Radix Sort for a Natural Language

Final Project Report Fall 2021

Authors: Harshal Jaiswal, Sahil Sonawane, Mohit Negi

NUID: 001097734, 002193783, 001549103

Task: -

1. To implement MSD Radix Sort for Chinese language.
2. Implement Tim Sort, Dual-Pivot, Quick Sort, Husky Sort and LSD Radix Sort.
3. Benchmark using all these algorithms for various input size.
4. Compare the benchmarking results.

Conclusion: -

According to benchmarking results and below graphs we can conclude that the quickest sorting algorithm are in the following order: -

1) Radix LSD Sort
2) Radix MSD Sort
3) Dual-Pivot
4) Tim Sort
5) Merge Sort
6) Husky Sort

Implementation of Sorting methods: -

1. **Pre Processing:**

By passing the path of the file "Chinese_names.txt" as resources to :

```
String[] pin= ChineseToEnglish.generateList(resource)
```

We obtain the Chinese characters in an array format. Further we use the Pinyin4j library to convert these Chinese names into English names. This pre processing step is common for all the sorting algorithms that we use.

The newly formed array of English characters is what we pass as arguments to the sorting methods.

2. **Sorting Algorithms Implemented:**
    a. Tim Sort:
        i. We use four different classes for the implementation of the Tim sort.
        ii. Firstly, for pre processing
        iii. Secondly, we use the Tim sort function to initiate the sorting function.
        iv. Thirdly, we require us to extend our sort class to standard Sort.java class.
        v. Lastly, we use the post processing class for conversion of English names to Chinese and then to write our result in a CSV file that we save in the resources.
    b. MSD Radix Sort:
        i. We use same pre and post processing as we did in Tim sort
        ii. Firstly, for pre processing
        iii. Secondly, we use the RadixMSD sort function to initiate the sorting function.
        iv. Thirdly, we require the use of InsertionSorting algorithm in order for us to implement the radix sort. algo.
    c. LSD Radix Sort:
        i. We use same pre and post processing as we did in Tim sort
        ii. Secondly, we use the RadixLSD sort function to initiate the sorting function.
        iii. The LSD radix sort that we wrote works for strings that have the same length therefore we assert the length of the elements in the array to 3 which is the length of the shortest word in the array.

d. Dual-Pivot: -
    i. We use same pre and post processing as we did in Tim sort
    ii. Secondly, we use the sort function which intern calls less, exch, isSorted function to perform the dual quick pivot sort.
e. Husky Sort:
    i. We use same pre and post processing as we did in Tim sort
    ii. Here, we use the PureHuskySort java provided by Prof.Robin to implement our benchmarking and then t compare the results.
f. 3-way Quick Radix Sort:
    i. 3-way Quick Radix Sort: We use same pre and post processing as we did in Tim sort
    ii. Secondly, we use the ThreeWayQuick sort, swap and charAt function to initiate the sorting function.
g. Merge Sort:
    i. We use same pre and post processing as we did in Tim sort
    ii. Secondly, we use the Merge sort function to initiate the sorting function.
    iii. The sorting here is different from all the other sorting algo we wrote. Here we divide the array into two equal sized array and then we sort each of them individually and then merge the two into a single array.

3. Post processing:

After the sorting algorithm produces the sorted array, we go for a road not taken.

`In general case we would use a pre-existing library to convert the English names to the Chinese names but in the initial case we observed that takes a lot of time and processing power goes to waste. Therefore, we implement binary search algorithms from the EnglishToChinese.java class:`

- Unsorted Chinese Array {Pin}
- Unsorted English Names {beSort}
- Sorted English names {chiToEng}
- Sorted Chinese Names {res}

Steps for the post processing:

- We identify the position of the word from the beSort array in the chiToEng array from binary search then we sequentially assign the empty res array with the elements from the Pin array from the identified position.

Our Observations: -

- Benchmarking works differently on different processors. All the below readings were benchmarked on below machine: -
- Processer: - Intel(R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz
- RAM: 16GB, 2400 MHz
- 64-bit operating system, x64-based processor
- GPU: - NVIDIA GeForce GTX 1050ti 4GB
- Benchmarking Husky Sort was taking longer time for our set of run times we could gather very limited data as it was taking hours to execute for 200 runs and henceforth. We were able to benchmark for 100 runs till 4 million of data.
- We created different functions to time the pre and post processing of the sorting of various algorithms and added the timings to the sort time. Pre-processing which includes converting the string to pinyin and the post includes converting it back to Chinese characters and adding the data in a CSV file.

Output & Graphical Representation: -

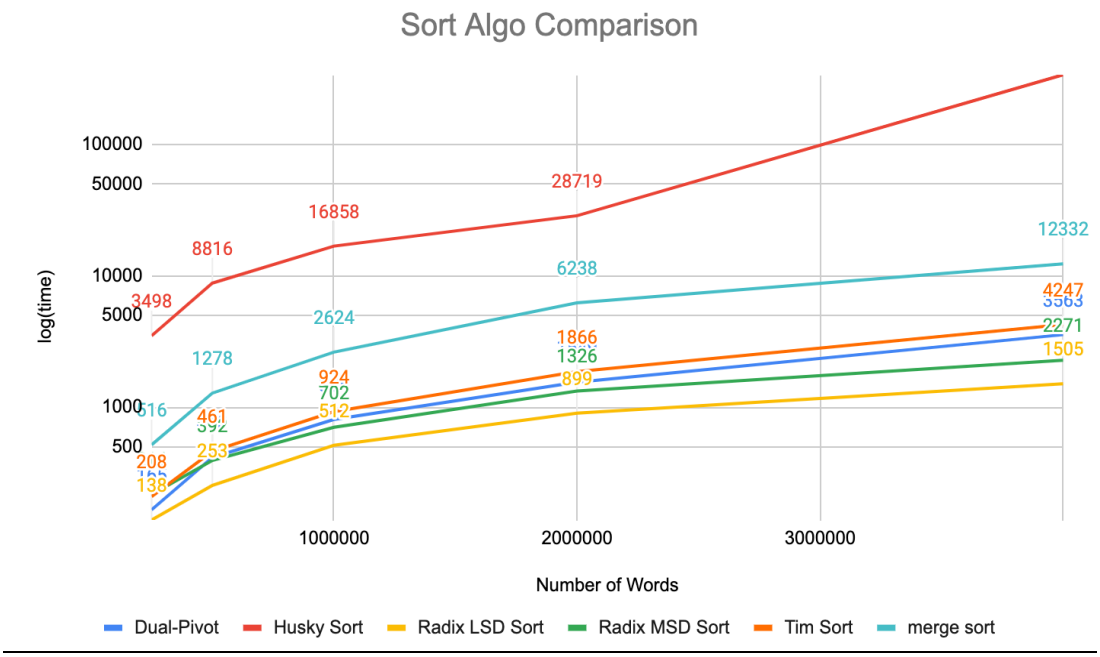

Fig 1: Sorting Algorithms Comparison

From the graph above it can be observed that the sorting technique with least processing time to highest processing time are as follows:

1) Radix LSD Sort
2) Radix MSD Sort
3) Dual-Pivot
4) Tim Sort
5) Merge Sort
6) Husky Sort

The following data was observed on Pre-process and Post process:

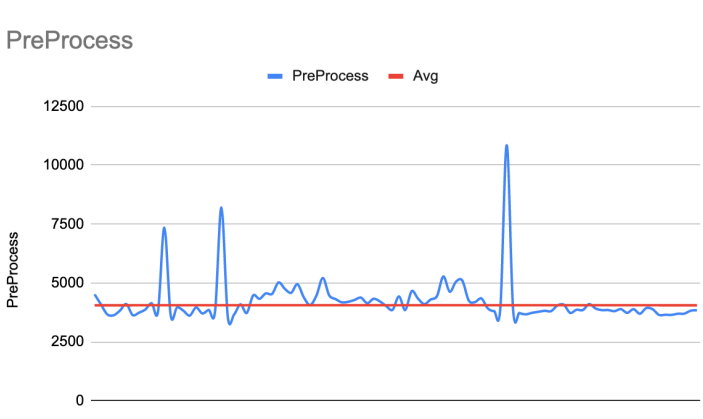

Fig 2: Preprocess time

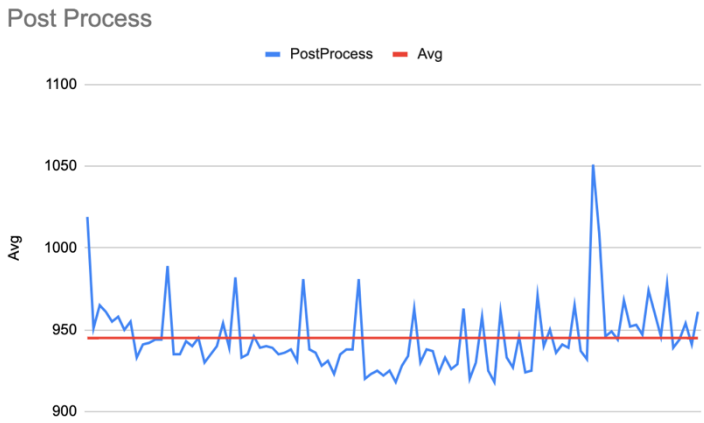

Fig 3: Postprocess time

From the above graph it can be observed that average time for Pre-processing time for 100 runs is nearly equal to 4056 ms and for the Post-processing it was nearly 945 ms.

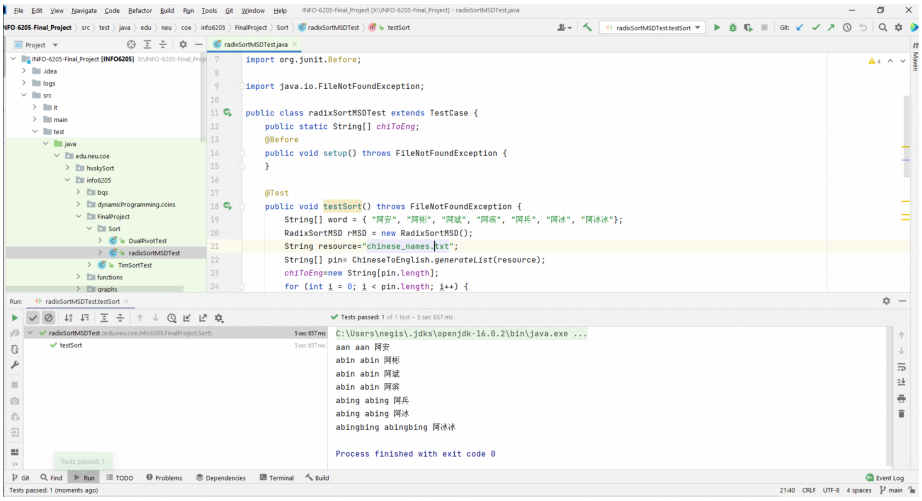

Fig 4: Radix MSD Sort test verification

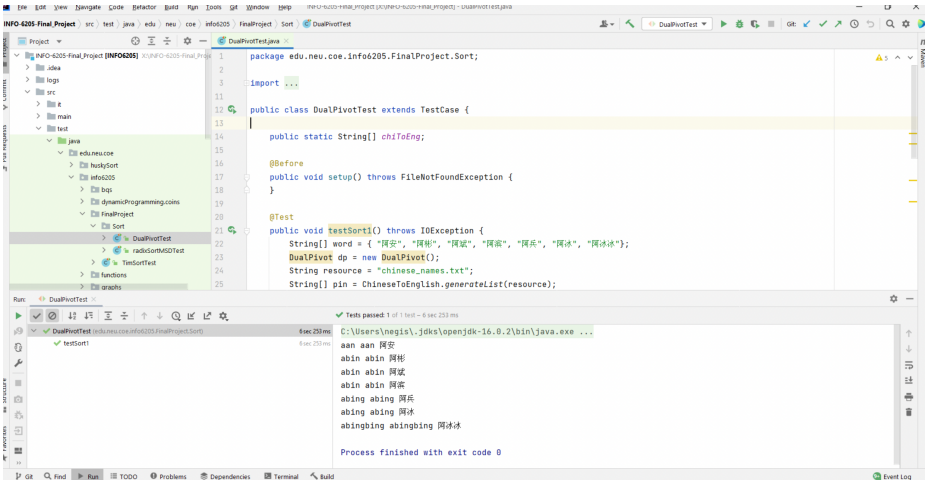

Fig 5: Dual Pivot Sort test verification

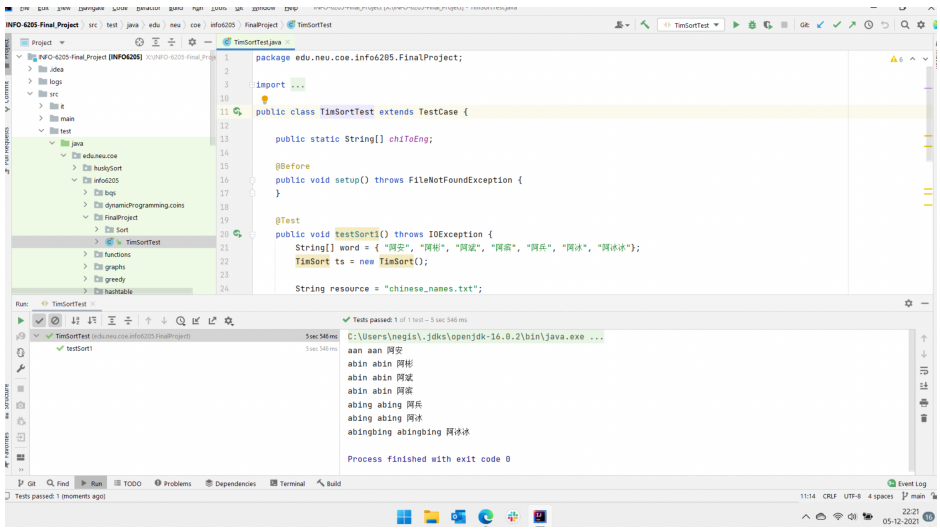

Fig 6: Tim Sort test verification

Output for Test Cases:

1. Tim sort: all test cases passed
2. Dual Pivot: all test cases passed
3. Radix Sort MSD: all test cases passed

Comment: These 3 were written by us therefore, we wrote the test cases class in order to verify our sorting output produced by the algorithms. The test cases for other sorting sorting algorithms were not written because the algo's were written in the Prof.Robin's codebase and had the test cases provided.

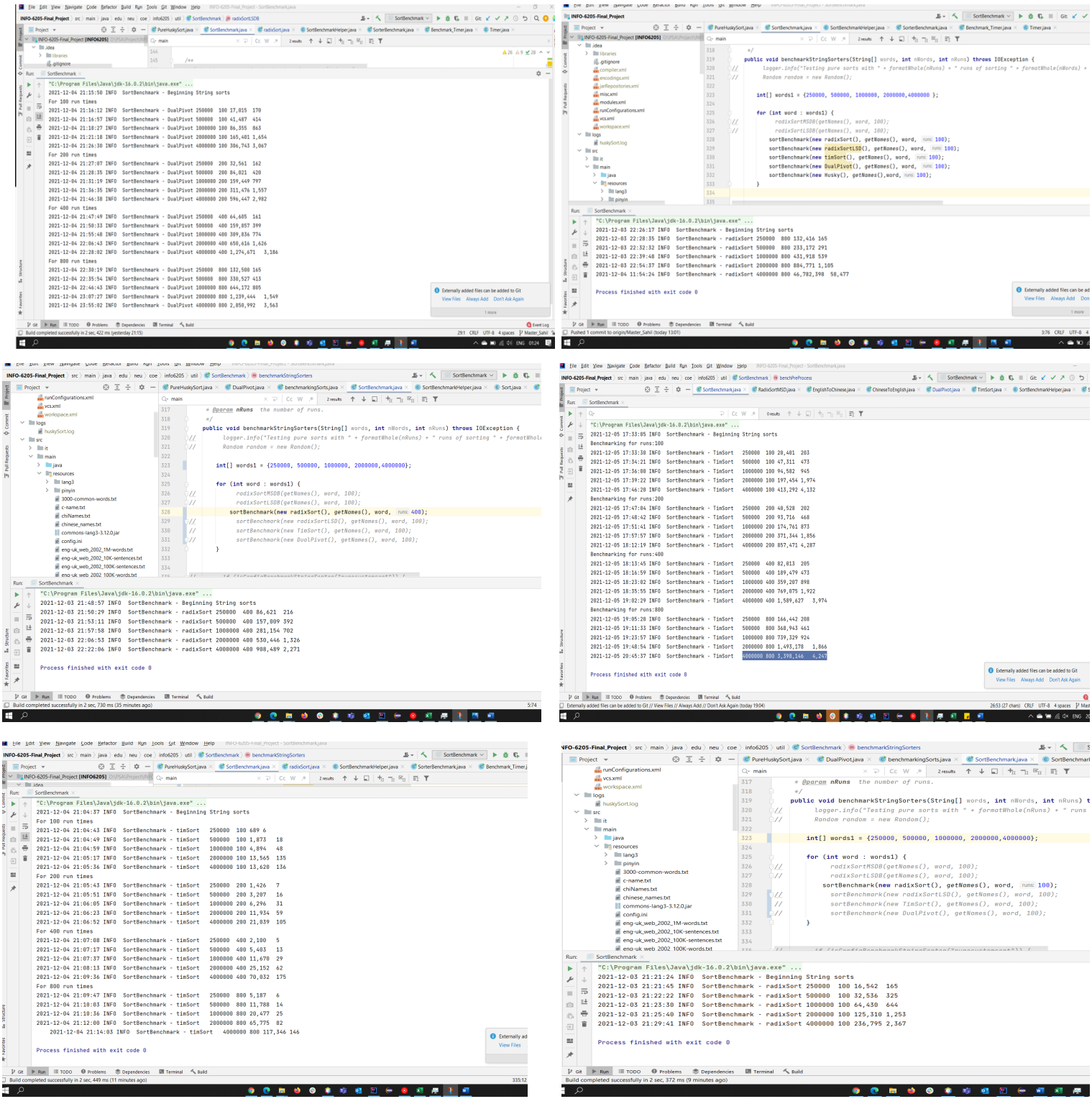

Fig 7: Benchmarking Result {SortBenchmart.java}

References:

a. https://algs4.cs.princeton.edu/23quicksort/QuickDualPivot.java.html
b. https://github.com/rchillyard/The-repository-formerly-known-as.git
c. https://github.com/rchillyard/INFO6205.git
d. https://github.com/artbez/TimSort/blob/master/src/main/java/timsort/TimSort.java
e. https://www.researchgate.net/publication/3593069_A_new_efficient_radix_sort
f. https://www.usenix.org/legacy/publications/compsystems/1993/win_mcilroy.pdf