# Sorting Analysis

By Renee Ruan

## Abstract

This paper will dive into the efficiency and runtime of four sorting algorithms—bubble sort, selection sort, insertion sort, and merge sort—on arrays of varying size.

## Methods and Procedures

In this experiment, two test sets of test arrays were used, each sorted with the four sorting algorithms.

The first set of test arrays contains 12 array-lengths varying from 2 to 4096, incremented by powers of two. For each length, four arrays are created for each of the four sorting algorithms.

The second set of test arrays contains arrays of length 4096. For the first round of tests in this second set, the four sorting algorithms are run on arrays ordered in reverse (large to small). For the final round of tests in this second set, the four sorting algorithms are run on arrays already in order from small to large.

The key statistics recorded for each algorithm and array type are the number of comparisons occurring, the number of moves made, and the number of nanoseconds elapsed for the algorithm to run. These are the indicators used for analysis in the results section.

## Results

The first set of tests were conducted on arrays of varying length, filled with random numbers from 0 to 9999 (inclusive).

Table: Test Set #1 (Number of Moves)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Moves | | | |
| Array Size | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| 2 | 3 | 6 | 1 | 1 |
| 4 | 9 | 12 | 7 | 5 |
| 8 | 36 | 24 | 31 | 17 |
| 16 | 126 | 48 | 139 | 48 |
| 32 | 810 | 96 | 545 | 122 |
| 64 | 2967 | 192 | 2115 | 309 |
| 128 | 12246 | 384 | 7643 | 740 |
| 256 | 49401 | 768 | 31885 | 1737 |
| 512 | 200880 | 1536 | 128217 | 3960 |
| 1024 | 783972 | 3072 | 525831 | 8953 |
| 2048 | 3127800 | 6144 | 2088963 | 19934 |
| 4096 | 12598080 | 12288 | 8482487 | 43938 |

Table: Test Set #1 (Number of Comparisons)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Comparisons | | | |
| Array Size | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| 2 | 2 | 3 | 1 | 1 |
| 4 | 12 | 10 | 7 | 5 |
| 8 | 35 | 36 | 31 | 17 |
| 16 | 165 | 136 | 139 | 48 |
| 32 | 744 | 528 | 545 | 122 |
| 64 | 2961 | 2080 | 2115 | 309 |
| 128 | 14732 | 8256 | 7643 | 740 |
| 256 | 62475 | 32896 | 31885 | 1737 |
| 512 | 254989 | 131328 | 128217 | 3960 |
| 1024 | 1042437 | 524800 | 525831 | 8953 |
| 2048 | 4155410 | 2098176 | 2088963 | 19934 |
| 4096 | 16298100 | 8390656 | 8482487 | 43938 |

Table: Test Set #1 (Number of Nanoseconds)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Nanoseconds | | | |
| Array Size | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| 2 | 1500 | 2800 | 800 | 41500 |
| 4 | 2800 | 2400 | 1500 | 8400 |
| 8 | 6400 | 5700 | 5300 | 17100 |
| 16 | 25400 | 17300 | 21700 | 36900 |
| 32 | 120400 | 59200 | 81700 | 81800 |
| 64 | 481200 | 220900 | 311800 | 181200 |
| 128 | 2440800 | 1031000 | 1328500 | 523200 |
| 256 | 9493100 | 4153100 | 5632200 | 626000 |
| 512 | 7737300 | 4740100 | 5479900 | 135100 |
| 1024 | 1594700 | 576900 | 24289400 | 369800 |
| 2048 | 8635900 | 3253900 | 4410000 | 1135100 |
| 4096 | 24418600 | 8116100 | 12950600 | 871900 |

The second set of tests were conducted on arrays of varying order (reverse and in-order).

Table: Test Set #2 (Number of Moves)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Moves | | | |
|  | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| Reverse Order | 25159680 | 12288 | 16777213 | 24576 |
| In Order | 0 | 12288 | 4095 | 24576 |

Table: Test Set #2 (Number of Comparisons)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Comparisons | | | |
|  | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| Reverse Order | 16773120 | 8390656 | 16777213 | 24576 |
| In Order | 4095 | 8390656 | 4095 | 24576 |

Table: Test Set #2 (Number of Nanoseconds)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Nanoseconds | | | |
|  | Bubble Sort | Selection Sort | Insertion Sort | Merge Sort |
| Reverse Order | 22515800 | 7806300 | 25335400 | 742200 |
| In Order | 3700 | 7902500 | 8300 | 645700 |

## Conclusion

Based on the results on the first test set of arrays, it is apparent that merge sort is the optimal sorting algorithm for larger datasets. For an array of length 4096, the merge sort algorithm took only 871,900 nanoseconds to run, while the other three algorithms all took over 8,000,000 nanoseconds. The merge sort required 43,938 moves, fewer moves than both the bubble and insertion sort algorithms.

While the selection sort algorithm required the least number of moves, it was the second most effective in terms of run-time, clocking in at 8,116,100 nanoseconds for an array of length 4096.

Graph: Test Set #1 (Number of Moves)

Graph: Test Set #1 (Number of Comparisons)

Graph: Test Set #1 (Number of Nanoseconds)

Based on the results of the second test set of arrays, it seems that for the most part, merge sort is once again most effective. For arrays ordered in reverse, the merge sort took only 742,200 nanoseconds, with selection sort coming in second at around 7 million nanoseconds.

However, for the arrays already in order, bubble sort was the most efficient—likely due to the fact that no moves were necessary for the bubble sort algorithm in the case that the array is already sorted.