Program Structures and Algorithms
Spring 2023(SEC –1)

NAME: Prem Kumar Raghava Manoharan
NUID: 002726784

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), or something else.

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).

Relationship Conclusion:

For Merge Sort, the number of comparisons tends to be a good predictor of the total execution time, as the algorithm's running time is dominated by comparisons. However, the number of swaps and array accesses (hits) are also important factors to consider, especially as the input size increases.

For Heap Sort, the number of swaps tends to be a good predictor of the total execution time, as the algorithm is primarily concerned with moving elements around in the heap. However, the number of array accesses (hits) is also important, as each swap involves accessing elements in the array.

Dual Pivot Quick Sort is an optimized version of Quick Sort that uses two pivots to partition the input array, rather than just one. In this case, the number of comparisons and swaps are still important factors to consider, but the specifics of the algorithm can also influence the total execution time.

For Dual Pivot Quick Sort, the number of comparisons is still an important predictor of the total execution time, as the algorithm is still dominated by comparison operations. However, the number of swaps/copies can be less important than in standard Quick Sort, as Dual Pivot Quick Sort tends to do fewer swaps. Instead, the number of array accesses (hits) can be more important, as Dual Pivot Quick Sort accesses elements in the array more frequently than standard Quick Sort.

Additionally, the choice of pivot elements can also impact the total execution time of Dual Pivot Quick Sort. In some cases, choosing the wrong pivot elements can result in poor performance, such as when the pivots are selected from the same or similar ranges of the input array. Therefore, the choice of pivot elements can be an important factor to consider when analysing the total execution time of Dual Pivot Quick Sort.

Evidence to support that conclusion:

Merge Sort :

	Merge Sort	Merge Sort	Merge Sort	Merge Sort	Merge Sort
N	Hits	Copies	Swaps	Fixes	Compares
10	91	20	10	23	24
20	264	80	20	95	66
40	683	240	40	390	170
80	1687	640	80	1580	417
160	4014	1600	160	6360	992
320	9307	3840	320	25521	2302
640	21175	8960	640	102213	5343
1280	47472	20480	1280	409224	11764

Heap Sort:

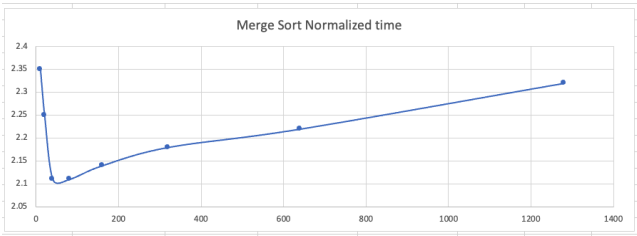
	Heap Sort	Heap Sort	Heap Sort	Heap Sort	Heap Sort
N	Hits	Copies	Swaps	Fixes	Compares
10	184	0	27	47	39
20	516	0	72	229	115
40	1338	0	181	1029	306
80	3300	0	440	4407	770
160	7863	0	1036	18352	1859
320	18271	0	2390	75147	4356
640	41644	0	5416	304691	9990
1280	93513	0	12108	1228374	22540

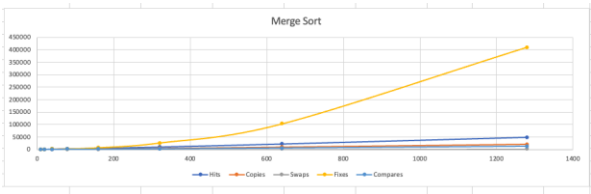
Dual Pivot Quick Sort:

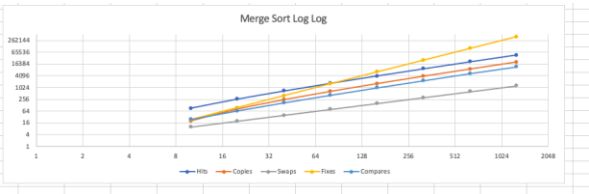
	Quick Sort	Quick Sort	Quick Sort	Quick Sort	Quick Sort
N	Hits	Copies	Swaps	Fixes	Compares
10	82	0	14	19	25
20	225	0	38	88	72
40	579	0	96	388	192
80	1429	0	234	1652	487
160	3413	0	553	6869	1186
320	7948	0	1278	28110	2804
640	18172	0	2906	113935	6484
1280	40892	0	6507	458719	14730

Graphical Representation:

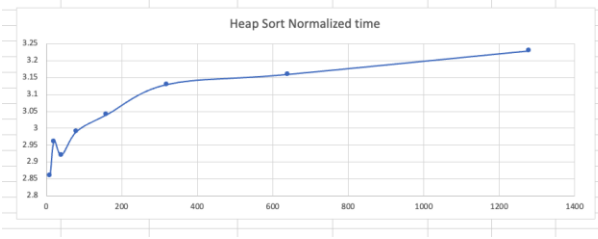
Merge sort:

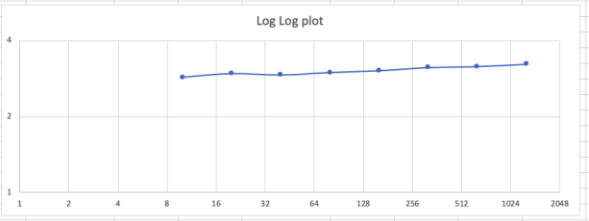


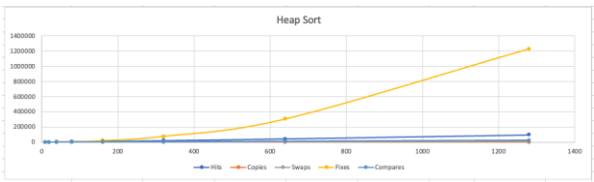


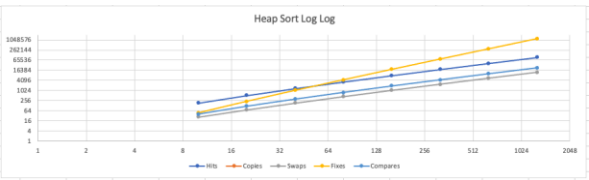


Heap Sort:

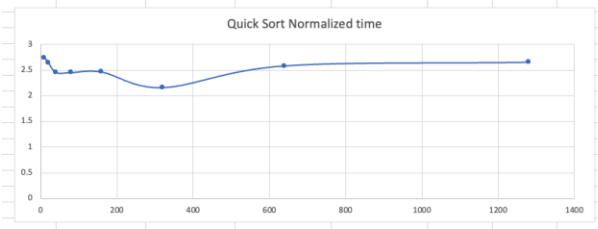


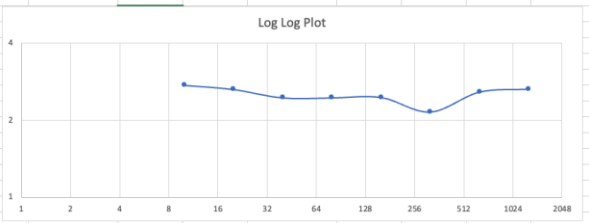


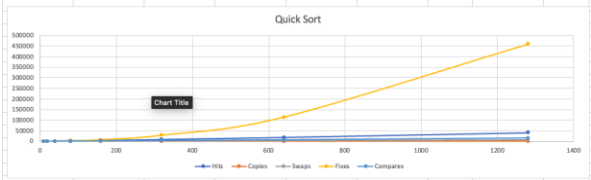


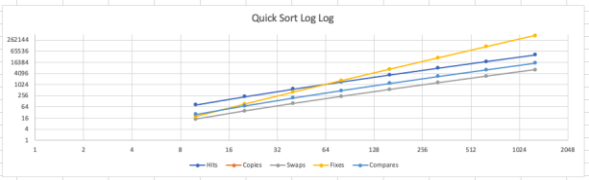


Dual-pivot Quick sort:









Unit Test Screenshots: