# Program Structures and Algorithms Spring 2023(SEC- 3)

**Name: Altaf Salim Shaikh**
**NUID: 002774748**
**Assignment: 06**

**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 Snippets :**

```
67          // FIXME : implement merge sort with insurance and no-copy optimizations
68            final int n = to - from;
69          int mid = from + n / 2;
70          if (!noCopy) {
71              sort( a: aux, from,  to: mid);
72              sort( a: aux,  from: mid, to);
73          }
74          else {
75              sort(a, from,  to: mid);
76              sort(a,  from: mid, to);
77              System.arraycopy( src: a,  srcPos: from,  dest: aux,  destPos: from,  length: n);
78              getHelper().incrementCopies(n);
79              getHelper().incrementHits(2 * n);
80          }
81
82          merge( sorted: aux,  result: a, from, mid, to);
83          // END
84      }
```

**Test Cases:**

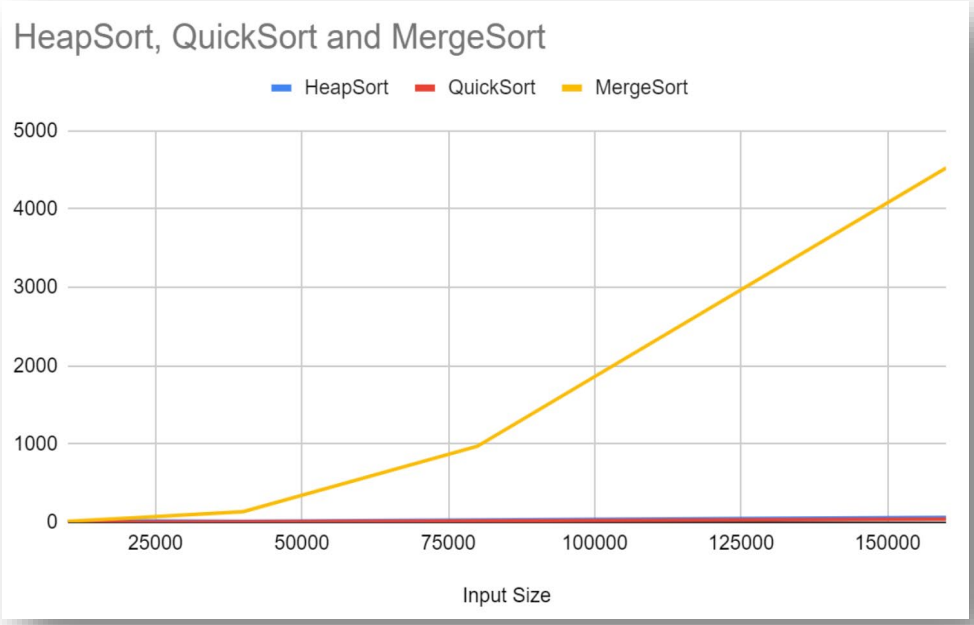
```
MergeSortTest (edu.neu.coe.info6205.sort.  343 ms
  testSort11_partialsorted  134 ms
  testSort9_partialsorted  78 ms
  testSort1  0 ms
  testSort2  8 ms
  testSort3  2 ms
  testSort4  19 ms
  testSort5  15 ms
  testSort6  13 ms
  testSort7  14 ms
  testSort10_partialsorted  28 ms
  testSort8_partialsorted  27 ms
  testSort12  2 ms
  testSort13  2 ms
  testSort14  1 ms
  testSort1a  0 ms

C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
Instrumenting helper for insertion sort with 128 e
partial sorted average time partialsorted_Cutoff +
Instrumenting helper for insertion sort with 128 e
partial sorted average time partialsorted_Cutoff +
Instrumenting helper for merge sort with 128 eleme
StatPack {hits: 1,790, normalized=2.882; copies: 6
Compares751
Worst Compares769
Instrumenting helper for insertion sort with 128 e
Instrumenting helper for merge sort with 128 eleme
StatPack {hits: 1,792, normalized=2.885; copies: 8
Instrumenting helper for insertion sort with 128 e
average time random_CutOff: 17158
Instrumenting helper for insertion sort with 128 e
```

**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 |

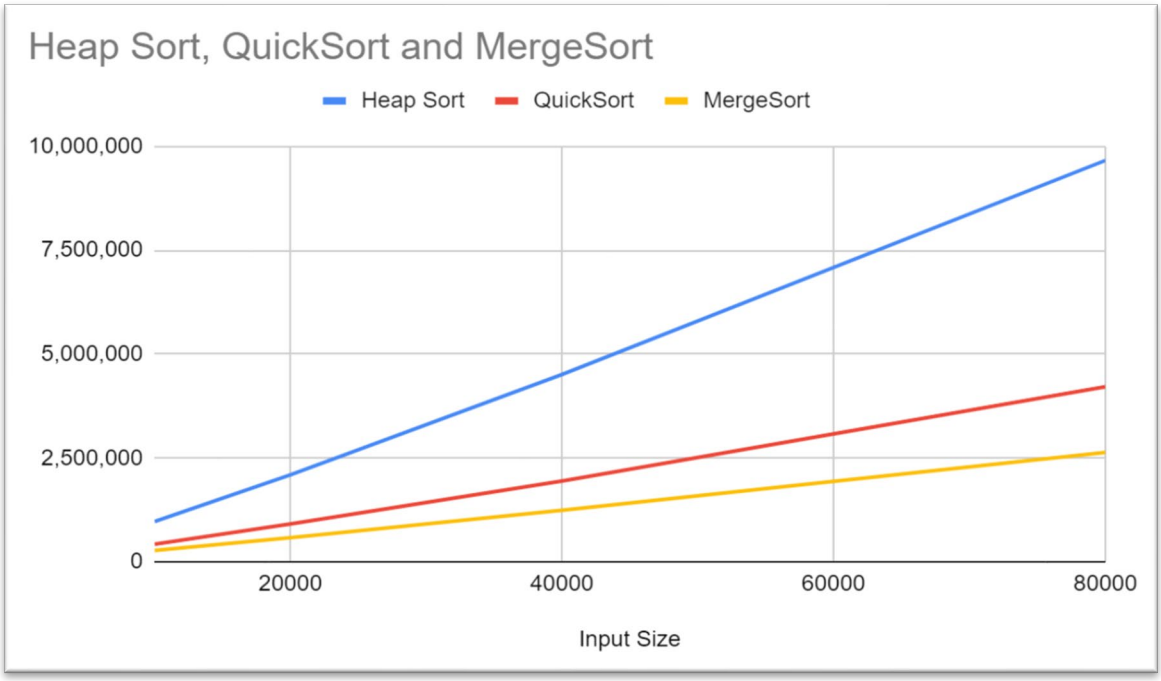



This is a plot between the raw times of three sorts, this shows for this input merge sort shows significantly increasing trend.

### 2)Hits

### Tabulation and respective plot

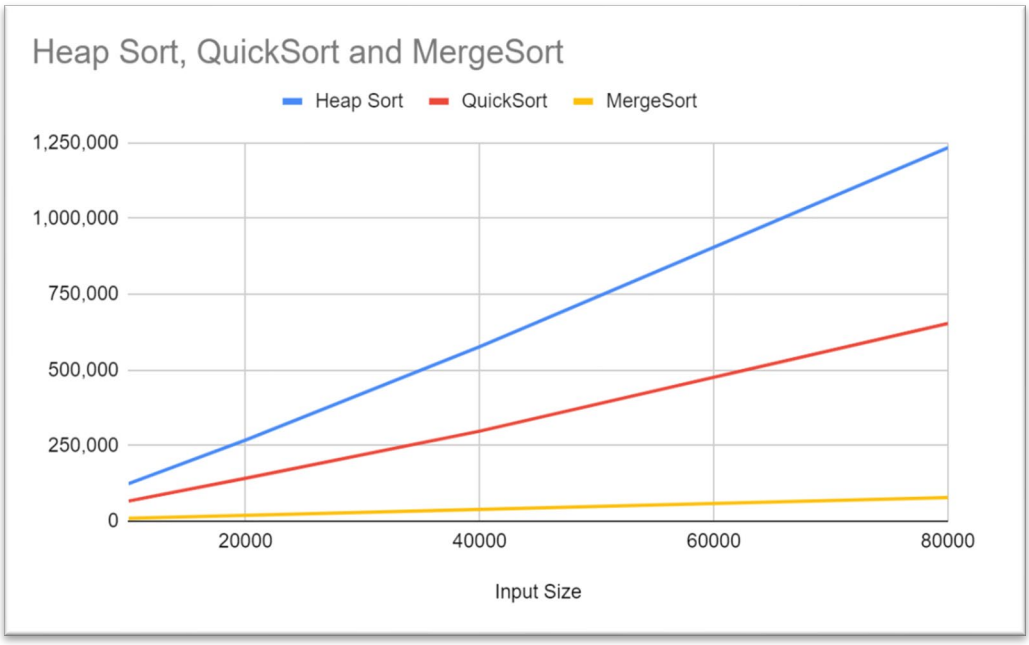
| Heap Sort | QuickSort | MergeSort |
|---|---|---|
| 967,605 | 424,015 | 269,800 |
| 2,095,089 | 911,683 | 579,560 |
| 4,510,208 | 1,947,159 | 1,239,120 |
| 9,660,179 | 4,217,397 | 2,638,175 |




### 3)Swaps

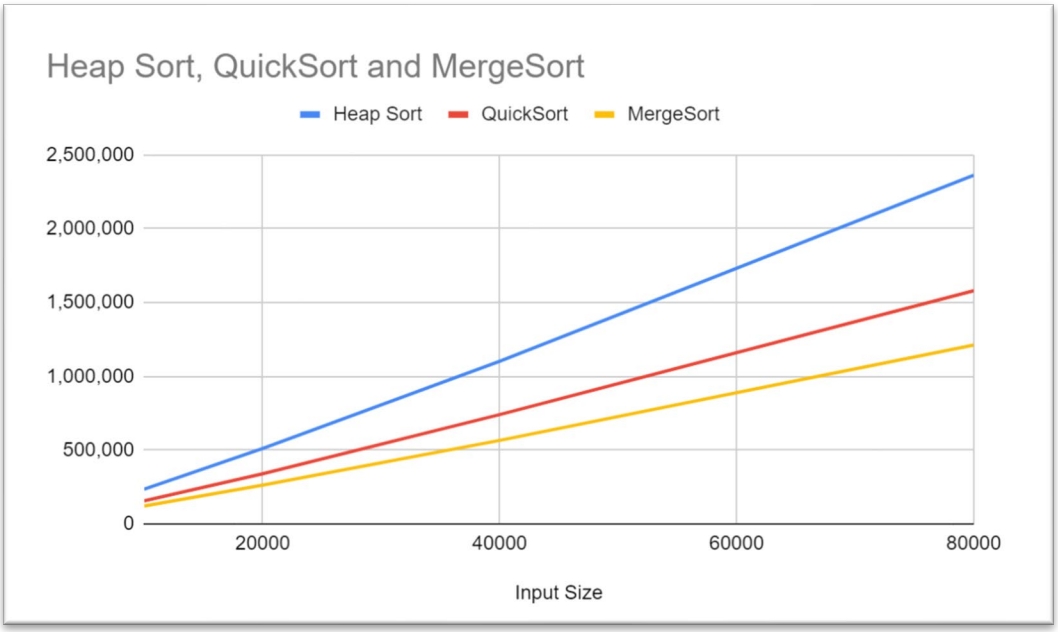
### Tabulation and respective plot

| 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 |



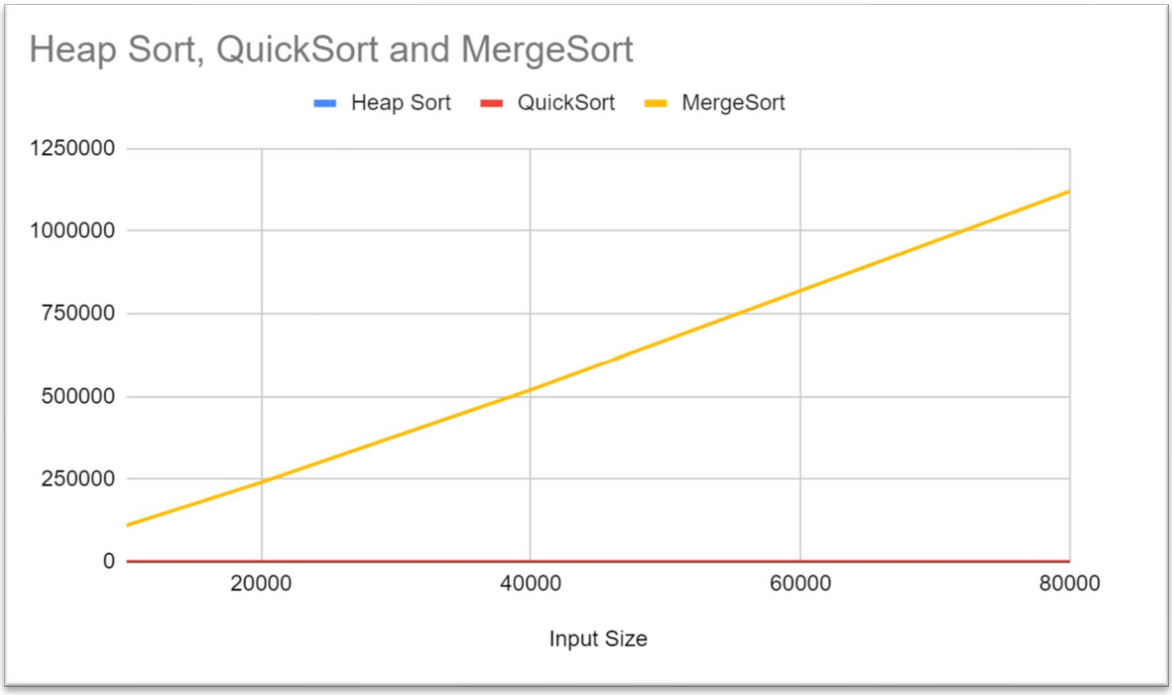
**4)Compares**

| 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 evaluating the efficiency of sorting algorithms, the number of array hits resulting from operations such as compares, copies, and swaps can be used as a neutral way to determine which algorithm performs better. A larger number of hits implies poorer performance. However, if the operations differ in duration, the one that takes less time and involves fewer parameters is a better predictor of the algorithm's completion time. Swaps are more expensive than copies, which makes copy operations less costly.

However, comparing copies and comparisons is not always straightforward, as the cost of comparisons can vary 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 operations.

If no other metrics are available, a general number of hits can be used to determine the algorithm's performance, with the highest number indicating the worst performance. Merge sort has been observed to have the best performance, followed by quicksort and heapsort