Program Structures and Algorithms Spring 2024

NAME: PRANAV ARUN KAPSE
NUID: 002871241

GitHub Link: https://github.com/kapsep/INFO6205_PSA

TASK: Assignment 6 (Hits as time predictor)
Analysis Report on Hits as time predictor

Aim:

The goal of this task is to run benchmarks for merge sort, quick sort (dual-pivot), and heap sort in order to identify the best predictor of total execution time. The parameters listed below are examined to identify the most accurate predictor.
Hits, Swaps, Copies Compares

Results:

- Merge Sort: Exhibited minimal impact of hits, swaps/copies, and comparisons on overall execution time. Hits were slightly higher than swaps/copies and comparisons.
- Quick Sort (Dual Pivot): Showed significant reliance on hits and comparisons for execution time prediction. Swaps/copies were negligible as this algorithm doesn't involve copy operations.
- Heap Sort: Displayed a strong correlation between hits and execution time, with comparisons also playing a significant role. Swaps/copies had minimal impact due to the nature of the algorithm.

Conclusion:

For comparison-based sorting algorithms such as quick sort, merge sort, and heap sort, the number of array access, or hits, is generally the best predictor of total execution time. Because accessing elements in an array can be a costly and time-

consuming process, array access can affect how quickly sorting algorithms execute. A sorting algorithm's performance can also be affected by how elements in an array are accessed. An algorithm can frequently encounter cache misses, for which the data being accessed is not already in the cache and must be retrieved from main memory, which may be slower than accessing data in the cache if it is necessary to access elements in a random order.

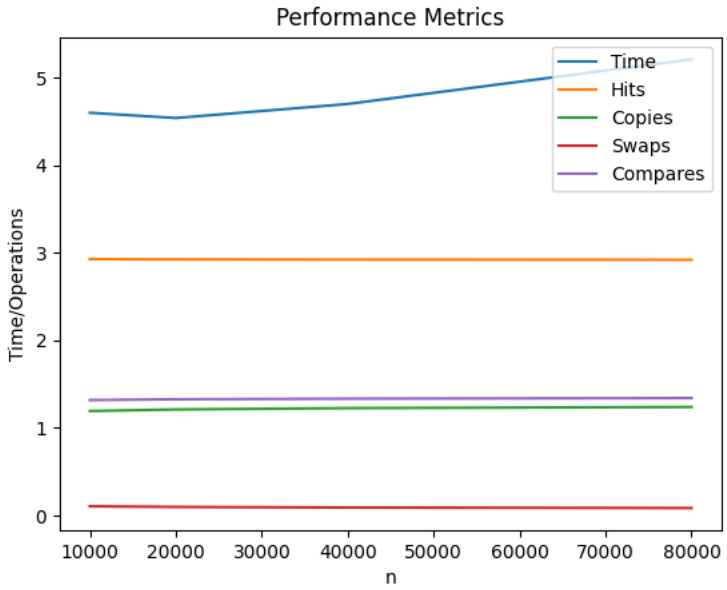
There is no swapping involved in the merge sort process; instead, additional memory is needed for copying values. Pointers in the subarrays are moved via comparisons made while copying values from the auxiliary memory. As a result, Hits, Copies, Compare, and Swap have the lowest effects on the overall time (which is almost zero). But since heap sort and quicksort rely on continuously comparing and swapping elements to create appropriate partitions within the array, they don't require additional memory, so copy operations happen quickly. As a result, Hits, Compare, and Swap have the lowest effects on the overall time. Here, copy has no bearing at all.

The hits graph is also very close to the time graph for all sorting algorithms when t he normalized values of all the metrics are plotted, showing that it is the best pre dictor of total execution time.

- For comparison-based sorting algorithms like quick sort, merge sort, and heap sort, the number of array accesses (hits) emerges as the best predictor of total execution time.
- Hits provide valuable insights into how efficiently elements are accessed within the array, affecting cache utilization and overall performance.
- Comparisons also play a crucial role, particularly in algorithms like quick sort, where partitioning relies heavily on comparison operations.
- Swaps/copies have minimal impact on overall execution time, especially in algorithms like merge sort and heap sort, where swapping or copying operations are minimal or non-existent.

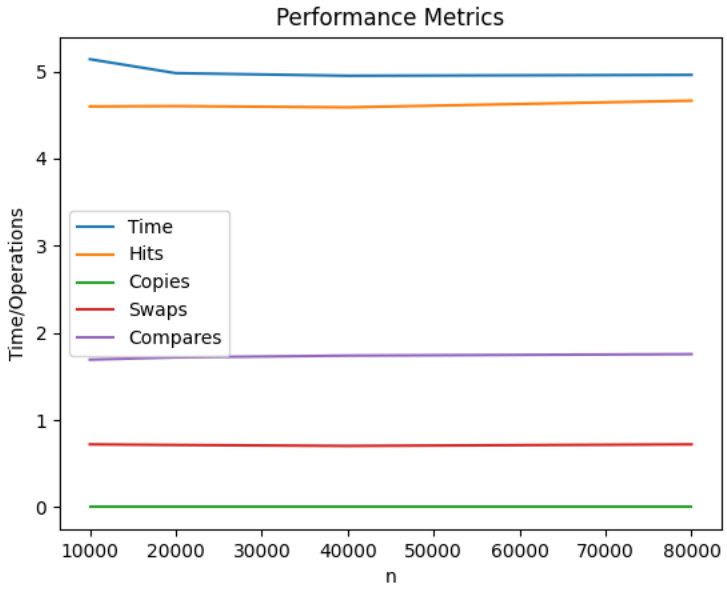
Evidence for the assignment:

Merge Sort (Instrumented)



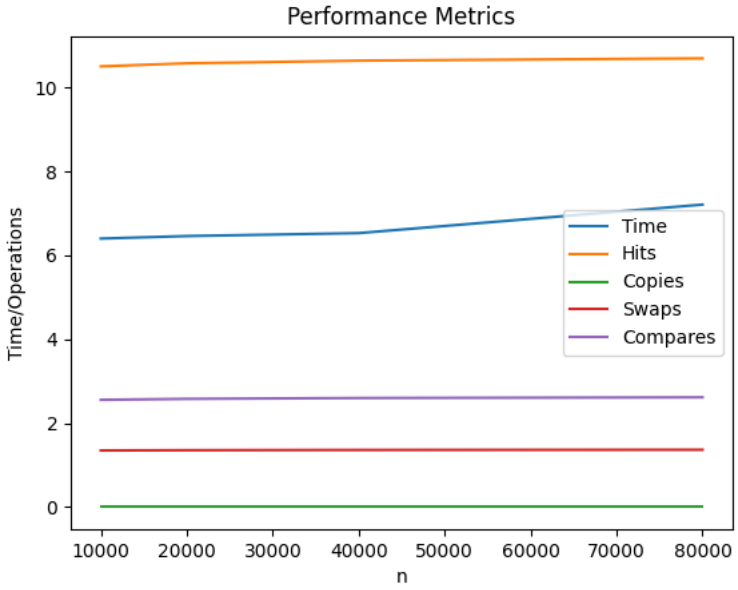
N	Time	Hits	Copies	Swaps	Compares
10000	4.60	2.929	1.194	0.106	1.319
20000	4.54	2.926	1.212	0.099	1.328
40000	4.70	2.923	1.227	0.092	1.335
80000	5.21	2.921	1.240	0.086	1.342

QuickSort dual pivot (Instrumented)



N	Time	Hits	Copies	Swaps	Compares
10000	5.14	4.598	0	0.721	1.693
20000	4.98	4.602	0	0.714	1.719
40000	4.95	4.588	0	0.703	1.739
80000	4.96	4.664	0	0.721	1.756

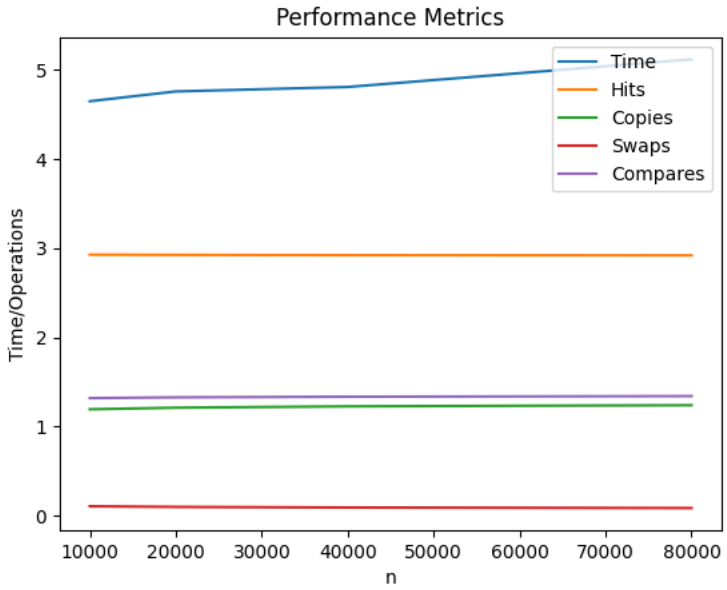
Heap Sort (Instrumented)



N	Time	Hits	Copies	Swaps	Compares

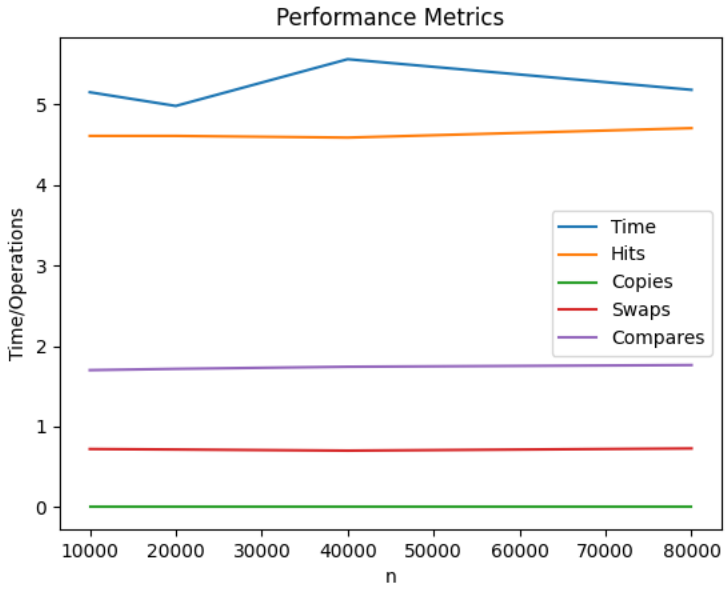
10000	6.40	10.505	0	1.349	2.556
20000	6.46	10.578	0	1.355	2.579
40000	6.53	10.641	0	1.361	2.599
80000	7.21	10.696	0	1.366	2.616

Merge Sort (Normal)



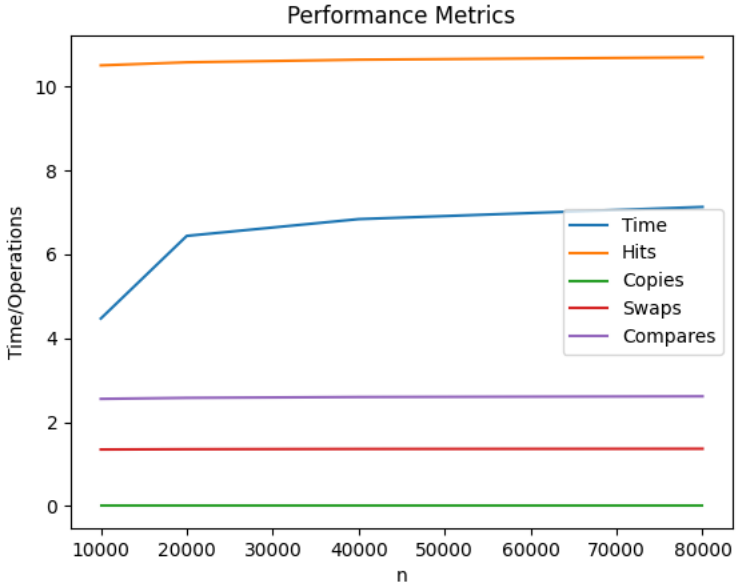
N	Time	Hits	Copies	Swaps	Compares
10000	4.65	2.929	1.194	0.106	1.319
20000	4.76	2.926	1.212	0.099	1.328
40000	4.81	2.923	1.227	0.092	1.335
80000	5.12	2.921	1.240	0.086	1.342

QuickSort dual pivot (Normal)



N	Time	Hits	Copies	Swaps	Compares
10000	5.15	4.608	0	0.722	1.701
20000	4.98	4.608	0	0.716	1.717
40000	5.56	4.587	0	0.702	1.743
80000	5.18	4.704	0	0.729	1.764

Heap Sort (Normal)



N	Time	Hits	Copies	Swaps	Compares
10000	4.47	10.505	0	1.348	2.555

20000	6.44	10.578	0	1.355	2.579
40000	6.84	10.640	0	1.361	2.599
80000	7.13	10.696	0	1.366	2.616

Code:

SortBenchmark.java

```
/*
 (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.generateRandomLocalDateTimeArray;
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;
}
```

Logs:

```
2024-03-15 21:19:19 INFO SortBenchmark - SortBenchmark.main: 1.0.0 (sortbenchmark) with word counts: [10000, 20000, 40000, 80000]
2024-03-15 21:19:19 INFO Benchmark_Timer - Begin run: intArraysorter with 100 runs
2024-03-15 21:19:20 INFO TimeLogger - Raw time per run (mSec): 7.95
2024-03-15 21:19:20 INFO TimeLogger - Normalized time per run (n log n): .87
2024-03-15 21:19:20 INFO Benchmark_Timer - Begin run: integerArraysorter with 100 runs
2024-03-15 21:19:23 INFO TimeLogger - Raw time per run (mSec): 22.42
2024-03-15 21:19:23 INFO TimeLogger - Normalized time per run (n log n): 2.46
2024-03-15 21:19:23 INFO SortBenchmark - Beginning String sorts
2024-03-15 21:19:23 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:19:23 INFO SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
2024-03-15 21:19:23 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:
2024-03-15 21:19:23 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 10,000 elements with 844 runs
2024-03-15 21:19:27 INFO TimeLogger - Raw time per run (mSec): 3.30
2024-03-15 21:19:27 INFO TimeLogger - Normalized time per run (n log n): 4.65
2024-03-15 21:19:27 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=269,795; stdDev=296, normalized=2.929; copies: 110,000, normalized=1.194; inversions: <unset>; swaps: mean=9,763; stdDev=89, normalized=0.106; fixes: <unset>; compares: mean=121,507; stdDev=82, normalized=1.319}
2024-03-15 21:19:27 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
2024-03-15 21:19:27 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 10,000 elements with 844 runs
2024-03-15 21:19:31 INFO TimeLogger - Raw time per run (mSec): 3.66
```

2024-03-15 21:19:31 INFO TimeLogger - Normalized time per run (n log n): 5.15
2024-03-15 21:19:31 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=424,382; stdDev=20,065, normalized=4.608; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=66,491; stdDev=4,243, normalized=0.722; fixes: <unset>; compares: mean=156,647; stdDev=6,675, normalized=1.701}
2024-03-15 21:19:31 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
2024-03-15 21:19:31 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2024-03-15 21:19:35 INFO TimeLogger - Raw time per run (mSec): 4.47
2024-03-15 21:19:35 INFO TimeLogger - Normalized time per run (n log n): 6.30
2024-03-15 21:19:35 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=967,534; stdDev=489, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,200; stdDev=80, normalized=1.348; fixes: <unset>; compares: mean=235,367; stdDev=96, normalized=2.555}
2024-03-15 21:19:35 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:19:35 INFO SortBenchmark - Testing with 844 runs of sorting 10,000 words and instrumented
2024-03-15 21:19:35 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:
2024-03-15 21:19:35 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 10,000 elements with 844 runs
2024-03-15 21:19:39 INFO TimeLogger - Raw time per run (mSec): 3.27
2024-03-15 21:19:39 INFO TimeLogger - Normalized time per run (n log n): 4.60
2024-03-15 21:19:39 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=269,791; stdDev=287, normalized=2.929; copies: 110,000, normalized=1.194; inversions: <unset>; swaps: mean=9,763; stdDev=85, normalized=0.106; fixes: <unset>; compares: mean=121,507; stdDev=77, normalized=1.319}
2024-03-15 21:19:39 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
2024-03-15 21:19:39 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 10,000 elements with 844 runs
2024-03-15 21:19:43 INFO TimeLogger - Raw time per run (mSec): 3.65
2024-03-15 21:19:43 INFO TimeLogger - Normalized time per run (n log n): 5.14
2024-03-15 21:19:43 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=423,469; stdDev=18,937, normalized=4.598; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=66,430; stdDev=4,077, normalized=0.721; fixes: <unset>; compares: mean=155,970; stdDev=6,494, normalized=1.693}
2024-03-15 21:19:43 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
2024-03-15 21:19:43 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2024-03-15 21:19:47 INFO TimeLogger - Raw time per run (mSec): 4.55
2024-03-15 21:19:47 INFO TimeLogger - Normalized time per run (n log n): 6.40
2024-03-15 21:19:47 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits:

mean=967,575; stdDev=472, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,206; stdDev=78, normalized=1.349; fixes: <unset>; compares: mean=235,375; stdDev=94, normalized=2.556}
2024-03-15 21:19:47 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:19:47 INFO SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
2024-03-15 21:19:47 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:
2024-03-15 21:19:47 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 20,000 elements with 389 runs
2024-03-15 21:19:51 INFO TimeLogger - Raw time per run (mSec): 7.34
2024-03-15 21:19:51 INFO TimeLogger - Normalized time per run (n log n): 4.76
2024-03-15 21:19:51 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=579,557; stdDev=422, normalized=2.926; copies: 240,000, normalized=1.212; inversions: <unset>; swaps: mean=19,518; stdDev=125, normalized=0.099; fixes: <unset>; compares: mean=263,017; stdDev=116, normalized=1.328}
2024-03-15 21:19:51 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
2024-03-15 21:19:51 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 20,000 elements with 389 runs
2024-03-15 21:19:55 INFO TimeLogger - Raw time per run (mSec): 7.68
2024-03-15 21:19:55 INFO TimeLogger - Normalized time per run (n log n): 4.98
2024-03-15 21:19:55 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=912,702; stdDev=38,926, normalized=4.608; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=141,885; stdDev=8,227, normalized=0.716; fixes: <unset>; compares: mean=340,084; stdDev=13,306, normalized=1.717}
2024-03-15 21:19:55 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
2024-03-15 21:19:55 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2024-03-15 21:19:59 INFO TimeLogger - Raw time per run (mSec): 9.93
2024-03-15 21:19:59 INFO TimeLogger - Normalized time per run (n log n): 6.44
2024-03-15 21:19:59 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=2,095,113; stdDev=666, normalized=10.578; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,404; stdDev=110, normalized=1.355; fixes: <unset>; compares: mean=510,749; stdDev=130, normalized=2.579}
2024-03-15 21:19:59 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:19:59 INFO SortBenchmark - Testing with 389 runs of sorting 20,000 words and instrumented
2024-03-15 21:19:59 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:
2024-03-15 21:19:59 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 20,000 elements with 389 runs
2024-03-15 21:20:03 INFO TimeLogger - Raw time per run (mSec): 7.00

2024-03-15 21:20:03 INFO TimeLogger - Normalized time per run (n log n): 4.54
2024-03-15 21:20:03 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=579,574; stdDev=423, normalized=2.926; copies: 240,000, normalized=1.212; inversions: <unset>; swaps: mean=19,523; stdDev=124, normalized=0.099; fixes: <unset>; compares: mean=263,005; stdDev=117, normalized=1.328}
2024-03-15 21:20:03 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: QuickSort dual pivot
2024-03-15 21:20:03 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 20,000 elements with 389 runs
2024-03-15 21:20:06 INFO TimeLogger - Raw time per run (mSec): 7.67
2024-03-15 21:20:06 INFO TimeLogger - Normalized time per run (n log n): 4.98
2024-03-15 21:20:06 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=911,553; stdDev=36,117, normalized=4.602; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=141,499; stdDev=7,852, normalized=0.714; fixes: <unset>; compares: mean=340,463; stdDev=12,389, normalized=1.719}
2024-03-15 21:20:06 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
2024-03-15 21:20:06 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2024-03-15 21:20:11 INFO TimeLogger - Raw time per run (mSec): 9.95
2024-03-15 21:20:11 INFO TimeLogger - Normalized time per run (n log n): 6.46
2024-03-15 21:20:11 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=2,095,111; stdDev=686, normalized=10.578; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,404; stdDev=113, normalized=1.355; fixes: <unset>; compares: mean=510,748; stdDev=136, normalized=2.579}
2024-03-15 21:20:11 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:20:11 INFO SortBenchmark - Testing pure sorts with 181 runs of sorting 40,000 words
2024-03-15 21:20:11 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:
2024-03-15 21:20:11 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 40,000 elements with 181 runs
2024-03-15 21:20:14 INFO TimeLogger - Raw time per run (mSec): 16.00
2024-03-15 21:20:14 INFO TimeLogger - Normalized time per run (n log n): 4.81
2024-03-15 21:20:14 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=1,239,078; stdDev=582, normalized=2.923; copies: 520,000, normalized=1.227; inversions: <unset>; swaps: mean=39,026; stdDev=170, normalized=0.092; fixes: <unset>; compares: mean=566,006; stdDev=165, normalized=1.335}
2024-03-15 21:20:14 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
2024-03-15 21:20:14 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 40,000 elements with 181 runs
2024-03-15 21:20:19 INFO TimeLogger - Raw time per run (mSec): 18.48
2024-03-15 21:20:19 INFO TimeLogger - Normalized time per run (n log n): 5.56
2024-03-15 21:20:19 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack

{hits: mean=1,944,403; stdDev=75,085, normalized=4.587; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=297,380; stdDev=16,007, normalized=0.702; fixes: <unset>; compares: mean=739,001; stdDev=27,618, normalized=1.743}
2024-03-15 21:20:19 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
2024-03-15 21:20:19 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 40,000 elements with 181 runs
2024-03-15 21:20:23 INFO TimeLogger - Raw time per run (mSec): 22.74
2024-03-15 21:20:23 INFO TimeLogger - Normalized time per run (n log n): 6.84
2024-03-15 21:20:23 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=4,510,139; stdDev=854, normalized=10.640; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=576,796; stdDev=140, normalized=1.361; fixes: <unset>; compares: mean=1,101,478; stdDev=172, normalized=2.599}
2024-03-15 21:20:23 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2024-03-15 21:20:23 INFO SortBenchmark - Testing with 181 runs of sorting 40,000 words and instrumented
2024-03-15 21:20:23 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: MergeSort:
2024-03-15 21:20:23 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 40,000 elements with 181 runs
2024-03-15 21:20:27 INFO TimeLogger - Raw time per run (mSec): 15.62
2024-03-15 21:20:27 INFO TimeLogger - Normalized time per run (n log n): 4.70
2024-03-15 21:20:27 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=1,239,104; stdDev=629, normalized=2.923; copies: 520,000, normalized=1.227; inversions: <unset>; swaps: mean=39,031; stdDev=185, normalized=0.092; fixes: <unset>; compares: mean=566,009; stdDev=180, normalized=1.335}
2024-03-15 21:20:27 INFO SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 181 runs using sorter: QuickSort dual pivot
2024-03-15 21:20:27 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 40,000 elements with 181 runs
2024-03-15 21:20:31 INFO TimeLogger - Raw time per run (mSec): 16.46
2024-03-15 21:20:31 INFO TimeLogger - Normalized time per run (n log n): 4.95
2024-03-15 21:20:31 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=1,944,625; stdDev=67,961, normalized=4.588; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=297,865; stdDev=14,722, normalized=0.703; fixes: <unset>; compares: mean=737,306; stdDev=27,536, normalized=1.739}
2024-03-15 21:20:31 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
2024-03-15 21:20:31 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 40,000 elements with 181 runs
2024-03-15 21:20:35 INFO TimeLogger - Raw time per run (mSec): 21.71
2024-03-15 21:20:35 INFO TimeLogger - Normalized time per run (n log n): 6.53
2024-03-15 21:20:35 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=4,510,190; stdDev=988, normalized=10.641; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=576,800; stdDev=160, normalized=1.361; fixes: <unset>; compares:

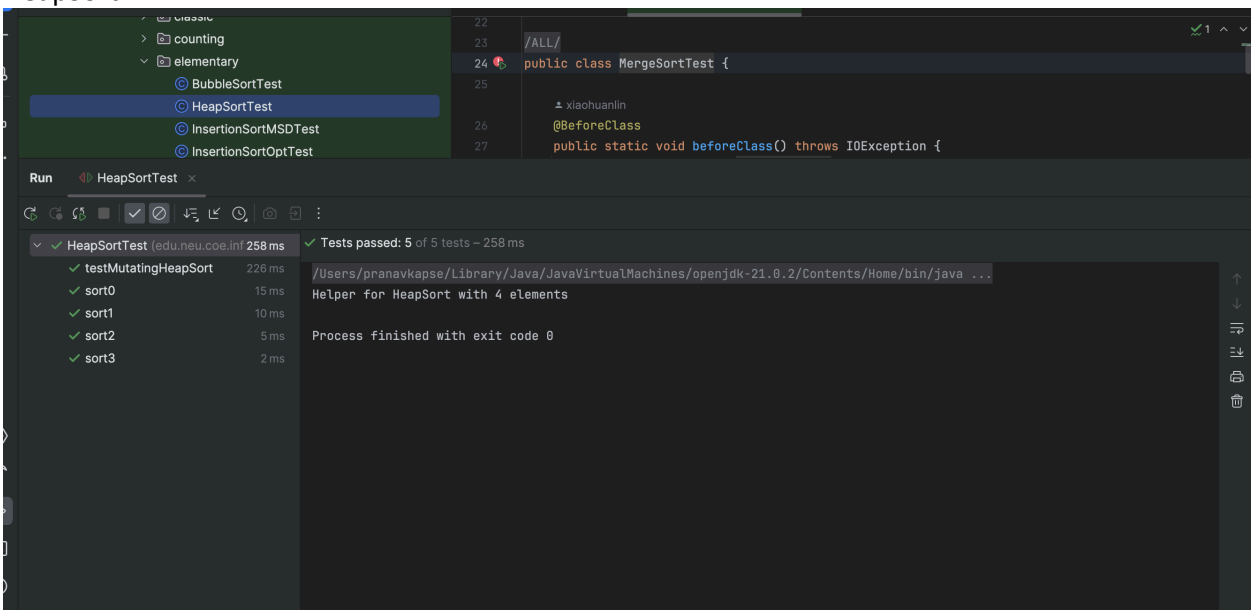
mean=1,101,494; stdDev=198, normalized=2.599}
2024-03-15 21:20:36 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2024-03-15 21:20:36 INFO SortBenchmark - Testing pure sorts with 84 runs of sorting 80,000 words
2024-03-15 21:20:36 INFO SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.String from 81,546 total elements and 84 runs using sorter: MergeSort:
2024-03-15 21:20:36 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 80,000 elements with 84 runs
2024-03-15 21:20:40 INFO TimeLogger - Raw time per run (mSec): 36.51
2024-03-15 21:20:40 INFO TimeLogger - Normalized time per run (n log n): 5.12
2024-03-15 21:20:40 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=2,638,363; stdDev=764, normalized=2.921; copies: 1,120,000, normalized=1.240; inversions: <unset>; swaps: mean=78,102; stdDev=228, normalized=0.086; fixes: <unset>; compares: mean=1,212,067; stdDev=215, normalized=1.342}
2024-03-15 21:20:40 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
2024-03-15 21:20:40 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 80,000 elements with 84 runs
2024-03-15 21:20:44 INFO TimeLogger - Raw time per run (mSec): 36.91
2024-03-15 21:20:44 INFO TimeLogger - Normalized time per run (n log n): 5.18
2024-03-15 21:20:44 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=4,248,533; stdDev=167,333, normalized=4.704; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=658,432; stdDev=36,467, normalized=0.729; fixes: <unset>; compares: mean=1,593,045; stdDev=49,565, normalized=1.764}
2024-03-15 21:20:44 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
2024-03-15 21:20:44 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 80,000 elements with 84 runs
2024-03-15 21:20:49 INFO TimeLogger - Raw time per run (mSec): 50.78
2024-03-15 21:20:49 INFO TimeLogger - Normalized time per run (n log n): 7.13
2024-03-15 21:20:50 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=9,660,337; stdDev=1,162, normalized=10.696; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,233,599; stdDev=190, normalized=1.366; fixes: <unset>; compares: mean=2,362,970; stdDev=233, normalized=2.616}
2024-03-15 21:20:50 INFO SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.txt
2024-03-15 21:20:50 INFO SortBenchmark - Testing with 84 runs of sorting 80,000 words and instrumented
2024-03-15 21:20:50 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:
2024-03-15 21:20:50 INFO Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with 80,000 elements with 84 runs
2024-03-15 21:20:54 INFO TimeLogger - Raw time per run (mSec): 37.10
2024-03-15 21:20:54 INFO TimeLogger - Normalized time per run (n log n): 5.21
2024-03-15 21:20:54 INFO SorterBenchmark - Instrumentation:::: MergeSort:: StatPack {hits: mean=2,638,261; stdDev=785, normalized=2.921; copies: 1,120,000, normalized=1.240; inversions:

<unset>; swaps: mean=78,082; stdDev=234, normalized=0.086; fixes: <unset>; compares: mean=1,211,998; stdDev=209, normalized=1.342}
2024-03-15 21:20:54 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
2024-03-15 21:20:54 INFO Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 80,000 elements with 84 runs
2024-03-15 21:20:58 INFO TimeLogger - Raw time per run (mSec): 35.33
2024-03-15 21:20:58 INFO TimeLogger - Normalized time per run (n log n): 4.96
2024-03-15 21:20:58 INFO SorterBenchmark - Instrumentation:::: QuickSort dual pivot: StatPack {hits: mean=4,212,046; stdDev=174,116, normalized=4.664; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=651,131; stdDev=34,541, normalized=0.721; fixes: <unset>; compares: mean=1,585,682; stdDev=54,498, normalized=1.756}
2024-03-15 21:20:58 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
2024-03-15 21:20:58 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 80,000 elements with 84 runs
2024-03-15 21:21:04 INFO TimeLogger - Raw time per run (mSec): 51.34
2024-03-15 21:21:04 INFO TimeLogger - Normalized time per run (n log n): 7.21
2024-03-15 21:21:04 INFO SorterBenchmark - Instrumentation:::: Heapsort: StatPack {hits: mean=9,660,512; stdDev=1,222, normalized=10.696; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=1,233,631; stdDev=207, normalized=1.366; fixes: <unset>; compares: mean=2,362,993; stdDev=233, normalized=2.616}
2024-03-15 21:21:04 INFO SortBenchmark - Beginning LocalDateTime sorts

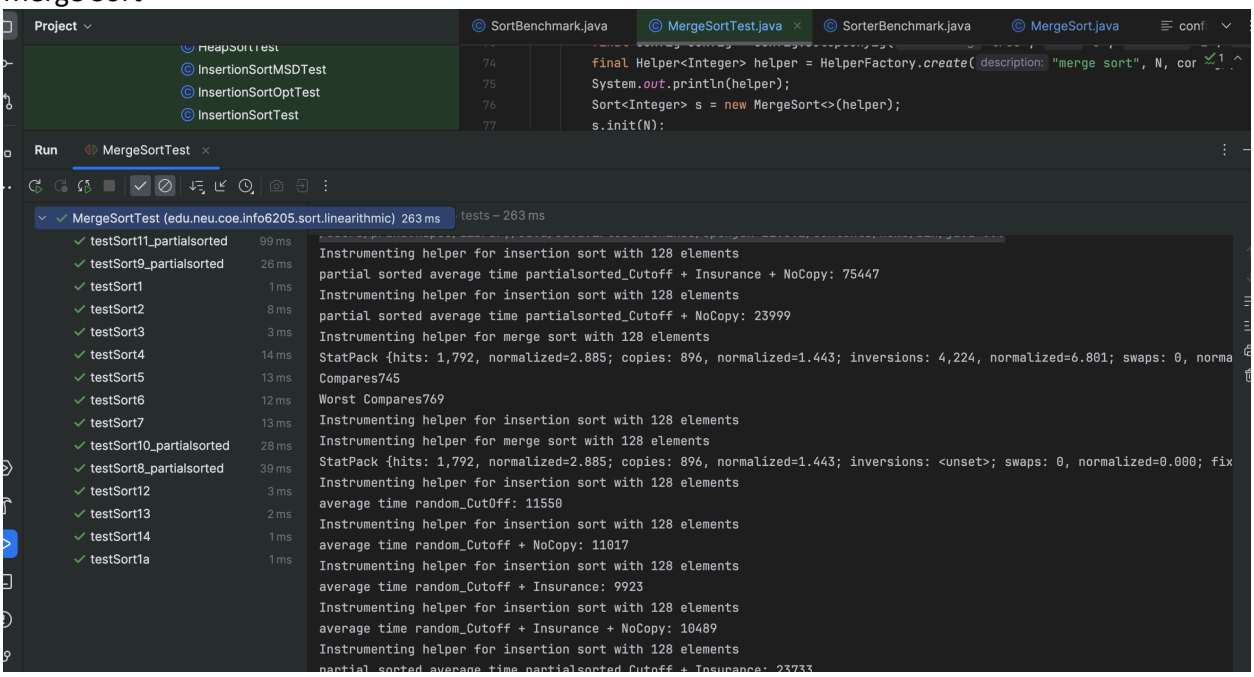
Process finished with exit code 0

Test cases:

HeapSort



Merge Sort



QuickSort Dual Pivot