5734454

The most common sorting algorithms are bubble sort, insertion sort, selection sort, quick sort, and merge sort. In this experiment, these were compared to each other in running time. In the code each individual sorting method is timed and then a loop was created to time them up to 1000 times. The base number n was then incremented by 100 each time the loop ran.

In the graphs in the following experiment there is significant noise, with some research I found that this is due to background applications and processes running on the computer. This is what caused the large spikes in the last four graphs.

In the graphs, included in the excel file different comparisons were made using line graphs. The line graphs compared the number of elements in the array by the average time of execution. The first graph is of all of them compared through all random values up to the value of n is 100,000. The graph directly below that the comparison was the same graph but with less values of n. It’s in these two graphs, that it is seen that bubble sort is largely the worst method. This we know from studying Big-O notation. Which states that bubble sorts best case is O(n2). Therefore, the graph for that line looks like a steady upward slope. This means bubble sort will work faster the less numbers there are meaning that as the input increases the time to run bubble sort grows at the rate of its square. This makes this way of sorting impractical.

The next line that is seen in the graphs is for selection sort. Selection sort seems to behave much like bubble sort but a little better. It behaves almost as O(n log n) but we learned in class that it actually behaves as O(n2). This can be seen in the graph towards the end it begins to slightly curve.

For the rest of the graph, the rest of the sorting algorithms cannot be seen therefore more graphs had to be made. The following three graphs compare the last three algorithms. For the third and fourth graphs, quick sort was compared against insertion and merge sort. Quick sort in both these graphs seems faster than insertion sort and merger sort. This could be because quick sort at its worst case behaves as O(n2), but we learned in class that in practice behaves much better.

The last graph is comparing insertion sort and merge sort. Here is where we see that insertion sort has a better time complexity than merge sort. This is probably due to the fact that insertion sort has the best case of O(n) and an average case of O(n2). It could be that here we are witnessing its best case.

Here the listed values in nanoseconds and the increments of n can be seen for the first ten runs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | bubblesort | insertionsort | mergesort | quicksort | selectionsort |
| 100 | 134500 | 4500 | 59300 | 3900 | 77600 |
| 200 | 580800 | 5600 | 120900 | 2900 | 302900 |
| 300 | 1240700 | 41200 | 52000 | 1100 | 1729400 |
| 400 | 301100 | 10600 | 46400 | 900 | 248900 |
| 500 | 454000 | 14400 | 51100 | 800 | 349300 |
| 600 | 514500 | 16300 | 60500 | 1100 | 524600 |
| 700 | 428200 | 18000 | 71300 | 800 | 723200 |
| 800 | 539400 | 21600 | 79600 | 800 | 935300 |
| 900 | 658700 | 24300 | 89200 | 900 | 1137100 |
| 1000 | 785900 | 26600 | 99400 | 1000 | 1387000 |

Even in the numbers alone we can see which has a better time complexity. Ranked by least to most time complex fully for this experiment we see: quick sort < insertion sort < merge sort < selection sort < bubble sort.