Program Structures and Algorithms
Spring 2023(SEC - 8)

NAME: Dinesh Singh Shekhawat
NUID: 002776484

TASK:
To determine the best predictor of total execution time by running benchmarks for merge sort, quick sort (dual-pivot), and heap sort. Following parameters are checked for determining the best predictor

a. swaps
b. compares
c. copies
d. hits

RELATIONSHIP CONCLUSION:
The number of array access i.e., hits, is overall the best predictor of total execution time for comparison-based sorting algorithms like quick sort, merge sort and heap sort.

Array access can impact the execution time of sorting algorithms because accessing elements in an array can be an expensive operation in terms of time and resources.

Additionally, the way elements are accessed in an array can also affect the performance of a sorting algorithm. For example, if an algorithm needs to access elements in a random order, this can lead to frequent cache misses, where the data being accessed is not already in the cache, and must be retrieved from main memory, which can be slower than accessing data in the cache.

For merge sort extra memory is required for copying values and there is no swapping taking place. While copying values from the auxiliary memory comparisons are done to move pointers in the sub arrays. Therefore, the decreasing order of effect on the total time is Hits, Copies, Compare and swap (which is almost zero).

However, for heap sort and quicksort there is no need for extra memory hence no time is taken for copy operations since they rely on creating proper partitions within the array itself by continuously comparing and swapping elements. Therefore, the decreasing order of effect on the total time is Hits, Compare and swap. Copy doesn't play any role here at all.

Also on plotting the normalized values of all the metrics it can be seen that the hits graph is very close to the time graph for all the sorting algorithms suggesting that it is the best predictor of total execution time.

EVIDENCE TO SUPPORT THAT CONCLUSION

Source Code

SortBenchmark

```
/*
  (c) Copyright 2018, 2019 Phasmid Software
 */
package edu.neu.coe.info6205.util;

import edu.neu.coe.info6205.sort.BaseHelper;
import edu.neu.coe.info6205.sort.Helper;
import edu.neu.coe.info6205.sort.HelperFactory;
import edu.neu.coe.info6205.sort.SortWithHelper;
import edu.neu.coe.info6205.sort.elementary.BubbleSort;
import edu.neu.coe.info6205.sort.elementary.HeapSort;
import edu.neu.coe.info6205.sort.elementary.InsertionSort;
import edu.neu.coe.info6205.sort.elementary.RandomSort;
import edu.neu.coe.info6205.sort.elementary.ShellSort;
import edu.neu.coe.info6205.sort.linearithmic.TimSort;
import edu.neu.coe.info6205.sort.linearithmic.*;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.reflect.Array;
import java.time.LocalDateTime;
import java.time.chrono.ChronoLocalDateTime;
import java.util.*;
import java.util.function.Consumer;
import java.util.function.Supplier;
import java.util.function.UnaryOperator;
import java.util.regex.Pattern;
import java.util.stream.Stream;

import static
edu.neu.coe.info6205.util.SortBenchmarkHelper.generateRandomLocalDateTimeAr
ray;
import static edu.neu.coe.info6205.util.SortBenchmarkHelper.getWords;
import static edu.neu.coe.info6205.util.Utilities.formatWhole;

public class SortBenchmark {
```

```
    public SortBenchmark(Config config) {
        this.config = config;
    }

    public static void main(String[] args) throws IOException {
        Config config = Config.load(SortBenchmark.class);
        logger.info("SortBenchmark.main: " + config.get("sortbenchmark",
"version") + " with word counts: " + Arrays.toString(args));
        if (args.length == 0) logger.warn("No word counts specified on the
command line");
        SortBenchmark benchmark = new SortBenchmark(config);
        benchmark.sortIntegersByShellSort(config.getInt("shellsort", "n",
100000));
        benchmark.sortStrings(Arrays.stream(args).map(Integer::parseInt));
        benchmark.sortLocalDateTimes(config.getInt("benchmarkdatesorters",
"n", 100000), config);
    }

    public void sortLocalDateTimes(final int n, Config config) throws
IOException {
        logger.info("Beginning LocalDateTime sorts");
        // CONSIDER why do we have localDateTimeSupplier IN ADDITION TO
localDateTimes?
        Supplier<LocalDateTime[]> localDateTimeSupplier = () ->
generateRandomLocalDateTimeArray(n);
        Helper<ChronoLocalDateTime<?>> helper = new
BaseHelper<>("DateTimeHelper", config);
        final LocalDateTime[] localDateTimes =
generateRandomLocalDateTimeArray(n);

        // CONSIDER finding the common ground amongst these sorts and get
them all working together.

        // NOTE Test on date using pure tim sort.
        if (isConfigBenchmarkDateSorter("timsort"))
            logger.info(benchmarkFactory("Sort LocalDateTimes using
Arrays::sort (TimSort)", Arrays::sort,
null).runFromSupplier(localDateTimeSupplier, 100) + "ms");

        // NOTE this is supposed to match the previous benchmark run
exactly. I don't understand why it takes rather less time.
        if (isConfigBenchmarkDateSorter("timsort")) {
```

```
            logger.info(benchmarkFactory("Repeat Sort LocalDateTimes using
timSort::mutatingSort", new TimSort<>(helper)::mutatingSort,
null).runFromSupplier(localDateTimeSupplier, 100) + "ms");
            // NOTE this is intended to replace the run two lines previous.
It should take the exact same amount of time.
            runDateTimeSortBenchmark(LocalDateTime.class, localDateTimes,
n, 100);
        }
    }

    /**
     * Method to run pure (non-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 benchmarkStringSorters(String[] words, int nWords, int nRuns) {
        logger.info("Testing pure sorts with " + formatWhole(nRuns) + "
runs of sorting " + formatWhole(nWords) + " words");
        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);
            */

            runStringSortBenchmark(words, nWords, nRuns, new
MergeSortBasic<>(nWords, config), timeLoggersLinearithmic);
        }
```

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

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

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

        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), timeLoggersLinearithmic);

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

        // 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), timeLoggersQuadratic);

        // 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), timeLoggersQuadratic);

    }

    /**
```

```
     * 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);
            */

            runStringSortBenchmark(words, nWords, nRuns, new
MergeSortBasic<>(nWords, config), timeLoggersLinearithmic);
        }

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

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

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

        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), timeLoggersLinearithmic);

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

        // 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), timeLoggersQuadratic);

        // 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), timeLoggersQuadratic);
    }

    // CONSIDER generifying common code (but it's difficult if not
impossible)
    private void sortIntegersByShellSort(final int n) {
        final Random random = new Random();

        // sort int[]
        final Supplier<int[]> intsSupplier = () -> {
            int[] result = (int[]) Array.newInstance(int.class, n);
            for (int i = 0; i < n; i++) result[i] = random.nextInt();
```

```
            return result;
        };

        final double t1 = new Benchmark_Timer<int[]>(
                "intArraysorter",
                (xs) -> Arrays.copyOf(xs, xs.length),
                Arrays::sort,
                null
        ).runFromSupplier(intsSupplier, 100);
        for (TimeLogger timeLogger : timeLoggersLinearithmic)
timeLogger.log(t1, n);

        // sort Integer[]
        final Supplier<Integer[]> integersSupplier = () -> {
            Integer[] result = (Integer[]) Array.newInstance(Integer.class,
n);
            for (int i = 0; i < n; i++) result[i] = random.nextInt();
            return result;
        };

        final double t2 = new Benchmark_Timer<Integer[]>(
                "integerArraysorter",
                (xs) -> Arrays.copyOf(xs, xs.length),
                Arrays::sort,
                null
        ).runFromSupplier(integersSupplier, 100);
        for (TimeLogger timeLogger : timeLoggersLinearithmic)
timeLogger.log(t2, n);
    }

    // This was added by a Student. Need to figure out what to do with it.
What's different from the method with int parameter??
    private void sortIntegersByShellSort() throws IOException {
        if (isConfigBenchmarkIntegerSorter("shellsort")) {
            final Random random = new Random();
            int N = 1000;
            for (int j = 0; j < 10; j++) {
                Integer[] numbers = new Integer[N];
                for (int i = 0; i < N; i++) numbers[i] = random.nextInt();

                SortWithHelper<Integer> sorter = new ShellSort<>(5);
                runIntegerSortBenchmark(numbers, N, 1000, sorter,
```

```
sorter::preProcess, timeLoggersLinearithmic);
                N = N * 2;
            }
        }
    }

    private void sortStrings(Stream<Integer> wordCounts) {
        logger.info("Beginning String sorts");

        // NOTE: common words benchmark
//        benchmarkStringSorters(getWords("3000-common-words.txt",
SortBenchmark::lineAsList), config.getInt("benchmarkstringsorters",
"words", 1000), config.getInt("benchmarkstringsorters", "runs", 1000));

        // NOTE: Leipzig English words benchmarks (according to
command-line arguments)
        wordCounts.forEach(this::doLeipzigBenchmarkEnglish);

        // NOTE: Leipzig Chines words benchmarks (according to command-line
arguments)
//        doLeipzigBenchmark("zho-simp-tw_web_2014_10K-sentences.txt",
5000, 1000);
    }

    private void doLeipzigBenchmarkEnglish(int x) {
        String resource = "eng-uk_web_2002_" + (x < 50000 ? "10K" : x <
200000 ? "100K" : "1M") + "-sentences.txt";
        try {
            doLeipzigBenchmark(resource, x, Utilities.round(100000000 /
minComparisons(x)));
        } catch (FileNotFoundException e) {
            logger.warn("Unable to find resource: " + resource, e);
        }
    }

    /**
     * Method to run a sorting benchmark, using an explicit preProcessor.
     *
     * @param words        an array of available words (to be chosen
randomly).
     * @param nWords       the number of words to be sorted.
     * @param nRuns        the number of runs of the sort to be preformed.
```

```
     * @param sorter       the sorter to use--NOTE that this sorter will be
closed at the end of this method.
     * @param preProcessor the pre-processor function, if any.
     * @param timeLoggers  a set of timeLoggers to be used.
     */
    static void runStringSortBenchmark(String[] words, int nWords, int
nRuns, SortWithHelper<String> sorter, UnaryOperator<String[]> preProcessor,
TimeLogger[] timeLoggers) {
        new SorterBenchmark<>(String.class, preProcessor, sorter, words,
nRuns, timeLoggers).run(nWords);
        sorter.close();
    }

    /**
     * Method to run a sorting benchmark using the standard preProcess
method of the sorter.
     *
     * @param words       an array of available words (to be chosen
randomly).
     * @param nWords      the number of words to be sorted.
     * @param nRuns       the number of runs of the sort to be preformed.
     * @param sorter      the sorter to use--NOTE that this sorter will be
closed at the end of this method.
     * @param timeLoggers a set of timeLoggers to be used.
     *                    <p>
     *                    NOTE: this method is public because it is
referenced in a unit test of a different package
     */
    public static void runStringSortBenchmark(String[] words, int nWords,
int nRuns, SortWithHelper<String> sorter, TimeLogger[] timeLoggers) {
        runStringSortBenchmark(words, nWords, nRuns, sorter,
sorter::preProcess, timeLoggers);
    }

    /**
     * Method to run a sorting benchmark, using an explicit preProcessor.
     *
     * @param numbers      an array of available integers (to be chosen
randomly).
     * @param n            the number of integers to be sorted.
     * @param nRuns        the number of runs of the sort to be preformed.
     * @param sorter       the sorter to use--NOTE that this sorter will be
```

```
closed at the end of this method.
     * @param preProcessor the pre-processor function, if any.
     * @param timeLoggers  a set of timeLoggers to be used.
     */
    static void runIntegerSortBenchmark(Integer[] numbers, int n, int
nRuns, SortWithHelper<Integer> sorter, UnaryOperator<Integer[]>
preProcessor, TimeLogger[] timeLoggers) {
        new SorterBenchmark<>(Integer.class, preProcessor, sorter, numbers,
nRuns, timeLoggers).run(n);
        sorter.close();
    }

    /**
     * For mergesort, the number of array accesses is actually 6 times the
number of comparisons.
     * That's because, in addition to each comparison, there will be
approximately two copy operations.
     * Thus, in the case where comparisons are based on primitives,
     * the normalized time per run should approximate the time for one
array access.
     */
    public final static TimeLogger[] timeLoggersLinearithmic = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger("Normalized time per run (n log n): ", (time, n)
-> time / minComparisons(n) / 6 * 1e6)
    };

    final static LazyLogger logger = new LazyLogger(SortBenchmark.class);

    final static Pattern regexLeipzig =
Pattern.compile("[~\\t]*\\t(([\\s\\p{Punct}\\uFF0C]*\\p{L}+)*)");

    /**
     * This is based on log2(n!)
     *
     * @param n the number of elements.
     * @return the minimum number of comparisons possible to sort n
randomly ordered elements.
     */
    static double minComparisons(int n) {
        double lgN = Utilities.lg(n);
        return n * (lgN - LgE) + lgN / 2 + 1.33;
```

```
    }

    /**
     * This is the mean number of inversions in a randomly ordered set of n
elements.
     * For insertion sort, each (low-level) swap fixes one inversion, so on
average there are this number of swaps.
     * The minimum number of comparisons is slightly higher.
     *
     * @param n the number of elements
     * @return one quarter n-squared more or less.
     */
    static double meanInversions(int n) {
        return 0.25 * n * (n - 1);
    }

    private static Collection<String> lineAsList(String line) {
        List<String> words = new ArrayList<>();
        words.add(line);
        return words;
    }

    public static Collection<String> getLeipzigWords(String line) {
        return getWords(regexLeipzig, line);
    }

    // CONSIDER: to be eliminated soon.
    private static Benchmark<LocalDateTime[]> benchmarkFactory(String
description, Consumer<LocalDateTime[]> sorter, Consumer<LocalDateTime[]>
checker) {
        return new Benchmark_Timer<>(
                description,
                (xs) -> Arrays.copyOf(xs, xs.length),
                sorter,
                checker
        );
    }

    private static void doPureBenchmark(String[] words, int nWords, int
nRuns, Random random, Benchmark<String[]> benchmark) {
        // CONSIDER we should manage the space returned by fillRandomArray
and deallocate it after use.
```

```
        final double time = benchmark.runFromSupplier(() ->
Utilities.fillRandomArray(String.class, random, nWords, r ->
words[r.nextInt(words.length)]), nRuns);
        for (TimeLogger timeLogger : timeLoggersLinearithmic)
timeLogger.log(time, nWords);
    }

//    private void dateSortBenchmark(Supplier<LocalDateTime[]>
localDateTimeSupplier, LocalDateTime[] localDateTimes,
Sort<ChronoLocalDateTime<?>> dateHuskySortSystemSort, String s, int i) {
//        logger.info(benchmarkFactory(s, dateHuskySortSystemSort::sort,
dateHuskySortSystemSort::postProcess).runFromSupplier(localDateTimeSupplier
, 100) + "ms");
//        // NOTE: this is intended to replace the run in the previous
line. It should take the exact same amount of time.
//        runDateTimeSortBenchmark(LocalDateTime.class, localDateTimes,
100000, 100, i);
//    }

    private void runMergeSortBenchmark(String[] words, int nWords, int
nRuns, Boolean insurance, Boolean noCopy) {
        Config x = config.copy(MergeSort.MERGESORT, MergeSort.INSURANCE,
insurance.toString()).copy(MergeSort.MERGESORT, MergeSort.NOCOPY,
noCopy.toString());
        runStringSortBenchmark(words, nWords, nRuns, new
MergeSort<>(nWords, x), timeLoggersLinearithmic);
    }

    private void doLeipzigBenchmark(String resource, int nWords, int nRuns)
throws FileNotFoundException {
        benchmarkStringSorters(getWords(resource,
SortBenchmark::getLeipzigWords), nWords, nRuns);
        if (isConfigBoolean(Config.HELPER, BaseHelper.INSTRUMENT))
            benchmarkStringSortersInstrumented(getWords(resource,
SortBenchmark::getLeipzigWords), nWords, nRuns);
    }

    @SuppressWarnings("SameParameterValue")
    private void runDateTimeSortBenchmark(Class<?> tClass,
ChronoLocalDateTime<?>[] dateTimes, int N, int m) throws IOException {
        final SortWithHelper<ChronoLocalDateTime<?>> sorter = new
TimSort<>();
```

```
        @SuppressWarnings("unchecked") final
SorterBenchmark<ChronoLocalDateTime<?>> sorterBenchmark = new
SorterBenchmark<>((Class<ChronoLocalDateTime<?>>) tClass, (xs) ->
Arrays.copyOf(xs, xs.length), sorter, dateTimes, m,
timeLoggersLinearithmic);
        sorterBenchmark.run(N);
    }

    /**
     * For (basic) insertionsort, the number of array accesses is actually
6 times the number of comparisons.
     * That's because, for each inversion, there will typically be one swap
(four array accesses) and (at least) one comparison (two array accesses).
     * Thus, in the case where comparisons are based on primitives,
     * the normalized time per run should approximate the time for one
array access.
     */
    private final static TimeLogger[] timeLoggersQuadratic = {
            new TimeLogger("Raw time per run (mSec): ", (time, n) -> time),
            new TimeLogger("Normalized time per run (n^2): ", (time, n) ->
time / meanInversions(n) / 6 * 1e6)
    };

    private static final double LgE = Utilities.lg(Math.E);

    private boolean isConfigBenchmarkStringSorter(String option) {
        return isConfigBoolean("benchmarkstringsorters", option);
    }

    private boolean isConfigBenchmarkDateSorter(String option) {
        return isConfigBoolean("benchmarkdatesorters", option);
    }

    private boolean isConfigBenchmarkIntegerSorter(String option) {
        return isConfigBoolean("benchmarkintegersorters", option);
    }

    private boolean isConfigBoolean(String section, String option) {
        return config.getBoolean(section, option);
    }
    private final Config config;
}
```

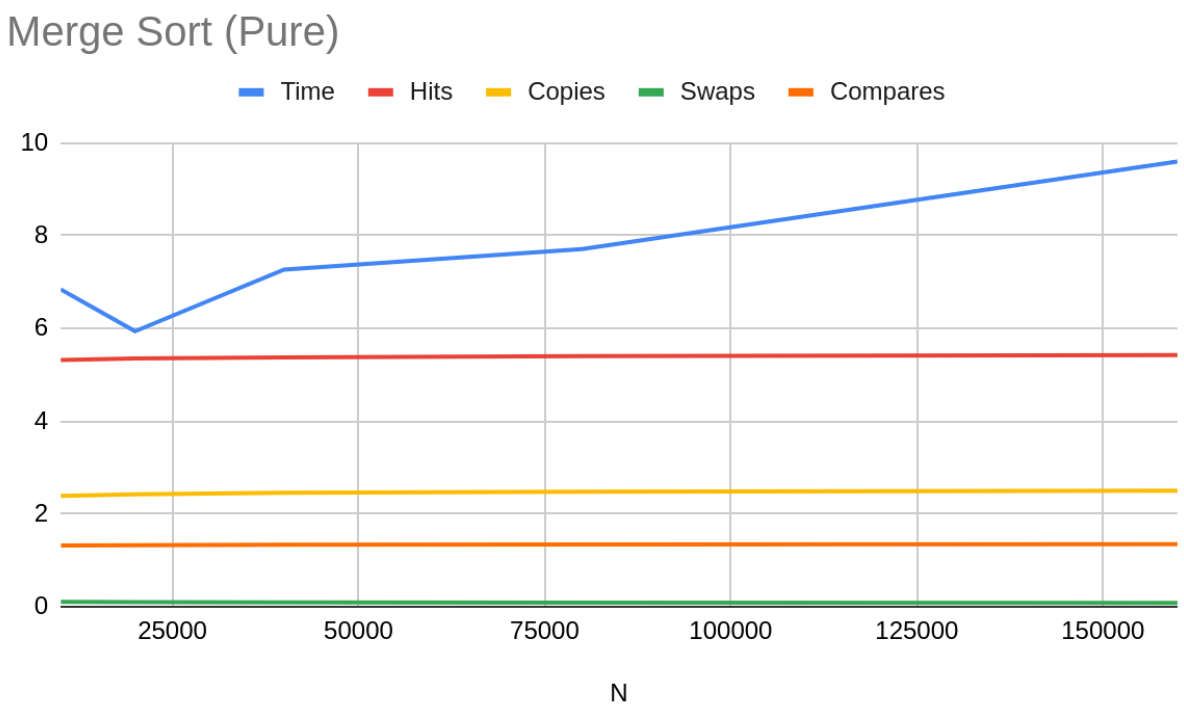
SorterBenchmark

```
package edu.neu.coe.info6205.util;

import static edu.neu.coe.info6205.util.Utilities.formatWhole;

import java.util.function.Consumer;
import java.util.function.UnaryOperator;
import edu.neu.coe.info6205.sort.Helper;
import edu.neu.coe.info6205.sort.InstrumentedHelper;
import edu.neu.coe.info6205.sort.SortWithHelper;
/**
 * Class to extend Benchmark_Timer for sorting an array of T values.
 * The default implementation of run in this class randomly selects a
subset of the array to be sorted.
 * Each sort is preceded (optionally) by a preProcessor and succeeded
(optionally) by a postProcessor.
 *
 * @param <T> the underlying type to be sorted.
 */
public class SorterBenchmark<T extends Comparable<T>> extends
Benchmark_Timer<T[]> {
    /**
     * Run a benchmark on a sorting problem with N elements.
     *
     * @param N the number of elements.
     *          Not to be confused with nRuns, an instance field, which
specifies the number of repetitions of the function.
     */
    public void run(int N) {
        logger.info("run: sort " + formatWhole(N) + " elements using " +
this);
        sorter.init(N);
        final double time = super.runFromSupplier(() ->
generateRandomArray(ts), nRuns);
        for (TimeLogger timeLogger : timeLoggers) timeLogger.log(time, N);
        Helper<T> helper = sorter.getHelper();
        if (helper.instrumented()) {
            InstrumentedHelper<T> instrumentedHelper =
(InstrumentedHelper<T>) helper;
            logger.info("Instrumentation:::: "  +
instrumentedHelper.showStats());
```

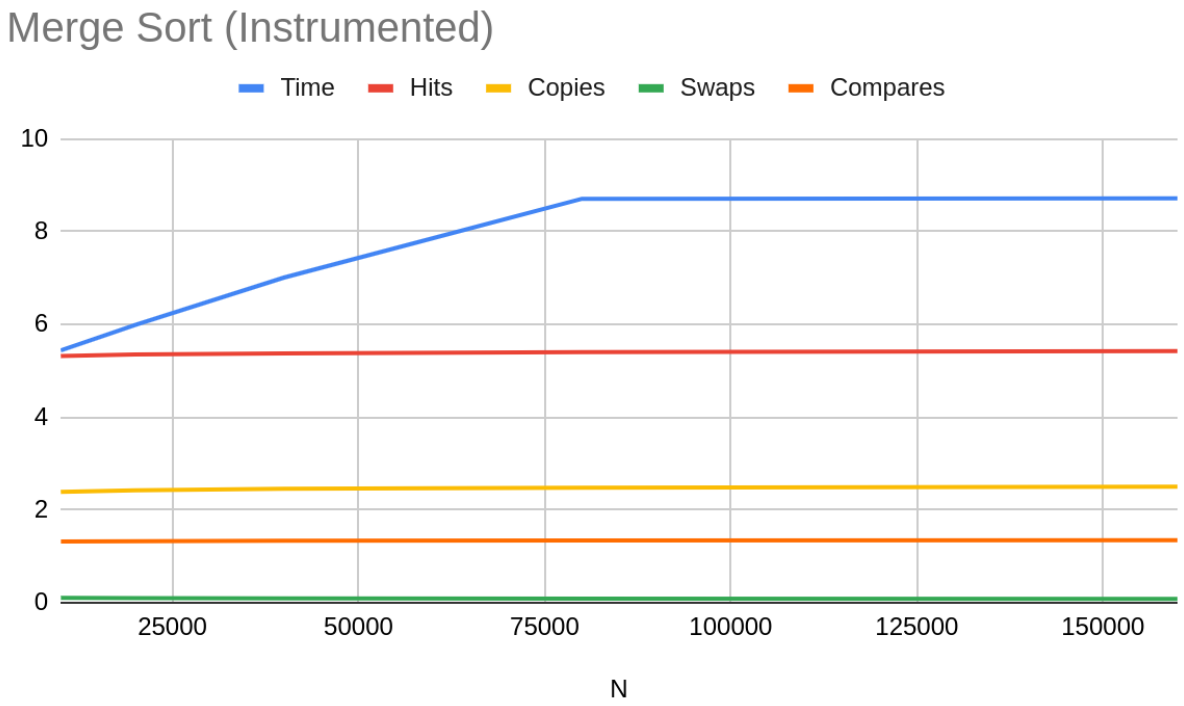
```
        } else {
            logger.info("NO INSTRUMENTATION::::"  + helper.showStats());
        }
    }
    @Override
    public String toString() {
        return "SorterBenchmark on " + tClass + " from " +
formatWhole(ts.length) + " total elements and " + formatWhole(nRuns) + "
runs using sorter: " + sorter.getHelper().getDescription();
    }
    /**
     * Constructor for a SorterBenchmark where we provide the following
parameters:
     *
     * @param tClass        the class of T.
     * @param preProcessor  an optional pre-processor which is applied
before each sort.
     * @param sorter        the sorter.
     * @param postProcessor an optional pre-processor which is applied
before each sort.
     * @param ts            the array of Ts.
     * @param nRuns         the number of runs to perform in this
benchmark.
     * @param timeLoggers   the time-loggers.
     */
    public SorterBenchmark(Class<T> tClass, UnaryOperator<T[]>
preProcessor, SortWithHelper<T> sorter, Consumer<T[]> postProcessor, T[]
ts, int nRuns, TimeLogger[] timeLoggers) {
        super(sorter.toString(), preProcessor, sorter::mutatingSort,
postProcessor);
        this.sorter = sorter;
        this.tClass = tClass;
        this.ts = ts;
        this.nRuns = nRuns;
        this.timeLoggers = timeLoggers;
    }
    /**
     * Constructor for a SorterBenchmark where we provide the following
parameters:
     * For this form of the constructor, the post-processor always checks
that the sort was successful.
     * @param tClass       the class of T.
```

```
     * @param preProcessor an optional pre-processor which is applied
before each sort.
     * @param sorter       the sorter.
     * @param ts           the array of Ts.
     * @param nRuns        the number of runs to perform in this benchmark.
     * @param timeLoggers  the time-loggers.
     */
    public SorterBenchmark(Class<T> tClass, UnaryOperator<T[]>
preProcessor, SortWithHelper<T> sorter, T[] ts, int nRuns, TimeLogger[]
timeLoggers) {
        this(tClass, preProcessor, sorter, sorter::postProcess, ts, nRuns,
timeLoggers);
    }
    /**
     * Constructor for a SorterBenchmark where we provide the following
parameters:
     * For this form of the constructor, the post-processor always checks
that the sort was successful.
     * For this form of the constructor, there is no pre-processor.
     *
     * @param tClass      the class of T.
     * @param sorter      the sorter.
     * @param ts          the array of Ts.
     * @param nRuns       the number of runs to perform in this benchmark.
     * @param timeLoggers the time-loggers.
     */
    public SorterBenchmark(Class<T> tClass, SortWithHelper<T> sorter, T[]
ts, int nRuns, TimeLogger[] timeLoggers) {
        this(tClass, null, sorter, ts, nRuns, timeLoggers);
    }
    private T[] generateRandomArray(T[] lookupArray) {
        return sorter.getHelper().random(tClass, (r) ->
lookupArray[r.nextInt(lookupArray.length)]);
    }
    protected final SortWithHelper<T> sorter;
    protected final T[] ts;
    protected final int nRuns;
    protected final TimeLogger[] timeLoggers;
    private final static LazyLogger logger = new
LazyLogger(SorterBenchmark.class);
    private final Class<T> tClass;
}
```

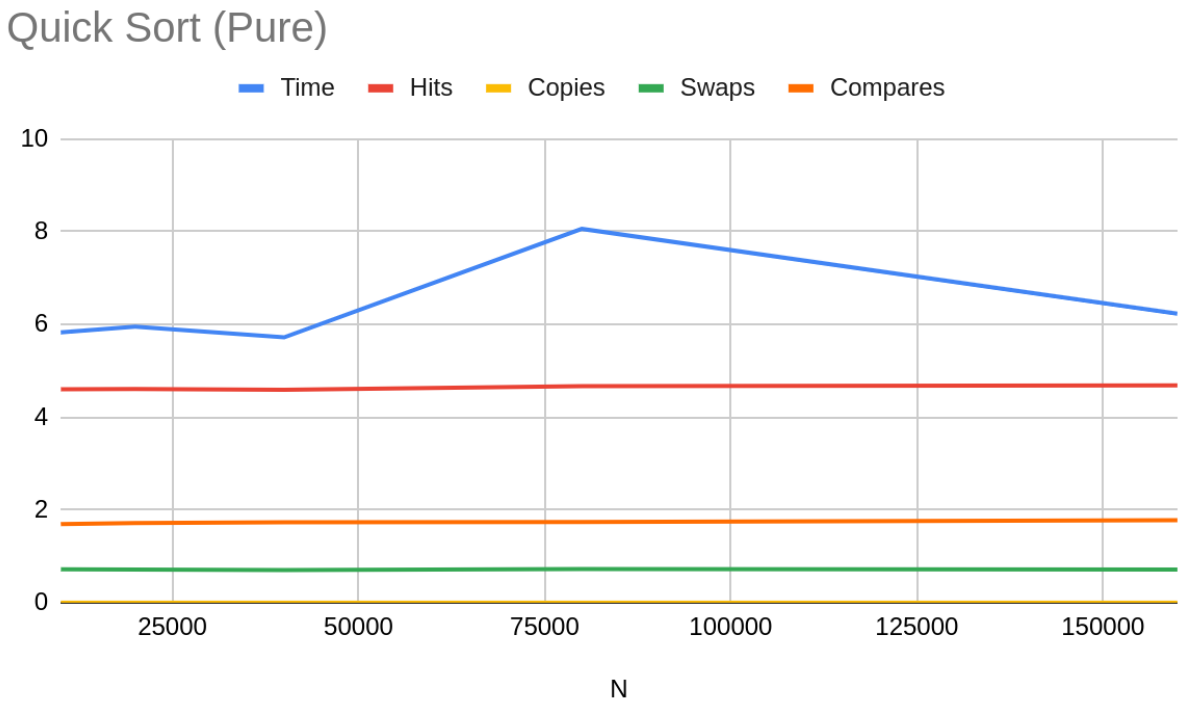
GRAPHICAL REPRESENTATION



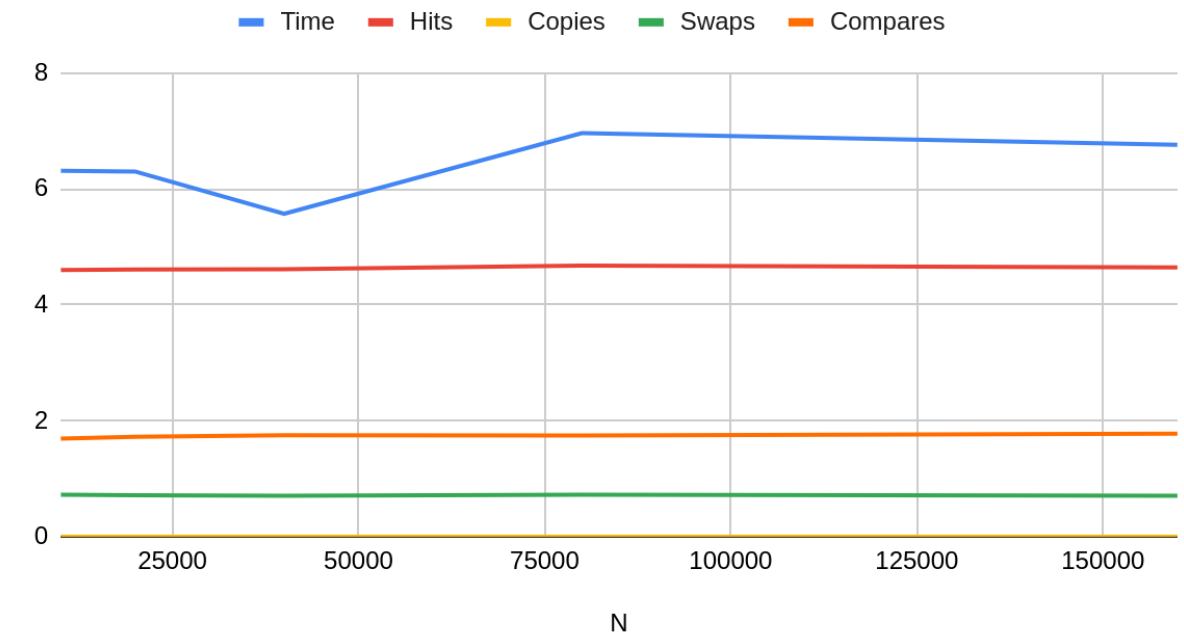
N	Time	Hits	Copies	Swaps	Compares
10000	6.84	5.318	2.389	0.106	1.319
20000	5.94	5.35	2.423	0.099	1.328
40000	7.27	5.377	2.454	0.092	1.335
80000	7.71	5.401	2.48	0.086	1.342
160000	9.6	5.423	2.504	0.081	1.348



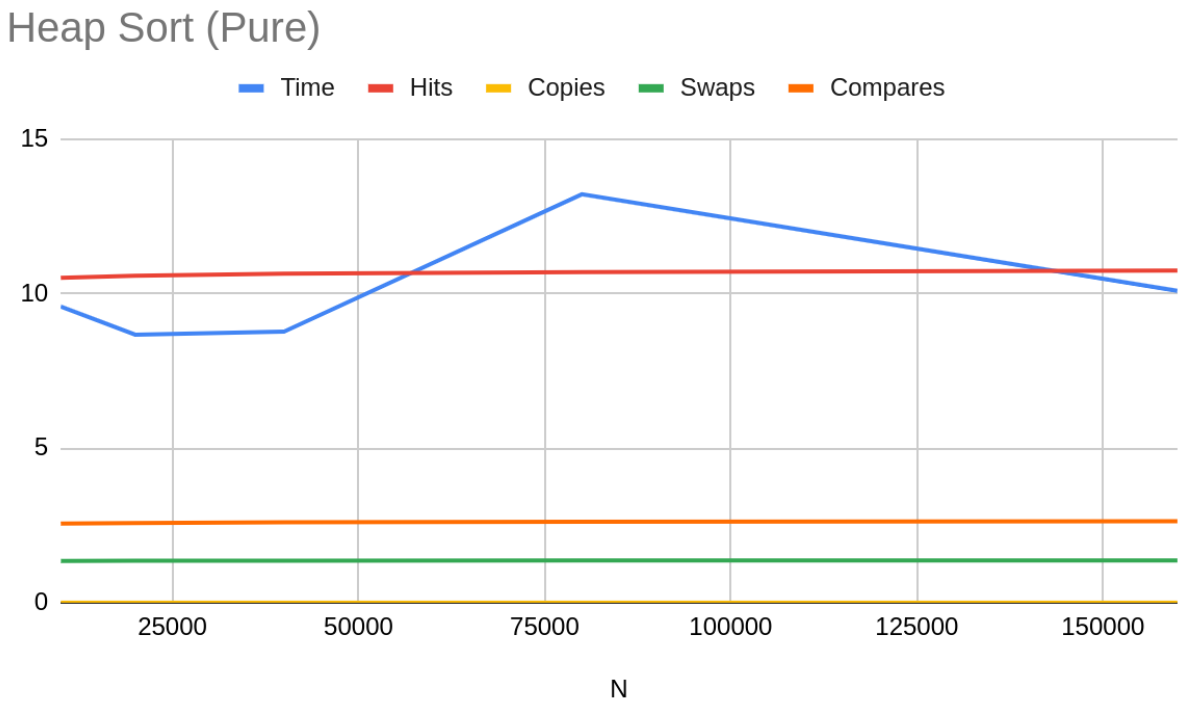
N	Time	Hits	Copies	Swaps	Compares
10000	5.44	5.318	2.389	0.106	1.319
20000	5.99	5.35	2.423	0.099	1.328
40000	7.01	5.377	2.454	0.092	1.335
80000	8.71	5.401	2.48	0.086	1.342
160000	8.72	5.423	2.504	0.081	1.348



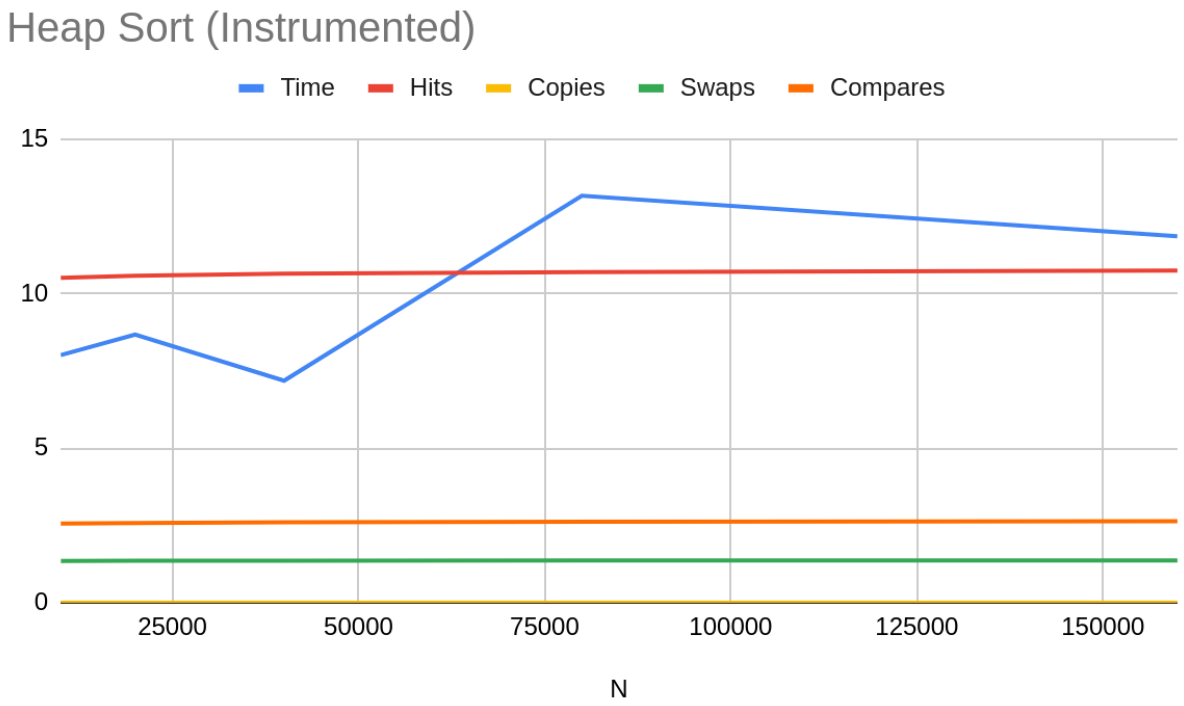
N	Time	Hits	Copies	Swaps	Compares
10000	5.83	4.604	0	0.722	1.697
20000	5.95	4.608	0	0.716	1.717
40000	5.72	4.592	0	0.705	1.734
80000	8.06	4.668	0	0.726	1.739
160000	6.23	4.684	0	0.717	1.779



N	Time	Hits	Copies	Swaps	Compares
10000	6.31	4.601	0	0.722	1.694
20000	6.3	4.608	0	0.715	1.722
40000	5.57	4.613	0	0.706	1.75
80000	6.96	4.675	0	0.726	1.748
160000	6.76	4.644	0	0.708	1.777



N	Time	Hits	Copies	Swaps	Compares
10000	9.58	10.505	0	1.349	2.556
20000	8.67	10.577	0	1.355	2.579
40000	8.77	10.641	0	1.361	2.599
80000	13.21	10.696	0	1.366	2.616
160000	10.09	10.745	0	1.37	2.632



N	Time	Hits	Copies	Swaps	Compares
10000	8.01	10.505	0	1.349	2.556
20000	8.67	10.577	0	1.355	2.579
40000	7.18	10.641	0	1.361	2.599
80000	13.16	10.696	0	1.366	2.616
160000	11.85	10.745	0	1.37	2.632

LOGS

2023-03-12 19:30:13 INFO SortBenchmark - SortBenchmark.main: 1.0.0 (sortbenchmark - instrumentation) with word counts: [10000, 20000, 40000, 80000, 160000]
2023-03-12 19:30:14 INFO Benchmark_Timer - Begin run: intArraysorter with 100 runs
2023-03-12 19:30:15 INFO TimeLogger - Raw time per run (mSec): 6.89
2023-03-12 19:30:15 INFO TimeLogger - Normalized time per run (n log n): .76
2023-03-12 19:30:15 INFO Benchmark_Timer - Begin run: integerArraysorter with 100 runs
2023-03-12 19:30:18 INFO TimeLogger - Raw time per run (mSec): 27.13
2023-03-12 19:30:18 INFO TimeLogger - Normalized time per run (n log n): 2.98
2023-03-12 19:30:18 INFO SortBenchmark - Beginning String sorts
2023-03-12 19:30:19 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:30:19 INFO SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
2023-03-12 19:30:19 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: MergeSort:
2023-03-12 19:30:19 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 10,000 elements with 844 runs
2023-03-12 19:30:24 INFO TimeLogger - Raw time per run (mSec): 4.86
2023-03-12 19:30:24 INFO TimeLogger - Normalized time per run (n log n): 6.84
2023-03-12 19:30:24 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=489,788; stdDev=305, normalized=5.318; copies: 220,000, normalized=2.389; inversions: <unset>; swaps: mean=9,762; stdDev=90, normalized=0.106; fixes: <unset>; compares: mean=121,506; stdDev=82, normalized=1.319}

2023-03-12 19:30:24 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: QuickSort dual pivot
2023-03-12 19:30:24 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 10,000 elements with 844 runs
2023-03-12 19:30:28 INFO TimeLogger - Raw time per run (mSec): 4.14
2023-03-12 19:30:28 INFO TimeLogger - Normalized time per run (n log n): 5.83
2023-03-12 19:30:28 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=424,065; stdDev=19,261, normalized=4.604; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=66,492; stdDev=4,057, normalized=0.722; fixes: <unset>; compares: mean=156,321; stdDev=6,501, normalized=1.697}

2023-03-12 19:30:28 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: Heapsort
2023-03-12 19:30:28 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2023-03-12 19:30:34 INFO TimeLogger - Raw time per run (mSec): 6.81
2023-03-12 19:30:34 INFO TimeLogger - Normalized time per run (n log n): 9.58
2023-03-12 19:30:34 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=967,551; stdDev=488, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,202; stdDev=80, normalized=1.349; fixes: <unset>; compares: mean=235,372; stdDev=96, normalized=2.556}

2023-03-12 19:30:35 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:30:35 INFO SortBenchmark - Testing with 844 runs of sorting 10,000 words and instrumented
2023-03-12 19:30:35 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: MergeSort:
2023-03-12 19:30:35 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 10,000 elements with 844 runs
2023-03-12 19:30:38 INFO TimeLogger - Raw time per run (mSec): 3.87
2023-03-12 19:30:38 INFO TimeLogger - Normalized time per run (n log n): 5.44
2023-03-12 19:30:38 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=489,782; stdDev=302, normalized=5.318; copies: 220,000, normalized=2.389; inversions: <unset>; swaps: mean=9,759; stdDev=89, normalized=0.106; fixes: <unset>; compares: mean=121,506; stdDev=80, normalized=1.319}

2023-03-12 19:30:38 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: QuickSort dual pivot
2023-03-12 19:30:38 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 10,000 elements with 844 runs
2023-03-12 19:30:43 INFO TimeLogger - Raw time per run (mSec): 4.49
2023-03-12 19:30:43 INFO TimeLogger - Normalized time per run (n log n): 6.31
2023-03-12 19:30:43 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=423,748; stdDev=18,734, normalized=4.601; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=66,497; stdDev=4,022, normalized=0.722; fixes: <unset>; compares: mean=155,988; stdDev=6,538, normalized=1.694}

2023-03-12 19:30:43 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: Heapsort
2023-03-12 19:30:43 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2023-03-12 19:30:48 INFO TimeLogger - Raw time per run (mSec): 5.69
2023-03-12 19:30:48 INFO TimeLogger - Normalized time per run (n log n): 8.01
2023-03-12 19:30:48 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=967,557; stdDev=455, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,204; stdDev=75, normalized=1.349; fixes: <unset>; compares: mean=235,371; stdDev=90, normalized=2.556}

2023-03-12 19:30:49 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:30:49 INFO SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words

2023-03-12 19:30:49 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: MergeSort:
2023-03-12 19:30:49 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 20,000 elements with 389 runs
2023-03-12 19:30:53 INFO TimeLogger - Raw time per run (mSec): 9.16
2023-03-12 19:30:53 INFO TimeLogger - Normalized time per run (n log n): 5.94
2023-03-12 19:30:53 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=1,059,577; stdDev=452, normalized=5.350; copies: 480,000, normalized=2.423; inversions: <unset>; swaps: mean=19,520; stdDev=135, normalized=0.099; fixes: <unset>; compares: mean=263,014; stdDev=126, normalized=1.328}

2023-03-12 19:30:53 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: QuickSort dual pivot
2023-03-12 19:30:53 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 20,000 elements with 389 runs
2023-03-12 19:30:57 INFO TimeLogger - Raw time per run (mSec): 9.18
2023-03-12 19:30:57 INFO TimeLogger - Normalized time per run (n log n): 5.95
2023-03-12 19:30:57 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=912,628; stdDev=38,159, normalized=4.608; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=141,852; stdDev=8,169, normalized=0.716; fixes: <unset>; compares: mean=340,125; stdDev=12,942, normalized=1.717}

2023-03-12 19:30:57 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: Heapsort
2023-03-12 19:30:57 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2023-03-12 19:31:03 INFO TimeLogger - Raw time per run (mSec): 13.37
2023-03-12 19:31:03 INFO TimeLogger - Normalized time per run (n log n): 8.67
2023-03-12 19:31:03 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=2,095,060; stdDev=652, normalized=10.577; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,396; stdDev=107, normalized=1.355; fixes: <unset>; compares: mean=510,738; stdDev=129, normalized=2.579}

2023-03-12 19:31:03 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:31:03 INFO SortBenchmark - Testing with 389 runs of sorting 20,000 words and instrumented
2023-03-12 19:31:03 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: MergeSort:
2023-03-12 19:31:03 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 20,000 elements with 389 runs
2023-03-12 19:31:07 INFO TimeLogger - Raw time per run (mSec): 9.24
2023-03-12 19:31:07 INFO TimeLogger - Normalized time per run (n log n): 5.99
2023-03-12 19:31:07 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=1,059,585; stdDev=421, normalized=5.350; copies: 480,000, normalized=2.423; inversions: <unset>; swaps: mean=19,526; stdDev=125, normalized=0.099; fixes: <unset>; compares: mean=263,005; stdDev=117, normalized=1.328}

2023-03-12 19:31:07 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: QuickSort dual pivot
2023-03-12 19:31:07 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 20,000 elements with 389 runs
2023-03-12 19:31:12 INFO TimeLogger - Raw time per run (mSec): 9.71
2023-03-12 19:31:12 INFO TimeLogger - Normalized time per run (n log n): 6.30
2023-03-12 19:31:12 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=912,651; stdDev=36,878, normalized=4.608; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=141,635; stdDev=7,838, normalized=0.715; fixes: <unset>; compares: mean=341,009; stdDev=13,522, normalized=1.722}

2023-03-12 19:31:12 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: Heapsort
2023-03-12 19:31:12 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2023-03-12 19:31:18 INFO TimeLogger - Raw time per run (mSec): 13.36
2023-03-12 19:31:18 INFO TimeLogger - Normalized time per run (n log n): 8.67
2023-03-12 19:31:18 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=2,095,082; stdDev=697, normalized=10.577; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,401; stdDev=115, normalized=1.355; fixes: <unset>; compares: mean=510,738; stdDev=136, normalized=2.579}

2023-03-12 19:31:18 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:31:18 INFO SortBenchmark - Testing pure sorts with 181 runs of sorting 40,000 words
2023-03-12 19:31:18 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: MergeSort:
2023-03-12 19:31:18 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 40,000 elements with 181 runs
2023-03-12 19:31:23 INFO TimeLogger - Raw time per run (mSec): 24.16
2023-03-12 19:31:23 INFO TimeLogger - Normalized time per run (n log n): 7.27
2023-03-12 19:31:23 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=2,279,156; stdDev=603, normalized=5.377; copies: 1,040,000, normalized=2.454; inversions: <unset>; swaps: mean=39,046; stdDev=181, normalized=0.092; fixes: <unset>; compares: mean=566,026; stdDev=160, normalized=1.335}

2023-03-12 19:31:23 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: QuickSort dual pivot
2023-03-12 19:31:23 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 40,000 elements with 181 runs
2023-03-12 19:31:27 INFO TimeLogger - Raw time per run (mSec): 18.99
2023-03-12 19:31:27 INFO TimeLogger - Normalized time per run (n log n): 5.72
2023-03-12 19:31:27 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=1,946,435; stdDev=77,357, normalized=4.592; copies: 0,

normalized=0.000; inversions: <unset>; swaps: mean=298,932; stdDev=16,581, normalized=0.705; fixes: <unset>; compares: mean=734,830; stdDev=24,418, normalized=1.734}

2023-03-12 19:31:27 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: Heapsort
2023-03-12 19:31:27 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 40,000 elements with 181 runs
2023-03-12 19:31:33 INFO TimeLogger - Raw time per run (mSec): 29.15
2023-03-12 19:31:33 INFO TimeLogger - Normalized time per run (n log n): 8.77
2023-03-12 19:31:33 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=4,510,140; stdDev=873, normalized=10.641; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=576,799; stdDev=144, normalized=1.361; fixes: <unset>; compares: mean=1,101,472; stdDev=172, normalized=2.599}

2023-03-12 19:31:33 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-03-12 19:31:33 INFO SortBenchmark - Testing with 181 runs of sorting 40,000 words and instrumented
2023-03-12 19:31:33 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: MergeSort:
2023-03-12 19:31:33 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 40,000 elements with 181 runs
2023-03-12 19:31:38 INFO TimeLogger - Raw time per run (mSec): 23.29
2023-03-12 19:31:38 INFO TimeLogger - Normalized time per run (n log n): 7.01
2023-03-12 19:31:38 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=2,279,216; stdDev=574, normalized=5.377; copies: 1,040,000, normalized=2.454; inversions: <unset>; swaps: mean=39,066; stdDev=169, normalized=0.092; fixes: <unset>; compares: mean=566,030; stdDev=149, normalized=1.335}

2023-03-12 19:31:38 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: QuickSort dual pivot
2023-03-12 19:31:38 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 40,000 elements with 181 runs
2023-03-12 19:31:42 INFO TimeLogger - Raw time per run (mSec): 18.49

2023-03-12 19:31:42 INFO TimeLogger - Normalized time per run (n log n): 5.57
2023-03-12 19:31:42 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=1,955,095; stdDev=79,046, normalized=4.613; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=299,375; stdDev=15,961, normalized=0.706; fixes: <unset>; compares: mean=741,741; stdDev=26,575, normalized=1.750}

2023-03-12 19:31:42 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: Heapsort
2023-03-12 19:31:42 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 40,000 elements with 181 runs
2023-03-12 19:31:47 INFO TimeLogger - Raw time per run (mSec): 23.86
2023-03-12 19:31:47 INFO TimeLogger - Normalized time per run (n log n): 7.18
2023-03-12 19:31:47 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=4,510,223; stdDev=944, normalized=10.641; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=576,806; stdDev=150, normalized=1.361; fixes: <unset>; compares: mean=1,101,500; stdDev=193, normalized=2.599}

2023-03-12 19:31:48 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2023-03-12 19:31:48 INFO SortBenchmark - Testing pure sorts with 84 runs of sorting 80,000 words
2023-03-12 19:31:48 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: MergeSort:
2023-03-12 19:31:48 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 80,000 elements with 84 runs
2023-03-12 19:31:54 INFO TimeLogger - Raw time per run (mSec): 54.90
2023-03-12 19:31:54 INFO TimeLogger - Normalized time per run (n log n): 7.71
2023-03-12 19:31:54 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=4,878,187; stdDev=825, normalized=5.401; copies: 2,240,000, normalized=2.480; inversions: <unset>; swaps: mean=78,062; stdDev=248, normalized=0.086; fixes: <unset>; compares: mean=1,211,999; stdDev=232, normalized=1.342}

2023-03-12 19:31:54 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: QuickSort dual pivot
2023-03-12 19:31:54 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 80,000 elements with 84 runs
2023-03-12 19:32:00 INFO TimeLogger - Raw time per run (mSec): 57.42
2023-03-12 19:32:00 INFO TimeLogger - Normalized time per run (n log n): 8.06
2023-03-12 19:32:00 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=4,215,904; stdDev=146,080, normalized=4.668; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=655,808; stdDev=32,123, normalized=0.726; fixes: <unset>; compares: mean=1,570,856; stdDev=49,616, normalized=1.739}

2023-03-12 19:32:00 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: Heapsort
2023-03-12 19:32:00 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 80,000 elements with 84 runs
2023-03-12 19:32:10 INFO TimeLogger - Raw time per run (mSec): 94.10
2023-03-12 19:32:10 INFO TimeLogger - Normalized time per run (n log n): 13.21
2023-03-12 19:32:10 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=9,660,361; stdDev=1,237, normalized=10.696; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,233,599; stdDev=189, normalized=1.366; fixes: <unset>; compares: mean=2,362,983; stdDev=270, normalized=2.616}

2023-03-12 19:32:11 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2023-03-12 19:32:11 INFO SortBenchmark - Testing with 84 runs of sorting 80,000 words and instrumented
2023-03-12 19:32:11 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: MergeSort:
2023-03-12 19:32:11 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 80,000 elements with 84 runs
2023-03-12 19:32:18 INFO TimeLogger - Raw time per run (mSec): 62.06

2023-03-12 19:32:18 INFO TimeLogger - Normalized time per run (n log n): 8.71
2023-03-12 19:32:18 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=4,878,325; stdDev=862, normalized=5.401; copies: 2,240,000, normalized=2.480; inversions: <unset>; swaps: mean=78,102; stdDev=252, normalized=0.086; fixes: <unset>; compares: mean=1,212,040; stdDev=238, normalized=1.342}

2023-03-12 19:32:18 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: QuickSort dual pivot
2023-03-12 19:32:18 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 80,000 elements with 84 runs
2023-03-12 19:32:23 INFO TimeLogger - Raw time per run (mSec): 49.61
2023-03-12 19:32:23 INFO TimeLogger - Normalized time per run (n log n): 6.96
2023-03-12 19:32:23 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=4,222,208; stdDev=142,611, normalized=4.675; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=655,429; stdDev=29,340, normalized=0.726; fixes: <unset>; compares: mean=1,578,592; stdDev=51,632, normalized=1.748}

2023-03-12 19:32:23 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: Heapsort
2023-03-12 19:32:23 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 80,000 elements with 84 runs
2023-03-12 19:32:32 INFO TimeLogger - Raw time per run (mSec): 93.77
2023-03-12 19:32:32 INFO TimeLogger - Normalized time per run (n log n): 13.16
2023-03-12 19:32:32 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=9,660,211; stdDev=1,243, normalized=10.696; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,233,577; stdDev=196, normalized=1.366; fixes: <unset>; compares: mean=2,362,951; stdDev=268, normalized=2.616}

2023-03-12 19:32:33 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2023-03-12 19:32:33 INFO SortBenchmark - Testing pure sorts with 39 runs of sorting 160,000 words
2023-03-12 19:32:33 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: MergeSort:
2023-03-12 19:32:33 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 160,000 elements with 39 runs
2023-03-12 19:32:41 INFO TimeLogger - Raw time per run (mSec): 146.08
2023-03-12 19:32:41 INFO TimeLogger - Normalized time per run (n log n): 9.60
2023-03-12 19:32:41 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=10,396,449; stdDev=937, normalized=5.423; copies: 4,800,000, normalized=2.504; inversions: <unset>; swaps: mean=156,145; stdDev=268, normalized=0.081; fixes: <unset>; compares: mean=2,583,968; stdDev=328, normalized=1.348}

2023-03-12 19:32:41 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: QuickSort dual pivot
2023-03-12 19:32:41 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 160,000 elements with 39 runs
2023-03-12 19:32:45 INFO TimeLogger - Raw time per run (mSec): 94.79
2023-03-12 19:32:45 INFO TimeLogger - Normalized time per run (n log n): 6.23
2023-03-12 19:32:45 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=8,979,800; stdDev=368,512, normalized=4.684; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,374,876; stdDev=72,051, normalized=0.717; fixes: <unset>; compares: mean=3,411,698; stdDev=111,873, normalized=1.779}

2023-03-12 19:32:45 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: Heapsort
2023-03-12 19:32:45 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 160,000 elements with 39 runs
2023-03-12 19:32:52 INFO TimeLogger - Raw time per run (mSec): 153.51
2023-03-12 19:32:52 INFO TimeLogger - Normalized time per run (n log n): 10.09
2023-03-12 19:32:52 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=20,600,900; stdDev=2,105, normalized=10.745; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=2,627,238; stdDev=344, normalized=1.370; fixes: <unset>; compares: mean=5,045,974; stdDev=390, normalized=2.632}

2023-03-12 19:32:53 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2023-03-12 19:32:54 INFO SortBenchmark - Testing with 39 runs of sorting 160,000 words and instrumented
2023-03-12 19:32:54 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: MergeSort:
2023-03-12 19:32:54 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 160,000 elements with 39 runs
2023-03-12 19:33:00 INFO TimeLogger - Raw time per run (mSec): 132.72
2023-03-12 19:33:00 INFO TimeLogger - Normalized time per run (n log n): 8.72
2023-03-12 19:33:00 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=10,396,714; stdDev=947, normalized=5.423; copies: 4,800,000, normalized=2.504; inversions: <unset>; swaps: mean=156,205; stdDev=280, normalized=0.081; fixes: <unset>; compares: mean=2,584,082; stdDev=284, normalized=1.348}

2023-03-12 19:33:00 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: QuickSort dual pivot
2023-03-12 19:33:00 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 160,000 elements with 39 runs
2023-03-12 19:33:05 INFO TimeLogger - Raw time per run (mSec): 102.82
2023-03-12 19:33:05 INFO TimeLogger - Normalized time per run (n log n): 6.76
2023-03-12 19:33:05 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=8,902,944; stdDev=255,538, normalized=4.644; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,356,636; stdDev=45,753, normalized=0.708; fixes: <unset>; compares: mean=3,407,868; stdDev=118,923, normalized=1.777}

2023-03-12 19:33:05 INFO SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 39 runs using sorter: Heapsort
2023-03-12 19:33:05 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 160,000 elements with 39 runs
2023-03-12 19:33:13 INFO TimeLogger - Raw time per run (mSec): 180.23
2023-03-12 19:33:13 INFO TimeLogger - Normalized time per run (n log n): 11.85
2023-03-12 19:33:13 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=20,600,791; stdDev=1,656, normalized=10.745; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=2,627,198; stdDev=269, normalized=1.370; fixes: <unset>; compares: mean=5,045,998; stdDev=357, normalized=2.632}
2023-03-12 19:33:13 INFO SortBenchmark - Beginning LocalDateTime sorts

UNIT TEST SCREENSHOTS

NOT APPLICABLE