AI ASSISTED CODING

ASSIGNMENT- 12.1

Name: B.Harshini

HT No:2303A52242

BT No:36

Task Description #1 (Sorting – Merge Sort Implementation)

• Task: Use AI to generate a Python program that implements the Merge Sort algorithm.

• Instructions:

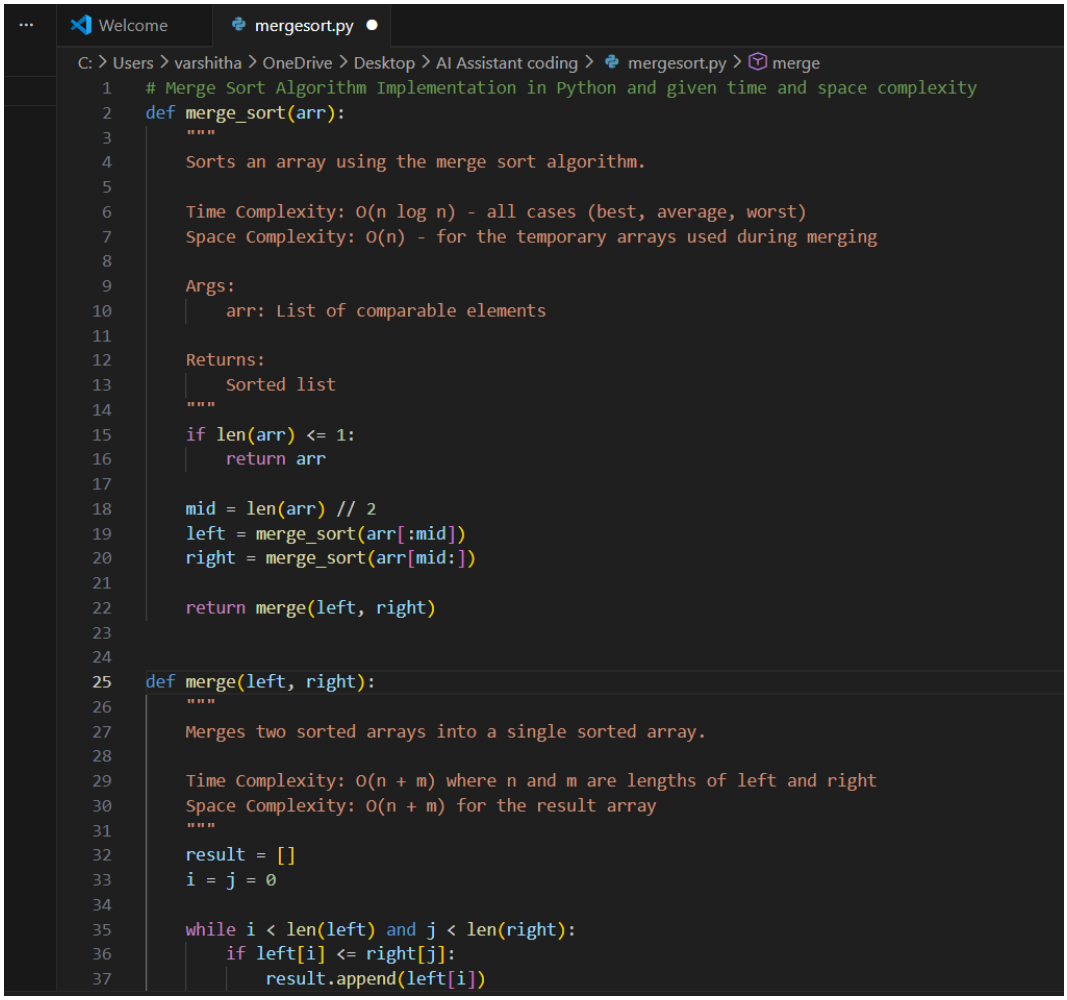
o Prompt AI to create a function merge_sort(arr) that sorts a list in ascending order.

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.



```
25  def merge(left, right):
        while i < len(left) and j < len(right):
36          if left[i] <= right[j]:
37              result.append(left[i])
38              i += 1
39          else:
40              result.append(right[j])
41              j += 1
42
43      result.extend(left[i:])
44      result.extend(right[j:])
45      return result
46  # Example usage:
47  if __name__ == "__main__":
48      arr = [38, 27, 43, 3, 9, 82, 10]
49      sorted_arr = merge_sort(arr)
50      print("Sorted array:", sorted_arr)
51
```

OUTPUT:

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.

Welcome | binarysearch.py

C: > Users > varshitha > OneDrive > Desktop > AI Assistant coding > binarysearch.py > ...

```python
#generate a code for binary search in python and Include docstrings explaining best, average, and
#wost-case complexities. Test with various inputs.
def binary_search(arr, target):
    """
    Perform binary search on a sorted array.

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

    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 array.
    Average Case: O(log n) - when the target is not at the middle but still found in the array.
    Worst Case: O(log n) - when the target is not found in the array or is at one of the ends.
    """
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = left + (right - left) // 2

        # 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

    # Target was not found in the array
    return -1
# Test cases
if __name__ == "__main__":
```

```python
    return -1
# Test cases
if __name__ == "__main__":
    # Test with various inputs
    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    target = 5
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")

    target = 1
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")

    target = 10
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")

    target = 11
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")

    target = 0
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")
    target = 6
    print(f"Searching for {target} in {arr}: Index = {binary_search(arr, target)}")
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\varshitha>
PS C:\Users\varshitha> & C:/Users/varshitha/anaconda3/python.exe "c:/Users/varshitha/OneDrive/Desktop/AI Assistant coding/binarysearch.py"
Searching for 5 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = 4
Searching for 1 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = 0
Searching for 10 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = 9
Searching for 11 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = -1
Searching for 0 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = -1
Searching for 6 in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: Index = 5
```

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.

AI.py | AI 7.py | AI 9.py | lab exam.py | lab exam2.py | que2.py | Ass-12.5.py

Ass-12.5.py > ...

```python
## implement a retail store's inventory system contains thousands of products, each with attributes like product I
# name, price, and stock quantity. Store staff need to: Quickly search for a product by ID or name. Sort products
#  price or quantity for stock analysis.mplement the recommended algorithms in Python. Justify the choice based
#  on dataset size, update frequency, and performance requirements.
class Product:
    def __init__(self, product_id, name, price, stock_quantity):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.stock_quantity = stock_quantity
class InventorySystem:
    def __init__(self):
        self.products = []

    def add_product(self, product_id, name, price, stock_quantity):
        product = Product(product_id, name, price, stock_quantity)
        self.products.append(product)

    def search_product_by_id(self, product_id):
        for product in self.products:
            if product.product_id == product_id:
                return product
        return None

    def search_product_by_name(self, name):
        for product in self.products:
            if product.name.lower() == name.lower():
                return product
        return None
```

```python
class InventorySystem:

    def sort_products_by_price(self):
        self.products.sort(key=lambda x: x.price)

    def sort_products_by_stock_quantity(self):
        self.products.sort(key=lambda x: x.stock_quantity)
# Example usage
inventory = InventorySystem()
inventory.add_product("P001", "Laptop", 999.99, 10)
inventory.add_product("P002", "Smartphone", 499.99, 20)
inventory.add_product("P003", "Headphones", 199.99, 15)
print("Search for product ID P001:", inventory.search_product_by_id("P001"))
print("Search for product name 'Smartphone':", inventory.search_product_by_name("Smartphone"))
inventory.sort_products_by_price()
print("Products sorted by price:", inventory.products)
inventory.sort_products_by_stock_quantity()
print("Products sorted by stock quantity:", inventory.products)
#justification for selected algorithm:
# For searching products by ID or name, we can use a linear search since the number of products may not
#  be very large, and it allows us to find the product without needing to sort the data first. The time
#  complexity of linear search is O(n) in the worst case. For sorting products by price or stock quantity,
# we can use the built-in sort method which implements Timsort. Timsort is efficient for real-world data and
#  has a time complexity of O(n log n) in the worst case and O(n) in the best case when the data is already sorted
# . This makes it suitable for our retail store's inventory system where we may often have partially sorted data
#  and need efficient sorting for stock analysis.
```

OUTPUT:

```
PROBLEMS  OUTPUT  TERMINAL  PORTS
DEBUG CONSOLE                TERMINAL                powershell
Filter (e.g. text, !exclude, \...
PS C:\Users\reddy\OneDrive\Desktop\AI> & C:\Users\reddy\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:
/Users/reddy/OneDrive/Desktop/AI/Ass-12.5.py
Search for product ID P001: <__main__.Product object at 0x000001FE01CA6BA0>
Search for product name 'Smartphone': <__main__.Product object at 0x000001FE01F48A50>
Products sorted by price: [<__main__.Product object at 0x000001FE01F48B90>, <__main__.Product object at 0x000
001FE01F48A50>, <__main__.Product object at 0x000001FE01CA6BA0>]
Products sorted by stock quantity: [<__main__.Product object at 0x000001FE01CA6BA0>, <__main__.Product object
 at 0x000001FE01F48B90>, <__main__.Product object at 0x000001FE01F48A50>]
PS C:\Users\reddy\OneDrive\Desktop\AI>
```

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
##implement A hospital maintains records of thousands of patients with detailssuch as patient ID, name,
#  severity level, admission date, and bill amount. Doctors and staff need to: Quickly search patient records
#  using patient ID or name. Sort patients based on severity level or bill amount for prioritization and billing.
#select the optimal algorithm and justify i
class Patient:
    def __init__(self, patient_id, name, severity_level, admission_date, bill_amount):
        self.patient_id = patient_id
        self.name = name
        self.severity_level = severity_level
        self.admission_date = admission_date
        self.bill_amount = bill_amount
class HospitalRecords:
    def __init__(self):
        self.patients = []

    def add_patient(self, patient_id, name, severity_level, admission_date, bill_amount):
        patient = Patient(patient_id, name, severity_level, admission_date, bill_amount)
        self.patients.append(patient)

    def search_patient_by_id(self, patient_id):
        for patient in self.patients:
            if patient.patient_id == patient_id:
                return patient
        return None

    def search_patient_by_name(self, name):
        for patient in self.patients:
            if patient.name.lower() == name.lower():
```

```python
class HospitalRecords:
    def search_patient_by_name(self, name):
        return None

    def sort_patients_by_severity_level(self):
        self.patients.sort(key=lambda x: x.severity_level)

    def sort_patients_by_bill_amount(self):
        self.patients.sort(key=lambda x: x.bill_amount)
# Example usage
hospital = HospitalRecords()
hospital.add_patient("PT001", "John Doe", 5, "2024-01-01", 2000.00)
hospital.add_patient("PT002", "Jane Smith", 3, "2024-01-02", 1500.00)
hospital.add_patient("PT003", "Alice Johnson", 4, "2024-01-03", 3000.00)
print("Search for patient ID PT001:", hospital.search_patient_by_id("PT001"))
print("Search for patient name 'Jane Smith':", hospital.search_patient_by_name("Jane Smith"))
hospital.sort_patients_by_severity_level()
print("Patients sorted by severity level:", hospital.patients)
hospital.sort_patients_by_bill_amount()
print("Patients sorted by bill amount:", hospital.patients)
#justification for selected algorithm:
# Similar to the retail store's inventory system, we can use a linear search for finding patient
# records by ID or name due to the manageable dataset size and the need for simplicity. The time complexity
# of linear search is O(n) in the worst case. For sorting patients by severity level or bill amount, we can again
# use the built-in sort method which implements Timsort. Timsort is efficient for real-world data and has a time
#  complexity of O(n log n) in the worst case and O(n) in the best case when the data is already sorted. This
#  makes it suitable for our hospital records system where we may often have partially sorted data and need
#  efficient sorting for prioritization and billing.
```

OUTPUT:

```
TERMINAL                                                    powershell
 at 0x000001FE01F48B90>, <__main__.Product object at 0x000001FE01F48A50>]
PS C:\Users\reddy\OneDrive\Desktop\AI> & C:\Users\reddy\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:
/Users/reddy/OneDrive/Desktop/AI/Ass-12.5.py
Search for product ID P001: <__main__.Product object at 0x000001C347836BA0>
Search for product name 'Smartphone': <__main__.Product object at 0x000001C347AC8A50>
Products sorted by price: [<__main__.Product object at 0x000001C347AC8B90>, <__main__.Product object at 0x000
001C347AC8A50>, <__main__.Product object at 0x000001C347836BA0>]
Products sorted by stock quantity: [<__main__.Product object at 0x000001C347836BA0>, <__main__.Product object
 at 0x000001C347AC8B90>, <__main__.Product object at 0x000001C347AC8A50>]
Search for patient ID PT001: <__main__.Patient object at 0x000001C347836E40>
Search for patient name 'Jane Smith': <__main__.Patient object at 0x000001C347AC8CD0>
Patients sorted by severity level: [<__main__.Patient object at 0x000001C347AC8CD0>, <__main__.Patient object
 at 0x000001C347AC8E10>, <__main__.Patient object at 0x000001C347836E40>]
Patients sorted by bill amount: [<__main__.Patient object at 0x000001C347AC8CD0>, <__main__.Patient object at
 0x000001C347836E40>, <__main__.Patient object at 0x000001C347AC8E10>]
PS C:\Users\reddy\OneDrive\Desktop\AI>
```

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.

Welcome | task5.py

C: > Users > varshitha > OneDrive > Desktop > AI Assistant coding > task5.py > ...

```python
#Generate a code A university processes examination results for thousands of studentscontaining roll number, name, subject, and marks
#Search student results using roll number.Sort students based on marks to generate rank lists.
# University Examination Result Processing System

class StudentResult:
    def __init__(self, roll_number, name, subject, marks):
        self.roll_number = roll_number
        self.name = name
        self.subject = subject
        self.marks = marks

    def __repr__(self):
        return f"StudentResult({self.roll_number}, {self.name}, {self.subject}, {self.marks})"


class ExaminationSystem:
    """
    University examination result processing system.
    Uses binary search for efficient searching and merge sort for sorting.
    """

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

    def add_result(self, roll_number, name, subject, marks):
        """Add a student result."""
        self.results.append(StudentResult(roll_number, name, subject, marks))

    def binary_search_by_roll(self, roll_number):
        """
        Binary search for student by roll number.
        Time Complexity: O(log n)
        Justification: Efficient for large datasets; requires sorted data.
        """
        sorted_results = sorted(self.results, key=lambda x: x.roll_number)
        left, right = 0, len(sorted_results) - 1
```

```
        while left <= right:
            mid = (left + right) // 2
            if sorted_results[mid].roll_number == roll_number:
                return sorted_results[mid]
            elif sorted_results[mid].roll_number < roll_number:
                left = mid + 1
            else:
                right = mid - 1

        return None

    def merge_sort_by_marks(self):
        """
        Merge sort students by marks in descending order.
        Time Complexity: O(n log n)
        Justification: Stable sort, consistent performance, efficient for large datasets.
        """
        def merge(arr, left, mid, right):
            left_arr = arr[left:mid+1]
            right_arr = arr[mid+1:right+1]
            i = j = 0
            k = left

            while i < len(left_arr) and j < len(right_arr):
                if left_arr[i].marks >= right_arr[j].marks:
                    arr[k] = left_arr[i]
                    i += 1
                else:
                    arr[k] = right_arr[j]
                    j += 1
                k += 1

            while i < len(left_arr):
                arr[k] = left_arr[i]
```

```
    def merge_sort_by_marks(self):
        def merge(arr, left, mid, right):
            while i < len(left_arr):
                arr[k] = left_arr[i]
                i += 1
                k += 1

            while j < len(right_arr):
                arr[k] = right_arr[j]
                j += 1
                k += 1

        def merge_sort_helper(arr, left, right):
            if left < right:
                mid = (left + right) // 2
                merge_sort_helper(arr, left, mid)
                merge_sort_helper(arr, mid + 1, right)
                merge(arr, left, mid, right)

        if self.results:
            merge_sort_helper(self.results, 0, len(self.results) - 1)

        return self.results

    def generate_rank_list(self):
        """Generate rank list sorted by marks."""
        sorted_results = self.merge_sort_by_marks()
        return [(i+1, result) for i, result in enumerate(sorted_results)]

    def display_rank_list(self):
        """Display rank list with ranks."""
        rank_list = self.generate_rank_list()
        print("\n{'RANK':<6}{'ROLL NO':<12}{'NAME':<20}{'SUBJECT':<15}{'MARKS':<8}")
        print("-" * 65)
```

```
    def display_rank_list(self):
        for rank, result in rank_list:
            print(f"{rank:<6}{result.roll_number:<12}{result.name:<20}{result.subject:<15}{result.marks:<8}")


# Example usage
if __name__ == "__main__":
    system = ExaminationSystem()

    # Add student results
    system.add_result(101, "Alice Johnson", "Mathematics", 95)
    system.add_result(102, "Bob Smith", "Mathematics", 87)
    system.add_result(103, "Charlie Brown", "Mathematics", 92)
    system.add_result(104, "Diana Prince", "Mathematics", 88)
    system.add_result(105, "Eve Davis", "Mathematics", 90)

    # Search by roll number
    print("Searching for Roll Number 103:")
    result = system.binary_search_by_roll(103)
    print(result if result else "Not found")

    # Generate and display rank list
    print("\nGenerated Rank List:")
    system.display_rank_list()
```

OUTPUT:

```
PS C:\Users\varshitha> & C:/Users/varshitha/anaconda3/python.exe "c:/Users/varshitha/OneDrive/Desktop/AI Assistant coding/task5.py"
Searching for Roll Number 103:
StudentResult(103, Charlie Brown, Mathematics, 92)

Generated Rank List:

{'RANK':<6}{'ROLL NO':<12}{'NAME':<20}{'SUBJECT':<15}{'MARKS':<8}
----------------------------------------------------------------
1     101         Alice Johnson       Mathematics    95
2     103         Charlie Brown       Mathematics    92

{'RANK':<6}{'ROLL NO':<12}{'NAME':<20}{'SUBJECT':<15}{'MARKS':<8}
----------------------------------------------------------------
1     101         Alice Johnson       Mathematics    95
2     103         Charlie Brown       Mathematics    92
3     105         Eve Davis           Mathematics    90
4     104         Diana Prince        Mathematics    88
5     102         Bob Smith           Mathematics    87
PS C:\Users\varshitha>
```

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.

Welcome | task6.py

C: > Users > varshitha > OneDrive > Desktop > AI Assistant coding > task6.py > ...

```python
#An online food delivery application stores thousands of orders with order ID, restaurant name, delivery time, price, and order status.
#platform needs to:1. Quickly find an order using order ID.2. Sort orders based on delivery time or price.

from dataclasses import dataclass
from typing import List
from datetime import datetime

@dataclass
class Order:
    order_id: int
    restaurant_name: str
    delivery_time: int  # in minutes
    price: float
    status: str

class FoodDeliveryPlatform:
    def __init__(self):
        self.orders: List[Order] = []
        self.order_index: dict = {}  # Hash map for O(1) search

    def add_order(self, order: Order) -> None:
        """Add order and maintain index for fast lookup."""
        self.orders.append(order)
        self.order_index[order.order_id] = order

    def search_by_order_id(self, order_id: int) -> Order:
        """
        Search order by ID using hash map.
        Time Complexity: O(1) - Constant time lookup
        Justification: Hash map provides instant access without scanning.
        """
        return self.order_index.get(order_id, None)

    def sort_by_delivery_time(self) -> List[Order]:
        """
        Sort orders by delivery time using Timsort.
        Time Complexity: O(n log n) - Python's built-in optimal
        Justification: Timsort handles partially sorted data efficiently.
        """
        return sorted(self.orders, key=lambda x: x.delivery_time)

    def sort_by_price(self) -> List[Order]:
        """
        Sort orders by price using Timsort.
        Time Complexity: O(n log n)
        Justification: Best for general-purpose sorting of comparable elements.
        """
        return sorted(self.orders, key=lambda x: x.price)

    def filter_and_sort(self, status: str, sort_by: str) -> List[Order]:
        """Filter by status and sort by delivery_time or price."""
        filtered = [o for o in self.orders if o.status == status]

        if sort_by == "delivery_time":
            return sorted(filtered, key=lambda x: x.delivery_time)
        elif sort_by == "price":
            return sorted(filtered, key=lambda x: x.price)
        return filtered


# Example Usage
if __name__ == "__main__":
    platform = FoodDeliveryPlatform()

    # Add sample orders
    platform.add_order(Order(101, "Pizza Hut", 30, 12.99, "delivered"))
    platform.add_order(Order(102, "Burger King", 45, 8.50, "pending"))
    platform.add_order(Order(103, "Subway", 20, 9.75, "delivered"))
    platform.add_order(Order(104, "McDonald's", 35, 6.99, "in_progress"))
```

C: > Users > varshitha > OneDrive > Desktop > AI Assistant coding > task6.py > ...

```
    # Search by order ID
    order = platform.search_by_order_id(102)
    print(f"Found Order: {order}\n")

    # Sort by delivery time
    print("Sorted by Delivery Time:")
    for order in platform.sort_by_delivery_time():
        print(f"  {order.order_id}: {order.delivery_time} min")

    # Sort by price
    print("\nSorted by Price:")
    for order in platform.sort_by_price():
        print(f"  {order.order_id}: ${order.price}")
```

OUTPUT:

```
PS C:\Users\varshitha> & C:/Users/varshitha/anaconda3/python.exe "c:/Users/varshitha/OneDrive/Desktop/AI Assistant coding/task6.py"
Found Order: Order(order_id=102, restaurant_name='Burger King', delivery_time=45, price=8.5, status='pending')

Sorted by Delivery Time:
  103: 20 min
  101: 30 min
  104: 35 min
  102: 45 min

Sorted by Price:
  104: $6.99
  102: $8.5
  103: $9.75

Sorted by Delivery Time:
  103: 20 min
  101: 30 min
  104: 35 min
  102: 45 min

Sorted by Price:
  104: $6.99
  102: $8.5
  103: $9.75
Sorted by Delivery Time:
  103: 20 min
  101: 30 min
  104: 35 min
  102: 45 min

Sorted by Price:
  104: $6.99
  102: $8.5
  103: $9.75
  101: 30 min
  104: 35 min
  102: 45 min

Sorted by Price:
  104: $6.99
  102: $8.5
```

```
Sorted by Price:
  104: $6.99
  102: $8.5
  103: $9.75
  102: 45 min

Sorted by Price:
  104: $6.99
  102: $8.5
  103: $9.75
  104: $6.99
  102: $8.5
  103: $9.75
  102: $8.5
  103: $9.75
  101: $12.99
PS C:\Users\varshitha>
```