Alejandro Santacoloma

COP 3530

March 23, 2018

Sorting Algorithm Analysis

Introduction

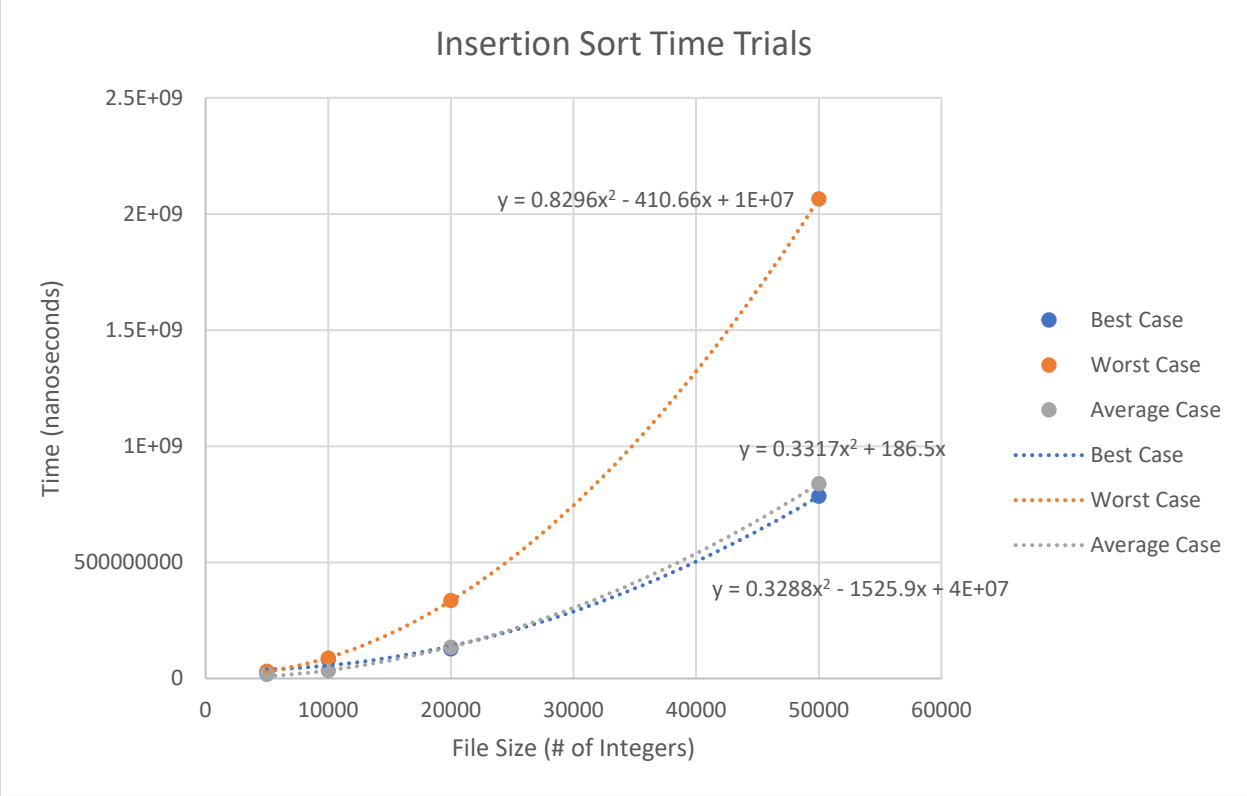
The objective of this report is to analyze and compare the performance of two sorting algorithms organizing an array of integers in ascending order. The two algorithms used in this analysis were insertion sort and merge sort. The algorithms sorted the same arrays containing 5000, 10000, 20000, and 50000 integers.

There were three variations of each array test case, the array in the first case was already set in ascending order before calling on the sorting algorithm to be the best case input for a sorting algorithm since the array would already be in order. The second case had the array set in descending order before calling on the sorting algorithm to be the worst case input for a sorting algorithm. The third case had the integers of the array in a random or shuffled order to represent the average case input for a sorting algorithm. The algorithms would be called on to sort the test cases and timed using the system clock. Each test case was run five times for each sorting method to obtain a more accurate run time value for each method and test case.

Insertion Sort

The insertion sort algorithm behaved as expected, run times between the three cases were similar when test cases had low file size. As the file sizes grew during testing, run times increased for all cases and differences in average run times also became more prominent between the ascending and descending test cases, with the ascending array test case having the shortest run time while the descending array having the worst run time and running 2.62 times slower than the ascending case.

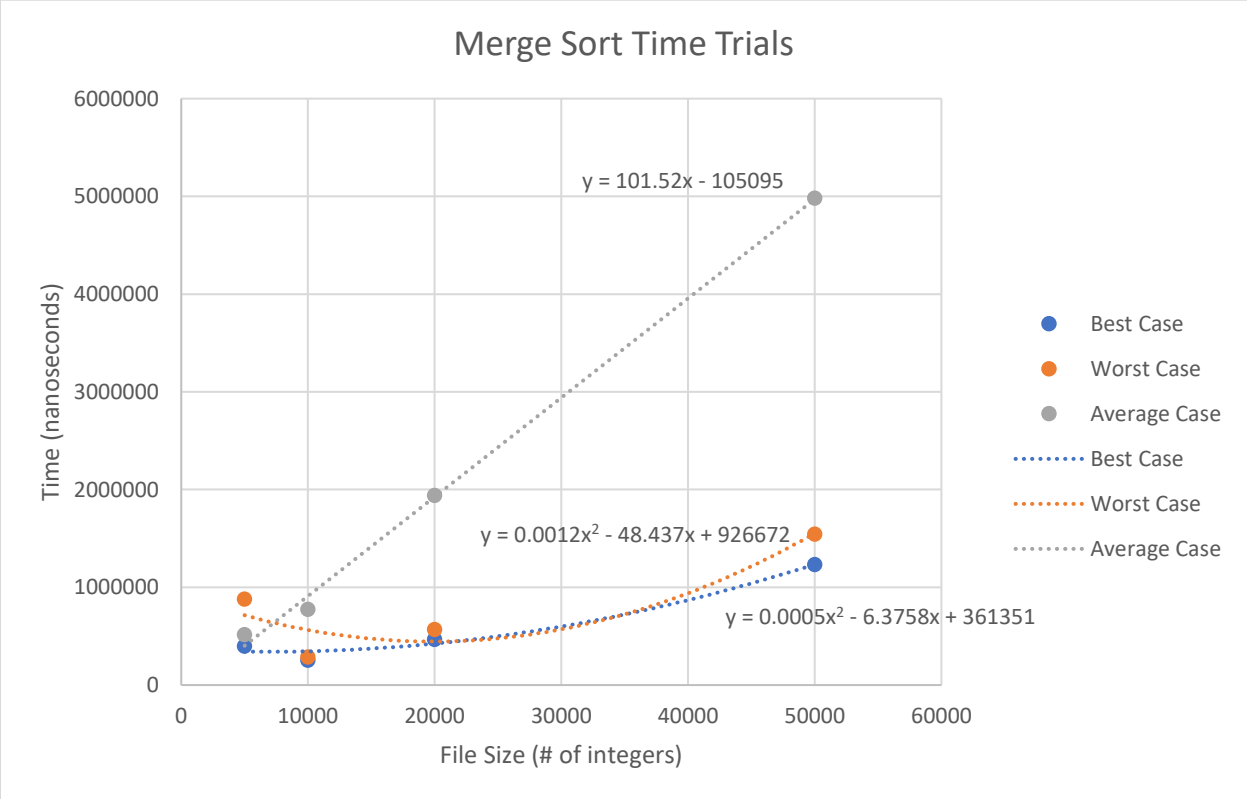
Quadratic trend lines fit well with the time trial data recorded for all 3 cases, agreeing with the theoretical time complexity of an insertion sort algorithm being $O(n^2)$ for the worst and average cases. The trend line of the best case scenario for insertion sort was expected to be linear as the theoretical time complexity is $O(n)$, however the line of best fit from the data was quadratic $O(n^2)$. This difference in theoretical and experimental time complexity may be due to deviations in time runs and the relatively small size of files tested, further testing and larger inputs will most likely reduce inaccuracies and cause the true time complexity of the algorithm to be more apparent and behave as expected.



Merge Sort

The merge sort algorithm behaved as expected in terms of having shorter run times compared to run times from insertion sort testing. However, the time runs from the test cases showed the worst case and best case closely matching each other in terms of trendline shape and time runs. Surprisingly the average case performed significantly worse than the worst case in terms of time, running 3.23 times slower than the worst case in the largest test cases when the array contained 50000 integers. Another unexpected pattern observed was the run times of the merge sort algorithm best test cases and worst test cases decreased when the input array size was increased from 5000 integers to 10000, later increases in file size did not decrease in run time.

Furthermore, the trendlines for the time complexity of the merge sort algorithm did not agree with the expected time complexities for the test cases. The trendline of the worst case and best case had behavior more closely matched to quadratic behavior $O(n^2)$ than the expected $O(n \log n)$ and $O(n)$ time complexity respectively. The trendline for the average case was linear $O(n)$, also not agreeing with the expected $O(n \log n)$ time complexity, however the trendline for the average case may begin to match more with the expected time complexity with further testing of larger file sizes. Similar to the differences in time complexity of the insertion sort algorithm, these differences in theoretical and experimental time complexity may be due to deviations in time runs and the relatively small size of files tested, further testing and larger inputs will most likely reduce inaccuracies and cause the true time complexity of the algorithm to be more apparent and behave as expected.



Algorithm Comparison

The merge sort algorithm relies on a divide and conquer strategy dividing the unsorted array into its smallest pieces using recursive calls of the algorithm, and then sorting and combining the pieces back into a single sorted array. The insertion sort algorithm on the other hand compares a single integer to the sorted portion of the array comparing values and placing in the correct spot between previously sorted values one by one. As was expected, the tested merge sort algorithm had significantly lower run times than the insertion sort algorithm tested. The run times of the insertion sort algorithm never outperformed the merge sort algorithm and the difference in their run times increased as the file size of the test cases grew. The greatest difference in runtime was in the worst case test scenario with a file size of 50000 integers, insertion sort average runtime was 1338.938 times slower in this test case compared to the average runtime using merge sort. The average runtimes for each test case for both algorithms as well as the ratio of runtimes for insertion sort over merge sort can be found below.

Insertion Sort Runtimes			
File Size (# of integers)	Best Case (Ascending Order) Runtime (nanoseconds)	Worst Case (Descending Order) Runtime (nanoseconds)	Average Case Runtime (nanoseconds)
5000	23717568	31201295	16678914
10000	81924018	87737611	34356580
20000	1.28E+08	3.36E+08	1.34E+08
50000	7.85E+08	2.06E+09	8.39E+08

Merge Sort Runtimes			
File Size (# of integers)	Best Case (Ascending Order) Runtime (nanoseconds)	Worst Case (Descending Order) Runtime (nanoseconds)	Average Case Runtime (nanoseconds)
5000	396058.8	879496.8	513974.4
10000	253038	286536.2	774699
20000	465595.4	568911.8	1938757
50000	1231408	1541923	4981629

Insertion Sort vs Merge Sort Runtime Ratio			
File Size (# of integers)	Best Case (Ascending Order) Runtime Ratio	Worst Case (Descending Order) Runtime Ratio	Average Case Runtime Ratio
5000	59.8839566	35.4763	32.45087
10000	323.76172	306.2008	44.3483
20000	274.833256	590.2541	69.14718
50000	637.581291	1338.938	168.3913