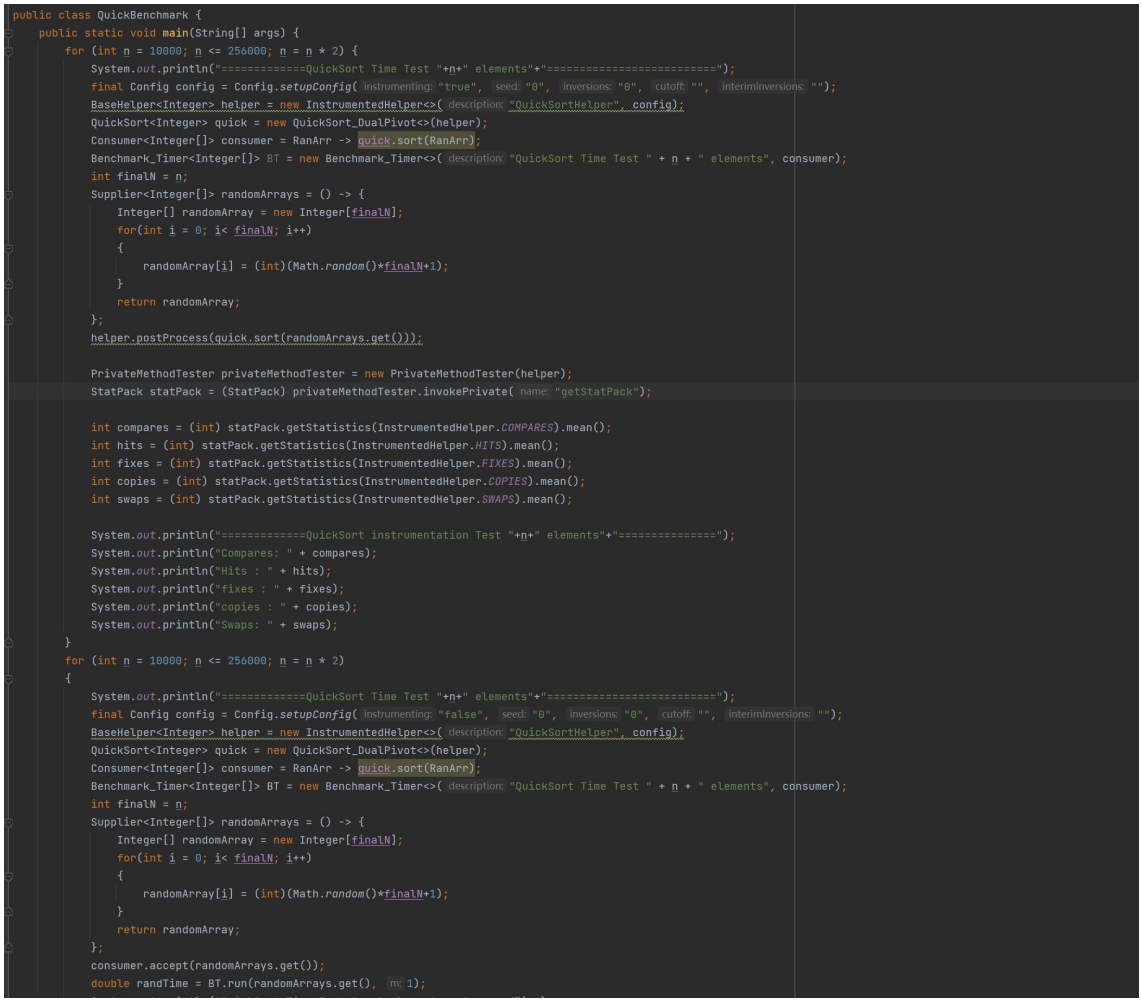
Assignment 6 (Hits as time predictor)

In this assignment, I create three class to find the relationship between the instruments and execution time.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/main/java/edu/neu/coe/info6205/util/ ”.

Screenshot of the three new class:

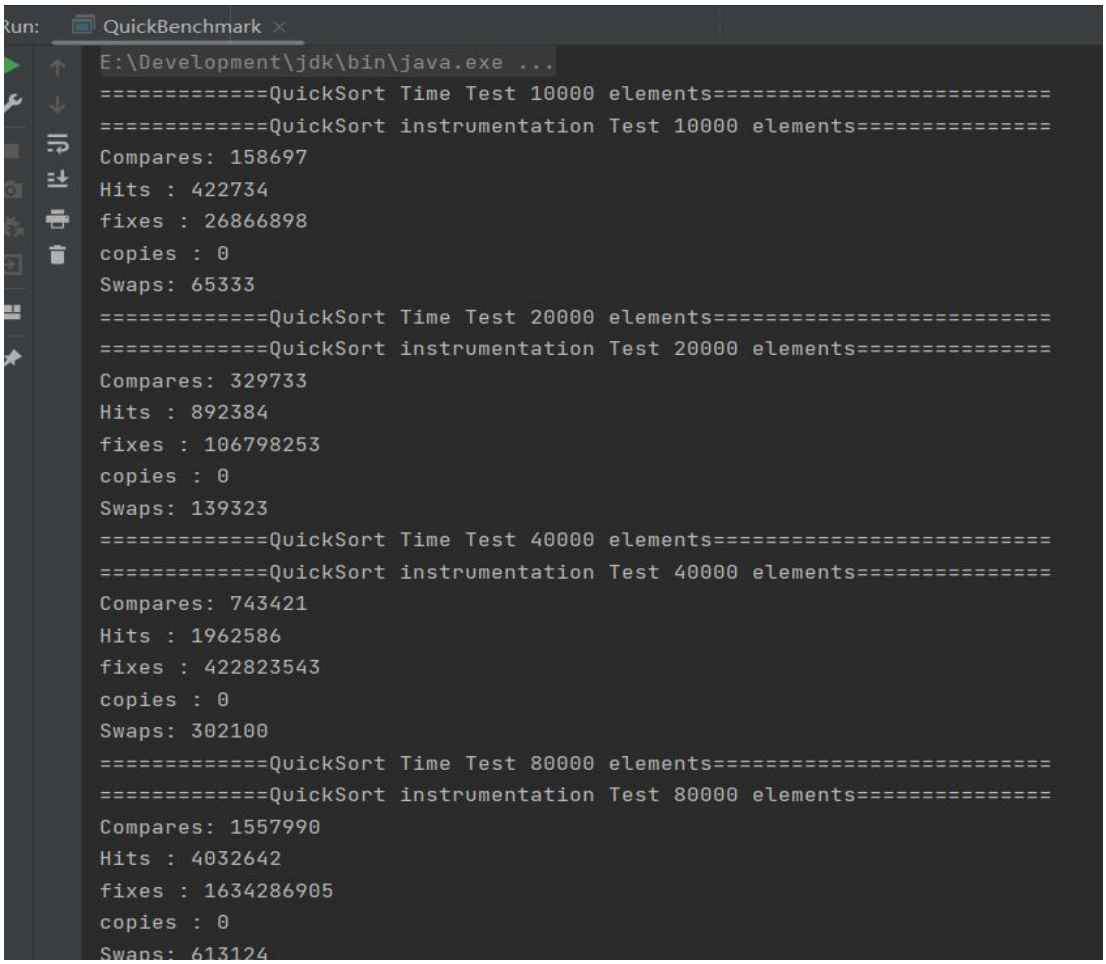


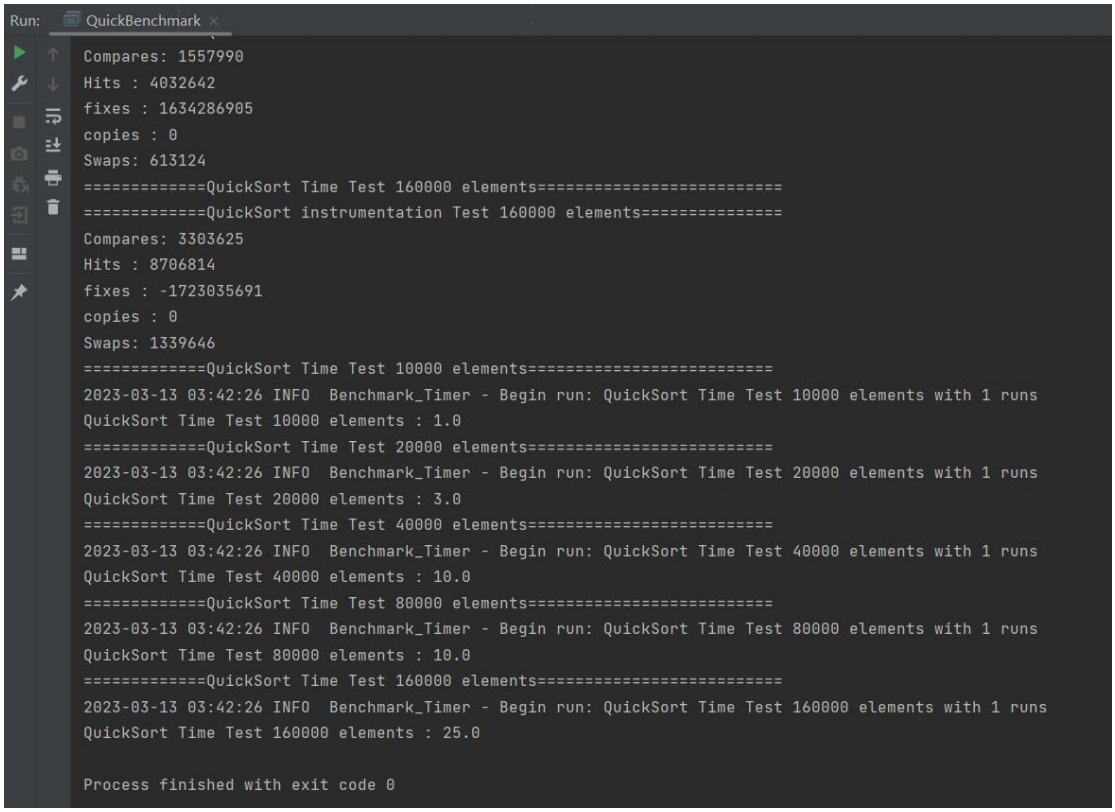
```
public class QuickBenchmark {
    public static void main(String[] args) {
        for (int n = 10000; n <= 256000; n = n * 2) {
            System.out.println("============QuickSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "true", seed: "0", inversions: "0", cutoff: "", interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "QuickSortHelper", config);
            QuickSort<Integer> quick = new QuickSort_DualPivot<>(helper);
            Consumer<Integer[]> consumer = RanArr -> quick.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "QuickSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
                Integer[] randomArray = new Integer[finalN];
                for(int i = 0; i< finalN; i++)
                {
                    randomArray[i] = (int)(Math.random()*finalN+1);
                }
                return randomArray;
            };
            helper.postProcess(quick.sort(randomArrays.get()));

            PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
            StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");

            int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
            int hits = (int) statPack.getStatistics(InstrumentedHelper.HITS).mean();
            int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
            int copies = (int) statPack.getStatistics(InstrumentedHelper.COPIES).mean();
            int swaps = (int) statPack.getStatistics(InstrumentedHelper.SWAPS).mean();

            System.out.println("============QuickSort instrumentation Test "+n+" elements"+"===============");
            System.out.println("Compares: " + compares);
            System.out.println("Hits : " + hits);
            System.out.println("fixes : " + fixes);
            System.out.println("copies : " + copies);
            System.out.println("Swaps: " + swaps);
        }
        for (int n = 10000; n <= 256000; n = n * 2)
        {
            System.out.println("============QuickSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "false", seed: "0", inversions: "0", cutoff: "", interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "QuickSortHelper", config);
            QuickSort<Integer> quick = new QuickSort_DualPivot<>(helper);
            Consumer<Integer[]> consumer = RanArr -> quick.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "QuickSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
                Integer[] randomArray = new Integer[finalN];
                for(int i = 0; i< finalN; i++)
                {
                    randomArray[i] = (int)(Math.random()*finalN+1);
                }
                return randomArray;
            };
            consumer.accept(randomArrays.get());
            double randTime = BT.run(randomArrays.get(), m: 1);
```

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

import edu.neu.coe.info6205.sort.BaseHelper;
import edu.neu.coe.info6205.sort.InstrumentedHelper;
import edu.neu.coe.info6205.sort.linearithmic.MergeSortBasic;
import java.util.function.Consumer;
import java.util.function.Supplier;

public class MergeBenchmark {
    public static void main(String[] args) {
        for(int n=10000;n<=256000;n=n*2)
        {
            System.out.println("==============MergeSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "true",  seed: "0",  inversions: "0",  cutoff: "",  interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "MergeSortHelper", config);
            MergeSortBasic<Integer> merge = new MergeSortBasic<>(helper);
            Consumer<Integer[]> consumer = RanArr -> merge.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "MergeSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
                Integer[] randomArray = new Integer[finalN];
                for(int i = 0; i< finalN; i++)
                {
                    randomArray[i] = (int)(Math.random()*finalN+1);
                }
                return randomArray;
            };
            helper.postProcess(merge.sort(randomArrays.get()));
            PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
            StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");

            int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
            int hits = (int) statPack.getStatistics(InstrumentedHelper.HITS).mean();
            int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
            int copies = (int) statPack.getStatistics(InstrumentedHelper.COPIES).mean();

            System.out.println("==============MergeSort instrumentation Test "+n+" elements"+"================");
            System.out.println("Compares: " + compares);
            System.out.println("Hits : " + hits);
            System.out.println("fixes : " + fixes);
            System.out.println("copies : " + copies);
        }
        for(int n=10000;n<=256000;n=n*2)
        {
            System.out.println("==============MergeSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "false",  seed: "0",  inversions: "0",  cutoff: "",  interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "MergeSortHelper", config);
            MergeSortBasic<Integer> merge = new MergeSortBasic<>(helper);
            Consumer<Integer[]> consumer = RanArr -> merge.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "MergeSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
                Integer[] randomArray = new Integer[finalN];
                for(int i = 0; i< finalN; i++)
                {
                    randomArray[i] = (int)(Math.random()*finalN+1);
                }
```

```java
import edu.neu.coe.info6205.sort.BaseHelper;
import edu.neu.coe.info6205.sort.InstrumentedHelper;
import edu.neu.coe.info6205.sort.elementary.HeapSort;

import java.util.function.Consumer;
import java.util.function.Supplier;

public class HeapBenchmark {
    public static void main(String[] args) {
        for(int n=10000;n<=256000;n=n*2)
        {
            System.out.println("==============HeapSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "true",  seed: "0",  inversions: "1",  cutoff: "",  interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "HeapSortHelper1", config);
            HeapSort<Integer> heap = new HeapSort<>(helper);

            Consumer<Integer[]> consumer =RanArr -> heap.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "HeapSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
                Integer[] randomArray = new Integer[finalN];
                for(int i = 0; i< finalN; i++)
                {
                    randomArray[i] = (int)(Math.random()*finalN+1);
                }
                return randomArray;
            };
            helper.postProcess(heap.sort(randomArrays.get()));
            PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
            StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack");
            int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
            int hits = (int) statPack.getStatistics(InstrumentedHelper.HITS).mean();
            int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
            int copies = (int) statPack.getStatistics(InstrumentedHelper.COPIES).mean();
            int swaps = (int) statPack.getStatistics(InstrumentedHelper.SWAPS).mean();
            System.out.println("==============HeapSort instrumentation Test "+n+" elements"+"================");
            System.out.println("Compares: " + compares);
            System.out.println("Hits : " + hits);
            System.out.println("fixes : " + fixes);
            System.out.println("copies : " + copies);
            System.out.println("Swaps: " + swaps);
        }

        for(int n=10000;n<=256000;n=n*2)
        {
            System.out.println("==============HeapSort Time Test "+n+" elements"+"==========================");
            final Config config = Config.setupConfig( instrumenting: "false",  seed: "0",  inversions: "1",  cutoff: "",  interimInversions: "");
            BaseHelper<Integer> helper = new InstrumentedHelper<>( description: "HeapSortHelper1", config);
            HeapSort<Integer> heap = new HeapSort<>(helper);

            Consumer<Integer[]> consumer =RanArr -> heap.sort(RanArr);
            Benchmark_Timer<Integer[]> BT = new Benchmark_Timer<>( description: "HeapSort Time Test " + n + " elements", consumer);
            int finalN = n;
            Supplier<Integer[]> randomArrays = () -> {
```

Screenshot of the Data result:





```
E:\Development\jdk\bin\java.exe ...
=============HeapSort Time Test 10000 elements==========================
=============HeapSort instrumentation Test 10000 elements===============
Compares: 235475
Hits : 967834
fixes : 75338009
copies : 0
Swaps: 124221
=============HeapSort Time Test 20000 elements==========================
=============HeapSort instrumentation Test 20000 elements===============
Compares: 510733
Hits : 2095014
fixes : 301923389
copies : 0
Swaps: 268387
=============HeapSort Time Test 40000 elements==========================
=============HeapSort instrumentation Test 40000 elements===============
Compares: 1101404
Hits : 4509968
fixes : 1209951669
copies : 0
Swaps: 576790
=============HeapSort Time Test 80000 elements==========================
=============HeapSort instrumentation Test 80000 elements===============
Compares: 2362698
Hits : 9659408
fixes : 543731184
copies : 0
Swaps: 1233503
```

```
=============HeapSort Time Test 160000 elements==========================
=============HeapSort instrumentation Test 160000 elements===============
Compares: 5046192
Hits : 20600992
fixes : -2105979274
copies : 0
Swaps: 2627152
=============HeapSort Time Test 10000 elements==========================
2023-03-13 04:02:27 INFO  Benchmark_Timer - Begin run: HeapSort Time Test 10000 elements with 1 runs
HeapSort Time Test 10000 elements : 1.0
=============HeapSort Time Test 20000 elements==========================
2023-03-13 04:02:27 INFO  Benchmark_Timer - Begin run: HeapSort Time Test 20000 elements with 1 runs
HeapSort Time Test 20000 elements : 4.0
=============HeapSort Time Test 40000 elements==========================
2023-03-13 04:02:27 INFO  Benchmark_Timer - Begin run: HeapSort Time Test 40000 elements with 1 runs
HeapSort Time Test 40000 elements : 8.0
=============HeapSort Time Test 80000 elements==========================
2023-03-13 04:02:27 INFO  Benchmark_Timer - Begin run: HeapSort Time Test 80000 elements with 1 runs
HeapSort Time Test 80000 elements : 17.0
=============HeapSort Time Test 160000 elements==========================
2023-03-13 04:02:27 INFO  Benchmark_Timer - Begin run: HeapSort Time Test 160000 elements with 1 runs
HeapSort Time Test 160000 elements : 39.0
```

```
E:\Development\jdk\bin\java.exe ...
=============MergeSort Time Test 10000 elements==========================
=============MergeSort instrumentation Test 10000 elements===============
Compares: 121547
Hits : 489950
fixes : 25095435
copies : 220000
=============MergeSort Time Test 20000 elements==========================
=============MergeSort instrumentation Test 20000 elements===============
Compares: 263191
Hits : 1060490
fixes : 100062644
copies : 480000
=============MergeSort Time Test 40000 elements==========================
=============MergeSort instrumentation Test 40000 elements===============
Compares: 565661
Hits : 2278506
fixes : 402024867
copies : 1040000
=============MergeSort Time Test 80000 elements==========================
=============MergeSort instrumentation Test 80000 elements===============
Compares: 1212487
Hits : 4879154
fixes : 1600467513
copies : 2240000
=============MergeSort Time Test 160000 elements==========================
=============MergeSort instrumentation Test 160000 elements===============
Compares: 2584609
Hits : 10397534
fixes : 2107667697
copies : 4800000
=============MergeSort Time Test 10000 elements==========================
2023-03-13 04:04:20 INFO  Benchmark_Timer - Begin run: MergeSort Time Test 10000 elements with 1 runs
MergeSort Time Test 10000 elements : 2.222
=============MergeSort Time Test 20000 elements==========================
2023-03-13 04:04:20 INFO  Benchmark_Timer - Begin run: MergeSort Time Test 20000 elements with 1 runs
MergeSort Time Test 20000 elements : 3.1381
=============MergeSort Time Test 40000 elements==========================
2023-03-13 04:04:20 INFO  Benchmark_Timer - Begin run: MergeSort Time Test 40000 elements with 1 runs
MergeSort Time Test 40000 elements : 6.7945
=============MergeSort Time Test 80000 elements==========================
2023-03-13 04:04:21 INFO  Benchmark_Timer - Begin run: MergeSort Time Test 80000 elements with 1 runs
MergeSort Time Test 80000 elements : 17.0486
=============MergeSort Time Test 160000 elements==========================
2023-03-13 04:04:21 INFO  Benchmark_Timer - Begin run: MergeSort Time Test 160000 elements with 1 runs
MergeSort Time Test 160000 elements : 32.5945
```

Screenshot of the Charts:

HeapSort

Size	Compares	Hits	Fixes	Copies	Swaps	Time
10000	235607	968562	75966412	0	124337	2
20000	510834	2095116	302057390	0	268362	4
40000	1101344	4510056	1210298222	0	576842	7
80000	2362666	9658376	542466792	0	1233261	22
160000	5045902	20599304	2102559515	0	2626875	37

QuickSort

Size	Compares	Hits	Fixes	Copies	Swaps	Time
10000	158697	422734	26866898	0	65333	1
20000	329733	892384	106798253	0	139323	3
40000	743421	1962586	422823543	0	302100	10
80000	1557990	4032642	1634286905	0	613124	10
160000	3303625	8706814	1723035691	0	1339646	25

MergeSort

Size	Compares	Hits	Fixes	Copies	Time
10000	121547	489950	25095435	220000	2.222
20000	263191	1060490	100062644	480000	3.1381
40000	565661	2278506	402024867	1040000	6.7945
80000	1212487	4879154	1600467513	2240000	17.0486
160000	2584609	10397534	2107667697	4800000	32.5945

HeapSort(LOG10)

Size	Compares	Hits	Fixes	Copies	Swaps	Time
10000	5.3722	5.9861	7.880621615	0	5.0946	2
20000	5.7083	6.3212	8.480089465	0	5.4287	4
40000	6.0419	6.6542	9.0829	0	5.7611	7
80000	6.3734	6.984904108	8.7344	0	6.0911	22
160000	6.7029	7.3139	9.3227	0	6.4194	37

QuickSort(LOG10)

Size	Compares	Hits	Fixes	Copies	Swaps	Time
10000	5.2006	5.6261	7.4292	0	4.8151	1
20000	5.5182	5.9506	8.0286	0	5.144	3
40000	5.8712	6.2928	8.6262	0	5.4802	10
80000	6.1926	6.6056	9.2133	0	5.7875	10
160000	6.519	6.9399	9.2363	0	6.127	25

MergeSort(LOG10)

Size	Compares	Hits	Fixes	Copies	Time
10000	5.0847	5.6902	7.3996	5.3424	2.222
20000	5.4203	6.0255	8.0003	5.6812	3.1381
40000	5.7526	6.3577	8.6043	6.017	6.7945
80000	6.0837	6.6883	9.2042	6.3502	17.0486
160000	6.4124	7.0169	9.3238	6.6812	32.5945

Conclusion：

From the analysis of the data, The best predictor for execution time varies for different sorting algorithms. For MergeSort, the data shows that the number of Swaps/Copies fits Execution Time perfectly, which means that the number of Swaps/Copies determines the Time. For QuickSort, the data shows that the number of Compares and Swaps determine the Execution Time. For HeapSort, the data shows the number of Compares and Swaps determine the Execution Time.This indicates that each sorting algorithm may have different aspects that affect their performance. . It can be observed that as the size of the input array increases, the execution time for all three sorting algorithms increases exponentially.5. Quick sort and HeapSort have similar performance in terms of execution time, but HeapSort requires a slightly higher number of swaps and copies compared to QuickSort. This suggests that QuickSort may be a better choice in scenarios where memory usage is a concern. From the given data, it can be concluded that Merge sort is the most efficient algorithm among the three sorting algorithms.