**SOFTWARE ENGINEERING DEPARTMENT**

**Total Marks: 100**

**Obtained Marks:**

**Project Assignment**

**Last date of Submission: 28 May 2025**

**Submitted To: Shakeel Ahmad**

**Submitted By: M.Sohaib Kamran (4784-FOC/BSSE/F23)**

**Soban Abbass (4805-FOC/BSSE/F23)**

**M.Huzaifa**  **(4801-FOC/BSSE/F23)**

**M.Zayd Dawood (4818-FOC/BSSE/F23)**

**Data Structures & Algorithms – Group Project Assignment**

Semester Project | Group Size: 4 Students | Programming Language: C++

**Performance Evaluation of Data Structures and Algorithms :-**

**In the provided C++ program, we utilize arrays to store member information, payments, and membership types. The program implements linear search for finding members and bubble sort for sorting members. Below is a comparative analysis of the chosen data structures and algorithms in terms of time and space complexity.**

**Data Structures Used:-**

**1. Arrays:**

- **Space Complexity:** O(n), where n is the number of members (up to “MAX\_MEMBERS”).

- **Time Complexity**: Accessing an element is O(1), but insertion and deletion are O(n) since shifting elements may be required.

**Searching Algorithms**

**1. Linear Search:**

**- Time Complexity:** O(n)

**- Space Complexity:** O(1)

**- Description:** The algorithm iterates through the array to find a member by CNIC or name. It is simple and works well for small datasets but becomes inefficient as the dataset grows.

**Sorting Algorithms**

**1. Bubble Sort:**

**- Time Complexity:** O(n^2)

**- Space Complexity:** O(1)

**- Description:** Bubble sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. It is easy to implement but inefficient for large datasets.

**Comparative Analysis**

**| Algorithm | Time Complexity (Best) | Time Complexity (Average) | Time Complexity (Worst) | Space Complexity | Use Case |**

**|--------------------|------------------------|---------------------------|-------------------------|------------------|-----------------------------------|**

**| Linear Search | O(1) | O(n) | O(n) | O(1) | Small datasets, unsorted arrays |**

**| Bubble Sort | O(n) | O(n^2) | O(n^2) | O(1) | Small datasets, educational purposes |**

**Efficiency and Trade-offs**

**- Linear Search:**

**- Pros:** Simple to implement, no additional space required.

**- Cons:** Inefficient for large datasets; performance degrades linearly with the number of elements.

- **Bubble Sort:**

**- Pros:** Simple to implement and understand; no additional space required.

**- Cons:** Highly inefficient for large datasets; performance degrades quadratically with the number of elements.

**Recommendations**

**1. Data Structure:**

- For larger datasets, consider using a dynamic data structure like `std::vector` instead of fixed-size arrays. This allows for dynamic resizing and better memory management.

**2. Searching Algorithm:**

- For improved search performance, consider using a hash table or binary search (after sorting) instead of linear search. This would reduce the average time complexity to O(1) for hash tables and O(log n) for binary search.

**3. Sorting Algorithm:**

- Replace bubble sort with a more efficient sorting algorithm like quicksort or mergesort, which have average and worst-case time complexities of O(n log n). This would significantly improve performance for larger datasets.

- **Linear Search:**

- A linear graph showing a direct proportionality between the number of elements and time taken.

- **Bubble Sort:**

- A quadratic graph showing a steep increase in time taken as the number of elements increases.

**Space Complexity Graphs**

- Both algorithms have constant space complexity, represented as a flat line across the graph.

**Conclusion**

The current implementation is suitable for small datasets due to its simplicity. However, for scalability and efficiency, it is recommended to adopt more advanced data structures and algorithms. The choice of algorithms and data structures should align with the expected size of the dataset and the required performance characteristics. By implementing these recommendations, the application can handle larger datasets more efficiently and effectively.

**-:End of Analysis:-**