Program Structures and Algorithms Spring 2023(SEC –1)
Assignment-6

Name: Naga Venkata Nishanth Sayana

NU ID: 002930970

Task:

Our task at hand is to 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).

Code Screenshots:

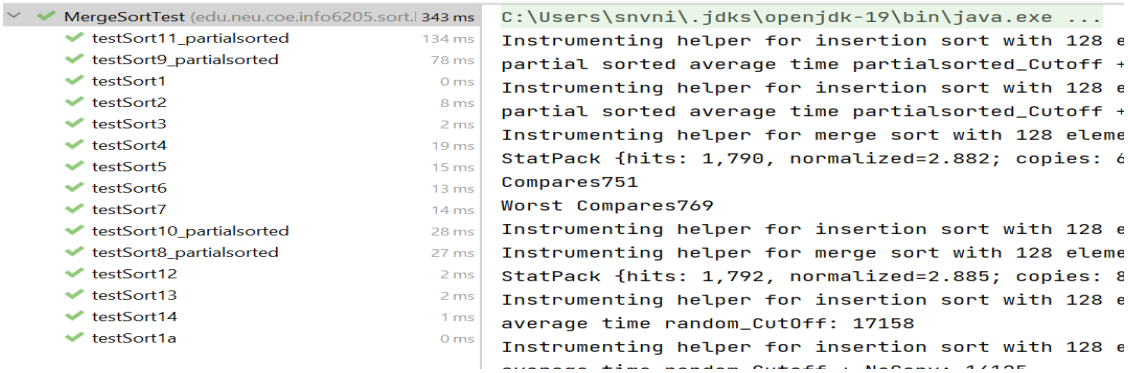
MergeSort

```
final int n = to - from;
int mid = from + n / 2;
if (noCopy) {
    sort(a, from, mid);
    sort(a, mid, to);
    System.arraycopy(a, from, aux, from, n);
    getHelper().incrementCopies(n);
    getHelper().incrementHits( i: 2 * n);

}
else {
    sort(aux, from, mid);
    sort(aux, mid, to);
}

merge(aux, a, from, mid, to);
```

TestCases:

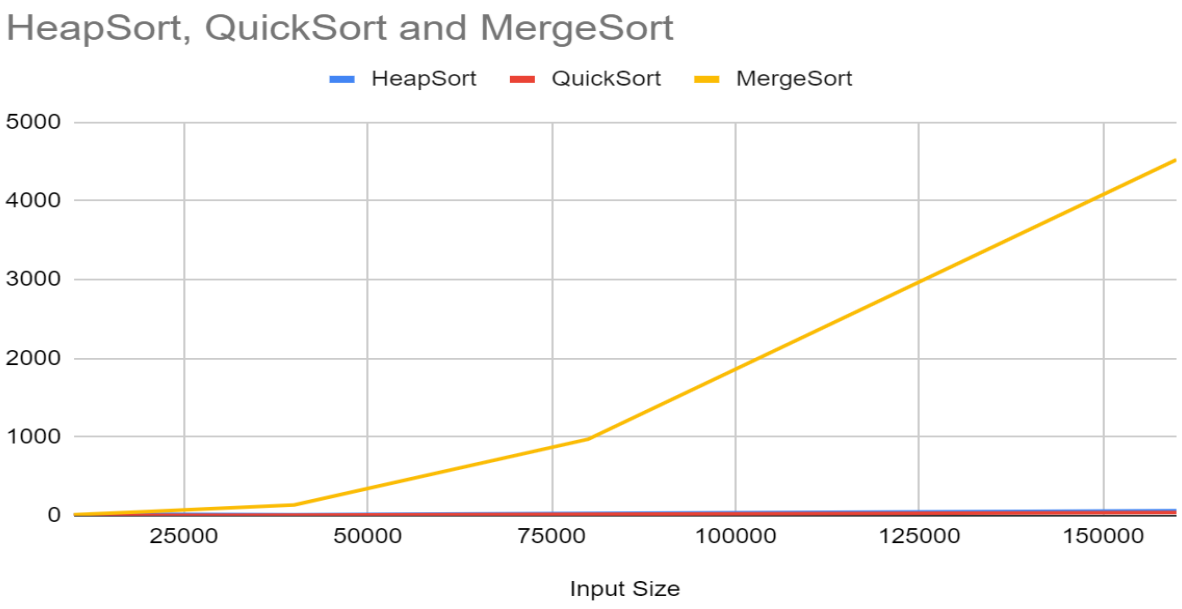


Observations:

Here is the table for timings for three different types of sorts.

1)Time

	Heap Sort	**QuickSort**	**MergeSort**
Element Size	Raw times pern Run(ms)		
10000	4.5	1.77	14.02
20000	17.52	3.5	53.35
40000	12.39	7.69	136.23
80000	28.37	17.9	970.76
160000	64.51	39.59	4523.79

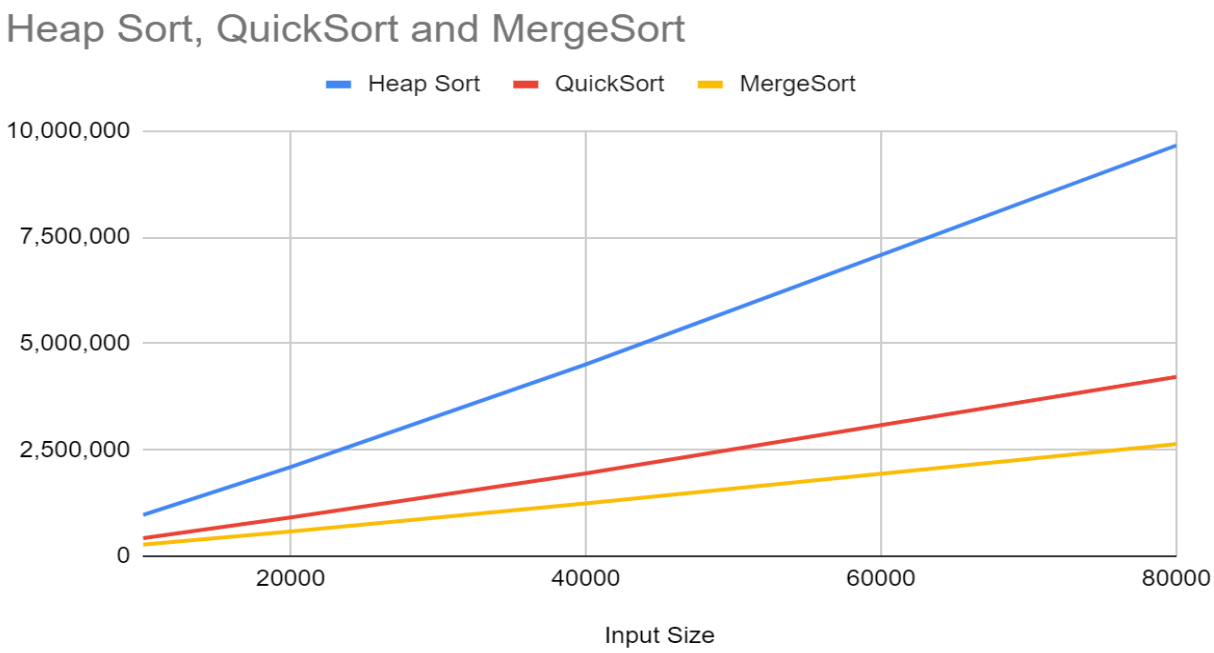


This is a plot between the raw times of three sorts, this shows for this input merge sort shows sigficantly increasing trend.

2)Hits

Tabulation and respective plot

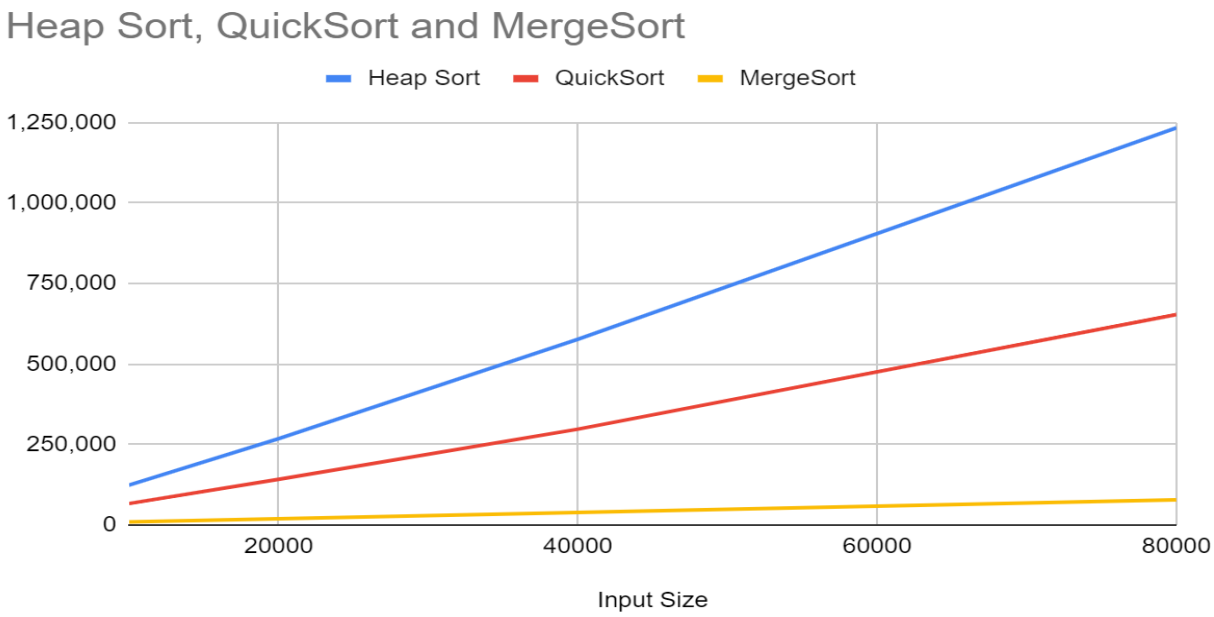
Heap Sort	QuickSort	MergeSort
967,605	424,015	269,800
2,095,089	911,683	579,560
4,510,208	1,947,159	1,239,120
9,660,179	4,217,397	2,638,175



3)Swaps

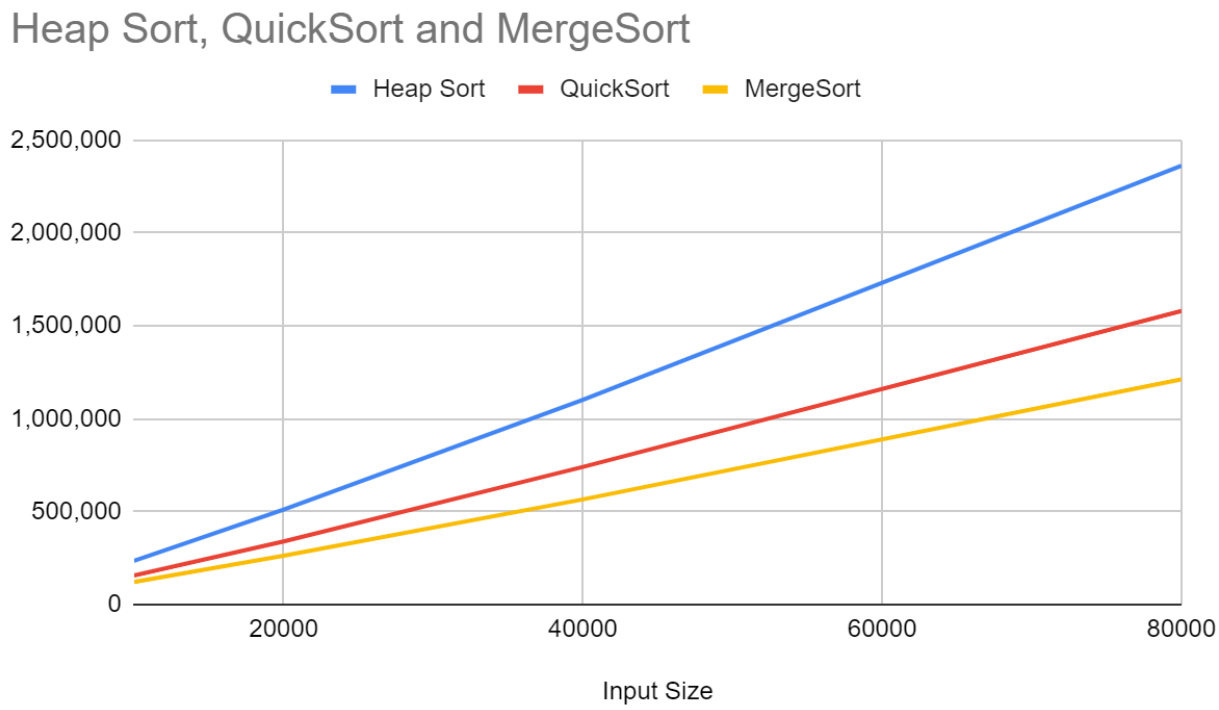
Tabulation and respective plot

Heap Sort	QuickSort	MergeSort
124,211	66,518	9,764
268,403	141,674	19,517
576,805	297,626	39,033
1,233,562	653,733	78,047



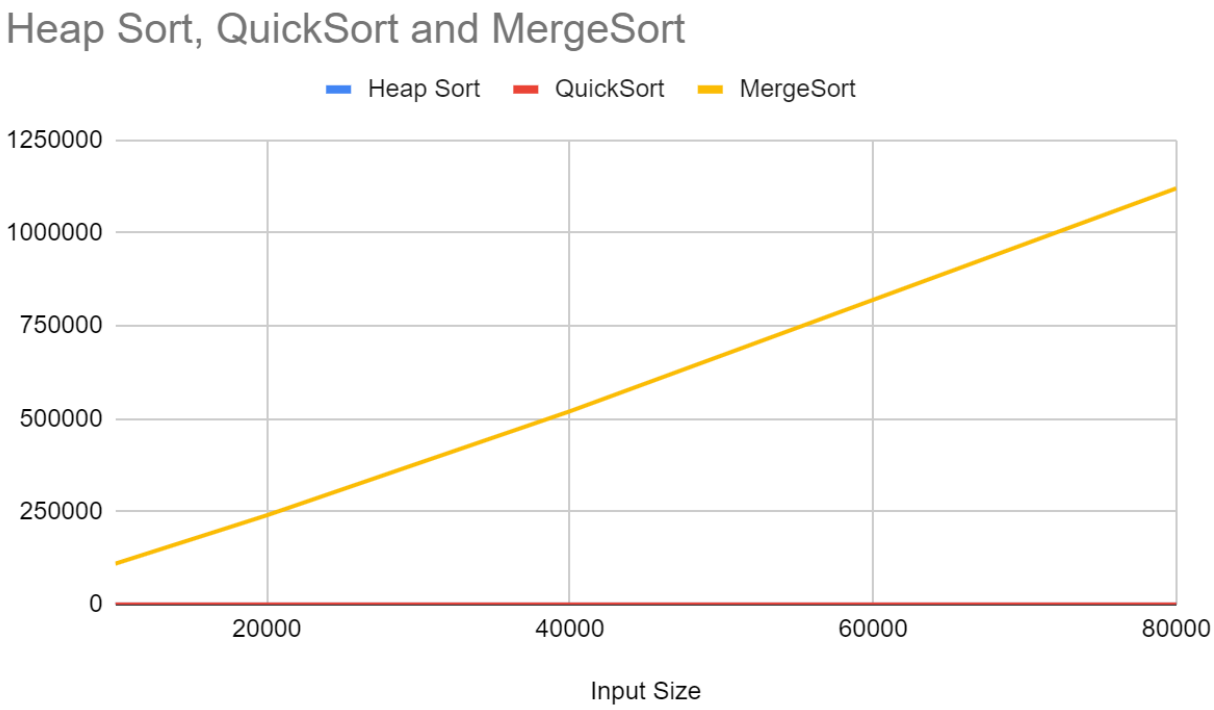
4)Compares

Heap Sort	QuickSort	MergeSort
235,380	156,168	121,510
510,740	339,892	263,004
1,101,494	740,781	566,015
2,362,965	1,580,634	1,212,040



5)Copies

Heap Sort	QuickSort	MergeSort
0	0	110,000
0	0	240,000
0	0	520,000
0	0	1,120,000



Conclusion:

When considering the performance of sorting algorithms, operations such as compares, copies, and swaps involve array hits. Therefore, the number of hits can serve as a neutral way to determine which algorithm performs better, with a larger number indicating poorer performance. However, if the operations being performed are not of the same duration, the one that takes less time and involves fewer parameters would be a better predictor of the algorithm's completion time. Swaps tend to be more costly than copies, making copy a less expensive operation.

However, comparing copies and comparisons is not straightforward since comparisons could be more expensive, depending on the hardware. In terms of determining which algorithm performs worse, the one with the most swaps has the worst performance, followed by copy and comparison. If no metrics are available, a general number of hits can be used, with the highest number indicating the worst performance.

Based on our observations, merge sort has better performance followed by quicksort and heapsort.