# 2021 Fall INFO6205-Final Project Report

**Team member**

| | |
|---|---|
| Peng Chen | 001098655 |
| Fengyi Zhang | 001564247 |
| Shibo Lu | 001548075 |


## 1. Requirement Abstract

Your task is to implement MSD radix sort for a natural language which uses Unicode characters. You may choose your own language or (Simplified) Chinese. Additionally, you will complete a literature survey of relevant papers and you will compare your method with Timsort, Dual-pivot Quicksort, Huskysort, and LSD radix sort.

## 2. Methodology

The project aims to sort person names in Chinese with five sort algorithms, such as MSD radix sort, LSD radix sort, Dual-pivot Quicksort, Huskysort and Timsort.

Firstly, we wrote a Java class to read Chinese person names from a txt file, and stored the names as a String[].

Secondly, to sort Chinese names, we transferred Chinese words, a String[], to pinyins, a String[], by pinyin4j which is a Java package. In the meanwhile, we built a HashMap to save the relationship between the Chinese person names and the pinyins. The keys of HashMap are pinyins, and the values of HashMap are Chinese person names corresponding to the key, pinyin. Because of polyphone in Chinese, one pinyin may correspond to multiple Chinese names. Thus, we used String[] to store Chinese names as the value of HashMap, such as, HashMap = {“a1bin1”: [“阿斌”, “阿彬”]}.

Thirdly, we wrote MSD radix sort and Dual-pivot Quicksort methods, modified LSD radix sort method and directly referred Huskysort and Timsort. And then, we ran these sort algorithms with pinyins, a String[], as input. After the sorting step, we got sorted pinyins.

Finally, we can translate the sorted pinyins back to Chinese person names via the HashMap which we have gotten in second step. The Chinese person names are sorted.

We also wrote the unit test for each sorting algorithms and a benchmark to test the run time of these algorithms. We benchmarked the results of all methods for 250k, 500k and 1M, 2M, 4M names. We recorded the first 1000 sorted Chinese name into five txt files.

## 3. Implement

When we were executing our methodology, we used pinyin without tone at first. However, there are many Chinese words share same pinyin, but different tones. So the pinyin without tone may cause inaccurate sort results. For example, without tone, both the pinyin of “阿琛” and the pinyin of “阿臣” are “achen”. With tone, the pinyin of “阿琛” is “a1chen1” and the pinyin of “阿臣” is “a1chen2”. Thus, we decided to transfer Chinese words to the pinyin with tone by pinyin4j, which made the accuracy of our sorting method get better.

We also found there are only 999998 Chinese names in the shuffledChinese.txt, not exact 1M names. We benchmarked the results of all methods for 250k, 500k and 1M, 2M, 4M names. For 1M, 2M and 4M part, we only used 999998, 999998 * 2 and 999998 * 4 names.

## 4. Result Analysis

According to analyzing the result of sorting, we found that the efficiency of single sorting algorithm is limited. By combining different sorting thoughts, and by having different sorting algorithms on applying to different data types could amplify the stability of sorting algorithms and contribute to the algorithm efficiency as well.

Figure 1: Relative sort times for Huskysort, dual-pivot quicksort, system sort

| N | Huskysort | DP quicksort | system sort |
|---|---|---|---|
| 1000 | 0.60 | 1.00 | 0.84 |
| 2000 | 0.78 | 1.00 | 1.19 |
| 4000 | 0.81 | 1.00 | 1.23 |
| 8000 | 0.77 | 1.00 | 1.27 |
| 16000 | 0.74 | 1.00 | 1.24 |
| 32000 | 0.74 | 1.00 | 1.24 |
| 64000 | 0.85 | 1.00 | 1.29 |
| 125000 | 0.76 | 1.00 | 1.51 |
| 250000 | 0.92 | 1.00 | 1.32 |
| 500000 | 0.97 | 1.00 | 1.42 |
| 1000000 | 0.95 | 1.00 | 1.45 |
| 2000000 | 0.93 | 1.00 | 1.54 |
| 4000000 | 0.91 | 1.00 | 1.64 |

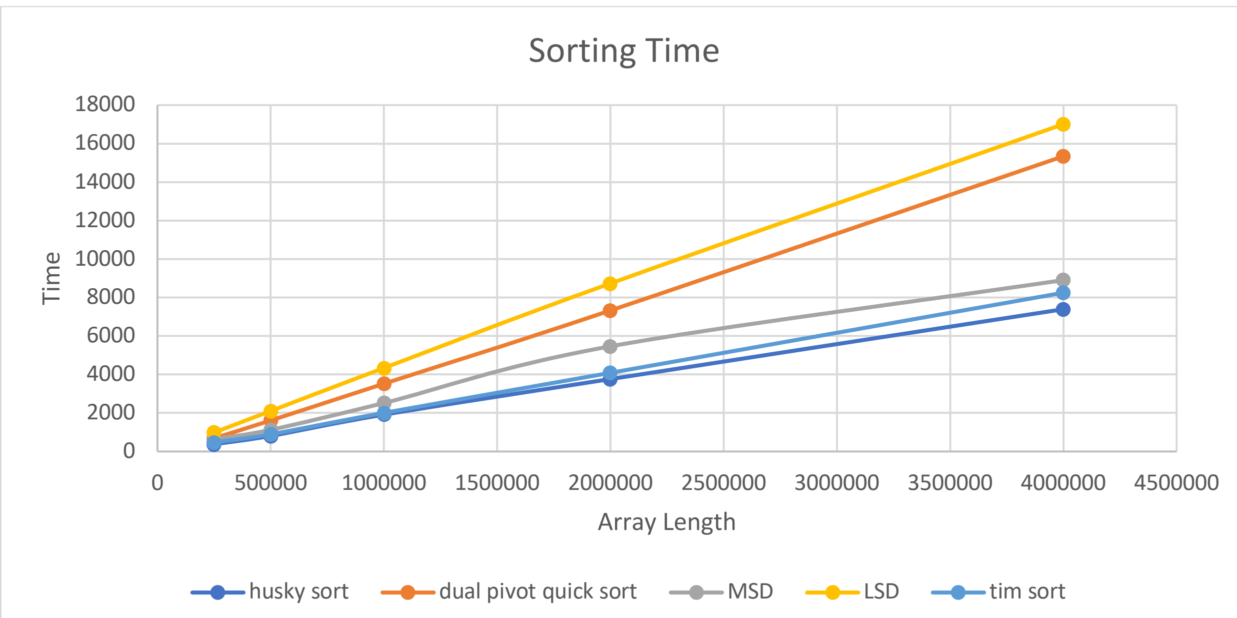
In Husky sort, the fastest algorithm of our 5 different sorting algorithms, they reduce the number of expensive comparisons in the linearithmic phase as much as possible by using huskyCode to help compare the types of objects which are expensive to compare. There will be very few inversions (elements out of place) remaining after the dual-pivot quicksort phase [1]. Then using timsort to make all the data sorted which is very useful to sort the part-sorted data.

In the Figure 1, we can see that the Huskysort extends the advantage both of dual-pivot quicksort and Timsort.

Finding the part which waste a lot of time and replace it by efficient way is a very useful way to improve the data sorting process.

# 5. Conclusion

As the calculation time of different sorting methods, we found that times of all sorting methods increase linearly as the array length increases doubly. Husky sort uses least time to sort the array we passed in. Timsort, MSD radix sort, dual pivot quick sort gets the 2$^{nd}$, 3$^{rd}$ and 4$^{th}$ place of the sorting competition. LSD radix sort is the slowest sorting algorithm of these 5 algorithms set.




# 6. Reference


[1] Hillyard, R., Liaozheng, Y. and Vineeth K.R, S., 2020. Huskysort. Professor. Northeastern University.


# 7. Appendix

## Code File Path:

**MSD radix sort**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/MSDStringSort.java

Unit Test: INFO6205-Final-ZCL/INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/sort/finalProject/MSDStringSortTest.java

**LSD radix sort**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/LSDStringSort.java

Unit Test: INFO6205-Final-ZCL/INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/sort/finalProject/LSDStringSortTest.java

**Dual-pivot Quicksort**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/DualPivotQuickSort.java

Unit Test: INFO6205-Final-ZCL/INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/sort/finalProject/QuickSortDualPivotTest.java

**Huskysort**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/huskySort/sort/huskySort/PureHuskySort.java

Unit Test: INFO6205-Final-ZCL/INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/sort/finalProject/huskySort/sort/huskySort/PureHuskySortTest.java

**Timsort**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/TimSort.java

Unit Test: INFO6205-Final-ZCL/INFO6205-Fall2021/src/test/java/edu/neu/coe/info6205/sort/finalProject/TimSortTest.java

**Benchmark**

Code: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/SortingTest.java

**Transfer to Pinyin**: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/Pinyin.java

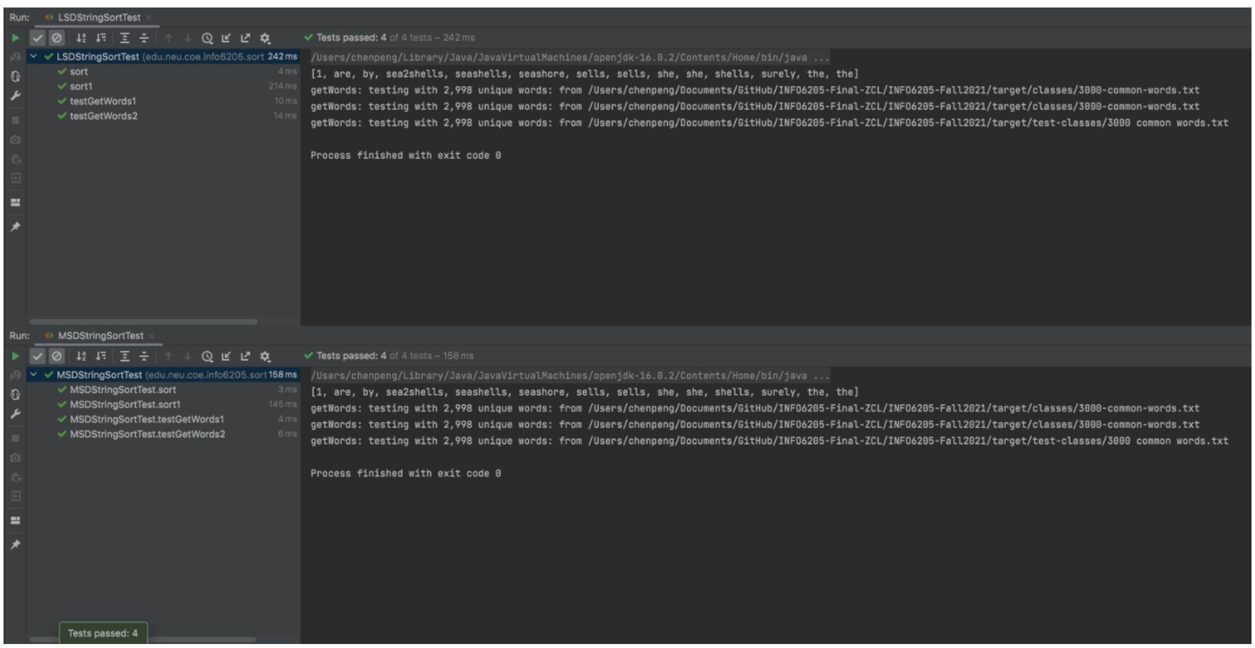
**Read from a Txt File**: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/ReadTxt.java

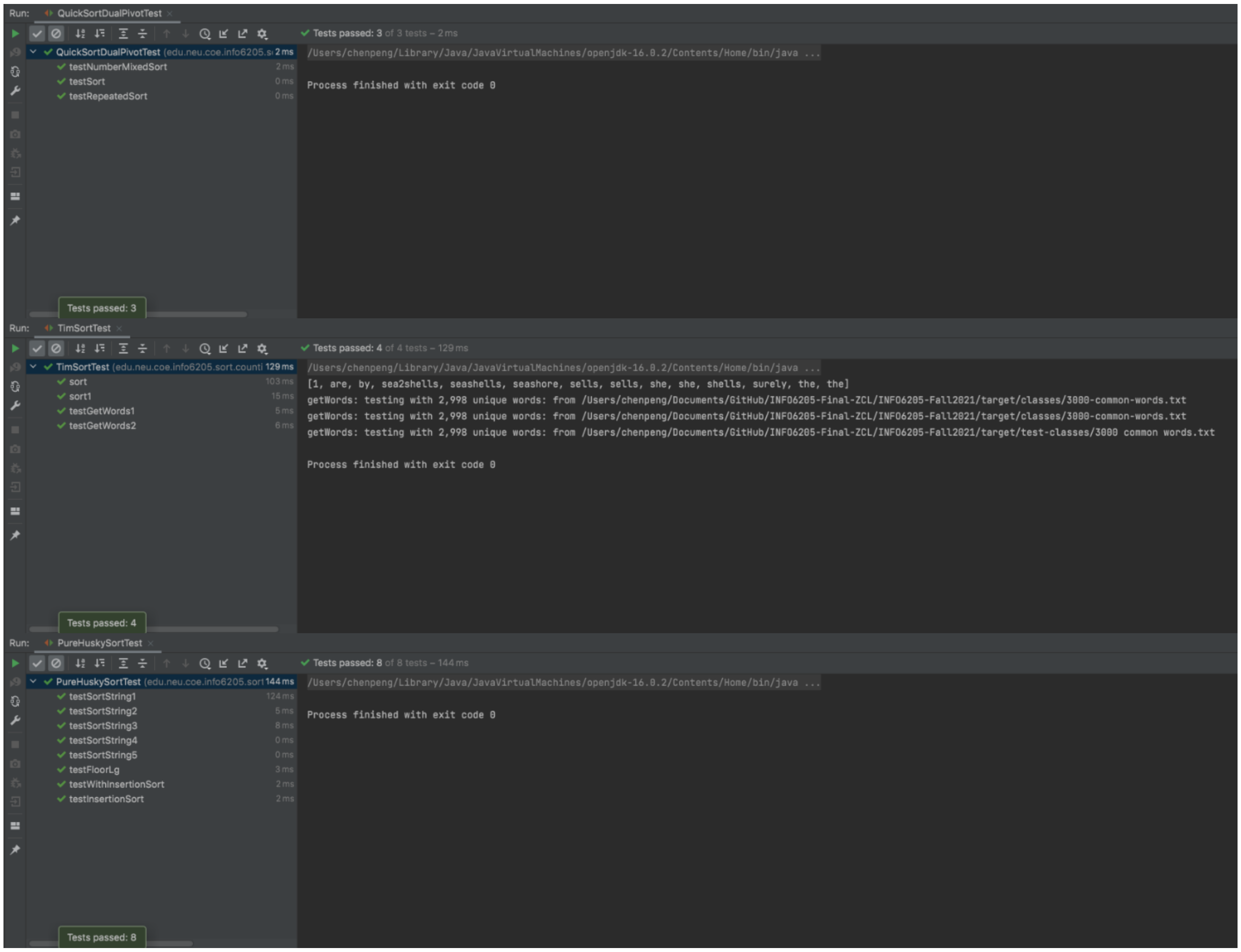
**Write from a Txt File**: INFO6205-Final-ZCL/INFO6205-Fall2021/src/main/java/edu/neu/coe/info6205/sort/finalProject/WriteTxt.java

## Outputs:

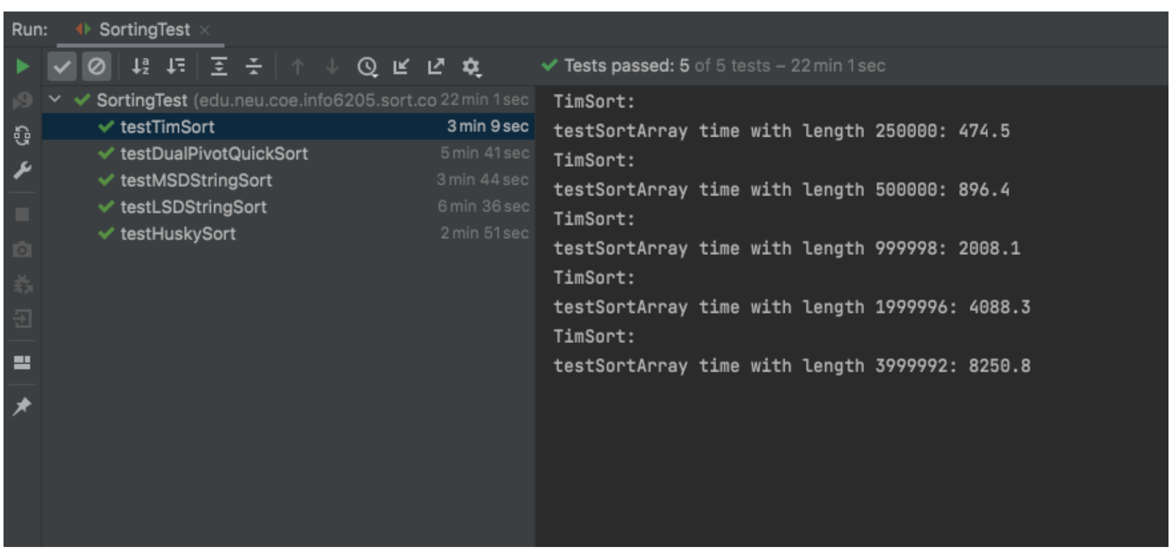
**MSD radix sort**: INFO6205-Final-ZCL/MSDStringSorted.txt
**LSD radix sort**: INFO6205-Final-ZCL/LSDStringSorted.txt
**Dual-pivot Quicksort**: INFO6205-Final-ZCL/DualPivotQuickSorted.txt
**Huskysort**: INFO6205-Final-ZCL/PureHuskySorted.txt
**Timsort**: INFO6205-Final-ZCL/TimSorted.txt

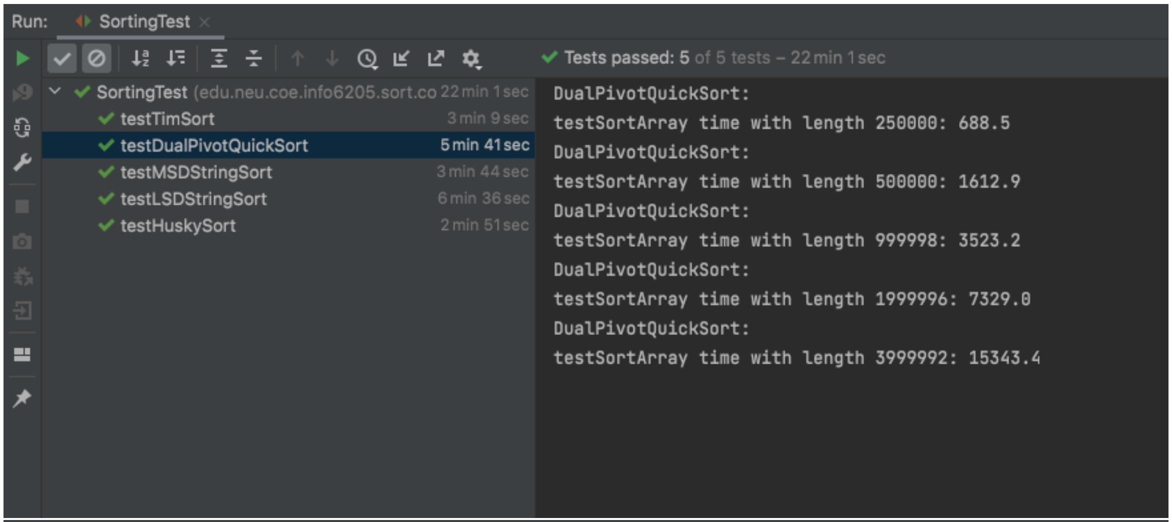
## Unit Tests Evidence:

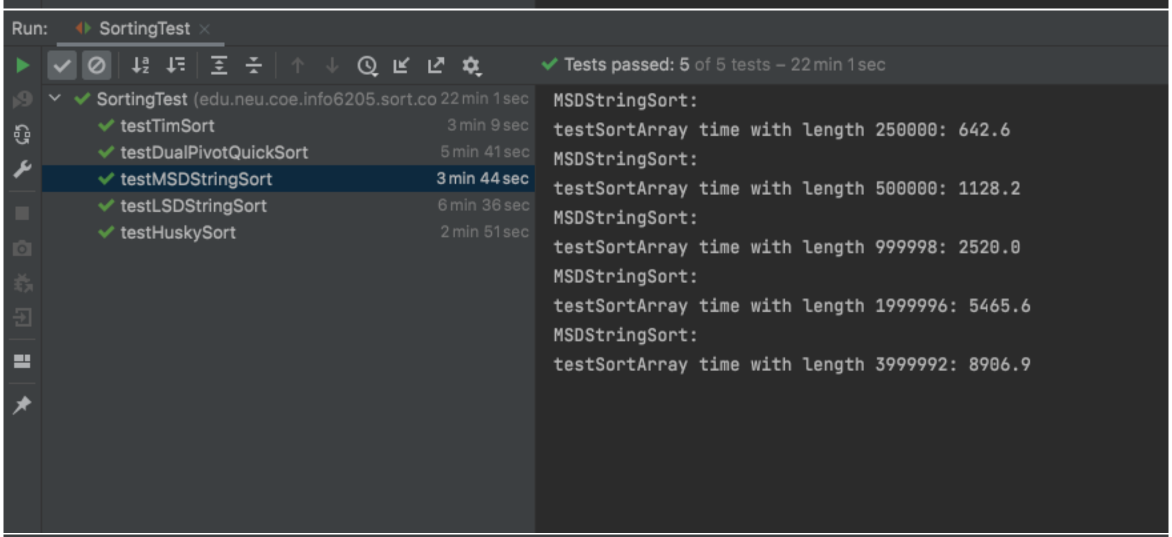


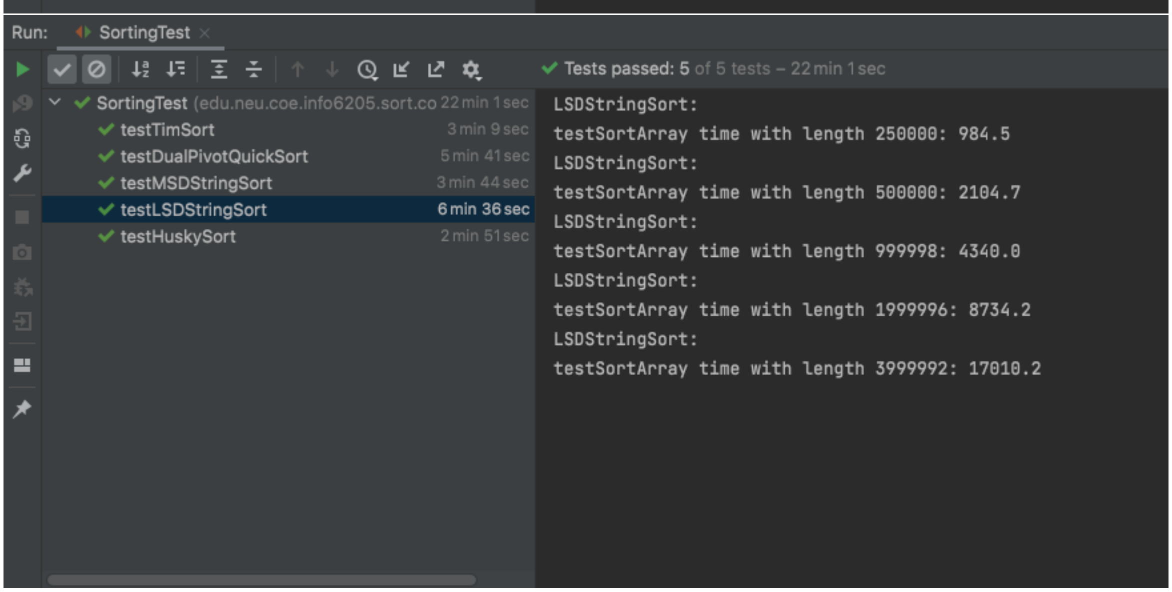


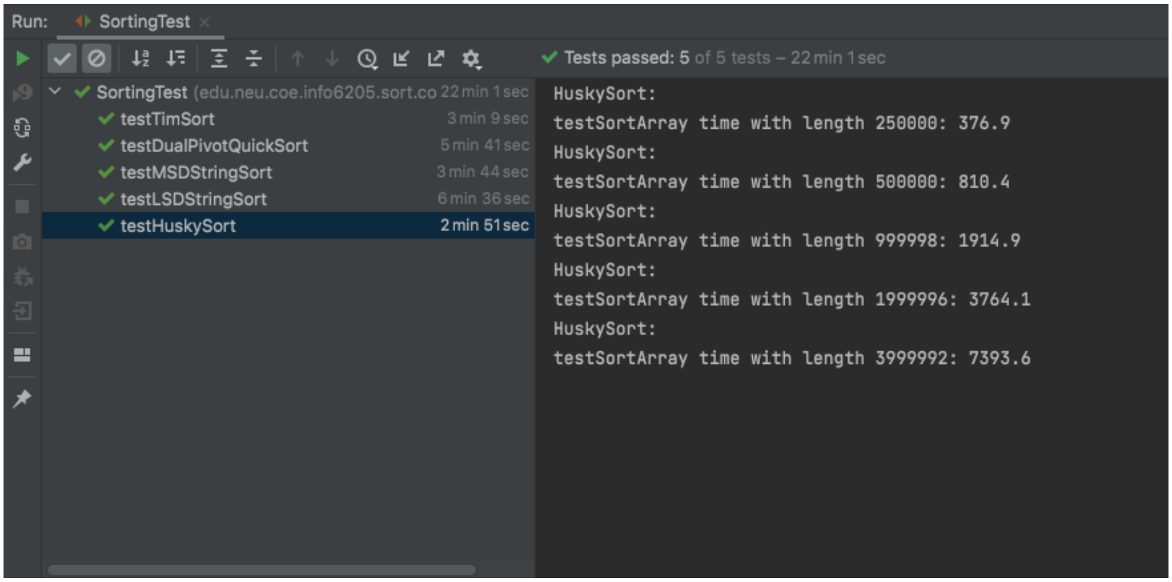

## Benchmarks Evidence:




Run: SortingTest
Tests passed: 5 of 5 tests – 22 min 1 sec
SortingTest (edu.neu.coe.info6205.sort.co 22 min 1 sec
testTimSort 3 min 9 sec
testDualPivotQuickSort 5 min 41 sec
testMSDStringSort 3 min 44 sec
testLSDStringSort 6 min 36 sec
testHuskySort 2 min 51 sec
DualPivotQuickSort:
testSortArray time with length 250000: 688.5
DualPivotQuickSort:
testSortArray time with length 500000: 1612.9
DualPivotQuickSort:
testSortArray time with length 999998: 3523.2
DualPivotQuickSort:
testSortArray time with length 1999996: 7329.0
DualPivotQuickSort:
testSortArray time with length 3999992: 15343.4


Run: SortingTest
Tests passed: 5 of 5 tests – 22 min 1 sec
SortingTest (edu.neu.coe.info6205.sort.co 22 min 1 sec
testTimSort 3 min 9 sec
testDualPivotQuickSort 5 min 41 sec
testMSDStringSort 3 min 44 sec
testLSDStringSort 6 min 36 sec
testHuskySort 2 min 51 sec
MSDStringSort:
testSortArray time with length 250000: 642.6
MSDStringSort:
testSortArray time with length 500000: 1128.2
MSDStringSort:
testSortArray time with length 999998: 2520.0
MSDStringSort:
testSortArray time with length 1999996: 5465.6
MSDStringSort:
testSortArray time with length 3999992: 8906.9


Run: SortingTest
Tests passed: 5 of 5 tests – 22 min 1 sec
SortingTest (edu.neu.coe.info6205.sort.co 22 min 1 sec
testTimSort 3 min 9 sec
testDualPivotQuickSort 5 min 41 sec
testMSDStringSort 3 min 44 sec
testLSDStringSort 6 min 36 sec
testHuskySort 2 min 51 sec
LSDStringSort:
testSortArray time with length 250000: 984.5
LSDStringSort:
testSortArray time with length 500000: 2104.7
LSDStringSort:
testSortArray time with length 999998: 4340.0
LSDStringSort:
testSortArray time with length 1999996: 8734.2
LSDStringSort:
testSortArray time with length 3999992: 17010.2


Run: SortingTest
Tests passed: 5 of 5 tests – 22 min 1 sec
SortingTest (edu.neu.coe.info6205.sort.co 22 min 1 sec
testTimSort 3 min 9 sec
testDualPivotQuickSort 5 min 41 sec
testMSDStringSort 3 min 44 sec
testLSDStringSort 6 min 36 sec
testHuskySort 2 min 51 sec
HuskySort:
testSortArray time with length 250000: 376.9
HuskySort:
testSortArray time with length 500000: 810.4
HuskySort:
testSortArray time with length 999998: 1914.9
HuskySort:
testSortArray time with length 1999996: 3764.1
HuskySort:
testSortArray time with length 3999992: 7393.6