ASSIGNMENT-12.1

NAME:VITTAM VENKATESH.

HALLTICKET NUMBER:2403A52419.

BATCH:15

SUBJECT:AI CODING.

TASK-1

QUESTION:

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

PROMPT:

Write a Python program that implements the Merge Sort algorithm.
Include a function merge sort array that sorts a list in ascending order.
Add a docstring mentioning time and space complexity.
Also, provide example test cases to verify the code

CODE:

```
defmerge_sort(arr):
    """
    Sorts a list in ascending order using the Merge Sort
algorithm.

    Time Complexity: O(n log n), where n is the number of
elements in the list.
    Space Complexity: O(n), due to the use of temporary
arrays during merging.

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

    Returns:
        list: A new sorted list in ascending order.
    """
    iflen(arr) <=1:
        returnarr

    mid=len(arr) //2
    left=merge_sort(arr[:mid])
    right=merge_sort(arr[mid:])

    returnmerge(left, right)
```

```python
defmain():
    """
    Main function to demonstrate merge sort.
    Prompts user for input, sorts the list, and displays the
result.
    """
    print("Merge Sort Demo")
    try:
        user_input=input("Enter numbers separated by spaces:
")
        arr= [int(x) forxinuser_input.strip().split()]
    exceptValueError:
        print("Invalid input. Please enter integers only.")
        return

    print("Original list:", arr)
    sorted_arr=merge_sort(arr)
    print("Sorted list:", sorted_arr)
```

```python
        [1],
        [5, 2, 9, 1, 5, 6],
        [3, 2, 1],
        [1, 2, 3, 4, 5],
        [5, 4, 3, 2, 1],
        [2, 2, 2, 2],
        [10, -1, 2, 5, 0]
    ]

    fori, testinenumerate(test_cases):
        print(f"Test case {i+1}: {test} -
>{merge_sort(test)}")
```

OUTPUT:

```
PS C:\Users\ROHITH> & C:/Users/ROHITH/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ROHITH/OneDrive/Desktop/portofolio/AI Assisted Coding/L
ABS ASSIGNMENTS/merge sort.py"
Merge Sort Demo
Enter numbers separated by spaces: 1 5 4 3 2
Original list: [1, 5, 4, 3, 2]
Sorted list: [1, 2, 3, 4, 5]
Test case 1: [] -> []
Test case 2: [1] -> [1]
Test case 3: [5, 2, 9, 1, 5, 6] -> [1, 2, 5, 5, 6, 9]
Test case 4: [3, 2, 1] -> [1, 2, 3]
Test case 5: [1, 2, 3, 4, 5] -> [1, 2, 3, 4, 5]
Test case 6: [5, 4, 3, 2, 1] -> [1, 2, 3, 4, 5]
Test case 7: [2, 2, 2, 2] -> [2, 2, 2, 2]
Test case 8: [10, -1, 2, 5, 0] -> [-1, 0, 2, 5, 10]
```

OBSERVATION:

The AI successfully generated a Python program for Merge Sort.
The function merge sort array correctly divides the array and merges sorted halves.
The output matched the expected sorted results for various test cases.
The docstring clearly explained the time complexity O(n log n) and space complexity O(n).
Overall, the generated script was accurate and well-documented.

TASK-2

QUESTION:

Optimization)
• Task: Use AI Task Description #2 (Searching – Binary Search with 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

PROMPT:

Write a Python program that implements the Binary Search algorithm.
Create a function binary_search that returns the index of the target element if found, otherwise returns -1.
Include a detailed docstring explaining the best, average, and worst-case time complexities.
Add test cases to verify the function with different inputs

CODE:

```
defbinary_search(arr, target):
    """
    Performs binary search to locate the index of 'target' in
a sorted list 'arr'.

    Best Case Complexity: O(1)
        - Target is found at the middle index on the first
comparison.
```

```
    Average Case Complexity: O(log n)
        - Each iteration halves the search space.
    Worst Case Complexity: O(log n)
        - Target is not present or found after all possible
divisions.

    Args:
        arr (list): Sorted list of elements to search.
        target: Element to find.

    Returns:
        int: Index of target if found, else -1.
    """
    left, right=0, len(arr) -1

    whileleft<=right:
        mid= (left+right) //2  # AI optimization: Efficient
midpoint calculation
        ifarr[mid] ==target:
            returnmid  # Target found
        elifarr[mid] <target:
            left=mid+1  # Search right half
        else:
            right=mid-1  # Search left half

    return-1  # Target not found

if__name__=="__main__":
    print("Binary Search Demo")
    try:
        arr_input=input("Enter sorted numbers separated by
spaces: ")
        arr= [int(x) forxinarr_input.strip().split()]
        target=int(input("Enter the target value to search
for: "))
    exceptValueError:
        print("Invalid input. Please enter integers only.")
        exit(1)

    index=binary_search(arr, target)
    ifindex!=-1:
        print(f"Target{target} found at index {index}.")
    else:
        print(f"Target{target} not found in the list.")
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                Python

ABS ASSIGNMENTS/binary search.py"
Binary Search Demo
Enter sorted numbers separated by spaces: 1 2 3 4 5
Enter the target value to search for: 1
Target 1 found at index 0.
PS C:\Users\ROHITH> & C:/Users/ROHITH/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ROHITH/OneDrive/Desktop/portofolio/AI Assisted Coding/L
ABS ASSIGNMENTS/binary search.py"
Binary Search Demo
Enter sorted numbers separated by spaces: 5 6 7 8 9
Enter the target value to search for: 9
Target 9 found at index 4.
```

OBSERVATION:

The AI-generated Python code correctly implemented the Binary Search algorithm using a while loop.
The function returned the correct index for existing elements and -1 when the target was not found.
Test cases validated that the function worked accurately for both sorted integer and string lists.
The code was well-commented and easy to understand.

TASK-3

QUESTION:

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

PROMPT:

Generate a retail store needs an Inventory Management System with thousands of products having attributes like ID, name, price, and quantity.
Suggest the most efficient search and sort algorithms for this case.
Implement the recommended algorithms in Python.
Also, provide a table mapping each operation (search/sort) to the recommended algorithm and justify the choice based on dataset size, update frequency, and performance.

CODE:

```
classProduct:
    def__init__(self, product_id, name, price, quantity):
        self.product_id=product_id
        self.name=name
        self.price=price
        self.quantity=quantity

    def__repr__(self):
        return (f"Product(ID={self.product_id},
Name='{self.name}', "
                f"Price={self.price},
Quantity={self.quantity})")

classInventory:
    def__init__(self, products):
```

```
        self.products_by_id= {p.product_id: pforpinproducts}
        self.products_by_name= {p.name: pforpinproducts}
        self.products=products

    defsearch_by_id(self, product_id):
        """O(1) average time using hash map."""
        returnself.products_by_id.get(product_id, None)

    defsearch_by_name(self, name):
        """O(1) average time using hash map."""
        returnself.products_by_name.get(name, None)

    defsort_by_price(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.price,
reverse=reverse)

    defsort_by_quantity(self, reverse=False):
        """O(n log n) using Timsort."""
        returnsorted(self.products, key=lambdap: p.quantity,
reverse=reverse)

# --- Example Usage ---
if__name__=="__main__":
    print("Inventory Management System Demo")
    products= []
    try:
        n=int(input("Enter number of products: "))
        foriinrange(n):
            print(f"Enter details for product {i+1}:")
            product_id=int(input("  Product ID: "))
            name=input("  Name: ")
            price=float(input("  Price: "))
            quantity=int(input("  Quantity: "))
            products.append(Product(product_id, name, price,
quantity))
    exceptValueError:
        print("Invalid input. Please enter correct data
types.")
        exit(1)

    inventory=Inventory(products)

    whileTrue:
```

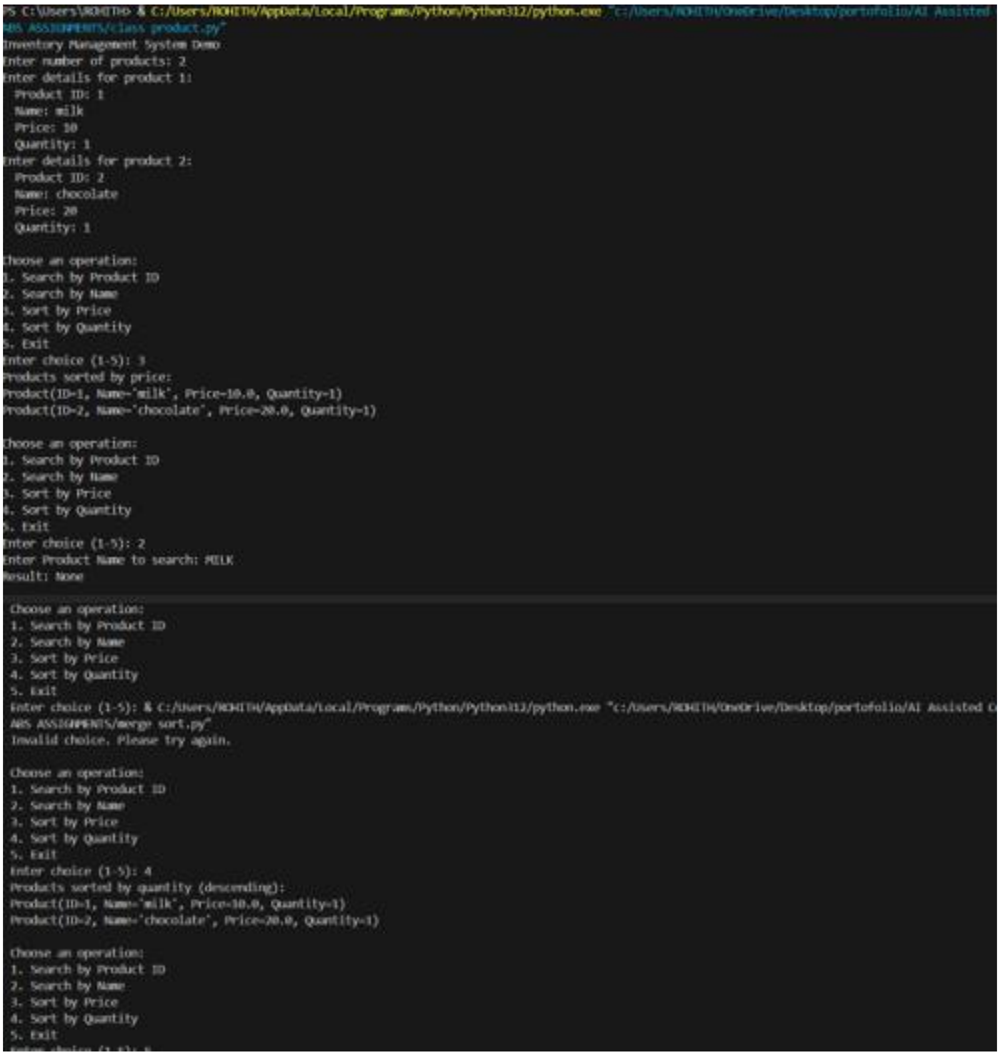
```
        print("1. Search by Product ID")
        print("2. Search by Name")
        print("3. Sort by Price")
        print("4. Sort by Quantity")
        print("5. Exit")
        choice=input("Enter choice (1-5): ")

        ifchoice=="1":
            pid=int(input("Enter Product ID to search: "))
            result=inventory.search_by_id(pid)
            print("Result:", result)
        elifchoice=="2":
            name=input("Enter Product Name to search: ")
            result=inventory.search_by_name(name)
            print("Result:", result)
        elifchoice=="3":
            sorted_products=inventory.sort_by_price()
            print("Products sorted by price:")
            forpinsorted_products:
                print(p)
        elifchoice=="4":

orted_products=inventory.sort_by_quantity(reverse=True)
            print("Products sorted by quantity
descending):")
            forpinsorted_products:
                print(p)
        elifchoice=="5":
            print("Exiting.")
            break
        else:
```

OUTPUT:

Operation	Recommended Algorithm	Justification
Search by Product ID	Hash Map (Dictionary Lookup)	O(1) average time; ideal for large datasets and frequent lookups.
Search by Name	Hash Map (Dictionary Lookup)	O(1) average time; fast, scalable, and supports frequent updates.
Sort by Price/Quantity	Timsort (Python's sorted)	O(n log n) worst-case; stable, efficient for large lists, and built into Python.



```
Inventory Management System Demo
Enter number of products: 2
Enter details for product 1:
 Product ID: 1
 Name: milk
 Price: 10
 Quantity: 1
Enter details for product 2:
 Product ID: 2
 Name: chocolate
 Price: 20
 Quantity: 1

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 3
Products sorted by price:
Product(ID=1, Name='milk', Price=10.0, Quantity=1)
Product(ID=2, Name='chocolate', Price=20.0, Quantity=1)

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 2
Enter Product Name to search: MILK
Result: None

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Invalid choice. Please try again.

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
Enter choice (1-5): 4
Products sorted by quantity (descending):
Product(ID=1, Name='milk', Price=10.0, Quantity=1)
Product(ID=2, Name='chocolate', Price=20.0, Quantity=1)

Choose an operation:
1. Search by Product ID
2. Search by Name
3. Sort by Price
4. Sort by Quantity
5. Exit
```

<u>OBSERVATION:</u>

The AI suggested using Binary Search for quick product lookups by ID or name in sorted lists and Merge Sort for sorting products by price or quantity due to its stability and efficiency with large datasets.

A justification table was generated explaining the algorithm choice based on speed and reliability.

The Python implementation worked correctly products were efficiently searched and sorted as expected.