

# INFO6205 - Sort Algorithm Time and Instruments

In this assignment, I will create some drivers to discover the relationship between instruments like Compares, Swaps/Copies, Inversions, and Execution time.

Here is the screenshot of “MergeSortTest.java”

The screenshot shows an IDE with the following components:

- Project Explorer:** A tree view on the left showing the project structure. The path is `src/test/java/edu/neu/coe/info6205/sort`. The file `MergeSortTest.java` is selected.
- Code Editor:** The main window displays the code for `MergeSortTest.java`. It includes imports for JUnit, a `@BeforeClass` method to load configuration, and a `@Test` method `testSort1()` that sets up an array `xs` and calls `s.sort(xs)` using a `MergeSort` instance.
- Run Console:** The bottom panel shows the output of running the tests. It lists 15 tests passed in 415ms. The output includes detailed statistics for each test, such as execution time, number of comparisons, swaps, and fixes.

```
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

//ALL
public class MergeSortTest {

    @BeforeClass
    public static void beforeClass() throws IOException {
        config = Config.load(MergeSortTest.class);
    }

    @Test
    public void testSort1() throws Exception {
        Integer[] xs = new Integer[4];
        xs[0] = 3;
        xs[1] = 4;
        xs[2] = 2;
        xs[3] = 1;

        // NOTE: first we ensure that there is no cutoff to insertion sort going on.
        final Config config = Config.setupConfig( Instrumenting: "true", seed: "", inversions: "0", cutoff: "1", interinversions: "" );
        GenericSort<Integer> s = new MergeSort<>(xs.length, config);
        Integer[] ys = s.sort(xs);
        assertEquals(Integer.valueOf( 1 ), ys[0]);
    }
}
```

Run: MergeSortTest (edu.neu.coe.info6205 415ms)

Tests passed: 15 of 15 tests - 415ms

Instrumenting helper for insertion sort with 128 elements  
partial sorted average time partialsorted.Cutoff + Insurance + NoCopy: 101321  
Instrumenting helper for insertion sort with 128 elements  
partial sorted average time partialsorted.Cutoff + NoCopy: 123838  
Instrumenting helper for merge sort with 128 elements  
StatPack (hits: 1,790, normalized=2.882; copies: 640, normalized=1.030; inversions: 4,224, normalized=6.801; swaps: 101, normalized=0.163; fixes: 4,224, normalized=6.801; compares: 751)  
Worst Compares: 769  
Instrumenting helper for insertion sort with 128 elements  
Instrumenting helper for merge sort with 128 elements  
StatPack (hits: 1,792, normalized=2.885; copies: 896, normalized=1.443; inversions: <unset>; swaps: 0, normalized=0.000; fixes: 0, normalized=0.000; compare  
average time random.Cutoff: 58837  
Instrumenting helper for insertion sort with 128 elements

I created 3 drivers for MergeSort, HeapSort and QuickSort Dual Pivots to get the number of execution time and instruments. All drivers are store in `“src/test/java/edu/neu/coe/info6205/sort”`.

Here is the screenshot of “MergeSortDriver.java”.

```
17 public class MergeSortDriver {
18     @BeforeClass
19     public static void beforeClass() throws IOException {
20         config = Config.load(MergeSortDriver.class);
21     }
22
23     @Test
24     public void instrumentsTest() {
25         System.out.println("----- MergeSort Instrumentation -----");
26
27         Config config1 = config.copy(sectionName: "helper", optionName: "instrument", value: "true");
28         Config config2 = config1.copy(MergeSort.MERGESORT, MergeSort.INSURANCE, value: "true");
29         Config config3 = config2.copy(MergeSort.MERGESORT, MergeSort.NOCOPY, value: "true");
30
31         for(int n = 10000; n <= 256000; n *= 2) {
32             System.out.println("MergeSort instrument variables of " + n + "-sized array");
33
34             final Helper<Integer> helper = HelperFactory.create(description: "MergeSort", n, config3);
35             Sort<Integer> s = new MergeSort<>(helper);
36             s.init(n);
37             int finalN = n;
38             final Integer[] xs = helper.random(Integer.class, n -> n * nextInt(finalN));
```

Run: MergeSortDriver

Tests passed: 2 of 2 tests - 11 sec 862 ms

----- MergeSort Execution Time -----

2023-03-08 17:44:17 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 10000 - 4.55774675ms

2023-03-08 17:44:18 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 20000 - 2.9740625ms

2023-03-08 17:44:18 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 40000 - 5.433757470000001ms

2023-03-08 17:44:19 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 80000 - 11.6217228ms

2023-03-08 17:44:20 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 160000 - 27.93994952ms

2023-03-08 17:44:23 INFO Benchmark\_Timer - Begin run: MergeSort with 100 runs

Array size: 256000 - 54.997912539999994ms

(src/test/java/edu/neu/coe/info6205/sort/linearithmic/MergeSortDriver.java)

Here is the screenshot of “HeapSortDriver.java”

```
13 public class HeapSortDriver {
14     @Test
15     public void instrumentsTest() {
16         System.out.println("----- HeapSort Instrument Variables -----");
17
18         final Config config = Config.setupConfig(instrumenting: "true", seed: "0", inversions: "1", cutoff: "", interinversions: "");
19
20         for(int n = 10000; n <= 256000; n *= 2) {
21             System.out.println("HeapSort instrument variables of " + n + "-sized array");
22
23             Helper<Integer> helper = HelperFactory.create(description: "HeapSort", n, config);
24             helper.init(n);
25             final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
26             final StatPack statPack = (StatPack) privateMethodTester.invokePrivate(name: "getStatPack");
27             Integer[] xs = arrayCreate(n);
28             SortWithHelper<Integer> sorter = new HeapSort<Integer>(helper);
29             sorter.preProcess(xs);
30             Integer[] ys = sorter.sort(xs);
31             sorter.postProcess(ys);
32
33             final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
34             final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
35             final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
```

Run: HeapSortDriver

Tests passed: 2 of 2 tests - 4 min 11 sec

----- HeapSort Execution Time -----

2023-03-08 17:44:55 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 10000 - 1.62686881ms

2023-03-08 17:44:56 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 20000 - 2.97873586ms

2023-03-08 17:44:56 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 40000 - 6.41209746ms

2023-03-08 17:44:57 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 80000 - 15.034961699999998ms

2023-03-08 17:44:58 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 160000 - 31.97837673ms

2023-03-08 17:45:02 INFO Benchmark\_Timer - Begin run: HeapSort with 100 runs

Array size: 256000 - 54.70560214ms

(src/test/java/edu/neu/coe/info6205/sort/elementary/HeapSortDriver.java)

Here is the screenshot of “QuickSortDualPivotsDriver.java”

```

src/main/java/edu/neu/coe/info6205/sort/linearithmic/QuickSortDualPivotDriver.java
src/test/java/edu/neu/coe/info6205/sort/linearithmic/QuickSortDualPivotDriver.java

@Test
public void timeTest() {
    System.out.println("----- QuickSort Execution Time -----");
    for(int n = 10000, m = 100; n <= 256000; n *= 2) {
        Benchmark_Timer<Integer> bm = new Benchmark_Timer<>{
            description: "QuickSort Dual Pivots",
            print: null,
            (Integer N) -> {
                SortableHelper<Integer> sorter = new QuickSort_DualPivot<>(N, config);
                Helper<Integer> helper = sorter.getHelper();
                Integer[] arr = helper.random(Integer.class, r -> r.nextInt(N));
                Integer[] sorted = sorter.sort(arr);
                // "helper.sorted(array)" used to check the array is ordered or not;
                assertTrue(helper.sorted(sorted));
            },
            print: null
        };
        double x = bm.run(n, m);
        System.out.println("Array size: " + n + " - " + x + "ms.");
        if(n == 160000) n = 128000;
    }
}

```

Run: QuickSortDualPivotDriver

Tests passed: 2 of 2 tests - 1min 48 sec

```

/Users/dengyan/Library/Java/JavaVirtualMachines/corretto-11.0.18/Contents/Home/bin/java ...
QuickSortDualPivotDriver (edu.neu.coe.info6205.sort.linearithmic)
timeTest 7sec 988ms
instrumentsTest 1min 40sec
----- QuickSort Execution Time -----
2023-03-08 17:49:40 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 10000 - 2.36828289ms.
2023-03-08 17:49:41 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 20000 - 2.07852364ms.
2023-03-08 17:49:41 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 40000 - 4.36964827ms.
2023-03-08 17:49:41 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 80000 - 9.67583871ms.
2023-03-08 17:49:42 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 160000 - 22.238766249999998ms.
2023-03-08 17:49:45 INFO Benchmark_Timer - Begin run: QuickSort Dual Pivots with 100 runs
Array size: 256000 - 34.27388884ms.
----- QuickSort Dual Pivots Instrument Variables -----

```

(src/test/java/edu/neu/coe/info6205/sort/linearithmic/QuickSortDualPivotDriver.java)

With these drivers, we can get the data about execution time and number of different instruments.

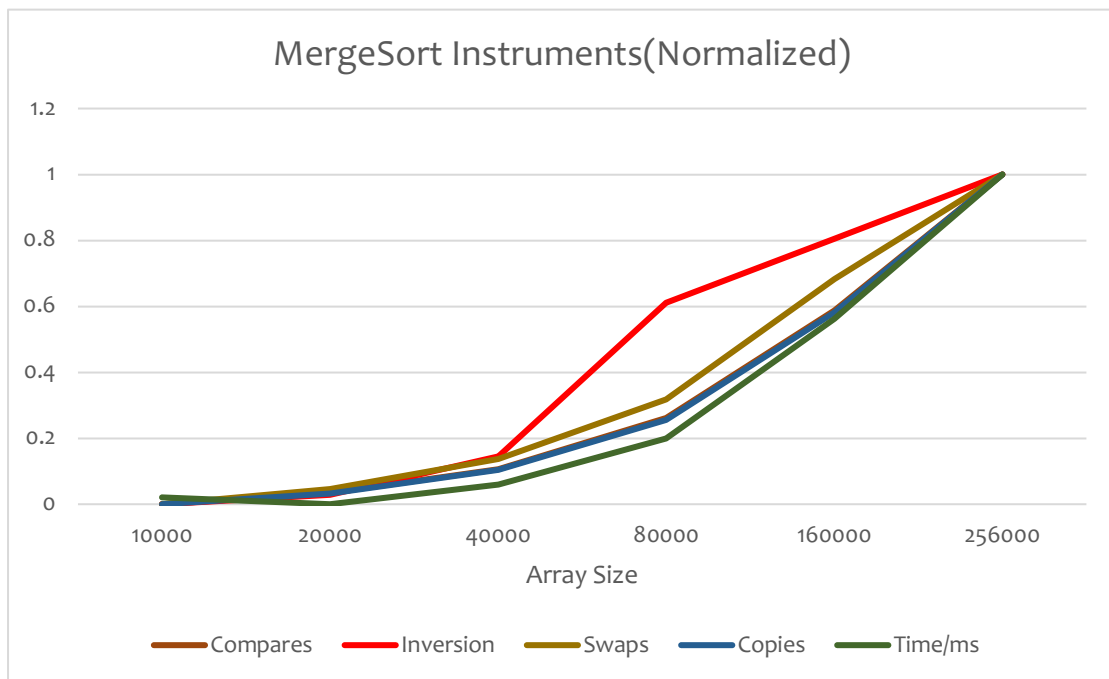
MergeSort					
Size	Compares	Inversions	Swaps	Copies	Time/ms
10000	123561	24905797	9807	109942	3.69917165
20000	267211	99814902	19562	239912	2.74883333
40000	573986	399884459	39231	519817	5.43663459
80000	1228013	1597155228	78016	1119565	11.60084536
160000	2616557	2097953446	156226	2399266	27.79857249
256000	4361038	2600000000	224070	4049912	47.29149669

HeapSort					
Size	Compares	Inversions	Swaps	Copies	Time/ms
10000	235502	25137166	124349	0	1.4379617
20000	510812	100113321	268480	0	3.07158838
40000	1101105	400315862	576605	0	6.44107251
80000	2362754	1608166123	1233241	0	14.1264979
160000	5045744	2097934506	2627385	0	31.98909374
256000	8411045	2400000000	4372370	0	55.38631043

QuickSort Dual Pivots					
Size	Compares	Inversions	Swaps	Copies	Time/ms
10000	151787	24835299	62014	0	2.78224168
20000	338489	99832908	143257	0	2.02734504
40000	712249	399394145	312090	0	4.36833037
80000	1534839	1600340564	626596	0	9.63244332
160000	3265825	2086294638	1377444	0	20.48452921
256000	5525002	2500000000	2190632	0	33.67086083

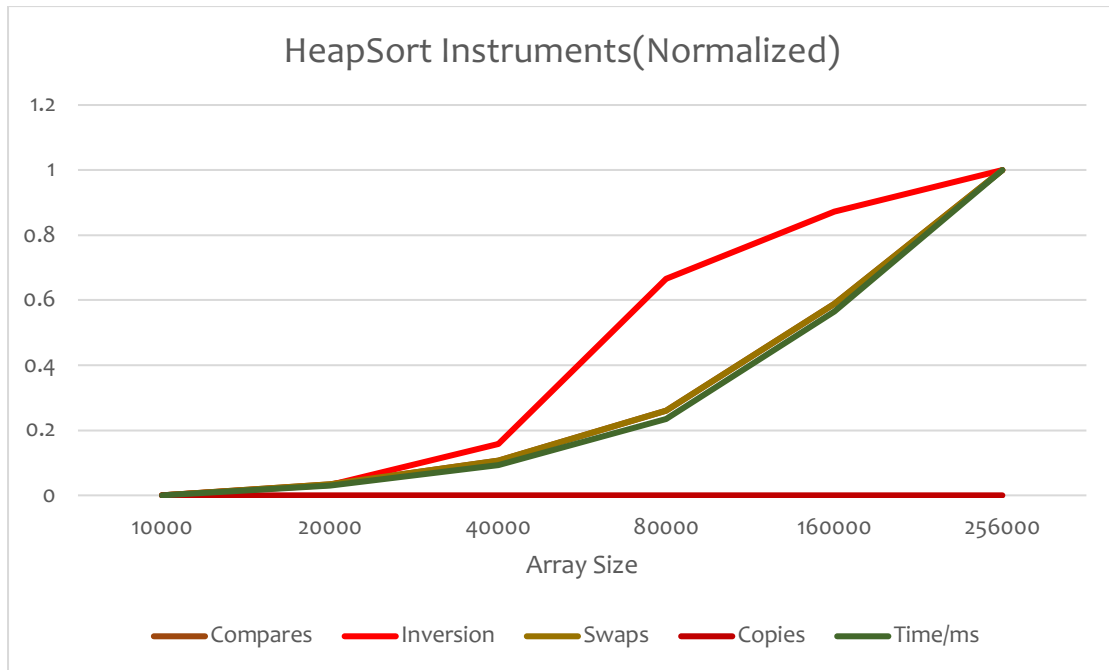
By normalizing the number and draw the graph of those data, we can get the relationship between execution time and instruments in different sort algorithm.

### MergeSort:



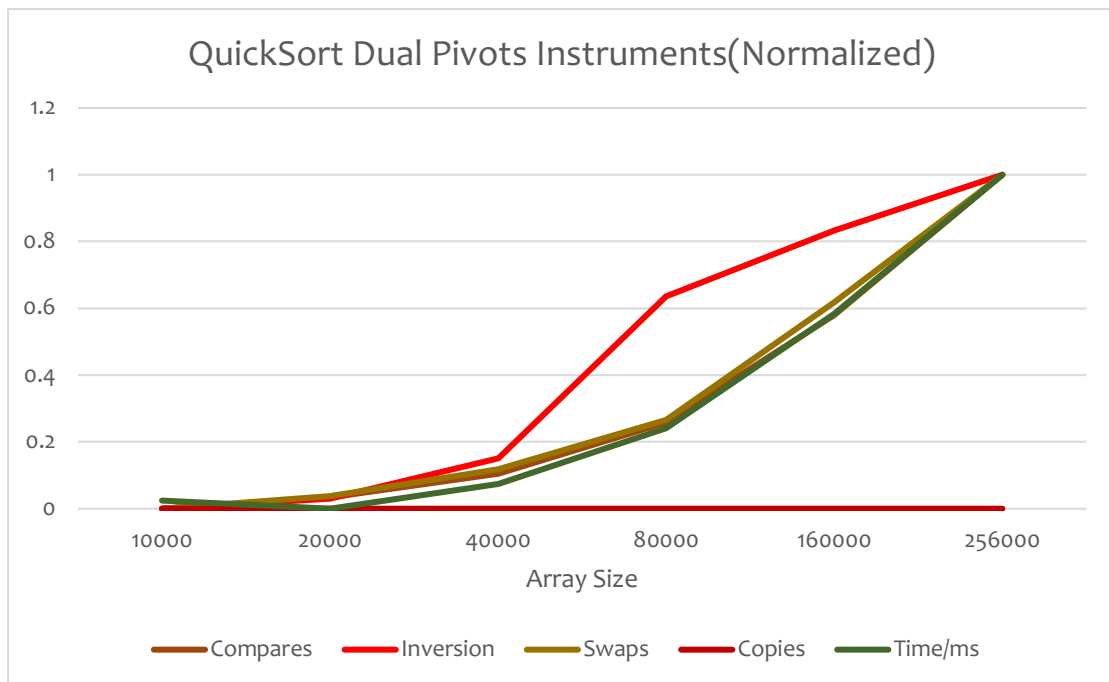
It shows that the number of Swaps/Copies fits Execution Time perfectly, which means that the number of Swaps/Copies determines the Time in MergeSort algorithm.

### HeapSort:



It shows the number of Compares and Swaps determine the Execution Time in HeapSort algorithm.

### QuickSort Dual Pivots



It shows that the number of Compares and Swaps determine the Execution Time in QuickSort Dual Pivots algorithm.