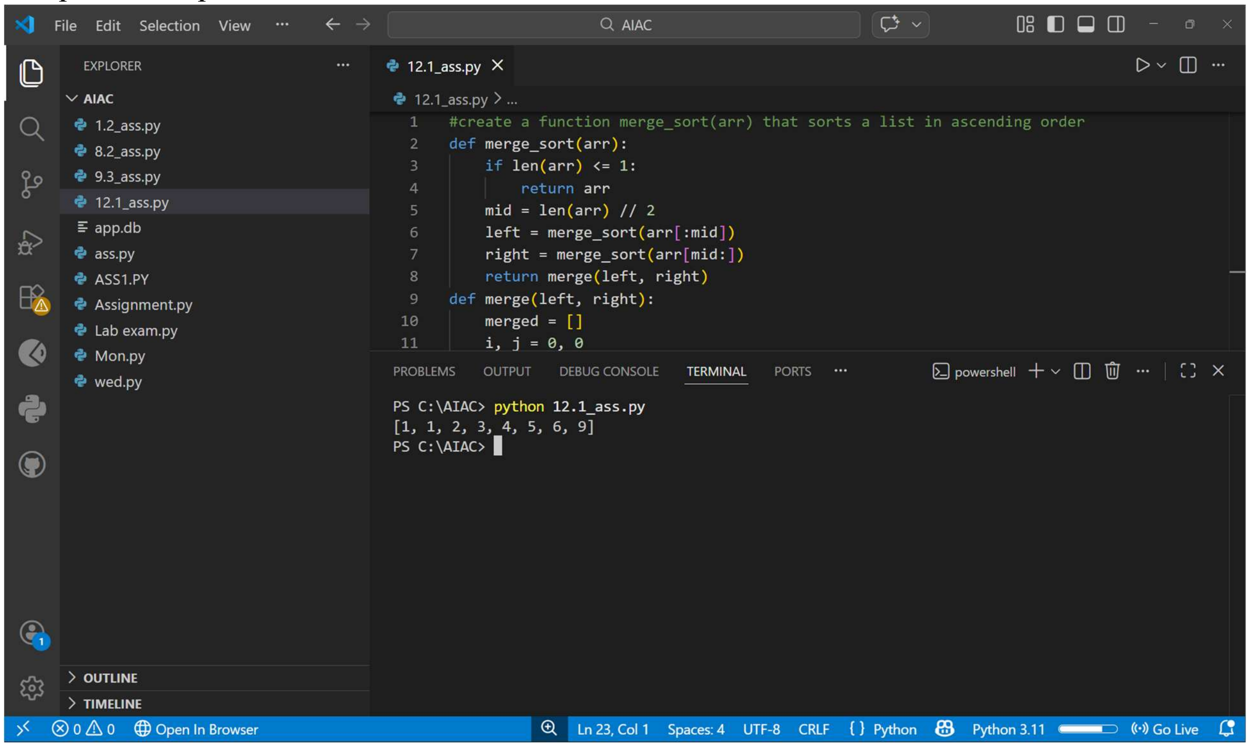
ASSIGNMENT-12.1

Name: J.Vyshnavi

HT. No: 2303A51895

Batch: 08
Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms
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:

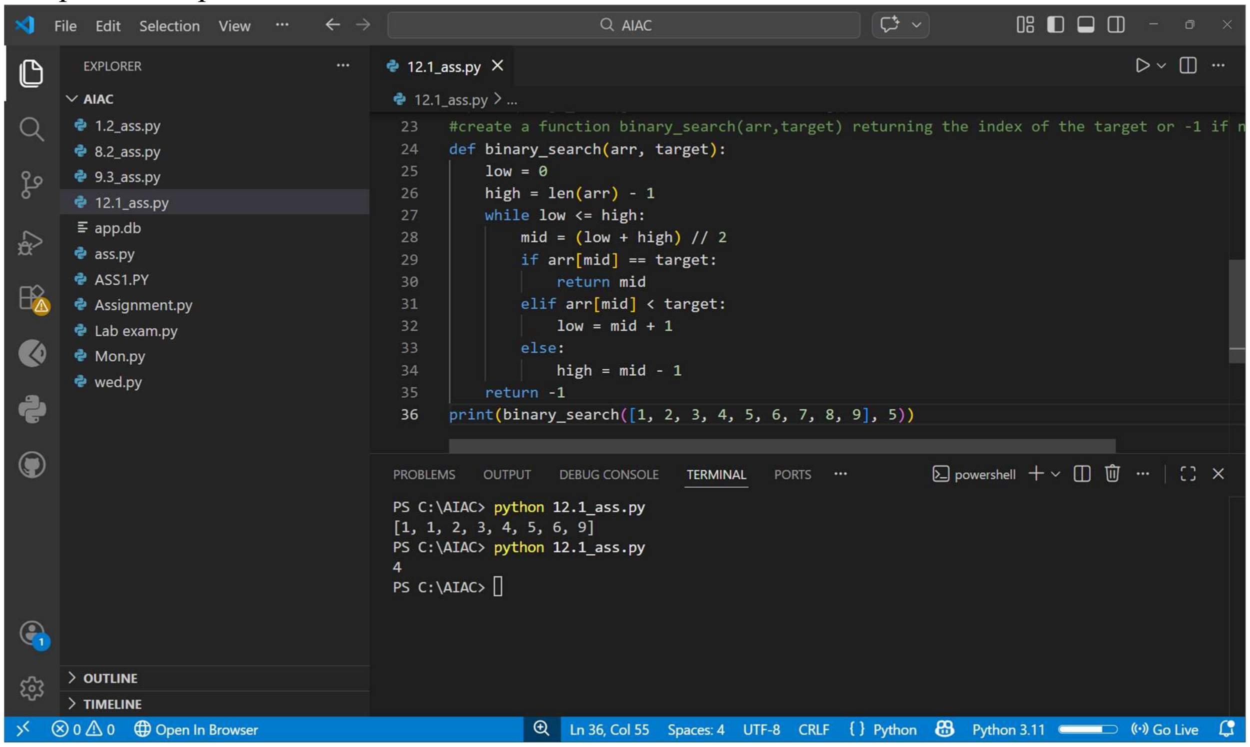


```python
#create a function merge_sort(arr) that sorts a list in ascending order
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
def merge(left, right):
    merged = []
    i, j = 0, 0
```

```
PS C:\AIAC> python 12.1_ass.py
[1, 1, 2, 3, 4, 5, 6, 9]
PS C:\AIAC>
```

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

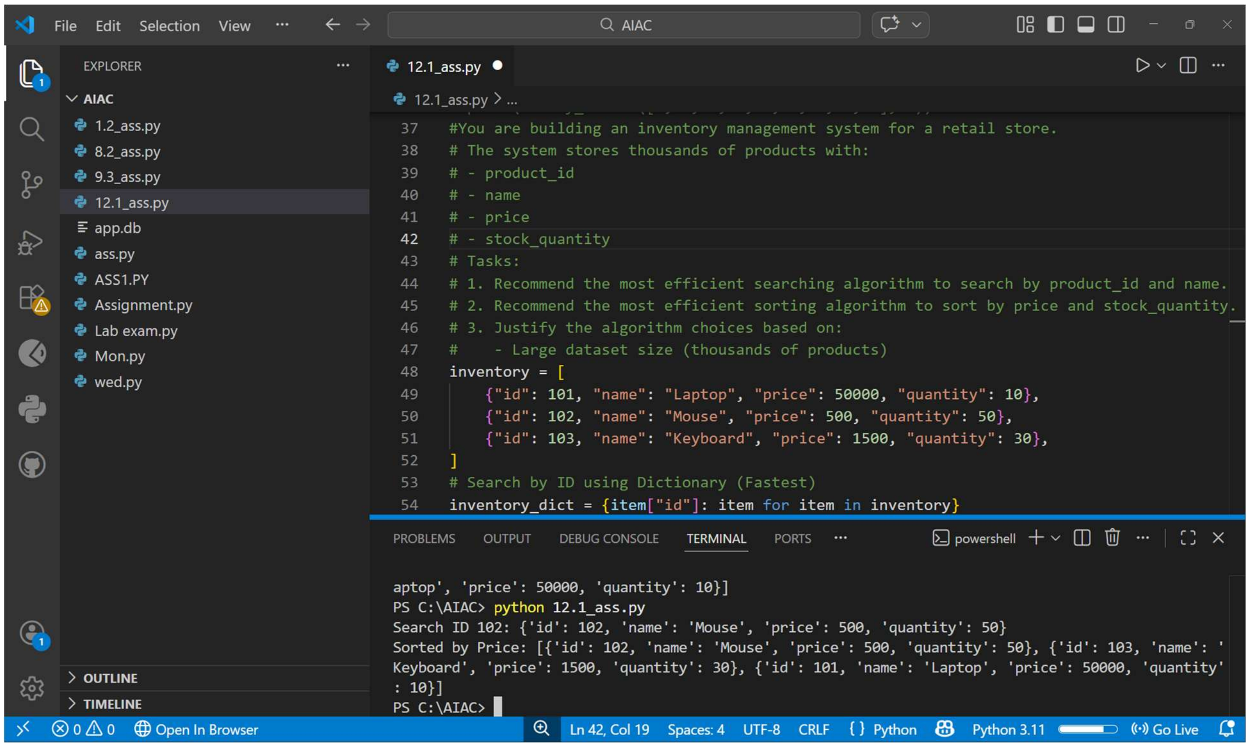
worst-case complexities.
o Test with various inputs.
• Expected Output:



```
#create a function binary_search(arr,target) returning the index of the target or -1 if n
def binary_search(arr, target):
    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
print(binary_search([1, 2, 3, 4, 5, 6, 7, 8, 9], 5))
```

```
PS C:\AIAC> python 12.1_ass.py
[1, 1, 2, 3, 4, 5, 6, 9]
PS C:\AIAC> python 12.1_ass.py
4
PS C:\AIAC> 
```

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:



```python
#You are building an inventory management system for a retail store.
# The system stores thousands of products with:
# - product_id
# - name
# - price
# - stock_quantity
# Tasks:
# 1. Recommend the most efficient searching algorithm to search by product_id and name.
# 2. Recommend the most efficient sorting algorithm to sort by price and stock_quantity.
# 3. Justify the algorithm choices based on:
#    - Large dataset size (thousands of products)
inventory = [
    {"id": 101, "name": "Laptop", "price": 50000, "quantity": 10},
    {"id": 102, "name": "Mouse", "price": 500, "quantity": 50},
    {"id": 103, "name": "Keyboard", "price": 1500, "quantity": 30},
]
# Search by ID using Dictionary (Fastest)
inventory_dict = {item["id"]: item for item in inventory}
```

```
aptop', 'price': 50000, 'quantity': 10}]
PS C:\AIAC> python 12.1_ass.py
Search ID 102: {'id': 102, 'name': 'Mouse', 'price': 500, 'quantity': 50}
Sorted by Price: [{'id': 102, 'name': 'Mouse', 'price': 500, 'quantity': 50}, {'id': 103, 'name': '
Keyboard', 'price': 1500, 'quantity': 30}, {'id': 101, 'name': 'Laptop', 'price': 50000, 'quantity'
: 10}]
PS C:\AIAC>
```

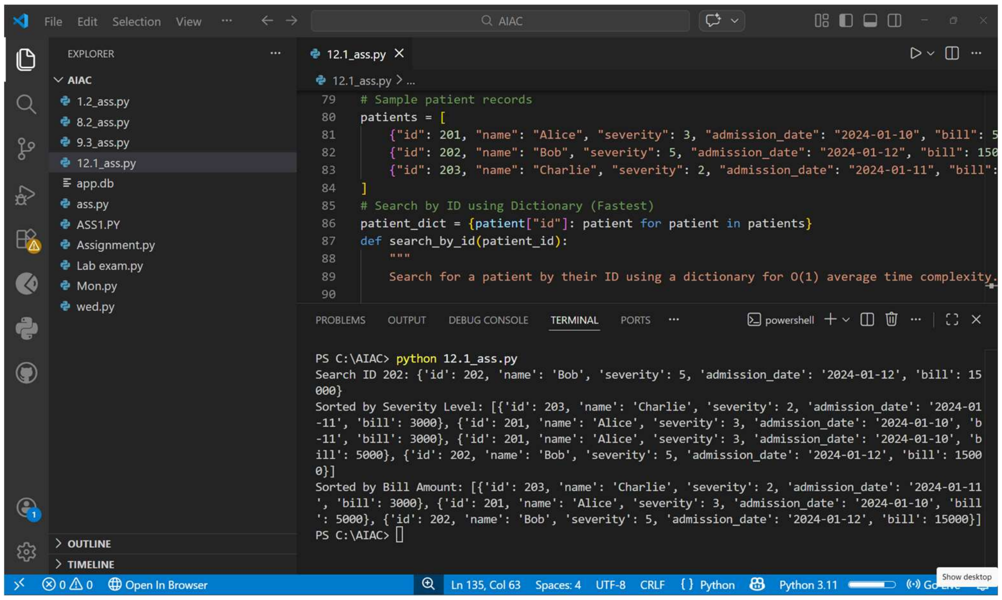
Task description #4: Smart Hospital Patient Management System

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
# Sample patient records
patients = [
    {"id": 201, "name": "Alice", "severity": 3, "admission_date": "2024-01-10", "bill": 5
    {"id": 202, "name": "Bob", "severity": 5, "admission_date": "2024-01-12", "bill": 150
    {"id": 203, "name": "Charlie", "severity": 2, "admission_date": "2024-01-11", "bill":
]
# Search by ID using Dictionary (Fastest)
patient_dict = {patient["id"]: patient for patient in patients}
def search_by_id(patient_id):
    """
    Search for a patient by their ID using a dictionary for O(1) average time complexity.

```

```
PS C:\AIAC> python 12.1_ass.py
Search ID 202: {'id': 202, 'name': 'Bob', 'severity': 5, 'admission_date': '2024-01-12', 'bill': 15
000}
Sorted by Severity Level: [{'id': 203, 'name': 'Charlie', 'severity': 2, 'admission_date': '2024-01
-11', 'bill': 3000}, {'id': 201, 'name': 'Alice', 'severity': 3, 'admission_date': '2024-01-10', 'b
-11', 'bill': 3000}, {'id': 201, 'name': 'Alice', 'severity': 3, 'admission_date': '2024-01-10', 'b
ill': 5000}, {'id': 202, 'name': 'Bob', 'severity': 5, 'admission_date': '2024-01-12', 'bill': 1500
0}]
Sorted by Bill Amount: [{'id': 203, 'name': 'Charlie', 'severity': 2, 'admission_date': '2024-01-11
', 'bill': 3000}, {'id': 201, 'name': 'Alice', 'severity': 3, 'admission_date': '2024-01-10', 'bill
': 5000}, {'id': 202, 'name': 'Bob', 'severity': 5, 'admission_date': '2024-01-12', 'bill': 15000}]
PS C:\AIAC>
```

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:

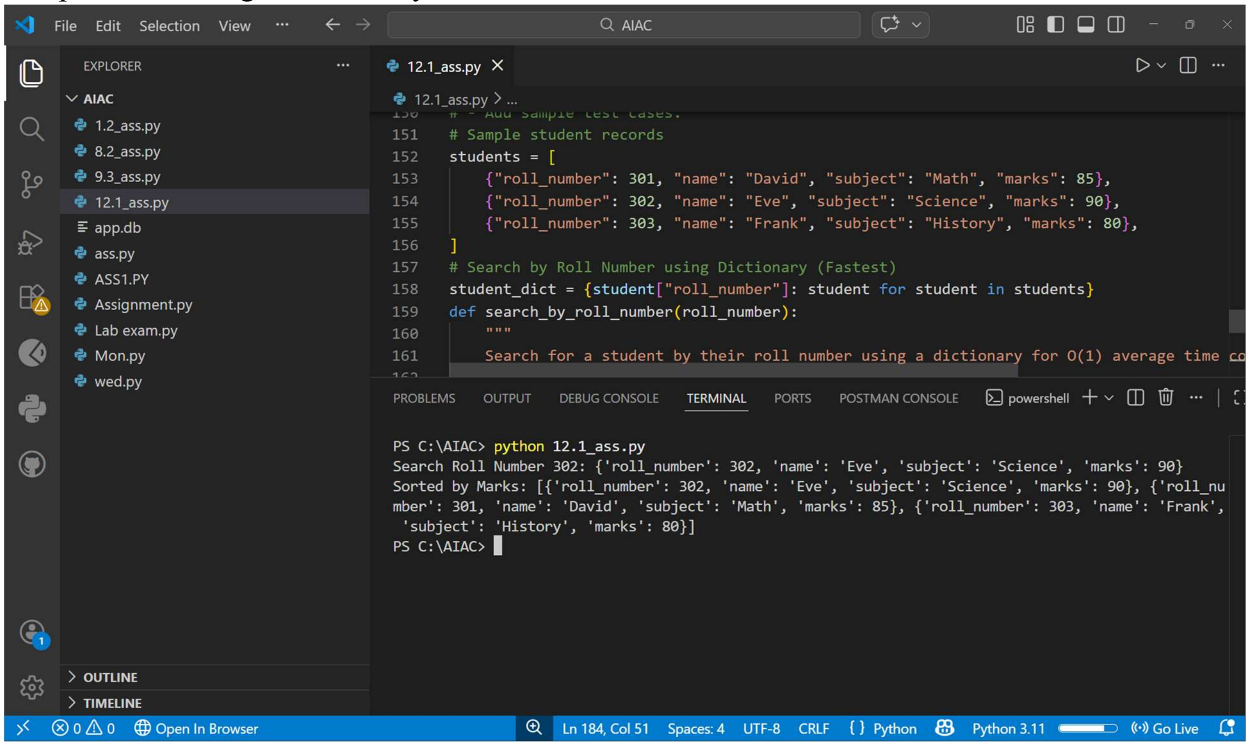
1. Search student results using roll number.
2. Sort students based on marks to generate rank lists.

Student Task

• Identify efficient searching and sorting algorithms using AI assistance.

• Justify the choice of algorithms.

• Implement the algorithms in Python.



```python
# Sample student records
students = [
    {"roll_number": 301, "name": "David", "subject": "Math", "marks": 85},
    {"roll_number": 302, "name": "Eve", "subject": "Science", "marks": 90},
    {"roll_number": 303, "name": "Frank", "subject": "History", "marks": 80},
]
# Search by Roll Number using Dictionary (Fastest)
student_dict = {student["roll_number"]: student for student in students}
def search_by_roll_number(roll_number):
    """
    Search for a student by their roll number using a dictionary for O(1) average time co
```

```
PS C:\AIAC> python 12.1_ass.py
Search Roll Number 302: {'roll_number': 302, 'name': 'Eve', 'subject': 'Science', 'marks': 90}
Sorted by Marks: [{'roll_number': 302, 'name': 'Eve', 'subject': 'Science', 'marks': 90}, {'roll_nu
mber': 301, 'name': 'David', 'subject': 'Math', 'marks': 85}, {'roll_number': 303, 'name': 'Frank',
 'subject': 'History', 'marks': 80}]
PS C:\AIAC>
```

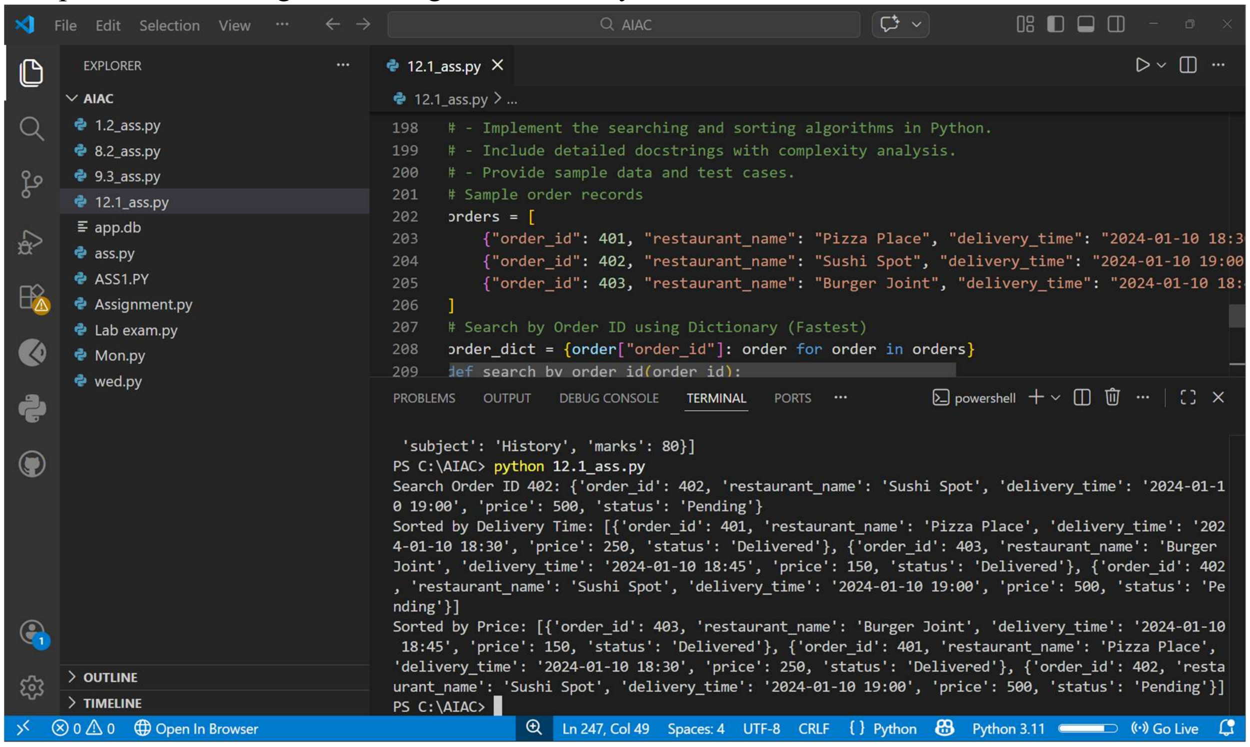
Task Description #6: Online Food Delivery Platform

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
# - Implement the searching and sorting algorithms in Python.
# - Include detailed docstrings with complexity analysis.
# - Provide sample data and test cases.
# Sample order records
orders = [
    {"order_id": 401, "restaurant_name": "Pizza Place", "delivery_time": "2024-01-10 18:3
    {"order_id": 402, "restaurant_name": "Sushi Spot", "delivery_time": "2024-01-10 19:00
    {"order_id": 403, "restaurant_name": "Burger Joint", "delivery_time": "2024-01-10 18:
]
# Search by Order ID using Dictionary (Fastest)
order_dict = {order["order_id"]: order for order in orders}
def search_by_order_id(order_id):
```

```
 'subject': 'History', 'marks': 80}]
PS C:\AIAC> python 12.1_ass.py
Search Order ID 402: {'order_id': 402, 'restaurant_name': 'Sushi Spot', 'delivery_time': '2024-01-1
0 19:00', 'price': 500, 'status': 'Pending'}
Sorted by Delivery Time: [{'order_id': 401, 'restaurant_name': 'Pizza Place', 'delivery_time': '202
4-01-10 18:30', 'price': 250, 'status': 'Delivered'}, {'order_id': 403, 'restaurant_name': 'Burger
Joint', 'delivery_time': '2024-01-10 18:45', 'price': 150, 'status': 'Delivered'}, {'order_id': 402
, 'restaurant_name': 'Sushi Spot', 'delivery_time': '2024-01-10 19:00', 'price': 500, 'status': 'Pe
nding'}]
Sorted by Price: [{'order_id': 403, 'restaurant_name': 'Burger Joint', 'delivery_time': '2024-01-10
 18:45', 'price': 150, 'status': 'Delivered'}, {'order_id': 401, 'restaurant_name': 'Pizza Place',
'delivery_time': '2024-01-10 18:30', 'price': 250, 'status': 'Delivered'}, {'order_id': 402, 'resta
urant_name': 'Sushi Spot', 'delivery_time': '2024-01-10 19:00', 'price': 500, 'status': 'Pending'}]
PS C:\AIAC>
```