Final project

Sorting Algorithms in Chinese Characters

Team Members:

001582498 Xingyu Huang

001528318 Zhen Luo

001048178 Haoyuan Qin

1. Project Description

Unlike English letters which are encoded by ASCII, Chinese characters normally use Unicode to encode. Hence, how to sort Chinese characters in order, became a real problem we may face in the real world. So, we borrowed the idea of ‘Pinyin’, which is a kind of phonetic symbol to pronounce Chinese characters, to sort Chinese characters in order.

1.1 Problems

However, one Chinese character may have different pronunciations and one pronunciation may correspond to different Chinese characters. In order to deal with these situations, we import a library called pinyin4 which can generate pinyin from imported Chinese characters String even though there are some English letters in the String.

1.2 Sort-Algorithms

We used MSD Radix-sort, LSD Radix-sort, Dual-pivot Quicksort, Timsort and Husky Sort to sort Chinese characters. These algorithms may have different performance in sorting Chinese characters. So our goal is not only to sort Chinese characters with

different algorithms but also to find the best algorithms in doing such things.

2. Project Design

This project is made up of different classes, methods and libraries. In order to sort Chinese characters, we build an Object called “MingZi” which includes “HanZi” (Store Chinese characters), “PinYin” (Store pinyin) and “longest” (Store the length of the longest String).

Before we start to sort our Strings, the TXT.java can help us to read strings from txt file and convert it to “MingZi”. So now we can sort our strings by sort the “PinYin” and then give out the result of sorting.

As we implement different sorting algorithms, the performance of each algorithm is quite different from each other and the correctness of each algorithm in this case is still a myth. So, we implement test cases and benchmarks to see these results and differences.

Furthermore, in order to increase the performance of tests and benchmark to prevent over time error. Multi-threads are used in our

project which can largely increase the performance and give us the best result.

3. Project Realization

3.1 Theory Estimation

Before we jump into the tests and benchmark. We'd like to analyze each sorting algorithm.

For timsort, the average time consumption is NlgN, and for quicksort it's 2NlgN and for Husky sort it is NlgN as well.(Hillyard) However, the quicksort's time consuming will be decreased if quicksort has more pivots and the best is quicksort with a median of three pivots.(Lakshmi)

Theoretically, as the number of elements increased, radix sort did not perform so well.(Shukla) MSD Radix performed well in broad uniform and normal distributions, in the case of MSD Radix, performing exceptionally well, but with catastrophic performance on narrower distributions. (S.Thiel)

So we should have a test to see whether every sorting algorithm are suitable for sorting Chinese names, so here comes our benchmark test and other tests.

In this case we may suppose that LSD and MSD might be the fastest,husky sort might be the second and then quick sort and Timsort.

3.2 Tests:

Before we do our test, we first generate random strings by using the Chinese name strings generation util tool, which will help us to generate new random Chinese name strings with different common Chinese characters.

In the test phase, we could use those util tools to help us produce large amounts of test data resources easily. But we will use the Chinese names file given by the professor as our data resource to make the conclusion in the end phase.

We have designed three different kinds of unit tests including Function Test, Num Test and Order Test. In order to guarantee every sort algorithm handles the same data resource in every test and obtain a good individual performance (not being influenced by other processes at the same time), we used the thread pool for every sort algorithm to handle the same size of data. It improves our unit tests performance to a fantastic extent.

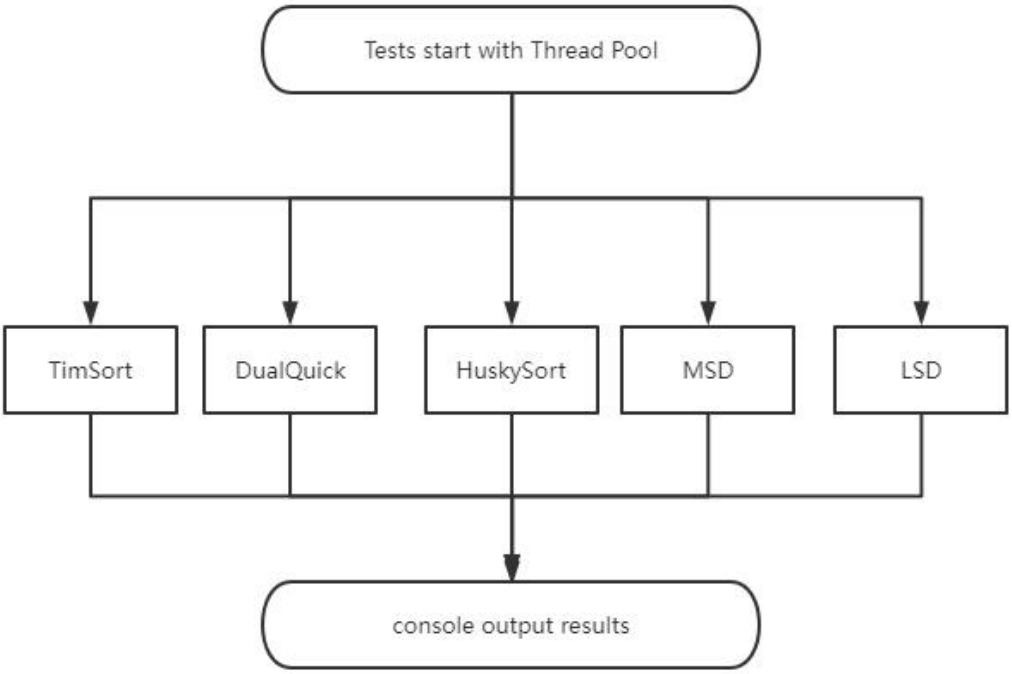

Figure 1: Tests structure

3.2.1 Function Test

Function Test tests the accuracy and stability of the sort algorithms, it guarantees our sort algorithms run in the right way. For every algorithm, we input 1M data to handle, and then we could compare the result of every sort to the right order list to check if this algorithm function is right. It’s a basic function test.

3.2.2 Num Test

Num Test tests the running time of all kinds of sort algorithms in different amounts of data(from 1M to 8M random data),every unit in Num Test presents the performance of different sort algorithms in the same amounts of data. We could use these results as our final observation to help us get the conclusion.

3.2.3 Order Test

Order Test tests the performance of all kinds of sort algorithms with the same amount of data(1M) in different orders including random, ordered and reversed data. We could use these results to help us find the advantages and disadvantages of different sort algorithms in the same order situation.

3.3 Benchmarks

We also did benchmarks to see different sorting algorithms' performance. In our benchmark experience, we tested different lengths of names(250k, 500k, 1M, 2M and 4M).

We use System.currentTimeMillis() to get the timestamp. We get a timestamp at the beginning, and get a timestamp after all work is

done. So that we can get the time duration by simply subtracting two timestamps.

We use our own Object to store the data, so we also count the time of transforming the string array to “MingZi” except Husky Sort, this is because Husky Sort does not need “PinYin” to sort Chinese characters. It can sort Chinese in a string array directly. Therefore, because Husky Sort is an NlogN algorithm we found it is faster than other sorts.

We use tests to run benchmarks and output results.

4. Results

4.1 Tests Results

We can see from the pictures below that our project is able to complete the job correctly and efficiently.

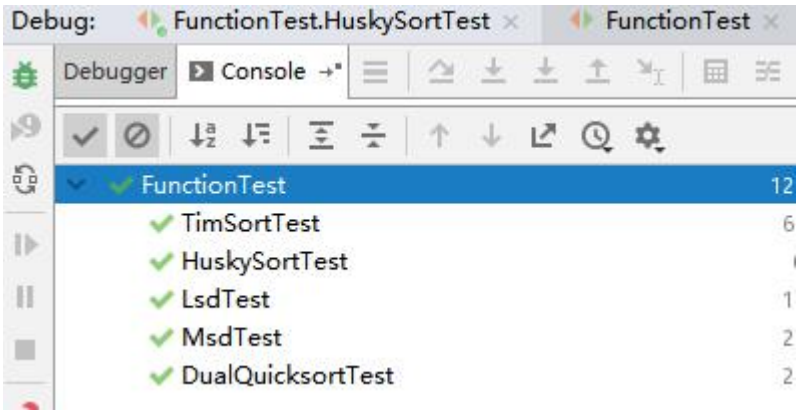

Figure 2: Function Test

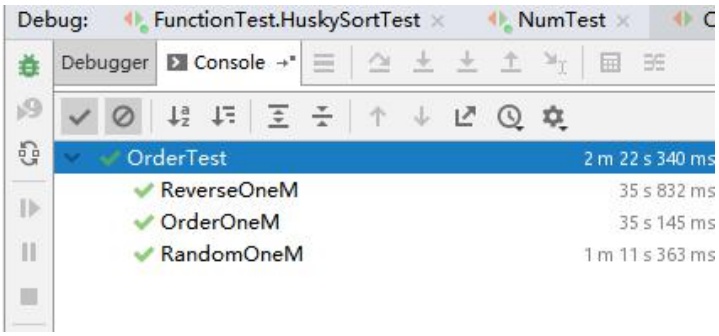

Figure 3: Order Test

Figure 4: Num Test

```
pre working, prepare data resource
preworking finished,

TimSort thread start
HuskySort thread start
LsdSort thread start
QuickSort thread start
MsdSort thread start

TimSort in random  milliseconds:15046
QuickSort in random  milliseconds:14906
LsdSort in random  milliseconds:12208
MsdSort in random  milliseconds:14320
HuskySort in random  milliseconds:16797

test end
```

Figure 5: Test Result

4.2 Benchmark Results

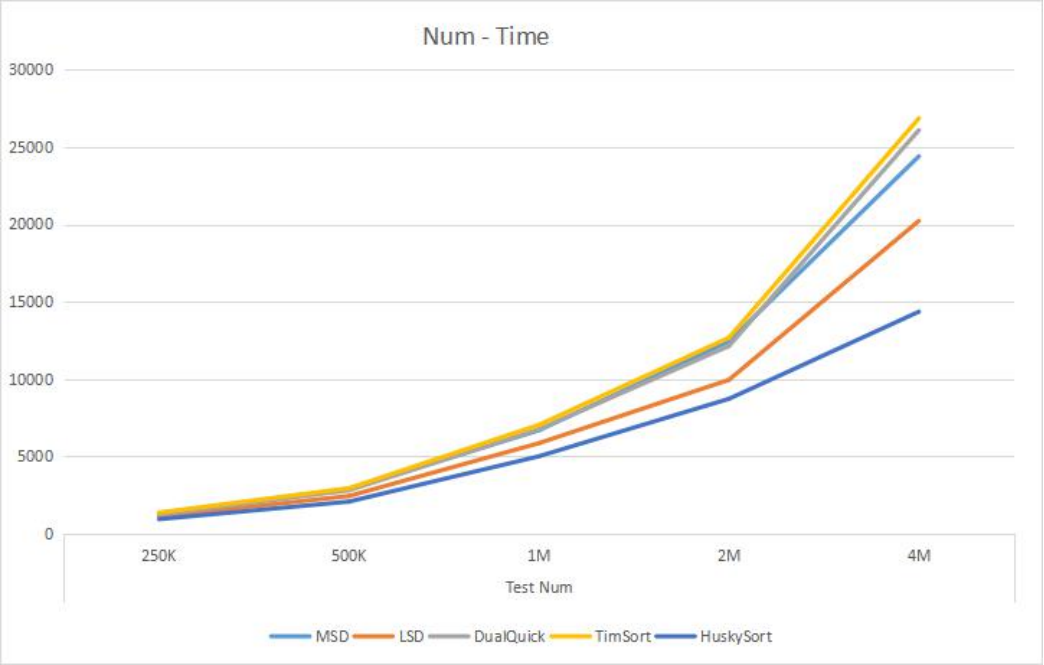

Figure 6: Benchmark Results

From the picture above, we can under stand that when sorting Chinese characters by Pinyin order, every algorithm can perfectly complete the job and Husky sort is the fastest algorithm, then LSD then MSD and then Dual-pivot Quicksort and TimSort. These margins increase when the number of data increases.

5. Conclusion

After we arrange every sorting algorithm, we can sort Chinese names and strings correctly and within an acceptable time consumption. With the help of object Mingzi, we can sort and display the result easily. And come up with a result that Huskysort is the quickest, LSD radix sort is the second and rest of three algorithms do not have big difference when sort Chinese name strings. The result we get from tests and benchmark is not quit different with the guess which comes from theory above.

6. References

Timsort. GeeksforGeeks. (2021, June 26). Retrieved December 5, 2021, from https://www.geeksforgeeks.org/timsort.

Tim Peters 2002, Timsort description, accessed 28 November 2021, http://svn.python.org/projects/python/ trunk/Objects/listsort.txt

MSD(most significant digit) radix sort. GeeksforGeeks. (2021, August 11). Retrieved December 5, 2021, from https://www.geeksforgeeks.org/msd-most-significant-digit-radix-sort/.

MSD(most significant digit) radix sort. GeeksforGeeks. (2021, August 11). Retrieved December 5, 2021, from https://www.geeksforgeeks.org/msd-most-significant-digit-radix-sort/.

Lakshmi, D. I. (2018). Performance analysis of four different types of sorting algorithms using different languages. *International Journal of Trend in Scientific Research and Development*, *Volume-2*(Issue-2), 535–541. https://doi.org/10.31142/ijtsrd9464

Wild, S., Schneider, K., Nebel, M., Sedgewick, R., & Dietzfelbinger, M. (2016). *Dual-pivot quicksort and beyond: Analysis of multiway partitioning and its practical potential*. Technischen Universität Kaiserslautern.

Shukla, A., Saxena, A.K., Baruah, P.K., Bandyopadhyay, S., Sahni, S., Jiménez-González, D., Navarro, J.J., Larrba-Pey, J., Cederman, D., & Tsigas, P. (2012). ‘ Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set in Distributed Computing Environment ’.

Relaxing the counting requirement for least significant digit radix sorts. PDF Free Download. (n.d.). Retrieved December 5, 2021, from https://docplayer.net/197070654-Relaxing-the-counting-requirement-for-least-significant-digit-radix-sorts.html.

Hillyard, Robin & Liaozheng, Yunlu & R, Sai. (2020). Huskysort.