

# **ASSIGNMENT-12.1**

## **AMGOTH VIKAS NAYAK**

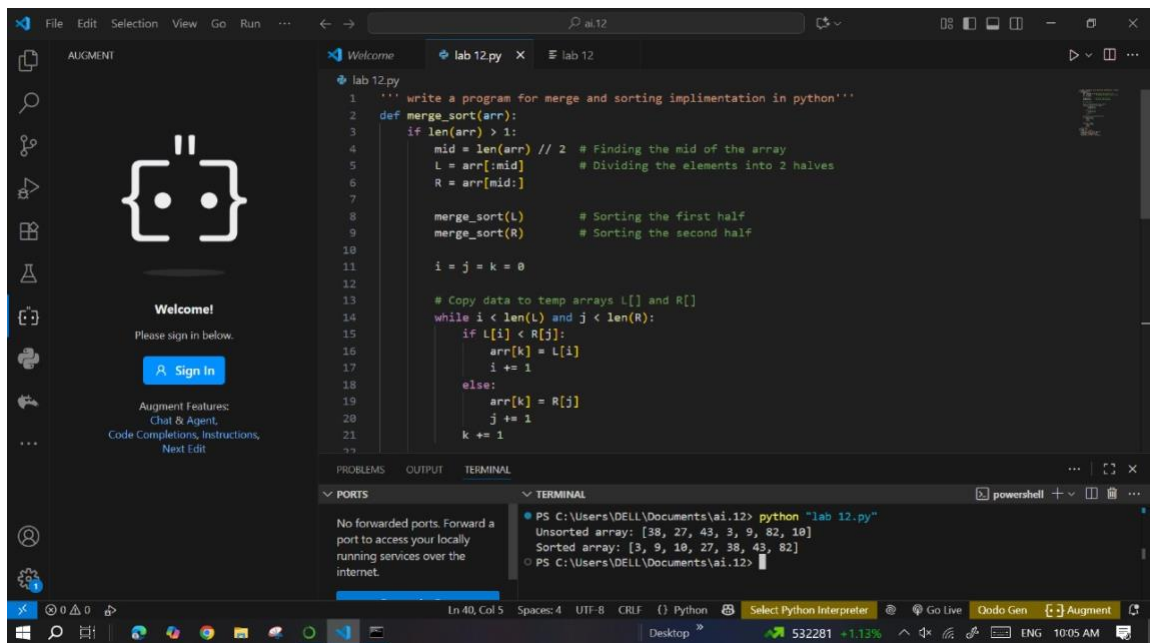
### **Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms**

#### **Lab Objectives:**

- \* Apply AI-assisted programming to implement and optimize sorting and searching algorithms.**
- \* Compare different algorithms in terms of efficiency and use**
- \* Understand how AI tools can suggest optimized code and complexity improvements.**

#### **Task Description #1 (Sorting – Merge Sort Implementation)**

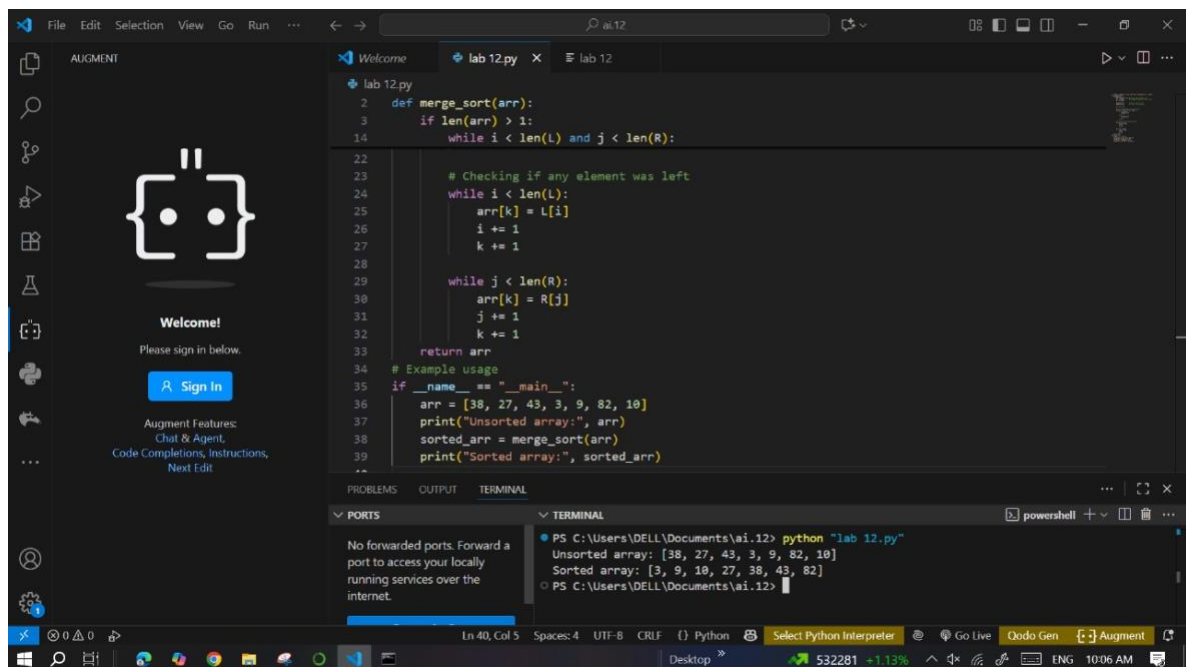
- \* Task: Use AI to generate a Python program that implements the Merge Sort algorithm.**
  - \* Instructions:**
    - o Prompt AI to create a function `merge_sort(arr)` that sorts a list in ascending order.**
    - o Ask AI to include time complexity and space complexity in the function docstring**
    - o Verify the generated code with test cases.**



```
1 ''' write a program for merge and sorting implimentation in python'''
2 def merge_sort(arr):
3     if len(arr) > 1:
4         mid = len(arr) // 2 # Finding the mid of the array
5         L = arr[:mid] # Dividing the elements into 2 halves
6         R = arr[mid:]
7
8         merge_sort(L) # Sorting the first half
9         merge_sort(R) # Sorting the second half
10
11     i = j = k = 0
12
13     # Copy data to temp arrays L[] and R[]
14     while i < len(L) and j < len(R):
15         if L[i] < R[j]:
16             arr[k] = L[i]
17             i += 1
18         else:
19             arr[k] = R[j]
20             j += 1
21         k += 1
```

Terminal Output:

```
PS C:\Users\DELL\Documents\ai.12> python "lab 12.py"
Unsorted array: [38, 27, 43, 3, 9, 82, 10]
Sorted array: [3, 9, 10, 27, 38, 43, 82]
```

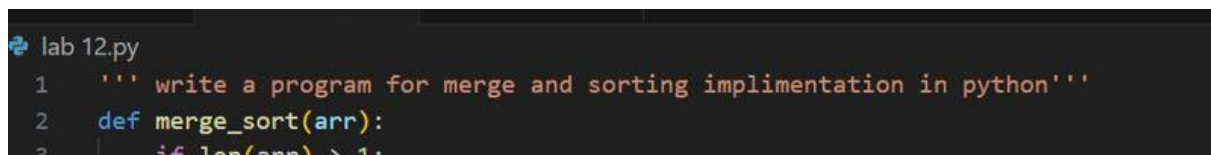


```
2 def merge_sort(arr):
3     if len(arr) > 1:
4         while i < len(L) and j < len(R):
5
6         # Checking if any element was left
7         while i < len(L):
8             arr[k] = L[i]
9             i += 1
10            k += 1
11
12         while j < len(R):
13             arr[k] = R[j]
14             j += 1
15             k += 1
16
17         return arr
18
19 # Example usage
20 if __name__ == "__main__":
21     arr = [38, 27, 43, 3, 9, 82, 10]
22     print("Unsorted array:", arr)
23     sorted_arr = merge_sort(arr)
24     print("Sorted array:", sorted_arr)
```

Terminal Output:

```
PS C:\Users\DELL\Documents\ai.12> python "lab 12.py"
Unsorted array: [38, 27, 43, 3, 9, 82, 10]
Sorted array: [3, 9, 10, 27, 38, 43, 82]
```

Prompt:



```
1 ''' write a program for merge and sorting implimentation in python'''
2 def merge_sort(arr):
3     if len(arr) > 1:
```

Expected output:

- o A functional Python script implementing Merge Sort with proper documentation

```
▼ TERMINAL
● PS C:\Users\DELL\Documents\ai.12> python "lab 12.py"
  Unsorted array: [38, 27, 43, 3, 9, 82, 10]
  Sorted array: [3, 9, 10, 27, 38, 43, 82]
○ PS C:\Users\DELL\Documents\ai.12> █
```

## 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:**

```
lab 12.py
''' write a python program for binary search fpr a target one from the sorted list'''
def binary_search(arr, target):
```

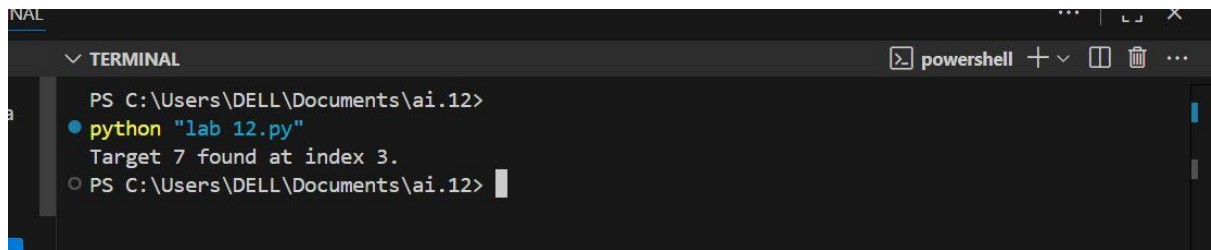
**Code:**

```
lab 12.py
1  ''' write a python program for binary search fpr a target one from the sorted list'''
2  def binary_search(arr, target):
3      left, right = 0, len(arr) - 1
4      while left <= right:
5          mid = (left + right) // 2
6          if arr[mid] == target:
7              return mid
8          elif arr[mid] < target:
9              left = mid + 1
10         else:
11             right = mid - 1
12     return -1
13
14 # Example usage
15 sorted_list = [1, 3, 5, 7, 9, 11, 13, 15]
16 target = 7
17 result = binary_search(sorted_list, target)
18 if result != -1:
19     print(f'Target {target} found at index {result}.')
20 else:
21     print(f'Target {target} not found in the list.')
22
23 # Output: Target 7 found at index 3.
```

**\* Expected Output:**

- o Python code implementing binary search with AI-

generated comments and docstrings.



```
PS C:\Users\DELL\Documents\ai.12>
python "lab 12.py"
Target 7 found at index 3.
PS C:\Users\DELL\Documents\ai.12>
```

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

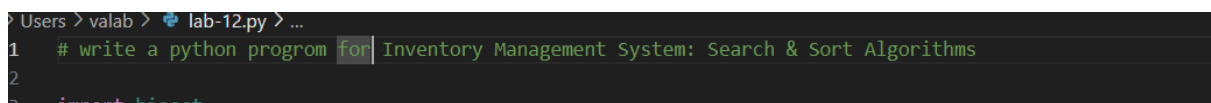
**\* Scenario: A retail store’s inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:**

- 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.**

**Prompt:**



```
> Users > valab > lab-12.py > ...
1 # write a python program for Inventory Management System: Search & Sort Algorithms
2
3 import bisect
```

```
# Inventory Management System: Search & Sort Algorithms

import bisect

# Sample inventory: list of dicts
inventory = [
    {"id": 101, "name": "Apple", "price": 0.5, "quantity": 120},
    {"id": 102, "name": "Banana", "price": 0.3, "quantity": 200},
    {"id": 103, "name": "Orange", "price": 0.7, "quantity": 150},
    {"id": 104, "name": "Milk", "price": 1.2, "quantity": 80},
    {"id": 105, "name": "Bread", "price": 1.0, "quantity": 60},
]

def build_index(inventory):
    id_map = {item["id"]: item for item in inventory}
    name_map = {item["name"].lower(): item for item in inventory}
    return id_map, name_map

id_map, name_map = build_index(inventory)

def search_by_id(product_id):
    """Search product by ID using hash map."""
    return id_map.get(product_id)

def search_by_name(product_name):
    """Search product by name using hash map (case-insensitive)."""
    return name_map.get(product_name.lower())

def sort_by_price(inventory, reverse=False):
```

```

def sort_by_price(inventory, reverse=False):
    """Sort inventory by price using Timsort."""
    return sorted(inventory, key=lambda x: x["price"], reverse=reverse)

def sort_by_quantity(inventory, reverse=False):
    return sorted(inventory, key=lambda x: x["quantity"], reverse=reverse)

# --- Test Cases ---
def test_search_by_id():
    assert search_by_id(101)["name"] == "Apple"
    assert search_by_id(105)["price"] == 1.0
    assert search_by_id(999) is None

def test_search_by_name():
    assert search_by_name("banana")["id"] == 102
    assert search_by_name("MILK")["quantity"] == 80
    assert search_by_name("unknown") is None

def test_sort_by_price():
    sorted_inv = sort_by_price(inventory)
    assert sorted_inv[0]["name"] == "Banana"
    assert sorted_inv[-1]["name"] == "Milk"
    sorted_inv_desc = sort_by_price(inventory, reverse=True)
    assert sorted_inv_desc[0]["name"] == "Milk"

def test_sort_by_quantity():
    sorted_inv = sort_by_quantity(inventory)
    assert sorted_inv[0]["name"] == "Bread"

```

```

Al.py lab-12.py
C: > Users > valab > lab-12.py > sort_by_quantity
48 def test_sort_by_price():
49     sorted_inv = sort_by_price(inventory)
50     assert sorted_inv[0]["name"] == "Banana"
51     assert sorted_inv[-1]["name"] == "Milk"
52     sorted_inv_desc = sort_by_price(inventory, reverse=True)
53     assert sorted_inv_desc[0]["name"] == "Milk"
54
55 def test_sort_by_quantity():
56     sorted_inv = sort_by_quantity(inventory)
57     assert sorted_inv[0]["name"] == "Bread"
58     assert sorted_inv[-1]["name"] == "Banana"
59     sorted_inv_desc = sort_by_quantity(inventory, reverse=True)
60     assert sorted_inv_desc[0]["name"] == "Banana"
61
62 # --- Run Tests ---
63 if __name__ == "__main__":
64     test_search_by_id()
65     test_search_by_name()
66     test_sort_by_price()
67     test_sort_by_quantity()
68     print("All tests passed.")
69
70

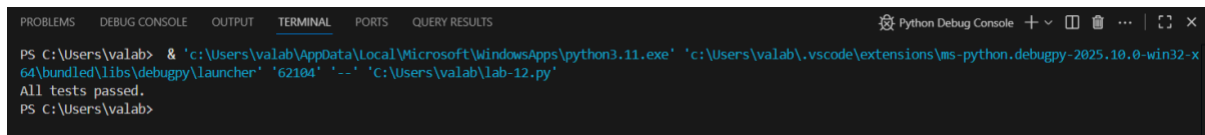
```

**\* Expected Output:**

- o A table mapping operation → recommended algorithm → justification.**
- o Working Python functions for searching and sorting the inventory.**

**Deliverables (For All Tasks)**

- 1. AI-generated prompts for code and test case generation.**
- 2. At least 3 assert test cases for each task.**
- 3. AI-generated initial code and execution screenshots.**
- 4. Analysis of whether code passes all tests.**
- 5. Improved final version with inline comments and explanation.**
- 6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.**



The image shows a terminal window from a code editor. The title bar at the top reads 'Python Debug Console'. The terminal content shows a PowerShell command being executed: `PS C:\Users\valab> & 'c:\Users\valab\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\valab\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '62104' '-...' 'C:\Users\valab\lab-12.py'`. The output of the command is `All tests passed.`. The prompt `PS C:\Users\valab>` is visible at the bottom.