Data Structures and Algorithms in Python

Assignment 15 – Experimental Analysis of Sorting Algorithms

In the field of computer science, experimental analysis plays a crucial role in understanding and evaluating the performance of algorithms. Experimental analysis allows developers to gather real-world data on how algorithms behave under various conditions. This method contrasts with theoretical analysis, which relies on mathematical models to predict performance without actual implementation. Experimental analysis is particularly valuable when theoretical models cannot fully capture complex real-world environments or when actual performance metrics are needed for specific hardware and software configurations.

Sorting algorithms, which organize data into a specific order, are fundamental to the study of computer science and essential for efficient data processing and retrieval. These algorithms serve as the backbone for a wide range of applications, from database management systems to search engines and beyond. The importance of sorting lies in its ability to facilitate quicker search operations, optimize the use of storage, and improve the overall efficiency of computer programs. Sorting is a classic problem in computer science, with a wide array of algorithms developed over the years, each optimized for different scenarios and performance criteria. In educational settings, sorting algorithms serve as a canonical example to illustrate fundamental concepts in algorithm design, analysis, and optimization. The comparative study of these algorithms through experimental analysis not only reinforces understanding of these core principles but also highlights the importance of context and efficiency in algorithm selection.

For this assignment you will implement several different sorting algorithms and create plots of running time versus input size. We will implement an example algorithm in class – Bubble Sort. Your task will be to choose three additional algorithms to test, options include:

- Insertion Sort
- Selection Sort
- Merge Sort
- Quick Sort
- Python's built-in .sort() method for lists

You should upload your Python code and a PDF file with at least 4 graphs created using Pandas and Seaborn to CREEKnet. For full credit:

- Your graphs have properly labeled axes.
- You tested a wide range of input sizes.
- You averaged each running time measurement over 3 or more trials.