AI ASSISTED CODING LAB

ASSIGNMENT 12.2

ENROLLMENT NO :2503A51L26

BATCH NO: 19

NAME: V.VISHNU VARDHAN

TASK1

TASK1 DESCRIPTION:-

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

PROMPT:-

Generate a python program with merge_sort(art) implementing Merge Sort, include docstring with time/space complexity, do not modify input, and add simple tests/examples under __main__.

CODE:-

```python
t1.py > ...
def merge_sort(art):
    """...
    if len(art) <= 1:
        return art[:]  # return a shallow copy

    def merge(left, right):
        i = j = 0
        merged = []
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1
        # append remaining elements
        if i < len(left):
            merged.extend(left[i:])
        if j < len(right): ...
        return merged

    mid = len(art) // 2
    left_sorted = merge_sort(art[:mid])
    right_sorted = merge_sort(art[mid:])
    return merge(left_sorted, right_sorted)


if __name__ == "__main__":
    # Basic test cases to verify correctness
    tests = [...

    for t in tests:
        result = merge_sort(t)
        expected = sorted(t)
        assert result == expected, f"merge_sort({t}) -> {result}, expected {expected}"
    print("All tests passed. Examples:")

    examples = [
        [9, 7, 5, 3, 1, 2, 4, 6, 8],
        [42, 42, 1, 0, -5],
    ]
    for ex in examples:
        print(f"input: {ex} -> output: {merge_sort(ex)}")
```

OUTPUT:-

```
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2> & C:/Users/khaja/anaconda3/python.exe
Screenshots/cyc/New folder/12.2/t1.py"
All tests passed. Examples:
input: [9, 7, 5, 3, 1, 2, 4, 6, 8] -> output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
input: [42, 42, 1, 0, -5] -> output: [-5, 0, 1, 42, 42]
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2>
```

OBSERVATION:-

In this task AI generated merge_sort(art) along with a clear docstring that explains the algorithm and states time (O(n log n)) and space (O(n)) complexity. The implementation returns a new sorted list without modifying the input and includes basic tests and example prints to verify correctness (empty list, single element, duplicates, negatives). This demonstrates how AI can quickly produce both working logic and useful documentation, plus ready-made tests so the implementation can be validated immediately.

TASK2

TASK2 DESCRIPTION:-

- Use AI to create a binary search function that finds a target element in a sorted list.
- Instructions:
    - Prompt AI to create a function binary search (art, target) returning the index of the target or -1 if not found.
    - Include docstrings explaining best, average, and worst-case complexities.
    - Test with various inputs.

PROMPT:-

Create a Python program that implements binary_search(art, target): the function should assume art is sorted in ascending order and return the index of target or -1 if not found, include a docstring explaining the algorithm and best/average/worst-case time complexities and space complexity, use an iterative approach, and include varied tests/examples under if **name** == "**main**" to validate behavior.

CODE:-

```python
t2.py > ...
def binary_search(art, target):
    """...
    left, right = 0, len(art) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if art[mid] == target:
            return mid
        if art[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1


if __name__ == "__main__":
    # Test cases
    tests = [
        ([], 3, -1),                    # empty list
        ([1], 1, 0),                    # single element found
        ([1], 2, -1),                   # single element not found
        ([1, 2, 3, 4, 5], 3, 2),        # middle element
        ([1, 2, 3, 4, 5], 6, -1),       # not present
        ([-3, -1, 0, 2, 4], -1, 1),     # negatives and positives
        ([1, 2, 2, 2, 3], 2, "any"),    # duplicates (any matching index is acceptable)
    ]

    for arr, tgt, expected in tests:
        idx = binary_search(arr, tgt)
        if expected == "any":
            assert idx != -1 and arr[idx] == tgt, f"binary_search({arr}, {tgt}) -> {idx}, expected any index with value {tgt}"
        else:
            assert idx == expected, f"binary_search({arr}, {tgt}) -> {idx}, expected {expected}"

    print("All tests passed. Examples:")

    examples = [
        ([1, 3, 5, 7, 9], 7),
        ([0, 2, 4, 6, 8], 1),
        ([10, 20, 30, 40], 25),
    ]
    for arr, tgt in examples:
        print(f"input: {arr}, target: {tgt} -> index: {binary_search(arr, tgt)}")
```

OUTPUT:-

```
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2> & C:/Users/khaja/anaconda3/python.exe
Screenshots/cyc/New folder/12.2/t2.py"
All tests passed. Examples:
input: [1, 3, 5, 7, 9], target: 7 -> index: 3
input: [0, 2, 4, 6, 8], target: 1 -> index: -1
input: [10, 20, 30, 40], target: 25 -> index: -1
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2>
```

OBSERVATION:-

AI produced an iterative binary_search(art, target) that assumes a sorted input and returns the index or -1; the docstring lists best/average/worst time complexities and space complexity (best O(1), average/worst O(log n), space O(1)). The file includes tests covering empty arrays, single-element cases, not-found cases and duplicates, making it straightforward to confirm correctness. The result shows AI speeds up development by supplying a concise, well-documented, and testable search routine.

TASK3

TASK3 DESCRIPTION:-

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.

PROMPT:-

Create a Python program with a Product dataclass (product_id, name, price, quantity) and an Inventory class implementing add_product, find_by_id, find_by_name(name, exact=True/False), sort_by_price, and sort_by_quantity; document complexity notes (O(1) id lookup via dict, O(n) substring search, O(n log n) sorting), handle duplicate IDs/names sensibly, and include an example dataset plus assertions and demonstration prints under if **name** == "**main**".

CODE:-

t3.py > ...

```python
from dataclasses import dataclass
from typing import List, Optional, Dict


@dataclass(frozen=True)
class Product:
    product_id: str
    name: str
    price: float
    quantity: int


class Inventory:
    """
    Inventory supporting:
    - O(1) lookup by product ID using a dict index.
    - O(1) exact-name lookup (if unique) using a name->list dict.
    - substring name search in O(n) time.
    - sorting by price or quantity in O(n log n) time.

    Space complexity: O(n) for stored products and indexes.
    """

    def __init__(self, products: Optional[List[Product]] = None):
        self._products: List[Product] = []
        self._id_index: Dict[str, Product] = {}
        self._name_index: Dict[str, List[Product]] = {}
        if products:
            for p in products:
                self.add_product(p)

    def add_product(self, product: Product) -> None:
        """Add product and update indexes. If ID exists, replace product."""
        # replace in list if exists
        if product.product_id in self._id_index:
            # remove old instance from list and name index
            old = self._id_index[product.product_id]
            try:
                self._products.remove(old)
            except ValueError:
                pass
            lname = old.name.lower()
            self._name_index.get(lname, []).remove(old)

        self._products.append(product)
```

t3.py > ...

```python
class Inventory:
            return sorted(self._products, key=lambda p: p.quantity, reverse=descending)


if __name__ == "__main__":
    # Example dataset
    samples = [
        Product("P001", "USB Cable", 3.99, 150),
        Product("P002", "Wireless Mouse", 15.49, 40),
        Product("P003", "Keyboard", 22.0, 25),
        Product("P004", "HDMI Cable", 7.5, 80),
        Product("P005", "USB-C Adapter", 5.25, 60),
        Product("P006", "Wireless Mouse", 17.99, 10),  # duplicate name, different ID
    ]

    inv = Inventory(samples)

    # Search by ID
    p = inv.find_by_id("P003")
    assert p is not None and p.name == "Keyboard"

    # Exact name search (multiple results possible)
    mice = inv.find_by_name("Wireless Mouse", exact=True)
    assert len(mice) == 2 and all(m.name == "Wireless Mouse" for m in mice)

    # Partial name search
    usb_items = inv.find_by_name("usb", exact=False)
    assert len(usb_items) >= 2 and all("usb" in it.name.lower() for it in usb_items)

    # Sort by price ascending
    by_price = inv.sort_by_price()
    prices = [p.price for p in by_price]
    assert prices == sorted(prices)

    # Sort by quantity descending
    by_qty_desc = inv.sort_by_quantity(descending=True)
    qtys = [p.quantity for p in by_qty_desc]
    assert qtys == sorted(qtys, reverse=True)

    # Demonstration prints
    print("Find by ID P004 ->", inv.find_by_id("P004"))
    print("Exact name 'Wireless Mouse' ->", inv.find_by_name("Wireless Mouse"))
    print("Partial name 'usb' ->", inv.find_by_name("usb", exact=False))
    print("Sorted by price (asc) ->", [(p.product_id, p.price) for p in by_price])
    print("Sorted by quantity (desc) ->", [(p.product_id, p.quantity) for p in by_qty_desc])
```

OUTPUT:-

```
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2> & C:/Users/khaja/anaconda3/python.exe "c:/Users/khaja/OneDrive/Pictures/
Screenshots/cyc/New folder/12.2/t3.py"
Find by ID P004 -> Product(product_id='P004', name='HDMI Cable', price=7.5, quantity=80)
Exact name 'Wireless Mouse' -> [Product(product_id='P002', name='Wireless Mouse', price=15.49, quantity=40), Product(product_id='P006', name=
'Wireless Mouse', price=17.99, quantity=10)]
Partial name 'usb' -> [Product(product_id='P001', name='USB Cable', price=3.99, quantity=150), Product(product_id='P005', name='USB-C Adapter
', price=5.25, quantity=60)]
Sorted by price (asc) -> [('P001', 3.99), ('P005', 5.25), ('P004', 7.5), ('P002', 15.49), ('P006', 17.99), ('P003', 22.0)]
Sorted by quantity (desc) -> [('P001', 150), ('P004', 80), ('P005', 60), ('P002', 40), ('P003', 25), ('P006', 10)]
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\12.2>
```

OBSERVATION:-

AI created a Product dataclass and an Inventory class with add_product, find_by_id, find_by_name (exact and substring), sort_by_price, and sort_by_quantity, and documented the complexity and design decisions (dict for O(1) ID lookup, name->list index, substring search O(n), sorting O(n log n)). An example dataset and assertions verify behavior including duplicate names and ID replacement. This highlights how AI can scaffold a small production-like module with indexing, documentation, and tests, enabling quick validation and iteration.