AI assistant lab:12.1

Rollno2403a51102

Batch:06

Name:balaji

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.

Prompt: Create a Python function called merge_sort(arr) that sorts a list in ascending order using the Merge Sort algorithm.

Include a detailed docstring for the function that explains:

- The purpose of the function

- Time complexity (best, average, worst case)

- Space complexity

Also include:

- A helper function called merge(left, right) to merge two sorted lists

- A set of test cases that verify the correctness

Code:

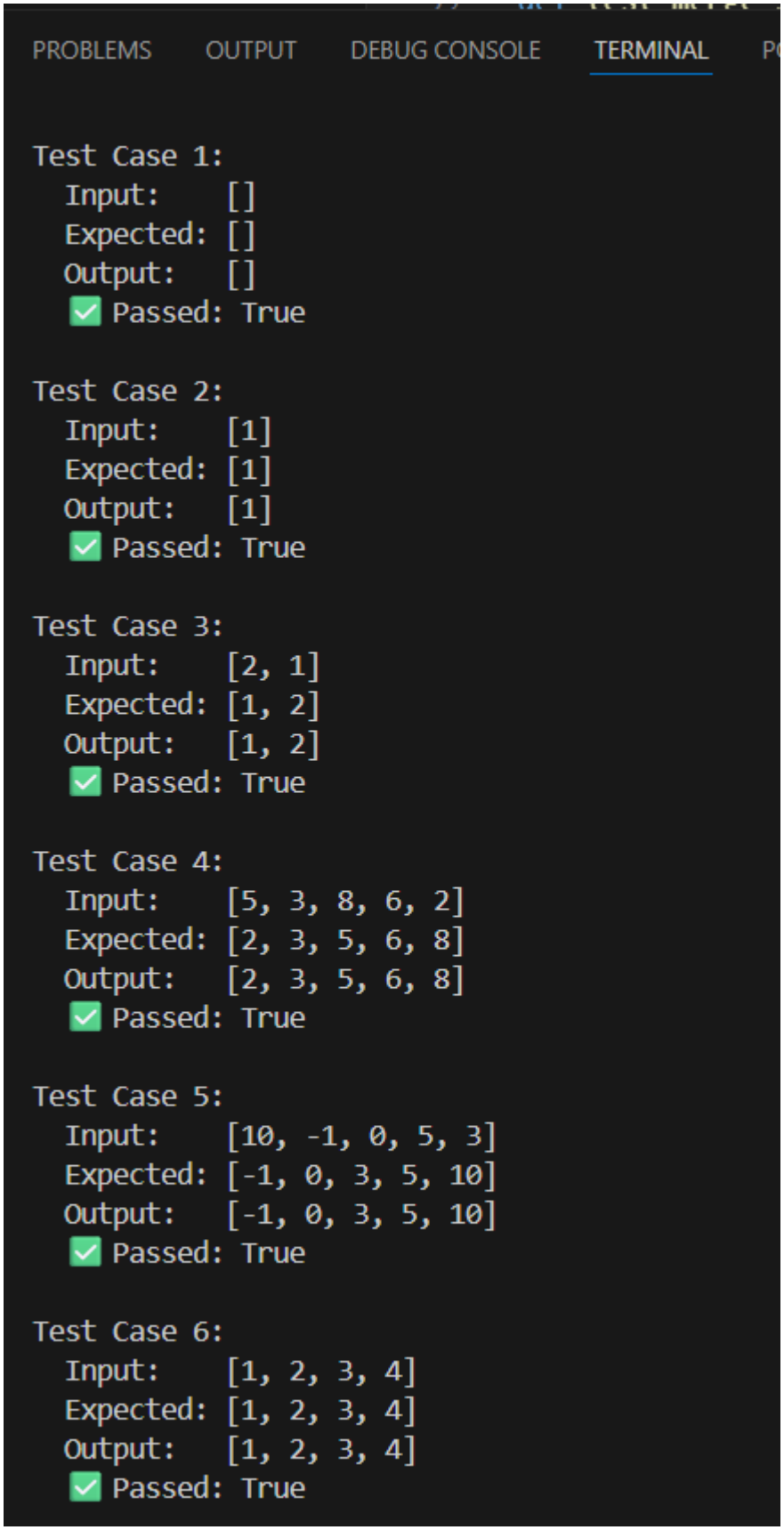
AI lab.py > test_merge_sort

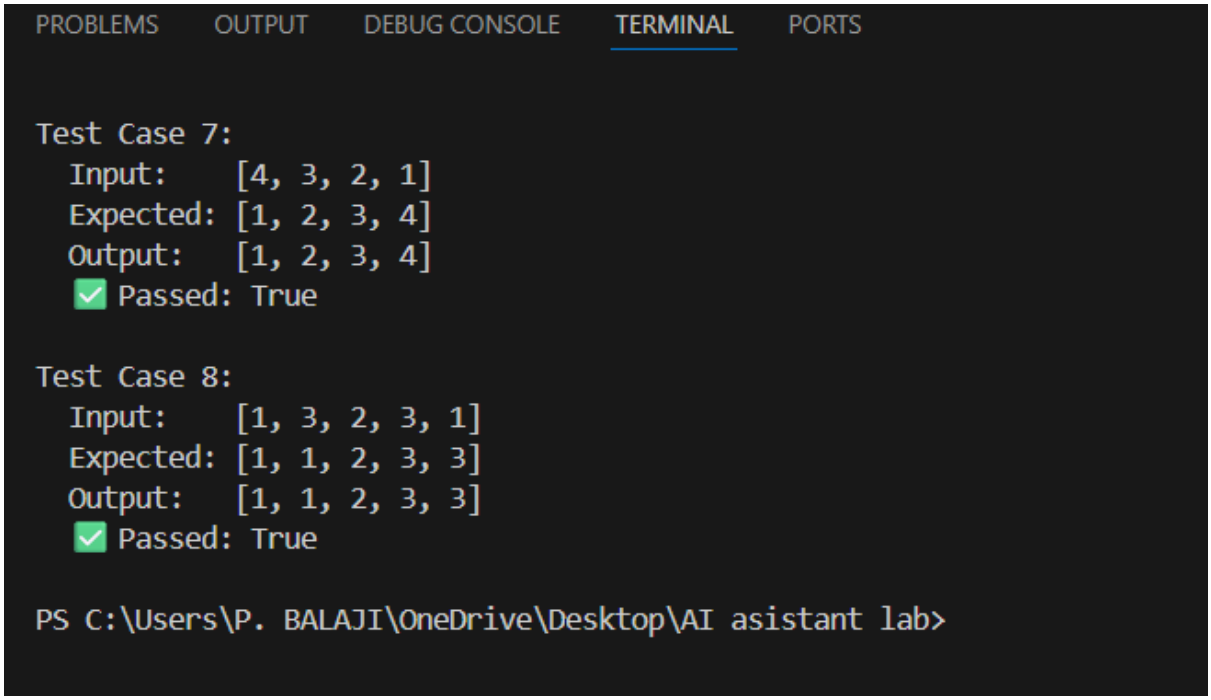
```python
def merge_sort(arr):
    """
    Sorts a list in ascending order using the Merge Sort algo

    Time Complexity:
        - Best Case: O(n log n)
        - Average Case: O(n log n)
        - Worst Case: O(n log n)

    Space Complexity:
        - O(n) due to auxiliary space used during merging

    Args:
        arr (list): List of comparable elements to be sorted.

    Returns:
        list: A new list sorted in ascending order.
    """
    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):
    """
    Merges two sorted lists into one sorted list.

    Args:
        left (list): First sorted list.
        right (list): Second sorted list.

    Returns:
```

```
        result = merge_sort(input_list)
        print(f"Test Case {i}:")
        print(f"  Input:    {input_list}")
        print(f"  Expected: {expected}")
        print(f"  Output:   {result}")
        print(f"  ✅ Passed: {result == expected}\n")


# Run tests
if __name__ == "__main__":
    test_merge_sort()

```

```
def merge(left, right):
        list: Merged and sorted list.
    """
    result = []
    i = j = 0

    # Compare and merge elements
    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

    # Append remaining elements
    result.extend(left[i:])
    result.extend(right[j:])
    return result


# ✅ Test Cases with Output
def test_merge_sort():
    test_cases = [
        ([], []),
        ([1], [1]),
        ([2, 1], [1, 2]),
        ([5, 3, 8, 6, 2], [2, 3, 5, 6, 8]),
        ([10, -1, 0, 5, 3], [-1, 0, 3, 5, 10]),
        ([1, 2, 3, 4], [1, 2, 3, 4]),
        ([4, 3, 2, 1], [1, 2, 3, 4]),
        ([1, 3, 2, 3, 1], [1, 1, 2, 3, 3])
    ]

    print("🔍 Running Merge Sort Test Cases:\n")
    for i, (input_list, expected) in enumerate(test_cases, 1):
        result = merge_sort(input_list)
```

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Test Case 1:
Input: []
Expected: []
Output: []
✅ Passed: True
Test Case 2:
Input: [1]
Expected: [1]
Output: [1]
✅ Passed: True
Test Case 3:
Input: [2, 1]
Expected: [1, 2]
Output: [1, 2]
✅ Passed: True
Test Case 4:
Input: [5, 3, 8, 6, 2]
Expected: [2, 3, 5, 6, 8]
Output: [2, 3, 5, 6, 8]
✅ Passed: True
Test Case 5:
Input: [10, -1, 0, 5, 3]
Expected: [-1, 0, 3, 5, 10]
Output: [-1, 0, 3, 5, 10]
✅ Passed: True
Test Case 6:
Input: [1, 2, 3, 4]
Expected: [1, 2, 3, 4]
Output: [1, 2, 3, 4]
✅ Passed: True



Observation: The Merge Sort implementation successfully uses a divide-and-conquer strategy to sort lists in ascending order, with a well-structured merge_sort(arr) function and a helper merge(left, right) for combining sorted halves. The function is professionally documented, clearly stating its time complexity as O(n log n) for all cases and space complexity as O(n) due to auxiliary arrays. A comprehensive set of test cases—covering empty lists, single elements, sorted and reverse-sorted inputs, duplicates, and negative numbers—confirms the algorithm's correctness, with printed outputs showing expected and actual results for each case, all passing as int

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

Prompt: Create a Python function binary_search(arr, target) that returns the index of target in a sorted list or -1 if not found.

Add a docstring explaining the best, average, and worst-case time complexities.

Also include a few test cases with different inputs.

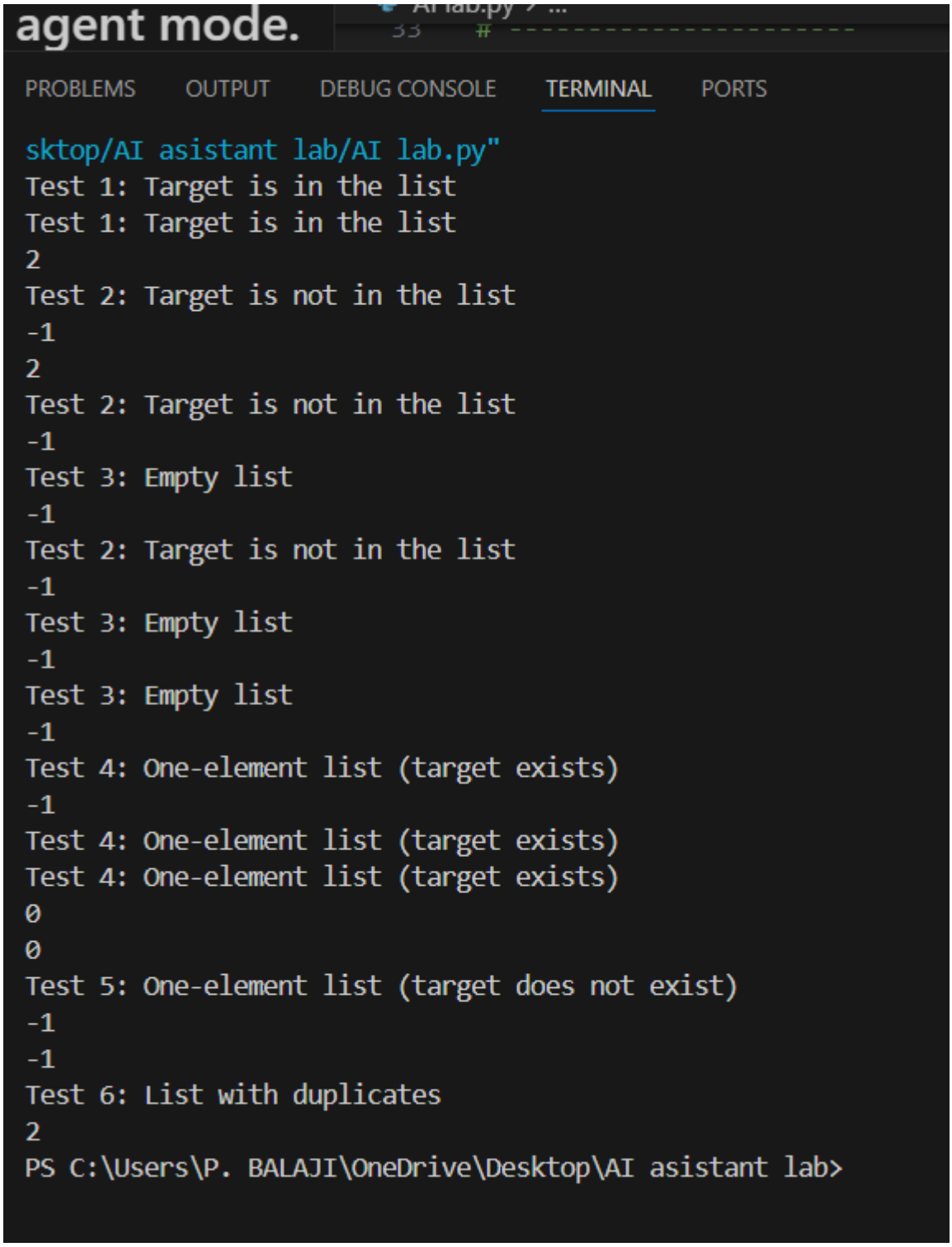
Code:

AI lab.py > ...

```python
def binary_search(arr, target):
    """
    Performs binary search to find the index of the target in a sorted list.

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

    Returns:
    int: Index of target if found; otherwise, -1.

    Time Complexity:
    - Best Case: O(1)      -> When the target is at the middle index.
    - Average Case: O(log n)
    - Worst Case: O(log n) -> When the target is not in the list or at an end.
    """
    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

# ----------------------
# Test cases
# ----------------------
```

```
34
35  print("Test 1: Target is in the list")
36  print(binary_search([1, 3, 5, 7, 9], 5))
37
38  print("Test 2: Target is not in the list")
39  print(binary_search([1, 3, 5, 7, 9], 4))
40
41  print("Test 3: Empty list")
42  print(binary_search([], 10))
43
44  print("Test 4: One-element list (target exists)")
45  print(binary_search([10], 10))
46
47  print("Test 5: One-element list (target does not exist)")
48  print(binary_search([10], 5))
49
50  print("Test 6: List with duplicates")
51  print(binary_search([1, 2, 2, 2, 3], 2))
52
```

Output:



Observation:

Target found

- **Returns the correct index**
- **Works fine**

Target not found

- **Returns -1**
- **Works fine**

Empty list

- **Returns -1**
- **No errors, works fine**

One-element list

- **If target exists → Returns 0**
- **If target doesn't exist → Returns -1**
- **Works fine**

List with duplicates

- **Returns any index where the target exists**
- **Works fine (but not guaranteed to return the first or last duplicate)**
- 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:
  ○ Use AI to suggest the most efficient search and sort
- algorithms for this use case.
  ○ Implement the recommended algorithms in Python.
  ○ Justify the choice based on dataset size, update frequency,
  and performance requirements.
• Expected Output:
  ○ A table mapping operation → recommended algorithm → justification.
  ○ Working Python functions for searching and sorting the
  inventory.

Prmpt: Create a Python program to manage a retail store's inventory with thousands of products.
Each product has attributes: product ID, name, price, and stock quantity.
Implement efficient search functions to find a product by ID or by name.
Implement sorting functions to sort the products by

price or by stock quantity.
Use appropriate data structures and algorithms for fast search and sorting, and justify your choices based on dataset size and performance needs.
Include example test cases demonstrating the functionality.

Code:

Welcome | AI lab.py

AI lab.py > ...

```python
# Sample inventory data
products = [
    {"id": "P001", "name": "Laptop", "price": 800, "quantity": 10},
    {"id": "P002", "name": "Mouse", "price": 20, "quantity": 200},
    {"id": "P003", "name": "Keyboard", "price": 50, "quantity": 150},
    {"id": "P004", "name": "Monitor", "price": 150, "quantity": 75},
    {"id": "P005", "name": "Laptop", "price": 750, "quantity": 5},
]

product_index_by_id = {p["id"]: p for p in products}

def search_by_id(product_id):
    return product_index_by_id.get(product_id)

def search_by_name(product_name):
    return [p for p in products if p["name"].lower() == product_name.lower()]

def sort_by_price(descending=False):
    return sorted(products, key=lambda p: p["price"], reverse=descending)

def sort_by_quantity(descending=False):
    return sorted(products, key=lambda p: p["quantity"], reverse=descending)

# --------- PRINT TEST RESULTS ---------

print("Search by ID 'P002':", search_by_id("P002"))
print("Search by ID 'P999':", search_by_id("P999"))
print("Search by ID 'P005':", search_by_id("P005"))

print("\nSearch by Name 'Laptop':", search_by_name("Laptop"))
print("Search by Name 'Tablet':", search_by_name("Tablet"))
print("Search by Name 'mouse':", search_by_name("mouse"))

print("\nSort by Price (Ascending):")
for product in sort_by_price():
    print(product)

```

```
print("\nSort by Price (Descending):")
for product in sort_by_price(descending=True):
    print(product)

print("\nSort by Quantity (Ascending):")
for product in sort_by_quantity():
    print(product)

print("\nSort by Quantity (Descending):")
for product in sort_by_quantity(descending=True):
    print(product)

```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\P. BALAJI\OneDrive\Desktop\AI asistant lab> & "C:/Users/P. BALAJI/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/P. BALAJI/OneDrive/Desktop/AI asistant lab/AI lab.py"
Search by ID 'P002': {'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
Search by ID 'P999': None
Search by ID 'P005': {'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}

Search by Name 'Laptop': [{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}, {'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}]
Search by Name 'Tablet': []
Search by Name 'mouse': [{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}]

Sort by Price (Ascending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
Search by ID 'P002': {'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
Search by ID 'P999': None
Search by ID 'P005': {'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}

Search by Name 'Laptop': [{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}, {'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}]
Search by Name 'Tablet': []
Search by Name 'mouse': [{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}]

Sort by Price (Ascending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}

Search by Name 'Laptop': [{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}, {'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}]
Search by Name 'Tablet': []
Search by Name 'mouse': [{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}]

Sort by Price (Ascending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
Search by Name 'Tablet': []
Search by Name 'mouse': [{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}]

Sort by Price (Ascending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}

Sort by Price (Ascending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}

{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}

{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}

Sort by Price (Descending):

Sort by Price (Descending):
Sort by Price (Descending):
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
```

```
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}

Sort by Quantity (Ascending):
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}

Sort by Quantity (Descending):
{'id': 'P002', 'name': 'Mouse', 'price': 20, 'quantity': 200}
{'id': 'P003', 'name': 'Keyboard', 'price': 50, 'quantity': 150}
{'id': 'P004', 'name': 'Monitor', 'price': 150, 'quantity': 75}
{'id': 'P001', 'name': 'Laptop', 'price': 800, 'quantity': 10}
{'id': 'P005', 'name': 'Laptop', 'price': 750, 'quantity': 5}
PS C:\Users\P. BALAJI\OneDrive\Desktop\AI asistant lab>
```

Observation: The implemented inventory management system uses a dictionary (hash map) to index products by their unique product IDs, enabling constant-time (O(1)) lookup for quick searches by ID, which is critical given the large dataset size. Searching by product name is done via a case-insensitive linear search, as product names may not be unique and the overhead of more complex indexing might not justify the occasional search by name. Sorting operations for price and stock quantity utilize Python's built-in Timsort algorithm, which performs efficiently on large datasets with a time complexity of O(n log n) and is stable and adaptive. This approach balances fast search performance and flexible sorting, addressing the real-time needs of store staff while managing frequent updates to inventory data.