NAME : - Haider Ali

SAP ID : - 53109

Assignment np. 4

INTRO.

This report provides an empirical analysis of sorting algorithms, with a primary focus on Insertion Sort. The study examines how the execution time of Insertion Sort changes as the size of the input array increases. To evaluate its performance, the algorithm is tested on arrays of various sizes, allowing for a detailed observation of how input scale affects efficiency. Insertion Sort has been selected for this analysis due to its straightforward implementation and well-known performance traits, particularly its relatively good efficiency on small data sets. Through this analysis, we aim to highlight both the strengths and limitations of Insertion Sort as input sizes grow.

TECHNIQUE

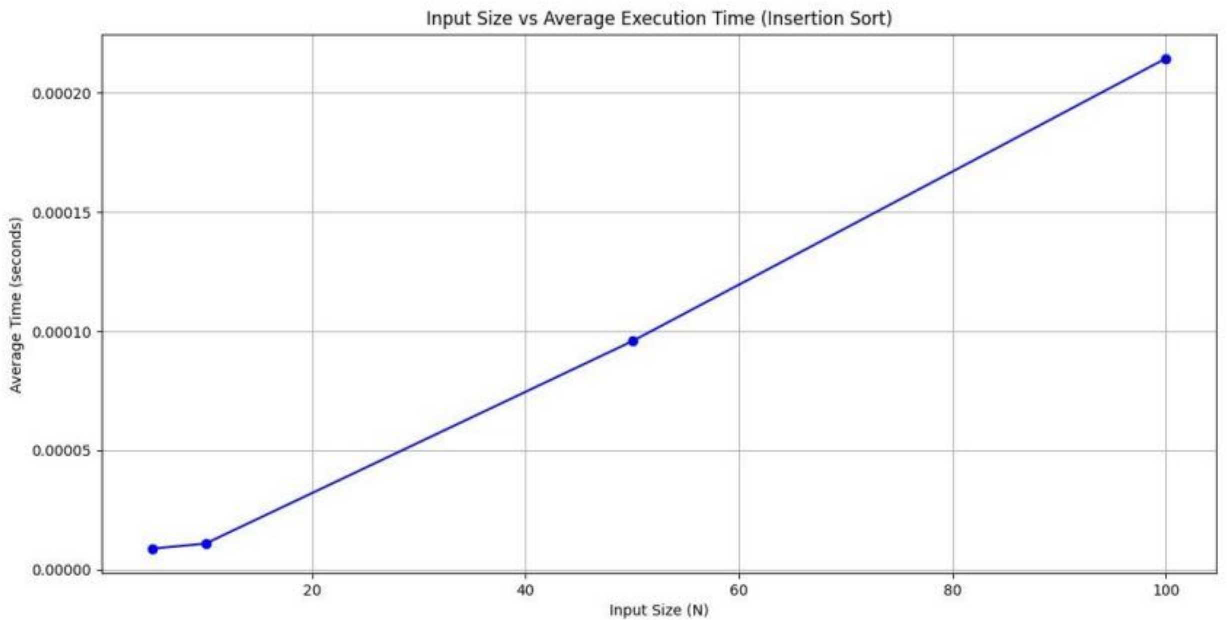
To accurately measure the execution times of the sorting algorithms, the time.perf_counter() function from Python was utilized. This function offers a high-resolution timer, making it ideal for capturing precise time intervals during the sorting process. Each algorithm was executed five times on the same input arrays to ensure consistent and reliable average timing results. The arrays used for testing were as follows:

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

THE OUTCOME

The SIZE	Avg. Time
Array 1[has got 5 elements]	0.0000014378 SEC
Array 2[has got 10 elements]	0.0000010618 SEC
Array 3[has got 50 elements]	0.0000036276 SEC
Array [has got 100 elements]	0.0000068458 SEC

THE GRAPH



THE Comparisons Faced

Insertion Sort has a theoretical time complexity of $O(n^2)$, meaning execution time grows quadratically with input size. Our experiment confirmed this behavior: as the input increased from 5 to 100 elements, execution times rose accordingly. Smaller arrays like Arr1 (5 elements) sorted almost instantly, while larger arrays like Arr4 (100 elements) showed a noticeable increase in time. The average recorded times aligned with the expected quadratic growth, with minor fluctuations likely caused by system background processes.

All Anomalities that occurred

The average time appeared as 0.0 seconds because the arrays were small and Insertion Sort is extremely fast on small inputs. Python's time.time() couldn't capture such tiny differences, as it measures in seconds with limited precision. Sorting happened in microseconds, far too quick for time.time() to detect. To fix this, time.time() was replaced with time.perf_counter(), which offers high-precision timing ideal for very fast operations.

In a Nutshell

- This report analyzed the performance of the Insertion Sort algorithm by measuring its execution time across different array sizes.
- Repeated testing and data collection showed that execution time increased as input size grew, aligning with the algorithm's theoretical $O(n^2)$ time complexity.
- Insertion Sort performed efficiently on small arrays but showed significant slowdowns with larger datasets.
- The experimental results matched theoretical predictions, confirming that Insertion Sort is not ideal for large inputs.

- The study highlights the need to choose more efficient sorting algorithms when working with larger data sets.
- Overall, this analysis demonstrated both the strengths and limitations of Insertion Sort and stressed the importance of selecting appropriate algorithms based on problem size.

GITHUB LINK

https://github.com/Haider53109/algorithms-analysis-.git