Sorting Algorithms

Condevaux Louis

December 2019

1 Quick Thinking About Results

The differences between the five algorithms were greater than I expected. At first, no significant differences were found because of a small number of number tried. However, the higher the numbers, the more efficient the Divide and Conquer algorithms became. On the other hand, Bubble sort, Insertion sort, and Selection sort were impacted drastically by the increase of the array size.

2 Explanation of Algorithms

BubbleSort was by far the slowest algorithm of all, followed by insertionSort, selectionSort, mergeSort, and quickSort. A few explanations for these results could be:

BubbleSort: Quite inefficient by swapping each value based on an index but easy to implement. Always good to know for emergencies. Runs in O(n*n)

SelectionSort: Works the same way as BubbleSort, but involves less swapping. It will select a value based on the index and swap it until it reaches its place. Easy to implement and works well on small data sets. Runs in O(n*n)

InsertionSort: Used if the data is small and the array is already presorted. It could be considered as middle ground algorithm that can be really efficient if used the right way. Runs in O(n*n)

MergeSort: Divide and Conquer, breaks down the list into sub-lists until everything is sorted again in a big list. The main problem is the use of extra memory to store the newly created arrays. Runs in O(n)

QuickSort: Divide and Conquer, often considered as the fastest algorithm since it does not allocate any extra memory. It uses a pivot system to sort the data efficiently. Runs in O(log n)

3 Results

3.1 1000 floats

BubbleSort: 0.003791 seconds to execute.

insertionSort: 0.001238 seconds to execute.

selectionSort: 0.001559 seconds to execute.
quickSort: 0.000167 seconds to execute.
mergeSort: 0.000195 seconds to execute.

3.2 25000 floats

BubbleSort: 2.87519 seconds to execute.
insertionSort: 0.518498 seconds to execute.
selectionSort: 0.887946 seconds to execute.
quickSort: 0.004092 seconds to execute.
mergeSort: 0.006003 seconds to execute.

3.3 50000 floats

BubbleSort: 11.5507 seconds to execute.
insertionSort: 2.0394 seconds to execute.
selectionSort: 3.6617 seconds to execute.
quickSort: 0.00992 seconds to execute.
mergeSort: 0.013784 seconds to execute.

3.4 75000 floats

BubbleSort: 32.6727 seconds to execute.
insertionSort: 5.23063 seconds to execute.
selectionSort: 9.26074 seconds to execute.
quickSort: 0.014973 seconds to execute.
mergeSort: 0.022198 seconds to execute.

3.5 100000 floats

BubbleSort: 56.9524 seconds to execute.
insertionSort: 8.96408 seconds to execute.
selectionSort: 16.8387 seconds to execute.
quickSort: 0.024955 seconds to execute.
mergeSort: 0.040293 seconds to execute.