**ENROLLNO:2503A51L33**

**NAME: T. SHIVA**

**ASSIGNMENT 12**

**TASK 1:**

Write a Python implementation of Bubble Sort.

**PROMPT:**

Write a Python program for Bubble Sort with AI-generated inline comments explaining each step.  
Also, include time complexity analysis at the end.

**CODE:**

A computer screen shot of a program code

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

**OUTPUT:**

**A black screen with blue text

AI-generated content may be incorrect.**

**OBSERVATION:**

The Bubble Sort algorithm repeatedly compares adjacent elements and swaps them if they are in the wrong order.

After each pass, the largest element moves (“bubbles up”) to its correct position at the end of the list.

The number of comparisons decreases with each pass.

**TASK 2:**

Provide Bubble Sort code to AI and ask it to suggest a  
more efficient algorithm for partially sorted arrays.

**PROMPT:**

Write a single Python program that implements **Bubble Sort** and **Insertion Sort**.  
Compare their performance on a **nearly sorted array** and display which algorithm is faster.  
Add clear **inline comments** and an **observation** at the end.

**CODE:**

A computer screen shot of a program code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**OUTPUT:**

**A black screen with blue and yellow text

AI-generated content may be incorrect.**

**OBSERVATION:**

Both Bubble Sort and Insertion Sort correctly sorted the nearly sorted array.

Insertion Sort performed fewer comparisons and shifts, completing faster than Bubble Sort.

Bubble Sort required multiple passes even though the array was almost sorted, while Insertion Sort efficiently inserted elements in their correct positions

**TASK 3:**

Implement both Linear Search and Binary Search.

**PROMPT:**

Write a Python program for **Linear Search** and **Binary Search**.  
Add **comments** to explain the code.  
Test both on **sorted and unsorted lists**.  
Explain **when Binary Search is better**.

**CODE:**

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**OUTPUT:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**OBSERVATION:**

**Linear Search** is simple and works on **both sorted and unsorted lists**, but it checks each element one by one. This makes it **slow for large datasets**, especially when the target is near the end or not present.

**Binary Search** is much **faster** because it divides the search space in half with each step. However, it **only works on sorted lists**. If the list isn’t sorted, Binary Search gives incorrect results.

**TASK 4:**

Implement Quick Sort and Merge Sort using recursion.

**PROMPT:**

Implement Quick Sort and Merge Sort using recursion in Python.

1. Complete the recursive logic for both sorting algorithms.

2. Add proper docstrings explaining the function, parameters, and return values.

3. Compare the performance of both algorithms on three types of lists:

- Random list

- Already sorted list

- Reverse-sorted list

4. Print the execution time for each algorithm on each type of list.

5. Use lists of different sizes (e.g., 10, 100, 1000 elements).

Make the code fully runnable and beginner-friendly.

**CODE:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**OUTPUT:**

**A computer screen with blue and white text

AI-generated content may be incorrect.**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**OBSERVATION:**

**Quick Sort** is generally **faster on random data** due to its in-place partitioning and low overhead. However, its performance heavily depends on the choice of pivot:

* **Merge Sort** is **more stable and consistent**:
* It always guarantees **O(n log n)** time complexity, regardless of input order.
* It performs equally well on **random**, **sorted**, and **reverse-sorted** lists.
* However, it uses **extra space** for merging, which can be a drawback for memory-constrained environments.