Program Structures and Algorithms
Spring 2023 (SEC –3)
Assignment-6: Hits as Time Predictor.

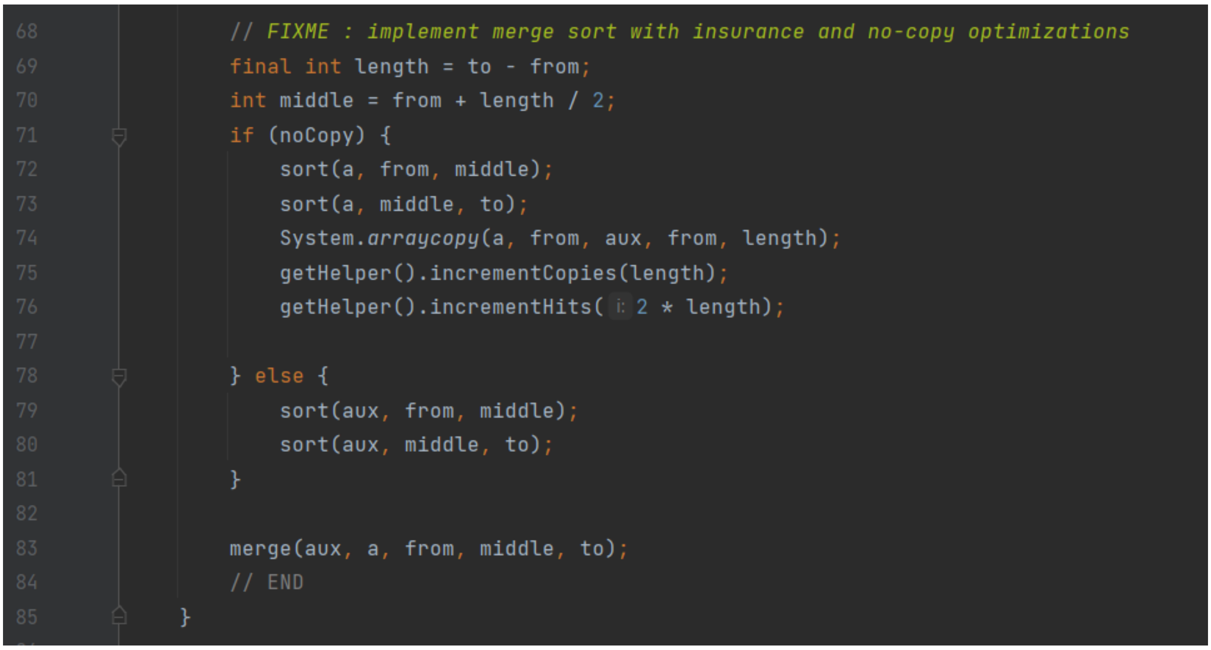
NAME: Venkatesha Matam
NUID: 002740702

Task:

- To find the best predictor of total execution time for sorting algorithms by sorting randomly generated arrays of size between 10,000 and 256,000 elements – doubling the size each time.

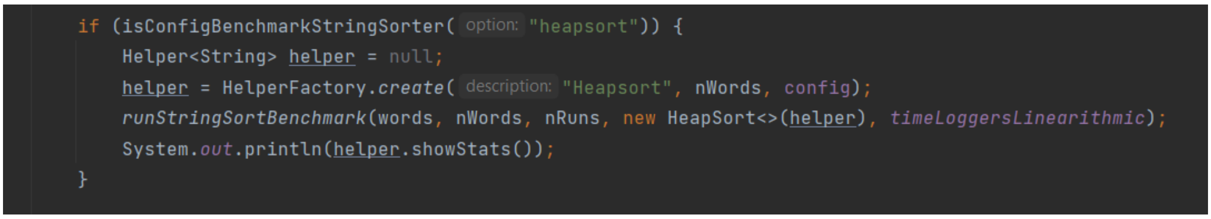
Code Change Snapshots:

1) **sort() method in MergeSort.java –**



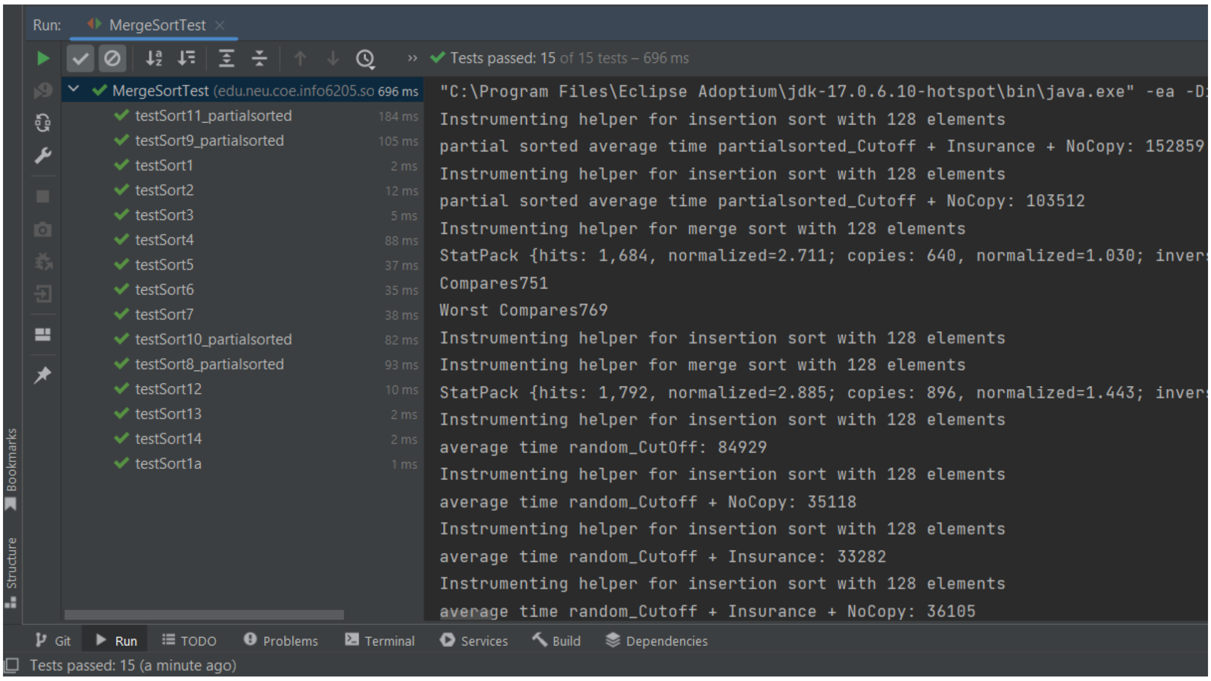
```
        // FIXME : implement merge sort with insurance and no-copy optimizations
        final int length = to - from;
        int middle = from + length / 2;
        if (noCopy) {
            sort(a, from, middle);
            sort(a, middle, to);
            System.arraycopy(a, from, aux, from, length);
            getHelper().incrementCopies(length);
            getHelper().incrementHits(2 * length);

        } else {
            sort(aux, from, middle);
            sort(aux, middle, to);
        }

        merge(aux, a, from, middle, to);
        // END
    }
```

2) **SortBenchMark.java file**



```
if (isConfigBenchmarkStringSorter("heapsort")) {
    Helper<String> helper = null;
    helper = HelperFactory.create("Heapsort", nWords, config);
    runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper), timeLoggersLinearithmic);
    System.out.println(helper.showStats());
}
```

Testcases Snapshot:



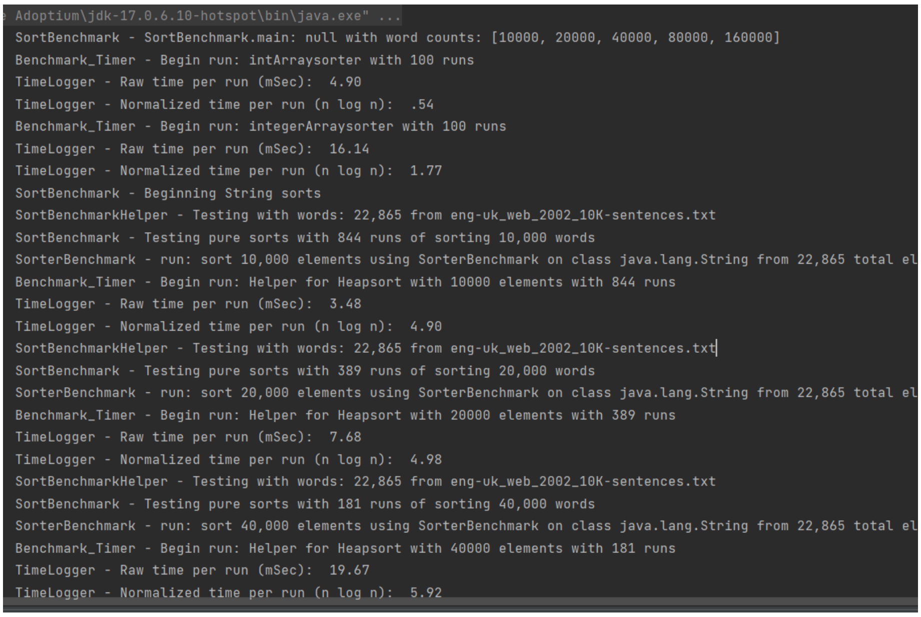
✓ Tests passed: 15 of 15 tests – 696 ms

```
✓ MergeSortTest (edu.neu.coe.info6205.so 696 ms
    ✓ testSort11_partialsorted    184 ms
    ✓ testSort9_partialsorted     105 ms
    ✓ testSort1                     2 ms
    ✓ testSort2                    12 ms
    ✓ testSort3                     5 ms
    ✓ testSort4                    88 ms
    ✓ testSort5                    37 ms
    ✓ testSort6                    35 ms
    ✓ testSort7                    38 ms
    ✓ testSort10_partialsorted     82 ms
    ✓ testSort8_partialsorted      93 ms
    ✓ testSort12                   10 ms
    ✓ testSort13                    2 ms
    ✓ testSort14                    2 ms
    ✓ testSort1a                    1 ms
```

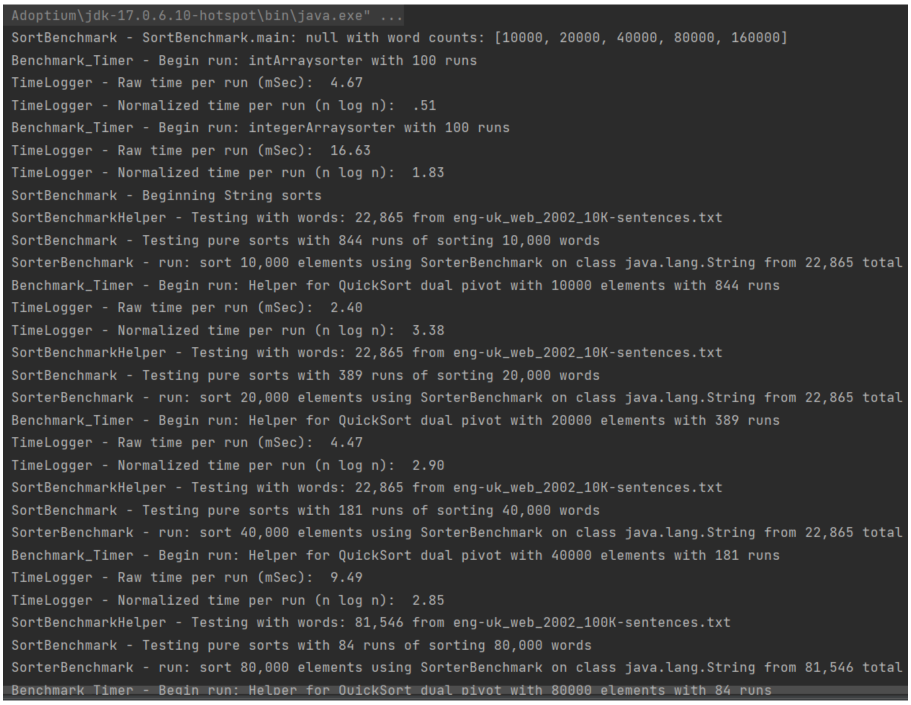
```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" -ea -D:
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance + NoCopy: 152859
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + NoCopy: 103512
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,684, normalized=2.711; copies: 640, normalized=1.030; invers
Compares751
Worst Compares769
Instrumenting helper for insertion sort with 128 elements
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,792, normalized=2.885; copies: 896, normalized=1.443; invers
Instrumenting helper for insertion sort with 128 elements
average time random_CutOff: 84929
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + NoCopy: 35118
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance: 33282
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance + NoCopy: 36105
```

Tests passed: 15 (a minute ago)

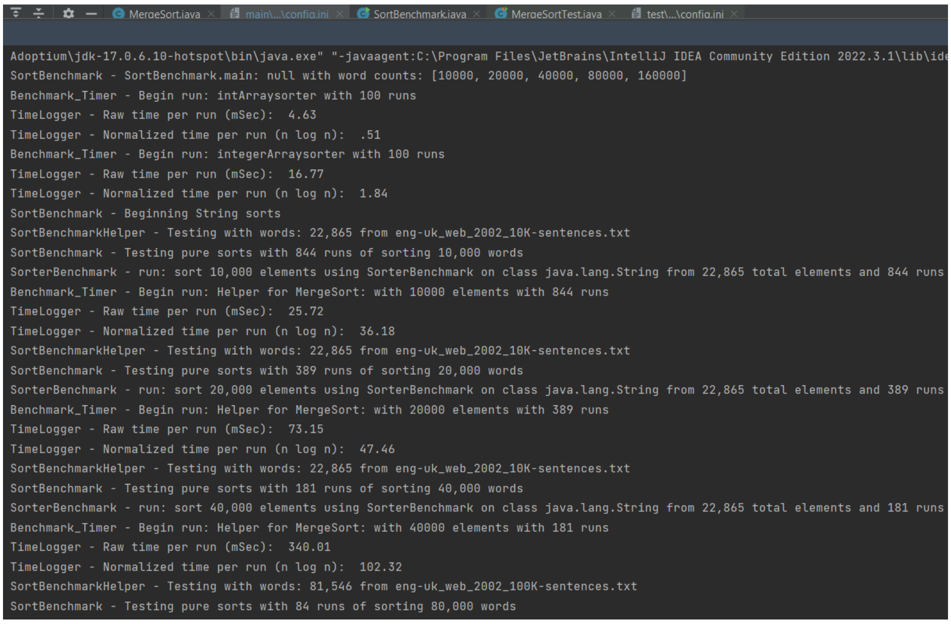
Output Snapshots:



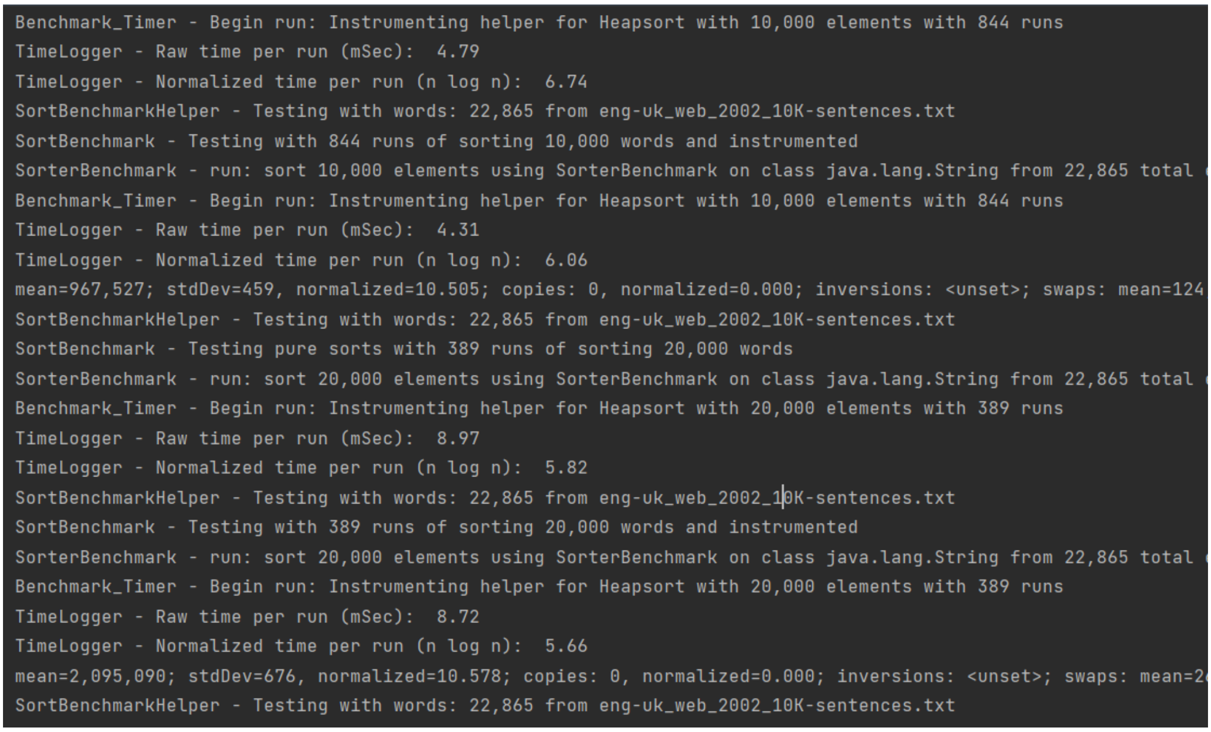
```
e Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" ...
SortBenchmark - SortBenchmark.main: null with word counts: [10000, 20000, 40000, 80000, 160000]
Benchmark_Timer - Begin run: intArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  4.90
TimeLogger - Normalized time per run (n log n):  .54
Benchmark_Timer - Begin run: integerArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  16.14
TimeLogger - Normalized time per run (n log n):  1.77
SortBenchmark - Beginning String sorts
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total el
Benchmark_Timer - Begin run: Helper for Heapsort with 10000 elements with 844 runs
TimeLogger - Raw time per run (mSec):  3.48
TimeLogger - Normalized time per run (n log n):  4.90
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total el
Benchmark_Timer - Begin run: Helper for Heapsort with 20000 elements with 389 runs
TimeLogger - Raw time per run (mSec):  7.68
TimeLogger - Normalized time per run (n log n):  4.98
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 181 runs of sorting 40,000 words
SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total el
Benchmark_Timer - Begin run: Helper for Heapsort with 40000 elements with 181 runs
TimeLogger - Raw time per run (mSec):  19.67
TimeLogger - Normalized time per run (n log n):  5.92
```



```
Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" ...
SortBenchmark - SortBenchmark.main: null with word counts: [10000, 20000, 40000, 80000, 160000]
Benchmark_Timer - Begin run: intArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  4.67
TimeLogger - Normalized time per run (n log n):  .51
Benchmark_Timer - Begin run: integerArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  16.63
TimeLogger - Normalized time per run (n log n):  1.83
SortBenchmark - Beginning String sorts
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Helper for QuickSort dual pivot with 10000 elements with 844 runs
TimeLogger - Raw time per run (mSec):  2.40
TimeLogger - Normalized time per run (n log n):  3.38
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Helper for QuickSort dual pivot with 20000 elements with 389 runs
TimeLogger - Raw time per run (mSec):  4.47
TimeLogger - Normalized time per run (n log n):  2.90
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 181 runs of sorting 40,000 words
SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Helper for QuickSort dual pivot with 40000 elements with 181 runs
TimeLogger - Raw time per run (mSec):  9.49
TimeLogger - Normalized time per run (n log n):  2.85
SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
SortBenchmark - Testing pure sorts with 84 runs of sorting 80,000 words
SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total
Benchmark_Timer - Begin run: Helper for QuickSort dual pivot with 80000 elements with 84 runs
```



```
Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.1\lib\ide
SortBenchmark - SortBenchmark.main: null with word counts: [10000, 20000, 40000, 80000, 160000]
Benchmark_Timer - Begin run: intArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  4.63
TimeLogger - Normalized time per run (n log n):  .51
Benchmark_Timer - Begin run: integerArraysorter with 100 runs
TimeLogger - Raw time per run (mSec):  16.77
TimeLogger - Normalized time per run (n log n):  1.84
SortBenchmark - Beginning String sorts
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs
Benchmark_Timer - Begin run: Helper for MergeSort: with 10000 elements with 844 runs
TimeLogger - Raw time per run (mSec):  25.72
TimeLogger - Normalized time per run (n log n):  36.18
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs
Benchmark_Timer - Begin run: Helper for MergeSort: with 20000 elements with 389 runs
TimeLogger - Raw time per run (mSec):  73.15
TimeLogger - Normalized time per run (n log n):  47.46
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 181 runs of sorting 40,000 words
SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs
Benchmark_Timer - Begin run: Helper for MergeSort: with 40000 elements with 181 runs
TimeLogger - Raw time per run (mSec):  340.01
TimeLogger - Normalized time per run (n log n):  102.32
SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
SortBenchmark - Testing pure sorts with 84 runs of sorting 80,000 words
```

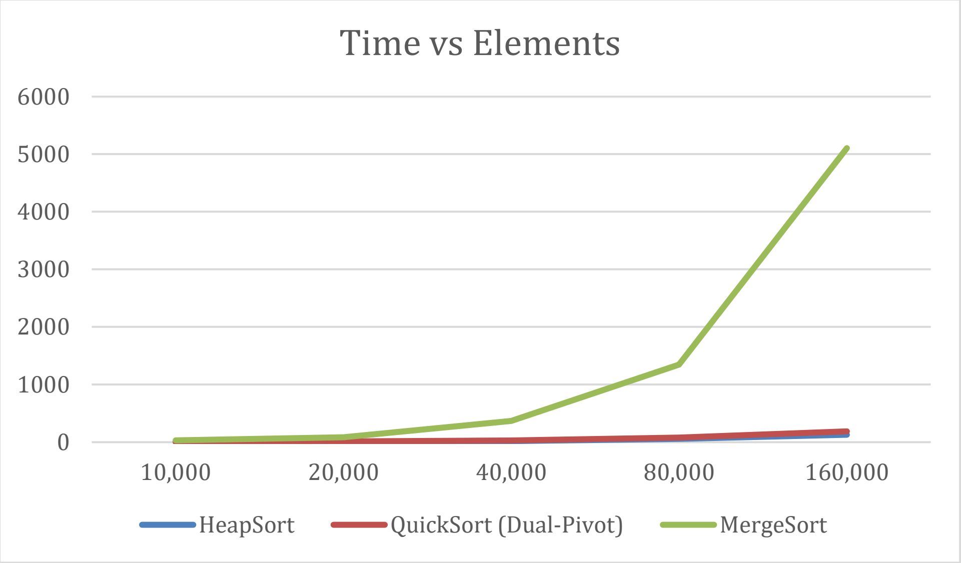


```
Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
TimeLogger - Raw time per run (mSec):  4.79
TimeLogger - Normalized time per run (n log n):  6.74
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing with 844 runs of sorting 10,000 words and instrumented
SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
TimeLogger - Raw time per run (mSec):  4.31
TimeLogger - Normalized time per run (n log n):  6.06
mean=967,527; stdDev=459, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
TimeLogger - Raw time per run (mSec):  8.97
TimeLogger - Normalized time per run (n log n):  5.82
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
SortBenchmark - Testing with 389 runs of sorting 20,000 words and instrumented
SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total
Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
TimeLogger - Raw time per run (mSec):  8.72
TimeLogger - Normalized time per run (n log n):  5.66
mean=2,095,090; stdDev=676, normalized=10.578; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=2
SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
```

Observations:

1) Time (Without Instrumentation)

Number of Elements	Raw Time per Run (ms)	Raw Time per Run (ms)	Raw Time per Run (ms)
	HeapSort	**QuickSort (Dual-Pivot)**	**MergeSort**
10,000	3.48	2.4	25.72
20,000	7.68	4.47	73.15
40,000	19.67	9.49	340.01
80,000	55.49	26.93	1261.51
160,000	129.38	58.46	4917.52

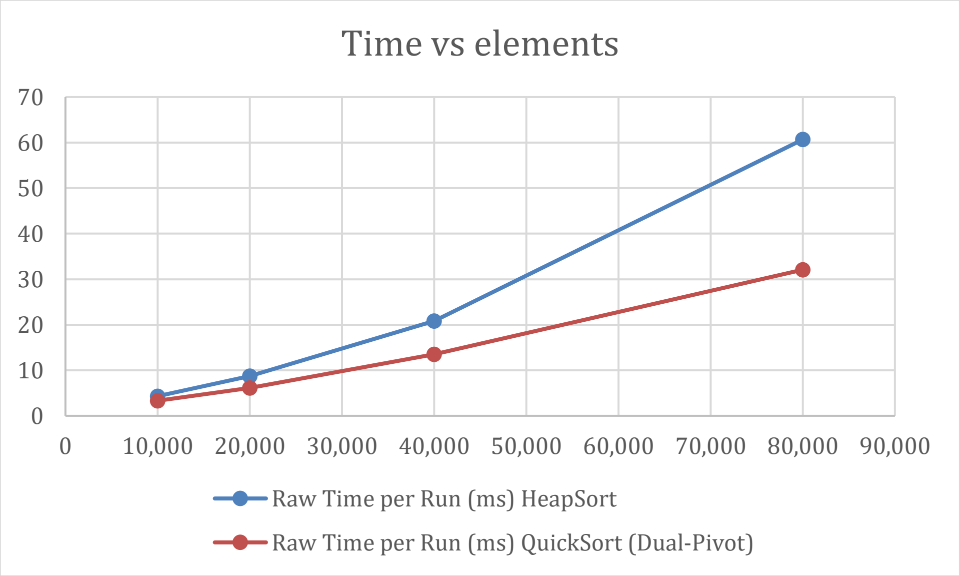


This is a plot of the raw times of three sorts, and it shows that the trend for this input merge sort is significantly increasing.

2) With Instrumentation:

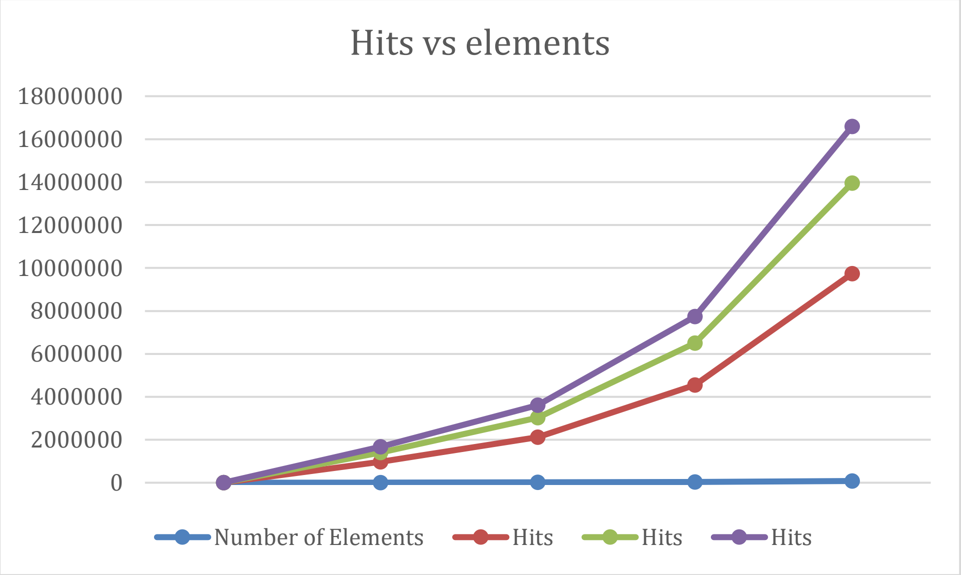
Time –

Number of Elements	Raw Time per Run (ms)	Raw Time per Run (ms)
	HeapSort	**QuickSort (Dual-Pivot)**
10,000	4.31	3.31
20,000	8.72	6.12
40,000	20.85	13.51
80,000	60.67	32.08



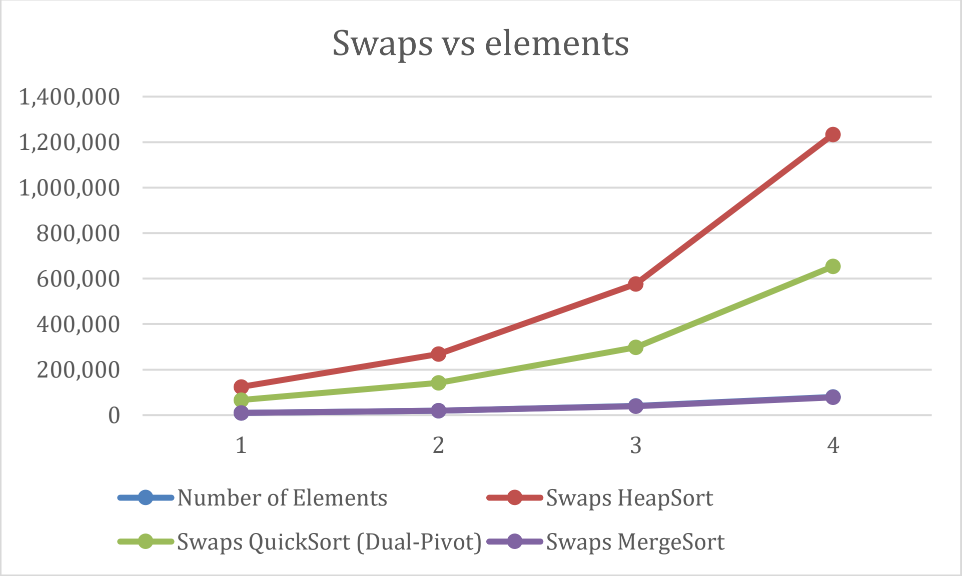
Hits –

Number of Elements	Hits	Hits	Hits
	HeapSort	**QuickSort (Dual-Pivot)**	**MergeSort**
10,000	967,527	424,020	269,812
20,000	2,095,090	911,702	579,571
40,000	4,510,212	1,947,170	1,239,132
80,000	9,660,492	4,217,408	2,638,185



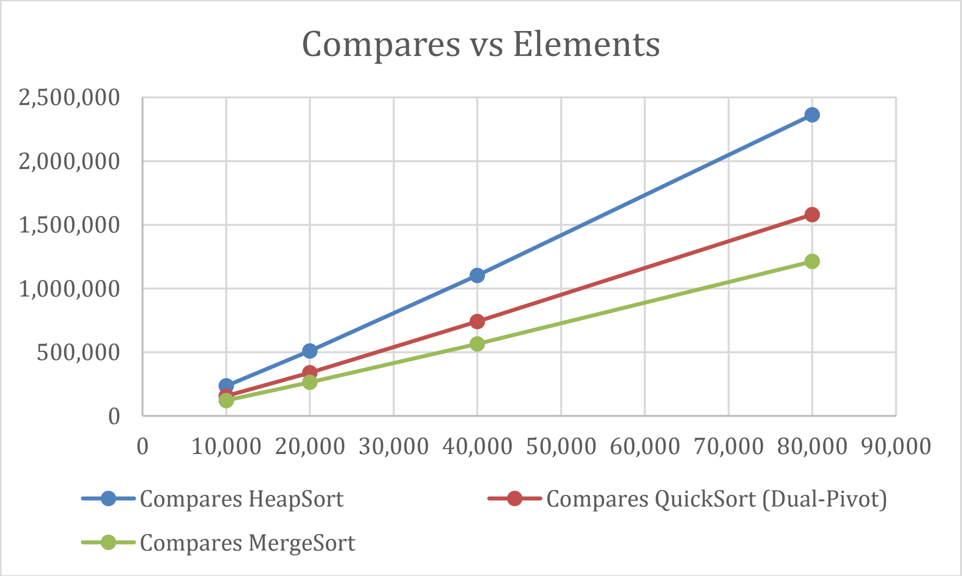
Swaps –

Number of Elements	Swaps	Swaps	Swaps
	HeapSort	**QuickSort (Dual-Pivot)**	**MergeSort**
10,000	124,198	66,528	9,780
20,000	268,402	141,684	19,525
40,000	576,805	297,634	39,044
80,000	1,233,627	653,741	78,061



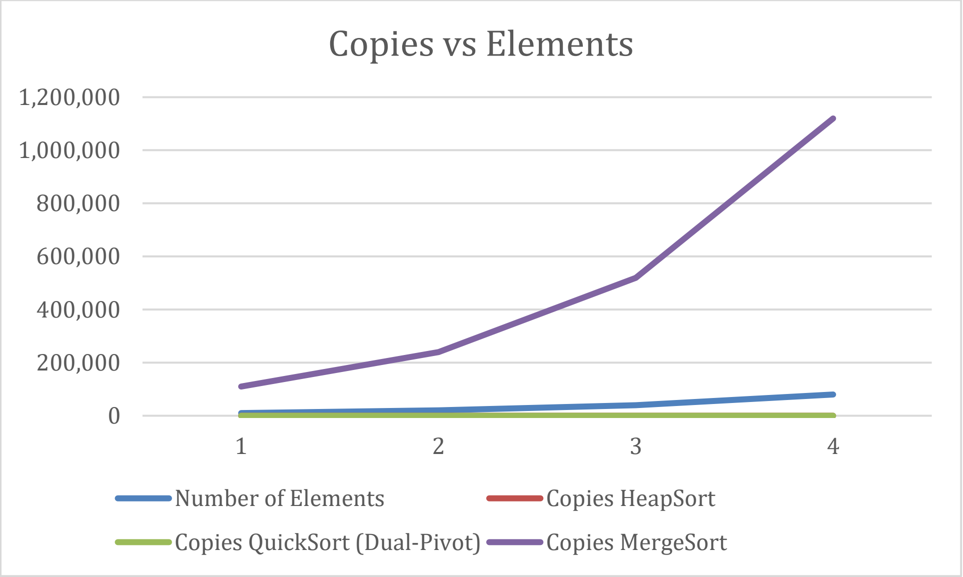
Compares –

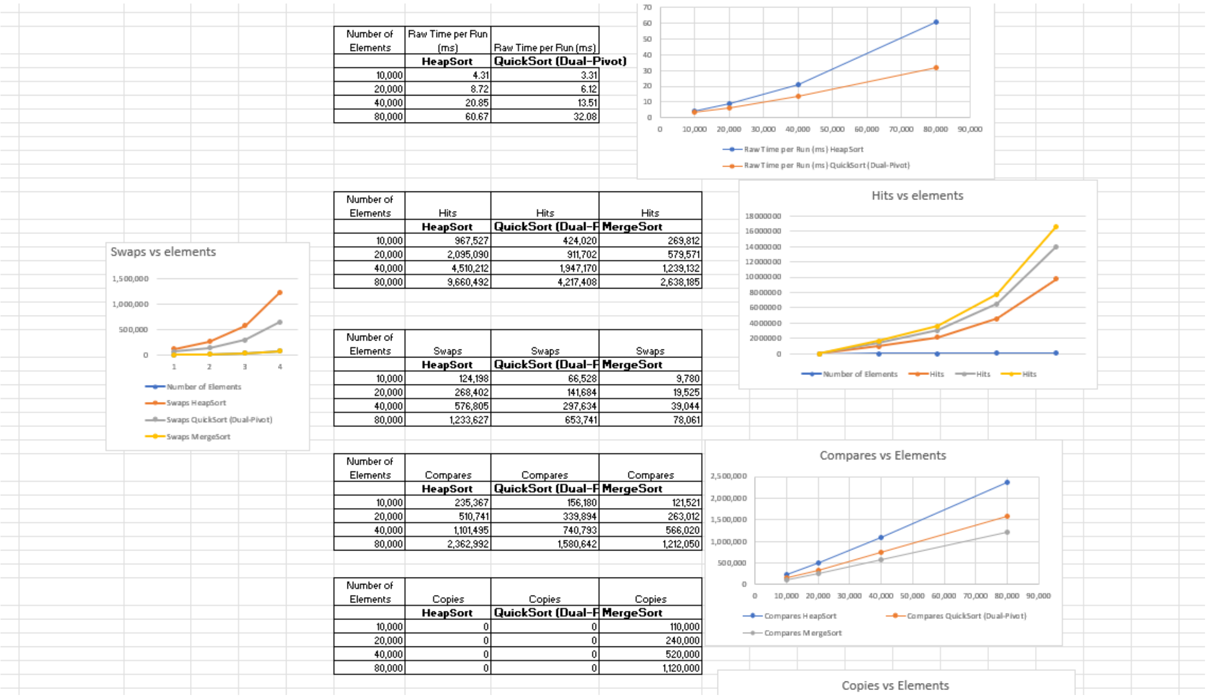
Number of Elements	Compares	Compares	Compares
	HeapSort	**QuickSort (Dual-Pivot)**	**MergeSort**
10,000	235,367	156,180	121,521
20,000	510,741	339,894	263,012
40,000	1,101,495	740,793	566,020
80,000	2,362,992	1,580,642	1,212,050



Copies –

Number of Elements	Copies	Copies	Copies
	HeapSort	**QuickSort (Dual-Pivot)**	**MergeSort**
10,000	0	0	110,000
20,000	0	0	240,000
40,000	0	0	520,000
80,000	0	0	1,120,000





Number of Elements	Raw Time per Run (ms)	Raw Time per Run (ms)
	HeapSort	**QuickSort (Dual-Pivot)**
10,000	4.31	3.31
20,000	8.72	6.12
40,000	20.85	13.51
80,000	60.67	32.08

Number of Elements	Hits	Hits	Hits
	HeapSort	**QuickSort (Dual-F**	**MergeSort**
10,000	967,527	424,020	269,812
20,000	2,095,090	911,702	579,571
40,000	4,510,212	1,947,170	1,239,132
80,000	9,660,492	4,217,408	2,638,185

Number of Elements	Swaps	Swaps	Swaps
	HeapSort	**QuickSort (Dual-F**	**MergeSort**
10,000	124,198	66,528	9,780
20,000	268,402	141,684	19,525
40,000	576,805	297,634	39,044
80,000	1,233,627	653,741	78,061

Number of Elements	Compares	Compares	Compares
	HeapSort	**QuickSort (Dual-F**	**MergeSort**
10,000	235,367	156,180	121,521
20,000	510,741	339,894	263,012
40,000	1,101,495	740,793	566,020
80,000	2,362,992	1,580,642	1,212,050

Number of Elements	Copies	Copies	Copies
	HeapSort	**QuickSort (Dual-F**	**MergeSort**
10,000	0	0	110,000
20,000	0	0	240,000
40,000	0	0	520,000
80,000	0	0	1,120,000

Conclusion:

- Sorting algorithms' performance can be evaluated based on operations like compares, copies, and swaps that involve array hits.
- The number of hits can serve as a neutral way to compare the algorithms, with a larger number indicating poorer performance.
- If the operations have different durations, the one that takes less time and involves fewer parameters would be a better predictor of the algorithm's completion time.
- Swaps tend to be more costly than copies, making copy a less expensive operation.
- However, comparisons could be more expensive depending on the hardware, making it challenging to compare copies and comparisons.
- The algorithm with the most swaps has the worst performance, followed by copy and comparison.
- If no metrics are available, a general number of hits can be used, with the highest number indicating the worst performance.

Based on observations, merge sort has better performance than quicksort and heapsort.