

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.

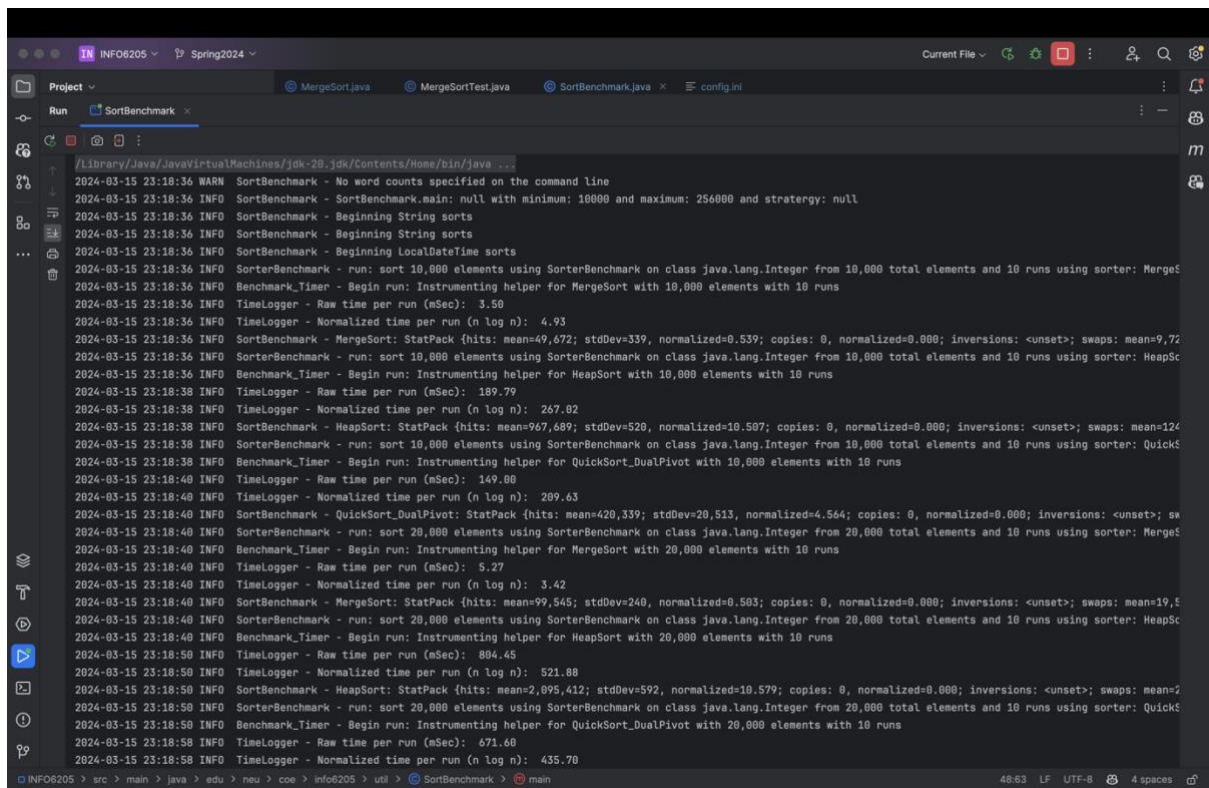
You must support your (clearly stated) conclusions with evidence from the benchmarks (you should provide log/log charts and spreadsheets typically).

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:



```
INFO6205 > Spring2024 > Current File > MergeSort.java > MergeSortTest.java > SortBenchmark.java > config.ini
Run > SortBenchmark > /Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...
2024-03-15 23:18:36 WARN SortBenchmark - No word counts specified on the command line
2024-03-15 23:18:36 INFO SortBenchmark - SortBenchmark.main: null with minimum: 10000 and maximum: 256000 and strategy: null
2024-03-15 23:18:36 INFO SortBenchmark - Beginning String sorts
2024-03-15 23:18:36 INFO SortBenchmark - Beginning String sorts
2024-03-15 23:18:36 INFO SortBenchmark - Beginning LocalDateTime sorts
2024-03-15 23:18:36 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 10 runs using sorter: MergeS
2024-03-15 23:18:36 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 10,000 elements with 10 runs
2024-03-15 23:18:36 INFO TimeLogger - Raw time per run (mSec): 3.50
2024-03-15 23:18:36 INFO TimeLogger - Normalized time per run (n log n): 4.93
2024-03-15 23:18:36 INFO SortBenchmark - MergeSort: StatPack {hits: mean=49,672; stdDev=339, normalized=0.539; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=9,72
2024-03-15 23:18:36 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 10 runs using sorter: HeapSc
2024-03-15 23:18:36 INFO Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 10,000 elements with 10 runs
2024-03-15 23:18:36 INFO TimeLogger - Raw time per run (mSec): 189.79
2024-03-15 23:18:36 INFO TimeLogger - Normalized time per run (n log n): 267.02
2024-03-15 23:18:36 INFO SortBenchmark - HeapSort: StatPack {hits: mean=967,689; stdDev=520, normalized=10.507; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124
2024-03-15 23:18:36 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 10 runs using sorter: QuickS
2024-03-15 23:18:36 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort_DualPivot with 10,000 elements with 10 runs
2024-03-15 23:18:40 INFO TimeLogger - Raw time per run (mSec): 149.00
2024-03-15 23:18:40 INFO TimeLogger - Normalized time per run (n log n): 209.63
2024-03-15 23:18:40 INFO SortBenchmark - QuickSort_DualPivot: StatPack {hits: mean=420,339; stdDev=20,513, normalized=4.564; copies: 0, normalized=0.000; inversions: <unset>; sw
2024-03-15 23:18:40 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sorter: MergeS
2024-03-15 23:18:40 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 20,000 elements with 10 runs
2024-03-15 23:18:40 INFO TimeLogger - Raw time per run (mSec): 5.27
2024-03-15 23:18:40 INFO TimeLogger - Normalized time per run (n log n): 3.42
2024-03-15 23:18:40 INFO SortBenchmark - MergeSort: StatPack {hits: mean=99,545; stdDev=240, normalized=0.503; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=19,5
2024-03-15 23:18:40 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sorter: HeapSc
2024-03-15 23:18:40 INFO Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 20,000 elements with 10 runs
2024-03-15 23:18:50 INFO TimeLogger - Raw time per run (mSec): 804.45
2024-03-15 23:18:50 INFO TimeLogger - Normalized time per run (n log n): 521.88
2024-03-15 23:18:50 INFO SortBenchmark - HeapSort: StatPack {hits: mean=2,095,412; stdDev=592, normalized=10.579; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=2
2024-03-15 23:18:50 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 10 runs using sorter: QuickS
2024-03-15 23:18:50 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort_DualPivot with 20,000 elements with 10 runs
2024-03-15 23:18:58 INFO TimeLogger - Raw time per run (mSec): 671.60
2024-03-15 23:18:58 INFO TimeLogger - Normalized time per run (n log n): 435.70
INFO6205 > src > main > java > edu > neu > coe > info6205 > util > SortBenchmark > main 48:53 LF UTF-8 4 spaces
```

Unit Test Benchmark:

Merge Sort

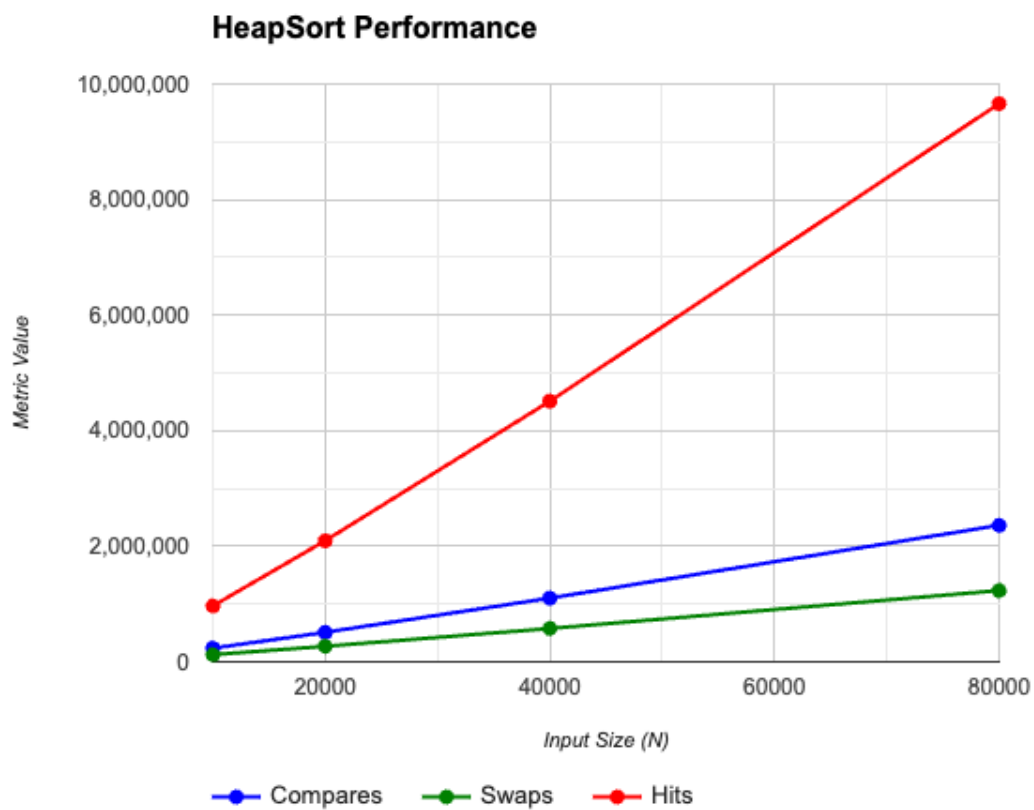
```
INFO6205 Spring2024
Project MergeSort.java MergeSortTest.java SorterBenchmark.java config.ini
Run MergeSortTest
MergeSortTest [edu.neu.coe.jsr] 232 ms Tests passed: 15 of 15 tests - 232 ms
  ✓ testSort11_partialsorted 66 ms
  ✓ testSort9_partialsorted 32 ms
  ✓ testSort1 2 ms
  ✓ testSort2 4 ms
  ✓ testSort3 1 ms
  ✓ testSort4 51 ms
  ✓ testSort5 14 ms
  ✓ testSort6 11 ms
  ✓ testSort7 9 ms
  ✓ testSort10_partialsorted 18 ms
  ✓ testSort8_partialsorted 4 ms
  ✓ testSort12 0 ms
  ✓ testSort14 1 ms
  ✓ testSort1a 1 ms

/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance + NoCopy: 51966
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + NoCopy: 38987
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,790, normalized=2.882; copies: 640, normalized=1.030; inversions: 4,224, normalized=6.801; swaps: 101, normalized=0.163; fi
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; inversions: <unset>; swaps: 0, normalized=0.000; fixes: 0, normalized
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff: 50225
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + NoCopy: 13285
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance: 9502
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance + NoCopy: 8443
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance: 17148
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff: 17476
testing Helper for MergeSort: with insurance comparison with 8 elements
testing Helper for MergeSort: with no copy with 8 elements
testing Helper for MergeSort: with insurance comparison with no copy with 8 elements

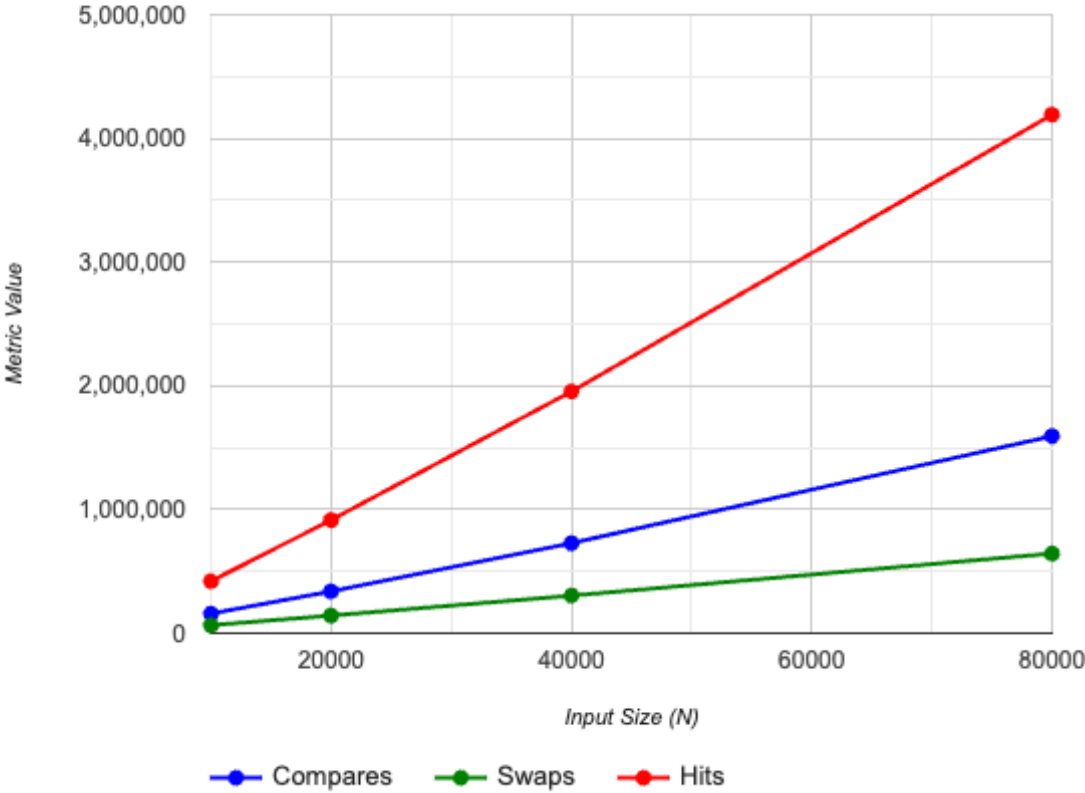
Process finished with exit code 0
INFO6205 > src > test > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSortTest 24:14 LF UTF-8 4 spaces
```

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



MergeSort Performance

