Abstract

The primary objective of this project is to implement and compare the performance of various sorting algorithms on a substantial dataset. The implementation is carried out using Python, focusing on four key sorting methods: Selection Sort, Insertion Sort, Bubble Sort, and Merge Sort. The project aims to provide an educational resource for understanding the working principles, computational complexities, and real-time execution performance of these fundamental algorithms.

Introduction

Sorting algorithms are essential in computer science, playing a critical role in data processing and manipulation. This project provides a practical demonstration of four classic sorting techniques, each implemented within a Python class called sortingMethods. The class operates on a predefined array of integers, showcasing the steps each algorithm takes to sort the data.

Methodology

1.Selection Sort:

This algorithm divides the input list into two parts: the sorted part at the left end and the unsorted part at the right end. It repeatedly selects the smallest (or largest, depending on the sorting order) element from the unsorted part and moves it to the sorted part.

Time Complexity: O(n^2)
Space Complexity: O(1)

2.Insertion Sort:

This algorithm builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.

Time Complexity: O(n^2)
Space Complexity: O(1)

3.Bubble Sort:

This simple sorting algorithm repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

Time Complexity: O(n^2)
Space Complexity: O(1)

4.Merge Sort:

This efficient, stable, and comparison-based sorting algorithm divides the unsorted list into n sublists until each sublist contains a single element, then merges those sublists to produce new sorted sublists until there is only one sublist remaining.

Time Complexity: O(n log n)
Space Complexity: O(n)

Implementation

The sortingMethods class encapsulates the array and provides methods for each sorting algorithm. The array is initialized with a predefined set of integers to facilitate benchmarking and comparison. Each sorting method records the start and end time to compute the execution duration, providing insights into the performance differences among the algorithms.

Results

Upon execution, each sorting method outputs the sorted array and the time taken for the operation. The results highlight the efficiency of Merge Sort over larger datasets compared to Selection, Insertion, and Bubble Sort, which exhibit quadratic time complexity.

Conclusion

This project successfully demonstrates the implementation of Selection Sort, Insertion Sort, Bubble Sort, and Merge Sort in Python. It provides a clear comparison of their performance on a large dataset, reinforcing theoretical computational complexities with practical runtime data. This serves as an educational tool for understanding the strengths and limitations of different sorting algorithms in real-world applications.