**Report: Empirical Analysis of the Sorting Algorithms**

**1. Language Used:**

* **Programming Language**: Python
* **Reason for Choosing Python**: Python was chosen because of its simplicity, extensive libraries for numerical operations, and its ability to efficiently handle array manipulations. Additionally, Python's visualization libraries like Matplotlib make it easier to graphically compare the performance of different sorting algorithms.

**2. Tools Used:**

* **Libraries and Frameworks**:
  + **NumPy**: For array operations and numerical computations.
  + **Matplotlib**: For plotting and visualizing the performance of the sorting algorithms.
* **Sorting Algorithms**:
  + **Counting Sort**: A non-comparative sorting algorithm that works efficiently for small ranges of integers.
  + **Radix Sort**: A digit-by-digit sorting algorithm that sorts numbers based on individual digits.
  + **Bucket Sort**: A distribution sort that divides an array into several smaller buckets, then sorts these individually.
  + **Bubble Sort**: A simple algorithm that repeatedly swaps adjacent elements if they are in the wrong order until the list is sorted
  + **Selection Sort**: An algorithm that repeatedly selects the smallest (or largest) element from the unsorted part and moves it to the sorted part.
  + **Insertion Sort**: An algorithm that builds a sorted array one element at a time by finding the correct position for each element in the sorted portion.
  + **Merge Sort:** A divide-and-conquer algorithm that splits the array into subarrays, sorts them, and merges them back together in sorted order.
  + **Heap Sort:** An algorithm that builds a binary heap from the array and repeatedly extracts the maximum element while maintaining the heap structure.
  + **Quick Sort:** An efficient algorithm that selects a pivot, partitions the array into elements less than and greater than the pivot, and recursively sorts the subarrays.
* **Visualization**: Matplotlib was used to visualize the execution times of the sorting algorithms for different array sizes.

**3. Instructions to Run the Code:**

1. **Dependencies**:
   * Install the necessary libraries:

bash

Copy code

pip install numpy matplotlib

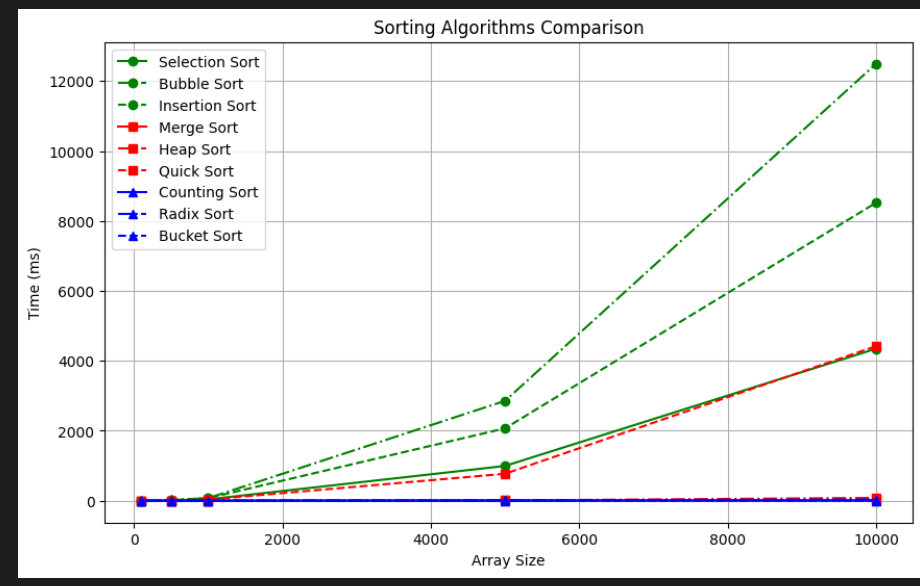
1. **Running the Code**:
   * Clone the repository or download the code files.
   * Open the Python file containing the sorting algorithms and performance evaluation.
   * Run the file using a Python interpreter:

bash

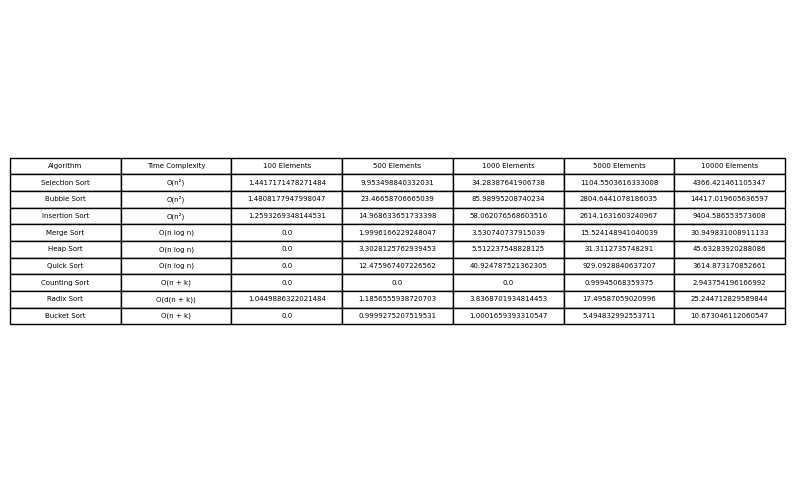
Copy code

python sorting\_algorithms.py

1. **Customization**:
   * The array size can be adjusted within the script by modifying the sizes variable.
   * The script will automatically run the sorting algorithms on arrays of different sizes and plot the comparison.
2. **Screenshots:**

****

**Table:**



**5. References:**

* **NumPy Documentation**: https://numpy.org/doc/
* **Matplotlib Documentation**: https://matplotlib.org/stable/contents.html
* Additional references used during implementation can be listed here.