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).

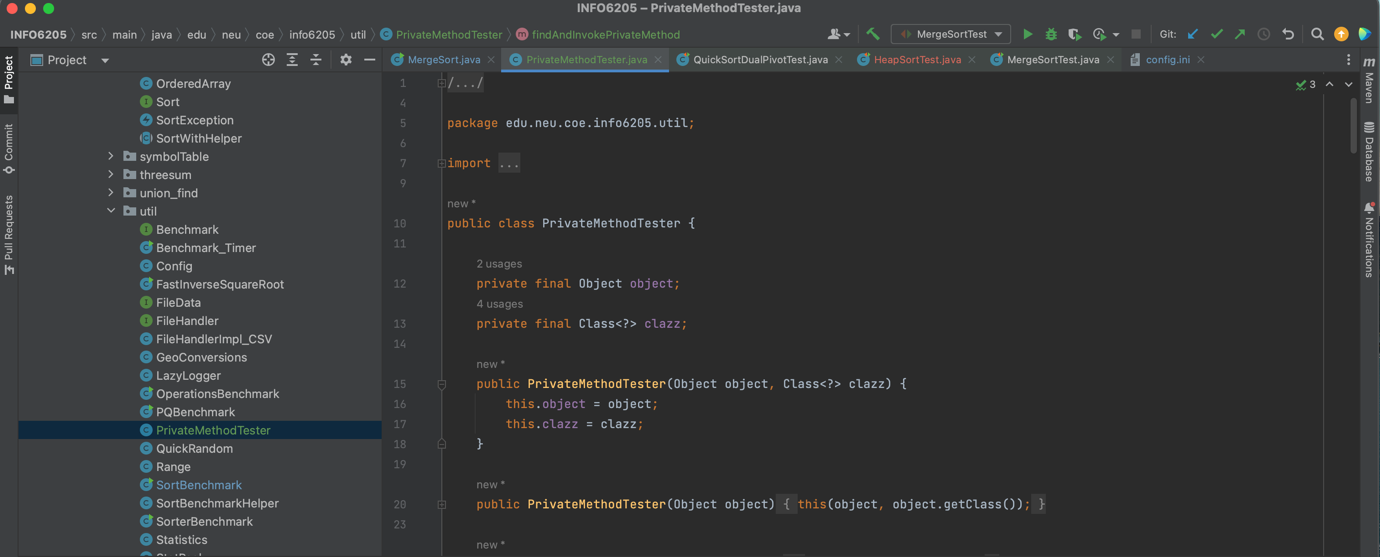
All of the code to count comparisons, swaps/copies, and hits, is already implemented in the *InstrumentedHelper* class. You can see examples of the usage of this kind of analysis in:

* 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.



This helps for the instrumentation execution required for the sorts.

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

Text

Description automatically generated

1. 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,

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

While the time of execution 2 sorts where 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 :-

Text

Description automatically generated

Text

Description automatically generated

Quick Dual Pivot Sort Output :-

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Heap Sort Output :-

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated

**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.

Chart, line chart

Description automatically generated

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.

Chart, line chart

Description automatically generated

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 Compares | 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.

Chart, line chart

Description automatically generated

**Unit Test Cases:**

**Screenshot for MergeSort Test Cases:-**

**Graphical user interface, text

Description automatically generated**

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

**Graphical user interface, text

Description automatically generated**

**Screenshot for HeapSort Test cases:-**

**Graphical user interface, text, application

Description automatically generated**