Analysis of Sorting Algorithms

Rohit Chakraborty

September 3, 2024

- **What kind of machine did you use?**
  - The computer used for the experiments is equipped with an Intel Core i5-1235U processor, 8 GB of RAM, and a 512 GB SSD.
- **What timing mechanism?**
  - To measure the execution time of the sorting algorithms, the `chrono` library in C++ was employed. This library provides high-resolution clocks to capture time with great accuracy. The following Code shows how the library is being used:

```
auto start = std::chrono::high_resolution_clock::now();
//Code to execute the sorting algorithm
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
```

    In this Code:
    - `start` captures the time immediately before the sorting operation begins.
    - `end` captures the time immediately after the sorting operation ends.
    - `duration` computes the difference between `end` and `start`, providing the elapsed time in seconds.
- **How many times did you repeat each experiment?**

  - The experiment was repeated 10 times for each sorting algorithm to ensure stable results.

- **What times are reported?**
  - The average of the results were taken to plot the graph.

- **How did you select the inputs?**
  - Initially the algorithms were run on three types of input data - Increasing data, Decreasing data, and Random data. Then we observed the changes in the algorithms' behavior based on the input data type.

- **Did you use the same inputs for all sorting algorithms?**
  - The input files were generated once by a separate Data Generation program, and this was taken by the sorting algorithms, and it gave the output.

1 Which of the Three Versions of Quick Sort Seems to Perform the Best?

The performance of the three Quick Sort variations—First Pivot, Last Pivot, and Randomized Pivot—differs depending on the nature of the input data. Below the graphs have been plotted for three types of Quick Sort Algorithms in different scenarios.

- **Graph the best case running time as a function of input size n for all three versions.**

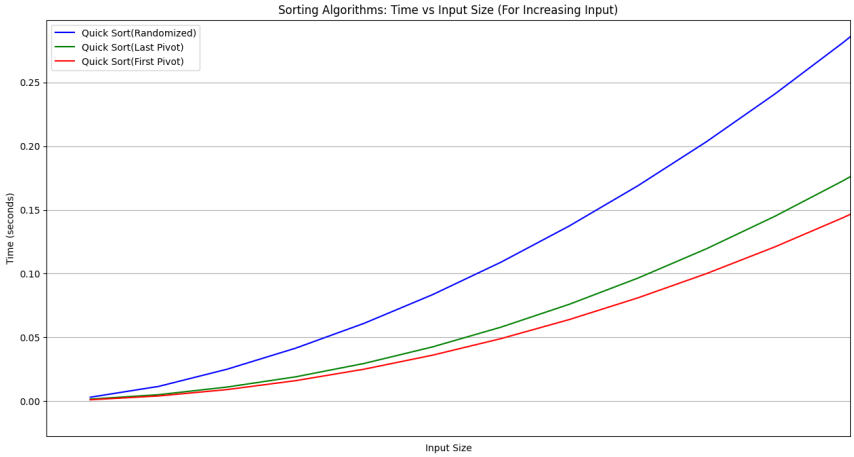

Figure 1: Best-case running time of three types of Quick Sort as a function of input size n.

- **Graph the worst case running time as a function of input size n for all three versions.**

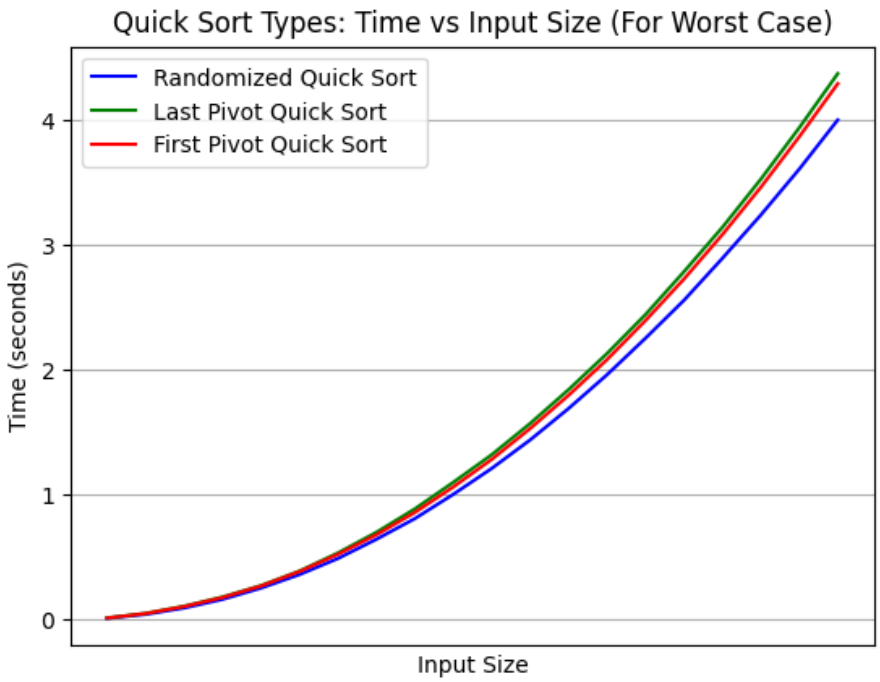

Figure 2: Worst-case running time of three types of Quick Sort as a function of input size n.

- **Graph the average case running time as a function of input size n for all three versions.**

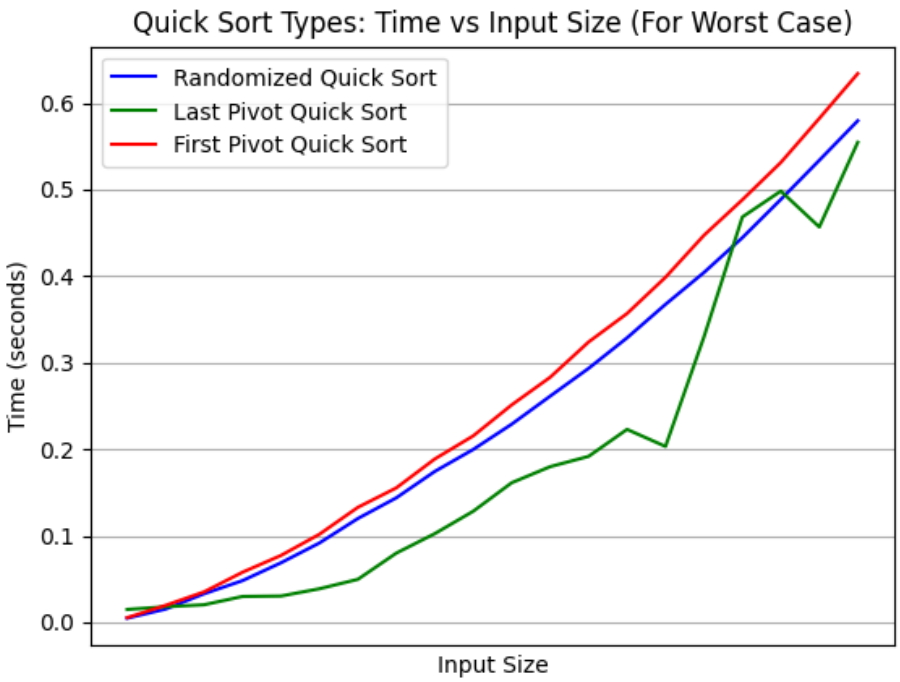

Figure 3: Worst-case running time of three types of Quick Sort as a function of input size n.

2 Which of the Six Sorts Seems to Perform the Best? (Consider the Best Version of Quick Sort)

We can consider the best sorting algorithms for different scenarios:

- For General Purpose we often use Quick Sort due to its average case time complexity and practicality.
- For stability we should consider Merge Sort.
- And for specific cases where the length of the integers are known we ca use Radix sort.

- **Graph the best case running time as a function of input size n for the six sorts.**

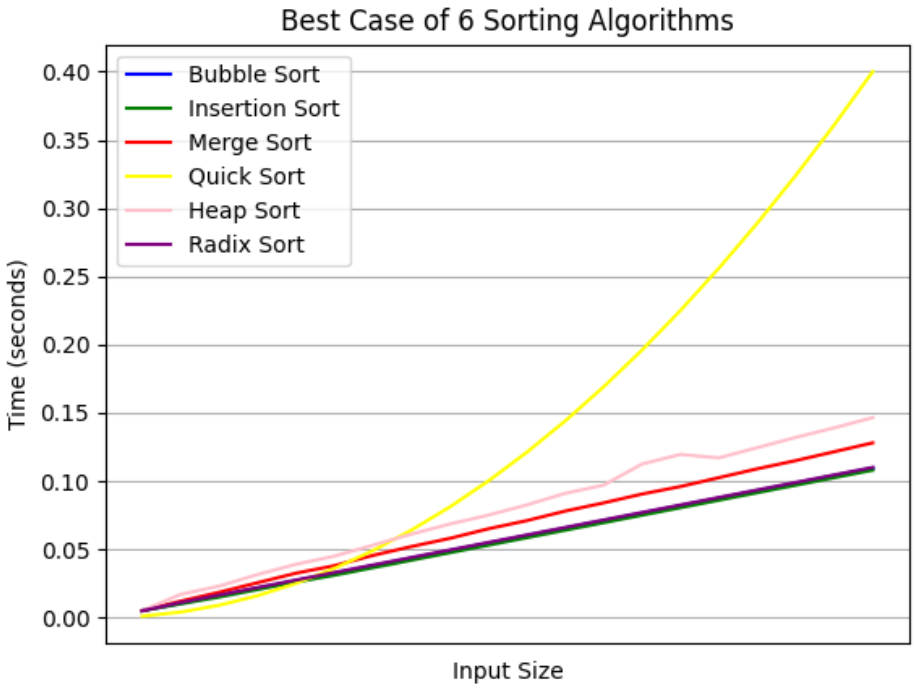

Figure 4: Best Case Graph of six sorting algorithms

- **Graph the worst case running time as a function of input size n for the six sorts.**

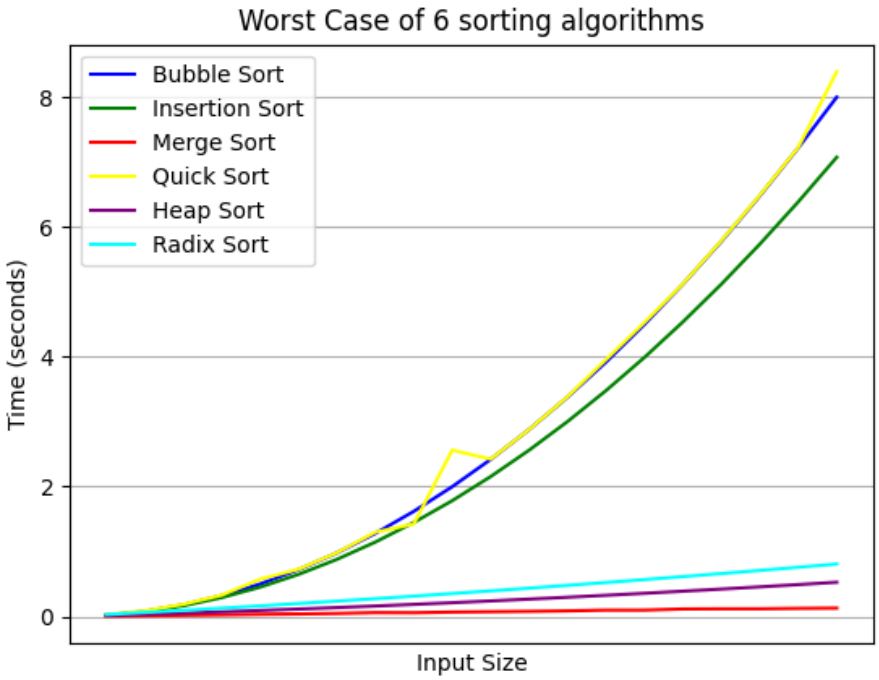

Figure 5: Worst Case Graph of six sorting algorithms

- **Graph the average case running time as a function of input size n for the six sorts.**

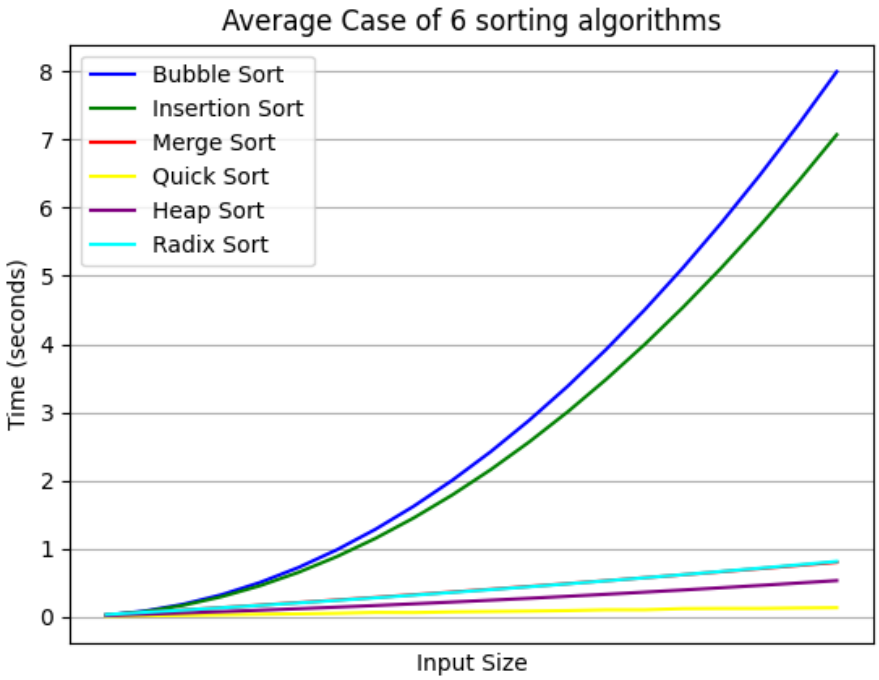

Figure 6: Average Case Graph of six sorting algorithms