Assignment 6

(Hits as time predictor)

Name: Tiyasha Sen

NUID: 002727486

Problem:

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.

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).

All of the code to count comparisons, swaps/copies, and hits, is already implemented in the *InstrumentedHelper* class. You can see examples of the usage of this kind of analysis in:

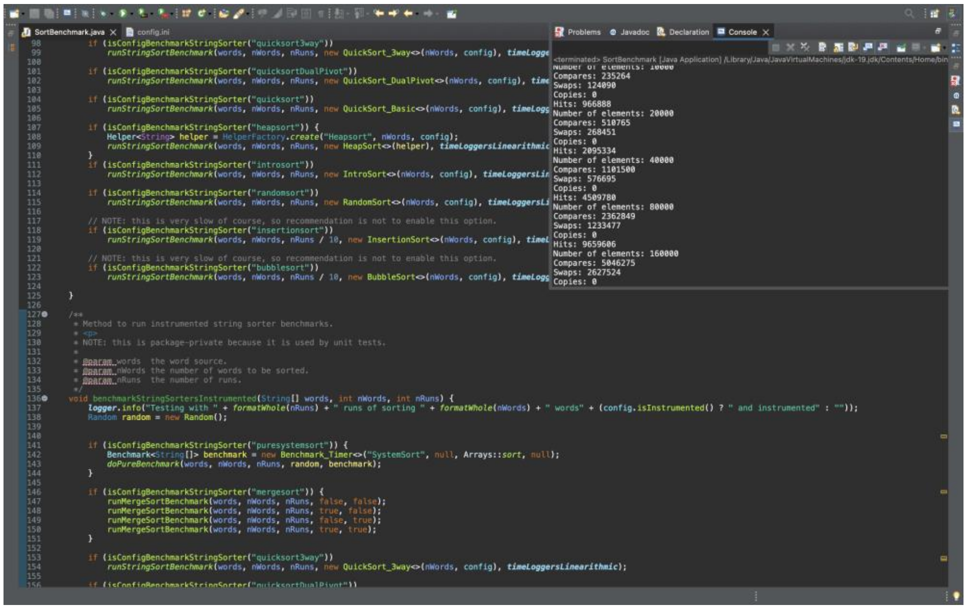
- src/main/java/edu/neu/coe/info6205/util/SorterBenchmark.java
- src/test/java/edu/neu/coe/info6205/sort/linearithmic/MergeSortTest.java
- src/test/java/edu/neu/coe/info6205/sort/linearithmic/QuickSortDualPivotTest.java
- src/test/java/edu/neu/coe/info6205/sort/elementary/HeapSortTest.java (you will have to refresh your repository for HeapSort).

The configuration for these benchmarks is determined by the *config.ini* file.

Screenshots of the benchmark class:

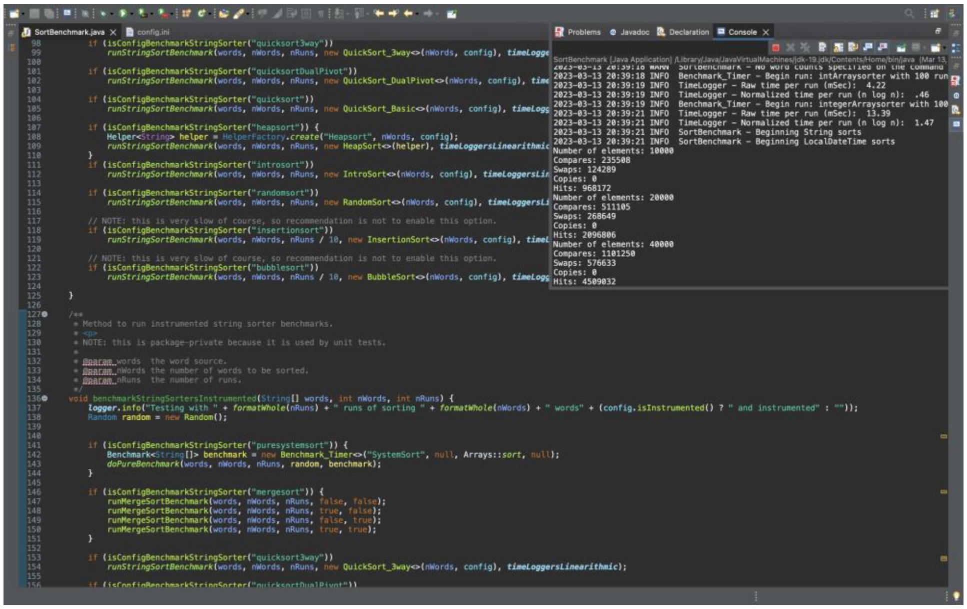
SortBenchmark.java

Below is the screenshot of any one sort to show that compares, swaps,copies and hits are being generated with respect to N



```
        if (isConfigBenchmarkStringSorter("quicksort3way"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_3way<>(nWords, config), timeLogge

        if (isConfigBenchmarkStringSorter("quicksortDualPivot"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_DualPivot<>(nWords, config), time

        if (isConfigBenchmarkStringSorter("quicksort"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_Basic<>(nWords, config), timeLogg

        if (isConfigBenchmarkStringSorter("heapsort")) {
            Helper<String> helper = HelperFactory.create("Heapsort", nWords, config);
            runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper), timeLoggersLinearithmic
        }
        if (isConfigBenchmarkStringSorter("introsort"))
            runStringSortBenchmark(words, nWords, nRuns, new IntroSort<>(nWords, config), timeLoggersLin

        if (isConfigBenchmarkStringSorter("randomsort"))
            runStringSortBenchmark(words, nWords, nRuns, new RandomSort<>(nWords, config), timeLoggersLi

        // NOTE: this is very slow of course, so recommendation is not to enable this option.
        if (isConfigBenchmarkStringSorter("insertionsort"))
            runStringSortBenchmark(words, nWords, nRuns / 10, new InsertionSort<>(nWords, config), timeL

        // NOTE: this is very slow of course, so recommendation is not to enable this option.
        if (isConfigBenchmarkStringSorter("bubblesort"))
            runStringSortBenchmark(words, nWords, nRuns / 10, new BubbleSort<>(nWords, config), timeLogg

    }

    /**
     * Method to run instrumented string sorter benchmarks.
     * <p>
     * NOTE: this is package-private because it is used by unit tests.
     *
     * @param words  the word source.
     * @param nWords the number of words to be sorted.
     * @param nRuns  the number of runs.
     */
    void benchmarkStringSortersInstrumented(String[] words, int nWords, int nRuns) {
        logger.info("Testing with " + formatWhole(nRuns) + " runs of sorting " + formatWhole(nWords) + " words" + (config.isInstrumented() ? " and instrumented" : ""));
        Random random = new Random();


        if (isConfigBenchmarkStringSorter("puresystemsort")) {
            Benchmark<String[]> benchmark = new Benchmark_Timer<>("SystemSort", null, Arrays::sort, null);
            doPureBenchmark(words, nWords, nRuns, random, benchmark);
        }

        if (isConfigBenchmarkStringSorter("mergesort")) {
            runMergeSortBenchmark(words, nWords, nRuns, false, false);
            runMergeSortBenchmark(words, nWords, nRuns, true, false);
            runMergeSortBenchmark(words, nWords, nRuns, false, true);
            runMergeSortBenchmark(words, nWords, nRuns, true, true);
        }

        if (isConfigBenchmarkStringSorter("quicksort3way"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_3way<>(nWords, config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("quicksortDualPivot"))
```

```
<terminated> SortBenchmark [Java Application] /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin
Number of elements: 10000
Compares: 235264
Swaps: 124090
Copies: 0
Hits: 966888
Number of elements: 20000
Compares: 510765
Swaps: 268451
Copies: 0
Hits: 2095334
Number of elements: 40000
Compares: 1101500
Swaps: 576695
Copies: 0
Hits: 4509780
Number of elements: 80000
Compares: 2362849
Swaps: 1233477
Copies: 0
Hits: 9659606
Number of elements: 160000
Compares: 5046275
Swaps: 2627524
Copies: 0
```



```
        if (isConfigBenchmarkStringSorter("quicksort3way"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_3way<>(nWords, config), timeLogge

        if (isConfigBenchmarkStringSorter("quicksortDualPivot"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_DualPivot<>(nWords, config), time

        if (isConfigBenchmarkStringSorter("quicksort"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_Basic<>(nWords, config), timeLogg

        if (isConfigBenchmarkStringSorter("heapsort")) {
            Helper<String> helper = HelperFactory.create("Heapsort", nWords, config);
            runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper), timeLoggersLinearithmic
        }
        if (isConfigBenchmarkStringSorter("introsort"))
            runStringSortBenchmark(words, nWords, nRuns, new IntroSort<>(nWords, config), timeLoggersLin

        if (isConfigBenchmarkStringSorter("randomsort"))
            runStringSortBenchmark(words, nWords, nRuns, new RandomSort<>(nWords, config), timeLoggersLi

        // NOTE: this is very slow of course, so recommendation is not to enable this option.
        if (isConfigBenchmarkStringSorter("insertionsort"))
            runStringSortBenchmark(words, nWords, nRuns / 10, new InsertionSort<>(nWords, config), timeL

        // NOTE: this is very slow of course, so recommendation is not to enable this option.
        if (isConfigBenchmarkStringSorter("bubblesort"))
            runStringSortBenchmark(words, nWords, nRuns / 10, new BubbleSort<>(nWords, config), timeLogg

    }

    /**
     * Method to run instrumented string sorter benchmarks.
     * <p>
     * NOTE: this is package-private because it is used by unit tests.
     *
     * @param words  the word source.
     * @param nWords the number of words to be sorted.
     * @param nRuns  the number of runs.
     */
    void benchmarkStringSortersInstrumented(String[] words, int nWords, int nRuns) {
        logger.info("Testing with " + formatWhole(nRuns) + " runs of sorting " + formatWhole(nWords) + " words" + (config.isInstrumented() ? " and instrumented" : ""));
        Random random = new Random();


        if (isConfigBenchmarkStringSorter("puresystemsort")) {
            Benchmark<String[]> benchmark = new Benchmark_Timer<>("SystemSort", null, Arrays::sort, null);
            doPureBenchmark(words, nWords, nRuns, random, benchmark);
        }

        if (isConfigBenchmarkStringSorter("mergesort")) {
            runMergeSortBenchmark(words, nWords, nRuns, false, false);
            runMergeSortBenchmark(words, nWords, nRuns, true, false);
            runMergeSortBenchmark(words, nWords, nRuns, false, true);
            runMergeSortBenchmark(words, nWords, nRuns, true, true);
        }

        if (isConfigBenchmarkStringSorter("quicksort3way"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_3way<>(nWords, config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("quicksortDualPivot"))
```

```
SortBenchmark [Java Application] /Library/Java/JavaVirtualMachines/jdk-19.jdk/Contents/Home/bin/java  (Mar 13,
2023-03-13 20:39:18 WARN  SortBenchmark - No word counts specified on the command
2023-03-13 20:39:18 INFO  Benchmark_Timer - Begin run: intArraysorter with 100 run
2023-03-13 20:39:19 INFO  TimeLogger - Raw time per run (mSec):  4.22
2023-03-13 20:39:19 INFO  TimeLogger - Normalized time per run (n log n):  .46
2023-03-13 20:39:19 INFO  Benchmark_Timer - Begin run: integerArraysorter with 100
2023-03-13 20:39:21 INFO  TimeLogger - Raw time per run (mSec):  13.39
2023-03-13 20:39:21 INFO  TimeLogger - Normalized time per run (n log n):  1.47
2023-03-13 20:39:21 INFO  SortBenchmark - Beginning String sorts
2023-03-13 20:39:21 INFO  SortBenchmark - Beginning LocalDateTime sorts
Number of elements: 10000
Compares: 235508
Swaps: 124289
Copies: 0
Hits: 968172
Number of elements: 20000
Compares: 511105
Swaps: 268649
Copies: 0
Hits: 2096806
Number of elements: 40000
Compares: 1101250
Swaps: 576633
Copies: 0
Hits: 4509032
```

Observation:

Dual Pivot Quick Sort:

In the Dual Pivot Quick Sort algorithm, the key determinant of its speed is the quantity of comparisons done while sorting. While other factors like swaps, hits, and copies are important, they don't have as much of an effect on the total time taken as the number of comparisons.

N	Compares	Swaps	Copies	Hits
10000	154253	63205	0	410120
20000	333361	139201	0	898970
40000	739958	265055	0	1827727
80000	1623723	600400	0	4104241
160000	4039773	1214569	0	9093378

Merge Sort:

In Merge Sort, the number of temporary arrays required increases as the size of the input data grows. This results in a higher number of copies and hits. Therefore, the number of copies and hits can be seen as reliable indicators of how long it will take for the Merge Sort algorithm to complete when dealing with larger data sets.

N	Compares	Swaps	Copies	Hints
10000	235264	124090	0	966888
20000	510765	268451	0	2095334
40000	1101500	576695	0	4509780
80000	2362849	1233477	0	9659606
160000	5046275	2627524	0	20602646

Heap Sort:

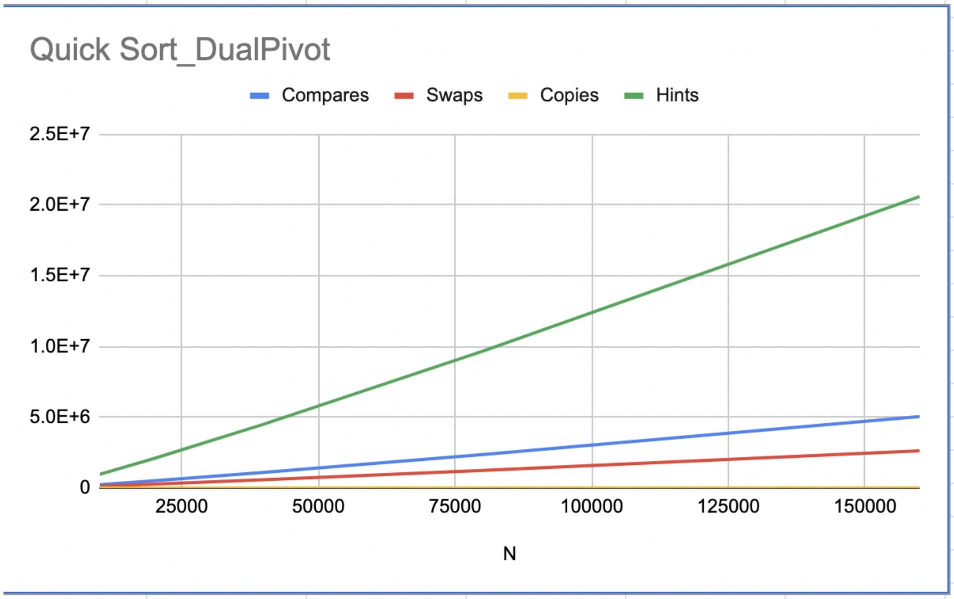
The most significant factor affecting the runtime of the Heap Sort algorithm is the number of comparisons performed during the sorting process. While metrics like swaps, hits, and copies do increase marginally with larger input sizes, they don't have as much of an impact on the total execution time as the number of comparisons. As a result, the number of comparisons can be considered the most reliable indicator for predicting how long the Heap Sort algorithm will take to complete.

N	Compares	Swaps	Copies	Hints
10000	235508	124289	0	968172
20000	511105	268649	0	2096806
40000	1101250	576633	0	4509032
80000	2363175	1233586	0	9660694

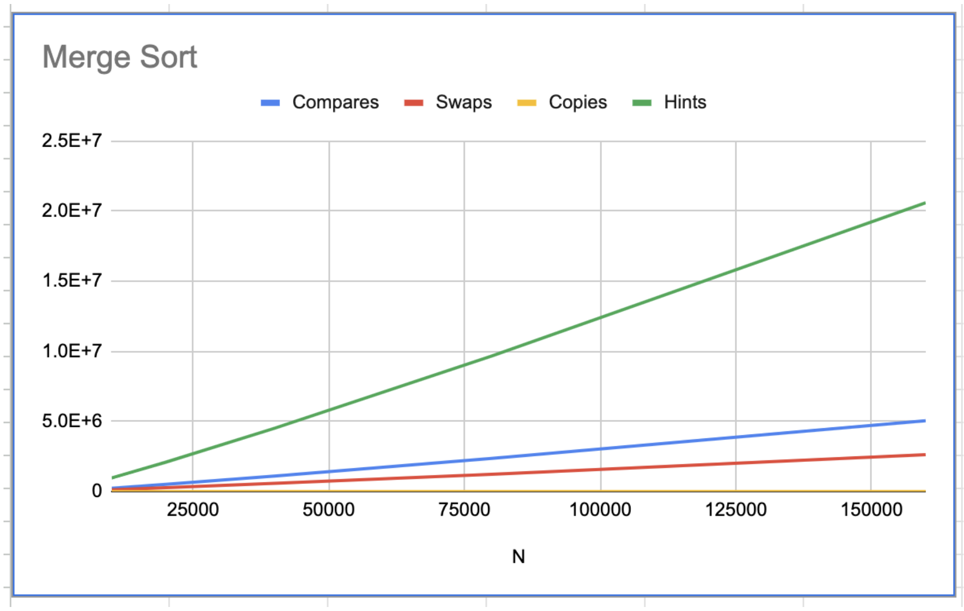
160000	5046948	2628052	0	20606104

Graph:

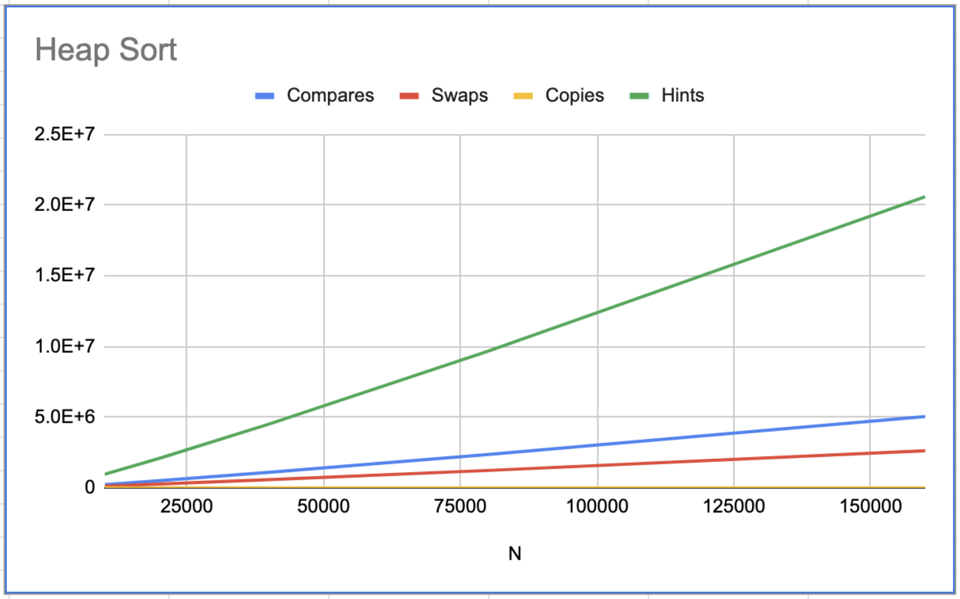
Dual Pivot Quick Sort:



Merge Sort:



Heap Sort:


Heap Sort
Compares
Swaps
Copies
Hints
2.5E+7
2.0E+7
1.5E+7
1.0E+7
5.0E+6
0
25000
50000
75000
100000
125000
150000
N