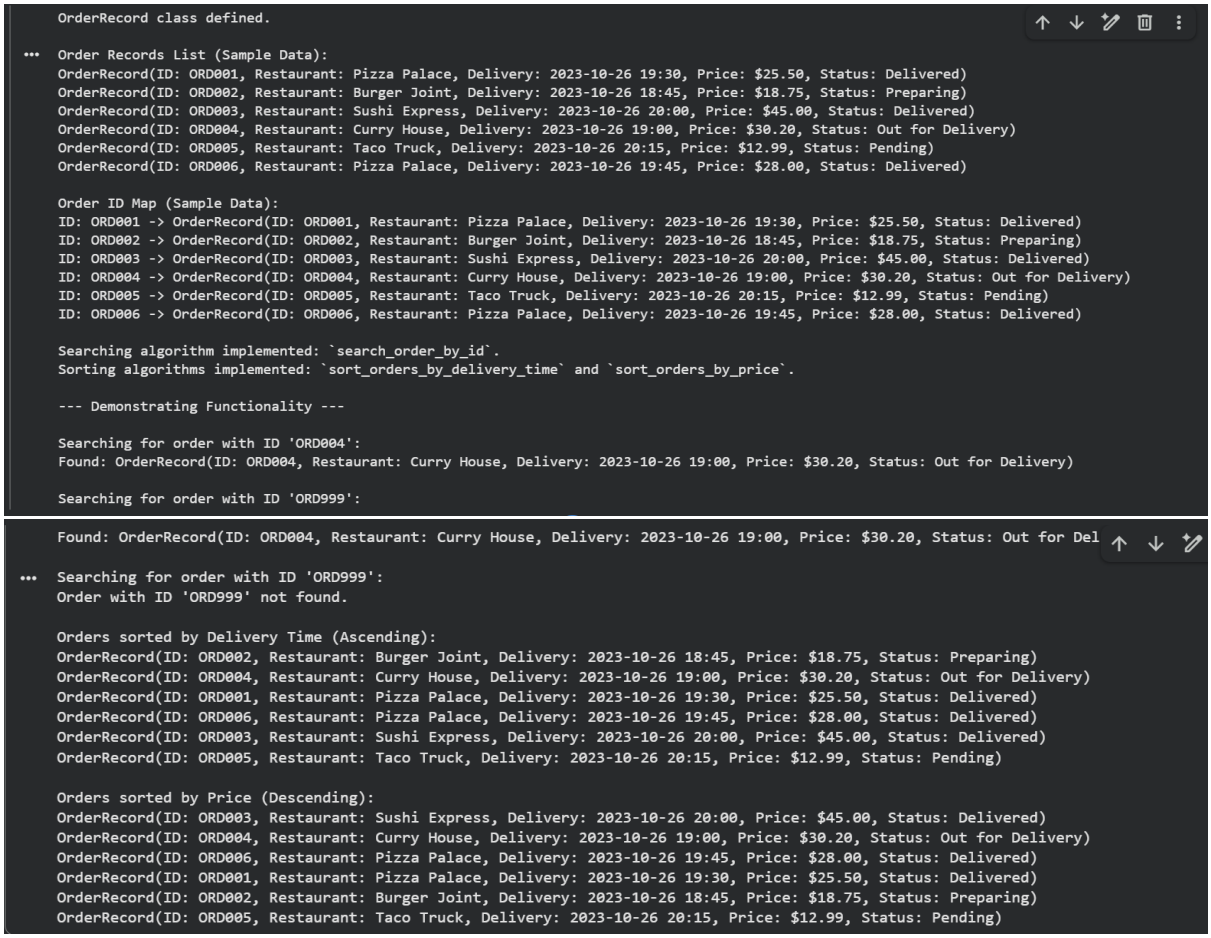
```python
import datetime

# Define Order Data Structure
class OrderRecord:
    def __init__(self, order_id, restaurant_name, delivery_time, price, order_status):
        self.order_id = order_id
        self.restaurant_name = restaurant_name
        self.delivery_time = delivery_time # datetime.datetime object
        self.price = price
        self.order_status = order_status

    def __repr__(self):
        return (f"OrderRecord(ID: {self.order_id}, Restaurant: {self.restaurant_name}, "
                f"Delivery: {self.delivery_time.strftime('%Y-%m-%d %H:%M')}, "
                f"Price: ${self.price:.2f}, Status: {self.order_status})")

print("OrderRecord class defined.")

# Implement Data Structure and Populate with Sample Data
order_records = []
order_id_map = {}
```

```python
order1 = OrderRecord("ORD001", "Pizza Palace", datetime.datetime(2023, 10, 26, 19, 30), 25.50, "Delivered")
order2 = OrderRecord("ORD002", "Burger Joint", datetime.datetime(2023, 10, 26, 18, 45), 18.75, "Preparing")
order3 = OrderRecord("ORD003", "Sushi Express", datetime.datetime(2023, 10, 26, 20, 00), 45.00, "Delivered")
order4 = OrderRecord("ORD004", "Curry House", datetime.datetime(2023, 10, 26, 19, 00), 30.20, "Out for Delivery")
order5 = OrderRecord("ORD005",  OrderRecord: order2                2023, 10, 26, 20, 15), 12.99, "Pending")
order6 = OrderRecord("ORD006",  __main__.OrderRecord instance      e(2023, 10, 26, 19, 45), 28.00, "Delivered")

order_records.extend([order1, order2, order3, order4, order5, order6])

for order in order_records:
    order_id_map[order.order_id] = order

print("\nOrder Records List (Sample Data):")
for record in order_records:
    print(record)

print("\nOrder ID Map (Sample Data):")
for order_id, record in order_id_map.items():
    print(f"ID: {order_id} -> {record}")

# Implement Searching Algorithms
def search_order_by_id(order_id_map, target_id):
```

```python
# Implement Searching Algorithms
def search_order_by_id(order_id_map, target_id):
    """
    Searches for an order record by ID using the order_id_map (hash map).
    Returns the OrderRecord object if found, otherwise None.
    """
    return order_id_map.get(target_id)

print("\nSearching algorithm implemented: `search_order_by_id`.")

# Implement Sorting Algorithms
def sort_orders_by_delivery_time(order_records):
    """
    Sorts order records by delivery time in ascending order.
    """
    return sorted(order_records, key=lambda order: order.delivery_time)

def sort_orders_by_price(order_records):
    """
    Sorts order records by price in descending order.
    """
    return sorted(order_records, key=lambda order: order.price, reverse=True)
```

```python
if found_order_id:
    print(f"Found: {found_order_id}")
else:
    print(f"Order with ID '{search_id}' not found.")

search_id_not_found = "ORD999"
found_order_id_not_found = search_order_by_id(order_id_map, search_id_not_found)
print(f"\nSearching for order with ID '{search_id_not_found}':")
if found_order_id_not_found:
    print(f"Found: {found_order_id_not_found}")
else:
    print(f"Order with ID '{search_id_not_found}' not found.")

# Test Sort by Delivery Time
sorted_by_delivery_time = sort_orders_by_delivery_time(order_records)
print("\nOrders sorted by Delivery Time (Ascending):")
for order in sorted_by_delivery_time:
    print(order)

# Test Sort by Price
sorted_by_price = sort_orders_by_price(order_records)
print("\nOrders sorted by Price (Descending):")
for order in sorted_by_price:
    print(order)
```

OUTPUT:



```
OrderRecord class defined.

Order Records List (Sample Data):
OrderRecord(ID: ORD001, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:30, Price: $25.50, Status: Delivered)
OrderRecord(ID: ORD002, Restaurant: Burger Joint, Delivery: 2023-10-26 18:45, Price: $18.75, Status: Preparing)
OrderRecord(ID: ORD003, Restaurant: Sushi Express, Delivery: 2023-10-26 20:00, Price: $45.00, Status: Delivered)
OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Delivery)
OrderRecord(ID: ORD005, Restaurant: Taco Truck, Delivery: 2023-10-26 20:15, Price: $12.99, Status: Pending)
OrderRecord(ID: ORD006, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:45, Price: $28.00, Status: Delivered)

Order ID Map (Sample Data):
ID: ORD001 -> OrderRecord(ID: ORD001, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:30, Price: $25.50, Status: Delivered)
ID: ORD002 -> OrderRecord(ID: ORD002, Restaurant: Burger Joint, Delivery: 2023-10-26 18:45, Price: $18.75, Status: Preparing)
ID: ORD003 -> OrderRecord(ID: ORD003, Restaurant: Sushi Express, Delivery: 2023-10-26 20:00, Price: $45.00, Status: Delivered)
ID: ORD004 -> OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Delivery)
ID: ORD005 -> OrderRecord(ID: ORD005, Restaurant: Taco Truck, Delivery: 2023-10-26 20:15, Price: $12.99, Status: Pending)
ID: ORD006 -> OrderRecord(ID: ORD006, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:45, Price: $28.00, Status: Delivered)

Searching algorithm implemented: `search_order_by_id`.
Sorting algorithms implemented: `sort_orders_by_delivery_time` and `sort_orders_by_price`.

--- Demonstrating Functionality ---

Searching for order with ID 'ORD004':
Found: OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Delivery)

Searching for order with ID 'ORD999':
```



```
Found: OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Del

Searching for order with ID 'ORD999':
Order with ID 'ORD999' not found.

Orders sorted by Delivery Time (Ascending):
OrderRecord(ID: ORD002, Restaurant: Burger Joint, Delivery: 2023-10-26 18:45, Price: $18.75, Status: Preparing)
OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Delivery)
OrderRecord(ID: ORD001, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:30, Price: $25.50, Status: Delivered)
OrderRecord(ID: ORD006, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:45, Price: $28.00, Status: Delivered)
OrderRecord(ID: ORD003, Restaurant: Sushi Express, Delivery: 2023-10-26 20:00, Price: $45.00, Status: Delivered)
OrderRecord(ID: ORD005, Restaurant: Taco Truck, Delivery: 2023-10-26 20:15, Price: $12.99, Status: Pending)

Orders sorted by Price (Descending):
OrderRecord(ID: ORD003, Restaurant: Sushi Express, Delivery: 2023-10-26 20:00, Price: $45.00, Status: Delivered)
OrderRecord(ID: ORD004, Restaurant: Curry House, Delivery: 2023-10-26 19:00, Price: $30.20, Status: Out for Delivery)
OrderRecord(ID: ORD006, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:45, Price: $28.00, Status: Delivered)
OrderRecord(ID: ORD001, Restaurant: Pizza Palace, Delivery: 2023-10-26 19:30, Price: $25.50, Status: Delivered)
OrderRecord(ID: ORD002, Restaurant: Burger Joint, Delivery: 2023-10-26 18:45, Price: $18.75, Status: Preparing)
OrderRecord(ID: ORD005, Restaurant: Taco Truck, Delivery: 2023-10-26 20:15, Price: $12.99, Status: Pending)
```