**CCDSALG Term 3, AY 2019 – 2020**

Project 1 – Comparing Sorting Algorithms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Section** | **Names** | **Task 1** | **Task 2** | **Task 3** | **Task 4** |
| S12 | Gan, John Matthew Ong | X | X | X | X |
| S16 | Noblefranca, Jose Noel Cleofe | X | X |  |  |
| S15 | Remudaro, Angelo Alvarez |  | X |  | X |

Fill this part with your section and names. For the tasks, put an X mark if you have performed the specified task. Please refer to the project specifications for the tasks.

**LIST OF SORTING ALGORITHMS**

|  |  |  |
| --- | --- | --- |
| **Sorting Algorithm** | **Author (if available)** | **Downloaded From** |
| Bubble sort | programmingsimplified.com | <https://www.programmingsimplified.com/c/source-code/c-program-bubble-sort> |
| Insertion sort | hackerearth.com | <https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/tutorial/> |
| Selection sort | geeksforgeeks.org | <https://www.geeksforgeeks.org/selection-sort/> |
| Merge sort | geeksforgeeks.org | <https://www.geeksforgeeks.org/merge-sort/> |
| Quick sort | geeksforgeeks.org | <https://www.geeksforgeeks.org/quick-sort/> |
| Radix sort | geeksforgeeks.org | <https://www.geeksforgeeks.org/radix-sort/> |

**COMPARISON TABLE**

**M = (10)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Size** | **Average Machine Execution Time (in milliseconds)** | | | | | |
| **Bubble** | **Insertion** | **Selection** | **Merge** | **Quick** | **Radix** |
| **1024** | 2.292 | 0.674 | 1.204 | 0.251 | 0.199 | 0.000 |
| **2048** | 8.282 | 2.096 | 4.993 | 0.399 | 0.298 | 0.399 |
| **4096** | 35.876 | 9.872 | 20.748 | 0.400 | 0.397 | 0.399 |
| **8192** | 245.050 | 58.808 | 136.192 | 2.294 | 1.993 | 1.395 |
| **16384** | 1110.002 | 213.791 | 453.668 | 4.083 | 3.695 | 2.587 |
| **32768** | 4354.261 | 772.954 | 1537.165 | 7.590 | 6.191 | 4.292 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Size** | **Average Counter Value (in millions)** | | | | | |
| **Bubble** | **Insertion** | **Selection** | **Merge** | **Quick** | **Radix** |
| **1024** | 0.524799 | 0.273881 | 0.524800 | 0.011276 | 0.012477 | 0.010285 |
| **2048** | 2.098175 | 1.043734 | 2.098176 | 0.024589 | 0.027369 | 0.020525 |
| **4096** | 8.390655 | 4.246568 | 8.390656 | 0.053262 | 0.058675 | 0.041005 |
| **8192** | 33.558528 | 16.522178 | 33.558528 | 0.114703 | 0.128216 | 0.081965 |
| **16384** | 134.225920 | 66.937032 | 134.225920 | 0.245776 | 0.290103 | 0.163885 |
| **32768** | 536.887296 | 266.814240 | 536.887296 | 0.524305 | 0.617586 | 0.327725 |

**GRAPHS**

Copy/paste the graphs here, make sure it is big enough to see the trend in the increase of the average Machine Execution Time (MET) and the average counter value.

**DISCUSSION**

Explain interesting findings based on your experiments.

* The fastest sorting algorithm based on growth rate and MET is Radix sort, while the slowest is Bubble sort.
* Despite that Bubble sort and Selection sort have a very similar growth rate, Bubble sort has a significantly longer MET than Selection sort, which makes Bubble sort slower than Selection sort in terms of time complexity.
* When plotted based on average frequency count, the graph type of each sorting algorithms corresponds to the growth rate in their respective Big O.
* In most cases, the average growth rate based on the average frequency count and the average MET of each sorting algorithm is directly proportional. Therefore, the slower the growth rate, the longer the MET.
* Sorting algorithms that are recursive in nature (Merge, Quick, Radix) have faster average MET and smaller average frequency count than those that are non-recursive (Bubble, Insertion, and Selection).
* Therefore, recursive sorting algorithms scale better with larger N values than non-recursive sorting algorithms.