Assignment 6

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).

The configuration for these benchmarks is determined by the config.ini file. It should be reasonably easy to figure out how it all works. The config.ini file should look something like this:
[sortbenchmark]

version = 1.0.0 (sortbenchmark)

[helper] instrument = true seed = 0

cutoff =

[instrumenting]

# The options in this section apply only if instrument (in [helper]) is set to true. swaps = true

compares = true

copies = true

```
fixes = false

hits = true

# This slows everything down a lot so keep this small (or zero) inversions = 0


[benchmarkstringsorters]

words = 1000 # currently ignored runs = 20 # currently ignored mergesort = true

timsort = false

quicksort = false

introsort = false

insertionsort = false

bubblesort = false

quicksort3way = false quicksortDualPivot = true randomsort = false


[benchmarkdatesorters] timsort = false

n = 100000


[mergesort] insurance = false nocopy = true


[shellsort] n = 100000


[operationsbenchmark] nlargest = 10000000 repetitions = 10
```

There is no config.ini entry for heapsort. You will have to work that one out for yourself. The number of runs is actually determined by the problem sizes using a fixed formula.

One more thing: the sizes of the experiments are actually defined in the command line (if you are running in IntelliJ/IDEA then, under Edit Configurations for the SortBenchmark, enter 10000 20000 etc. in the box just above CLI arguments to your application).

You will also need to edit the SortBenchmark class. Insert the following lines before the introsort section:

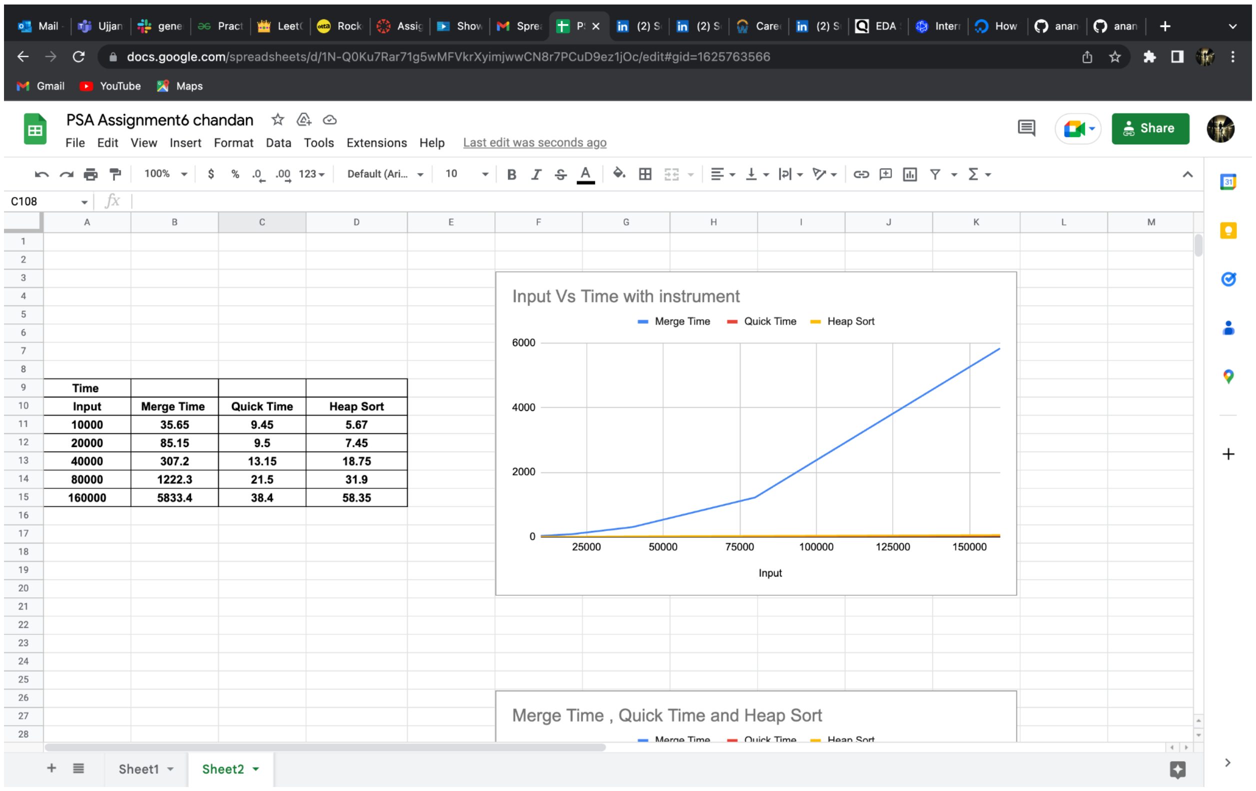
```
f (isConfigBenchmarkStringSorter("heapsort")) {

Helper<String> helper = HelperFactory.create("Heapsort", nWords, config);
runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper),



timeLoggersLinearithmic);

}
```

Then you can add the following extra line into the config.ini file (again, before the introsort line (which is 25 for me):

```
heapsort = true
```

Remember that your job is to determine the best predictor: that will mean the graph of the appropriate observation will match the graph of the timings most closely.

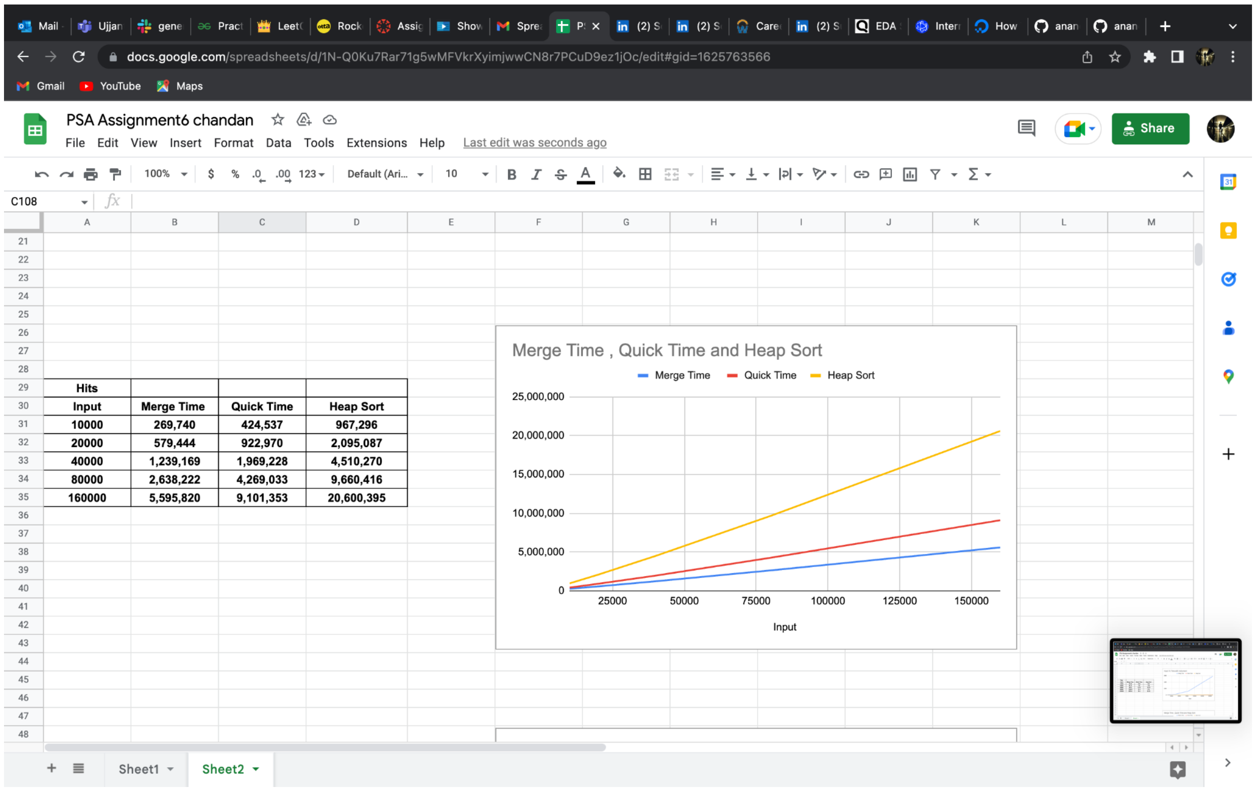
Time taken for merge sort,quick dual pivot and heap sort with instruments


docs.google.com/spreadsheets/d/1N-Q0Ku7Rar71g5wMFVkrXyimjwwCN8r7PCuD9ez1jOc/edit#gid=1625763566
PSA Assignment6 chandan
Input Vs Time with instrument
Merge Time
Quick Time
Heap Sort
6000
4000
2000
0
25000
50000
75000
100000
125000
150000
Input
Merge Time , Quick Time and Heap Sort
Sheet1
Sheet2

Time			
Input	Merge Time	Quick Time	Heap Sort
10000	35.65	9.45	5.67
20000	85.15	9.5	7.45
40000	307.2	13.15	18.75
80000	1222.3	21.5	31.9
160000	5833.4	38.4	58.35

From the above graph we can conclude that the time taken for merge sort is more compared to the quick and heap sort. But the heap sort is better than the quick sort in terms of the timing is concerned.

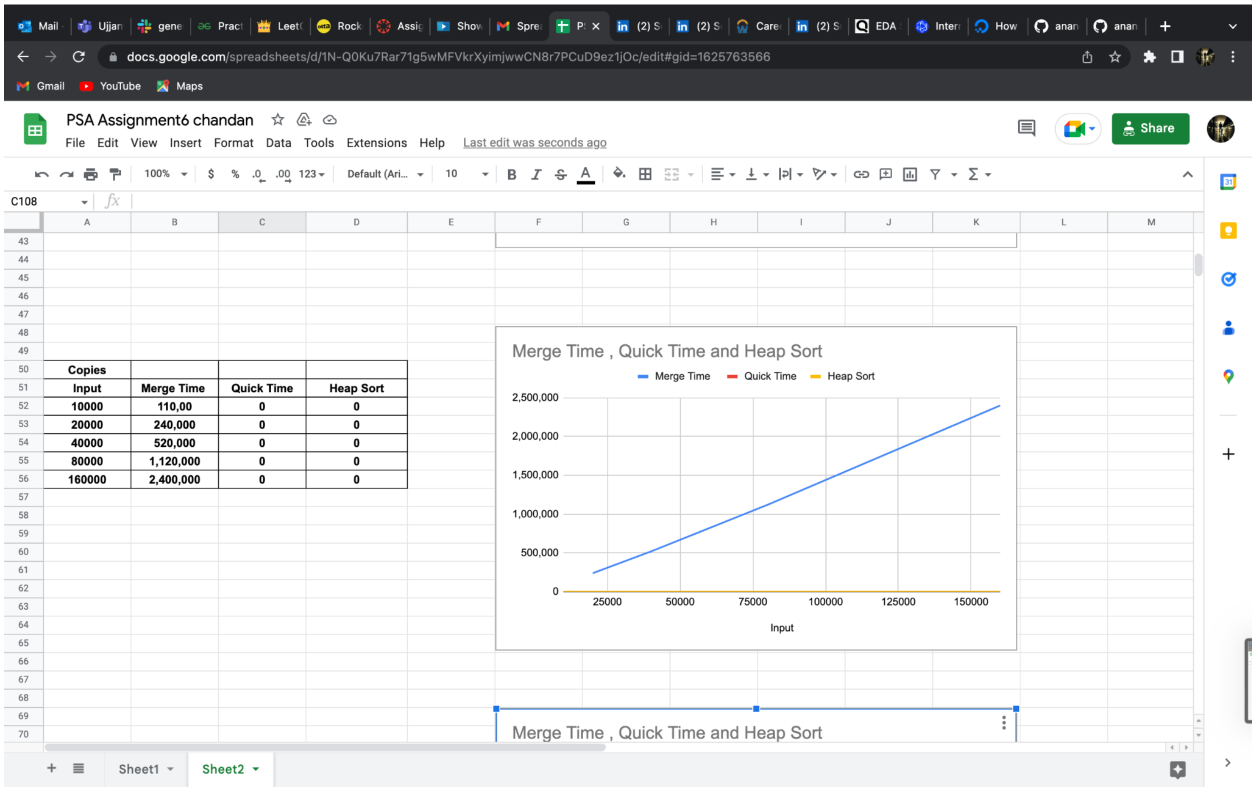
Hits taken by merge sort,quick dual pivot and heap sort with instruments



Hits			
Input	Merge Time	Quick Time	Heap Sort
10000	269,740	424,537	967,296
20000	579,444	922,970	2,095,087
40000	1,239,169	1,969,228	4,510,270
80000	2,638,222	4,269,033	9,660,416
160000	5,595,820	9,101,353	20,600,395

The number of hits taken by heap sort is more compared to merge and quick sort. But the quick sort is a bit higher compared to the merge sort in terms of the hit.

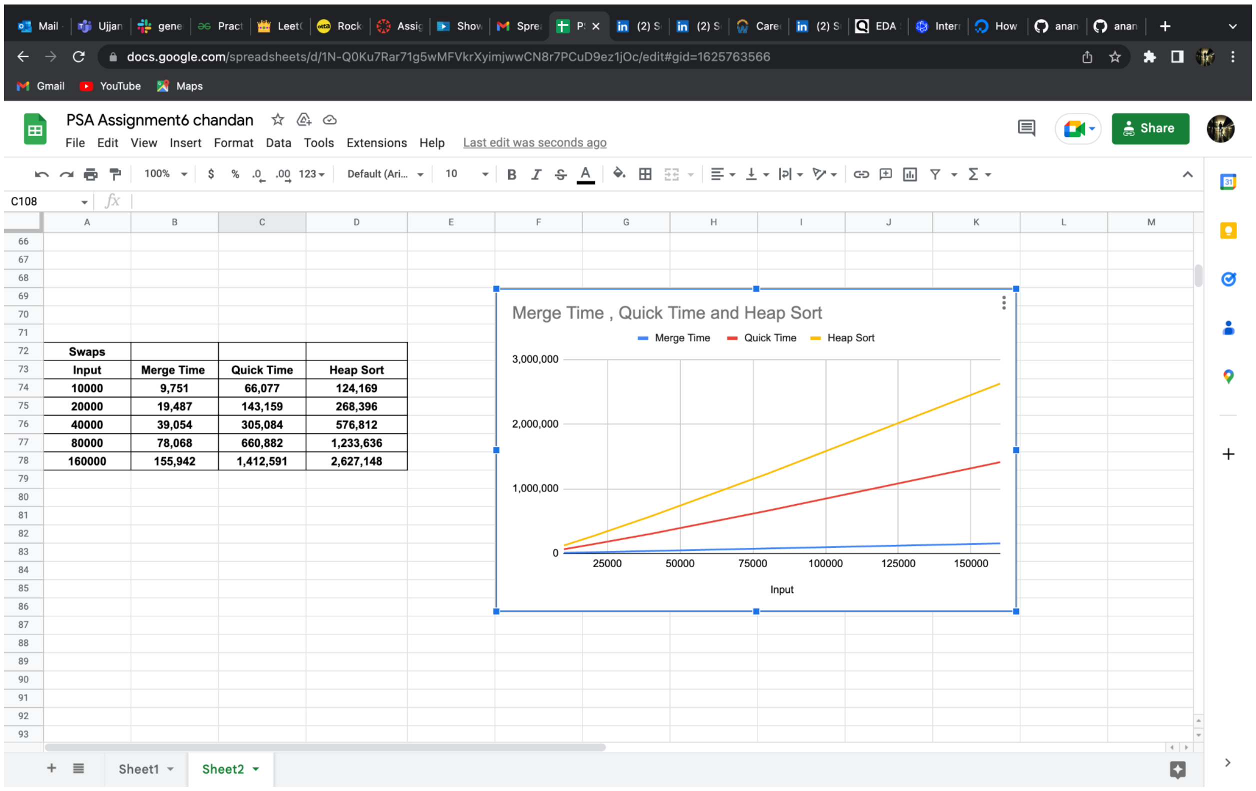
Copies taken by merge sort,quick dual pivot and heap sort with instruments



Copies			
Input	Merge Time	Quick Time	Heap Sort
10000	110,00	0	0
20000	240,000	0	0
40000	520,000	0	0
80000	1,120,000	0	0
160000	2,400,000	0	0

The quick and merge sort will not use the copy function and therefore the a duplicate copy is used by the merge sort so there merge sort has high number of copies compare to heap and quick sort.

Swaps taken by merge sort,quick dual pivot and heap sort with instruments

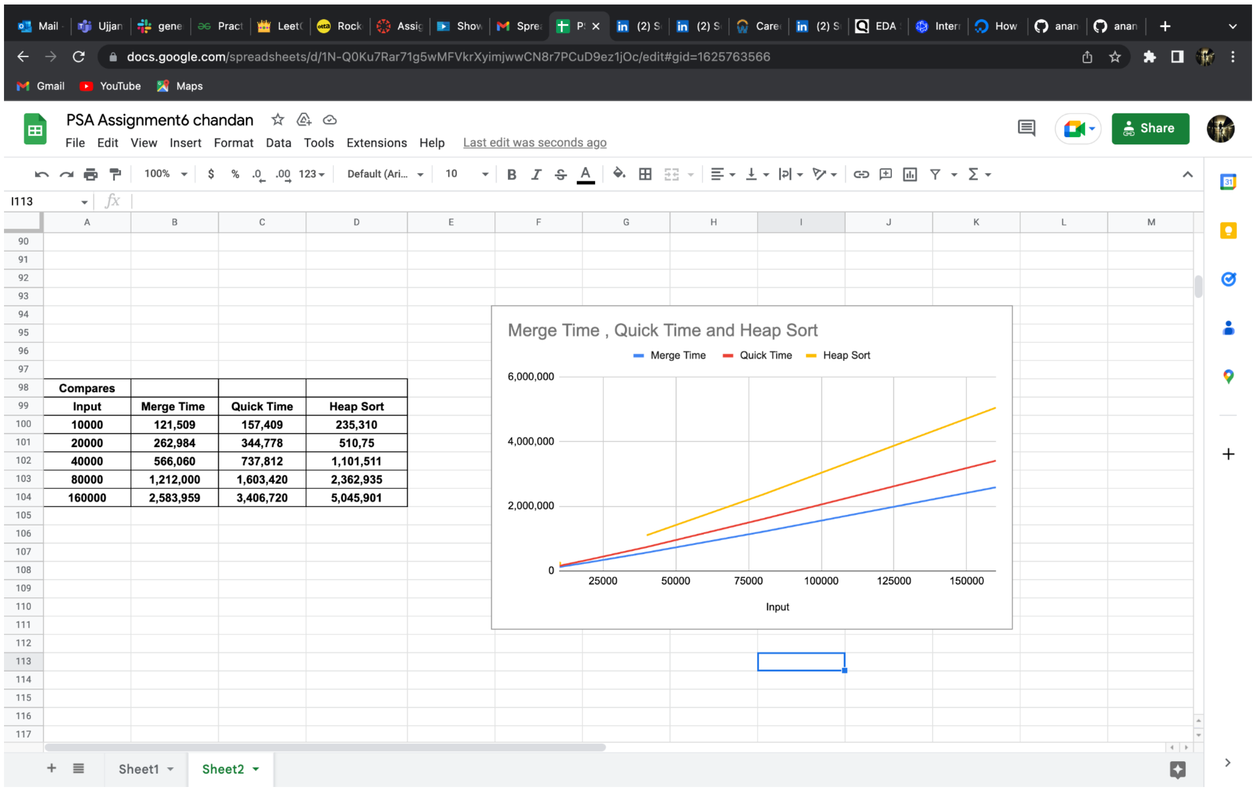


PSA Assignment6 chandan

Swaps			
Input	Merge Time	Quick Time	Heap Sort
10000	9,751	66,077	124,169
20000	19,487	143,159	268,396
40000	39,054	305,084	576,812
80000	78,068	660,882	1,233,636
160000	155,942	1,412,591	2,627,148

The number of swaps is more in heap sorts compared to quick sort and merge sort therefore the quick sort take more number of swaps compared to merge sort.

Compares taken by merge sort,quick dual pivot and heap sort with instruments



Compares			
Input	Merge Time	Quick Time	Heap Sort
10000	121,509	157,409	235,310
20000	262,984	344,778	510,75
40000	566,060	737,812	1,101,511
80000	1,212,000	1,603,420	2,362,935
160000	2,583,959	3,406,720	5,045,901

PSA Assignment6 chandan
docs.google.com/spreadsheets/d/1N-Q0Ku7Rar71g5wMFVkrXyimjwwCN8r7PCuD9ez1jOc/edit#gid=1625763566
Merge Time , Quick Time and Heap Sort
Merge Time
Quick Time
Heap Sort
6,000,000
4,000,000
2,000,000
0
25000
50000
75000
100000
125000
150000
Input
Sheet1
Sheet2

The number of compares with heap sort is more compared to merge and quick sort,but compared to quick sort merge sort takes less number of compares than quick sort.

Summary

From the above swaps, compares, copies,hits we can say swap, compares and copies to work the hits are included therefore hits will be a neutral way to determine the value of time. Therefore the higher the number of hits taken by the algorithms indicates that the poor performance of the algorithm so hits might be a best way to predator of time

Second where we compare swaps, compares, copies where the swaps are more costly than compares. And where as the copies takes 2 hits and the number of hits for swaps is 4 therefore higher the number of swaps poorer the performance of the algorithm therefore the swaps can be compared that with hits.The one with the highest number of hits would indicate the worst performance.