Comparison of Sorting Algorithms

Diego Rodrigues

September 21, 2024

1 Introduction

Sorting algorithms are fundamental in computer science, playing a critical role in optimizing data operations. This project compares the performance of four sorting algorithms: Insertion Sort, Quick Sort, Heap Sort, and Merge Sort. The algorithms are evaluated based on the number of comparisons, movements, and total execution time.

2 Project Design

The project is structured to empirically assess the efficiency of different sorting algorithms under varying input conditions.

2.1 Implemented Algorithms

- **Insertion Sort**: Builds the final sorted array one item at a time.
- **Quick Sort**: Employs a divide-and-conquer strategy by selecting a pivot and partitioning the array.
- **Heap Sort**: Utilizes a binary heap data structure to sort elements.
- **Merge Sort**: Divides the array into halves, recursively sorts them, and merges the sorted halves.

2.2 Measured Metrics

- **Comparisons**: Number of element comparisons.
- **Movements**: Number of element movements or swaps.
- **Total Time**: Execution time in nanoseconds.

2.3 Data Generation

Two types of arrays are used:

- **Random Order**: An array of 50,000 randomly generated integers.
- **Ascending Order**: The same array sorted in ascending order.

2.4 Experimental Procedure

Each algorithm sorts both arrays, recording performance metrics for analysis and ranking.

3 Results

3.1 Experimental Results

3.1.1 Random Order

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Insertion Sort	628,031,567	628,031,567	556,118,800
Quick Sort	996,787	547,941	9,069,200
Heap Sort	1,525,122	737,561	9,431,200
Merge Sort	718,283	718,283	9,957,000

3.1.2 Ascending Order

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Insertion Sort	49,999	49,999	269,200
Quick Sort	1,218,936,134	1,218,983,636	1,567,933,800
Heap Sort	1,597,540	773,770	2,326,400
Merge Sort	401,952	401,952	2,752,000

3.2 Ranking Tables

3.2.1 Random Order

Ranking by Comparisons

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Merge Sort	**718,283**	718,283	9,957,000
Quick Sort	**996,787**	547,941	9,069,200
Heap Sort	**1,525,122**	737,561	9,431,200
Insertion Sort	**628,031,567**	628,031,567	556,118,800

Ranking by Movements

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Quick Sort	996,787	**547,941**	9,069,200
Merge Sort	718,283	**718,283**	9,957,000
Heap Sort	1,525,122	**737,561**	9,431,200
Insertion Sort	628,031,567	**628,031,567**	556,118,800

Ranking by Total Time

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Quick Sort	996,787	547,941	**9,069,200**
Heap Sort	1,525,122	737,561	**9,431,200**
Merge Sort	718,283	718,283	**9,957,000**
Insertion Sort	628,031,567	628,031,567	**556,118,800**

3.2.2 Ascending Order

Ranking by Comparisons

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Insertion Sort	**49,999**	49,999	269,200
Merge Sort	**401,952**	401,952	2,752,000
Heap Sort	**1,597,540**	773,770	2,326,400
Quick Sort	**1,218,936,134**	1,218,983,636	1,567,933,800

Ranking by Movements

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Insertion Sort	49,999	**49,999**	269,200
Merge Sort	401,952	**401,952**	2,752,000
Heap Sort	1,597,540	**773,770**	2,326,400
Quick Sort	1,218,936,134	**1,218,983,636**	1,567,933,800

Ranking by Total Time

Sorting Algorithm	Comparisons	Movements	Total Time (ns)
Insertion Sort	49,999	49,999	**269,200**
Heap Sort	1,597,540	773,770	**2,326,400**
Merge Sort	401,952	401,952	**2,752,000**
Quick Sort	1,218,936,134	1,218,983,636	**1,567,933,800**

4 Conclusion

The results indicate that algorithm performance is highly dependent on the initial data order. For random data, Quick Sort and Merge Sort perform best in terms of time and efficiency. In contrast, Insertion Sort excels with already sorted data due to its linear time complexity in the best case. Quick Sort performs poorly on sorted data due to its pivot selection strategy, leading to unbalanced partitions.