PROBLEM SOLVING ALGORITHMS

NAME - RAHUL POTHIRENDI
NUID - 002889957
GITHUB - https://github.com/Pothirendirahul/INFO6205

TASK -

In this assignment, your task is to determine--for sorting algorithms--what is the best predictor of total execution time: comparisons, swaps/copies, hits (array accesses), memory used, or some combination of these.

You will run the benchmarks for merge sort, (dual-pivot) quick sort, and heap sort. You will sort randomly generated arrays of between 10,000 and 256,000 elements (doubling the size each time). If you use the *SortBenchmark*, as I expect, the number of runs is chosen for you. So, you can ignore the instructions about setting the number of runs.

For each experiment (a sort method of a given size), you will run it twice: once for the instrumentation, once (without instrumentation) for the timing.

Of course, you will be using the *Benchmark* and/or *Timer* classes, as you did in a previous assignment.

You must support your (clearly stated) conclusions with evidence from the benchmarks (you should provide log/log charts and spreadsheets typically).

All of the code to count comparisons, swaps/copies, and hits, is already implemented in the *InstrumentedHelper* class. You can see examples of the usage of this kind of analysis in:

- **src/main/java/edu/neu/coe/info6205/util/SorterBenchmark.java**
- **src/test/java/edu/neu/coe/info6205/sort/linearithmic/MergeSortTest.java**
- **src/test/java/edu/neu/coe/info6205/sort/linearithmic/QuickSortDualPivotTest.java**
- **src/test/java/edu/neu/coe/info6205/sort/elementary/HeapSortTest.java (you will have to refresh your repository for HeapSort).**

- QuickSort Dual Pivot generally performs better than MergeSort and HeapSort in terms of normalized time and raw time across different numbers of elements.

- It consistently has lower normalized time and raw time values compared to the other two algorithms.

- HeapSort tends to have the highest normalized time and raw time values among the three algorithms, indicating it is generally slower compared to QuickSort Dual Pivot and MergeSort.

- MergeSort generally performs better than HeapSort but slightly worse than QuickSort Dual Pivot in terms of time complexity.

- QuickSort Dual Pivot uses no extra memory for copying elements (copies = 0), whereas MergeSort and HeapSort involve copying elements, especially as the number of elements increases.

- This indicates QuickSort Dual Pivot has better memory efficiency in terms of avoiding additional memory allocations.

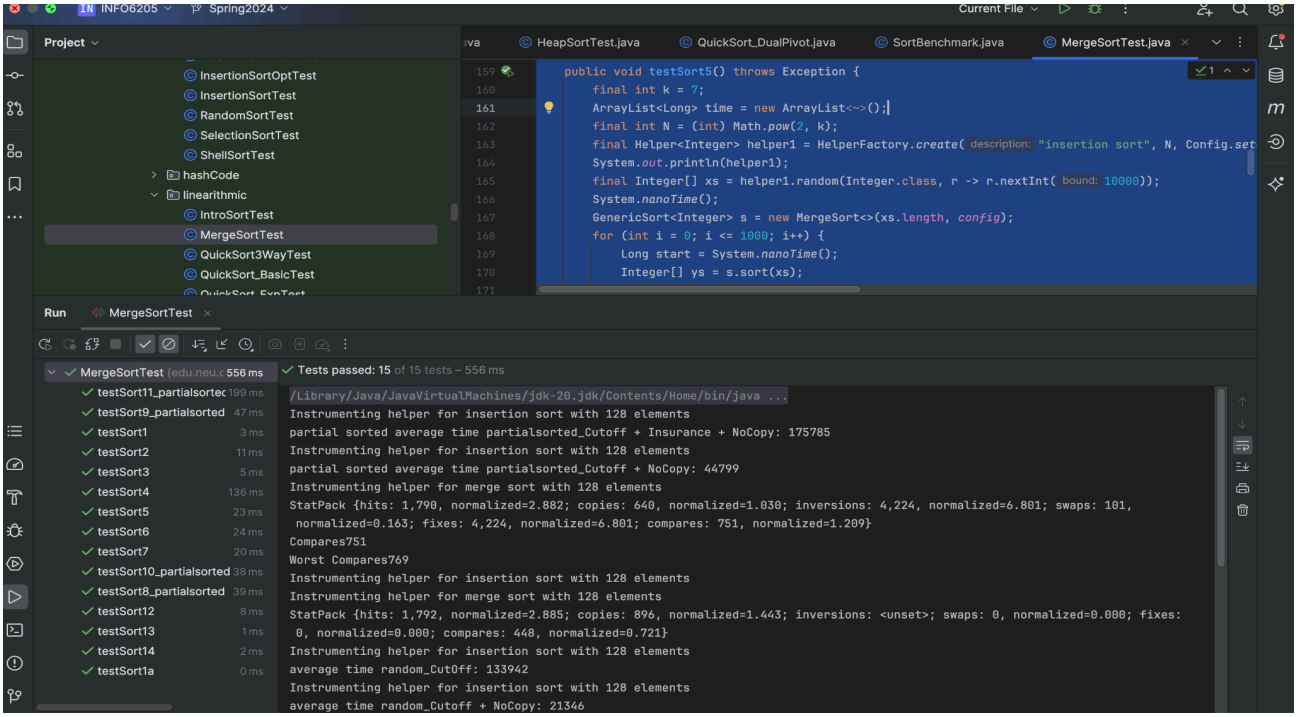
- HeapSort and MergeSort involve swaps, which indicates more memory operations compared to QuickSort Dual Pivot, which has fewer or no swaps.

- Merge Sort typically conducts the most comparisons, followed by Quick Sort and Heap Sort in the majority of instances. However, the difference in the number of comparisons between Merge Sort and Quick Sort gets smaller as the dataset gets bigger.

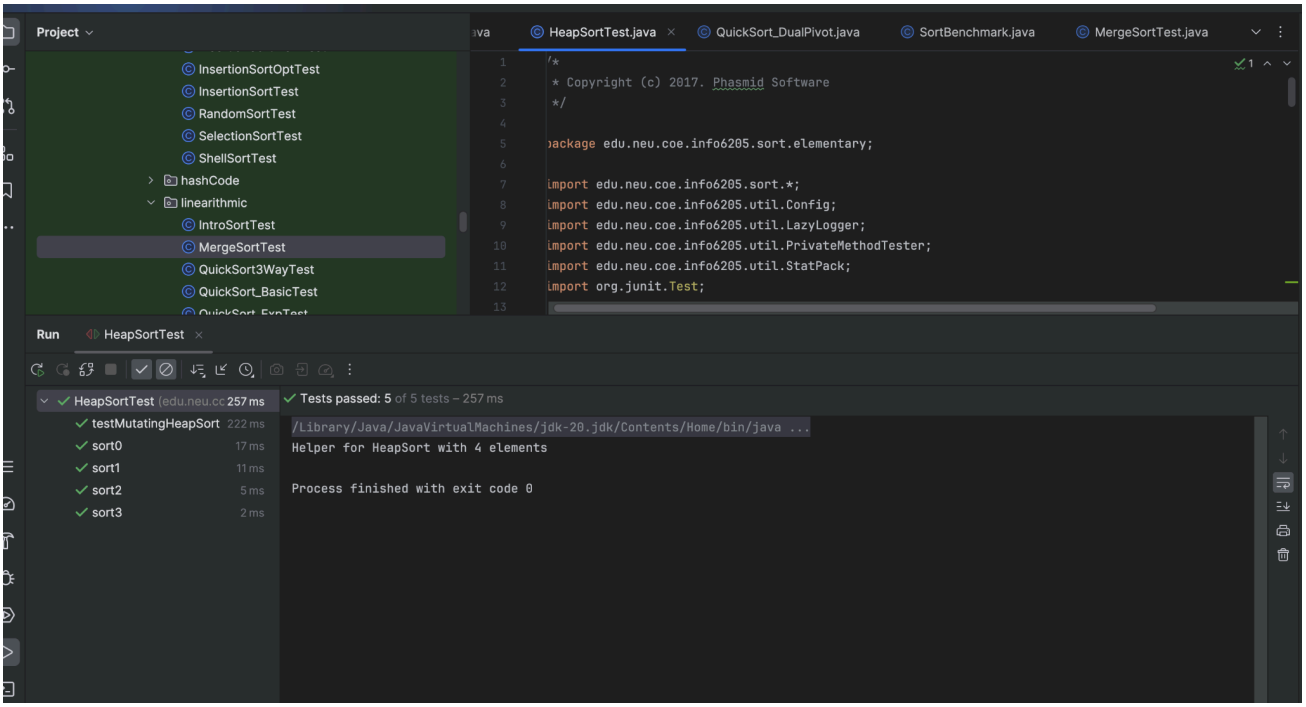
- Heap Sort is the algorithm that requires the most swaps in terms of frequency of operations; Quick Sort comes in second, while Merge Sort requires the least. Furthermore, with bigger datasets, the differences in swap operations between Heap Sort and Quick Sort become more noticeable.

- Combine Sort uses a method known as hits or merges, which is the process of merging two subsets together during the merging step. The count directly increases with the amount of the dataset.

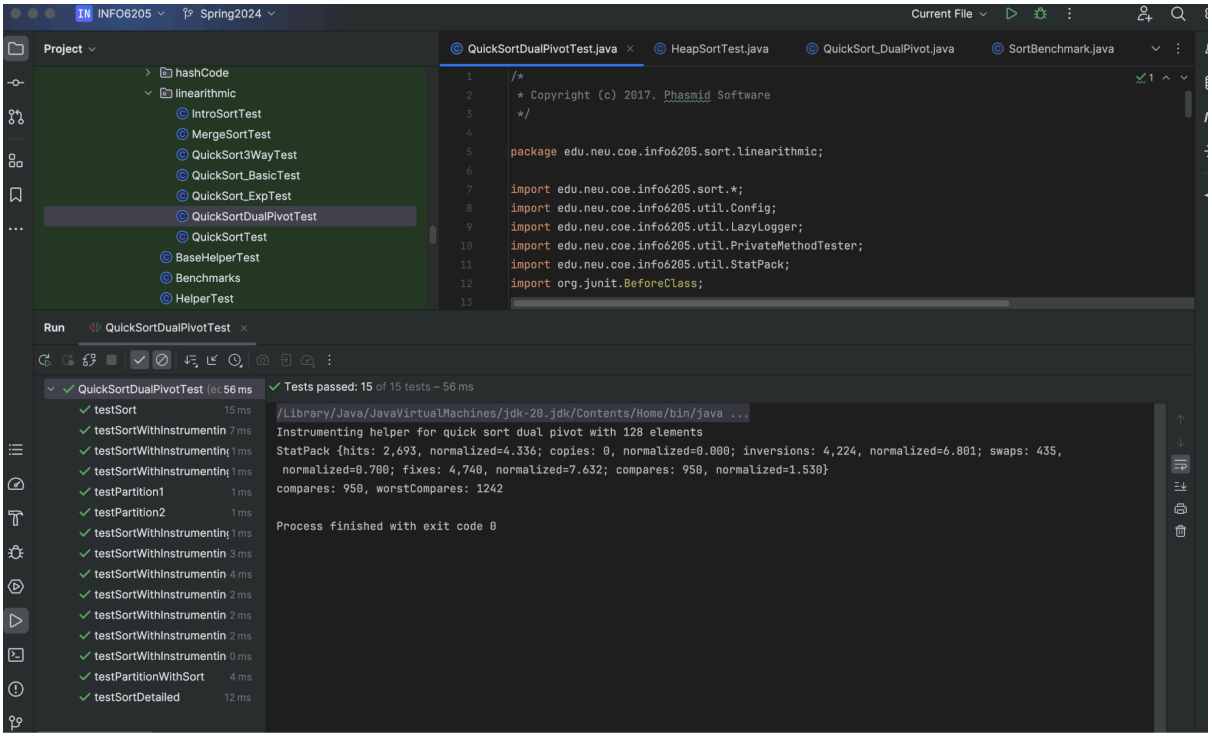
Unit Tests - Merge Sort



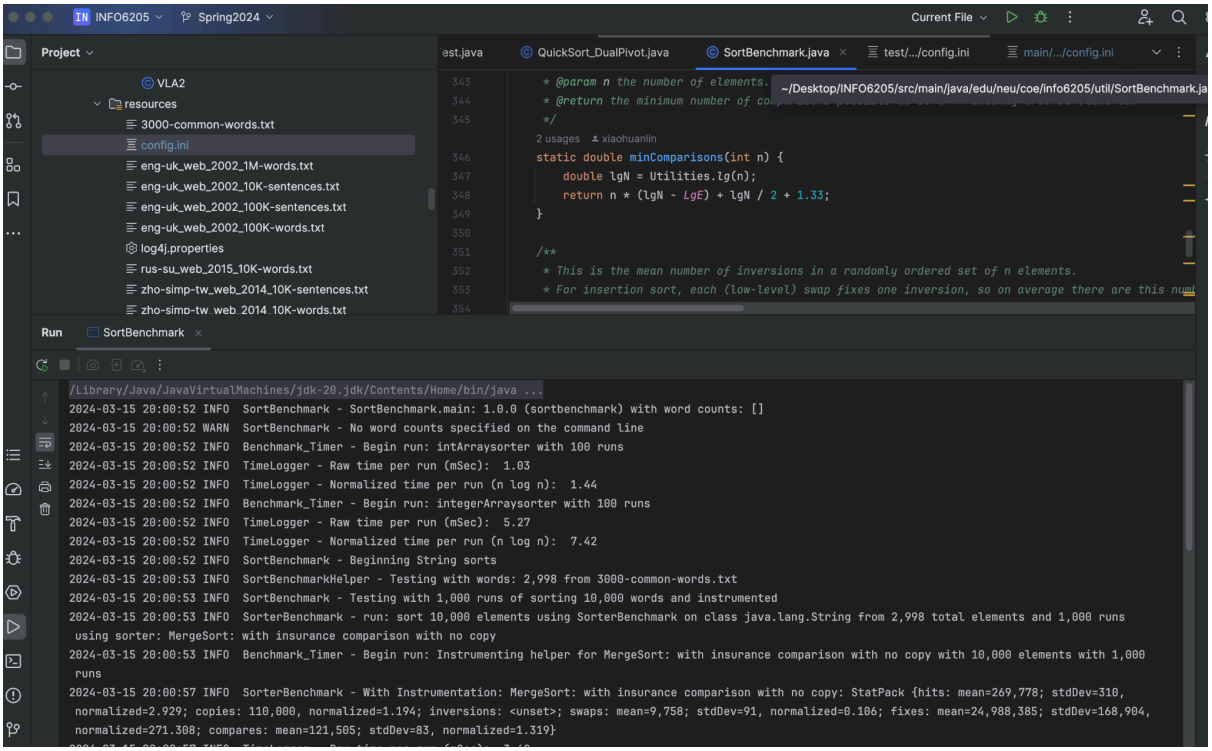
Heap Sort



Quick Sort Dual Pivot



SortBenchMark

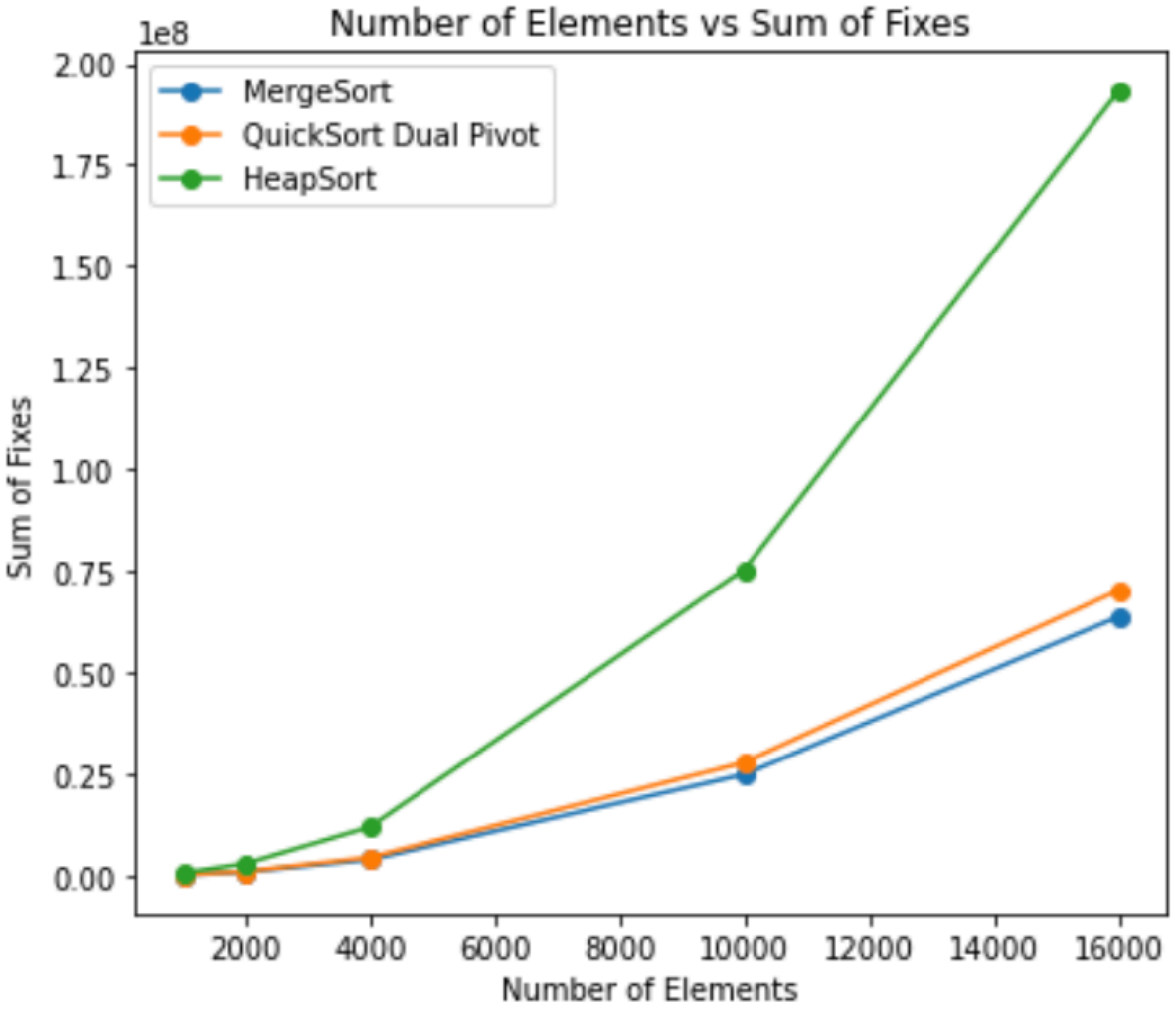


TABULAR OUTPUT

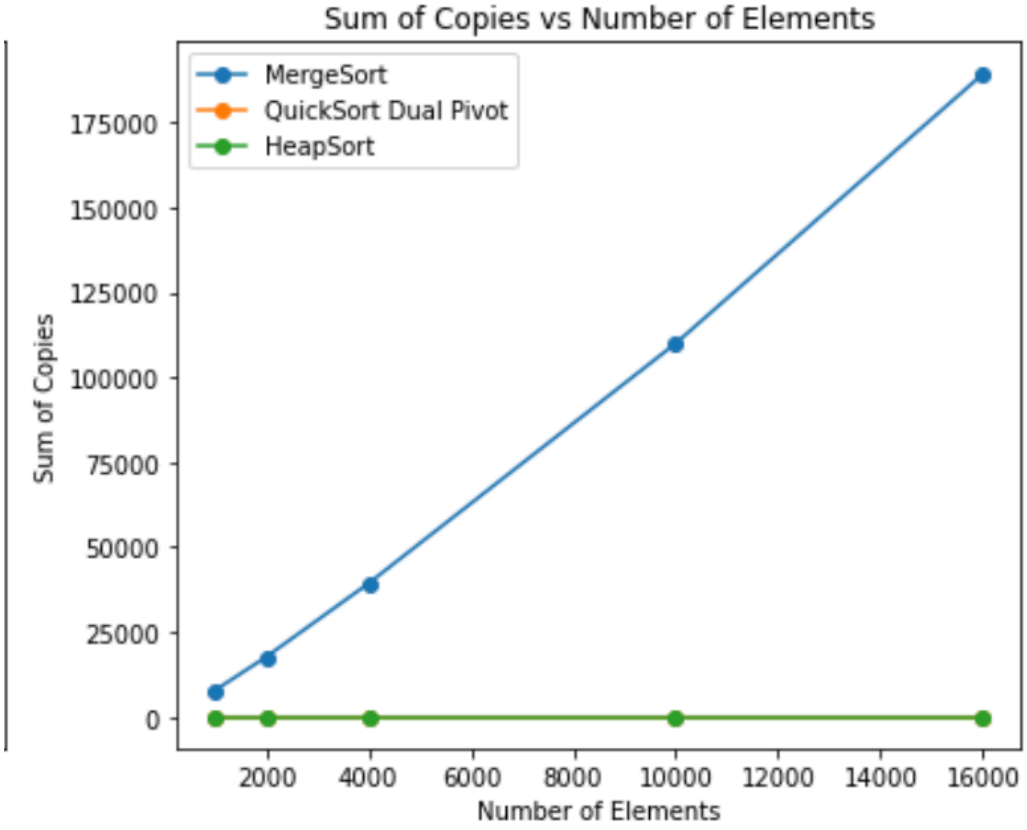
Sort Method	Number of Elements	Hits	Copies	Swaps	Fixes	Compares	NORMAL TIME	LOG TIME	Raw Time	log Number of Elements
MergeSo	1000	20158	7832	870	250592	8804	14.99	2.706	0.77	3.0
QuickSor pivot	1000	29702	0	4687	268343	10796	108.24	2.035	5.54	3.0
Heapsort	1000	70075	0	9086	748090	16865	205.11	2.312	10.50	3.0
MergeSo	2000	44279	1766	1729	999228	19589	17.64	2.875	2.02	3.301
QuickSor pivot	2000	67536	0	1070	109887	24283	189.96	2.829	21.72	3.301
Heapsort	2000	15610	0	2016	300666	37720	361.77	2.557	41.36	3.301
MergeSo	4000	96592	3932	3467	399728	43218	10.36	2.338	2.62	3.602
QuickSor pivot	4000	14758	0	2271	456960	55382	364.23	2.562	92.00	3.602
Heapsort	4000	34430	0	4435	120645	83450	646.59	2.811	163.33	3.602
MergeSo	10000	26977	1100	9758	249883	121505	5.19	0.715	3.69	4.0
QuickSor pivot	10000	40859	0	6095	279493	158608	779.21	2.891	553.82	4.0
Heapsort	10000	96749	0	1241	755198	235369	1444.44	3.160	1026.64	4.0

MergeSo	16000	45096	1893	1405	638560	204991	8.69	1.942	8.69	4.204
QuickSor pivot	16000	67398	0	9734	703893	271455	1354.41	3.130	1354.41	4.204
Heapsort	16000	16323	0	2092	193300	397680	2745.60	3.440	2745.60	4.204

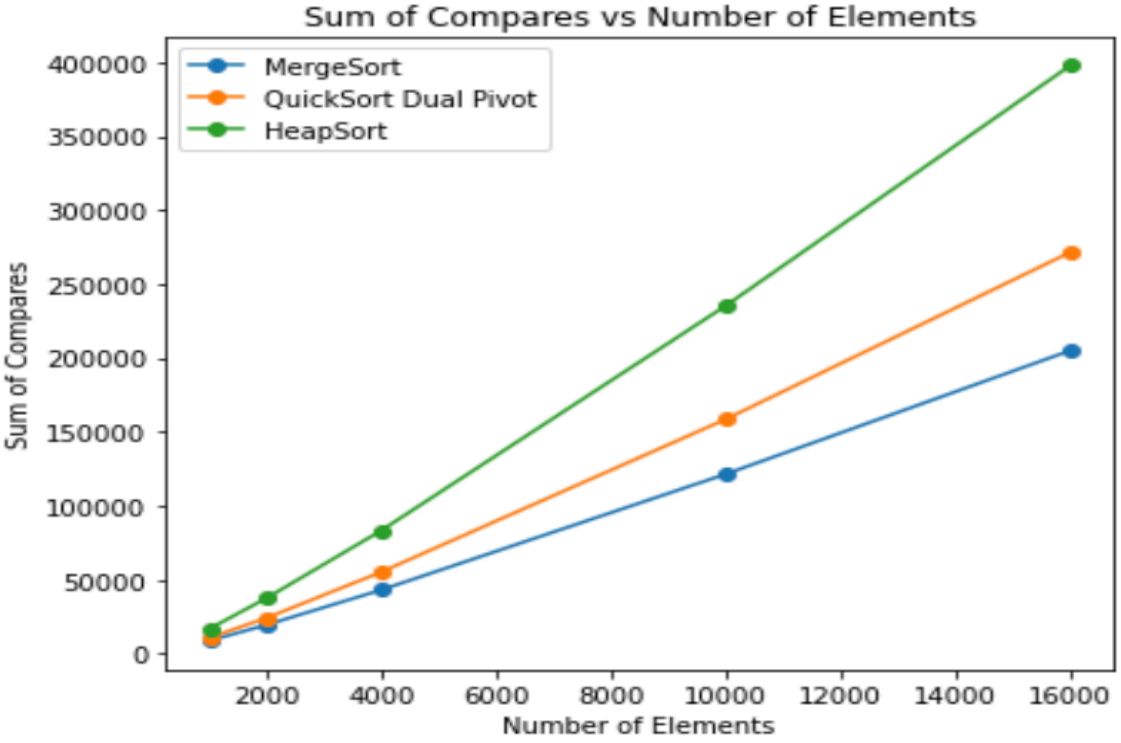
Graphical Output



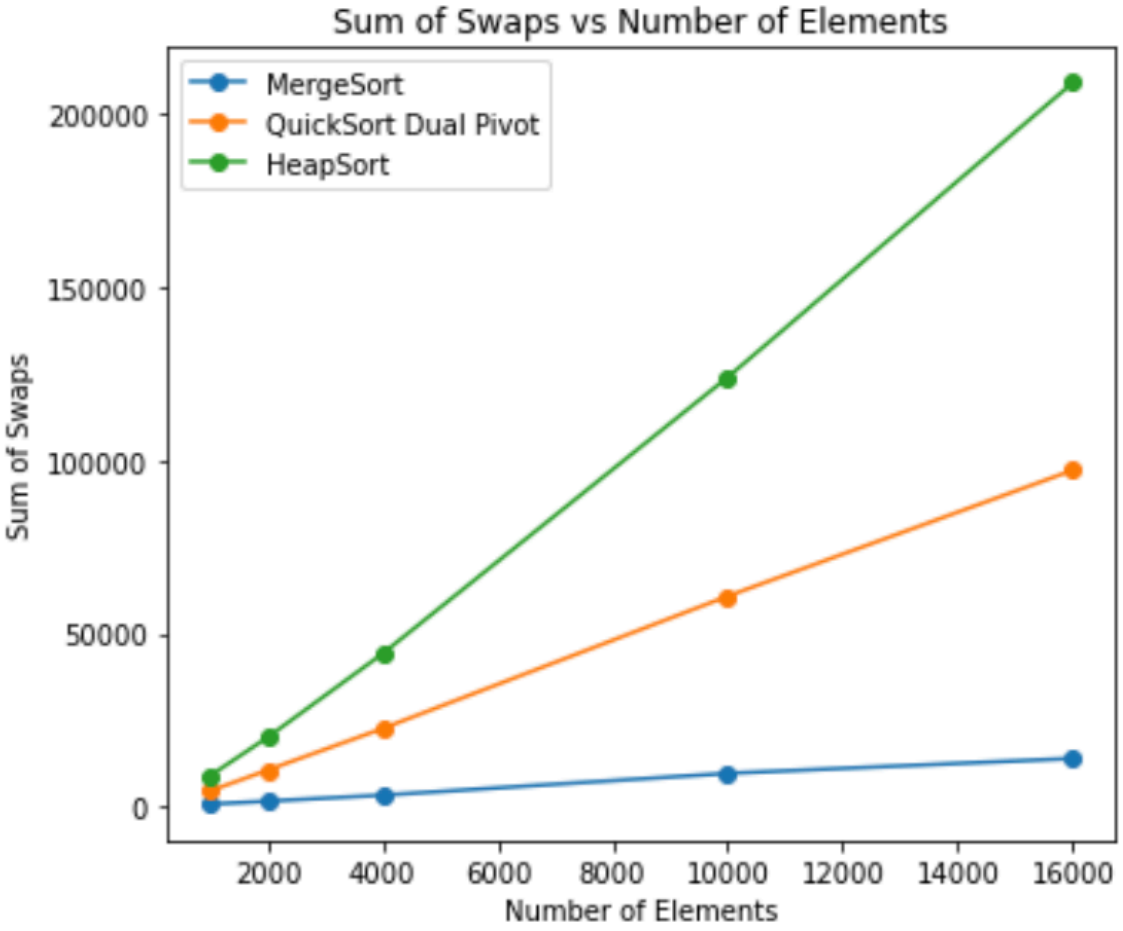
2)


Sum of Copies vs Number of Elements
MergeSort
QuickSort Dual Pivot
HeapSort
Sum of Copies
0
25000
50000
75000
100000
125000
150000
175000
2000
4000
6000
8000
10000
12000
14000
16000
Number of Elements

3)


Sum of Compares vs Number of Elements
MergeSort
QuickSort Dual Pivot
HeapSort
Sum of Compares
0
50000
100000
150000
200000
250000
300000
350000
400000
2000
4000
6000
8000
10000
12000
14000
16000
Number of Elements

4)


Sum of Swaps vs Number of Elements
MergeSort
QuickSort Dual Pivot
HeapSort
200000
150000
100000
50000
0
Sum of Swaps
2000
4000
6000
8000
10000
12000
14000
16000
Number of Elements