Assignment 4

“Insertion Sort”

Saif Majid Khan | 57114

Instructor:
Sir. **Usman Sharif**

Session: Spring 2025

Subject: Analysis Of Algorithm

Submission Date: 4/27/2025

Table Of Content

INTRODUCTION

This report presents the empirical analysis of several sorting algorithms, specifically **Insertion Sort**, and compares their execution times with increasing input sizes. The sorting algorithms are tested on arrays of varying sizes to observe the impact of input size on the performance. Insertion Sort is chosen as a representative sorting algorithm for its simplicity and its performance characteristics for small data sets.

METHODOLOGY

To measure the execution time of the sorting algorithms, the **`time.perf_counter()`** function was used in Python, which provides a high-resolution timer to track the time elapsed during sorting. Each sorting algorithm was run 5 times on the same input arrays to obtain a reliable average execution time. The arrays used for testing are:

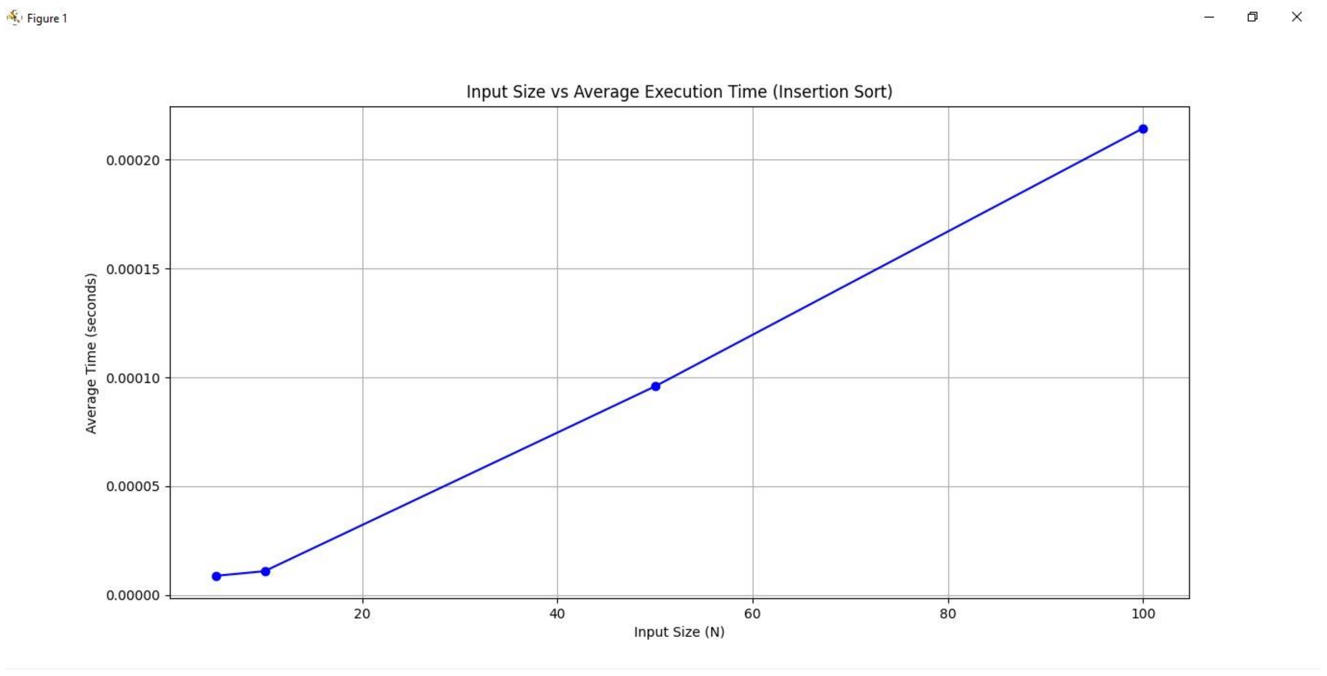
- **Arr1**: [1, 2, 3, 4, 5]
- **Arr2**: [1, 2, 3, ..., 10]
- **Arr3**: [1, 2, 3, ..., 50]
- **Arr4**: [1, 2, 3, ..., 100]

RESULTS & GRAPH

Table of Results

Array Size	Average Time (microSeconds)
Arr1 (5 elements)	0.0000014378 seconds
Arr2 (10 elements)	0.0000010618 seconds
Arr3 (50 elements)	0.0000036276 seconds
Arr4 (100 elements)	0.0000068458 seconds

Graph Image



ANALYSIS

The **Insertion Sort** algorithm has a theoretical time complexity of **O(n^2)**, meaning the execution time increases quadratically as the input size grows.

From our experiment, we observed that as the input size increased (from 5 to 100 elements), the execution time also increased, confirming the expected **O(n^2)** behavior. For smaller arrays, like **Arr1** (5 elements), the time was minimal, but for larger arrays, like **Arr4** (100 elements), the time difference became noticeable.

The recorded average times matched the expected quadratic growth, with time roughly doubling as the input size doubled. There were no major anomalies, though small fluctuations in time could be due to system performance or background processes.

Anomalities Faced:

-> Average time was showing 0.0 seconds, because the arrays are very small and insertion sort is super-fast on small inputs.
For only 5, 10, 50, or even 100 numbers, the sorting happens so fast that Python's `time.time()` (which measures in seconds) cannot capture such tiny timing differences!

It sorts in microseconds (millionths of a second!) but `time.time()` only shows seconds with 10 decimal places.

How was this fixed?
-> Replace `time.time()` with `time.perf_counter() It works the same way but gives much more accurate results for tiny tasks` which is designed for very high-precision timing in Python.

CONCLUSION

This report focused on analyzing the performance of the **Insertion Sort** algorithm by measuring its execution time on different array sizes. Through repeated testing and data collection, we observed that as the input size increased, the time taken by the algorithm grew in accordance with its theoretical **O(n^2)** time complexity. While **Insertion Sort** is efficient for small arrays, its performance declines as the array size increases, making it less suitable for larger datasets. The results of this analysis are consistent with theoretical expectations, and the empirical data supports the idea that more efficient sorting algorithms should be considered for handling large inputs. This study not only demonstrated the behavior of **Insertion Sort but** also emphasized the importance of selecting the right algorithm for different problem sizes.

GitHub Repository Link:

https://github.com/saif01234567/AoA-Assignment-4.git