Experimental Evaluation of Sorting Algorithms:

Dmitri Azih

1. Abstract:

This report investigates the performance of the implementation of Insertion Sort, Merge Sort, and Quicksort algorithms in Java, focusing on their efficiency in sorting large arrays. The report explores the algorithm's effectiveness in processing data and presents experimental results comparing their average time costs for sorting arrays of size 1,000,000. Key findings highlight the superiority of Merge Sort and Quicksort over Insertion Sort, emphasizing the importance of algorithm selection for optimizing sorting performance in data processing applications.

2. Platform Description

For this experiment, we utilized a desktop computer with the following configuration:

Processor: Intel® Core™ i9-9900K CPU @ 3.60GHz × 16
RAM: 32.0 GiB
Operating System: Ubuntu 22.04.2 LTS
Programming Language: Java (OpenJDK 21.0.1)

3. Test of Implementation

To verify the correctness of our implementation, we conducted several test cases for each sorting algorithm:

Arrays to Test Insertion, Merge and Quicksort:

Input: [1, 2, 3, 4, 5]
Expected Output: [1, 2, 3, 4, 5]

Input: [120, 98, 55, 32, 31]
Expected Output: [31, 32, 55, 98, 120]

Input: [38, 27, 43, 3, 9, 82, 10]
Expected Output: [3, 9, 10, 27, 38, 43, 82]

These test cases covered various scenarios including small and large input sizes, sorted and unsorted arrays, as well as arrays with duplicate elements.

4. Technical Discussion on Experimental Results

The experiment aims to calculate the time costs of each sorting algorithm implemented. A standalone function was written to calculate these costs. This function, “calculateTimeCosts”, begins by creating a 2D integer array holding space for 20 different arrays. Each array contains “maxInputSize” elements, and with each element ranging from -maxInputSize to maxInputSize. The function then records the time taken to sort each array and calculates each mean sorting time (time costs).

For this experiment, maxInputSize = 1,000,000. Upon conducting the experiments, we gathered the following time costs in milliseconds:

Insertion Sort:
Average time cost for an array of maxInputSize: *75041.90 ms* (*1.2 mins*)

Merge Sort:
Average time cost for an array of maxInputSize: *3795.11 ms* (*3.7 secs*)

Quicksort:
Average time cost for an array of maxInputSize: *238.71 ms* (*less than a second*)

From the results, I observed that both merge sort and quicksort significantly outperformed insertion sort in terms of execution time. This outcome aligns with my theoretical expectations, as merge sort and quicksort both have average time complexities of O(n log n), whereas insertion sort has an average time complexity of O(n^2).

Additionally, I noticed that quicksort showed slightly better performance compared to merge sort in my experiments. This could be attributed to the overhead involved in the merge step of merge sort, which requires additional memory allocation and copying.

It’s noted, however, that the performance of these sorting algorithms can vary depending on factors such as input size, the distribution of the input, and hardware characteristics.