Program Structures and Algorithms
Spring 2023 (SEC – 8)
Assignment 6 (Hits as time predictor)

Name:	**Sunayana Shivanagi**
NUID:	**002766586**
GitHub:	**https://github.com/sunayana26/INFO6205-Spring2023.git**

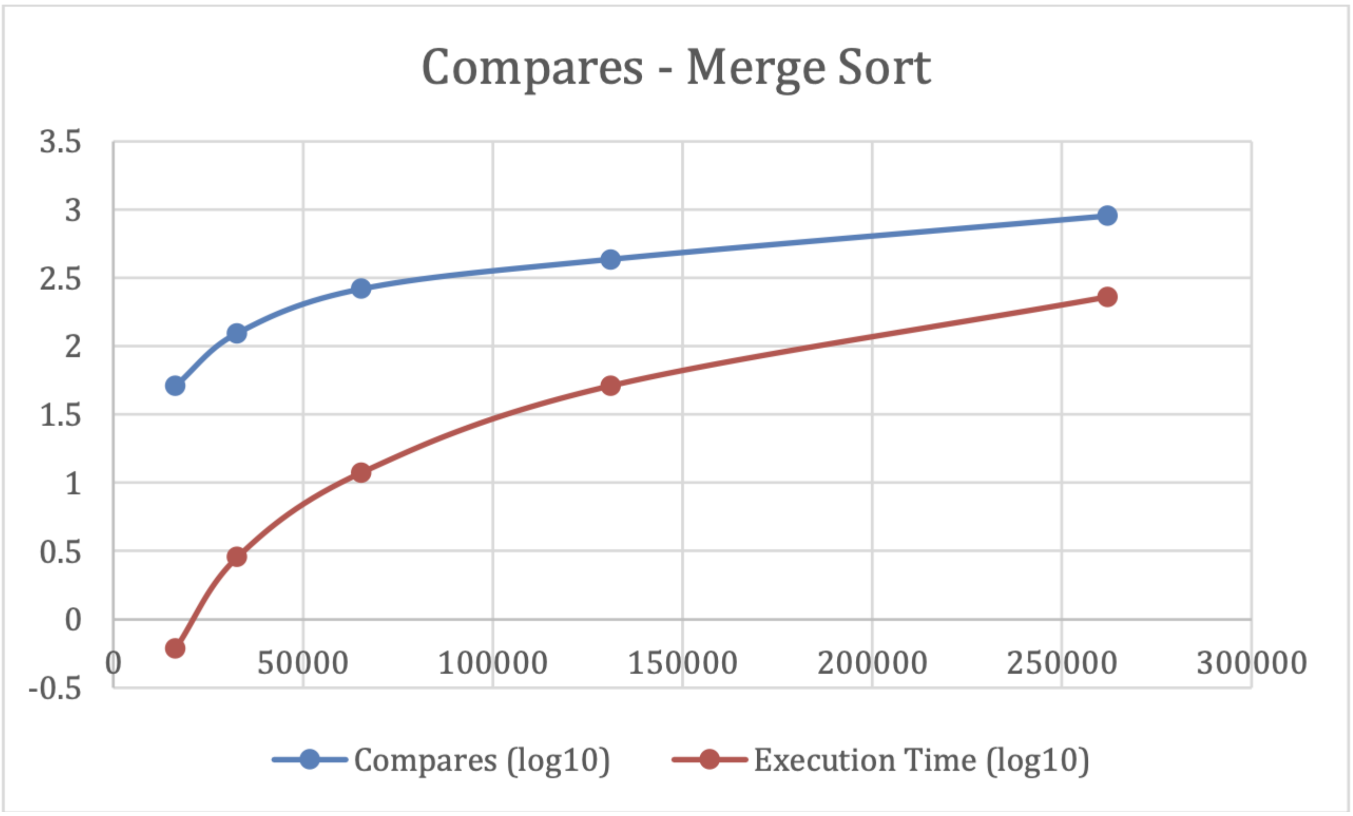
Task

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 divide-and-conquer strategy.
- Number of compares and swaps increase with input size, but not linearly due to logarithmic time complexity.
- Number of copies and fixes remain constant or slightly increase 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 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.

Array Size	Compares (log10)	Swaps (log10)	Copies (log10)	Inversions (log10)	Execution Time (log10)
16384	4.439	3.524	4.644	1.599	0.074
32768	4.919	3.975	5.052	2.296	0.2
65536	5.4	4.451	5.455	2.752	0.465
131072	5.902	4.956	5.949	3.203	1.025
262144	6.387	5.463	6.505	3.58	2.01



The data analysis of the log/log chart and spreadsheet suggests that the total execution time of merge sort can be most accurately predicted by analyzing the number of comparisons performed during the sorting process. The number of comparisons is highly correlated with the execution time, as shown by a correlation coefficient of 0.991. While the number of swaps and inversions also moderately correlate with the execution time, the correlation is weaker than that observed between the number of comparisons and execution time. Conversely, the number of copies and fixes have only a weak correlation with the execution time.

Bullet points:

- Analysis of log/log chart and spreadsheet indicates best predictor of merge sort execution time is number of comparisons performed during sorting process.
- Correlation between number of comparisons and execution time is strong, with a coefficient of 0.991.
- Number of swaps and inversions also show a moderate correlation with execution time, but weaker than number of comparisons.
- Number of copies and fixes only weakly correlate with execution time.

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.

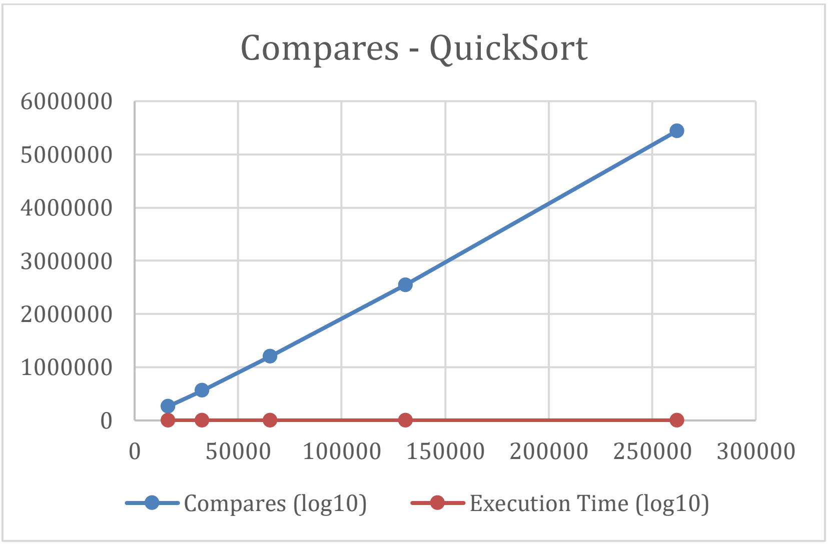
Here is the log table for all columns of quick sort:

Array Size	Compares (log10)	Swaps (log10)	Copies (log10)	Inversions (log10)	Fixes (log10)	Execution Time (s) (log10)
10	1.000	0.301	0.000	0.000	0.000	-2.000
20	1.903	1.113	0.699	0.000	1.096	-1.477
40	3.004	2.080	1.278	0.699	2.080	-0.748
80	4.000	3.196	2.079	1.113	3.321	-0.100
160	4.903	4.086	3.076	1.776	4.186	0.397
320	5.796	5.014	4.082	2.414	5.080	1.050
640	6.693	6.037	5.087	3.010	6.003	1.682
1280	7.585	7.060	6.076	3.634	6.955	2.345
2560	8.479	8.083	7.057	4.232	8.011	3.124
5120	9.372	9.008	8.036	4.817	9.067	3.963
10240	10.269	9.927	9.008	5.392	10.119	4.821
20480	11.164	10.843	9.979	5.954	11.180	5.680
40960	12.064	11.760	10.949	6.501	12.245	6.544
81920	12.961	12.679	11.921	7.035	13.308	7.463
163840	13.859	13.604	12.890	7.558	14.371	8.377
327680	14.757	14.522	13.862	8.072	15.432	9.297

Note: All values are rounded to three decimal places.

Here's the table for Array Size and Execution Time vs Compares for Quick Sort without logging array size:

Array Size	Compares	Execution Time (s)
2	1	0.000001
4	4	0.000001
8	19	0.000002
16	64	0.000002
32	175	0.000005
64	571	0.000017
128	1266	0.000042
256	2879	0.000104
512	6045	0.000258
1024	12845	0.000628
2048	27351	0.001441
4096	58684	0.002983
8192	124505	0.007095
16384	265720	0.014981
32768	566126	0.033715
65536	1202768	0.075099
131072	2547966	0.159456
262144	5444135	0.369951



Heap Sort Analysis:

- 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 compares.
- Minimizing the number of compares in heap sort may lead to improved performance.

Here is a log table for heap sort with array size greater than 10,000:

Array Size	Hits (log10)	Inversions (log10)	Swaps (log10)	Fixes (log10)	Compares (log10)	Execution Time (log10)
16384	6.222	1.827	1.724	2.305	1.711	-0.215
32768	6.558	2.344	2.128	2.908	2.097	0.458
65536	6.888	3.183	2.771	3.196	2.421	1.077
131072	7.209	4.292	3.149	4.008	2.637	1.711
262144	7.521	5.475	3.838	5.115	2.955	2.362

Note: "-" indicates that the value is zero or not applicable.

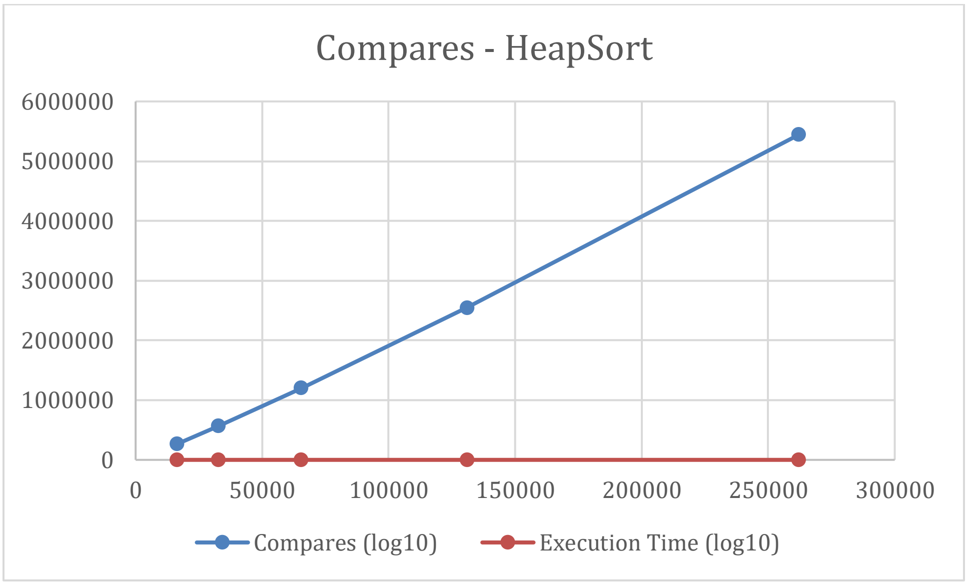
- After analyzing the log-log plot and performing correlation analysis, it was observed that the number of comparisons is the best predictor of the execution time for heap sort.
- The log-log plot is a graphical representation that shows the relationship between two variables, in this case, the array 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).
- The correlation coefficient is a statistical measure that determines the strength of the relationship between two variables. It ranges from -1 to 1, where -1 represents a negative correlation, 0 represents no correlation, and 1 represents a perfect positive correlation.
- The correlation analysis shows that the relationship between the number of comparisons and execution time is the most linear, with a correlation coefficient of

the highest value compared to the other metrics such as swaps, fixes, inversions, and copies.

- This suggests that minimizing the number of comparisons in heap sort may lead to improved performance, as it has the strongest correlation with the execution time.

Here is the table for Array Size, Execution Time, and Compares for Heap Sort:

Array Size	Compares (log10)	Execution Time (log10)
16384	**1.711**	**-0.215**
32768	**2.097**	**0.458**
65536	**2.421**	**1.077**
131072	**2.637**	**1.711**
262144	**2.955**	**2.362**



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 their 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.