

Program Structures and Algorithms  
Spring 2023(SEC 01)

NAME: Shivani Datar  
NUID: 002772160

**Task:**

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).

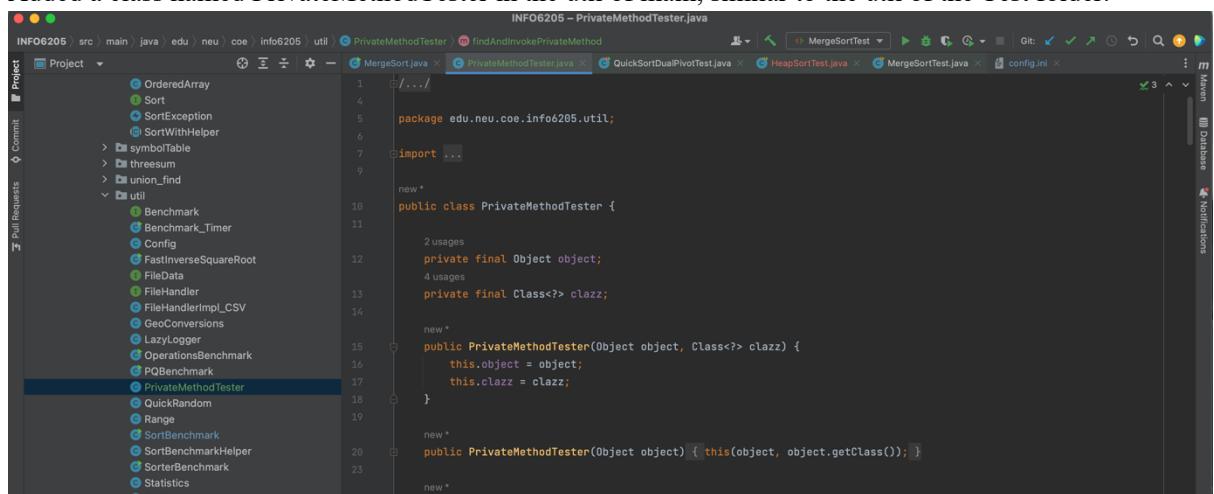
All of the code to count comparisons, swaps/copies, and hits, is already implemented in the *InstrumentedHelper* class. You can see examples of the usage of this kind of analysis in:

- 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).

**Output Screenshots:-**

**Code Changes:-**

1. Added a class named PrivateMethodTester in the util of main, similar to the util of the Test folder.



```
INFO6205 - PrivateMethodTester.java
INFO6205 src main java edu neu coe info6205 util
  PrivateMethodTester.java
    package edu.neu.coe.info6205.util;
    import ...
    new*
    public class PrivateMethodTester {
        2 usages
        private final Object object;
        4 usages
        private final Class<?> clazz;
        new*
        public PrivateMethodTester(Object object, Class<?> clazz) {
            this.object = object;
            this.clazz = clazz;
        }
        new*
        public PrivateMethodTester(Object object) { this(object, object.getClass()); }
    }
    new*
```

This helps for the instrumentation execution required for the sorts.

2. Added following code in the mergeSort for noCopy and Insurance.

```

5 usages  • Shivani Datar *
private void sort(X[] a, X[] aux, int from, int to) {
    final Helper<X> helper = getHelper();
    Config config = helper.getConfig();
    boolean insurance = config.getBoolean(MERGESORT, INSURANCE);
    boolean noCopy = config.getBoolean(MERGESORT, NOCOPY);
    if (to <= from + helper.cutoff()) {
        insertionSort.sort(a, from, to);
        return;
    }
    // FIXME : implement merge sort with insurance and no-copy optimizations
    int middle = from + (to - from) / 2;
    if (noCopy) {
        sort(aux, a, from, middle);
        sort(aux, a, middle, to);
        if (insurance && helper.less(aux, [middle - 1], middle)) {
            System.arraycopy(aux, from, a, from, [length: to - from]);
            helper.incrementCopies([n: to - from]);
        } else
            merge(aux, a, from, middle, to);
    } else {
        sort(a, aux, from, middle);
        sort(a, aux, middle, to);
        System.arraycopy(a, from, aux, from, [length: to - from]);
        if (insurance && helper.less(a[middle - 1], a[middle])) return;
        merge(aux, a, from, middle, to);
    }
}

// END

```

3. Added a main method in the MergeSort.java file, it has a Config c set which is later on used for all objects of the sorts. The elements in the random array generation has a bound 10000. The size of array starts from 10,000 until 160,000. This is generated using a for-loop. The sorting method is called out twice, once for the timing purposes, where I am using the Benchmark class object and passing either mergesort/quicksort/heapsort object's sort method in the fRun which is executed for 20 number of runs. The second call is done for the sorting method, the helper is generated while creation of the sort object, that helper is preprocess is done before the Benchmarking and after the second call postprocess is done on the helper, which enables us to get the statPack associated while sorting the xs array into the final sorted ys array. Then from this statPack we are getting our metrics involved which are number of compares, swaps and array hits, number of copies( in case of the merge Sort). Following is the code for the main method,

```

public static void main(String[] args){
    //Random rand = new Random();
    Config c = Config.setupConfig( instrumenting: "true", seed: "", inversions: "0", cutoff: "1", interimInversions: "" );
    final int num = 10000;
    // Merge Sort
    for (int n = 10000; n <= 256000; n = n * 2) {
        System.out.println("With N = "+n);
        final Helper<Integer> helper = HelperFactory.create( description: "Mergesort", n, c );
        final Integer[] xs = helper.random(Integer.class, r -> r.nextInt(num));
        System.out.println(helper);
        Sort<Integer> mergeSort = new MergeSort<>(helper);
        mergeSort.init(n);
        helper.preProcess(xs);
        Benchmark<Boolean> benchmarkRandom = new Benchmark_Timer<>(
            description: "MergeSort", b -> {
                mergeSort.sort(xs.clone(), from: 0, xs.length);
            });
        double resultRandom = benchmarkRandom.run( t: true, m: 20 );
        Integer[] ys = mergeSort.sort(xs);
        helper.postProcess(ys);
        PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "getStatPack" );
        //InstrumentedHelper stat = new InstrumentedHelper("MergeSort",n,c);
        int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
        int swaps = (int) statPack.getStatistics(InstrumentedHelper.SWAPS).mean();
        int hits = (int) statPack.getStatistics(InstrumentedHelper.HITS).mean();
        int copies = (int) statPack.getStatistics(InstrumentedHelper.COPIES).mean();
        System.out.println(" Time taken for Merge sort with random array : " + resultRandom );
        System.out.println("Number of compares = "+compares);

        //Quick sort dual pivot

        final BaseHelper<Integer> helper1 = (BaseHelper<Integer>) HelperFactory.create( description: "quick sort dual pivot", n, c );
        //System.out.println(helper1);
        Sort<Integer> quicksort = new QuickSort_DualPivot<>(helper1);
        quicksort.init(n);
        //final int f = 500000;
        final Integer[] xs1 = helper1.random(Integer.class, r -> r.nextInt(num));
        System.out.println(helper1);
        helper1.preProcess(xs1);
        Benchmark<Boolean> benchmarkRandom1 = new Benchmark_Timer<>(
            description: "QuickDualPivotSort", b -> {
                quicksort.sort(xs1);
            });
        double resultRandom1 = benchmarkRandom1.run( t: true, m: 20 );
        Integer[] ys1 = quicksort.sort(xs1);
        helper1.postProcess(ys1);
        final PrivateMethodTester privateMethodTester1 = new PrivateMethodTester(helper1);
        final StatPack statPack1 = (StatPack) privateMethodTester1.invokePrivate( name: "getStatPack" );
        int compares1 = (int) statPack1.getStatistics(InstrumentedHelper.COMPARES).mean();
        int swaps1 = (int) statPack1.getStatistics(InstrumentedHelper.SWAPS).mean();
        int hits1 = (int) statPack1.getStatistics(InstrumentedHelper.HITS).mean();
        System.out.println(" Time taken for Quick sort with random array : " + resultRandom1 );
        System.out.println("Number of compares = "+compares1);
    }
}

```

```
System.out.println("Number of swaps = "+swaps1);
System.out.println("Number of array accesses = "+hits1);

//Heap Sort

Helper<Integer> helper2 = HelperFactory.create( description: "HeapSort", n, c);
final Integer[] randomArray = helper2.random(Integer.class, r -> r.nextInt(num));
helper2.init(n);
SortWithHelper<Integer> heapsort = new HeapSort<Integer>(helper2);
heapsort.preProcess(randomArray);
Benchmark<Boolean> benchmarkRandom2 = new Benchmark_Timer<>(
    description: "HeapSort", b -> {
    heapsort.sort(randomArray);
});
double resultRandom2 = benchmarkRandom2.run( t: true, m: 20);
PrivateMethodTester privateMethodTester2 = new PrivateMethodTester(helper2);
StatPack statPack2 = (StatPack) privateMethodTester2.invokePrivate( name: "getStatPack");
Integer[] ys2 = heapsort.sort(randomArray);
helper2.postProcess(ys2);
int compares2 = (int) statPack2.getStatistics(InstrumentedHelper.COMPARES).mean();
int swaps2 = (int) statPack2.getStatistics(InstrumentedHelper.SWAPS).mean();
int hits2 = (int) statPack2.getStatistics(InstrumentedHelper.HITS).mean();
System.out.println(" Time taken for Heap sort with random array : " + resultRandom2);
System.out.println("Number of compares = "+comprises2);
System.out.println("Number of swaps = "+swaps2);
System.out.println("Number of array accesses = "+hits2);
```

While the time of execution 2 sorts were commented and only 1 sort was executed for the code efficiency. Following are the outputs generated for Merge, Quick and Heap Sort.

### 1. Merge Sort Output :-

The screenshot shows an IDE interface with the following details:

- Project:** INFO6205
- File:** MergeSort.java
- Code Snippet:** The code implements a merge sort algorithm with a complexity of  $O(n \log n)$ . It includes a helper class and a main loop for generating and sorting arrays of increasing size.

```
for (int n = 10000; n <= 256000; n = n * 2) {
    System.out.println("With N = " + n);
    final Helper<Integer> helper = HelperFactory.create(description: "Mergesort", n, o);
    final Integer[] xs = helper.random(Integer.class, r -> r.nextInt(num));
    System.out.println(helper);
    Sort<Integer> mergeSort = new MergeSort<>(helper);
```

- Run Output:** The output shows the execution of the program for array sizes 10,000, 20,000, 40,000, and 80,000. For each size, it prints the value of N, creates a helper object, generates a random array, and prints the helper object's state.

```
With N = 10000
Instrumenting helper for Mergesort with 10,000 elements
2023-03-12 20:58:43 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 7.357027050000001
Number of compares = 120477
Number of swaps = 0
Number of array accesses = 267232
Number of copies = 133616
With N = 20000
Instrumenting helper for Mergesort with 20,000 elements
2023-03-12 20:58:44 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 8.1472354
Number of compares = 260884
Number of swaps = 0
Number of array accesses = 574464
Number of copies = 287232
With N = 40000
Instrumenting helper for Mergesort with 40,000 elements
2023-03-12 20:58:44 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 16.132189500000003
Number of compares = 561772
Number of swaps = 0
Number of array accesses = 1228928
Number of copies = 614464
With N = 80000
```

- Bottom Bar:** Includes tabs for Git, Run, Debug, TODO, Problems, Terminal, Services, Profiler, Build, and Dependencies. A status bar at the bottom indicates the build was successful in 3 seconds.

INFO6205 - MergeSort.java

```

INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSort > config.ini
Project Run: MergeSort > config.ini > MergeSort.java
ComparableTuple 118
Complex 119
Counter 120
HuffmanCoding 121
Iteration 122
Matrix 123
for (int n = 10000; n <= 256000; n = n * 2) {
    System.out.println("With N = " + n);
    final Helper<Integer> helper = HelperFactory.create(description: "Mergesort", n, e);
    final Integer[] xs = helper.randomInteger.class, r -> r.nextInt(n));
    System.out.println(helper);
    Sort<Integer> mergeSort = new MergeSort<>(helper);
}

Instrumenting helper for Mergesort with 40,000 elements
2023-03-12 20:58:44 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 16.132189500000003
Number of compares = 561772
Number of swaps = 0
Number of array accesses = 1228928
Number of copies = 614464
With N = 80000
Instrumenting helper for Mergesort with 80,000 elements
2023-03-12 20:58:44 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 31.699177
Number of compares = 1203502
Number of swaps = 0
Number of array accesses = 2617856
Number of copies = 1308928
With N = 160000
Instrumenting helper for Mergesort with 160,000 elements
2023-03-12 20:58:45 INFO Benchmark_Timer - Begin run: MergeSort with 20 runs
Time taken for Merge sort with random array : 64.0574798
Number of compares = 2567277
Number of swaps = 0
Number of array accesses = 5555712
Number of copies = 2777856

Process finished with exit code 0

```

Git Run Debug TODO Problems Terminal Services Profiler Build Dependencies

Build completed successfully in 3 sec, 239 ms (a minute ago)

44:1 LF UTF-8 4 spaces main

### Quick Dual Pivot Sort Output :-

INFO6205 - MergeSort.java

```

INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSort > config.ini > HeapSortTest.java > QuickSortDualPivotTest.java
Project Run: MergeSort > config.ini > MergeSort.java
Matrix 154
MyDate
System.out.println(helper);
With N = 10000
Instrumenting helper for quick sort dual pivot with 10,000 elements
2023-03-03 13:21:36 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 231.1030667
Number of compares = 158452
Number of swaps = 72207
Number of array accesses = 460821
With N = 20000
Instrumenting helper for quick sort dual pivot with 20,000 elements
2023-03-03 13:21:42 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 1451.22262095
Number of compares = 420077
Number of swaps = 154985
Number of array accesses = 1066498
With N = 40000
Instrumenting helper for quick sort dual pivot with 40,000 elements
2023-03-03 13:22:15 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 3391.84108755
Number of compares = 750116
Number of swaps = 331705
Number of array accesses = 2132387
With N = 80000
Instrumenting helper for quick sort dual pivot with 80,000 elements
2023-03-03 13:23:33 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 11374.96573775
Number of compares = 1779383
Number of swaps = 632082
Number of array accesses = 4422928
With N = 160000

```

Git Run Debug TODO Problems Terminal Services Profiler Build Dependencies

Build completed successfully in 3 sec, 596 ms (today 1:21 PM)

100:2 LF UTF-8 4 spaces main

IntelliJ IDEA 2023.1.3

INFO6205 - MergeSort.java

```
INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSort
Project Run: MergeSort x
Instrumenting helper for quick sort dual pivot with 20,000 elements
2023-03-03 13:21:42 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 1451.22262095
Number of compares = 420677
Number of swaps = 154985
Number of array accesses = 1066498
With N = 40000
Instrumenting helper for quick sort dual pivot with 40,000 elements
2023-03-03 13:22:15 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 3391.84108755
Number of compares = 750116
Number of swaps = 331765
Number of array accesses = 2132387
With N = 80000
Instrumenting helper for quick sort dual pivot with 80,000 elements
2023-03-03 13:23:33 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 11374.96573775
Number of compares = 1779383
Number of swaps = 632882
Number of array accesses = 4422928
With N = 160000
Instrumenting helper for quick sort dual pivot with 160,000 elements
2023-03-03 13:27:53 INFO Benchmark_Timer - Begin run: QuickDualPivotSort with 20 runs
Time taken for Quick sort with random array : 66846.49938945
Number of compares = 4121721
Number of swaps = 1275698
Number of array accesses = 9459691
Process finished with exit code 0
```

Git Run Debug TODO Problems Terminal Services Profiler Build Dependencies

Build completed successfully in 3 sec, 598 ms (today 1:21 PM) 199/2 LF UTF-8 4 spaces main

## Heap Sort Output :-

IntelliJ IDEA 2023.1.3

INFO6205 - MergeSort.java

```
INFO6205 > src > main > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSort > main
Project Run: MergeSort x
GenericSort
GenericSortWithGenericHelper
heapsort.sort(randomArray);
```

Commit

Run: MergeSort x

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
With N = 10000
2023-03-12 11:53:04 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 306.26634789999997
Number of compares = 235460
Number of swaps = 124298
Number of array accesses = 968112
With N = 20000
2023-03-12 11:53:11 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 1292.82618755
Number of compares = 510735
Number of swaps = 268337
Number of array accesses = 2894818
With N = 40000
2023-03-12 11:53:40 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 5420.8616854
Number of compares = 1101413
Number of swaps = 576673
Number of array accesses = 4509518
With N = 80000
2023-03-12 12:09:11 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 22769.84913145
Number of compares = 2363155
Number of swaps = 1233747
Number of array accesses = 9661298
With N = 160000
2023-03-12 12:17:55 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 90969.34724795
Number of compares = 5046545
Number of swaps = 2627359
Number of array accesses = 28682526
Process finished with exit code 0
```

Git Run Debug TODO Problems Terminal Services Profiler Build Dependencies

Build completed successfully in 3 sec, 625 ms (today 11:53 AM) 34/1 LF UTF-8 4 spaces main

```

INFO6205 src main java edu neu coe info6205 sort linearithmic MergeSort.java main
INFO6205 - MergeSort.java
Project GenericSort GenericSortWithGenericHelper 182 heapsort.sort(randomArray);

Run: MergeSort
Time taken for Heap sort with random array : 306.26634789999997
Number of compares = 235460
Number of swaps = 124298
Number of array accesses = 968112
With N = 20000
2023-03-12 11:53:11 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 1292.82618755
Number of compares = 510735
Number of swaps = 268337
Number of array accesses = 2094818
With N = 40000
2023-03-12 11:53:40 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 5420.8616854
Number of compares = 1101413
Number of swaps = 576673
Number of array accesses = 4509518
With N = 80000
2023-03-12 12:09:11 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 22769.84913145
Number of compares = 2363155
Number of swaps = 1233747
Number of array accesses = 9661298
With N = 160000
2023-03-12 12:17:55 INFO Benchmark_Timer - Begin run: HeapSort with 20 runs
Time taken for Heap sort with random array : 90969.34724795
Number of compares = 5046545
Number of swaps = 2627359
Number of array accesses = 28602526

Process finished with exit code 0

```

Build completed successfully in 2 sec, 825 ms (today 11:53 AM)

### Relationship Conclusion :-

From the graphs we can say that the best predictor for the time is number of hits, as number of hits is the best fit for the graph against the time.

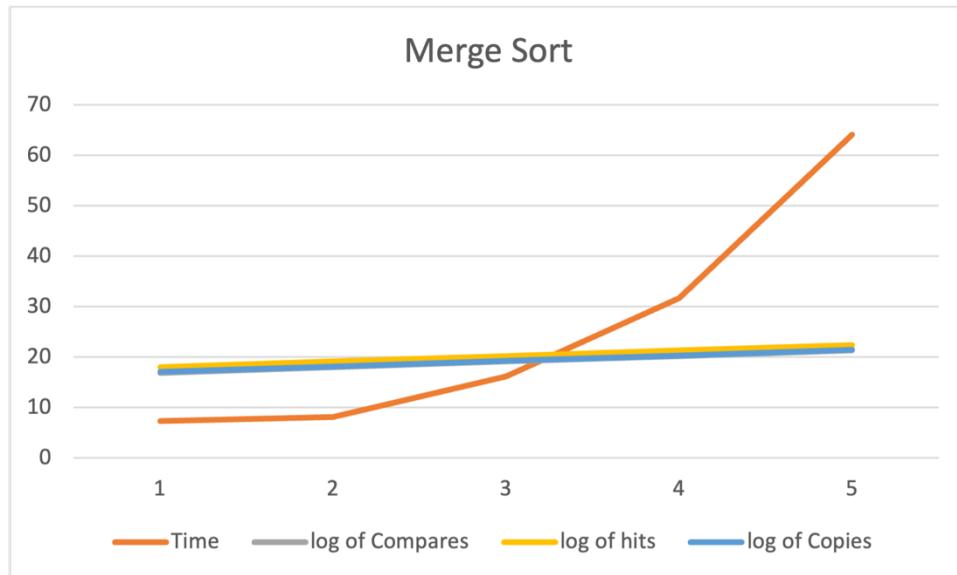
### Evidence to support that conclusion and Graphical Representation:

For merge Sort following is the table for N, time taken for 20 runs and number of compares, swaps, hits and copies, followed by the table with log values of compares, hits and copies to the base 2.

N	Time	Compares	Swaps	Hits	Copies
10000	7.35702705	120477	0	267232	133616
20000	8.1472354	260884	0	574464	287232
40000	16.1321895	561772	0	1228928	614464
80000	31.699177	1203502	0	2617856	1308928
160000	64.0574708	2567277	0	5555712	2777856

N	Time	log of Compares	log of hits	log of Copies
10000	7.35702705	16.87839823	18.0277333	17.0277333
20000	8.1472354	17.99304894	19.131857	18.131857
40000	16.1321895	19.09962519	20.228969	19.228969
80000	31.699177	20.19880711	21.3199543	20.3199543
160000	64.0574708	21.29180754	22.4055404	21.4055404

Graph for the Merge Sort:- taking time with the log values of the metrics involved.



For Quick Sort with Dual Pivot :-

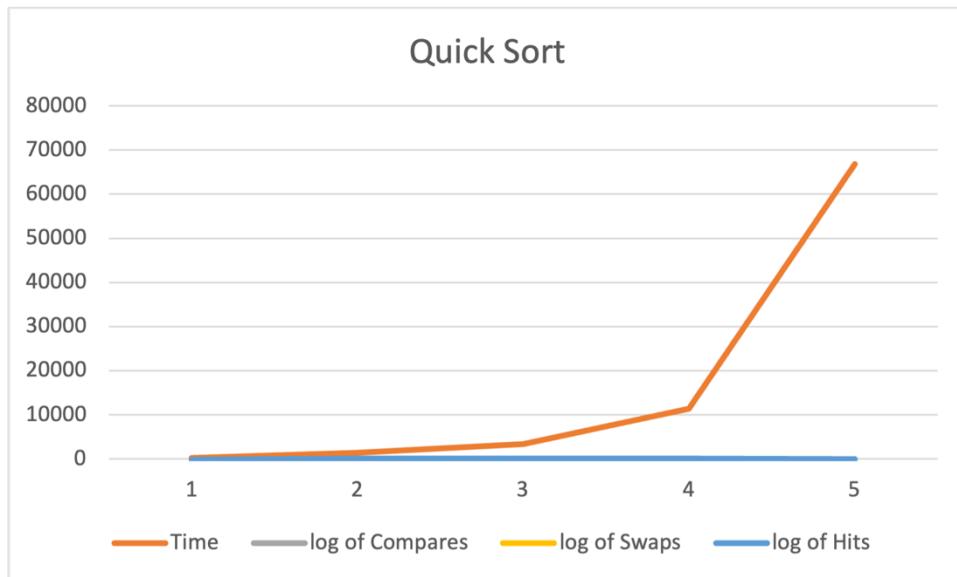
For quick Sort following is the table for  $N$ , time taken for 20 runs and number of compares, swaps, hits and followed by the table with log values of compares, swaps and hits to the base 2.

$N$	Time	Compares	Swaps	Hits
10000	231.103067	158452	72207	460021
20000	1451.22262	420077	154985	1066498
40000	3391.84109	750116	331705	2132387
80000	11374.9657	1779383	632082	4422928
160000	66846.4994	4121721	1275698	9459691

$N$	Time	$\log$ of Compares	$\log$ of Swaps	$\log$ of Hits
10000	231.103067	17.27368634	16.1398511	18.8113402
20000	1451.22262	18.68029427	17.2417691	20.0244498
40000	3391.84109	19.51675419	18.3395412	21.0240379
80000	11374.9657	20.76294564	19.2697522	22.0765703
160000	66846.4994	21.97481542	20.2828554	23.1733616

Graph for Quick Sort Dual Pivot :- taking time with the log values of the metrics involved.

Here we can see  $\log$  of compares, swaps and hits are close by and coincide with each other, but the  $\log$  of Hits has highest value, making it best fit with the time graph.



For Heap Sort:-

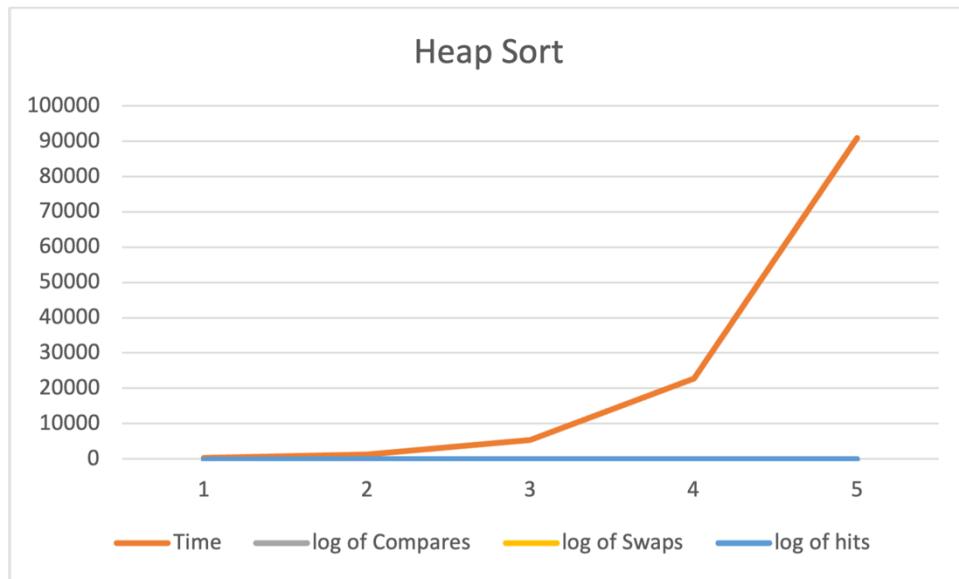
For Heap Sort following is the table for N, time taken for 20 runs and number of compares, swaps, hits and followed by the table with log values of compares, swaps and hits to the base 2.

N	Time	Comapres	Swaps	Hits
10000	306.266348	235460	124298	968112
20000	1292.82619	510735	268337	2094818
40000	5420.86169	1101413	576673	4509518
80000	22769.8491	2363155	1233747	9661298
160000	90969.3473	5046545	2627359	20602526

N	Time	log of Comparisons	log of Swaps	log of hits
10000	306.266348	17.84512247	16.9234436	19.8848144
20000	1292.82619	18.9622154	18.0336865	20.9983935
40000	5420.86169	20.07092411	19.137394	22.1045418
80000	22769.8491	21.17228283	20.2346152	23.2037856
160000	90969.3473	22.26686459	21.3251819	24.2963179

Graph for Heap Sort:- taking time with the log values of the metrics involved.

Here we can see log of compares, swaps and hits are close by and coincide with each other, but the log of Hits has highest value, making it best fit with the time graph.



### Unit Test Cases:

### Screenshot for MergeSort Test Cases:-

INFO6205 - MergeSortTest.java

```

INFO6205 > src > test > java > edu > neu > coe > info6205 > sort > linearithmic > MergeSortTest
Project Commit Pull Requests Bookmarks Structure Git Services Terminal Profiler Build Dependencies
Run: MergeSortTest > Tests passed: 15 of 15 tests – 541ms
MergeSortTest (edu.neu.coe.info6205.sort.linearithmic)
  ✓ testSort1_partialsorted 190 ms
  ✓ testSort1_partialsorted 68 ms
  ✓ testSort1 7 ms
  ✓ testSort2 15 ms
  ✓ testSort3 5 ms
  ✓ testSort4 79 ms
  ✓ testSort5 22 ms
  ✓ testSort6 29 ms
  ✓ testSort7 20 ms
  ✓ testSort10_partialsorted 38 ms
  ✓ testSort8_partialsorted 35 ms
  ✓ testSort12 28 ms
  ✓ testSort13 2 ms
  ✓ testSort14 1 ms
  ✓ testSort1a 2 ms
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance + NoCopy: 142660
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + NoCopy: 60298
Instrumenting helper for merge sort with 128 elements
StatPack {hits: 1,684, normalized=2.711; copies: 640, normalized=1.030; inversions: 4,224, normalized=6.801; swaps: 101, normalized=0.163; compares: 751}
Worst Comparisons769
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}
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff: 75778
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + NoCopy: 19330
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance: 26071
Instrumenting helper for insertion sort with 128 elements
average time random_Cutoff + Insurance + NoCopy: 19964
Instrumenting helper for insertion sort with 128 elements
partial sorted average time partialsorted_Cutoff + Insurance: 35202
Instrumenting helper for insertion sort with 128 elements

```

### Screenshot for Quick Sort Dual Pivot Test Cases:-

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** Shows the project structure with packages like `InsertionSortTest`, `RandomSortTest`, `SelectionSortTest`, `ShellSortTest`, `hashCode`, `linearithmic`, `IntroSortTest`, `MergeSortTest`, `QuickSort3WayTest`, and `QuickSort BasicTest`.
- Code Editor:** The `QuickSortDualPivotTest.java` file is open, containing Java code for a quick sort algorithm with dual pivot support.
- Run Tab:** The `QuickSortDualPivotTest` configuration is selected, and the output window shows the following test results:
  - 15 tests passed in 88ms.
  - Instrumenting helper for quick sort dual pivot with 128 elements.
  - StatPack metrics: hits: 2,619, normalized=4.217; copies: 0, normalized=0.000; inversions: 4,224, normalized=6.801; swaps: 435, normalized=0.768; compares: 950, worstComparisons: 1242.
  - Process finished with exit code 0.
- Bottom Navigation:** Includes tabs for Git, Run, Debug, TODO, Problems, Terminal, Services, Profiler, Build, and Dependencies.
- Status Bar:** Shows the time as 27:41, file format as LF, encoding as UTF-8, and code space as 4 spaces.

### **Screenshot for HeapSort Test cases:-**

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "INFO6205". The "elementary" package contains several test classes: BubbleSortTest, HeapSortTest, InsertionSortMSDTest, InsertionSortOptTest, InsertionSortTest, RandomSortTest, SelectionSortTest, and ShellSortTest.
- Code Editor:** The current file is "HeapSortTest.java". The code defines a public class "HeapSortTest" with a single test method "sort0".
- Run Tab:** The "Run" tab shows the test results for "HeapSortTest":
  - Tests passed: 5 of 5 tests - 283 ms
  - testMutatingHeapSort (edu.neu.coe.info6.HeapSortTest) took 235ms
  - sort0 (edu.neu.coe.info6.HeapSortTest) took 22ms
  - sort1 (edu.neu.coe.info6.HeapSortTest) took 12ms
  - sort2 (edu.neu.coe.info6.HeapSortTest) took 11ms
  - sort3 (edu.neu.coe.info6.HeapSortTest) took 3ms
- Bottom Status Bar:** Shows "Tests passed: 5 (moments ago)" and the system status: 22:14 LF UTF-8 4 spaces.