```java
package edu.neu.coe.info6205.sort.linearithmic;

import edu.neu.coe.info6205.sort.Helper;
import edu.neu.coe.info6205.sort.InstrumentedHelper;
import edu.neu.coe.info6205.sort.SortWithHelper;
import edu.neu.coe.info6205.sort.elementary.InsertionSort;
import edu.neu.coe.info6205.util.Benchmark;
import edu.neu.coe.info6205.util.Benchmark_Timer;
import edu.neu.coe.info6205.util.Config;

import java.util.Arrays;

public class MergeSort<X extends Comparable<X>> extends SortWithHelper<X> {

    public static final String DESCRIPTION = "MergeSort";

    /**
     * Constructor for MergeSort
     * <p>
     * NOTE this is used only by unit tests, using its own instrumented helper.
     *
     * @param helper an explicit instance of Helper to be used.
     */
    public MergeSort(Helper<X> helper) {
        super(helper);
        insertionSort = new InsertionSort<>(helper);
    }

    /**
     * Constructor for MergeSort
     *
     * @param N      the number elements we expect to sort.
     * @param config the configuration.
```

```
34       */
35      public MergeSort(int N, Config config) {
36          super(DESCRIPTION + ":" + getConfigString(
    config), N, config);
37          insertionSort = new InsertionSort<>(
    getHelper());
38      }
39
40      @Override
41      public X[] sort(X[] xs, boolean makeCopy) {
42          getHelper().init(xs.length);
43          X[] result = makeCopy ? Arrays.copyOf(xs,
    xs.length) : xs;
44          sort(result, 0, result.length);
45          return result;
46      }
47
48      @Override
49      public void sort(X[] a, int from, int to) {
50          // CONSIDER don't copy but just allocate
    according to the xs/aux interchange optimization
51          X[] aux = Arrays.copyOf(a, a.length);
52          sort(a, aux, from, to);
53      }
54
55      private void sort(X[] a, X[] aux, int from, int
     to) {
56          final Helper<X> helper = getHelper();
57          Config config = helper.getConfig();
58          boolean insurance = config.getBoolean(
    MERGESORT, INSURANCE);
59          boolean noCopy = config.getBoolean(
    MERGESORT, NOCOPY);
60          if (to <= from + helper.cutoff()) {
61              insertionSort.sort(a, from, to);
62              return;
63          } else {
64              int mid = from+(to-from)/2;
65              sort(a, aux, from, mid);
66              sort(a, aux, mid, to);
67              merge(a, aux, from, mid, to);
```

```
68          }
69
70          // FIXME : implement merge sort with
    insurance and no-copy optimizations
71          // END
72      }
73
74      // CONSIDER combine with MergeSortBasic
    perhaps.
75      private void merge(X[] sorted, X[] result, int
     from, int mid, int to) {
76          final Helper<X> helper = getHelper();
77          int i = from;
78          int j = mid;
79          for (int k = from; k < to; k++)
80              if (i >= mid) helper.copy(sorted, j
    ++, result, k);
81              else if (j >= to) helper.copy(sorted,
    i++, result, k);
82              else if (helper.less(sorted[j], sorted
    [i])) {
83                  helper.incrementFixes(mid - i);
84                  helper.copy(sorted, j++, result, k
    );
85              } else helper.copy(sorted, i++, result
    , k);
86      }
87
88      public static final String MERGESORT = "
    mergesort";
89      public static final String NOCOPY = "nocopy";
90      public static final String INSURANCE = "
    insurance";
91
92      private static String getConfigString(Config
    config) {
93          StringBuilder stringBuilder = new
    StringBuilder();
94          if (config.getBoolean(MERGESORT, INSURANCE
    )) stringBuilder.append(" with insurance
    comparison");
```

```
 95          if (config.getBoolean(MERGESORT, NOCOPY))
    stringBuilder.append(" with no copy");
 96          return stringBuilder.toString();
 97      }
 98
 99      private final InsertionSort<X> insertionSort;
100
101      public static void main (String[] args) {
102          int N = 1000;
103
104          while(N<=64000) {
105              InstrumentedHelper<Integer>
    instrumentedHelper = new InstrumentedHelper<>("
    MergeSort", Config.setupConfig("true", "0", "0",
    "", ""));
106              MergeSort<Integer> s = new MergeSort
    <>(instrumentedHelper);
107              int j = N;
108              s.init(j);
109              Integer[] xs = instrumentedHelper.
    random(Integer.class, r -> r.nextInt(j));
110              Benchmark<Boolean> benchmark = new
    Benchmark_Timer<>("Sorting", b -> s.sort(xs, 0, j
    ));
111              double nTime = benchmark.run(true, 20
    );
112              long nCompares = instrumentedHelper.
    getCompares();
113              long nSwaps = instrumentedHelper.
    getSwaps();
114              long nHits = instrumentedHelper.
    getHits();
115
116              System.out.println("When array size is
    : " + j);
117              System.out.println("Compares: " +
    nCompares);
118              System.out.println("Swaps: " + nSwaps
     );
119              System.out.println("Hits: " + nHits);
120              System.out.println("Time: " + nTime);
```

```
121
122                 System.out.println("\nFor referencs:\t
    " + j + "\t" + nCompares + "\t" + nSwaps + "\t" +
    nHits + "\t" + nTime + "\n");
123
124                 N = N*2;
125             }
126         }
127 }
128
129
```