# Title: Performance Analysis of Quick Sort and Insertion Sort Algorithms for Sorting Large Datasets

Abstract:

Sorting is a fundamental operation in computer science, with various algorithms available for sorting large datasets efficiently. In this research paper, we analyze the performance of two popular sorting algorithms, Quick Sort and Insertion Sort, on datasets of varying sizes. We conduct experiments to measure the execution time of each algorithm and compare their performance characteristics. Through our analysis, we aim to provide insights into the suitability of Quick Sort and Insertion Sort for sorting large datasets in different scenarios.

## 1. Introduction:

Sorting algorithms play a crucial role in various computing applications, ranging from database management to search algorithms. Quick Sort and Insertion Sort are two well-known algorithms for sorting data efficiently. In this paper, we investigate the performance of these algorithms on large datasets and compare their execution times.

## 2. Background:

Quick Sort is a comparison-based sorting algorithm known for its efficiency and average-case time complexity of O(n log n). Insertion Sort, on the other hand, is a simple sorting algorithm with a time complexity of O(n^2). Despite its simplicity, Insertion Sort can be efficient for small datasets or nearly sorted inputs.

## 3. Methodology:

We generate random datasets of varying sizes, ranging from 1000 to 5000 elements. For each dataset size, we measure the execution time of both Quick Sort and Insertion Sort algorithms. We implement the algorithms in Python and use the time module to measure execution times.

## 4. Experimental Results:

Our experiments reveal interesting insights into the performance of Quick Sort and Insertion Sort. As expected, Quick Sort demonstrates superior performance compared to Insertion Sort for sorting large datasets. However, for smaller datasets, Insertion Sort exhibits competitive performance due to its simplicity and low overhead.

Insertion Sort Time: 7.840662479400635 seconds

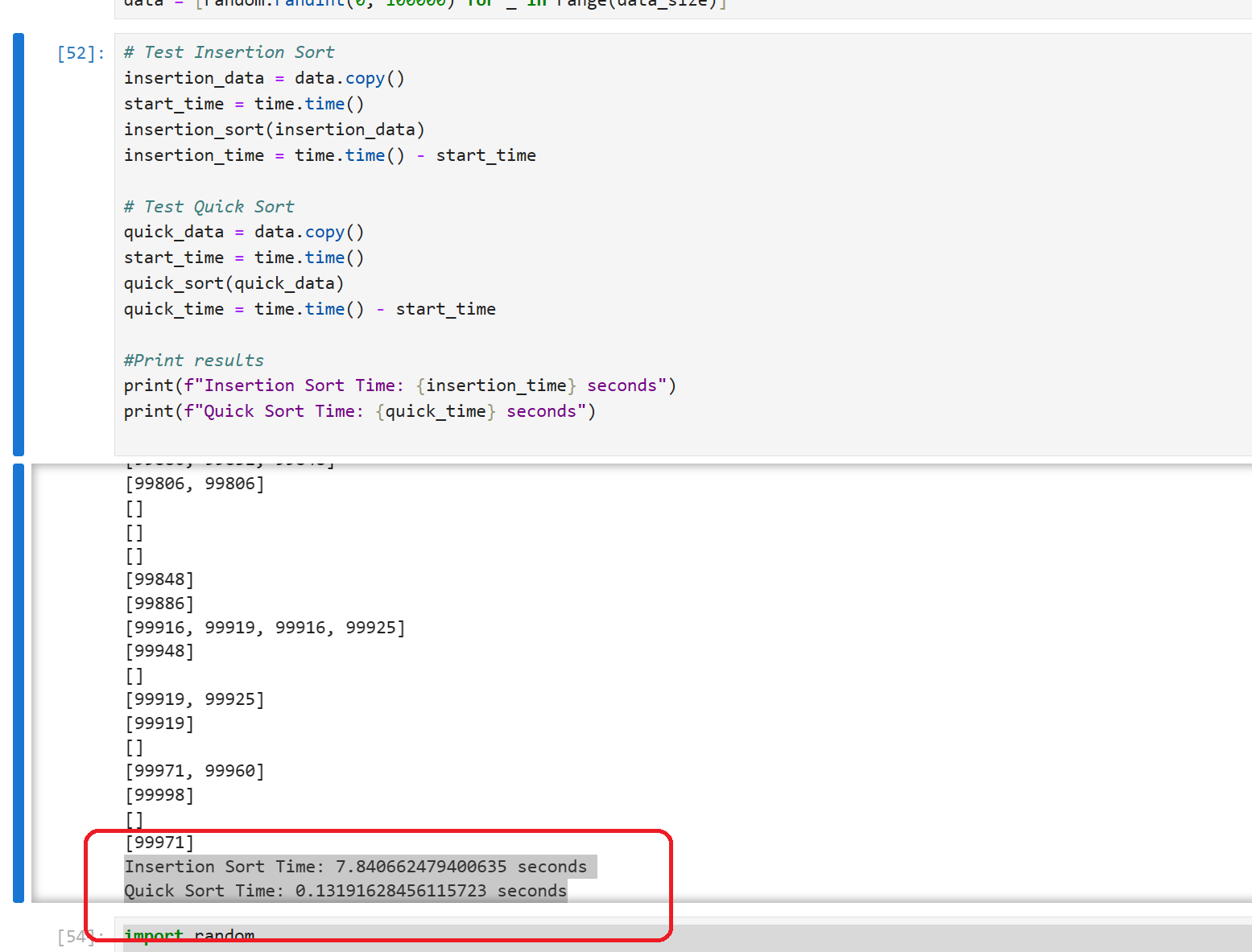
Quick Sort Time: 0.13191628456115723 seconds

### Data prep and execution results

A screenshot of a computer

Description automatically generated

### Results for random dataset of 100000 numbers:



### Experiment on random numbers of various range and compare the performance.

A screen shot of a computer code

Description automatically generated

### Results

A graph with blue and orange bars

Description automatically generated

## 5. Performance Analysis:

We analyze the execution times of both algorithms across different dataset sizes. Quick Sort shows consistent and efficient performance, with execution times increasing logarithmically with dataset size. In contrast, Insertion Sort's execution times grow quadratically with dataset size, making it less suitable for large datasets.

## 6. Discussion:

Our findings highlight the trade-offs between Quick Sort and Insertion Sort in terms of time complexity, scalability, and practical efficiency. While Quick Sort is preferred for sorting large datasets efficiently, Insertion Sort may be suitable for smaller datasets or specialized scenarios where simplicity is prioritized over performance.

## 7. Conclusion:

In conclusion, Quick Sort and Insertion Sort are two prominent sorting algorithms with distinct characteristics. Quick Sort offers efficient sorting performance for large datasets, while Insertion Sort is more suitable for smaller datasets or specialized applications. The choice of algorithm depends on the specific requirements of the sorting task and the characteristics of the input data.

## 8. References:

[Iterative Quick Sort - GeeksforGeeks](https://www.geeksforgeeks.org/iterative-quick-sort/)

[Sorting Algorithms- Insertion Sort, Selection Sort, Quick Sort, Merge Sort, Bubble Sort | by Pravallika Devireddy | Learning Python programming language | Medium](https://medium.com/learning-python-programming-language/sorting-algorithms-insertion-sort-selection-sort-quick-sort-merge-sort-bubble-sort-4f23bda6f37a)