LAB TEST -3

NAME :SUMITH

ROLL NO:2403A510A3

BATCH:06

Task 1: Basic Algorithm Improvement

Prompt

“Improve the following algorithm using AI assistance. Make it faster, remove unnecessary steps, describe runtime expectations, edge case handling, and clarify constraints.”

Code (Before & After Improvement)

✅ Original Code

```
def find_positive_numbers(nums):
    result = []
    for n in nums:
        if n > 0:
            result.append(n)
        else:
            pass  # unnecessary step
    return result
```

✅ Improved AI-Assisted Code

```
def find_positive_numbers(nums):
    # Constraints: nums must be an iterable of integers/floats
    # Edge cases handled: None, empty list, non-numeric entries
    if nums is None:
        return []
    result = []
    for n in nums:
        if isinstance(n, (int, float)) and n > 0:
            result.append(n)
    return result
```

Output

```
Input: [3, -1, 0, 7, -5]
Output: [3, 7]
```

Observation

AI removed unnecessary else branch, added input validation, clarified constraints, and improved readability.
Runtime remains **O(n)** but with better robustness and edge-case handling.

✅ Task 2: AI-Assisted Bubble Sort Optimization

Prompt

"Optimize bubble sort using AI assistance. Implement early-stop detection using a 'swapped' flag and explain time complexity."

Code

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False

        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        # Early stop: break if no swaps happened
        if not swapped:
            break
    return arr
```

Output

```
Input: [5, 1, 4, 2, 8]

Output: [1, 2, 4, 5, 8]
```

Observation

AI added the **early-stop optimization**, improving:

- Best-case time from **$O(n^2)$** → **$O(n)$**
- Average and worst-case remain **$O(n^2)$**
  Runs fewer passes when list becomes sorted early.

✅ Task 3: Refactoring Duplicate Code

Prompt

“Here are repeated code blocks. Refactor them into reusable functions using AI assistance.”

Original Duplicate Code

```
print("User:", name)
print("Age:", age)


print("User:", name2)
print("Age:", age2)
```

Refactored Code

```
def display_user(name, age):
    print("User:", name)
    print("Age:", age)


display_user("Ravi", 22)

display_user("Megha", 24)
```

Output

```
User: Ravi

Age: 22

User: Megha

Age: 24
```

Observation

AI replaced duplicate blocks with a single reusable function.
This improves:

- Maintainability
- Readability
- Scalability (add more users easily)

✅ Task 4: AI-Assisted Error Handling

Prompt

“Add proper error handling for invalid inputs, missing values, and incorrect types using AI assistance.”

Code

```
def safe_division(a, b):

    try:

        if a is None or b is None:

            raise ValueError("Values cannot be None")

        if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):

            raise TypeError("Inputs must be numeric")
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero"
    except Exception as e:
        return f"Error: {e}"
```

Output

```
Input: safe_division(10, 2)
Output: 5.0

Input: safe_division(10, 0)
Output: Error: Division by zero

Input: safe_division("a", 2)
Output: Error: Inputs must be numeric
```

Observation

AI added:

- Type validation
- None checks
- Try-except blocks
- User-friendly error messages

The program now handles all tested error scenarios gracefully without crashing.