GROUP 4

Shrusti Rajesh Chheda (002196756)
Kaeyang Hsieh (001092745)
Nienchi Hung (002980433)

INFO 6205 Program Structures & Algorithms
Fall 2021
Final Project

Goals

The goal is to sort Chinese in the alphabetical order of the toneless Pinyin. The sorting method should generate a unique and identical result irrespective of the order of the input list and it should be applicable to all the sorting algorithms (MSD Radix Sort, LSD Radix Sort, Dual-pivot Quicksort, Timsort, and Husky Sort).

Sorting Mechanism

1. Convert Chinese characters to its corresponding toneless Pinyin (Preprocess)

First, we use the open-source java library ICU4j (International Component for Unicode for Java) collator to convert chinese characters to their corresponding toneless Pinyin. In general expectation, the order of tones is from the first to the fifth tone. However, the alphabets representing toned vowels are in different orders in Unicodes. For example, the vowel "a" in Pinyin with tones are "ā á ǎ à a," but their order in Unicodes is "a à á ā ǎ," and à comes after a-z, which means that “边黎明=biān lí míng” will be prior to “阿雁=ā yàn.” To generate the result in alphabetical order, we decided to use toneless Pinyin. e.g. “阿雁” will be converted to “a yan” instead of “ā yàn”, and “边黎明” will be converted to “bian li ming” instead of “iān lí míng.”

2. Build a mapping of Pinyin to Chinese names (Preprocess)

What’s interesting about Chinese is that it's pretty common to have multiple words with exactly the same pronunciation. Using toneless Pinyin generates even more cases like this. For example, the names "刘佳林, 刘佳琳, 刘佳霖, 刘加林, 刘嘉林, 刘嘉琳, 刘嘉霖, 刘嘉麟, 刘家林, 刘家琳, 刘家霖, 刘家麟, 刘甲林, 柳佳林, 柳家林" all have the same toneless Pinyin "liu jia lin". To address this issue, we use Map<String, List<String>> to store the mapping of Pinyin to Chinese names, where the key is the Pinyin and the value is a List of Chinese names.

3. Sort the names with their Pinyin and number of strokes of radicals

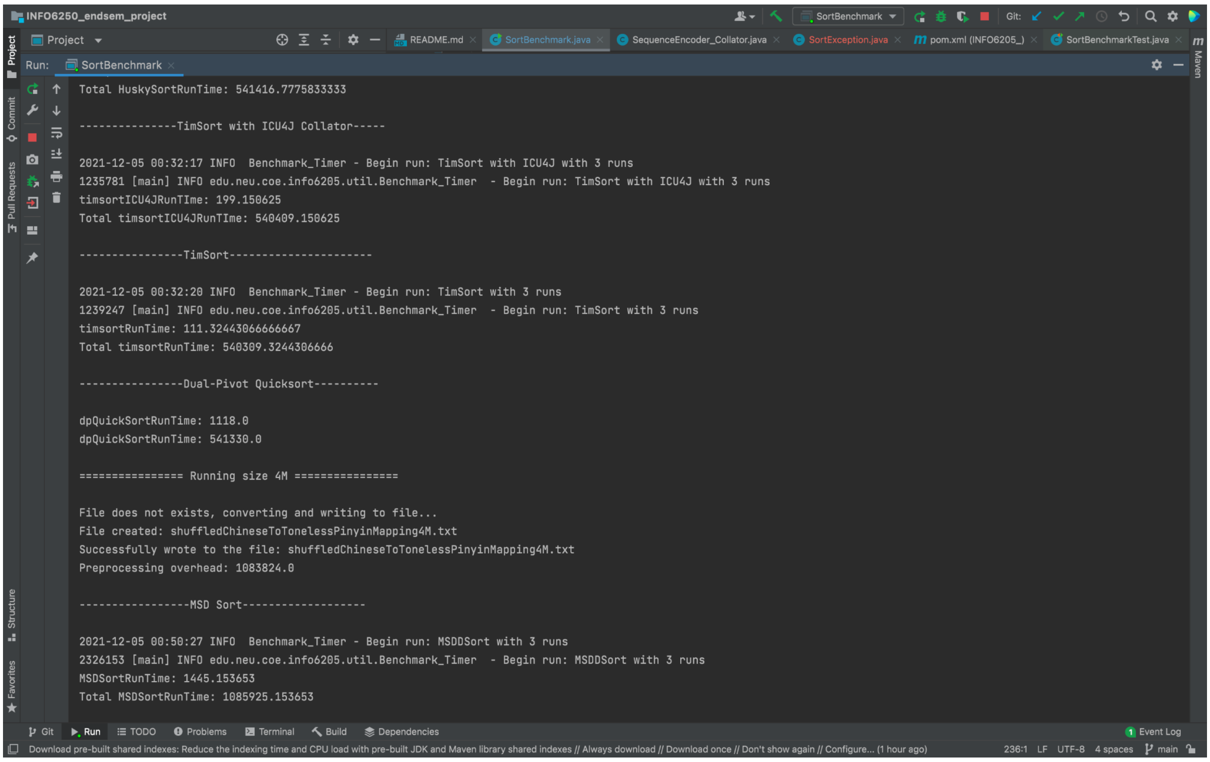
After constructing the mapping of Pinyin to Chinese names and storing all the Pinyin as an Array, we can now use MSD Radix Sort, LSD Radix Sort, Timsort, Quick Sort, and Husky Sort to sort the Array of Pinyin. This step is straight-forward since we are basically sorting a List of English words. For those names with the same toneless Pinyin, we have sorted them directly with system sort without converting to Pinyin. The order of Chinese characters in Unicodes is by the strokes of radicals. For example, "刘佳林 and 刘佳霖" have the same Pinyin but they have different radicals in the third character. The character "林" has the radical "木", and "霖" has the radical "雨". It's obvious that the latter one (雨) has more strokes than the former one (木), therefore, the order of these two names is " 刘佳林 刘佳霖."

According to CJK Unified Ideographs, The basic block named *CJK Unified Ideographs* (4E00–9FFF) contains 20,992 basic Chinese characters in the range U+4E00 through U+9FFF. And other uncommon characters are added later as extensions. Therefore, it's not guaranteed that the characters will always be in the "correct order". However, as we have already sorted the names in alphabetical order, the order won't be completely meaningless.

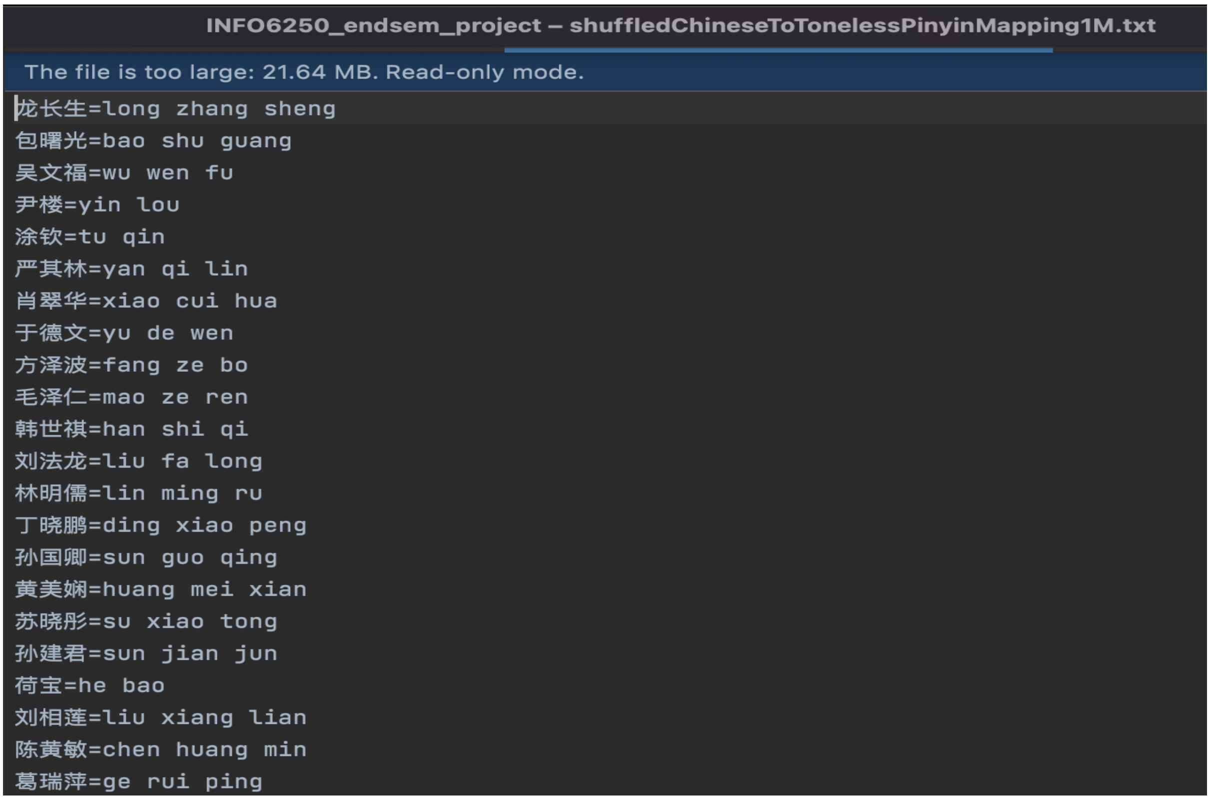
4. Convert Pinyin back to Chinese (Postprocess)

The last step is to convert the Pinyin back to Chinese using the map we built previously.

Code Snippets



Benchmarking output



Chinese to pinyin mapping file

Benchmarking

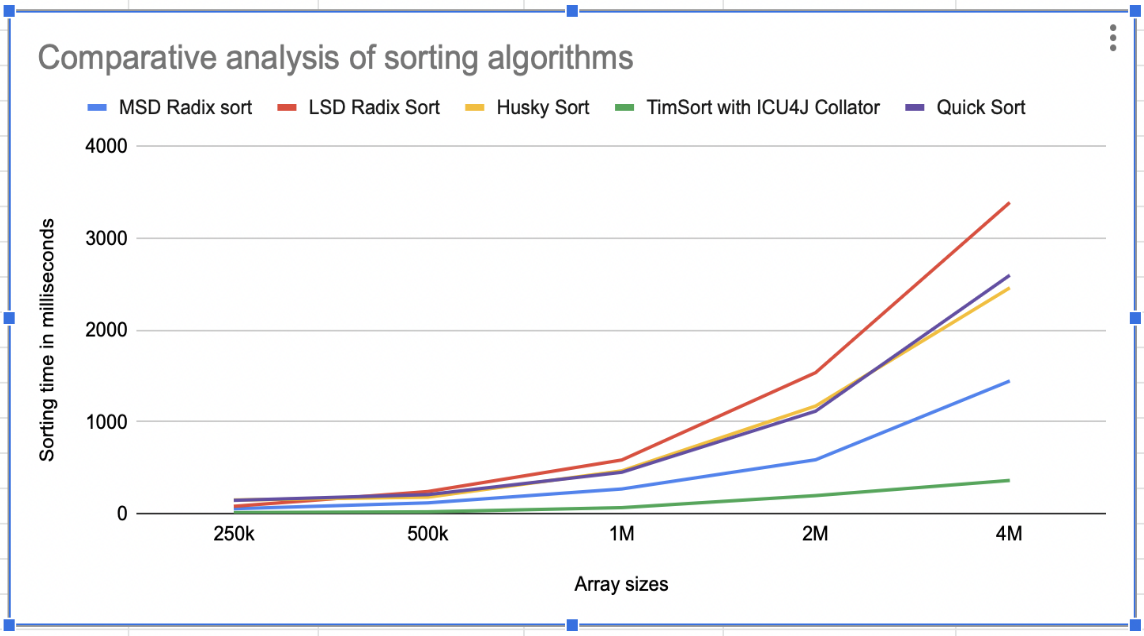
The runtime of the various sorting algorithms is a combination of time required for sorting, pre-processing the input array and some post processes. The table below lists the total runtime of the various sorting algorithms.

Array sizes	MSD Radix sort	Custom MSD Sort - CE1	Custom MSD Sort - CE2	LSD Radix Sort	Husky Sort	TimSort with ICU4J Collator	Tim Sort	Quicksort
250k	69235.05908	69485.0371	69466.87233	69175.97292	69243.43493	69106.40135	69097.01774	69241
500k	132472.2436	1263.527569	133640.0937	132595.5491	132522.3722	132357.8411	132353.0044	132549
1M	268028.421	272265.0727	272193.2222	268295.7637	268178.4829	267777.9818	267747.8359	268164
2M	540892.0959	559950.6165	559959.2756	541739.382	541416.7776	540409.1506	540309.3244	541330
4M	1085925.154	1172184.582	1178244.14	1087746.959	1086728.027	1084636.945	1084512.412	1086867

Total running time of sorting algorithms including preprocessing

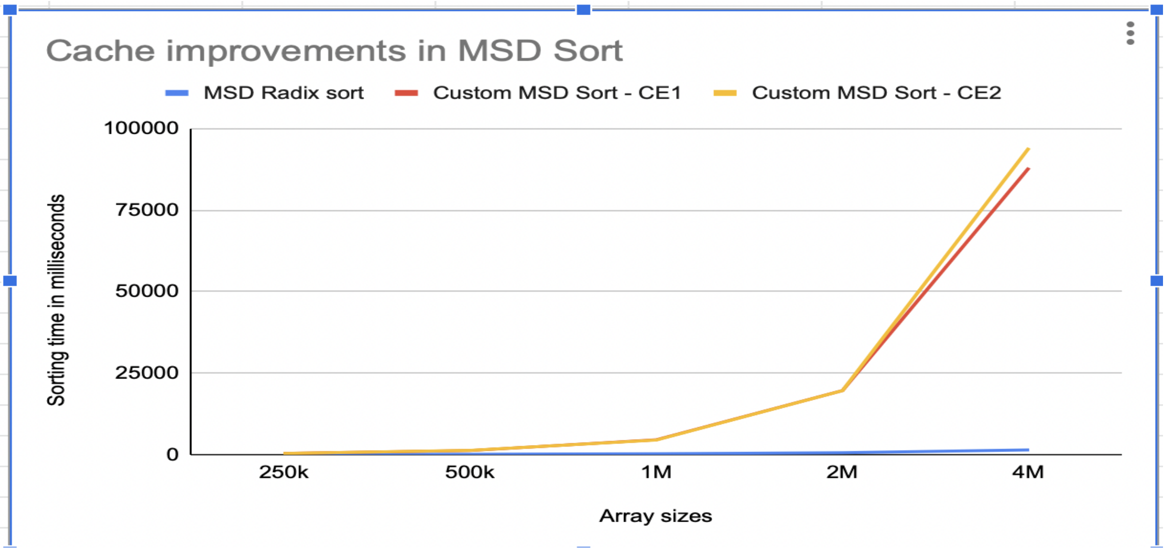
As the preprocessing time is much greater than the sorting time it masks the difference between the sort time of the various algorithms. Thus we have decided to consider only the sorting time for benchmarking. All the sorts have the same preprocessed input pinyin array and same post processes. Thus removing this constant time does not affect the benchmarking. The below table depicts the runtime of the various sorting algorithms.

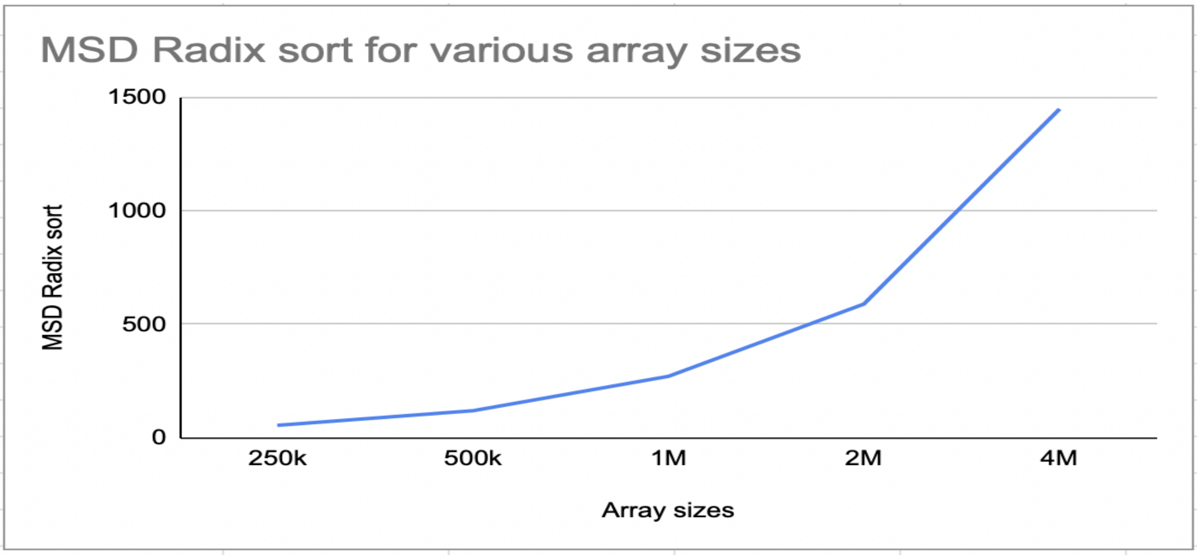
Array sizes	MSD Radix sort	LSD Radix Sort	Husky Sort	TimSort with ICU4J Collator	Quick Sort
250k	56.059083	82.972917	151.4349303	14.40134733	148
500k	120.243556	243.5490833	182.3721947	22.84113867	210
1M	271.421	586.7637497	468.4829167	67.98175	453
2M	589.0958613	1537.381986	1173.777583	199.150625	1118
4M	1445.153653	3385.959264	2458.026583	363.945389	2595



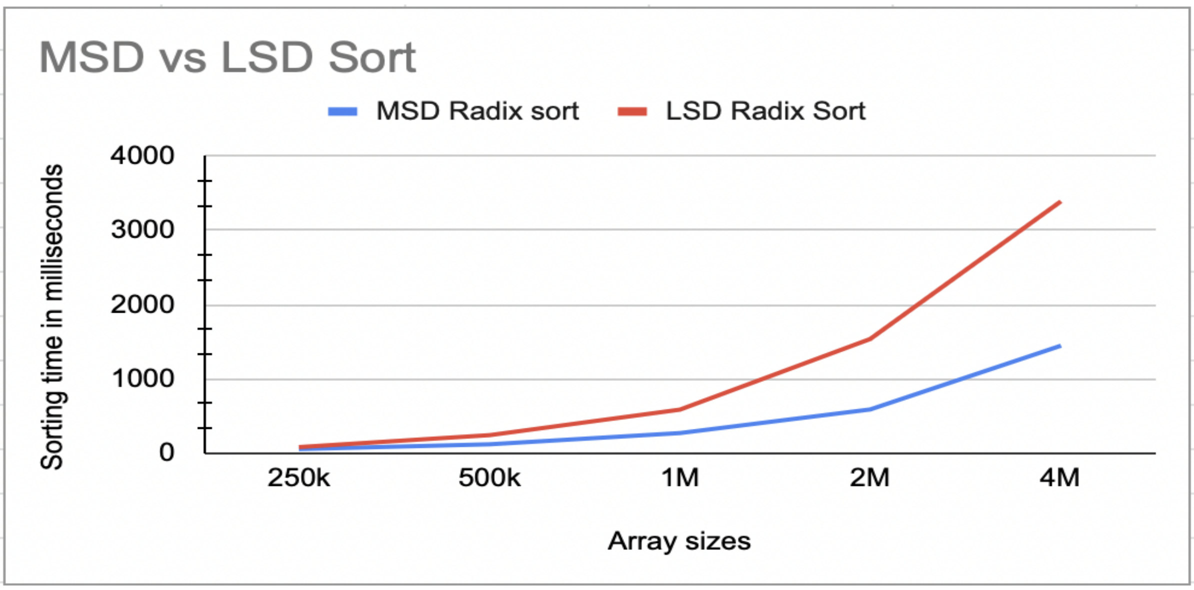
Sorting time of various sorting algorithms

We implemented some cache improvements to MSD sort but didn't observe the expected results. Details are discussed later in the report.

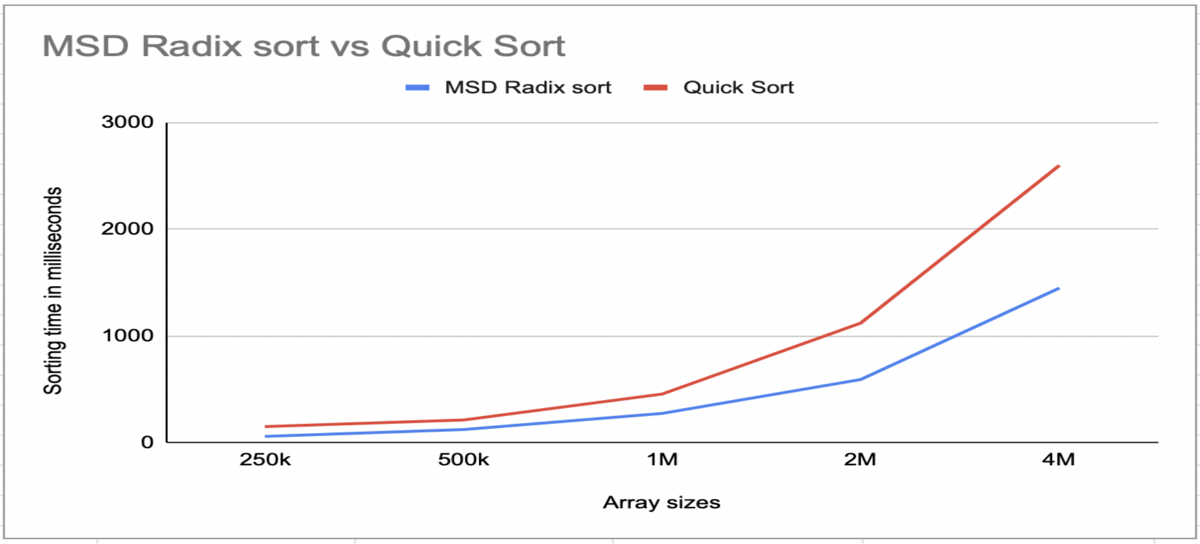




Sorting time of MSD Radix Sort on a pinyin array of various sizes



Comparative analysis of MSD and LSD Radix Sort



Comparative analysis of MSD and Dual-pivot Quicksort

The benchmarking tasks were performed in the below system environment.

System Environment

Discussions

We implemented the techniques[1] to increase cache efficiency; however, there is no improvement but deterioration(figure X, Y). We have added the discussions in the literature review paper.

We also used the MySQL feature to sort the shuffledChinese.txt as a reference.
The SQL command we used is "select name from names ORDER BY CONVERT(name USING gbk);" where the "gbk" is another character set using Pinyin order. It took 10.2 seconds to complete the job. However, we cannot compare the result with ours because the character set is different from ours, and we do not know how to apply the same character set to MySQL.

We met a stack overflow issue when benchmarking the dual-pivot quicksort algorithm. The root cause is that the default stack, 1M, is too small for a one million-sized array, as the quicksort generates a lot of recursive calls. After some experiments adjusting different stack sizes(table 1), the stack size should be increased to 48M to make the quicksort complete the job.

array size / stack size	3m	6m	12m	24m	48m
15626	2018 ms				
31250	5014 ms				
62500	29222 ms				
125000	stackoverflow	117522 ms			
250000		stackoverflow	508439 ms		
499999				1473944 ms	
999998					> 8 hours

Table 1

Conclusions

- MSD Radix Sort, sorts using the most significant bits first which is advantageous in case of strings as they are naturally compared left to right. Also MSD, by nature sorts strings without iterating through all of the digits in the strings and thus gives better results than LSD.

- According to our experiment, Timsort has the best performance among all the sorting algorithms, especially for the large arrays. As for MSD sort, because we convert Chinese names to Pinyin, the Strings we are actually sorting will be much longer than the original ones, which means that the MSD sort would have to recursively sort for each character many times (around 10 ~ 15 for each name). Also, the inner loop of MSD sort has lots of instructions, and it uses extra memory for the counting purpose, making it less ideal for large datasets if compared to Timsort.

- The benchmark results showed that the MSD Radix Sort is faster than Huskysort(PureHuskySort)[4], matching our expectations. The average-case time complexity of the MSD Radix Sort is O(N * D), where N is the number of keys, and D is the number of digits to be compared. The time complexity of the HuskySort is $O(6.4 N \ln N + 9N)$. In this case, D is smaller than $6.4 N \ln N$. So the Radix sort is faster than the Huskysort. If the digit of the keys, D, is larger than $6.4 N \ln N$, then the Radix sort will be slower than the Huskysort.

- From the graphs we observe MSD has better runtime for arrays with 4M chinese names. For a larger input that includes strings, MSD Radix sort would give better results than Dual Pivot Quicksort. Randomly ordered string arrays involve more rescanning of keys in case of quicksort whereas MSD may not examine all the keys. Thus we believe MSD Sort performs much better than Dual Pivot Quicksort for larger arrays.

Future Improvement

1. Sort with Tones

Since Chinese has five tones, if we ignore the tones when sorting, we won't be able to distinguish between many characters. In the chart below, there are 12 different names with four different toneless Pinyin. The first column is the result of our sorting method (Sort with toneless Pinyin first; for those with the identical toneless Pinyin, sort them with Chinese characters' Unicodes).

Chinese Names	Toneless Pinyin	Pinyin with Tones	Tone of First Character
付佳佳	fu jia jia	fù jiā jiā	4
伏佳佳	fu jia jia	fú jiā jiā	2
符佳佳	fu jia jia	fú jiā jiā	2
付佳杰	fu jia jie	fù jiā jié	4
傅佳杰	fu jia jie	fù jiā jié	4
傅家杰	fu jia jie	fù jiā jié	4
付家军	fu jia jun	fù jiā jùn	4
傅佳俊	fu jia jun	fù jiā jùn	4
傅家骏	fu jia jun	fù jiā jùn	4
付嘉丽	fu jia li	fù jiā jūn	4
傅嘉莉	fu jia li	fù jiā lì	4
符佳丽	fu jia li	fú jiā lì	2

The result of sorting with toneless Pinyin

We can see that the names with the same surname (first character of the name) are not arranged together and they are not sorted from tone 1 - 5. To resolve this, we can implement our own Comparator, and define the correct order of toned vowels.

2. Treat each character of the names as a key instead of treating the whole name as a String

Even if we sort with tones (not yet implemented our own Comparator so the order of tones is still wrong), the names with the same surname are still not ordered together. That is because we are treating the names as non-splittable String. Instead, we should treat each character of the names as a dependent key. To be more precise, we first sort the names with the first character, and then with the second character, and then with the third, so on and so forth.

Chinese Names	Toneless Pinyin	Pinyin with Tones	Tone of First Character
付佳杰	fu jia jie	fù jiā jié	4
傅佳杰	fu jia jie	fù jiā jié	4
傅家杰	fu jia jie	fù jiā jié	4
付佳佳	fu jia jia	fù jiā jiā	4
傅佳俊	fu jia jun	fù jiā jùn	4
傅家骏	fu jia jun	fù jiā jùn	4
付家军	fu jia jun	fù jiā jùn	4
付嘉丽	fu jia li	fù jiā jūn	4
傅嘉莉	fu jia li	fù jiā lì	4
伏佳佳	fu jia jia	fú jiā jiā	2
符佳佳	fu jia jia	fú jiā jiā	2
符佳丽	fu jia li	fú jiā lì	2

The result of sorting with toned Pinyin

3. Use Custom Encoding instead of Unicodes

MSD Radix sort and LSD Radix sort are non-comparative sorts. They use counting sort as the subroutine and utilize ASCII code or Unicode to determine the order, which means that even if we implement our own Comparator to define the order, it still cannot be used in the Radix sort. To address this issue, we can construct our own encoding table (as shown below). This way, we can determine the custom order of the Pinyin characters and the encoding table can be used both in Comparator and Radix sort.

```
HashMap<String, String> customEncodingMapping = new HashMap<String, String>(){{
    put("ā", "0"); put("á", "1"); put("ǎ", "2"); put("à", "3"); put("a", "4");
    put("b", "5"); put("c", "6"); put("d", "7"); put("ē", "8"); put("é", "9");
    put("ě", "a"); put("è", "b"); put("e", "c"); put("f", "d"); put("g", "e");
    put("h", "f"); put("ī", "10"); put("í", "11"); put("ǐ", "12"); put("ì", "13");
    put("i", "14"); put("j", "15"); put("k", "16"); put("l", "17"); put("m", "18");
    put("n", "19"); put("ō", "1a"); put("ó", "1b"); put("ǒ", "1c"); put("ò", "1d");
    put("o", "1e"); put("p", "1f"); put("q", "20"); put("r", "21"); put("s", "22");
    put("t", "23"); put("ū", "24"); put("ú", "25"); put("ǔ", "26"); put("ù", "27");
    put("u", "28"); put("v", "29"); put("w", "2a"); put("x", "2b"); put("y", "2c");
    put("z", "2d");
}};
```

The encoding table of all the characters used in toned Pinyin, the characters with smaller value should be prior to those with larger value

4. Adopt parallel methods like PARADIS[2] or Region sort[3]

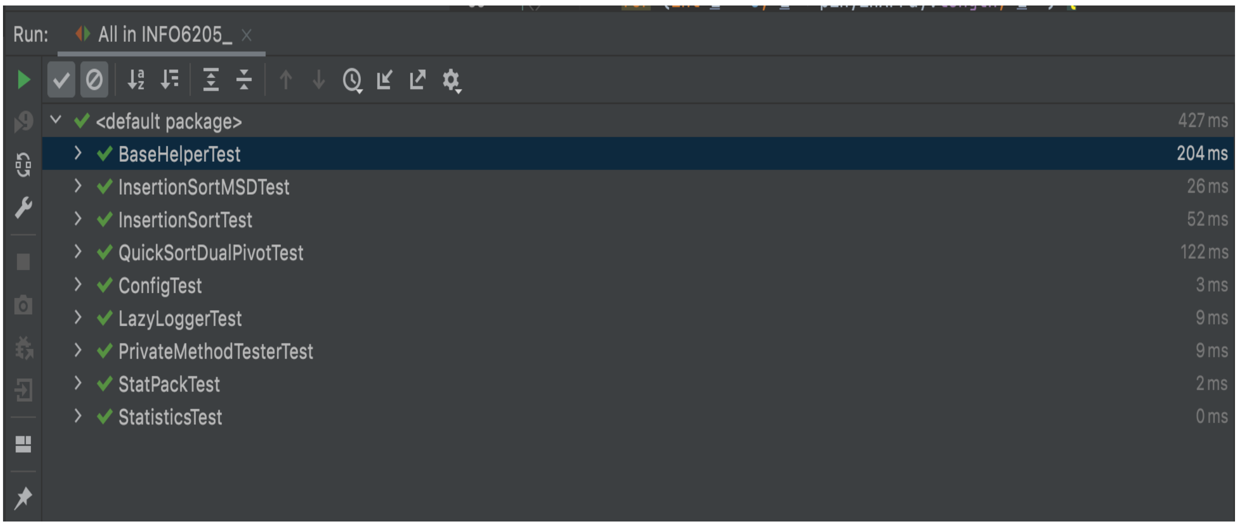
In the literature review, we selected two parallel sort papers which may be helpful for us to improve the MSD Radix sort performance. However, the parallel technique is too complicated to implement and due to time constraints it is out of scope for the project.

5. Improve the preprocessing time

The chinese names are converted to pinyin and this array is given as an input to the various sorting algorithms. These conversions are a part of preprocessing and are a significant overhead in terms of time taken. We can add methods to reduce this preprocessing time to make the overall process of sorting more efficient.

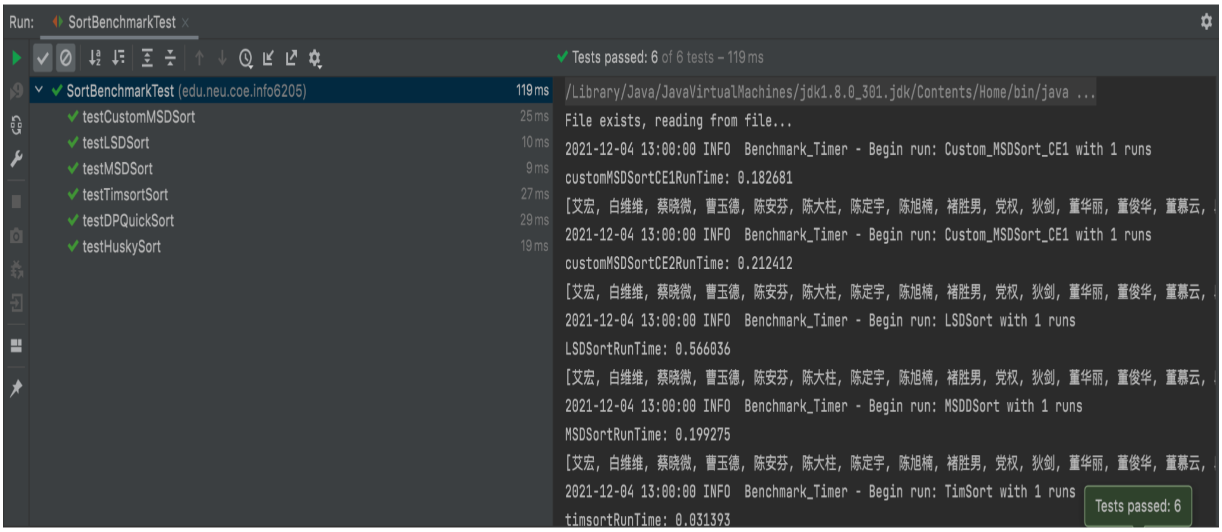
Unit Tests Result

We integrated the codes of TimSort.java, QuickSort_DualPivot.java, LSDStringSort.java, MSDStringSort.java, InsertionSortMSD.java, Benchmark_Timer.java, Timer.java, as well as their related Interfaces, parent classes, Helper classes, and utility functions from the INFO6205 repository. The unit tests of the above classes ran successfully as shown below.



The unit tests result of the classes integrated from the INFO6205 repository

We wrote our own unit test SortBenchmarkTest.java to test the logic in SortBenchmark.java. It tests whether each method in SortBenchmark can sort the list of Chinese names in the correct order. The unit test result is as shown below.



The unit test result of SortBenchmark.java

References

[1] Kärkkäinen, J., & Rantala, T. (2008, November). Engineering radix sort for strings. In International Symposium on String Processing and Information Retrieval (pp. 3-14). Springer, Berlin, Heidelberg.

[2] Cho, M., Brand, D., Bordawekar, R., Finkler, U., Kulandaisamy, V., & Puri, R. (2015). PARADIS: an efficient parallel algorithm for in-place radix sort. Proceedings of the VLDB Endowment, 8(12), 1518-1529. Chicago

[3] Obeya, O., Kahssay, E., Fan, E., & Shun, J. (2019, June). Theoretically-efficient and practical parallel in-place radix sorting. In The 31st ACM Symposium on Parallelism in Algorithms and Architectures (pp. 213-224).

[4] Hillyard, R. C., & Liaozheng, Y. (2020). Huskysort. arXiv preprint arXiv:2012.00866.