Project Title: Implementation and Comparative Analysis of Basic Sorting Algorithms (Bubble, Insertion, and Selection Sort)

Submitted by:

AARUSH C S
ATHUL K KOSHY
MOHAMMED RIZWAN PP
MOIDEEN NIHAL

---

1. Introduction

Sorting algorithms are fundamental to computer science and are widely used in areas such as data processing, searching, and organizing datasets. This project focuses on the implementation and comparison of three elementary sorting techniques: Bubble Sort, Insertion Sort, and Selection Sort. While these algorithms are simple and easy to understand, they form the foundation for more advanced sorting methods.

---

2. Problem Statement

Sorting is essential for efficient data management and retrieval. Many real-world applications require sorted data to function correctly and optimally. Although these basic sorting algorithms are not suitable for large-scale datasets due to their time complexity, they are still valuable in educational contexts and in applications with small to moderate data sizes.

---

3. Objectives

- **Understand the working principles of Bubble Sort, Insertion Sort, and Selection Sort.**
- **Implement all three algorithms using Python.**
- **Compare their performance on different dataset sizes.**
- **Analyze time and space complexity.**
- **Identify scenarios where each algorithm performs best.**

---

4. Literature Review

- **Bubble Sort: Compares adjacent elements and swaps them if they are in the wrong order. Simple but inefficient for large datasets.**
- **Insertion Sort: Builds the final sorted array one item at a time by comparing and inserting the current element at its correct position.**

- **Selection Sort: Repeatedly finds the minimum element and places it at the beginning of the array.**
  These algorithms have time complexity of O(n^2) in worst-case scenarios but are easy to implement and understand.

---

5. Methodology

1. **Implement the three sorting algorithms in Python.**
2. **Use random lists of integers of varying sizes (e.g., 10, 100, 1000) for testing.**
3. **Measure the execution time of each algorithm using the time module.**
4. **Visualize or print step-by-step sorting progress (optional).**
5. **Compare and analyze the performance using execution time and number of operations.**

---

6. Tools and Technologies

- **Programming Language: Python**
- **Libraries: random, time, matplotlib (optional for visual comparison)**

---

7. Expected Results

- **All algorithms will correctly sort the data.**
- **Insertion Sort will perform better on nearly sorted data.**
- **Selection Sort will perform consistently but slower on larger datasets.**
- **Bubble Sort will generally be the slowest due to repeated comparisons.**
- **The comparison will highlight the limitations and ideal use-cases of each algorithm.**

---

8. Challenges and Solutions

Challenge	**Solution**
High time complexity	**Limit testing to small/medium datasets for analysis**
Repeated logic in similar algorithms	**Use modular functions and reuse code where possible**

Challenge	**Solution**
Visualizing step-by-step sorting	**Use simple print statements or matplotlib for graphs**

---

9. Conclusion

This project emphasizes the importance of mastering basic sorting techniques, which are essential for understanding more complex algorithms. By comparing Bubble Sort, Insertion Sort, and Selection Sort, the project highlights their performance, limitations, and educational value. The insights gained can guide the choice of sorting methods in real-world applications.

---

10. References

1. **"Data Structures and Algorithms in Python" – Goodrich et al.**
2. **"Introduction to Algorithms" – Cormen et al.**
3. **GeeksforGeeks – Sorting Algorithm Tutorials**
4. **Python.org – Official Python Documentation**
5. **W3Schools – Python Sorting**

CODE:-

Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

Insertion Sort

```
def insertion_sort(arr):

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        while j >= 0 and arr[j] > key:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key
```

Selection Sort

```
def selection_sort(arr):

    n = len(arr)

    for i in range(n):

        min_idx = i

        for j in range(i + 1, n):

            if arr[j] < arr[min_idx]:

                min_idx = j

        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```