Program Structures and Algorithms
Fall 2023

NAME:Shangqing Hu
NUID:001374342

**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), memory used, or some combination of these.

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).

**Relationship Conclusion:**
**MergeSort:**
**Correlation between "Compares" and "Normalized Time": -0.813**
**Correlation between "Swaps" and "Normalized Time": 0.352**
**Correlation between "Hits" and "Normalized Time": -0.547**
**Correlation between "Copies" and "Normalized Time": 0.154**

**From these correlations, it's clear that "Compares" has the highest negative correlation with "Normalized Time." This suggests that the number of comparisons is the best predictor of total execution time for mergesort in this dataset. As the number of comparisons increases, the normalized time tends to decrease, indicating a strong inverse relationship. On the other hand, the other factors (swaps, hits, and copies) show weaker correlations with execution time.**

**Therefore, for mergesort in this dataset, the number of comparisons is the most influential predictor of total execution time.**

**Dual Pivot Quicksort:**
**Correlation between "Compares" and "Normalized Time": 0.389**
**Correlation between "Swaps" and "Normalized Time": -0.117**
**Correlation between "Hits" and "Normalized Time": 0.595**
**Correlation between "Copies" and "Normalized Time": NaN (since there are no copies in this dataset)**

**From these correlations, it's clear that "Hits" (array accesses) has the highest positive correlation with "Normalized Time." This suggests that the number of hits is the best predictor of total execution time for dual-pivot quicksort in this dataset. As the number of hits increases, the normalized time tends to increase, indicating a strong positive relationship. The number of comparisons also shows a positive but weaker correlation with execution time.**

**Swaps have a very weak negative correlation with execution time, but it is not a strong predictor in this dataset. Also, "Copies" does not exist in this dataset.**

**Therefore, for dual-pivot quicksort in this dataset, the number of hits (array accesses) is the most influential predictor of total execution time.**

**HeapSort:**
**Correlation between "Compares" and "Normalized Time": 0.804**
**Correlation between "Swaps" and "Normalized Time": 0.037**
**Correlation between "Hits" and "Normalized Time": 0.821**
**Correlation between "Copies" and "Normalized Time": NaN (since there are no copies in this dataset)**

**From these correlations, it's clear that both "Compares" and "Hits" have strong positive correlations with "Normalized Time." However, "Hits" (array accesses) has a slightly higher correlation than "Compares." This suggests that the number of hits is the best predictor of total execution time for heapsort in this dataset. As the number of hits increases, the normalized time tends to increase, indicating a strong positive relationship.**

**Swaps have a very weak positive correlation with execution time, and "Copies" does not exist in this dataset.**

**Therefore, for heapsort in this dataset, the number of hits (array accesses) is the most influential predictor of total execution time.**

**Evidence:**

MergeSort

| Size | Compares | Swaps | Hits | Copies | Normalized Time |
|---|---|---|---|---|---|
| 10000 | 1.319 | 0.106 | 2.929 | 1.194 | 3.48 |
| 20000 | 1.328 | 0.099 | 2.926 | 1.212 | 3.31 |
| 40000 | 1.335 | 0.092 | 2.923 | 1.227 | 4.13 |
| 80000 | 1.342 | 0.086 | 2.921 | 1.240 | 4.94 |
| 160000 | 1.348 | 0.082 | 2.919 | 1.252 | 4.20 |
| 256000 | 1.350 | 0.070 | 2.905 | 1.271 | 5.12 |

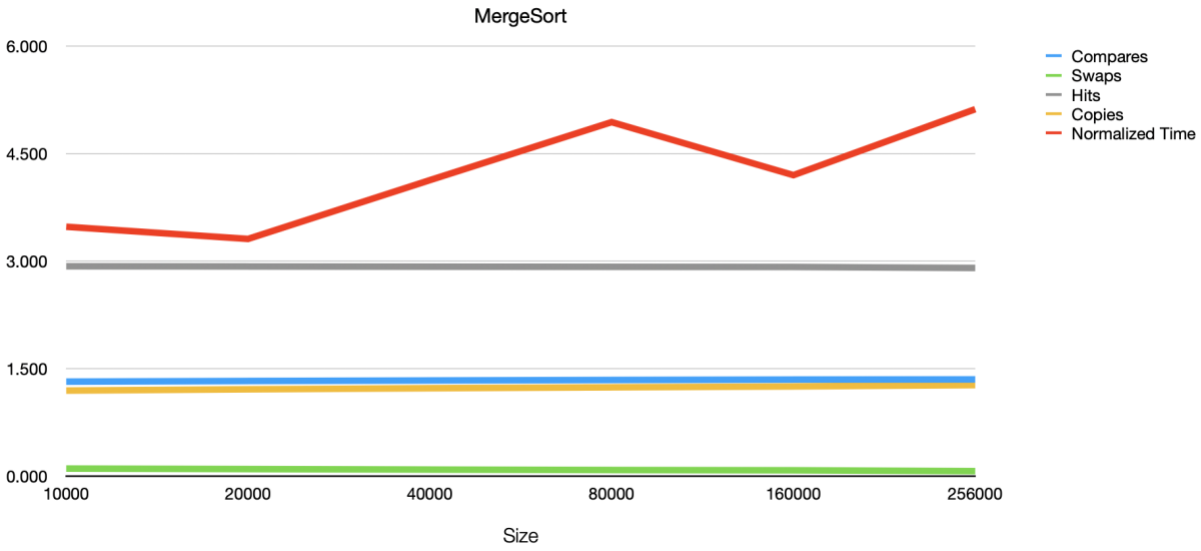
Dual-pivot QuickSort

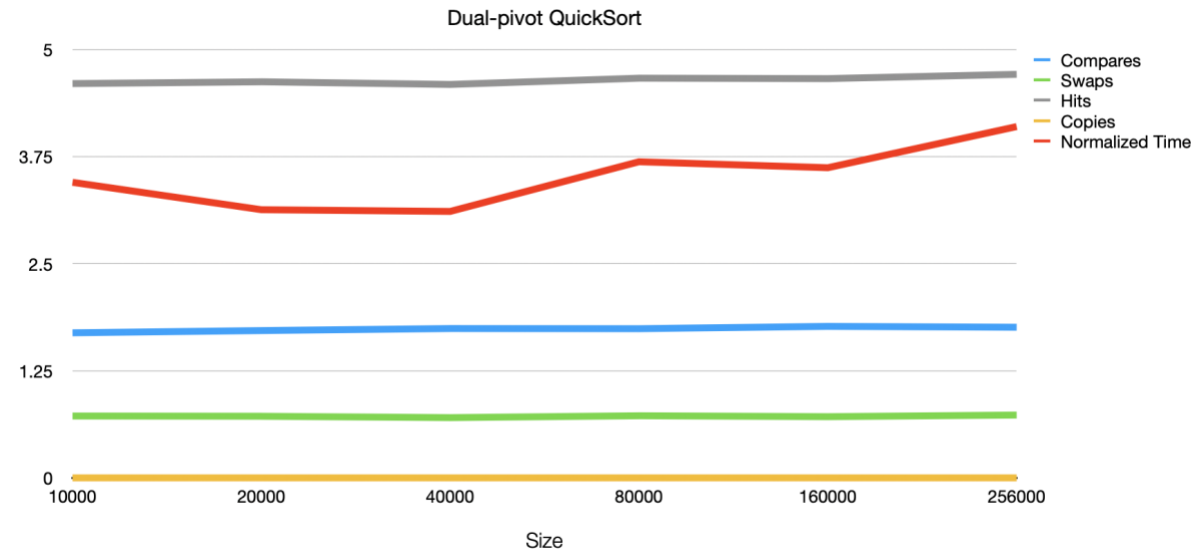
| Size | Compares | Swaps | Hits | Copies | Normalized Time |
|---|---|---|---|---|---|
| 10000 | 1.693 | 0.722 | 4.602 | 0.000 | 3.45 |
| 20000 | 1.719 | 0.719 | 4.623 | 0.000 | 3.13 |
| 40000 | 1.743 | 0.703 | 4.592 | 0.000 | 3.11 |
| 80000 | 1.741 | 0.725 | 4.665 | 0.000 | 3.69 |
| 160000 | 1.768 | 0.714 | 4.660 | 0.000 | 3.62 |
| 256000 | 1.758 | 0.733 | 4.710 | 0.000 | 4.10 |

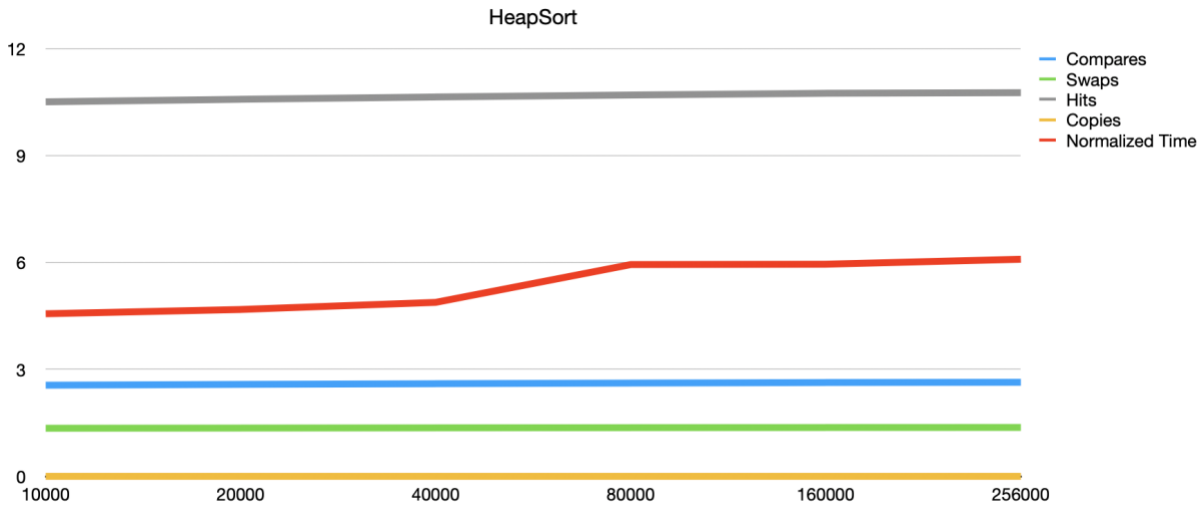
HeapSort

| Size | Compares | Swaps | Hits | Copies | Normalized Time |
|---|---|---|---|---|---|
| 10000 | 2.556 | 1.349 | 10.505 | 0 | 4.56 |
| 20000 | 2.579 | 1.355 | 10.577 | 0 | 4.68 |
| 40000 | 2.599 | 1.361 | 10.641 | 0 | 4.88 |
| 80000 | 2.616 | 1.366 | 10.696 | 0 | 5.94 |
| 160000 | 2.632 | 1.370 | 10.745 | 0 | 5.95 |
| 256000 | 2.638 | 1.371 | 10.762 | 0 | 6.09 |

**Graphical Representation:**

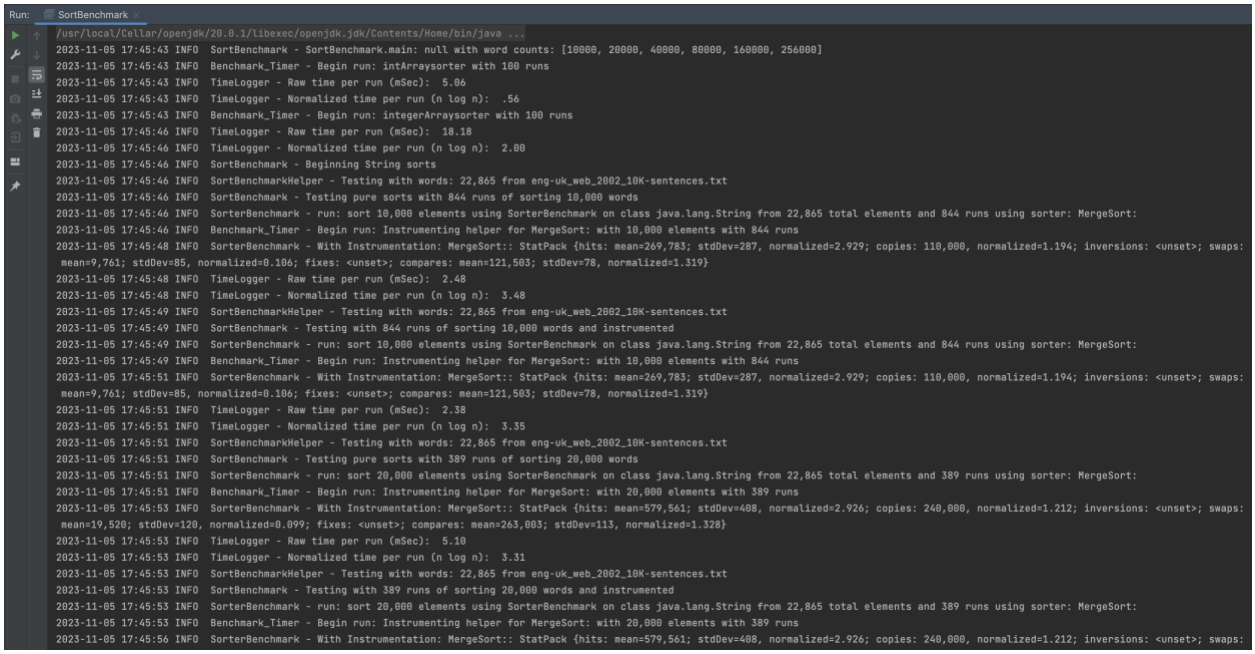


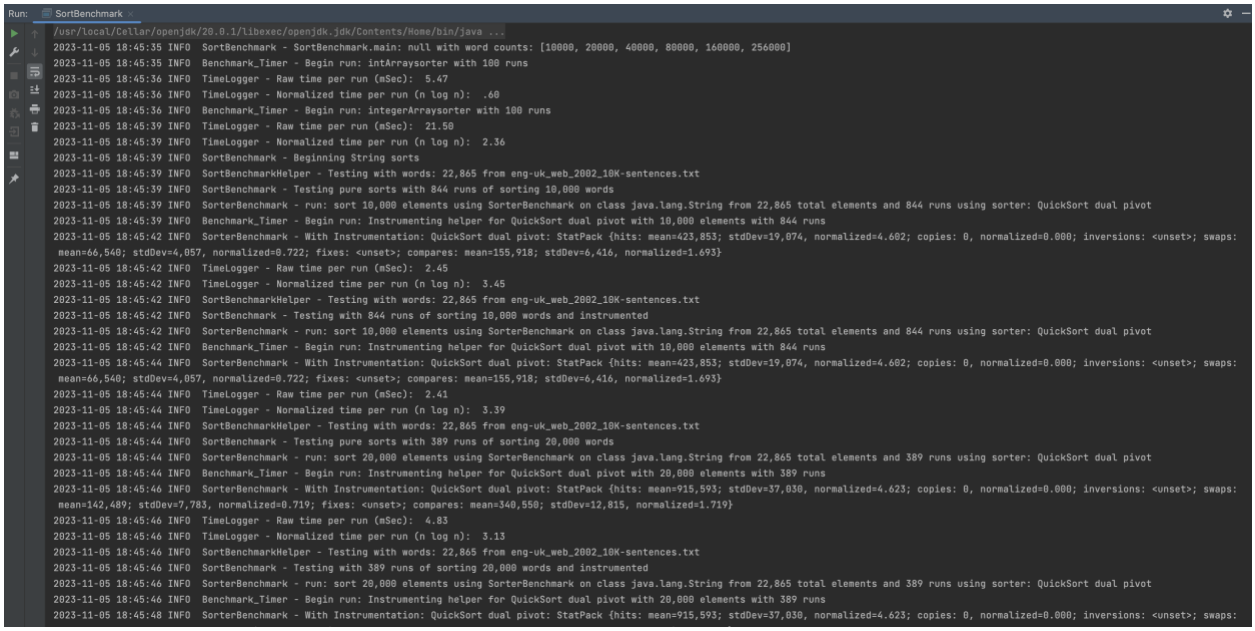



**Screenshots of run and/or Unit Test:**

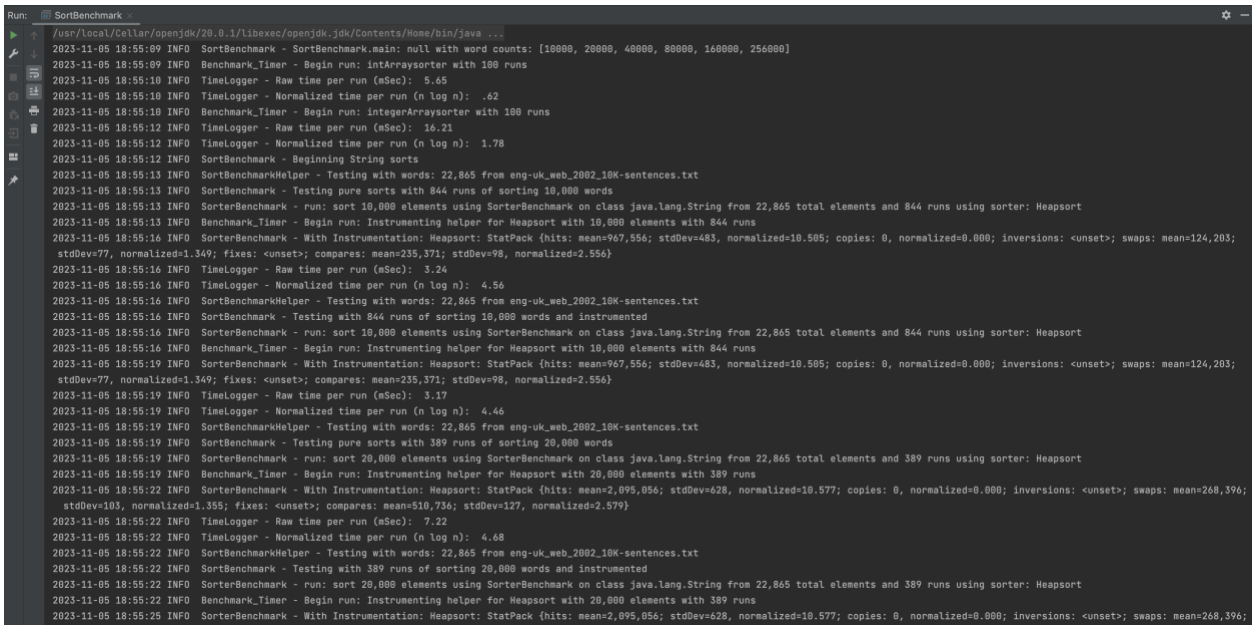
Merge Sort




Dual-pivot Quicksort



Heapsort


Run:
SortBenchmark
/usr/local/Cellar/openjdk/20.0.1/libexec/openjdk.jdk/Contents/Home/bin/java ...
2023-11-05 18:55:09 INFO SortBenchmark - SortBenchmark.main: null with word counts: [10000, 20000, 40000, 80000, 160000, 256000]
2023-11-05 18:55:09 INFO Benchmark_Timer - Begin run: intArraysorter with 100 runs
2023-11-05 18:55:10 INFO TimeLogger - Raw time per run (mSec): 5.65
2023-11-05 18:55:10 INFO TimeLogger - Normalized time per run (n log n): .62
2023-11-05 18:55:10 INFO Benchmark_Timer - Begin run: integerArraysorter with 100 runs
2023-11-05 18:55:12 INFO TimeLogger - Raw time per run (mSec): 16.21
2023-11-05 18:55:12 INFO TimeLogger - Normalized time per run (n log n): 1.78
2023-11-05 18:55:12 INFO SortBenchmark - Beginning String sorts
2023-11-05 18:55:13 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-11-05 18:55:13 INFO SortBenchmark - Testing pure sorts with 844 runs of sorting 10,000 words
2023-11-05 18:55:13 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: Heapsort
2023-11-05 18:55:13 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2023-11-05 18:55:16 INFO SorterBenchmark - With Instrumentation: Heapsort: StatPack {hits: mean=967,556; stdDev=483, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,203; stdDev=77, normalized=1.349; fixes: <unset>; compares: mean=235,371; stdDev=98, normalized=2.556}
2023-11-05 18:55:16 INFO TimeLogger - Raw time per run (mSec): 3.24
2023-11-05 18:55:16 INFO TimeLogger - Normalized time per run (n log n): 4.56
2023-11-05 18:55:16 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-11-05 18:55:16 INFO SortBenchmark - Testing with 844 runs of sorting 10,000 words and instrumented
2023-11-05 18:55:16 INFO SorterBenchmark - run: sort 10,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 844 runs using sorter: Heapsort
2023-11-05 18:55:16 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 10,000 elements with 844 runs
2023-11-05 18:55:19 INFO SorterBenchmark - With Instrumentation: Heapsort: StatPack {hits: mean=967,556; stdDev=483, normalized=10.505; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=124,203; stdDev=77, normalized=1.349; fixes: <unset>; compares: mean=235,371; stdDev=98, normalized=2.556}
2023-11-05 18:55:19 INFO TimeLogger - Raw time per run (mSec): 3.17
2023-11-05 18:55:19 INFO TimeLogger - Normalized time per run (n log n): 4.46
2023-11-05 18:55:19 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-11-05 18:55:19 INFO SortBenchmark - Testing pure sorts with 389 runs of sorting 20,000 words
2023-11-05 18:55:19 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: Heapsort
2023-11-05 18:55:19 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2023-11-05 18:55:22 INFO SorterBenchmark - With Instrumentation: Heapsort: StatPack {hits: mean=2,095,056; stdDev=628, normalized=10.577; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,396; stdDev=103, normalized=1.355; fixes: <unset>; compares: mean=510,736; stdDev=127, normalized=2.579}
2023-11-05 18:55:22 INFO TimeLogger - Raw time per run (mSec): 7.22
2023-11-05 18:55:22 INFO TimeLogger - Normalized time per run (n log n): 4.68
2023-11-05 18:55:22 INFO SortBenchmarkHelper - Testing with words: 22,865 from eng-uk_web_2002_10K-sentences.txt
2023-11-05 18:55:22 INFO SortBenchmark - Testing with 389 runs of sorting 20,000 words and instrumented
2023-11-05 18:55:22 INFO SorterBenchmark - run: sort 20,000 elements using SorterBenchmark on class java.lang.String from 22,865 total elements and 389 runs using sorter: Heapsort
2023-11-05 18:55:22 INFO Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 20,000 elements with 389 runs
2023-11-05 18:55:25 INFO SorterBenchmark - With Instrumentation: Heapsort: StatPack {hits: mean=2,095,056; stdDev=628, normalized=10.577; copies: 0, normalized=0.000; inversions: <unset>; swaps: mean=268,396;