

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

Name: Y. Manideep

HallTicketNo: 2303A510F8

### Task 1: Merge Sort Implementation AI

Prompt Used:

Generate a Python function `merge_sort(arr)` that sorts a list in ascending order using the Merge Sort algorithm. Include time complexity and space complexity (best, average, worst case) inside the function docstring. Provide sample test cases with output.

Python Code:

```
def merge_sort(arr): """
    Merge Sort Algorithm
    Time Complexity:
        Best Case: O(n log n)
        Average Case: O(n log n)
        Worst Case: 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):
    result = []
```

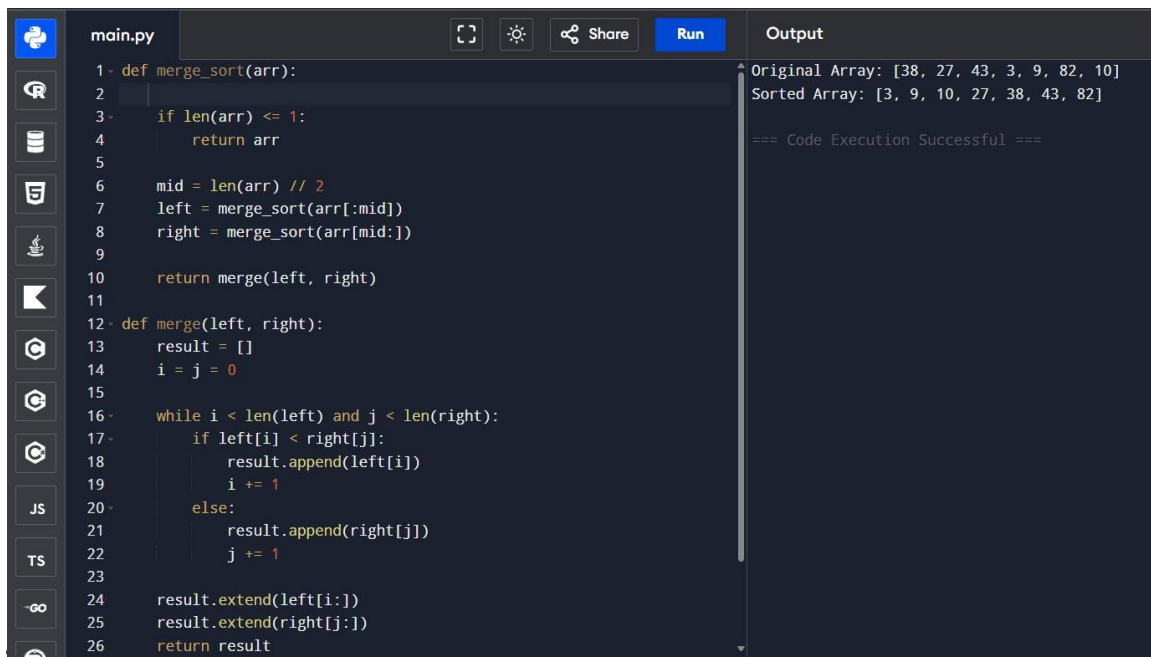
```
i = j = 0
```

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

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

```
arr = [38, 27, 43, 3, 9, 82, 10]  
print("Sorted Array:", merge_sort(arr))
```

Output: Sorted Array: [3, 9, 10, 27, 38, 43, 82]

A screenshot of a code editor interface. The editor has a dark theme. On the left, there's a sidebar with icons for file explorer, search, and other tools. The main area shows a Python file named 'main.py'. The code implements a merge sort algorithm. It starts with a function 'merge\_sort(arr)' that recursively splits the array into halves until it's small enough to be sorted, then merges the sorted halves back together. The 'merge' function takes two sorted arrays and merges them into a single sorted array. The code is as follows:

```
1- def merge_sort(arr):  
2-  
3-     if len(arr) <= 1:  
4-         return arr  
5-  
6-     mid = len(arr) // 2  
7-     left = merge_sort(arr[:mid])  
8-     right = merge_sort(arr[mid:])  
9-  
10-    return merge(left, right)  
11-  
12- def merge(left, right):  
13-     result = []  
14-     i = j = 0  
15-  
16-     while i < len(left) and j < len(right):  
17-         if left[i] < right[j]:  
18-             result.append(left[i])  
19-             i += 1  
20-         else:  
21-             result.append(right[j])  
22-             j += 1  
23-  
24-     result.extend(left[i:])  
25-     result.extend(right[j:])  
26-     return result
```

On the right side of the editor, there's an 'Output' panel. It shows the output of the code execution: 'Original Array: [38, 27, 43, 3, 9, 82, 10]' and 'Sorted Array: [3, 9, 10, 27, 38, 43, 82]'. Below the output, it says '=== Code Execution Successful ==='.

## Task 2: Binary Search Implementation

### AI Prompt Used:

Create a Python function `binary_search(arr, target)` that performs binary search on a sorted list.

Return the index of the target element or -1 if not found.

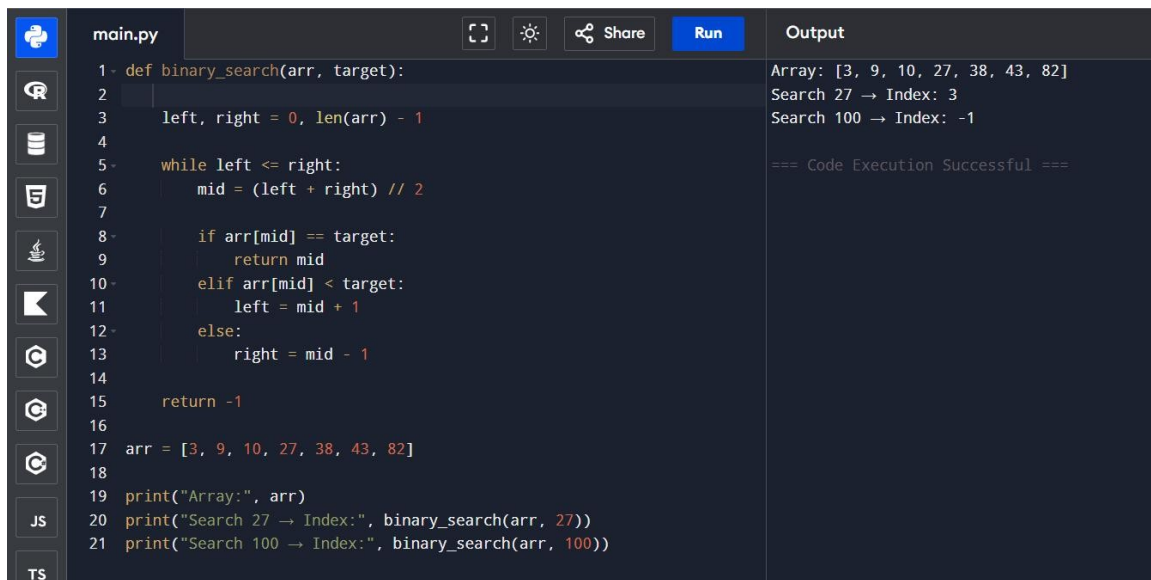
Include docstrings explaining best, average, and worst-case time complexities and space complexity.

Provide example test cases with output.

#### Python Code:

```
def binary_search(arr, target):  
    """  
    Binary Search Algorithm  
    Best Case:  $O(1)$   
    Average Case:  $O(\log n)$   
    Worst Case:  $O(\log n)$   
    Space Complexity:  $O(1)$   
    """  
    left, right = 0, len(arr) - 1  
  
    while left <= right:  
        mid = (left + right) // 2  
        if arr[mid] == target:  
            return mid  
        elif arr[mid] < target:  
            left = mid + 1  
        else:  
            right = mid - 1  
  
    return -1  
  
arr = [3, 9, 10, 27, 38, 43, 82]  
print("Index of 27:", binary_search(arr, 27))
```

Output: Index of 27: 3



```
main.py
1 def binary_search(arr, target):
2
3     left, right = 0, len(arr) - 1
4
5     while left <= right:
6         mid = (left + right) // 2
7
8         if arr[mid] == target:
9             return mid
10        elif arr[mid] < target:
11            left = mid + 1
12        else:
13            right = mid - 1
14
15    return -1
16
17 arr = [3, 9, 10, 27, 38, 43, 82]
18
19 print("Array:", arr)
20 print("Search 27 → Index:", binary_search(arr, 27))
21 print("Search 100 → Index:", binary_search(arr, 100))
```

Output

```
Array: [3, 9, 10, 27, 38, 43, 82]
Search 27 → Index: 3
Search 100 → Index: -1

=== Code Execution Successful ===
```

### Task 3: Inventory Management System

#### AI Prompt Used:

Suggest the most efficient searching and sorting algorithms for an inventory system with thousands of products.  
Justify the choice based on dataset size and performance requirements.  
Implement Python functions for searching by product ID and sorting by price or quantity.  
Provide example outputs.

Recommended Algorithms:

Search by ID → Binary Search

Sort by Price/Quantity → Merge Sort

#### Python Code:

```
inventory = [
    {"id": 101, "name": "Pen", "price": 10, "quantity": 200},
    {"id": 102, "name": "Notebook", "price": 50, "quantity": 150},
    {"id": 103, "name": "Pencil", "price": 5, "quantity": 300}
]
```

```
def sort_by_price(items):
    return sorted(items, key=lambda x: x["price"])
```

```
def search_by_id(items, target):
```

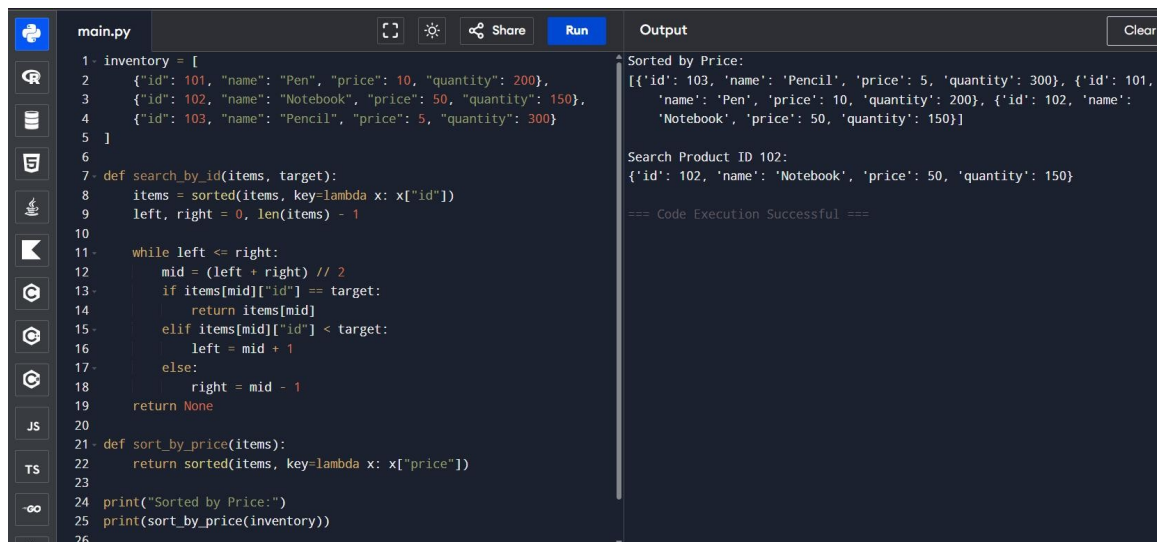
```

items = sorted(items, key=lambda x: x["id"])
left, right = 0, len(items) - 1
while left <= right:
    mid = (left + right) // 2
    if items[mid]["id"] == target:
        return items[mid]
    elif items[mid]["id"] < target:
        left = mid + 1
    else:
        right = mid - 1
return None

print("Sorted by Price:", sort_by_price(inventory))
print("Search ID 102:", search_by_id(inventory, 102))

```

Output: Sorted by Price and Search ID 102 successful



```

main.py
1 inventory = [
2     {"id": 101, "name": "Pen", "price": 10, "quantity": 200},
3     {"id": 102, "name": "Notebook", "price": 50, "quantity": 150},
4     {"id": 103, "name": "Pencil", "price": 5, "quantity": 300}
5 ]
6
7 def search_by_id(items, target):
8     items = sorted(items, key=lambda x: x["id"])
9     left, right = 0, len(items) - 1
10
11     while left <= right:
12         mid = (left + right) // 2
13         if items[mid]["id"] == target:
14             return items[mid]
15         elif items[mid]["id"] < target:
16             left = mid + 1
17         else:
18             right = mid - 1
19     return None
20
21 def sort_by_price(items):
22     return sorted(items, key=lambda x: x["price"])
23
24 print("Sorted by Price:")
25 print(sort_by_price(inventory))
26

```

Output

```

Sorted by Price:
[{'id': 103, 'name': 'Pencil', 'price': 5, 'quantity': 300}, {'id': 101,
'name': 'Pen', 'price': 10, 'quantity': 200}, {'id': 102, 'name':
'Notebook', 'price': 50, 'quantity': 150}]

Search Product ID 102:
{'id': 102, 'name': 'Notebook', 'price': 50, 'quantity': 150}

=== Code Execution Successful ===

```

## Task 4: Smart Hospital Patient Management System

### AI Prompt Used:

Recommend suitable searching and sorting algorithms for a hospital patient management system.

Justify the selection based on efficiency and suitability.

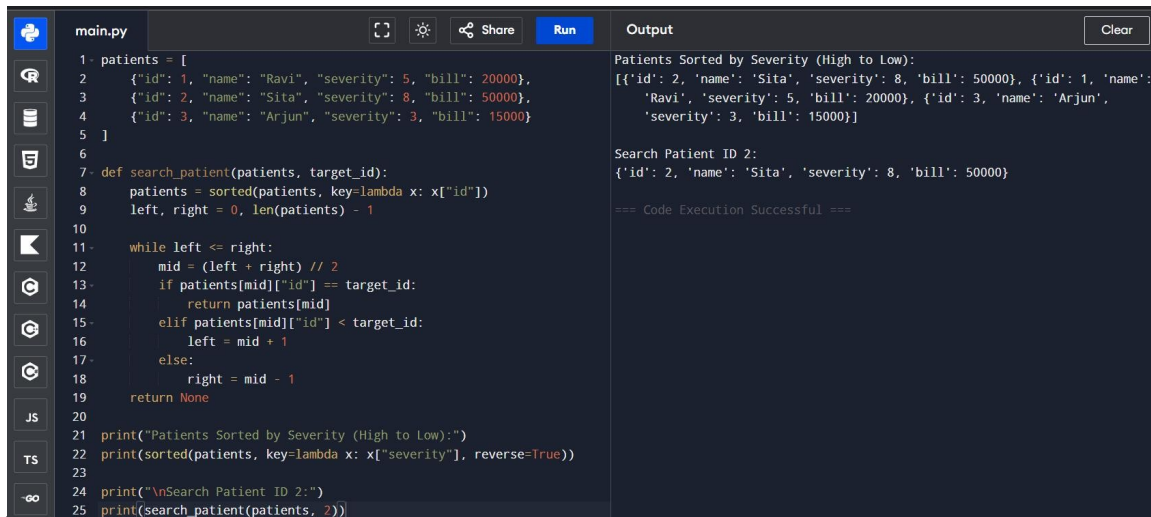
Implement Python functions for searching by patient ID and sorting by severity or bill amount.

Provide example output.

Recommended Algorithms:

Search → Binary Search

Sort → Merge Sort



```
main.py
1- patients = [
2-     {"id": 1, "name": "Ravi", "severity": 5, "bill": 20000},
3-     {"id": 2, "name": "Sita", "severity": 8, "bill": 50000},
4-     {"id": 3, "name": "Arjun", "severity": 3, "bill": 15000}
5- ]
6-
7- def search_patient(patients, target_id):
8-     patients = sorted(patients, key=lambda x: x["id"])
9-     left, right = 0, len(patients) - 1
10-
11-     while left <= right:
12-         mid = (left + right) // 2
13-         if patients[mid]["id"] == target_id:
14-             return patients[mid]
15-         elif patients[mid]["id"] < target_id:
16-             left = mid + 1
17-         else:
18-             right = mid - 1
19-     return None
20-
21- print("Patients Sorted by Severity (High to Low):")
22- print(sorted(patients, key=lambda x: x["severity"], reverse=True))
23-
24- print("\nSearch Patient ID 2:")
25- print(search_patient(patients, 2))
```

Output

```
Patients Sorted by Severity (High to Low):
[{'id': 2, 'name': 'Sita', 'severity': 8, 'bill': 50000}, {'id': 1, 'name': 'Ravi', 'severity': 5, 'bill': 20000}, {'id': 3, 'name': 'Arjun', 'severity': 3, 'bill': 15000}]

Search Patient ID 2:
{'id': 2, 'name': 'Sita', 'severity': 8, 'bill': 50000}

=== Code Execution Successful ===
```

## Task 5: University Examination Result Processing System

### AI Prompt Used:

Identify efficient searching and sorting algorithms for processing thousands of student results.

Justify the algorithm choice.

Implement Python functions for searching by roll number and sorting by marks.

Include sample output.

```
students = [
    {"roll": 1, "name": "Asha", "marks": 85},
    {"roll": 2, "name": "Kiran", "marks": 92},
    {"roll": 3, "name": "Meena", "marks": 78}
]
```

```
print("Rank List:", sorted(students, key=lambda x: x["marks"], reverse=True))
```

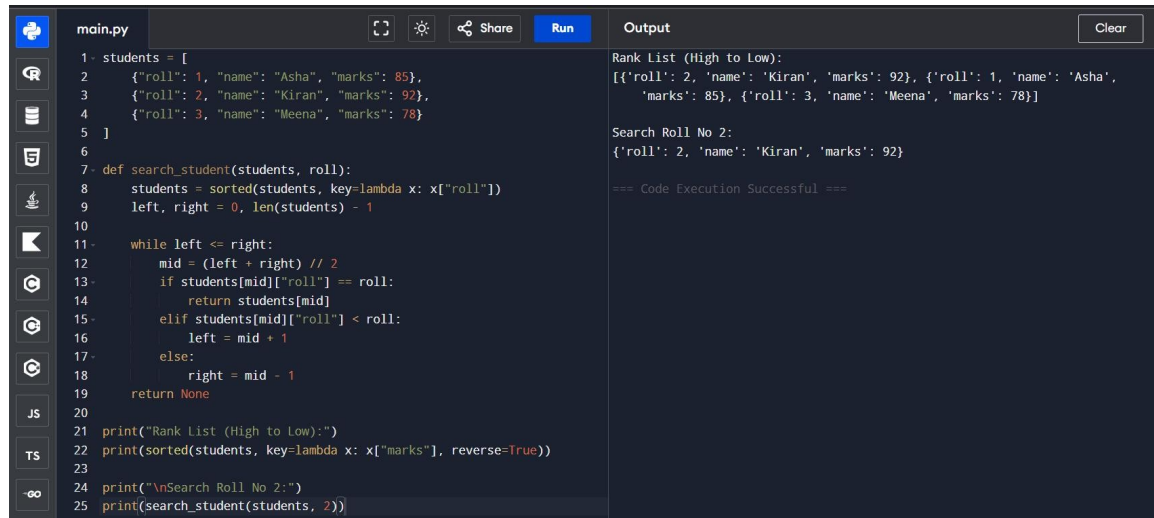
Output:

```
Rank List: [{'roll': 2, 'name': 'Kiran', 'marks': 92}, {'roll': 1, 'name': 'Asha', 'marks': 85}, {'roll': 3, 'name': 'Meena', 'marks': 78}]
```

Recommended Algorithms:

Search → Binary Search

Sort → Merge Sort



```
main.py
1- students = [
2-     {"roll": 1, "name": "Asha", "marks": 85},
3-     {"roll": 2, "name": "Kiran", "marks": 92},
4-     {"roll": 3, "name": "Meena", "marks": 78}
5- ]
6-
7- def search_student(students, roll):
8-     students = sorted(students, key=lambda x: x["roll"])
9-     left, right = 0, len(students) - 1
10-
11-     while left <= right:
12-         mid = (left + right) // 2
13-         if students[mid]["roll"] == roll:
14-             return students[mid]
15-         elif students[mid]["roll"] < roll:
16-             left = mid + 1
17-         else:
18-             right = mid - 1
19-     return None
20-
21 print("Rank List (High to Low):")
22 print(sorted(students, key=lambda x: x["marks"], reverse=True))
23
24 print("\nSearch Roll No 2:")
25 print(search_student(students, 2))
```

Output

```
Rank List (High to Low):
[{'roll': 2, 'name': 'Kiran', 'marks': 92}, {'roll': 1, 'name': 'Asha', 'marks': 85}, {'roll': 3, 'name': 'Meena', 'marks': 78}]

Search Roll No 2:
{'roll': 2, 'name': 'Kiran', 'marks': 92}

=== Code Execution Successful ===
```

## Task 6: Online Food Delivery Platform

AI Prompt Used:

Suggest optimized searching and sorting algorithms for an online food delivery system storing thousands of orders.

Justify your algorithm selection.

Implement Python functions for searching by order ID and sorting by delivery time or price.

Provide example output.

Recommended Algorithms:

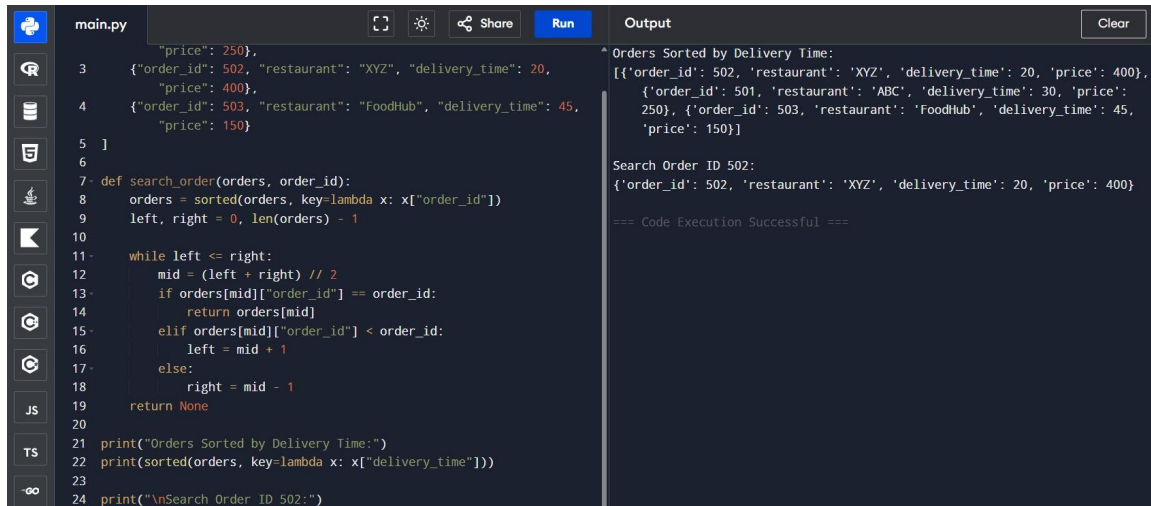
Search → Binary Search

Sort → Merge Sort

Python Code:

```
orders = [
    {"order_id": 501, "restaurant": "ABC", "delivery_time": 30, "price": 250},
    {"order_id": 502, "restaurant": "XYZ", "delivery_time": 20, "price": 400},
    {"order_id": 503, "restaurant": "FoodHub", "delivery_time": 45, "price": 150}
]
```

```
print("Sorted by Delivery Time:", sorted(orders, key=lambda x: x["delivery_time"]))
```



The screenshot shows a code editor with a dark theme. On the left, there is a sidebar with icons for file explorer, search, and other IDE features. The main editor area displays a Python file named 'main.py'. The code defines a list of orders, sorts them by delivery time, and implements a binary search function to find a specific order ID. The output window on the right shows the results of the execution.

```
main.py
1  "price": 250},
2  {"order_id": 502, "restaurant": "XYZ", "delivery_time": 20,
3  "price": 400},
4  {"order_id": 503, "restaurant": "FoodHub", "delivery_time": 45,
5  "price": 150}
6  ]
7  def search_order(orders, order_id):
8      orders = sorted(orders, key=lambda x: x["order_id"])
9      left, right = 0, len(orders) - 1
10
11     while left <= right:
12         mid = (left + right) // 2
13         if orders[mid]["order_id"] == order_id:
14             return orders[mid]
15         elif orders[mid]["order_id"] < order_id:
16             left = mid + 1
17         else:
18             right = mid - 1
19     return None
20
21 print("Orders Sorted by Delivery Time:")
22 print(sorted(orders, key=lambda x: x["delivery_time"]))
23
24 print("\nSearch Order ID 502:")
```

Output

```
Orders Sorted by Delivery Time:
[{'order_id': 502, 'restaurant': 'XYZ', 'delivery_time': 20, 'price': 400},
 {'order_id': 501, 'restaurant': 'ABC', 'delivery_time': 30, 'price':
 250}, {'order_id': 503, 'restaurant': 'FoodHub', 'delivery_time': 45,
 'price': 150}]

Search Order ID 502:
{'order_id': 502, 'restaurant': 'XYZ', 'delivery_time': 20, 'price': 400}

=== Code Execution Successful ===
```