Akash Bharadwaj Karthik (002787011)

Program Structures & Algorithms
Spring 2023 (Sec -8)
Assignment No. 6

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.

1. 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.

2. 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.

3. You must support your (clearly stated) conclusions with evidence from the benchmarks (you should provide log/log charts and spreadsheets typically).

Relationship Conclusion:

Swaps are the best predictor for execution time for Merge Sort:
Highest R^2 value observed in swaps for Merge sort both with and without instrumentation;
Highest value of 0.9486 observed with instrumentation and 0.9351 without instrumentation

Hits and swaps are good predictors for execution time of **Heap sort with instrumentation**, and **compares** are good predictors when **not using instrumentation.**
Highest R^2 value of **0.975 observed with instrumentation** and **0.9888 without instrumentation**

Hits and Swaps are good predictors for execution time for **Quick Sort with instrumentation, and swaps and compares** are good predictors **without instrumentation.**
Highest value of **0.9772 observed with instrumentation** and **0.9716 without instrumentation**

Metrics used for benchmarking:

Array sizes: 10,000 to 160,000 using doubling method.
Sort algorithms: **QuickSortDualPivot** , **BasicMergeSort** and **HeapSort**

Evidence used to support conclusion:

BasicMergeSort Table:

N	lg(compares)	lg(Swaps)	lg(Hits)	lg(Copies)	lg(Time w/o instrumenting)	lg(Time w instrumenting)
10000	16.89065675	13.251491	18.9017	17.747144	2.570462931	3.042644337
20000	18.00466306	14.251219	20.0149	18.87267488	4.063502942	3.584962501
40000	19.1104969	15.252797	21.1201	19.9881521	4.146492307	3.937344392
80000	20.2090035	16.253436	22.218	21.0950673	4.960233672	5.094658343
160000	21.30122112	17.252595	23.3096	22.19460298	6.372081484	6.238022518

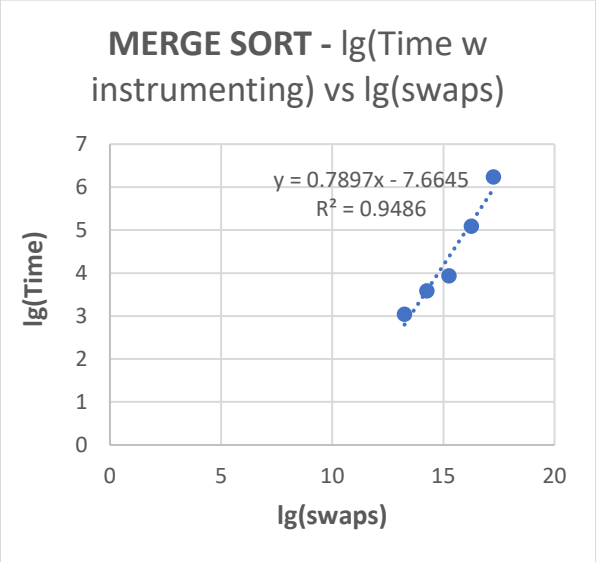
DualPivotQuickSort Table:

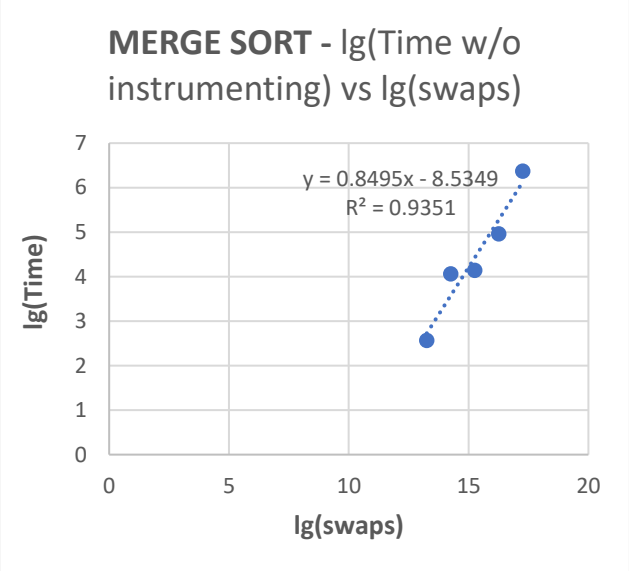
N	lg(compares)	lg(Swaps)	lg(Hits)	lg(Time w/o instrumenting)	lg(Time w instrumenting)
10000	17.25395484	15.987986	18.6767	1.726831217	2.176322773
20000	18.36776615	17.100775	19.7892	2.321928095	2.599317794
40000	19.48965891	18.215648	20.906	3.313245852	3.798050515
80000	20.59370942	19.304238	21.9999	4.523561956	4.925999419
160000	21.69093276	20.414587	23.1047	6.098874287	6.19061486

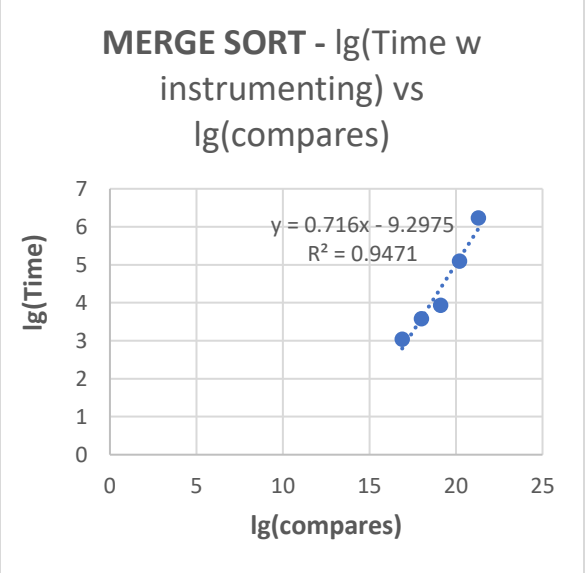
HeapSort Table

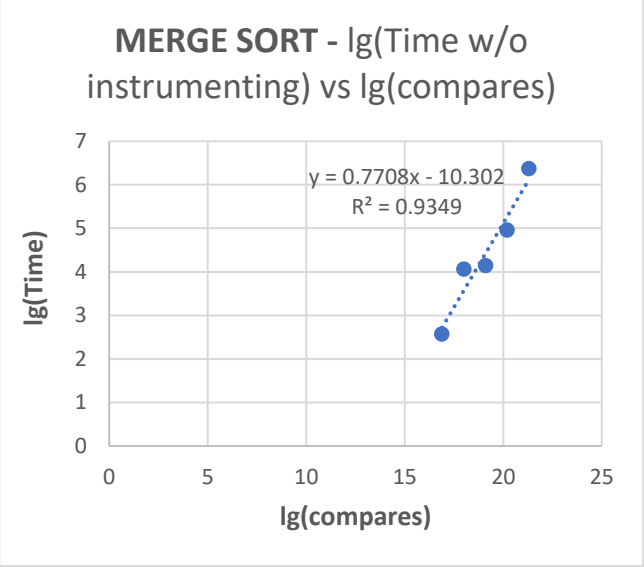
N	lg(compares)	lg(Swaps)	lg(Hits)	lg(Time w/o instrumenting)	lg(Time w instrumenting)
10000	17.84457258	16.922223	19.8839	1.744161096	1.887525271
20000	18.96225681	18.034021	20.9986	2.853995647	3.898208353
40000	20.07103212	19.137749	22.1048	4.193771743	4.544732656
80000	21.17217892	20.234435	23.2036	5.13996057	5.792074462
160000	22.26669099	21.325099	24.2962	6.990387652	7.676027579

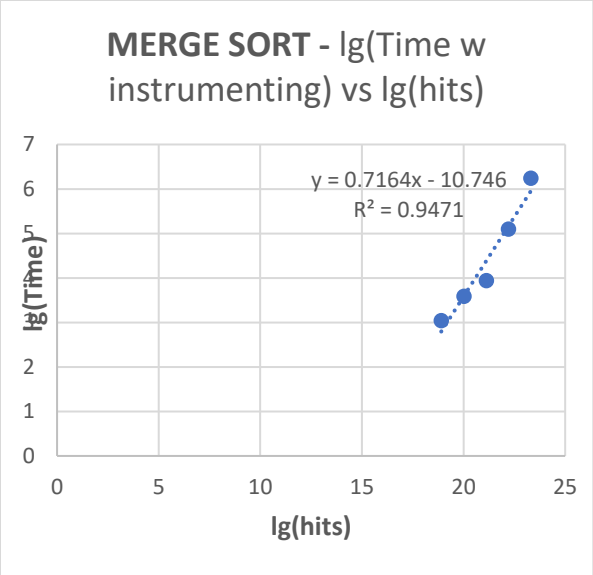
Graphical Representation

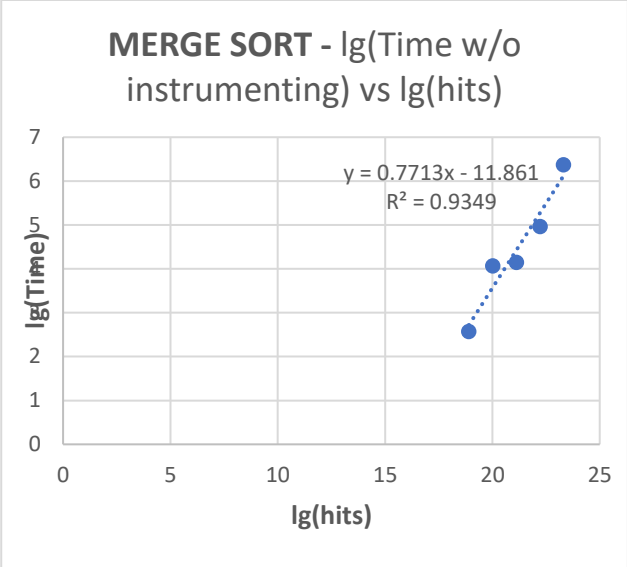


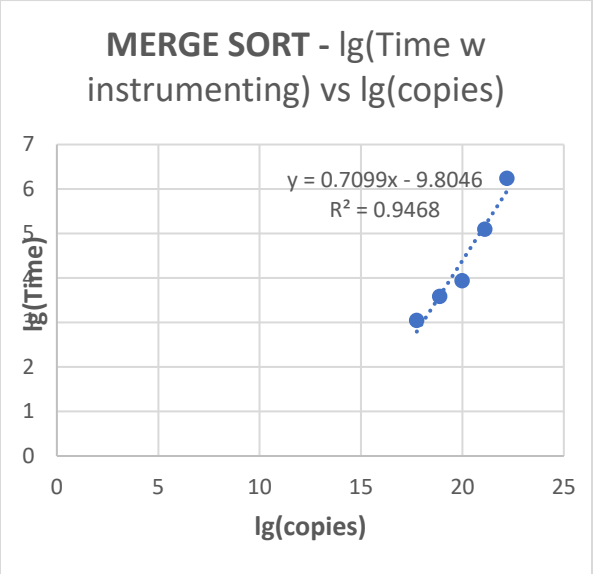


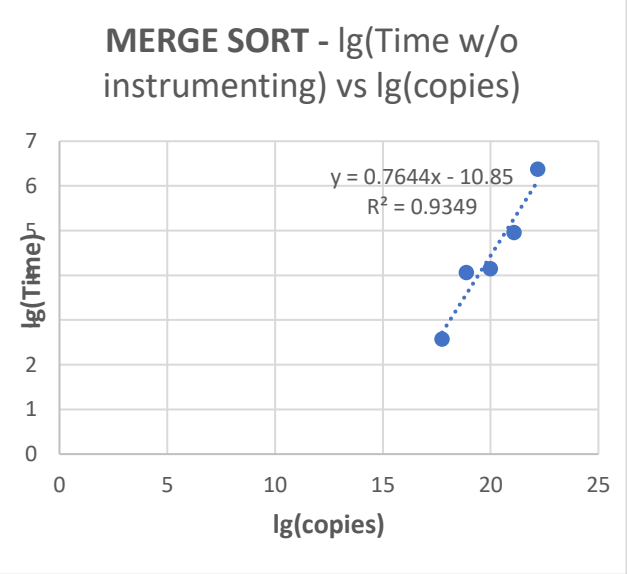




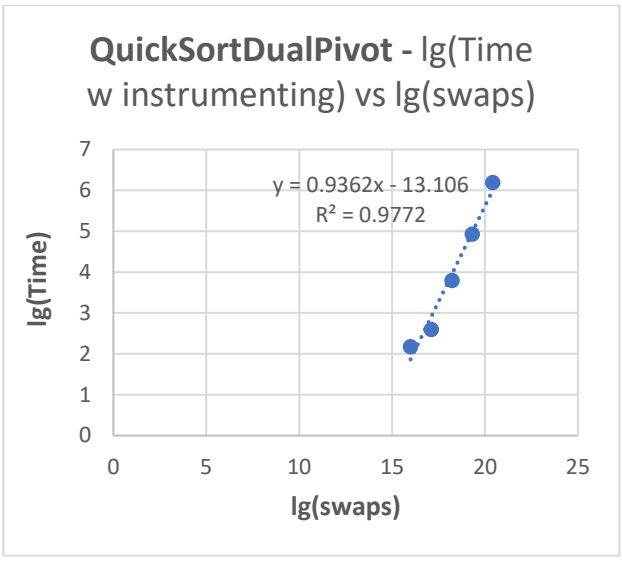


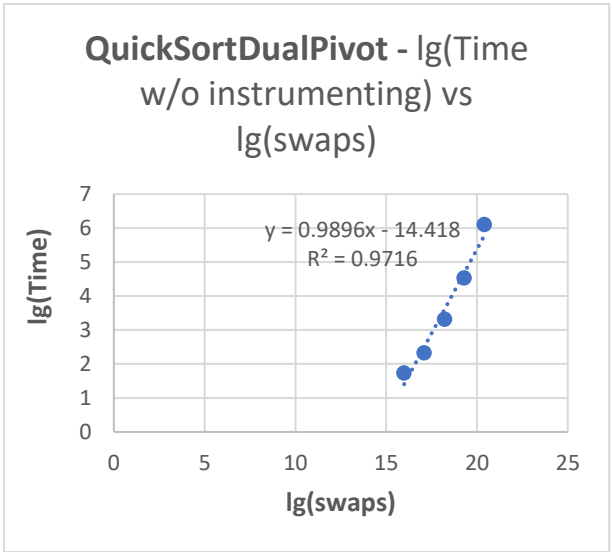


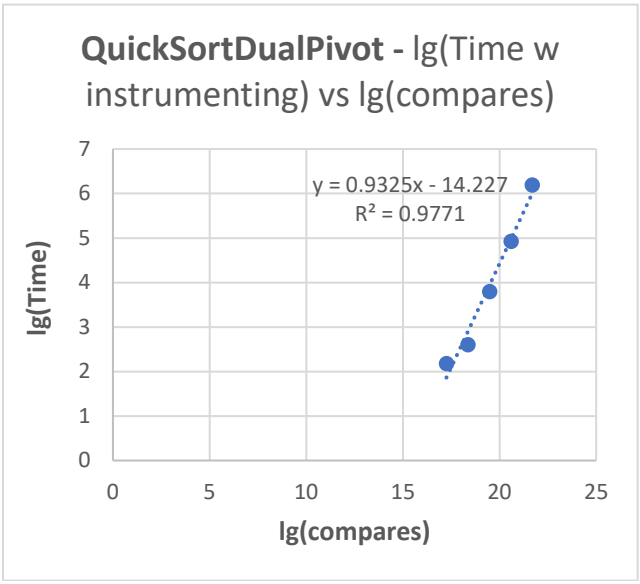


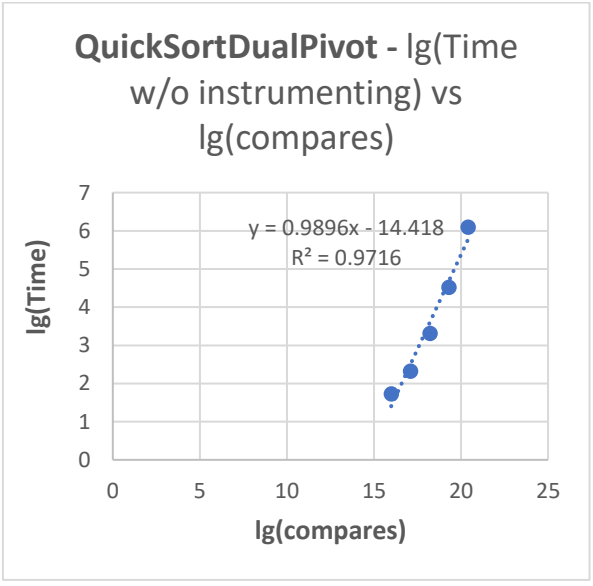


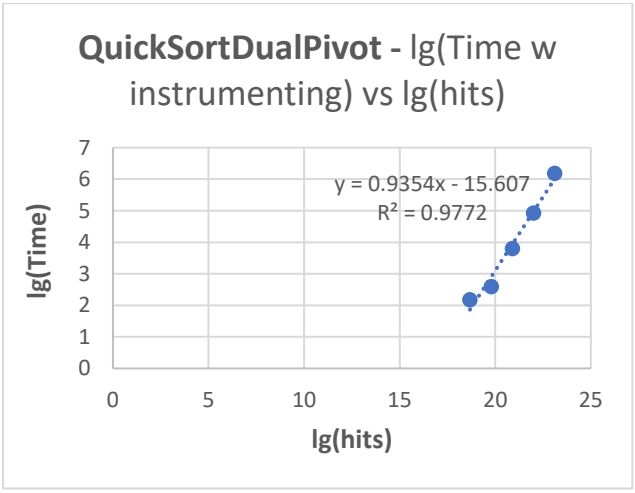
Highest R^2 value observed in **swaps for Merge sort** both with and without instrumentation; Highest value of **0.9486 observed with instrumentation** and **0.9351 without instrumentation**

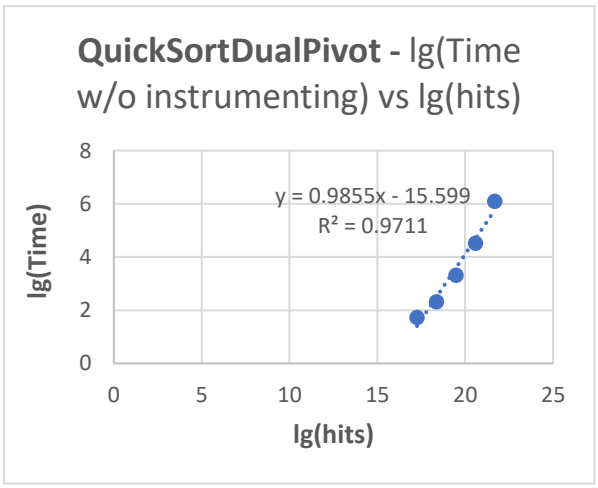

QuickSortDualPivot - lg(Time w instrumenting) vs lg(swaps)
y = 0.9362x - 13.106
R² = 0.9772
lg(Time)
lg(swaps)


QuickSortDualPivot - lg(Time w/o instrumenting) vs lg(swaps)
y = 0.9896x - 14.418
R² = 0.9716
lg(Time)
lg(swaps)


QuickSortDualPivot - lg(Time w instrumenting) vs lg(compares)
y = 0.9325x - 14.227
R² = 0.9771
lg(Time)
lg(compares)

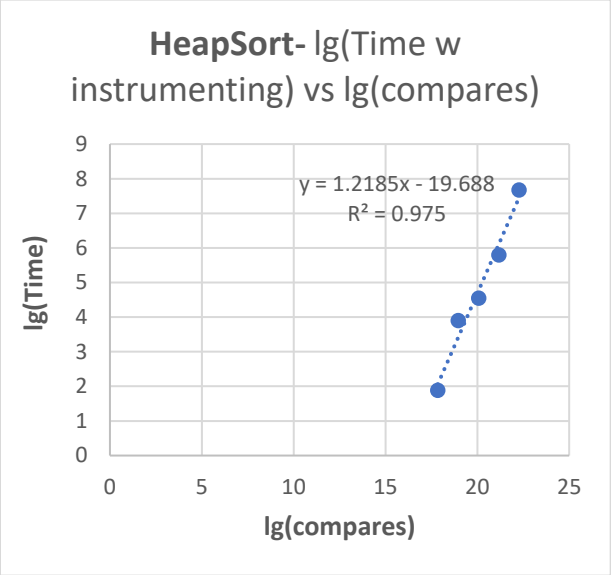

QuickSortDualPivot - lg(Time w/o instrumenting) vs lg(compares)
y = 0.9896x - 14.418
R² = 0.9716
lg(Time)
lg(compares)

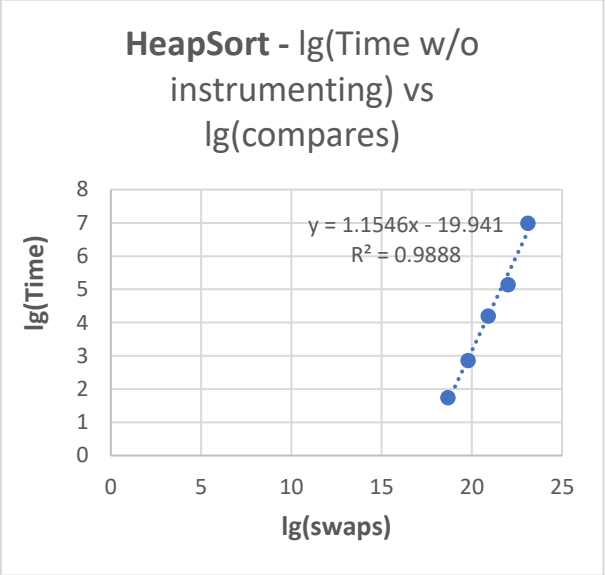

QuickSortDualPivot - lg(Time w instrumenting) vs lg(hits)
y = 0.9354x - 15.607
R² = 0.9772
lg(Time)
lg(hits)

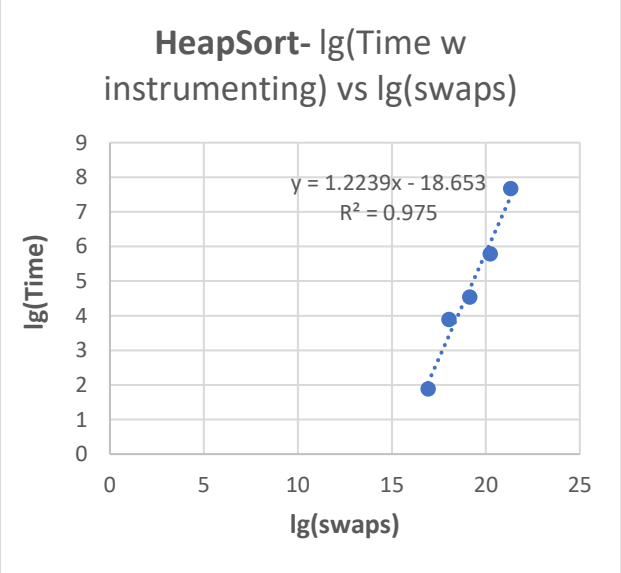

QuickSortDualPivot - lg(Time w/o instrumenting) vs lg(hits)
y = 0.9855x - 15.599
R² = 0.9711
lg(Time)
lg(hits)

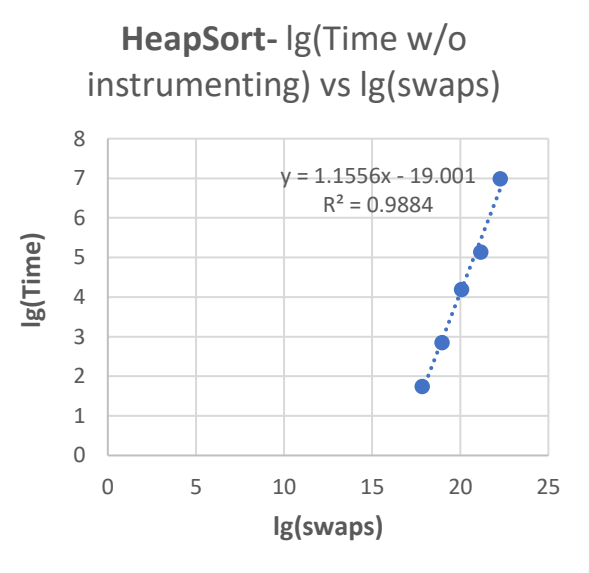
Highest R^2 value observed in **hits/swaps for Quick Sort with instrumentation** and **swaps/compare without instrumentation**;

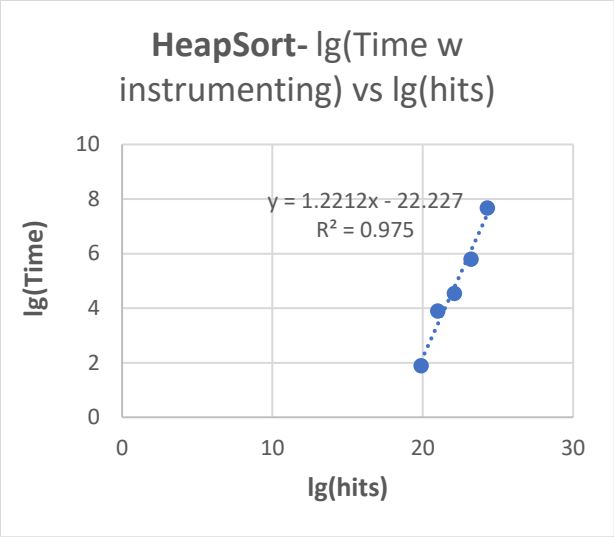
Highest value of **0.9772 observed with instrumentation** and **0.9716 without instrumentation**

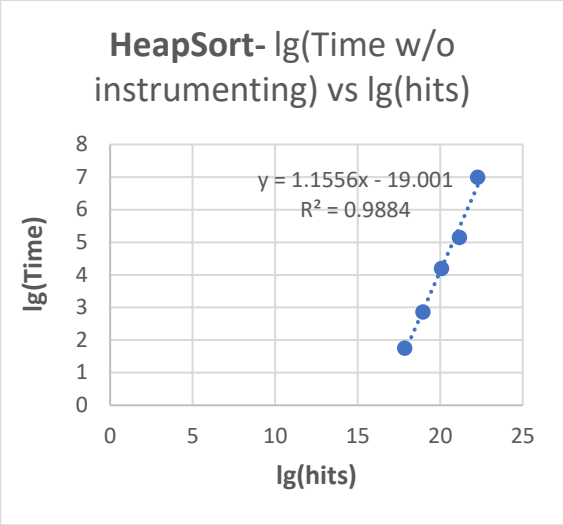








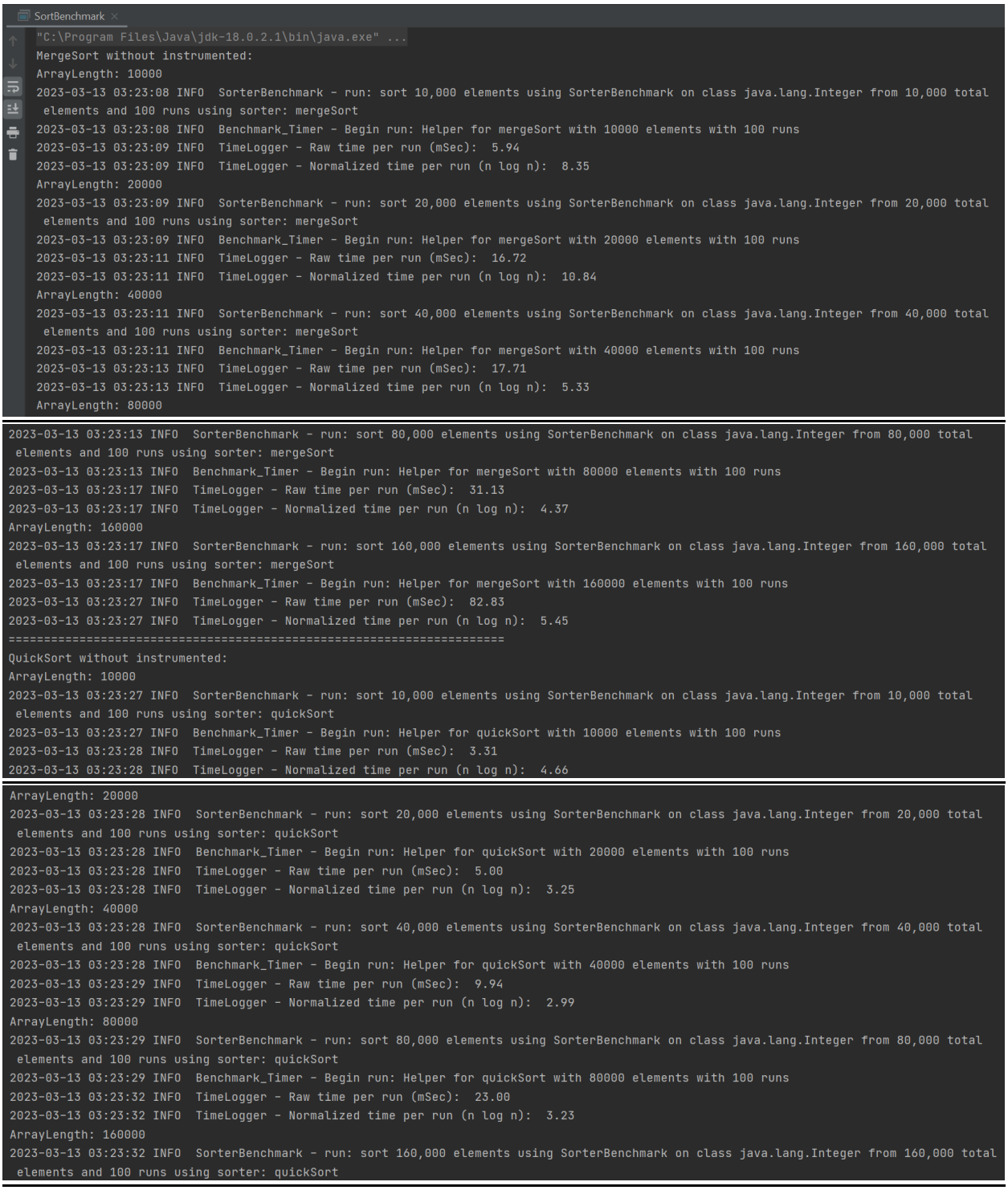




Highest R^2 value observed in **hits**,**swaps and compares equally for Heap Sort with instrumentation** and **compares without instrumentation**;

Highest value of **0.975 observed with instrumentation** and **0.9888 without instrumentation**

Console Output:



```
SortBenchmark
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
MergeSort without instrumented:
ArrayLength: 10000
2023-03-13 03:23:08 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total
 elements and 100 runs using sorter: mergeSort
2023-03-13 03:23:08 INFO  Benchmark_Timer - Begin run: Helper for mergeSort with 10000 elements with 100 runs
2023-03-13 03:23:09 INFO  TimeLogger - Raw time per run (mSec):  5.94
2023-03-13 03:23:09 INFO  TimeLogger - Normalized time per run (n log n):  8.35
ArrayLength: 20000
2023-03-13 03:23:09 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total
 elements and 100 runs using sorter: mergeSort
2023-03-13 03:23:09 INFO  Benchmark_Timer - Begin run: Helper for mergeSort with 20000 elements with 100 runs
2023-03-13 03:23:11 INFO  TimeLogger - Raw time per run (mSec):  16.72
2023-03-13 03:23:11 INFO  TimeLogger - Normalized time per run (n log n):  10.84
ArrayLength: 40000
2023-03-13 03:23:11 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total
 elements and 100 runs using sorter: mergeSort
2023-03-13 03:23:11 INFO  Benchmark_Timer - Begin run: Helper for mergeSort with 40000 elements with 100 runs
2023-03-13 03:23:13 INFO  TimeLogger - Raw time per run (mSec):  17.71
2023-03-13 03:23:13 INFO  TimeLogger - Normalized time per run (n log n):  5.33
ArrayLength: 80000
2023-03-13 03:23:13 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total
 elements and 100 runs using sorter: mergeSort
2023-03-13 03:23:13 INFO  Benchmark_Timer - Begin run: Helper for mergeSort with 80000 elements with 100 runs
2023-03-13 03:23:17 INFO  TimeLogger - Raw time per run (mSec):  31.13
2023-03-13 03:23:17 INFO  TimeLogger - Normalized time per run (n log n):  4.37
ArrayLength: 160000
2023-03-13 03:23:17 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total
 elements and 100 runs using sorter: mergeSort
2023-03-13 03:23:17 INFO  Benchmark_Timer - Begin run: Helper for mergeSort with 160000 elements with 100 runs
2023-03-13 03:23:27 INFO  TimeLogger - Raw time per run (mSec):  82.83
2023-03-13 03:23:27 INFO  TimeLogger - Normalized time per run (n log n):  5.45
================================================================
QuickSort without instrumented:
ArrayLength: 10000
2023-03-13 03:23:27 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total
 elements and 100 runs using sorter: quickSort
2023-03-13 03:23:27 INFO  Benchmark_Timer - Begin run: Helper for quickSort with 10000 elements with 100 runs
2023-03-13 03:23:28 INFO  TimeLogger - Raw time per run (mSec):  3.31
2023-03-13 03:23:28 INFO  TimeLogger - Normalized time per run (n log n):  4.66
ArrayLength: 20000
2023-03-13 03:23:28 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total
 elements and 100 runs using sorter: quickSort
2023-03-13 03:23:28 INFO  Benchmark_Timer - Begin run: Helper for quickSort with 20000 elements with 100 runs
2023-03-13 03:23:28 INFO  TimeLogger - Raw time per run (mSec):  5.00
2023-03-13 03:23:28 INFO  TimeLogger - Normalized time per run (n log n):  3.25
ArrayLength: 40000
2023-03-13 03:23:28 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total
 elements and 100 runs using sorter: quickSort
2023-03-13 03:23:28 INFO  Benchmark_Timer - Begin run: Helper for quickSort with 40000 elements with 100 runs
2023-03-13 03:23:29 INFO  TimeLogger - Raw time per run (mSec):  9.94
2023-03-13 03:23:29 INFO  TimeLogger - Normalized time per run (n log n):  2.99
ArrayLength: 80000
2023-03-13 03:23:29 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total
 elements and 100 runs using sorter: quickSort
2023-03-13 03:23:29 INFO  Benchmark_Timer - Begin run: Helper for quickSort with 80000 elements with 100 runs
2023-03-13 03:23:32 INFO  TimeLogger - Raw time per run (mSec):  23.00
2023-03-13 03:23:32 INFO  TimeLogger - Normalized time per run (n log n):  3.23
ArrayLength: 160000
2023-03-13 03:23:32 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total
 elements and 100 runs using sorter: quickSort
```

```
2023-03-13 03:23:32 INFO  Benchmark_Timer - Begin run: Helper for quickSort with 160000 elements with 100 runs
2023-03-13 03:23:41 INFO  TimeLogger - Raw time per run (mSec):  68.54
2023-03-13 03:23:41 INFO  TimeLogger - Normalized time per run (n log n):  4.51
================================================================
HeapSort without instrumented:
ArrayLength: 10000
2023-03-13 03:23:41 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total
 elements and 100 runs using sorter: heapSort
2023-03-13 03:23:41 INFO  Benchmark_Timer - Begin run: Helper for heapSort with 10000 elements with 100 runs
2023-03-13 03:23:41 INFO  TimeLogger - Raw time per run (mSec):  3.35
2023-03-13 03:23:41 INFO  TimeLogger - Normalized time per run (n log n):  4.72
ArrayLength: 20000
2023-03-13 03:23:41 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total
 elements and 100 runs using sorter: heapSort
2023-03-13 03:23:41 INFO  Benchmark_Timer - Begin run: Helper for heapSort with 20000 elements with 100 runs
2023-03-13 03:23:42 INFO  TimeLogger - Raw time per run (mSec):  7.23
2023-03-13 03:23:42 INFO  TimeLogger - Normalized time per run (n log n):  4.69
```

```
ArrayLength: 40000
2023-03-13 03:23:42 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total
 elements and 100 runs using sorter: heapSort
2023-03-13 03:23:42 INFO  Benchmark_Timer - Begin run: Helper for heapSort with 40000 elements with 100 runs
2023-03-13 03:23:44 INFO  TimeLogger - Raw time per run (mSec):  18.30
2023-03-13 03:23:44 INFO  TimeLogger - Normalized time per run (n log n):  5.51
ArrayLength: 80000
2023-03-13 03:23:44 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total
 elements and 100 runs using sorter: heapSort
2023-03-13 03:23:44 INFO  Benchmark_Timer - Begin run: Helper for heapSort with 80000 elements with 100 runs
2023-03-13 03:23:48 INFO  TimeLogger - Raw time per run (mSec):  35.26
2023-03-13 03:23:48 INFO  TimeLogger - Normalized time per run (n log n):  4.95
ArrayLength: 160000
2023-03-13 03:23:48 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total
 elements and 100 runs using sorter: heapSort
2023-03-13 03:23:48 INFO  Benchmark_Timer - Begin run: Helper for heapSort with 160000 elements with 100 runs
2023-03-13 03:24:03 INFO  TimeLogger - Raw time per run (mSec):  127.15
2023-03-13 03:24:03 INFO  TimeLogger - Normalized time per run (n log n):  8.36

Process finished with exit code 0
```