Program Structures and Algorithms

Spring 2023(SEC –01)

NAME: Hinal Prabodh Patel

NUID: 002766180

Task:

n 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 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:

if (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.

Evidence to support the conclusion:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | with Instrumentation = True | | |  |  |
| Heapsort  Array size | Runs | Raw Time (m sec) | Normalized  Time (m sec) | hits | Swaps | compares |
| 10,000 | 100 | 2.64 | 3.71 | 967,526 | 124,197 | 235,369 |
| 20,000 | 100 | 5.65 | 3.67 | 2,094,967 | 268,375 | 510,734 |
| 40,000 | 100 | 12.17 | 3.66 | 4,510,101 | 576,788 | 1,101,474 |
| 80,000 | 100 | 26.51 | 3.72 | 9,660,159 | 1,233,569 | 2,362,942 |
| 160,000 | 100 | 58.07 | 3.82 | 20,600,545 | 2,627,176 | 5,045,921 |
| 320,000 | 100 | 130.8 | 4.04 | 43,761,095 | 5,574,352 | 10,731,844 |
| 640,000 | 100 | 389.7 | 5.69 | 92,643,498 | 11,788,913 | 22,743,924 |
| 1,280,000 | 100 | 1016.56 | 7.02 | 195,525,370 | 24,857,519 | 48,047,648 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | with Instrumentation = False | | |
| Heapsort  Array size | Runs | Raw Time (m sec) | Normalized  Time (m sec) |  |
| 10,000 | 100 | 2.48 | 3.49 |  |
| 20,000 | 100 | 5.41 | 3.51 |  |
| 40,000 | 100 | 11.69 | 3.52 |  |
| 80,000 | 100 | 25.26 | 3.54 |  |
| 160,000 | 100 | 56.58 | 3.72 |  |
| 320,000 | 100 | 127.76 | 3.95 |  |
| 640,000 | 100 | 357.4 | 5.22 |  |
| 1,280,000 | 100 | 967.53 | 6.69 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | with Instrumentation = True | | |  |  |
| QuicksortDualPivot  Array size | Runs | Raw Time (m sec) | Normalized Time (m sec) | hits | Swaps | compares |
| 10,000 | 100 | 2.49 | 3.5 | 412,616 | 62,891 | 156,797 |
| 20,000 | 100 | 4.98 | 3.23 | 897,454 | 137,107 | 340,550 |
| 40,000 | 100 | 8.32 | 2.5 | 1,933,104 | 294,034 | 739,898 |
| 80,000 | 100 | 17.7 | 2.48 | 4,197,438 | 642,375 | 1,594,051 |
| 160,000 | 100 | 38.75 | 2.55 | 8,955,034 | 1,374,561 | 3,388,720 |
| 320,000 | 100 | 90.9 | 2.81 | 19,047,974 | 2,909,519 | 7,273,943 |
| 640,000 | 100 | 209.48 | 3.06 | 40,426,405 | 6,187,276 | 15,405,622 |
| 1,280,000 | 100 | 529.27 | 3.66 | 85,340,398 | 13,055,302 | 32,575,428 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | with Instrumentation = False | | |
| QuicksortDualPivot  Array size | Runs | Raw Time (m sec) | Normalized Time (m sec) |  |
| 10,000 | 100 | 2.3 | 3.24 |  |
| 20,000 | 100 | 4.14 | 2.69 |  |
| 40,000 | 100 | 8.13 | 2.45 |  |
| 80,000 | 100 | 17.23 | 2.42 |  |
| 160,000 | 100 | 38.87 | 2.56 |  |
| 320,000 | 100 | 87.71 | 2.71 |  |
| 640,000 | 100 | 213.12 | 3.11 |  |
| 1,280,000 | 100 | 465.63 | 3.22 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | with Instrumentation = True | | |  |  |  |
| Mergesort  Array size | Runs | Raw  Time (m sec) | Normalized  Time (m sec) | hits | Swaps | Compares | Copies |
| 10,000 | 100 | 2.08 | 2.93 | 489,797 | 9,761 | 121,526 | 220,000 |
| 20,000 | 100 | 3.69 | 2.39 | 1,059,565 | 19,522 | 263,001 | 480,000 |
| 40,000 | 100 | 7.76 | 2.34 | 2,279,001 | 39,003 | 565,974 | 1,040,000 |
| 80,000 | 100 | 16.3 | 2.29 | 4,878,212 | 78,065 | 1,212,007 | 2,240,000 |
| 160,000 | 100 | 36.39 | 2.39 | 10,396,380 | 156,117 | 2,584,013 | 4,800,000 |
| 320,000 | 100 | 80.46 | 2.49 | 22,072,950 | 312,272 | 5,488,135 | 10,240,000 |
| 640,000 | 100 | 227.26 | 3.32 | 46,706,518 | 624,770 | 11,616,301 | 21,760,000 |
| 1,280,000 | 100 | 577.14 | 3.99 | 98,532,611 | 1,249,366 | 24,512,472 | 46,080,000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | with Instrumentation = False | | |
| Mergesort  Array size | Runs | Raw  Time (m sec) | Normalized  Time (m sec) | |
| 10,000 | 100 | 1.88 | 2.65 |  |
| 20,000 | 100 | 3.52 | 2.28 |  |
| 40,000 | 100 | 7.24 | 2.18 |  |
| 80,000 | 100 | 15.24 | 2.14 |  |
| 160,000 | 100 | 34.49 | 2.27 |  |
| 320,000 | 100 | 76.53 | 2.37 |  |
| 640,000 | 100 | 206.1 | 3.01 |  |
| 1,280,000 | 100 | 520.4 | 3.6 |  |

Graphical Representation:

Relationship Conclusion:

Heapsort:

The normalized time appears to be the greatest predictor of overall heapsort execution time since it gives a standardized measure of performance across varied input sizes and allows for simple comparison across different data points.

But, if we look at the particular metrics of hits (array accesses), swaps, and comparisons, we can see that the number of swaps and compares rises about proportionally to the input size, however the number of hits increases faster than the input size. As a result, for lower input sizes, the number of swaps and comparisons may be effective forecasters of execution time, but the number of hits may become a better predictor for higher input sizes.

Quicksortdualpivot:

The number of comparisons done throughout the execution of the quicksortDualPivot algorithm would be the best predictor of overall execution time based on the information supplied. This is because the number of comparisons performed is directly related to the number of times the method must split the input array, and so has a substantial influence on the algorithm's overall running time.

While the amount of swaps/copies and hits (array accesses) can affect total running time, they are often considerably less relevant than the number of comparisons executed. As a result, while determining the overall execution time of the quicksortDualPivot algorithm, the number of comparisons made is likely the most essential aspect to consider.

Mergesortbasic:

Based on the information supplied, the number of copies created during the algorithm's execution would be the greatest predictor of overall execution time for Mergesort. This is due to the Mergesort algorithm's requirement for sub-array merging, which entails transferring components from the original array to temporary arrays. As a result, the number of copies created has a considerable influence on the algorithm's overall running duration.

While the amount of hits (array accesses) and comparisons can affect total running time, they are often considerably less relevant than the number of copies performed. As a result, when determining the overall execution time of the Mergesort algorithm, the number of copies created is likely the most essential aspect to consider.