The project is to analyze the efficiency of different sorting algorithms including MSD radix sort, Timsort, Dual-pivot Quicksort, Huskysort and LSD radix sort sorting on Unicode(Chinese). Chinese words are different from any other words. It is ordered by pinyin. As Chinese words can’t be sort simply by their Unicode order, we should use other method or library to sort Chinese.

At first, it turned out that there is a *Collator* class which can help us compare different Chinese words by setting *new Collator().getInstance(Locale.CHINA)* . However, when I set up an unit test for *Collator.compare()* , it turned out that the *Collator* class couldn’t sort Chinese words correctly. At first, I thought that maybe *Collator* class follows English names principle, which it treats people’s first name more significantly rather than last name. Then I tested comparison between single Chinese words. It turned that it still couldn’t compare Chinese words properly. So I searched for another method to sort Chinese words, which is*, pinyin4j.jar.* This library can help me transform Chinese words into pinyin, which the combination of the correct pronouncing order of Chinese words and the tones. These can be easily sorted as they are English strings and follows dictionary order.

I implemented all the sorting algorithms by using both *Collator* class and *pinyin4j* library. In sorting with Collator, I used a simple String array to store all the Chinese names. In sorting with pinyin4j, I used a two dimensional String array to store all the Chinese names. Each node contains original Chinese words and their pinyin expression.

MSD radix sort

By using *Collator* class, it’s difficult to define a proper *charAt()* method to return the char of a Chinese word. In order to solve this problem, I designed a map to keep each Chinese word and their order. At first, we obtained each Chinese word from shuffledChinese.txt and store themselves and their pinyin String transformed from *pinyin4j* in a key-value pair. Then we sort the pairs and initialize the map. After that, we can rewrite the *charAt()* method so that it will return *map.get(Chinese char)* which is the order of the Chinese word.

Text

Description automatically generated

Text

Description automatically generated

Then I just implemented the basic MSD radix sort and finished the algorithm.

For MSD radix sort with *pinyin4j*, we just use basic MSD radix sort and change the original array to two dimensional array.

Text

Description automatically generated

Three way string quicksort

In three way string quick sort, I just reuse the map obtained from MSD radix sort and modify the code a little. Modifications are made separately in sorting with *Collator* and sort with *pinyin4j*.

Text

Description automatically generated

LSD radix sort

The major problem of LSD radix sort to sort Chinese names is that the basic LSD radix sort can only sort strings of same length. I have to add something to solve this problem. Since LSD radix sort is to sort the least significant character first, I can add “ “(space) to the end of the names which are not as long as others. At the beginning of the algorithm, I iterate the string array to get the length of longest name, and then iterate again to add space to names which length aren’t equal to the longest until they are the same length.

After that, the problem will be easy both for Collator sort and pinyin4j sort. In Collator sort, I used the map gained from MSD radix sort, put key-value pair (space -> 0) in the map and use this to rewrite the *charAt()* method.

Text

Description automatically generated

In pinyin4j sort, I substitute all the string array to two dimensional string array.

Text

Description automatically generated

Timsort

In TImsort, I set the minimum merge number as 32, and use insertion sort to sort the Strings. As mentioned before, I use *Collator* class and *pinyin4j* library to sort the String array, and make modifications to each algorithms.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Dual-pivot Quicksort

In Collator sort, I can simply modify the less method by implementing *Collator.compare().*

Text

Description automatically generated

In pinyin4j sort, I substitute array to two dimensional array.

Text

Description automatically generated

Huskysort

In Huskysort, I set a Node class which implements HuskySortable and override *compareTo*() and *huskycode()* method. In order to get the correct order, I set Node with str as a data member which obtained from *pinyin4j* transforming Chinese names. And in *compareTo()* method, I compare attribute str of different Nodes. About *huskycode()*, I use *HuskyCoderFactory.asciiToLong()* method to get long of the Node’s str.

Text

Description automatically generated

While initializing, I create Node[] array and init *QuickHuskySort sorter*, then we use sorter to sort.

Text

Description automatically generated

Finally, I use *sorter.sort()* to finish the algorithm.