Program Structures and Algorithms Spring 2024

Name : Shaofan Wei

NUID: 002815198

GITHUB LINK:https://github.com/Arthurccone123/INFO6205

Task:

This task asks us to complete the MergeSort algorithm and analyze and compare the performance of different sorting algorithms when dealing with randomly generated arrays of different sizes.

Relationship Conclusion:

Dual-pivot quick sort shows the best performance across various data set sizes, both in terms of raw time and normalized time.
Merge sort's performance closely follows, especially on larger data sets, where its performance approaches that of dual-pivot quick sort.
Heap sort performs the worst in all cases, particularly on larger data sets where the gap is more significant.

Evidence to support that conclusion:

Word Count	Sort Method	Raw Time per Run (mSec)	Normalized Time per Run (n log n)
10000	MergeSort: with no copy	2.44	3.43
10000	QuickSort dual pivot	1.94	2.73
10000	Heapsort	2.54	3.58
20000	MergeSort: with no copy	4.39	2.85
20000	QuickSort dual pivot	3.90	2.53
20000	Heapsort	5.21	3.38
40000	MergeSort: with no copy	9.51	2.86
40000	QuickSort dual pivot	8.37	2.52

Word Count	Sort Method	Raw Time per Run (mSec)	Normalized Time per Run (n log n)
40000	Heapsort	11.48	3.45
80000	MergeSort: with no copy	25.41	3.57
80000	QuickSort dual pivot	20.35	2.86
80000	Heapsort	31.83	4.47
160000	MergeSort: with no copy	54.24	3.57
160000	QuickSort dual pivot	42.66	2.80
160000	Heapsort	69.04	4.54

```
2024-03-15 19:27:17 INFO  SorterBenchmark - run: sort 80,000 elements using SorterBenchmark on class java.lang.St
2024-03-15 19:27:17 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 80,000
2024-03-15 19:27:20 INFO  TimeLogger - Raw time per run (mSec):  19.12
2024-03-15 19:27:20 INFO  TimeLogger - Normalized time per run (n log n):  2.68
2024-03-15 19:27:20 INFO  SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.tx
2024-03-15 19:27:20 INFO  SortBenchmark - Testing pure sorts with 39 runs of sorting 160,000 words
2024-03-15 19:27:20 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.S
2024-03-15 19:27:20 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with no copy with 160,
2024-03-15 19:27:22 INFO  TimeLogger - Raw time per run (mSec):  54.24
2024-03-15 19:27:22 INFO  TimeLogger - Normalized time per run (n log n):  3.57
2024-03-15 19:27:22 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.S
2024-03-15 19:27:22 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 160,000
2024-03-15 19:27:24 INFO  TimeLogger - Raw time per run (mSec):  42.66
2024-03-15 19:27:24 INFO  TimeLogger - Normalized time per run (n log n):  2.80
2024-03-15 19:27:24 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.S
2024-03-15 19:27:24 INFO  Benchmark_Timer - Begin run: Instrumenting helper for Heapsort with 160,000 elements wi
2024-03-15 19:27:28 INFO  TimeLogger - Raw time per run (mSec):  69.04
2024-03-15 19:27:28 INFO  TimeLogger - Normalized time per run (n log n):  4.54
2024-03-15 19:27:28 INFO  SortBenchmarkHelper - Testing with words: 81,546 from eng-uk_web_2002_100K-sentences.tx
2024-03-15 19:27:28 INFO  SortBenchmark - Testing with 39 runs of sorting 160,000 words and instrumented
2024-03-15 19:27:28 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.S
2024-03-15 19:27:28 INFO  Benchmark_Timer - Begin run: Instrumenting helper for MergeSort: with no copy with 160,
2024-03-15 19:27:31 INFO  TimeLogger - Raw time per run (mSec):  56.96
2024-03-15 19:27:31 INFO  TimeLogger - Normalized time per run (n log n):  3.74
2024-03-15 19:27:31 INFO  SorterBenchmark - run: sort 160,000 elements using SorterBenchmark on class java.lang.S
2024-03-15 19:27:31 INFO  Benchmark_Timer - Begin run: Instrumenting helper for QuickSort dual pivot with 160,000
2024-03-15 19:27:33 INFO  TimeLogger - Raw time per run (mSec):  46.65
2024-03-15 19:27:33 INFO  TimeLogger - Normalized time per run (n log n):  3.07
```

Here is the code:

MergeSort:

```
private void sort(X[] a, X[] aux, int from, int to) {
        final Helper<X> helper = getHelper();
        Config config = helper.getConfig();
        boolean insurance = config.getBoolean(MERGESORT, INSURANCE);
        boolean noCopy = config.getBoolean(MERGESORT, NOCOPY);
        if (to <= from + helper.cutoff()) {
            insertionSort.sort(a, from, to);
            return;
        }
        int mid = from + (to - from ) / 2;
        sort(aux, a, from, mid);
        sort(aux, a, mid, to);
```

```
        merge(aux, a, from, mid, to);
        // TO BE IMPLEMENTED  : implement merge sort wit insurance and no-copy
optimizations
    }
```

SortBenchmark:

```
void benchmarkStringSorters(String[] words, int nWords, int nRuns) {
        logger.info("Testing pure sorts with " + formatWhole(nRuns) + " runs of
sorting " + formatWhole(nWords) + " words");
        Random random = new Random();

        if (isConfigBenchmarkStringSorter("puresystemsort"))
            runPureSystemSortBenchmark(words, nWords, nRuns, random);

        if (isConfigBenchmarkStringSorter("mergesort"))
            runMergeSortBenchmark(words, nWords, nRuns,
isConfigBenchmarkMergeSort("insurance"), isConfigBenchmarkMergeSort("nocopy"));

        if (isConfigBenchmarkStringSorter("quicksort3way"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_3way<>
(nWords, config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("quicksortDualPivot"))
            runStringSortBenchmark(words, nWords, nRuns, new
QuickSort_DualPivot<>(nWords, config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("quicksort"))
            runStringSortBenchmark(words, nWords, nRuns, new QuickSort_Basic<>
(nWords, config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("heapsort")) {
            Helper<String> helper = HelperFactory.create("Heapsort", nWords,
config);
            runStringSortBenchmark(words, nWords, nRuns, new HeapSort<>(helper),
timeLoggersLinearithmic);
        }

        if (isConfigBenchmarkStringSorter("introsort"))
            runStringSortBenchmark(words, nWords, nRuns, new IntroSort<>(nWords,
config), timeLoggersLinearithmic);

        if (isConfigBenchmarkStringSorter("randomsort"))
            runStringSortBenchmark(words, nWords, nRuns, new RandomSort<>(nWords,
config), timeLoggersLinearithmic);

        // NOTE: this is very slow of course, so recommendation is not to enable
this option.
        if (isConfigBenchmarkStringSorter("insertionsort"))
            runStringSortBenchmark(words, nWords, nRuns / 10, new InsertionSort<>
(nWords, config), timeLoggersQuadratic);

        // NOTE: this is very slow of course, so recommendation is not to enable
this option.
        if (isConfigBenchmarkStringSorter("bubblesort"))
```

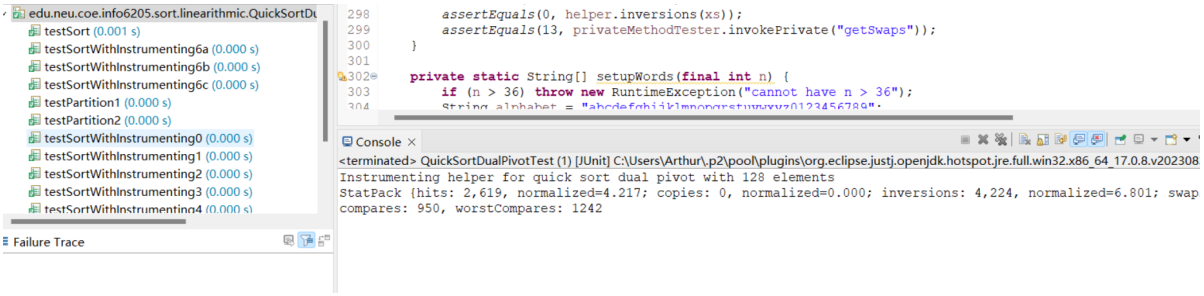
```
            runStringSortBenchmark(words, nWords, nRuns / 10, new BubbleSort<>
(nWords, config), timeLoggersQuadratic);

    }
```

config:

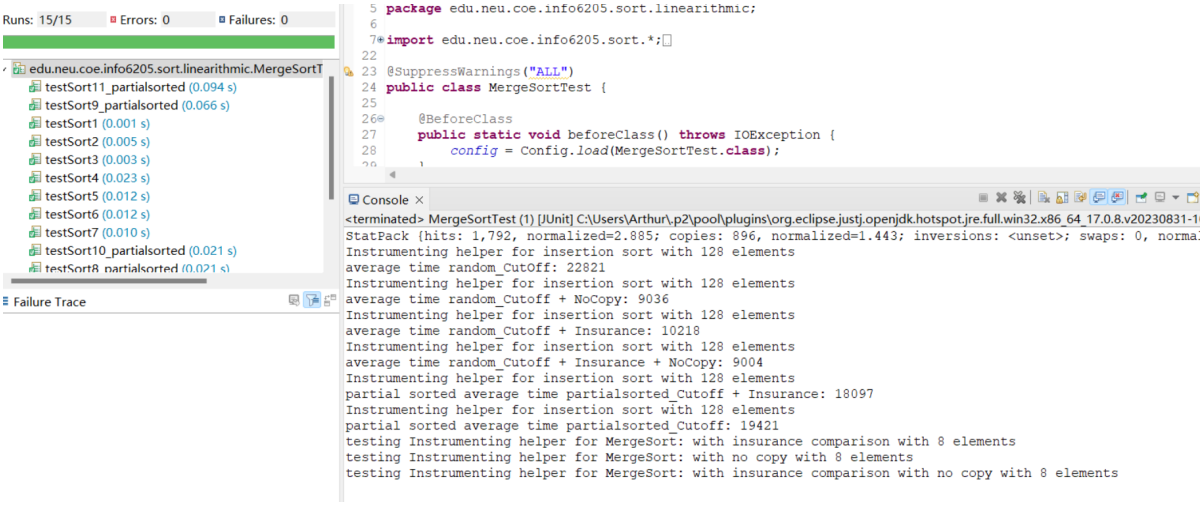
```
[sortbenchmark]
version = 1.0.0 (sortbenchmark)

[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
timsort = false
quicksort = false
introsort = false
insertionsort = false
bubblesort = false
quicksort3way = false
quicksortDualPivot = true
randomsort = false
heapsort = true
mergesort = true

[benchmarkdatesorters]
timsort = false
n = 100000

[mergesort]
insurance = false
nocopy = true

[shellsort]
n = 100000

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

Screen-shot:

QuickSortDualPivotTest:



MergeSortTest:



HeapSortTest: