## Assignment 6 - Program Structures and Algorithms

## Spring 2023

## Section - 01

**NAME: Naman Diwan** **NUID: 002724115**

In this assignment, your task is to determine--for sorting algorithms--what is the best predictor of total execution time: comparisons, swaps/copies, hits (array accesses), or something else.

You will run the benchmarks for merge sort, (dual-pivot) quick sort, and heap sort. You will sort randomly generated arrays of between 10,000 and 256,000 elements (doubling the size each time). If you use the *SortBenchmark*, as I expect, the number of runs is chosen for you. So, you can ignore the instructions about setting the number of runs.

For each experiment (a sort method of a given size), you will run it twice: once for the instrumentation, once (without instrumentation) for the timing.

Of course, you will be using the *Benchmark* and/or *Timer* classes, as you did in a previous assignment.

You must support your (clearly stated) conclusions with evidence from the benchmarks (you should provide log/log charts and spreadsheets typically).

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

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

The configuration for these benchmarks is determined by the *config.ini* file. It should be reasonably easy to figure out how it all works. The config.ini file should look something like this:

```
[sortbenchmark]
version = 1.0.0 (sortbenchmark)

[helper]
instrument = true
seed = 0
cutoff =

[instrumenting]
# The options in this section apply only if instrument (in [helper]) is set
to true.
```

```
swaps = true
compares = true
copies = true
fixes = false
hits = true
# This slows everything down a lot so keep this small (or zero)
inversions = 0

[benchmarkstringsorters]
words = 1000 # currently ignored
runs = 20 # currently ignored
mergesort = true
timsort  =  false
quicksort = false
introsort = false
insertionsort = false
bubblesort = false
quicksort3way = false
quicksortDualPivot = true
randomsort = false

[benchmarkdatesorters]
timsort = false
n = 100000

[mergesort]
insurance = false
nocopy = true

[shellsort]
n = 100000

[operationsbenchmark]
nlargest = 10000000
repetitions = 10
```

There is no *config.ini* entry for heapsort. You will have to work that one out for yourself.

The number of runs is actually determined by the problem sizes using a fixed formula.

One more thing: the sizes of the experiments are actually defined in the command line (if you are running in IntelliJ/IDEA then, under *Edit Configurations* for the *SortBenchmark*, enter 10000 20000 etc. in the box just above *CLI arguments to your application)*.

You will also need to edit the SortBenchmark class. Insert the following lines before the introsort section:

```
if (isConfigBenchmarkStringSorter("heapsort")) {
```

```
    Helper<String> helper = HelperFactory.create("Heapsort", nWords, config
);
    runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper), ti
meLoggersLinearithmic);
}
```

Then you can add the following extra line into the config.ini file (again, before the introsort line (which is 25 for me):

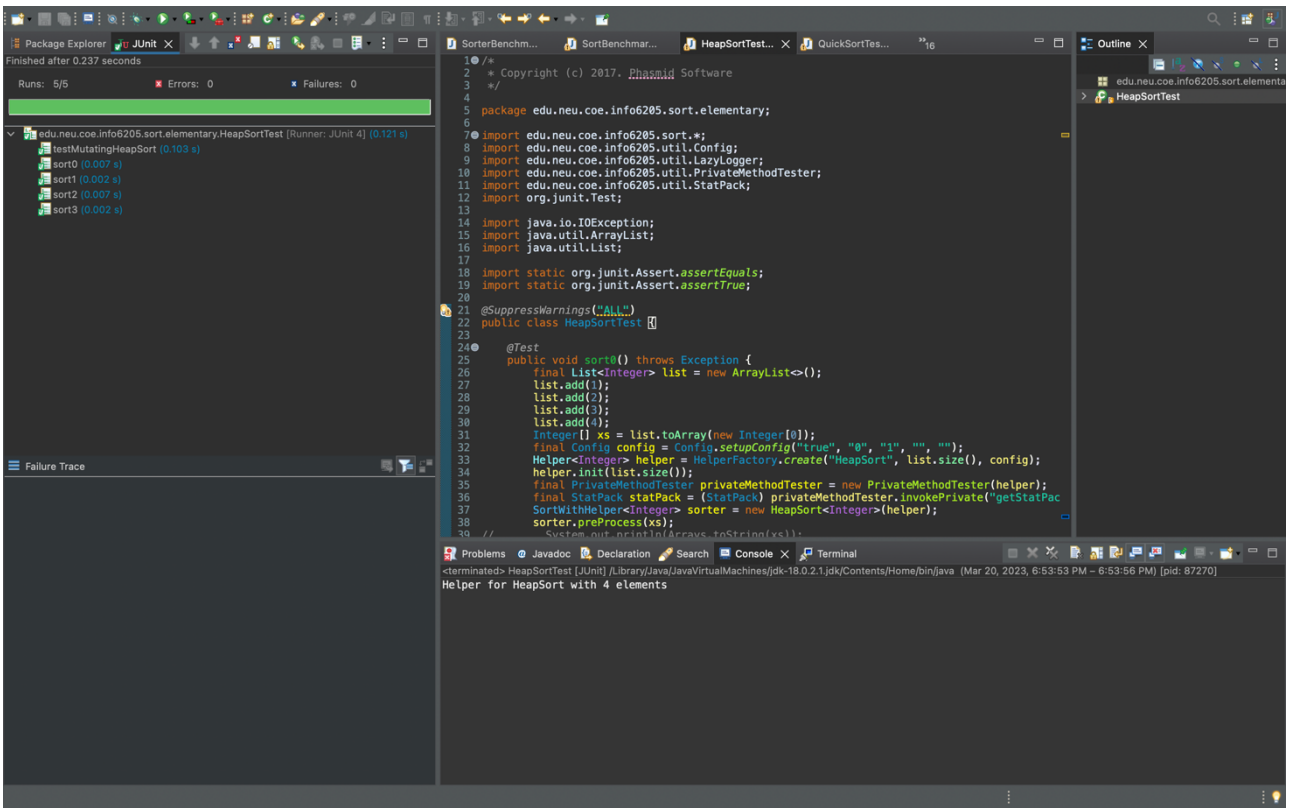
```
heapsort = true
```

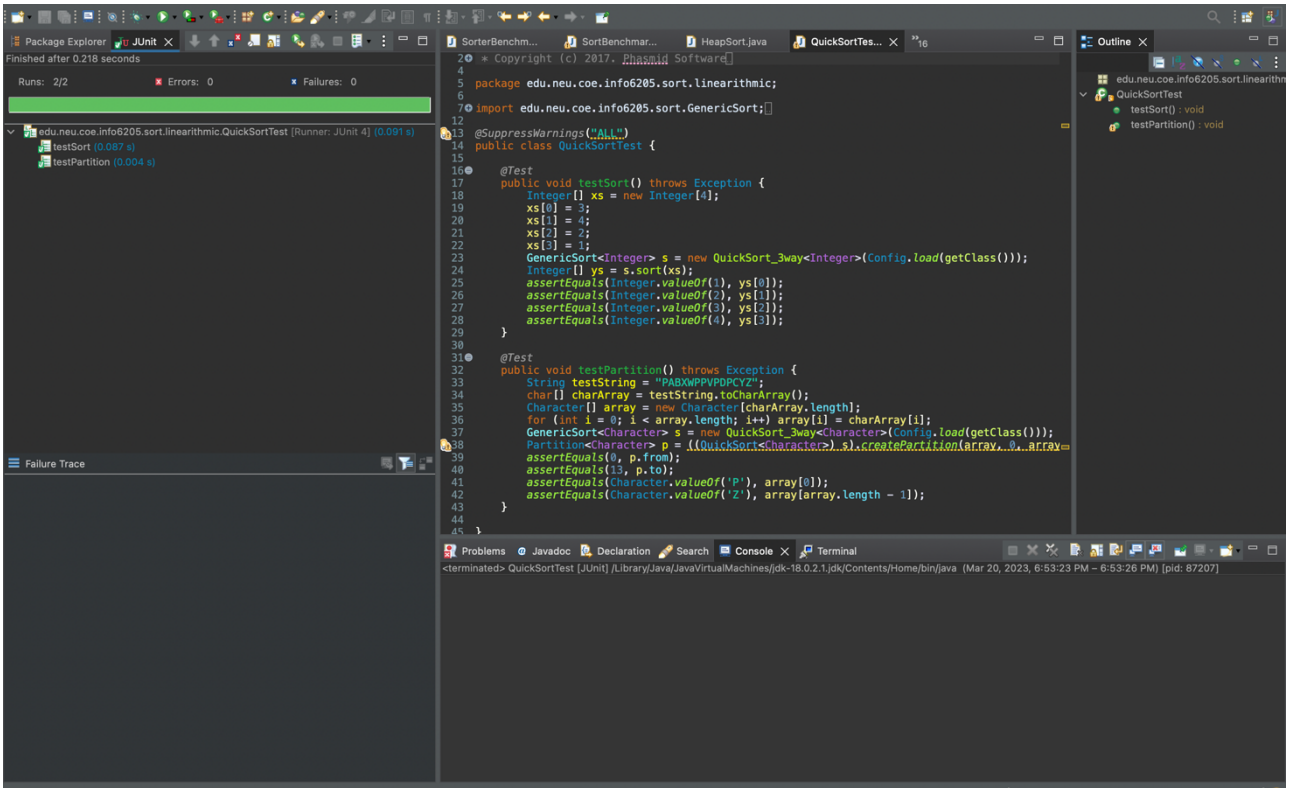
Remember that your job is to determine the best predictor: that will mean the graph of the appropriate observation will match the graph of the timings most closely.

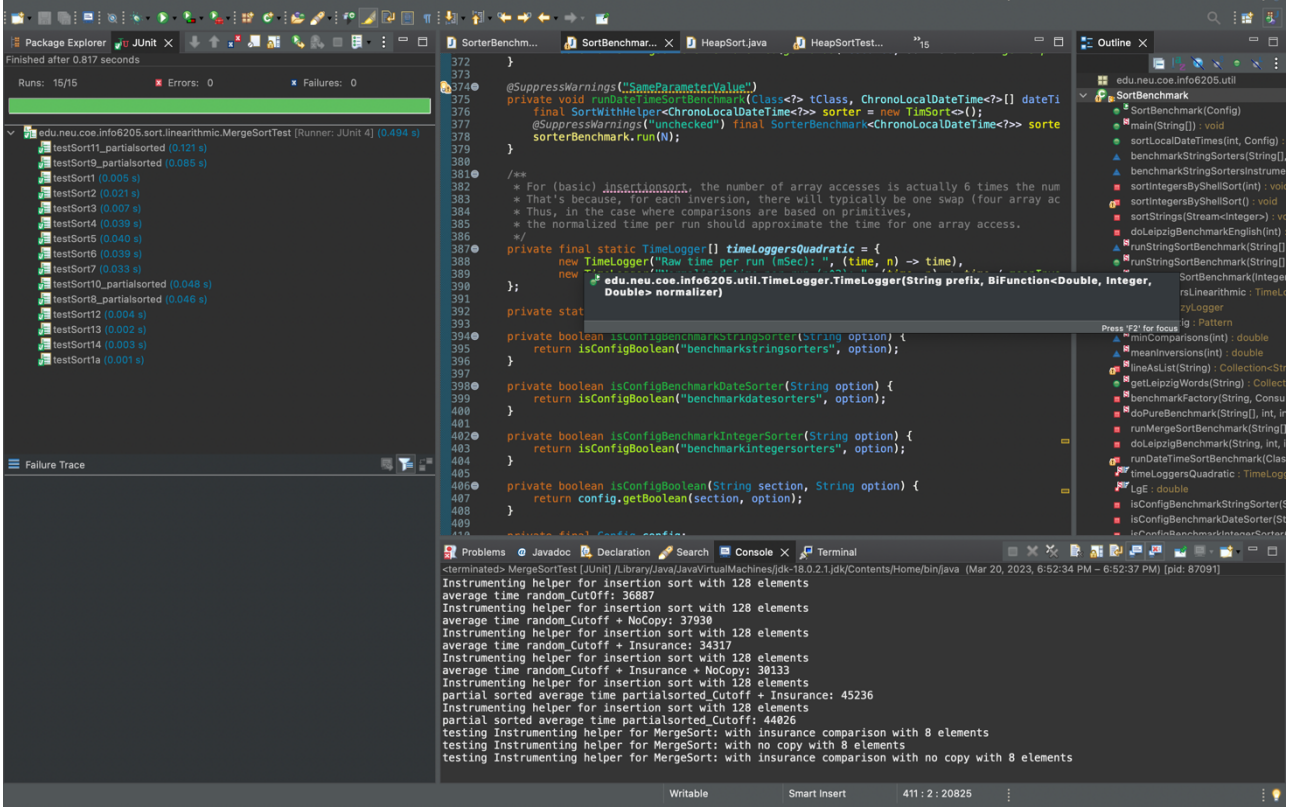
**Solution –**

**Code uploaded to - [https://github.com/namandiwan10/INFO-6205-Assignment-1](https://github.com/namandiwan10/INFO-6205-Assignment-1)**

**Unit Tests -**

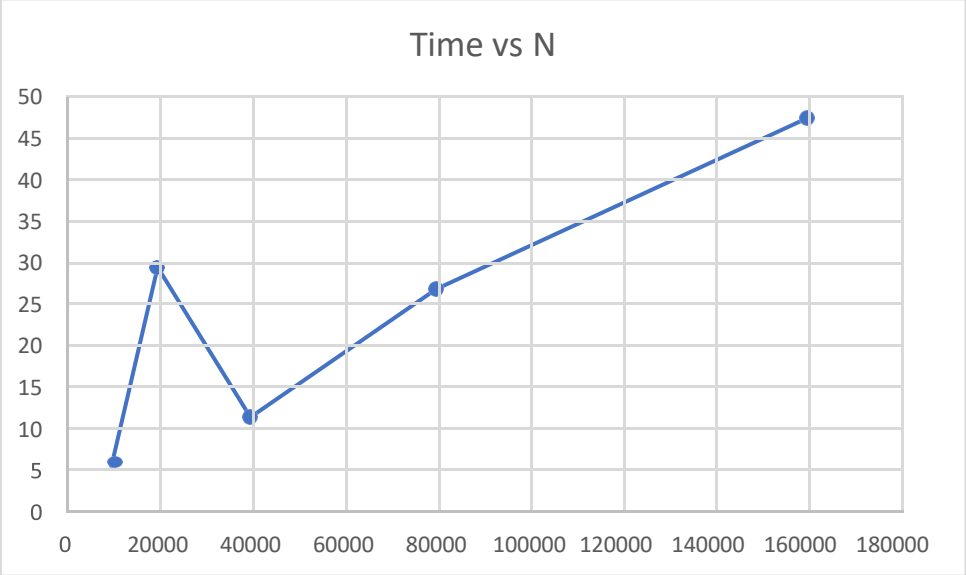


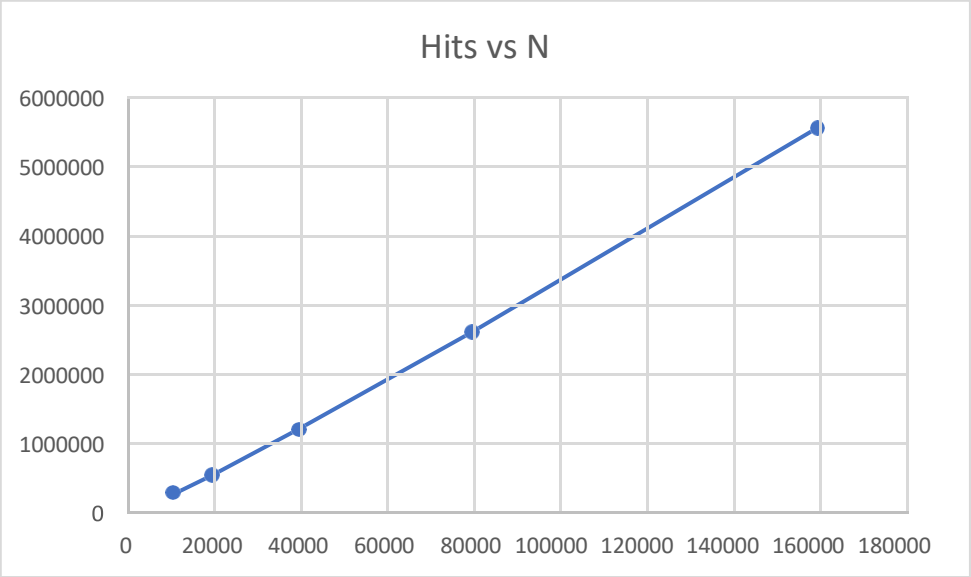

Package Explorer
JUnit
Finished after 0.218 seconds
Runs: 2/2
Errors: 0
Failures: 0
edu.neu.coe.info6205.sort.linearithmic.QuickSortTest [Runner: JUnit 4] (0.091 s)
testSort (0.087 s)
testPartition (0.004 s)
Failure Trace
SorterBenchm...
SortBenchmar...
HeapSort.java
QuickSortTes...
Outline
edu.neu.coe.info6205.sort.linearithm
QuickSortTest
testSort() : void
testPartition() : void
Problems
Javadoc
Declaration
Search
Console
Terminal
<terminated> QuickSortTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java (Mar 20, 2023, 6:53:23 PM – 6:53:26 PM) [pid: 87207]

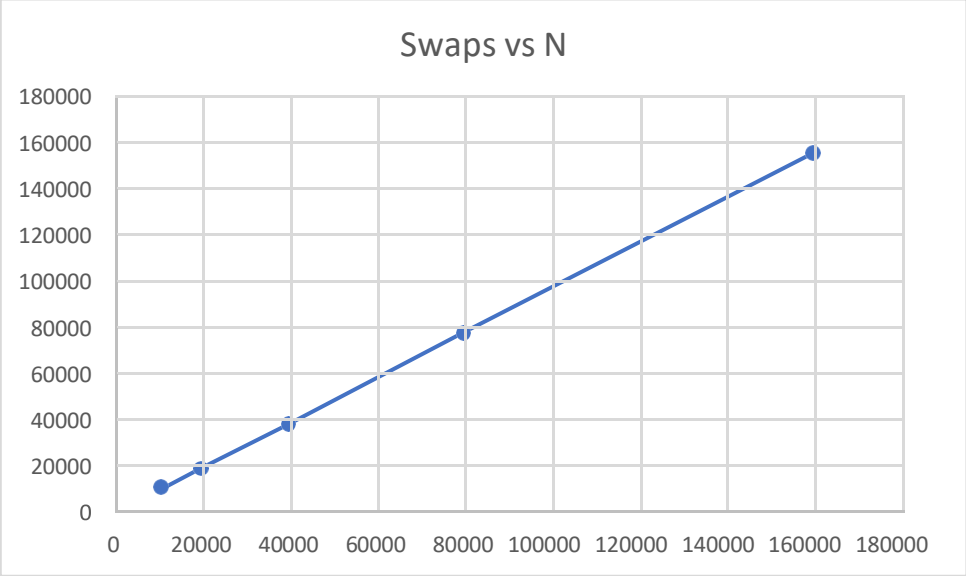

Package Explorer
JUnit
Finished after 0.817 seconds
Runs: 15/15
Errors: 0
Failures: 0
edu.neu.coe.info6205.sort.linearithmic.MergeSortTest [Runner: JUnit 4] (0.494 s)
testSort11_partialsorted (0.121 s)
testSort9_partialsorted (0.085 s)
testSort1 (0.005 s)
testSort2 (0.021 s)
testSort3 (0.007 s)
testSort4 (0.039 s)
testSort5 (0.040 s)
testSort6 (0.039 s)
testSort7 (0.033 s)
testSort10_partialsorted (0.048 s)
testSort8_partialsorted (0.046 s)
testSort12 (0.004 s)
testSort13 (0.002 s)
testSort14 (0.003 s)
testSort1a (0.001 s)
Failure Trace
SorterBenchm...
SortBenchmar...
HeapSort.java
HeapSortTest...
Outline
edu.neu.coe.info6205.util
SortBenchmark
SortBenchmark(Config)
main(String[]) : void
Problems
Javadoc
Declaration
Search
Console
Terminal
<terminated> MergeSortTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java (Mar 20, 2023, 6:52:34 PM – 6:52:37 PM) [pid: 87091]
Instrumenting helper for insertion sort with 128 elements
average time random_CutOff: 36887
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + NoCopy: 37930
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance: 34317
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance + NoCopy: 30133
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance: 45236
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff: 44026
testing Instrumenting helper for MergeSort: with insurance comparison with 8 elements
testing Instrumenting helper for MergeSort: with no copy with 8 elements
testing Instrumenting helper for MergeSort: with insurance comparison with no copy with 8 elements
Writable
Smart Insert
411 : 2 : 20825

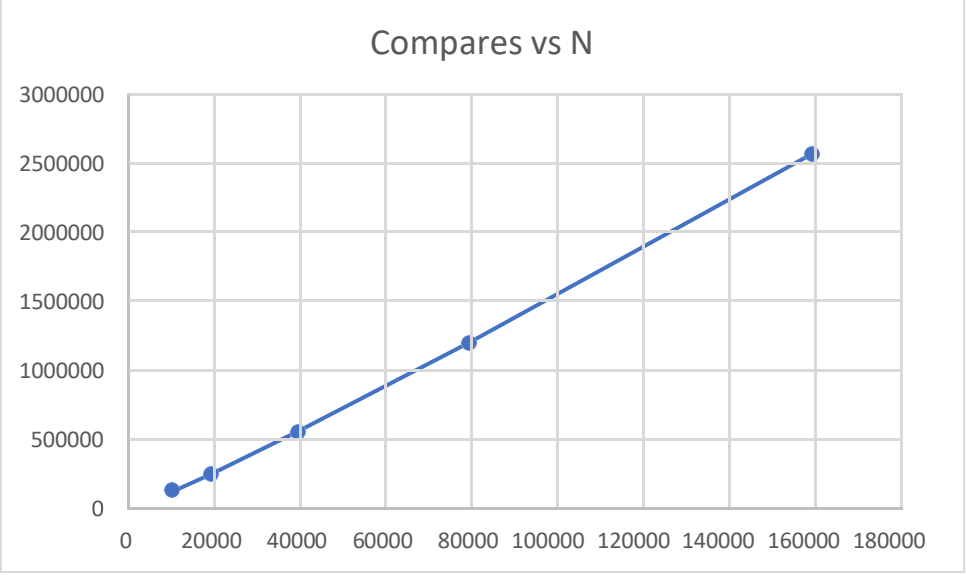
**Recorded Observations –**

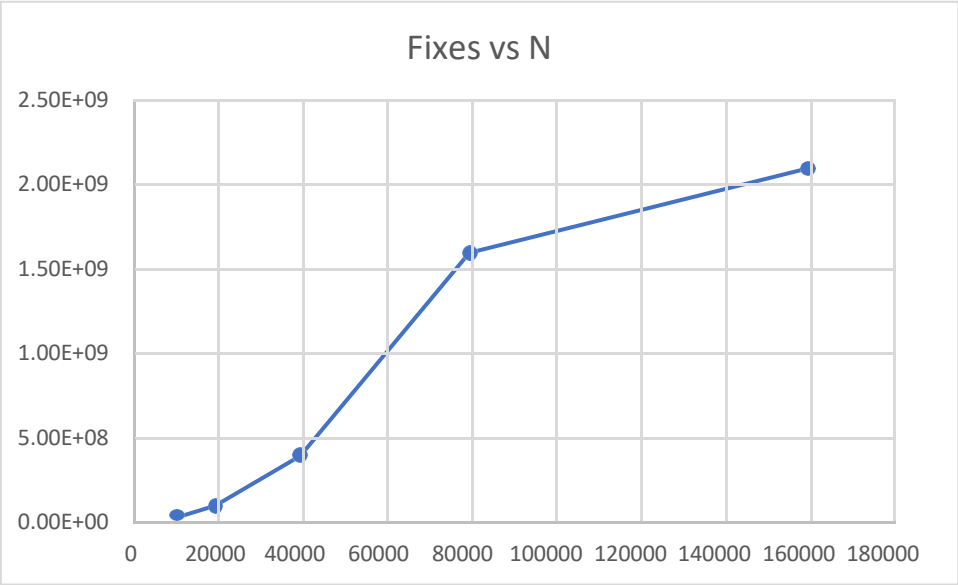
**MERGE SORT –**




Hits vs N
6000000
5000000
4000000
3000000
2000000
1000000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

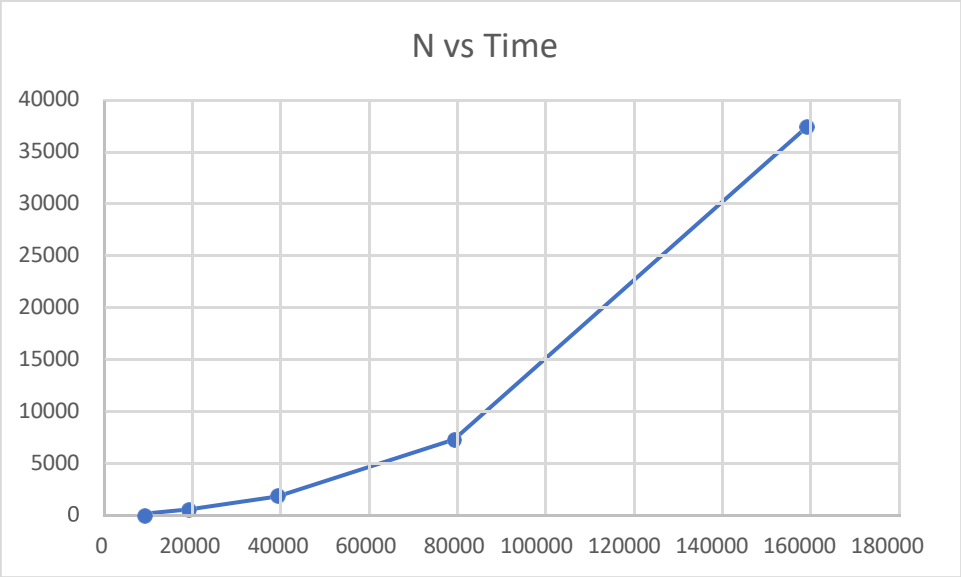

Swaps vs N
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

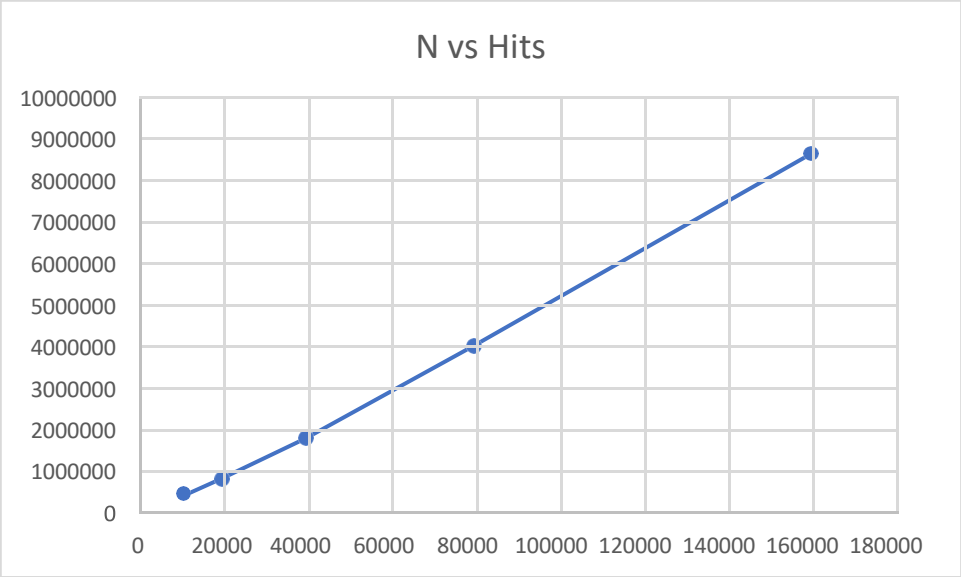

Compares vs N
3000000
2500000
2000000
1500000
1000000
500000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

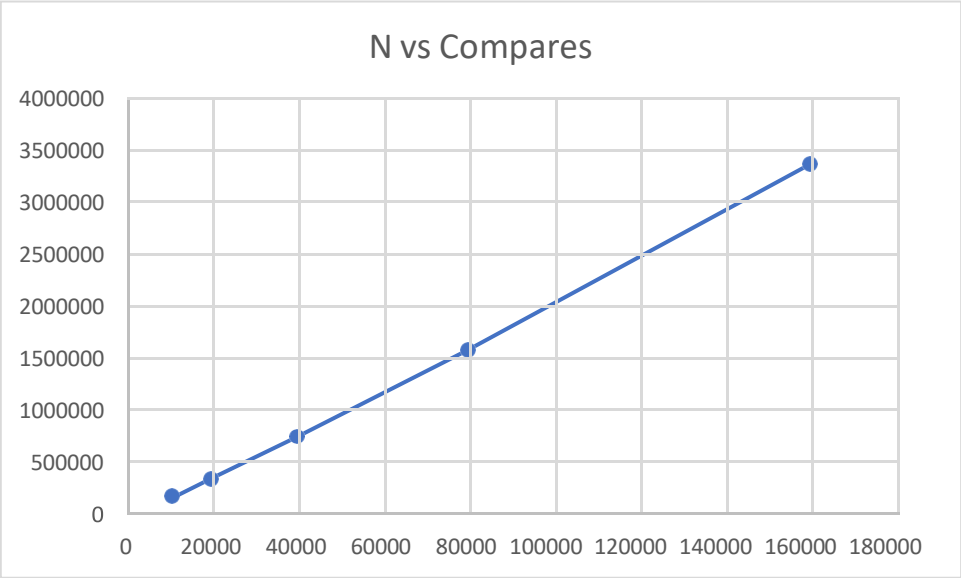

Fixes vs N
2.50E+09
2.00E+09
1.50E+09
1.00E+09
5.00E+08
0.00E+00
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

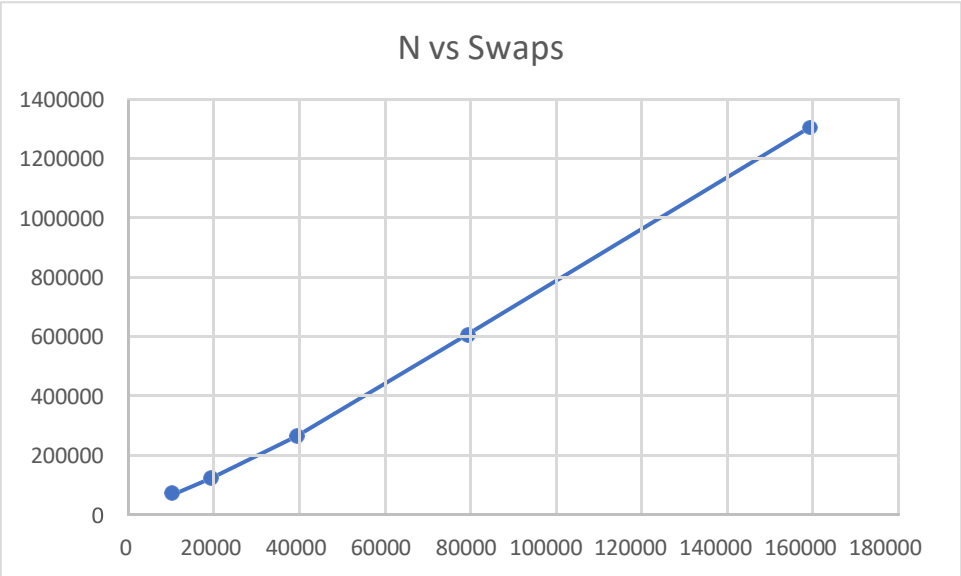
**Quicksort –**

---


N vs Time
40000
35000
30000
25000
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

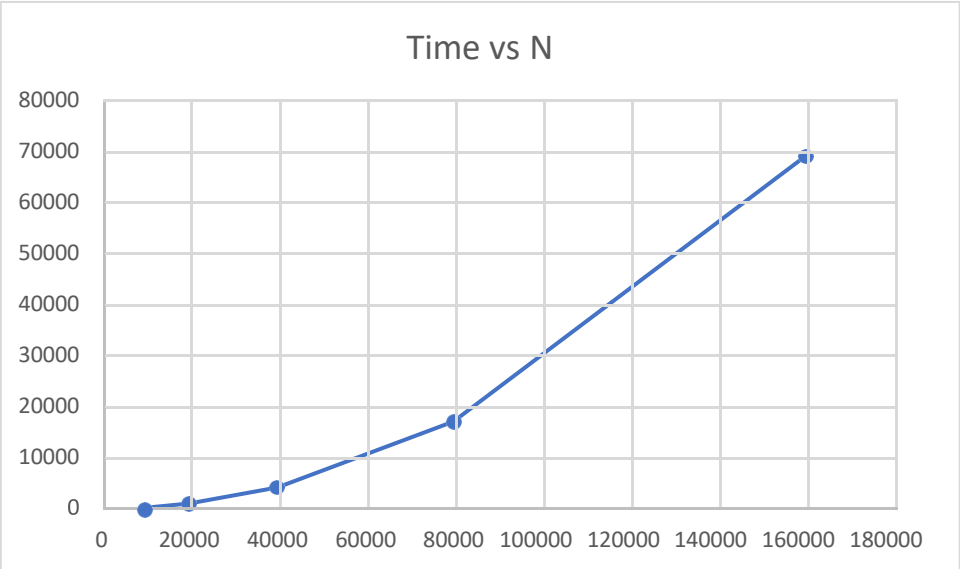

N vs Hits
10000000
9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

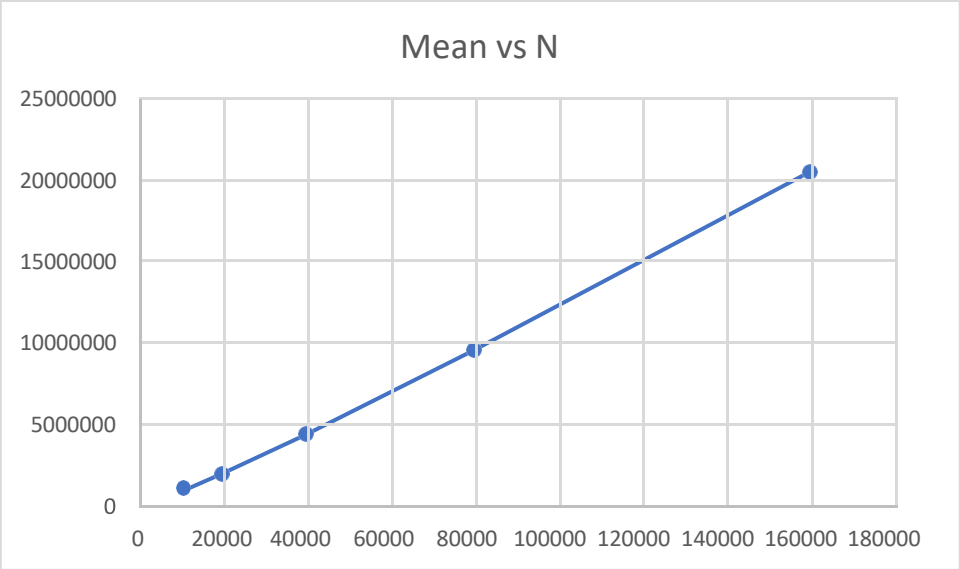

N vs Compares
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

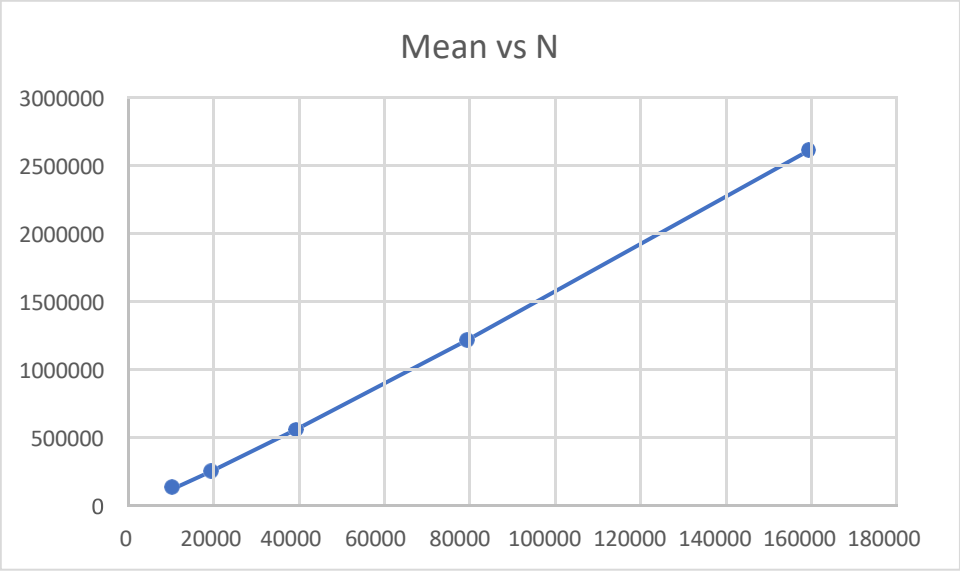

N vs Swaps
1400000
1200000
1000000
800000
600000
400000
200000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

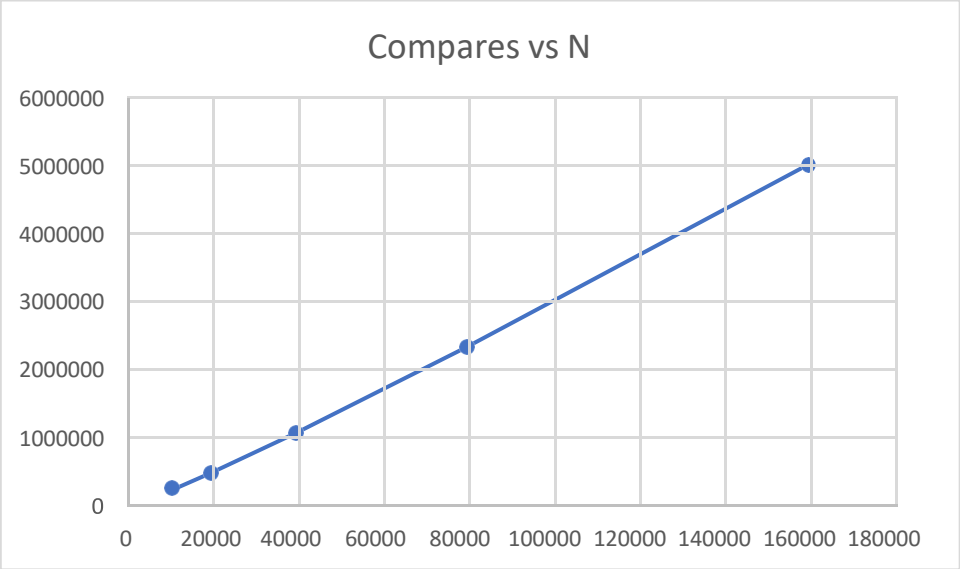
**Heap Sort –**

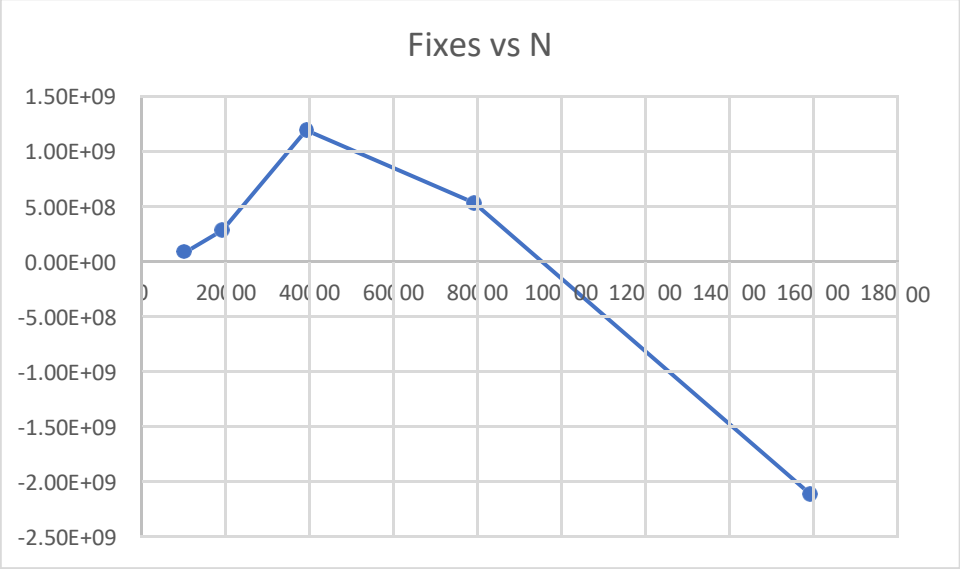
---

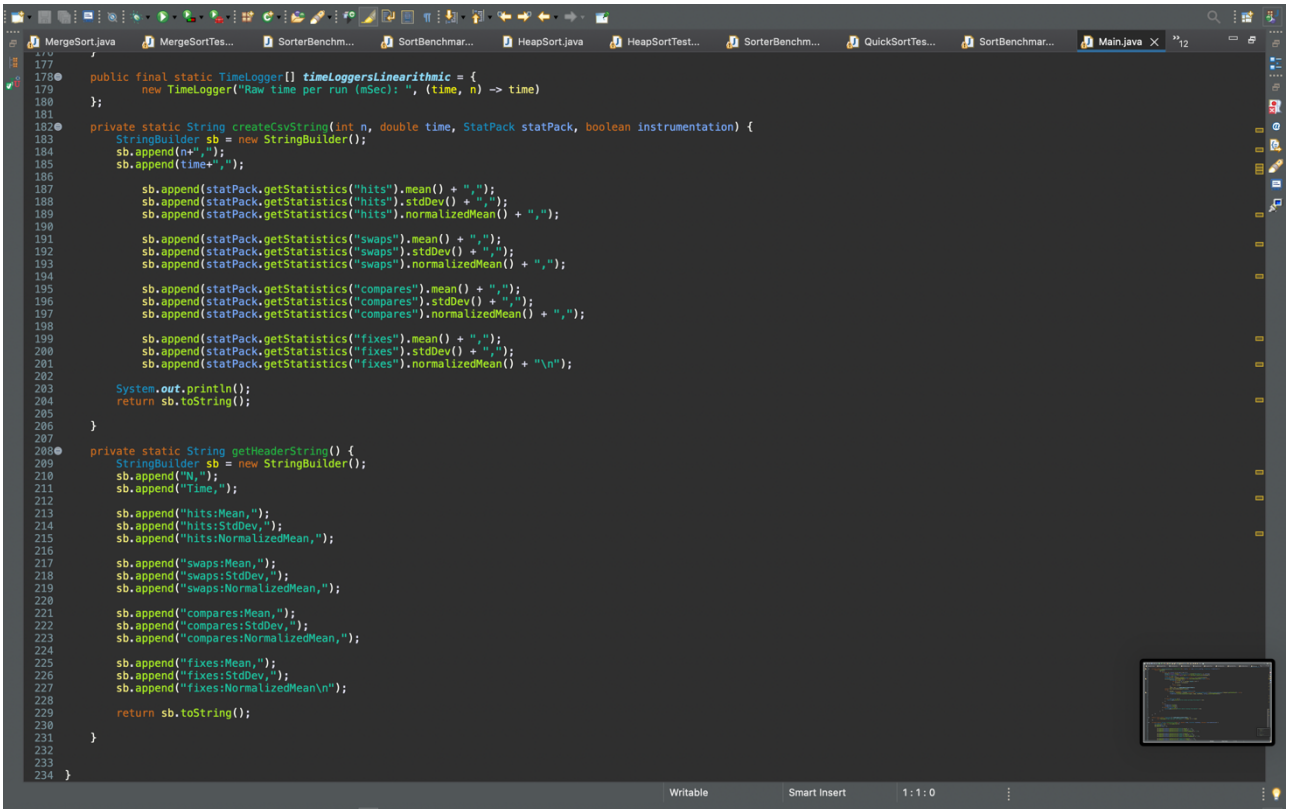

Time vs N
80000
70000
60000
50000
40000
30000
20000
10000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

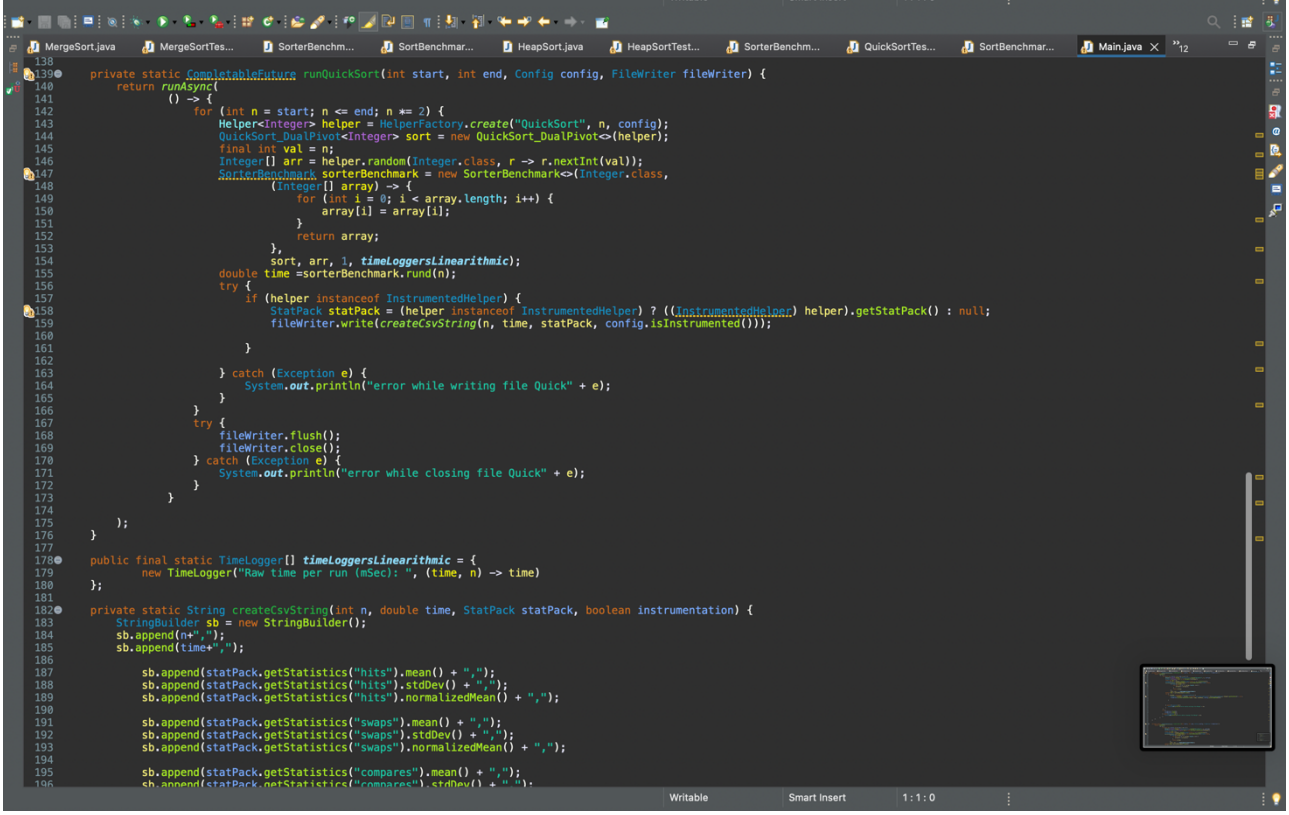

Mean vs N
25000000
20000000
15000000
10000000
5000000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

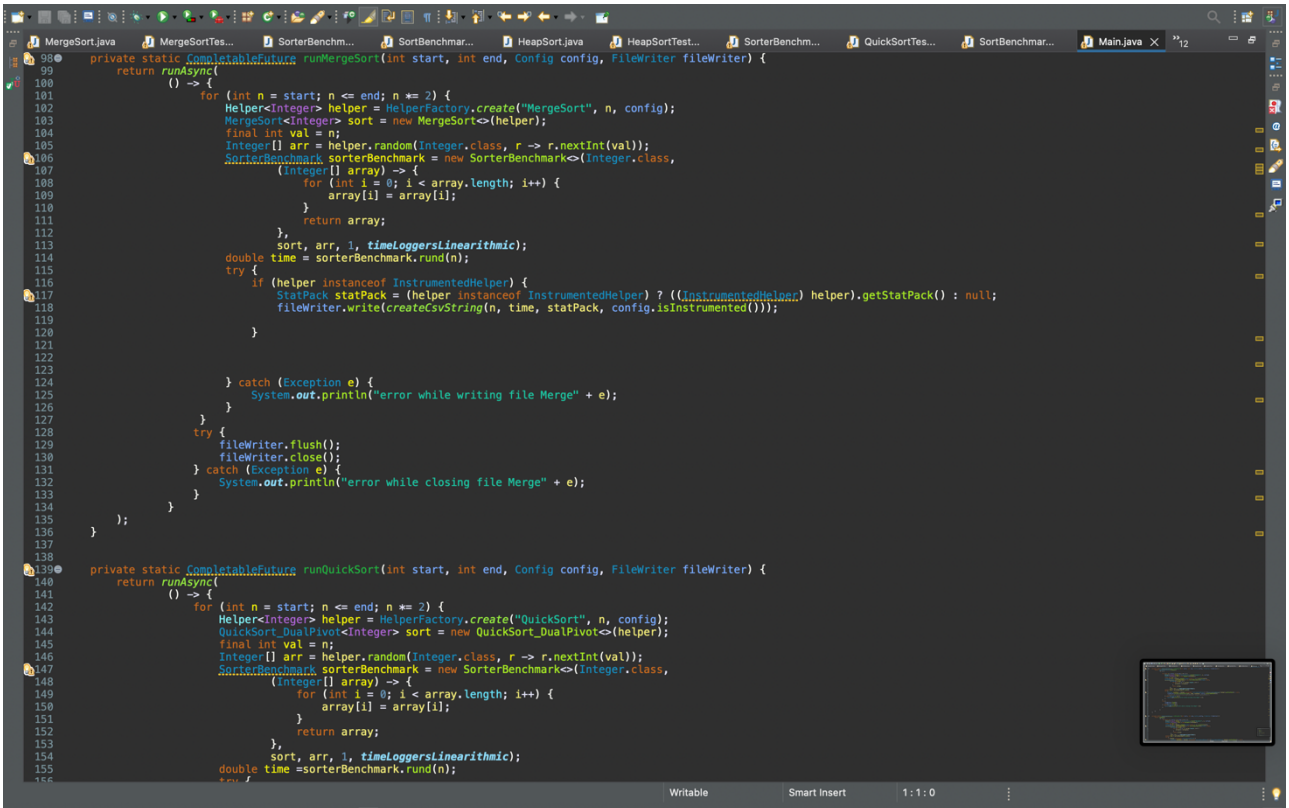

Mean vs N
3000000
2500000
2000000
1500000
1000000
500000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

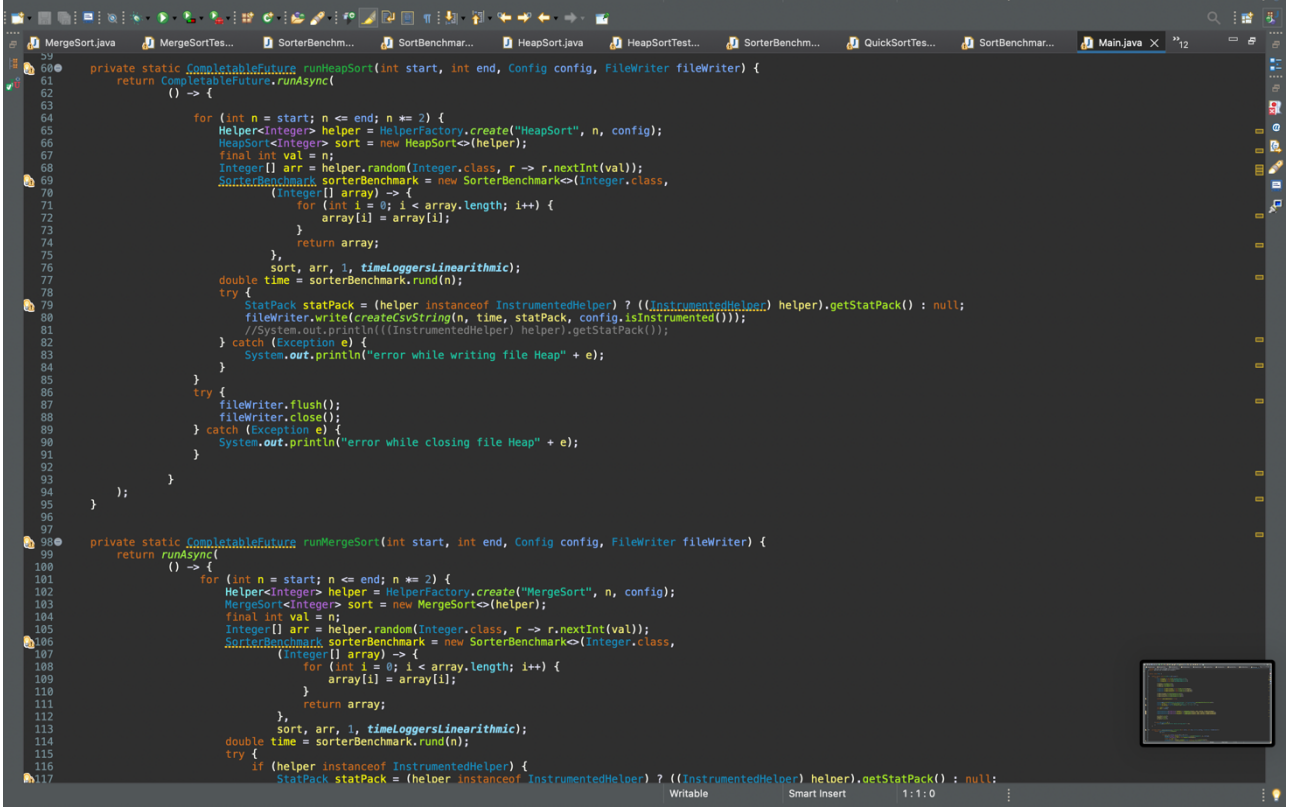

Compares vs N
6000000
5000000
4000000
3000000
2000000
1000000
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000

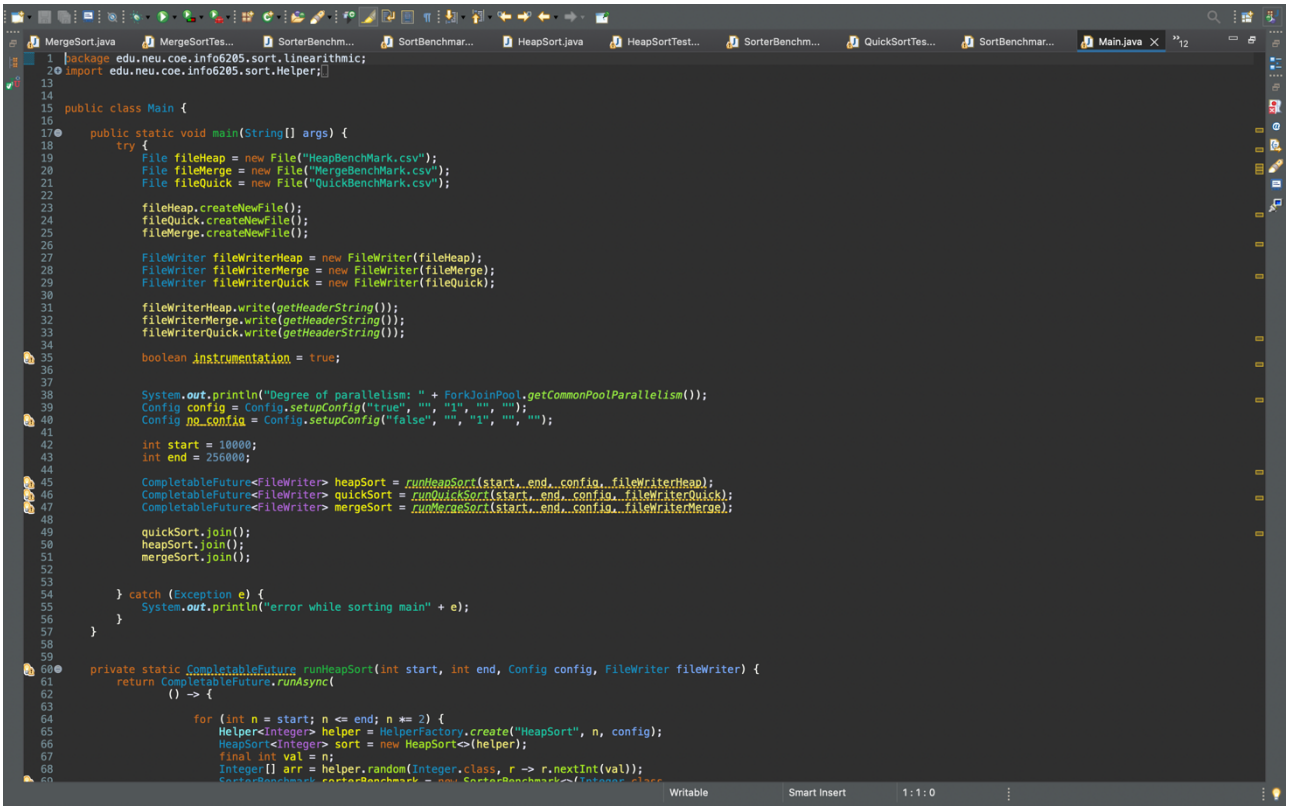

Fixes vs N
1.50E+09
1.00E+09
5.00E+08
0.00E+00
-5.00E+08
-1.00E+09
-1.50E+09
-2.00E+09
-2.50E+09


MergeSort.java
MergeSortTes...
SorterBenchm...
SortBenchmar...
HeapSort.java
HeapSortTest...
SorterBenchm...
QuickSortTes...
SortBenchmar...
Main.java
Writable
Smart Insert
1 : 1 : 0


MergeSort.java
MergeSortTes...
SorterBenchm...
SortBenchmar...
HeapSort.java
HeapSortTest...
SorterBenchm...
QuickSortTes...
SortBenchmar...
Main.java
Writable
Smart Insert
1 : 1 : 0

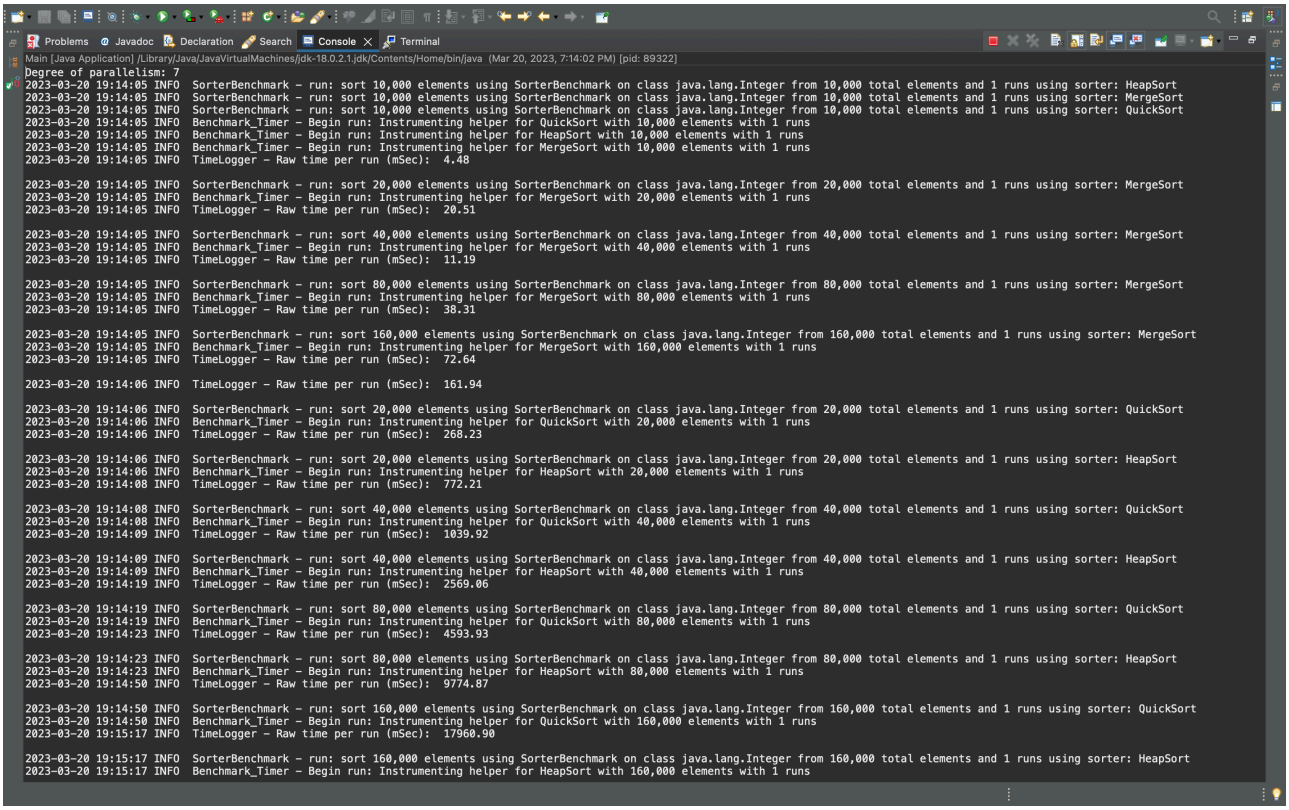

MergeSort.java
MergeSortTes...
SorterBenchm...
SortBenchmar...
HeapSort.java
HeapSortTest...
SorterBenchm...
QuickSortTes...
SortBenchmar...
Main.java
Writable
Smart Insert
1 : 1 : 0


MergeSort.java
MergeSortTes...
SorterBenchm...
SortBenchmar...
HeapSort.java
HeapSortTest...
SorterBenchm...
QuickSortTes...
SortBenchmar...
Main.java
Writable
Smart Insert
1 : 1 : 0



```
package edu.neu.coe.info6205.sort.linearithmic;
import edu.neu.coe.info6205.sort.Helper;


public class Main {

    public static void main(String[] args) {
        try {
            File fileHeap = new File("HeapBenchMark.csv");
            File fileMerge = new File("MergeBenchMark.csv");
            File fileQuick = new File("QuickBenchMark.csv");

            fileHeap.createNewFile();
            fileQuick.createNewFile();
            fileMerge.createNewFile();

            FileWriter fileWriterHeap = new FileWriter(fileHeap);
            FileWriter fileWriterMerge = new FileWriter(fileMerge);
            FileWriter fileWriterQuick = new FileWriter(fileQuick);

            fileWriterHeap.write(getHeaderString());
            fileWriterMerge.write(getHeaderString());
            fileWriterQuick.write(getHeaderString());

            boolean instrumentation = true;


            System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
            Config config = Config.setupConfig("true", "", "1", "", "");
            Config no_config = Config.setupConfig("false", "", "1", "", "");

            int start = 10000;
            int end = 256000;

            CompletableFuture<FileWriter> heapSort = runHeapSort(start, end, config, fileWriterHeap);
            CompletableFuture<FileWriter> quickSort = runQuickSort(start, end, config, fileWriterQuick);
            CompletableFuture<FileWriter> mergeSort = runMergeSort(start, end, config, fileWriterMerge);

            quickSort.join();
            heapSort.join();
            mergeSort.join();


        } catch (Exception e) {
            System.out.println("error while sorting main" + e);
        }
    }


    private static CompletableFuture runHeapSort(int start, int end, Config config, FileWriter fileWriter) {
        return CompletableFuture.runAsync(
                () -> {

                    for (int n = start; n <= end; n *= 2) {
                        Helper<Integer> helper = HelperFactory.create("HeapSort", n, config);
                        HeapSort<Integer> sort = new HeapSort<>(helper);
                        final int val = n;
                        Integer[] arr = helper.random(Integer.class, r -> r.nextInt(val));
```

Writable | Smart Insert | 1 : 1 : 0



```
Problems  Javadoc  Declaration  Search  Console  Terminal
Main [Java Application] /Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java  (Mar 20, 2023, 7:14:02 PM) [pid: 89322]
Degree of parallelism: 7
2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 1 runs using sorter: HeapSort
2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 1 runs using sorter: MergeSort
2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.Integer from 10,000 total elements and 1 runs using sorter: QuickSort
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort with 10,000 elements with 1 runs
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 10,000 elements with 1 runs
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 10,000 elements with 1 runs
2023-03-20 19:14:05 INFO  TimeLogger - Raw time per run (mSec):  4.48

2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 1 runs using sorter: MergeSort
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 20,000 elements with 1 runs
2023-03-20 19:14:05 INFO  TimeLogger - Raw time per run (mSec):  20.51

2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total elements and 1 runs using sorter: MergeSort
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 40,000 elements with 1 runs
2023-03-20 19:14:05 INFO  TimeLogger - Raw time per run (mSec):  11.19

2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total elements and 1 runs using sorter: MergeSort
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 80,000 elements with 1 runs
2023-03-20 19:14:05 INFO  TimeLogger - Raw time per run (mSec):  38.31

2023-03-20 19:14:05 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total elements and 1 runs using sorter: MergeSort
2023-03-20 19:14:05 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort with 160,000 elements with 1 runs
2023-03-20 19:14:05 INFO  TimeLogger - Raw time per run (mSec):  72.64

2023-03-20 19:14:06 INFO  TimeLogger - Raw time per run (mSec):  161.94

2023-03-20 19:14:06 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 1 runs using sorter: QuickSort
2023-03-20 19:14:06 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort with 20,000 elements with 1 runs
2023-03-20 19:14:06 INFO  TimeLogger - Raw time per run (mSec):  268.23

2023-03-20 19:14:06 INFO  SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.Integer from 20,000 total elements and 1 runs using sorter: HeapSort
2023-03-20 19:14:06 INFO  Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 20,000 elements with 1 runs
2023-03-20 19:14:08 INFO  TimeLogger - Raw time per run (mSec):  772.21

2023-03-20 19:14:08 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total elements and 1 runs using sorter: QuickSort
2023-03-20 19:14:08 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort with 40,000 elements with 1 runs
2023-03-20 19:14:09 INFO  TimeLogger - Raw time per run (mSec):  1039.92

2023-03-20 19:14:09 INFO  SorterBenchmark - run: sort 40,000 elements using SorterBenchmark on class java.lang.Integer from 40,000 total elements and 1 runs using sorter: HeapSort
2023-03-20 19:14:09 INFO  Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 40,000 elements with 1 runs
2023-03-20 19:14:19 INFO  TimeLogger - Raw time per run (mSec):  2569.06

2023-03-20 19:14:19 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total elements and 1 runs using sorter: QuickSort
2023-03-20 19:14:19 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort with 80,000 elements with 1 runs
2023-03-20 19:14:23 INFO  TimeLogger - Raw time per run (mSec):  4593.93

2023-03-20 19:14:23 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.Integer from 80,000 total elements and 1 runs using sorter: HeapSort
2023-03-20 19:14:23 INFO  Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 80,000 elements with 1 runs
2023-03-20 19:14:50 INFO  TimeLogger - Raw time per run (mSec):  9774.87

2023-03-20 19:14:50 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total elements and 1 runs using sorter: QuickSort
2023-03-20 19:14:50 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort with 160,000 elements with 1 runs
2023-03-20 19:15:17 INFO  TimeLogger - Raw time per run (mSec):  17960.90

2023-03-20 19:15:17 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.Integer from 160,000 total elements and 1 runs using sorter: HeapSort
2023-03-20 19:15:17 INFO  Benchmark_Timer - Begin run: Instrumenting helper for HeapSort with 160,000 elements with 1 runs
```