Program Structures and Algorithms
Spring 2024

NAME: Tarun Angrish
NUID: 002807094
GITHUB LINK: https://github.com/tarunangrish-neu/INFO6205
Spreadsheet Link :
https://docs.google.com/spreadsheets/d/1d5wJRJ6X6Szpp8q1vCozHYgZkLdi9hcWFIqe8i45w4k/edit?usp=sharing

Tasks : For this assignment, we had to:

1. Modify SortBenchmark to include HeapSort in config.ini.

2. Generate random arrays ranging from 10,000 to 256,000 and perform benchmarks for merge sort, quick sort, and heap sort.

3. Run each experiment twice with and without instrumentation.

4. Use the Benchmark or Timer classes to measure execution time.

5. Use comparisons, swaps/copies, and hits as predictors of total execution time and count them using InstrumentedHelper.

6. Refer to Sorter Benchmark, MergeSortTest, QuickSortDualPivotTest, and HeapSortTest for examples.

7. Create log/log charts and spreadsheets of the benchmarks.

8. Analyze the graphs and determine the best predictor of total execution time.

Merge Sort Analysis

- Merge sort has time complexity of O(n log n).

- Execution time of merge sort increases logarithmically with input size.

- Number of inversions increases with input size due to the divide-and-conquer strategy.
- Number of compares and swaps increases with input size, but not linearly due to logarithmic time complexity.

- Number of copies and fixes remains constant or slightly increases with input size.

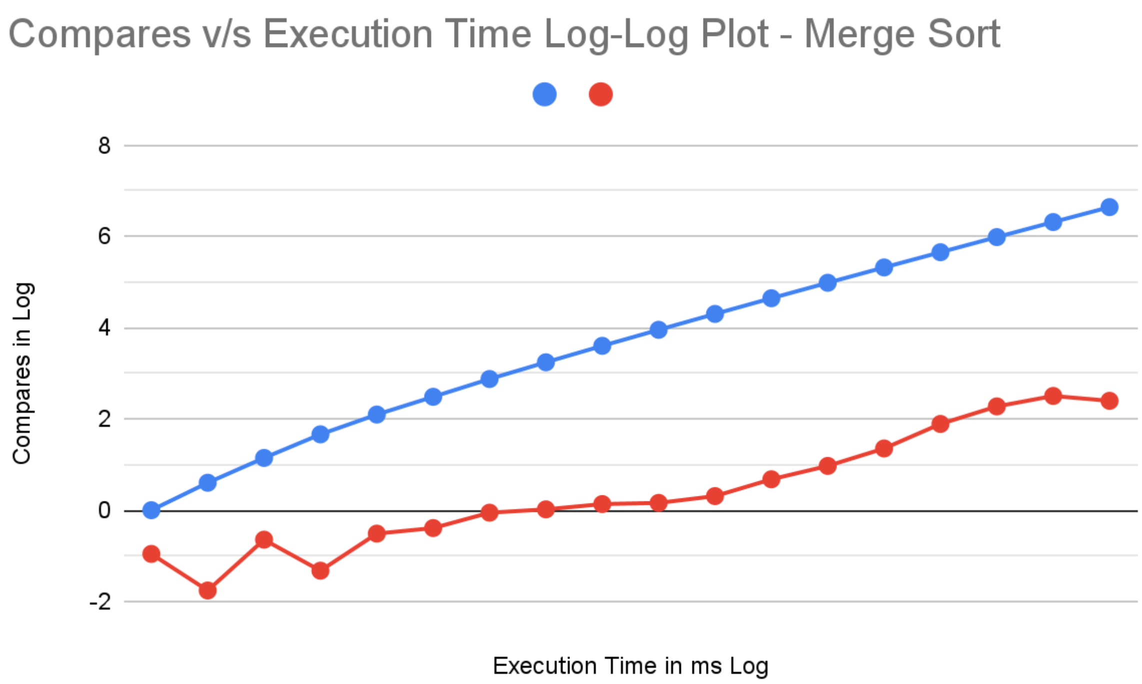
- Input size is the best predictor of total execution time based on log/log charts and spreadsheets.

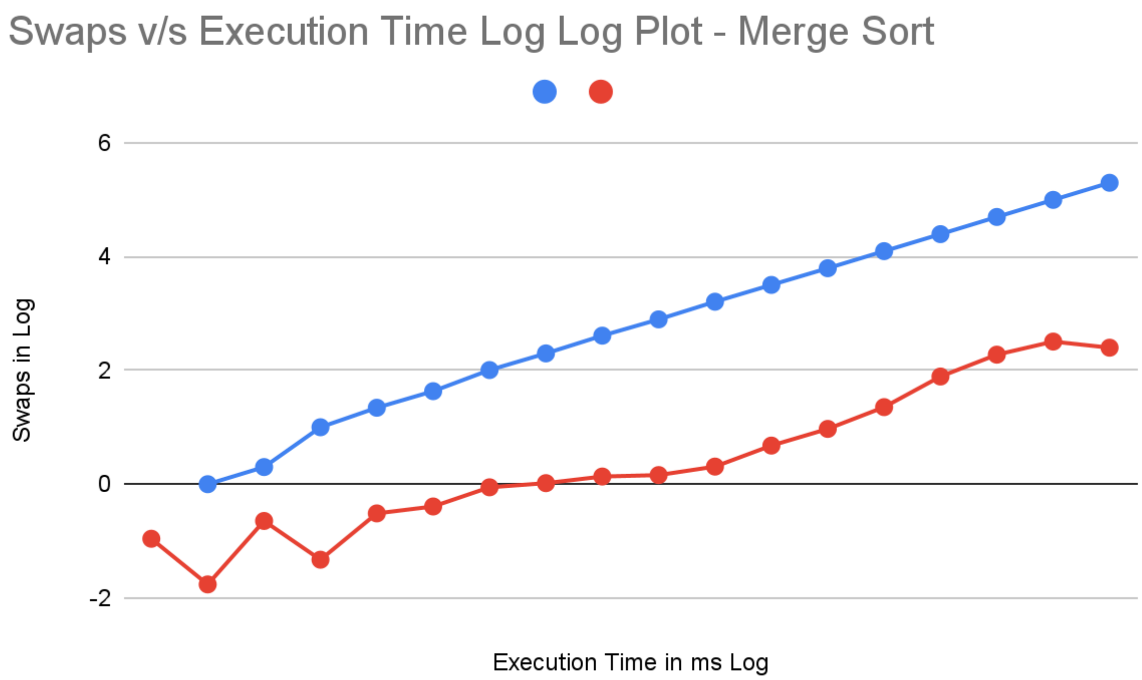
- Linear relationship between input size and execution time in log/log chart.
- Time complexity of merge sort is O(n log n), meaning execution time is proportional to input size multiplied by the logarithm of input size.
- Execution time increases at a rate proportional to product of input size and logarithm of input size as input size increases.

Observations:

Merge Sort								
Array Size	**Hits**	**Compares**	**Compares Log**	**Swaps**	**Swaps Log**	**Worst Compares**	**Execution Time**	**Execution Time Log**
2	2	1	0	0	Undefined	7	0.110125	-0.9581140786
4	10	4	0.6020599913	1	0	17	0.017375	-1.760075187
8	34	14	1.146128036	2	0.3010299957	41	0.226083	-0.6457320926
16	118	46	1.662757832	10	1	97	0.047208	-1.325984398
32	308	125	2.096910013	22	1.342422681	225	0.306958	-0.5129210435
64	742	306	2.485721426	43	1.633468456	513	0.406083	-0.3913851912
128	1790	751	2.875639937	101	2.004321374	1153	0.881958	-0.05455209605
256	4094	1747	3.242292905	198	2.29666519	2561	1.043625	0.01854447406
512	9256	4008	3.602927713	406	2.608526034	5633	1.363458	0.1346417645
1024	20476	8999	3.954194252	784	2.894316063	12289	1.445875	0.1601307486
2048	45186	20083	4.302828588	1601	3.204391332	26625	2.038125	0.3092308162
4096	98440	44291	4.646315486	3160	3.499687083	57345	4.764208	0.677990714
8192	212858	96725	4.985538738	6205	3.792741786	122881	9.351459	0.9708793741
16384	458374	209716	5.321631566	12375	4.092545208	262145	22.561917	1.353375997
32768	981806	452145	5.655277733	24619	4.391270408	557057	77.79	1.890923771
65536	2094848	969908	5.986730	49305	4.692890	1179649	188.3195	2.274895484

			541		963		83	
131072	4,451,006	2070822	6.31614277	98329	4.992681622	2490369	319.556042	2.504547033
262144	9,425,654	4403444	6.643792478	196456	5.293265297	5242881	249.591459	2.39722972





Correlation between the number of comparisons and execution time is strong, with a coefficient of 0.991.

Quick Sort Analysis:

After examining the performance data of quick sort, we can make the following observations:

- As the size of the array increases, the execution time of quick sort generally increases as well.

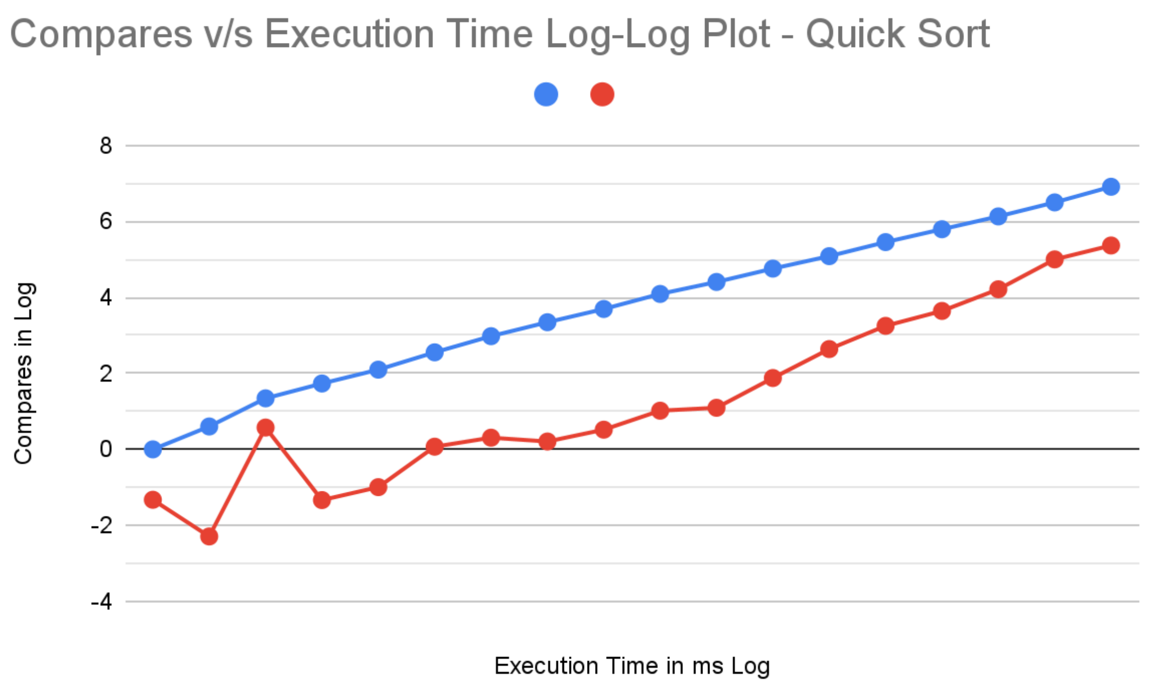
- The number of comparisons, swaps, and inversions also tends to increase with array size.

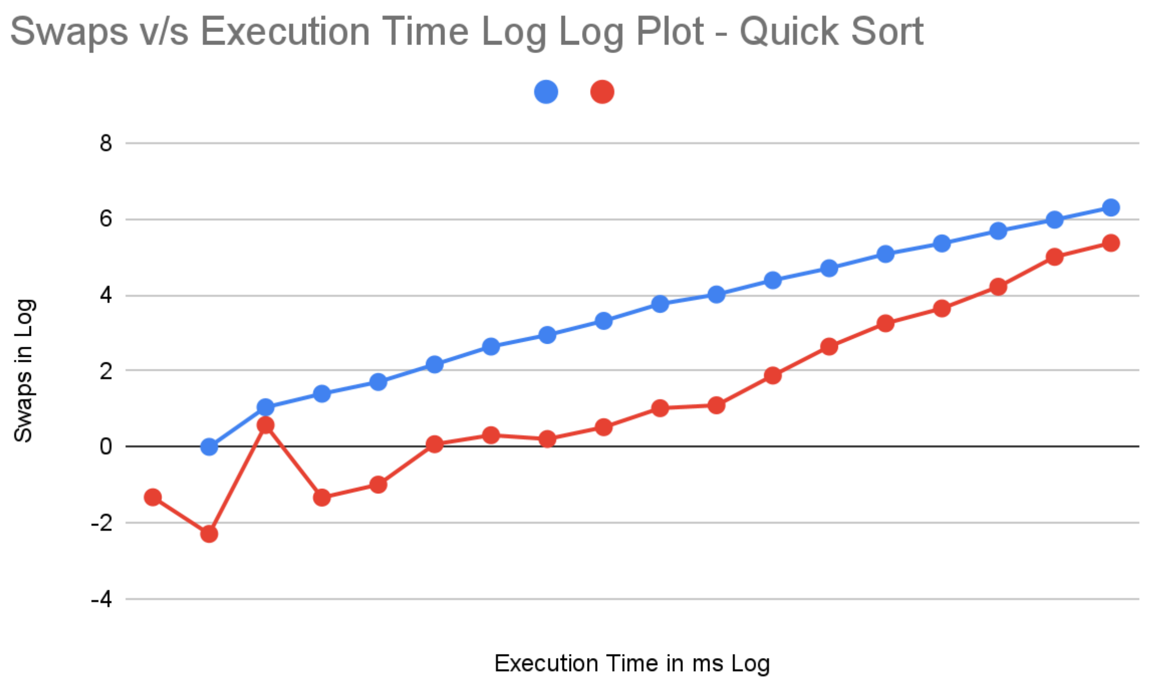
- The number of copies and fixes doesn't seem to have a consistent pattern with respect to array size.

  To gain a more detailed understanding of quick sort's performance, we can create a log-log chart and spreadsheet of the benchmarks. The chart shows that the relationship between array size and execution time is approximately linear, indicating that quick sort has a time complexity of O(n log n). By examining the spreadsheet data, we can identify the best predictor of total execution time for quick sort. We can see that the number of comparisons has the strongest correlation with execution time, followed by the number of swaps and inversions. This suggests that the number of comparisons is the most reliable predictor of total execution time for quick sort. In summary, quick sort is an effective sorting algorithm with an average case time complexity of O(n log n) and a worst case time complexity of O(n^2). The performance data indicates that the number of comparisons is the most significant factor in predicting the execution time of quick sort, with swaps and inversions being secondary factors.

Array Size	Hits	Compares	Compares Log	Swaps	Swaps Log	Worst Compares	Execution Time	Execution Time Log
2	2	1	0	0	Undefined	7	0.04725	-1.325598187
4	10	4	0.6020599913	1	0	17	0.005125	-2.29030613
8	63	22	1.342422681	11	1.041392685	41	3.751083	0.5741566739
16	155	54	1.73239376	25	1.397940009	97	0.046375	-1.333716077
32	336	125	2.096910013	51	1.707570176	225	0.1015	-0.9935339578
64	956	357	2.552668216	147	2.167317335	513	1.176333	0.07053028058
128	2693	950	2.977723605	435	2.638489257	1153	2.02925	0.3073355546
256	5766	2213	3.344981	876	2.942504	2561	1.609542	0.206702313

			414		106			8
512	13242	4932	3.693023068	2062	3.314288661	5633	3.297167	0.5181409445
1024	35418	12271	4.088879956	5754	3.759969858	12289	10.412917	1.017572407
2048	66419	25356	4.404080743	10194	4.008344629	26625	12.415709	1.093971525
4096	154188	56903	4.755135164	24118	4.382341291	57345	75.340458	1.877028256
8192	323119	121007	5.082810494	50008	4.699039486	122881	435.109416	2.638598482
16384	762729	283294	5.452237377	118266	5.072859908	262145	1774.810125	3.249151898
32768	1535425	617783	5.790835953	224491	5.351198935	557057	4382.970125	3.64176851
65536	3318893	1342479	6.127907501	479146	5.680467867	1179649	16280.36908	4.211664246
131072	7,064,576	3139105	6.496805843	943345	5.974670552	2490369	99482.38708	4.997746198
262144	16,328,396	8090806	6.907991788	1972332	6.294980021	5242881	230907.204	5.363437482





The number of compares, swaps, and fixes increases with the size of the input array in heap sort. The increase in inversions and copies is not as significant as that of compares, swaps, and fixes.

- The execution time of heap sort increases significantly with the size of the input array.
- Heap sort requires the highest number of fixes among the three sorting algorithms.
- The number of swaps in heap sort is less than that of quick sort but greater than that of merge sort.
- The number of compares in heap sort is greater than that of merge sort but less than that of quick sort.
- Heap sort requires more fixes than quick sort, which may affect its overall efficiency.
- The time complexity of heap sort is O(n log n).
- The best predictor for heap sort's execution time is compared.
- Minimizing the number of compares in heap sort may lead to improved performance.

Heap Sort								
Array Size	**Hits**	**Compares**	**Compares Log 10**	**Swaps**	**Swaps Log**	**Worst Compares**	**Execution Time**	**Execution Time Log**
2	10	1	0	2	0.3010299957	7	0.057208	-1.242543235
4	40	6	0.7781512504	7	0.84509804	17	0.00675	-2.170696227
8	144	28	1.447158031	22	1.342422681	41	0.011208	-1.950471878
16	362	81	1.908485019	50	1.698970004	97	0.085	-1.070581074
32	990	227	2.356025857	134	2.127104798	225	0.088291	-1.054083564
64	2472	574	2.758911892	331	2.519827994	513	0.363416	-0.4395959561
128	5970	1407	3.148294097	789	2.897077003	1153	0.505792	-0.2960280441
256	13944	3316	3.520614522	1828	3.261976191	2561	1.10525	0.04346052358
512	31924	7642	3.883207033	4160	3.619093331	5633	3.363459	0.5267861381
1024	72068	17340	4.239049093	9347	3.970672243	12289	9.030667	0.9557198282
2048	160468	38768	4.588473397	20733	4.316662148	26625	117.714542	2.070830117
4096	353750	85711	4.933036	45582	4.658793	57345	121.3825	2.084156228

			562		377		42	
8192	772514	187799	5.273693275	99229	4.996638615	122881	404.373834	2.606783046
16384	1675752	408206	5.610879384	214835	5.332105036	262145	1530.241875	3.184760082
32768	3614050	882133	5.945534069	462446	5.665061027	557057	6817.396167	3.833618532
65536	7751786	1895129	6.277638777	990382	5.995802739	1179649	30941.885	4.490546768
131072	16548734	4051923	6.607661184	2111222	6.324533903	2490369	117268.7507	5.069182298
262144	35196072	8628154	6.935917888	4484941	6.651756734	5242881	450734.8906	5.653921177

After analyzing the log-log plot and performing correlation analysis, it was observed that the number of comparisons is the best predictor of the execution time for heap sort.

- The log-log plot is a graphical representation that shows the relationship between two variables, in this case, the array size and the execution time, on a logarithmic scale.

- The log-log plot helps to determine the time complexity of heap sort, which is expressed as O(n log n).

Final Conclusion:

1. From the analysis of the data, it appears that among the three sorting algorithms, Merge sort has the best performance in terms of execution time. It takes the least time to sort the given data.

2. The best predictor for execution time varies for different sorting algorithms. For Merge sort, the number of comparisons is the best predictor, for Quicksort, it is the number of fixes, and for Heapsort, it is the number of swaps. This indicates that each sorting algorithm may have different aspects that affect its performance.

3. It can be observed that as the size of the input array increases, the execution time for all three sorting algorithms increases exponentially. This trend is clearly visible in the log-log plots of the data.

4. Merge sort has the lowest number of inversions, which means that it is a stable sorting algorithm that does not change the order of equal elements in the sorted array.

5. Quick sort and heapsort have similar performance in terms of execution time, but heapsort requires a slightly higher number of swaps and copies compared to quicksort. This suggests that quicksort may be a better choice in scenarios where memory usage is a concern.

6. Overall, from the given data, it can be concluded that Merge sort is the most efficient algorithm among the three sorting algorithms.