INFO6205 Project – Experiment Report

Zhilue Wang – 001522973; Shashwat Shrey – 002128122

Abstract

In this project, our task is to implement different sorting algorithms for sorting Chinese names in pinyin order. We reviewed and implemented several existing sorting algorithms, such as QuickSort, TimSort, HuskySort, RadixSort and its improvement and we found that RadixSort has much better performance compared to comparison-based algorithms such as QuickSort.

Task

In the experiment we will be testing several sorting methods including TimSort, QuickSort, Dual-Pivot QuickSort, HuskySort, LSD RadixSort and MSD RadixSort. These algorithms are introduced in our literature review so we will not elaborate on how they work in this report. We will mainly focus on the experiment itself and their code implementation. For implementation, we used standard one for most algorithms, but we also made modifications to some of them.

The input to sorting methods are Chinese names. Because we want to sort it in pinyin order, we will use Java's Collator class to perform comparison for comparison-based sorting and to generate comparable bytes array for non-comparison sorting.

Sorting methods and setup

For TimSort, the size of each run is set to 32, and we use insertion sort on each run.

For QuickSort and Dual-Pivot QuickSort, no cutoff methods are used.

For HuskySort, the code is copied from the HuskySort public repo. The husky code is generated by CollatorKey class's toByteArray method and take the first 8 bytes as the husky code. We used IntroSort version as first pass and system sort as second pass when fixing remaining inversions.

LSD radix sort is implemented in standard way. We will distribute each byte in the bytes array generated by CollatorKey class into different buckets. As such the number of buckets is the total number of byte values which is 256.

MSD radix sort is implemented like LSD radix sort. To test how different number of buckets can have impact on the performance, we also have a 16 bits version which will combine two bytes into one int and distribute that into buckets. Thus, the total number of buckets will be 2^{16}.

The better caching version of MSD radix sort is also implemented and tested.

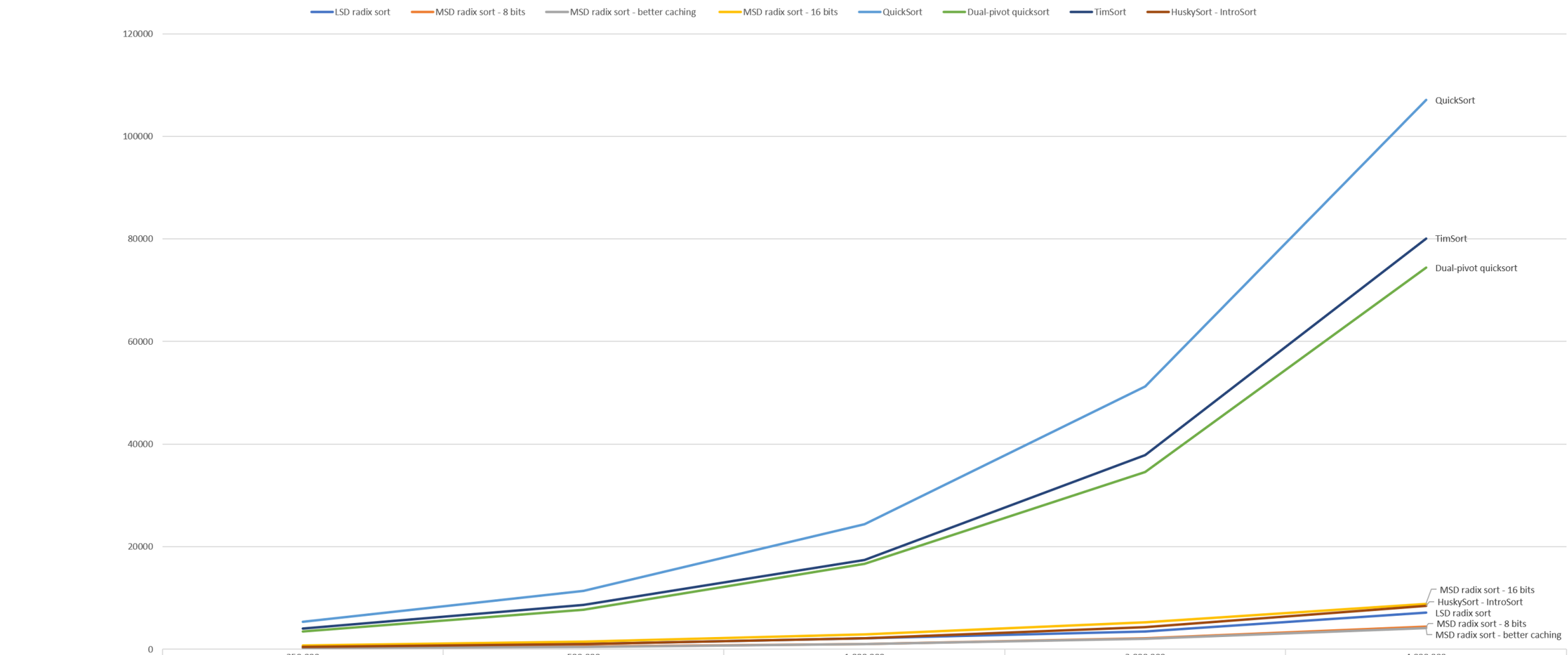
For each algorithm, we will test sorting on different Chinese names dataset with size 250K, 500K, 1M, 2M and 4M. We will perform 50 runs for each algorithm on each dataset.

Experiment environments

The experiments were carried out on a Windows machine with an Intel Core i9 processor model i9-10900K running at 3.70 GHz. The sizes of the processor's L1 and L2 caches are 64 kilobytes and 256 kilobytes, respectively. The machine was equipped with 32 gigabytes of RAM.

Results

The experiments results are shown below (in milliseconds):



	250,000	500,000	1,000,000	2,000,000	4,000,000
LSD radix sort	467.44	999.74	2190.32	3491.95	7139.2
MSD radix sort - 8 bits	237.61	513.67	1054.99	2146.05	4416.4
MSD radix sort - better caching	235.25	498.28	999.58	2091.1	4170.75
MSD radix sort - 16 bits	710.87	1496.87	2921.49	5220.11	8872.34
QuickSort	5393.2	11391.26	24355.56	51209.9	107071.64
Dual-pivot quicksort	3454.14	7732.48	16653.82	34530.65	74390.64
TimSort	4080.81	8620.38	17453.86	37871.64	80107.84
HuskySort - IntroSort	438.54	1019.87	2175.99	4293.57	8449.91

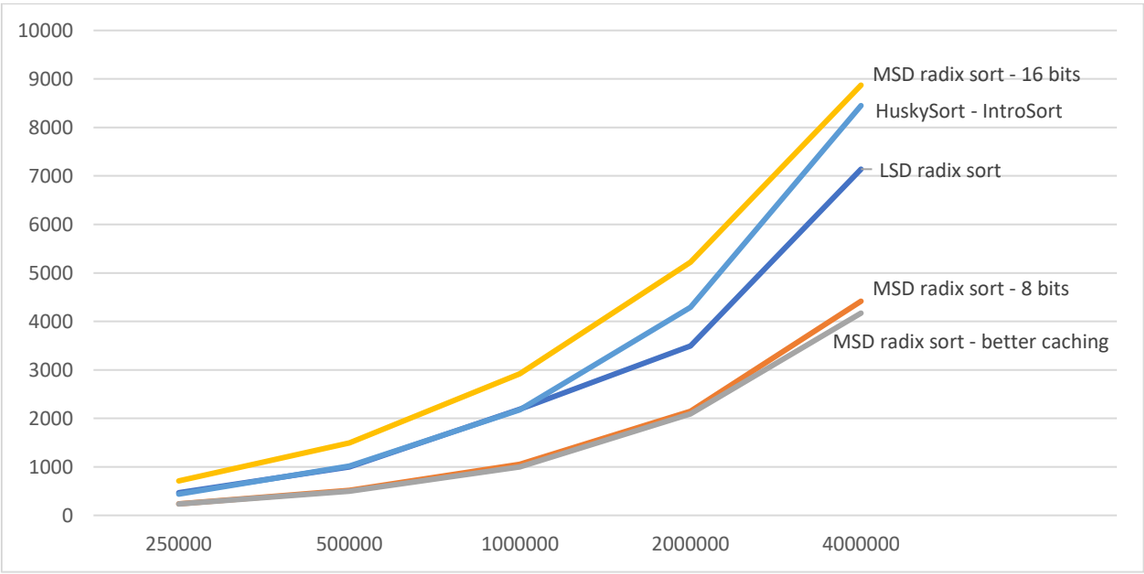
From the experiment result, we can see that RadixSort family has much better time performance than comparison-based sorting except HuskySort.

One thing to notice about HuskySort and Chinese strings is that even a one-character Chinese string will be converted into a 10 elements bytes array by Java Collator. This disabled perfect encoding in HuskySort and always requires a second pass to fix remaining inversions. However, for most short Chinese strings such as names, the first 8 bytes is enough to make it distinguishable from other strings. So even the husky encoding is imperfect, it is very close to be perfect and its performance is not impacted that much in this dataset.

HuskySort showed a close result compared to RadixSort. Because we expect the husky code to be close to perfect and the p_{crit} is very small, the number of array accesses are significantly reduced in the second pass of HuskySort and thus the time performance is greatly improved.

The time complexity of RadixSort is O(d * (n + b)) where d is the depth of bytes array and b is the number of buckets. When b is small, the main factor of time performance is the depth d. For a short Chinese string such as name, the usual size of bytes array converted from string is 14 to 18, which is smaller than log(n) when n is larger than 100000. Besides, we usually don't have to iterate through all 14 bytes to distinguish two strings. As we can expect, RadixSort will have even better improvement compared to comparison-based sorting when the number of elements N get larger.

Here is a closer look at RadixSort family's performance:



Comparing MSD radix sort 16 bits version with 8 bits version, we find that 16 bits version is almost twice slower than 8 bits version. One possible factor could be that, to get 16 bits array, we must combine two bytes which requires extra works. Another possibility is just that 16 bits are just too much, and it will slow down the algorithm. We will need more experiment to test which one is correct.

LSD radix sort is slower than MSD radix sort, as expected. What is exciting is that RadixSort with extra array to store each string's byte at that depth will indeed improve time performance, due to fewer cache misses. It has around 8% improvement compared to standard 8 bits MSD RadixSort on 4M dataset.

Conclusion

In summary, we tested several sorting algorithms' performance on Chinese names dataset. The results shows that RadixSort algorithms have much better running time than comparison-based algorithms except HuskySort, and HuskySort had the best performance among comparison-based algorithms. We can further improve the MSD RadixSort by adding a new array to reduce the cache misses, which aligns the idea behind HuskySort – avoid slow array accesses. For this particular use case - sorting Chinese names, I would recommend using MSD RadixSort as the solution.