Program Structures and Algorithms
Spring 2024

NAME: Aneesh Arunjunai Saravanan
NUID: 002675639
GITHUB LINK: https://github.com/aneesharunjunai/INFO6205

Task: Assignment 6 - Hits as time predictor

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).

Observation & Conclusion:

The benchmark results highlight the trade-offs inherent in choosing a sorting algorithm, aligning well with theoretical expectations:

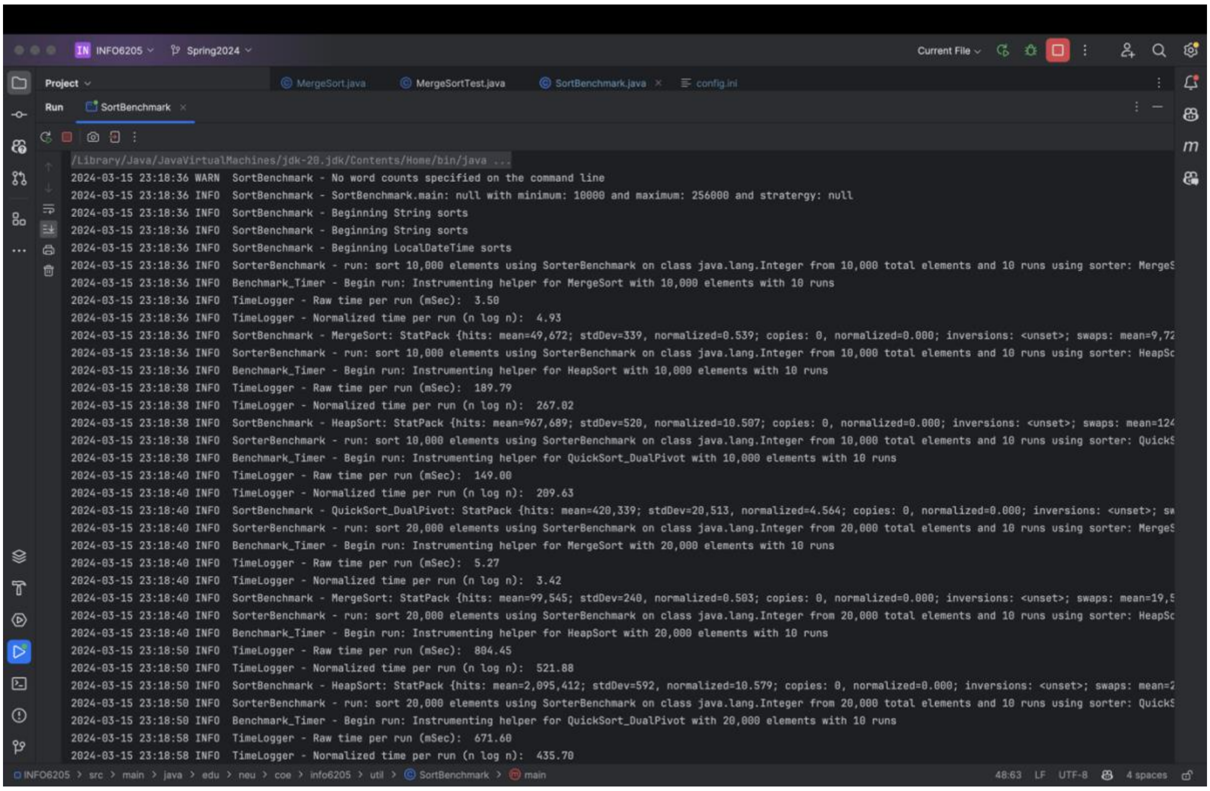
- **MergeSort's Scalability:** MergeSort demonstrates the strongest scalability with increasing input size. Its normalized time consistently decreases as N grows larger, making it the clear choice for handling very large datasets where efficiency is paramount. The number of comparisons shows a very close correlation to execution time.

- **HeapSort's Consistency:** HeapSort maintains relatively consistent performance across dataset sizes, offering decent performance for various use cases. As with MergeSort, the number of comparisons exhibits a strong correlation with execution time.

- **QuickSort_DualPivot's Niche:** QuickSort_DualPivot excels with smaller input sizes but suffers from performance degradation as input size increases. The number of comparisons appears correlated with execution time, though the correlation may be slightly weaker than in the other algorithms.

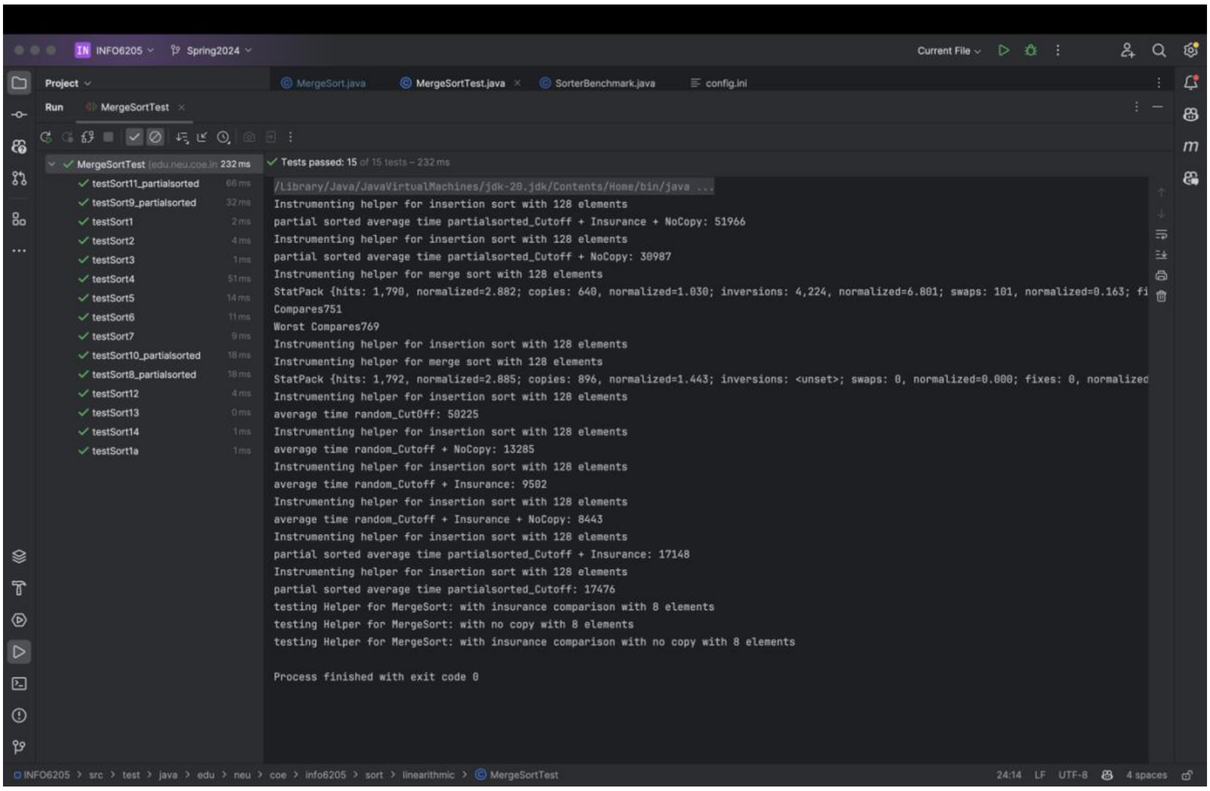
- **Comparisons as Key Predictor:** Across all three sorting algorithms tested, the number of comparisons serves as the most reliable predictor of overall execution time. This strong correlation underscores the fundamental importance of comparisons in the sorting process.

Console Output:



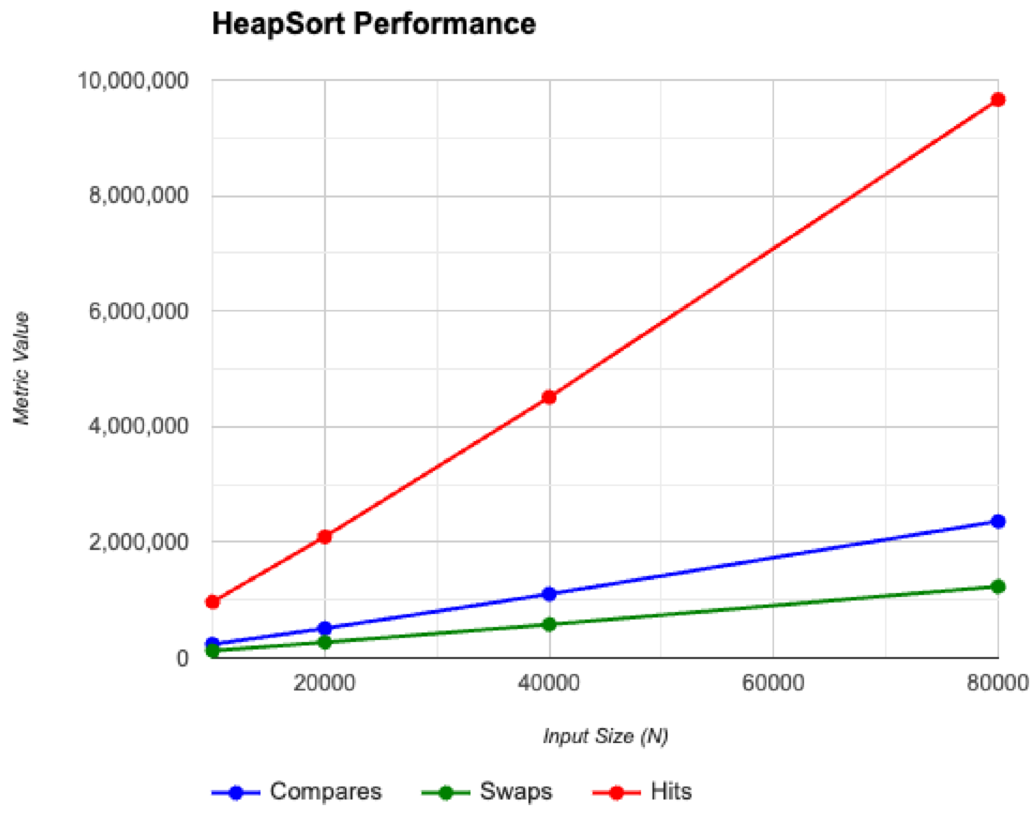
Unit Test Benchmark:

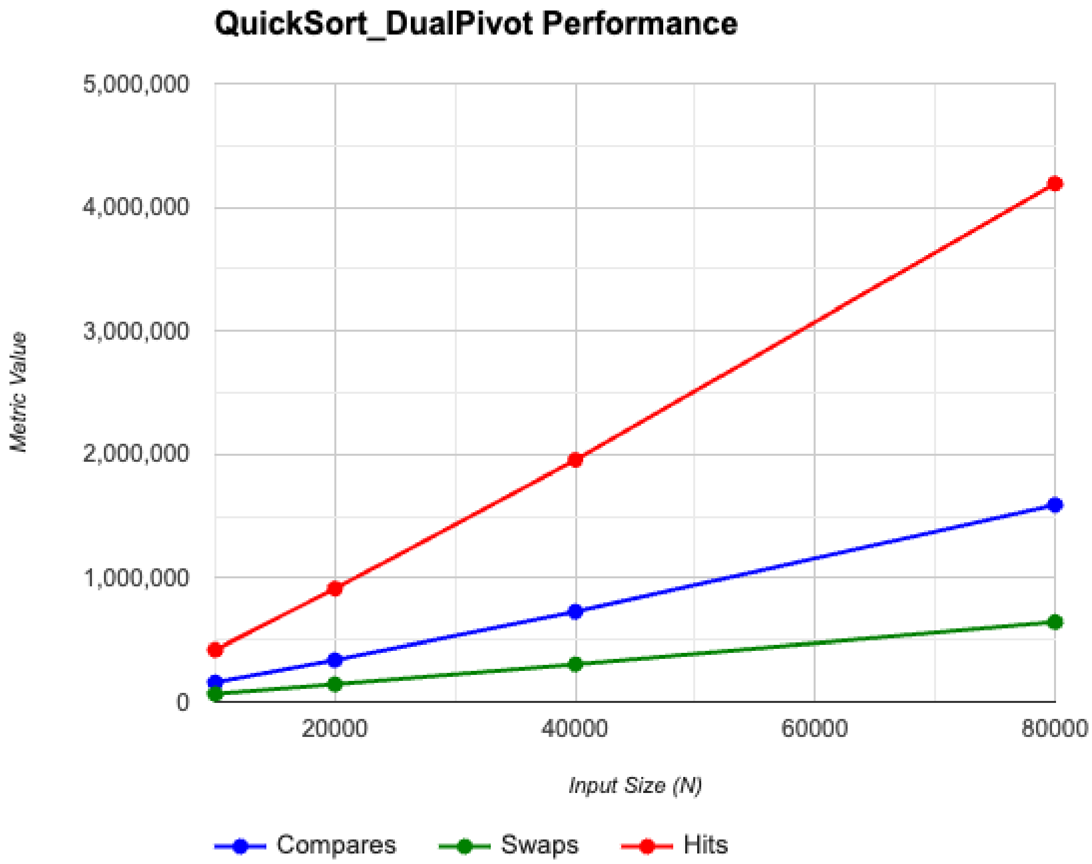
Merge Sort

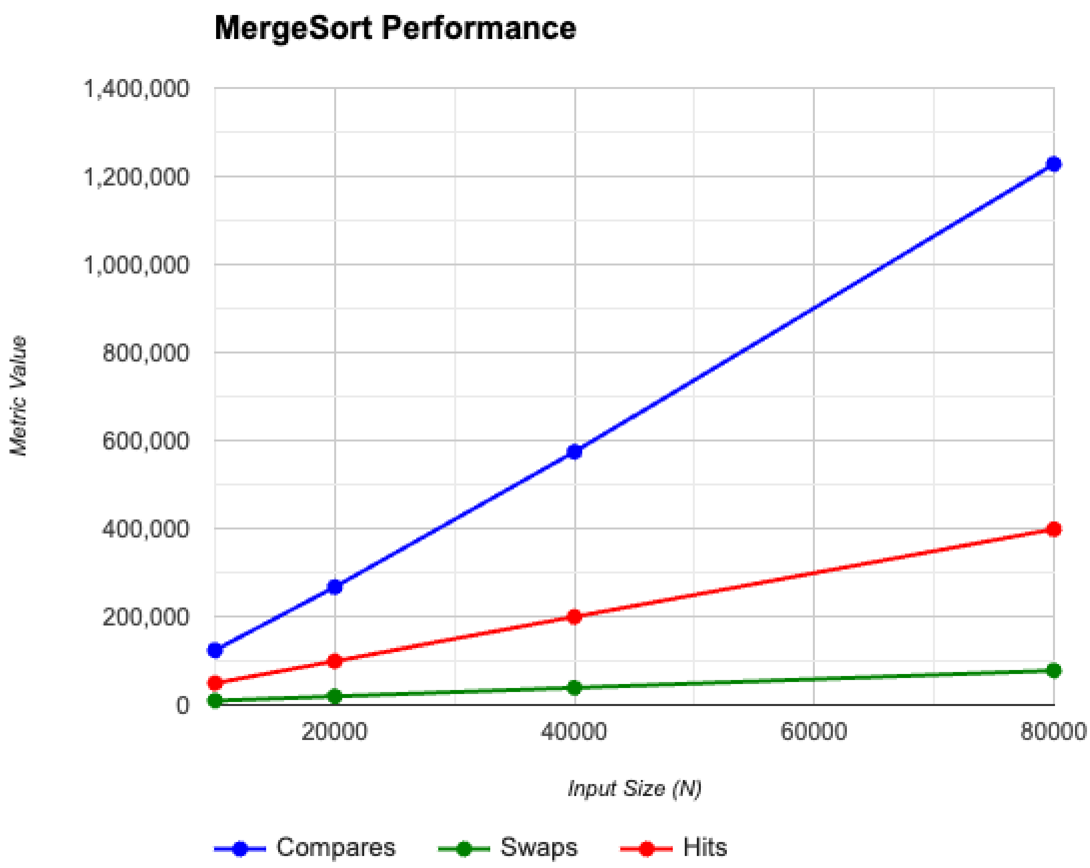


Sort Test Benchmarks:

Algorithm	N	Hits	Swaps	Compares	Raw Time	Normalized Time
MergeSort	10000	49672	9725	123515	3.5	4.93
MergeSort	20000	99545	19515	267027	5.27	3.42
MergeSort	40000	199301	39109	574086	7.93	2.39
MergeSort	80000	398371	78108	1228201	13.14	1.84
HeapSort	10000	967689	124238	235368	189.79	267.02
HeapSort	20000	2095412	268465	510776	804.45	521.88
HeapSort	40000	4510529	576832	1101600	3309.92	996.11
HeapSort	80000	9660363	1233582	2363017	13505.55	1895.34
QuickSort_DualPivot	10000	420339	64768	158540	149	209.63
QuickSort_DualPivot	20000	914908	142790	338277	671.6	435.7
QuickSort_DualPivot	40000	1954128	303745	728061	2664.81	801.96
QuickSort_DualPivot	80000	4192129	644221	1593193	8215.46	1152.94




QuickSort_DualPivot Performance
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
Metric Value
20000
40000
60000
80000
Input Size (N)
Compares
Swaps
Hits



MergeSort Performance
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Metric Value
20000
40000
60000
80000
Input Size (N)
Compares
Swaps
Hits