Program Structures and Algorithms
Spring 2023(SEC – 01)

NAME: Kasavaraju Venkata Pranay Kumar
NUID: 002701155

Task:

Determine--for sorting algorithms--what is the best predictor of total execution time: comparisons, swaps/copies, hits (array accesses)

Relationship Conclusion:

Dual pivot quick sort is better than merge sort and merge sort is better than heap sort. But, as the size increases, merge sort is comparable to dual pivot quick sort. Time is directly proportional to compares, swaps and hits. Although swap is a costly operation and number of hits has higher count, comparing the number of compares and size of the array contributes more to the raw time taken.

Evidence to support that conclusion:

Observations:

1) Merge sort

Size	**Time**	**Normalize time**	**Compares**	**Swaps**	**Hits**	**Log(size)**
16000	3.58	2.99	206767	14017	434704	13.96578428
32000	8.67	3.14	445486	28027	937504	14.96578428
64000	16.58	2.97	955810	56077	1994805	15.96578428
128000	37.71	3.16	2043518	112173	4245685	16.96578428
256000	79.29	3.14	4,303,219	224334	9003208	17.96578428

2) (dual-pivot) quick sort

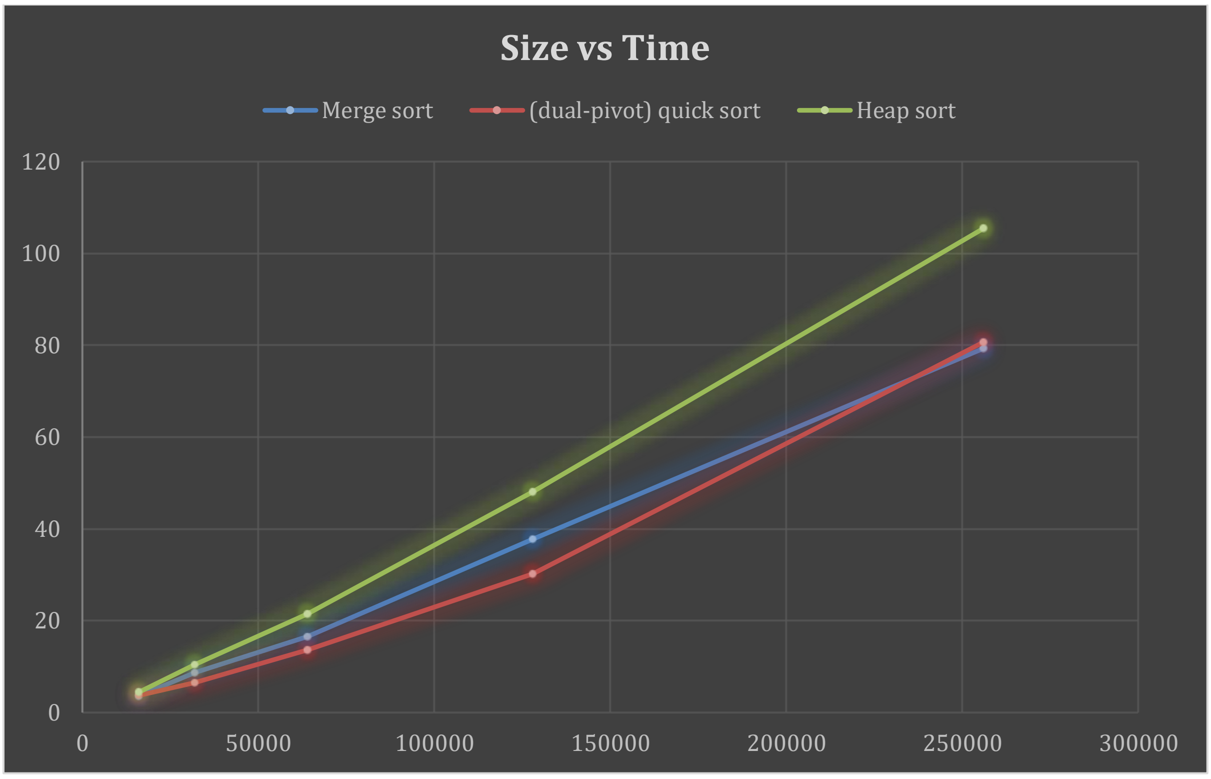
Size	**Time**	**Normalize time**	**Compares**	**Swaps**	**Hits**	**Log(size)**
16000	3.72	2.72	264,768	111,368	702,986	13.96578428
32000	6.52	2.51	574,025	240,782	1,522,039	14.96578428
64000	13.66	2.45	1,234,828	512,460	3,255,929	15.96578428
128000	30.27	2.54	2,649,857	1,115,510	7,051,684	16.96578428
256000	80.65	3.18	5,667,442	2,385,104	15,093,864	17.96578428

3)Array size: 500000

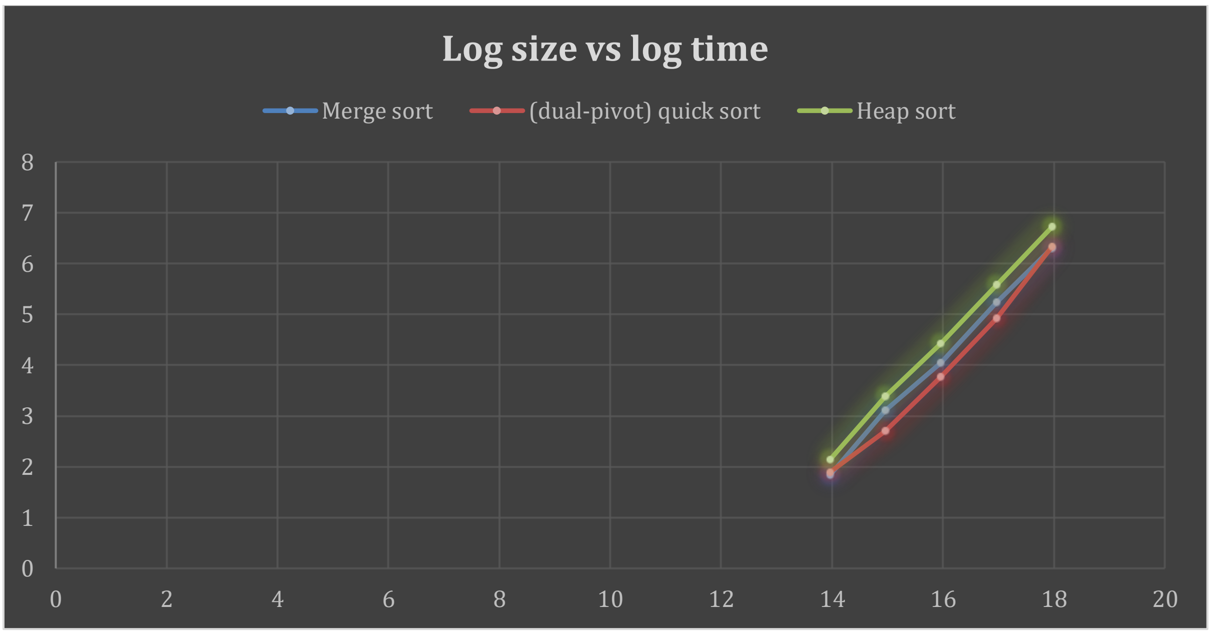
Size	**Time**	**Normalize time**	**Compares**	**Swaps**	**Hits**	**Log(size)**
16000	4.42	3.67	397,646	209,248	1,632,282	13.96578428
32000	10.44	4.02	859,312	450,508	3,520,656	14.96578428
64000	21.5	3.86	1,846,651	965,040	7,553,463	15.96578428
128000	48.06	4.03	3,949,237	2,057,957	16,130,303	16.96578428
256000	105.42	4.15	8,410,502	4,372,027	34,309,112	17.96578428

Graphical Representation:

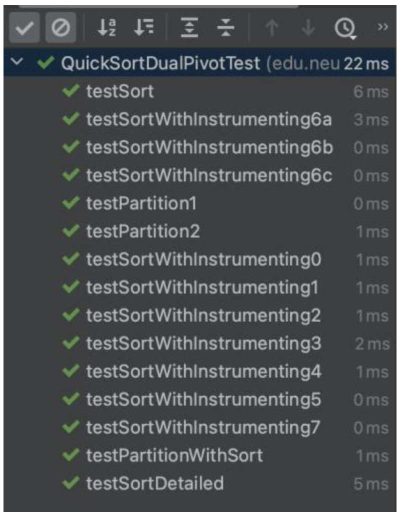
1. Size vs time



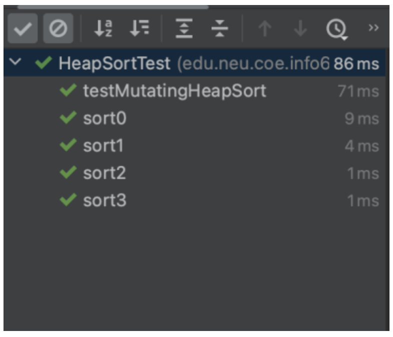
2. log(Size) vs log(time)



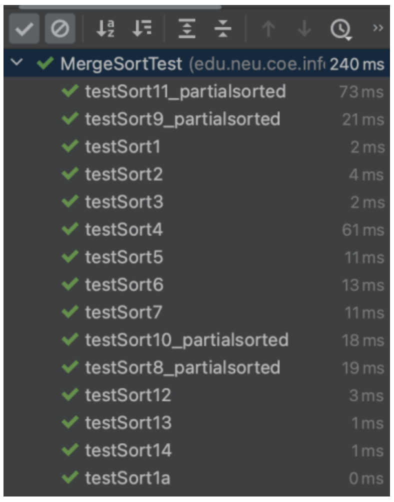
Unit Test Screenshots:



```
✓ QuickSortDualPivotTest (edu.neu 22 ms
    ✓ testSort                          6 ms
    ✓ testSortWithInstrumenting6a       3 ms
    ✓ testSortWithInstrumenting6b       0 ms
    ✓ testSortWithInstrumenting6c       0 ms
    ✓ testPartition1                    0 ms
    ✓ testPartition2                    1 ms
    ✓ testSortWithInstrumenting0        1 ms
    ✓ testSortWithInstrumenting1        1 ms
    ✓ testSortWithInstrumenting2        1 ms
    ✓ testSortWithInstrumenting3        2 ms
    ✓ testSortWithInstrumenting4        1 ms
    ✓ testSortWithInstrumenting5        0 ms
    ✓ testSortWithInstrumenting7        0 ms
    ✓ testPartitionWithSort             1 ms
    ✓ testSortDetailed                  5 ms
```



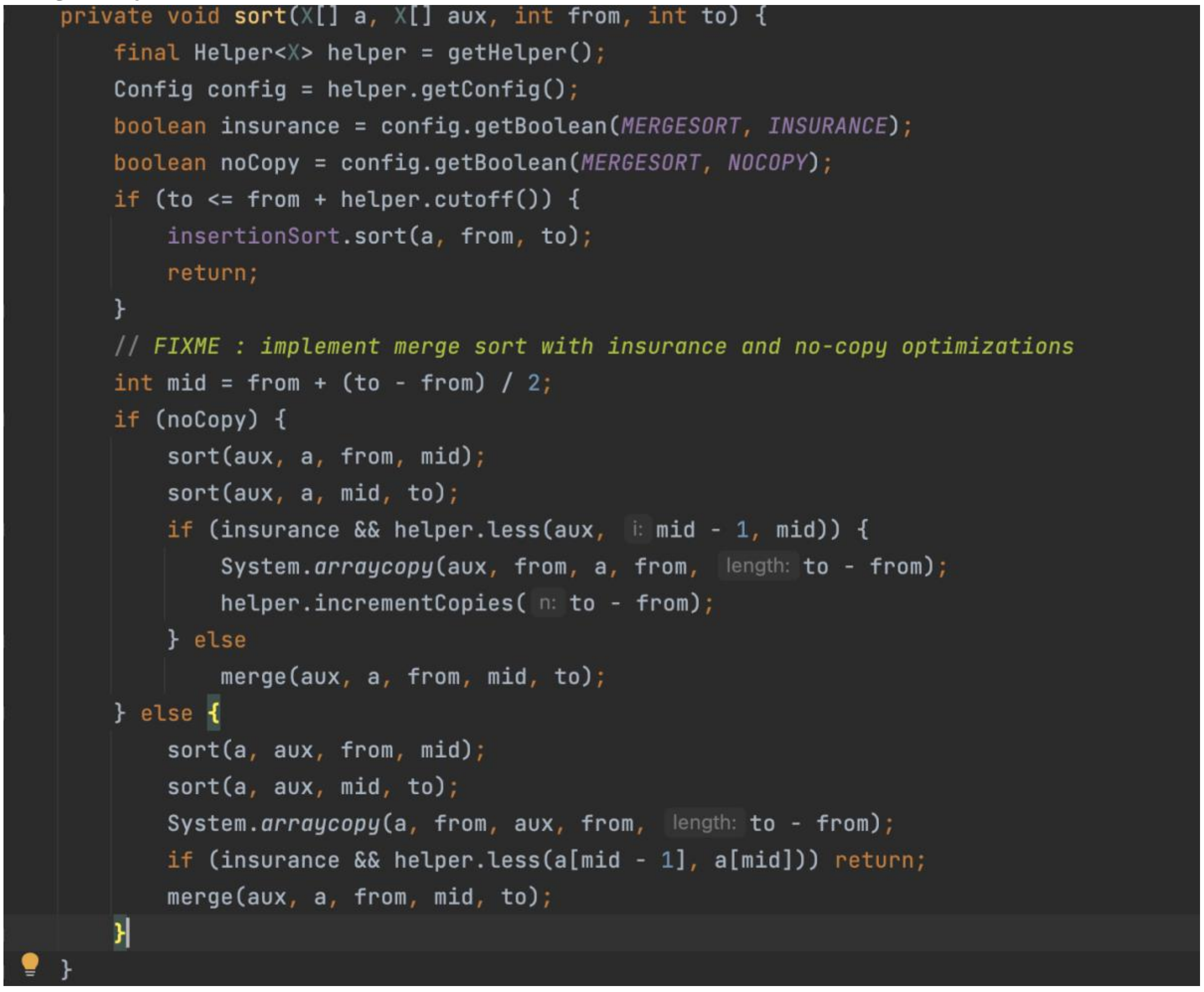
```
✓ HeapSortTest (edu.neu.coe.info6 86 ms
    ✓ testMutatingHeapSort    71 ms
    ✓ sort0                    9 ms
    ✓ sort1                    4 ms
    ✓ sort2                    1 ms
    ✓ sort3                    1 ms
```



```
✓ MergeSortTest (edu.neu.coe.inf 240 ms
    ✓ testSort11_partialsorted    73 ms
    ✓ testSort9_partialsorted     21 ms
    ✓ testSort1                    2 ms
    ✓ testSort2                    4 ms
    ✓ testSort3                    2 ms
    ✓ testSort4                   61 ms
    ✓ testSort5                   11 ms
    ✓ testSort6                   13 ms
    ✓ testSort7                   11 ms
    ✓ testSort10_partialsorted    18 ms
    ✓ testSort8_partialsorted     19 ms
    ✓ testSort12                   3 ms
    ✓ testSort13                   1 ms
    ✓ testSort14                   1 ms
    ✓ testSort1a                   0 ms
```

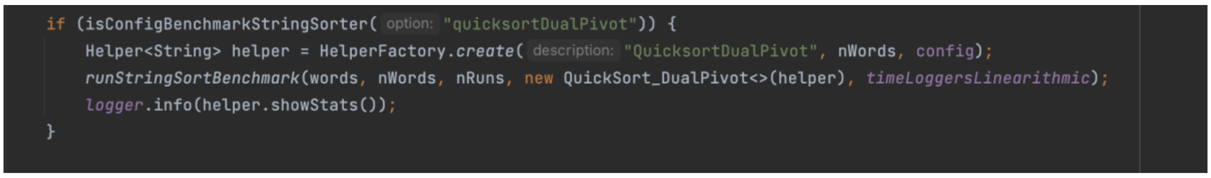
Code Snippets:

1. MergeSort.java: sort method

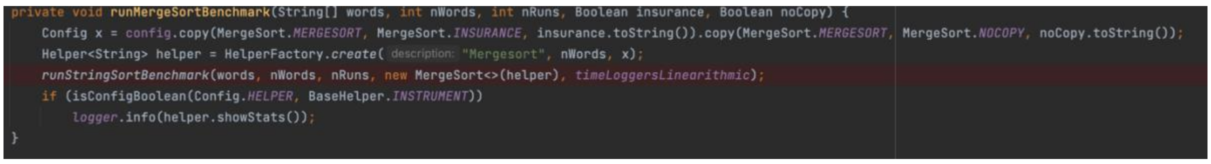


```java
private void sort(X[] a, X[] aux, int from, int to) {
    final Helper<X> helper = getHelper();
    Config config = helper.getConfig();
    boolean insurance = config.getBoolean(MERGESORT, INSURANCE);
    boolean noCopy = config.getBoolean(MERGESORT, NOCOPY);
    if (to <= from + helper.cutoff())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

2. SortBenchmark.java:



```java
if (isConfigBenchmarkStringSorter( option: "quicksortDualPivot")) {
    Helper<String> helper = HelperFactory.create( description: "QuicksortDualPivot", nWords, config);
    runStringSortBenchmark(words, nWords, nRuns, new QuickSort_DualPivot<>(helper), timeLoggersLinearithmic);
    logger.info(helper.showStats());
}
```



```java
private void runMergeSortBenchmark(String[] words, int nWords, int nRuns, Boolean insurance, Boolean noCopy) {
    Config x = config.copy(MergeSort.MERGESORT, MergeSort.INSURANCE, insurance.toString()).copy(MergeSort.MERGESORT, MergeSort.NOCOPY, noCopy.toString());
    Helper<String> helper = HelperFactory.create( description: "Mergesort", nWords, x);
    runStringSortBenchmark(words, nWords, nRuns, new MergeSort<>(helper), timeLoggersLinearithmic);
    if (isConfigBoolean(Config.HELPER, BaseHelper.INSTRUMENT))
        logger.info(helper.showStats());
}
```