INFO 6205 Fall 2021 Team Project

YANG SONG, College of Engineering, Northeastern University, USA

JING DAI, College of Engineering, Northeastern University, USA

Abstraction

The task of the project is to implement MSD radix sort for a natural language that is (Simplified) Chinese here with Unicode. We present the method using Collator and use the java library icu4j as the sorting rules and compare all the implements of Timsort, Dual-pivot Quicksort, Huskysort(pure Huskysort), and LSD radix sort with the same sorting rules of Chinese characters with Pinyin order. For the 5 algorithms we test to sort Chinese, Huskysort has the best performance and MSD Radix sort does not work well when sorting Chinese with our method.

This paper contains two parts: the project we do and the description of two related technical papers about MSD Radix sorting algorithms.

1 INTRODUCTION

Sorting is a fundamental problem in Algorithm, it's important even in the whole Computer science range We use different kinds of algorithms in both run-time and space complexity to improve performance when solving problems. There are many different dimensions to classify and distinguish a sorting algorithm such as comparison-based or non-comparison-based, in-order or not in-order sorts. A comparison-based algorithm sorts by comparing two elements at a time, and a non-comparison-based sort works by distributing the elements into buckets of their values.

Comparison-based sorts, such as bubble sort or selection sort take O($n2$) time, and researchers have made multiple improvements on comparison-based sorts with O($n\log n$) complexity like merge sort, quicksort [1] and heapsort [2] in average case. Theoretically, radix sorts can get an O(n) time as the best sorting algorithm when sorting string, but it still takes O($n\log n$) time in average case and the reality.

We put the survey of two technical papers in section 2. And in this project, we list the different performance of 5 algorithms and use them to sort Simplified Chinese characters. We review the algorithms and methods we used in section 3 and 4, describe the test result in section 5, and the final analysis and outcome conclusion in section 6.

2 SURVEY

2.1 Engineering Radix Sort for Strings

In this paper [7], author mainly discuss the MSD Radix algorithm.

This algorithm is divided into 3 parts. 1) Insertion sort for small buckets. 2) Iterating through all strings. 3) Iterating through all buckets. Therefore it has a O((σ/t + t)D) complexity, and the second part is usually close to the real behavior.

Then the paper concludes that there are two kinds of variants that can affect this algorithm: C for counting: Perform the distribution first time only counting the bucket sizes without actually moving the strings. D for dynamic buckets: Implement the buckets using some dynamically expanding data structure. Them they demonstrated the details of these two variants. For C-Variant, it introduces a new improvement based on the observation that both for loops do exactly the same sequence of slow character accesses that access each string for the first time in the first loop, copy the character into a separate array then the new loop generates much fewer cache. For D-Variant, the author implements different data structure to fulfill this and design a custom data structure which performs better.

With those implements, the paper concludes that distributing the strings into buckets requires an access to one character in each string. Two techniques, algorithmic caching and super alphabet, reduce the number of these slow accesses that improve the performance.

2.2 Formulation and analysis of in-place MSD radix sort algorithms

We researched a paper [8] about the in-place MSD sort algorithms called mate sort. It uses in-place partitioning to decrease the space complexity from O(n) to O(k) where k is the number of bits. The mate sort processes the input data one digit at a time. The author came up with three methods for partitioning. The partitional method is quite similar to partition method of quick sort except the extra "bitloc" parameter. The first is "sequential partitioning" which has to generate r – 1 calls to split data into r parts and recursively call mate sort method for each part. A better improvement is to partition around the middle digit value. The second method is using divide-and-conquer where the author proofed that the number of element accesses of the first digit location is (r - 1)n[n/2 logr]. The last method is called "permutation-loop", American Flag sort algorithm. The core algorithm of this method is preprocessing to find the count of elements belonging to a particular digit value so as to decide the place to put the element. However this method leads to worse performance than the second method.

This whole paper is based on English characters, so the highest character location (charcount * 8 - 1) is small. And this implementation could have better performance than Arrays.#sort() method.

3 PROJECT ALGORITHMS

3.1 Discussion of Collator

Instead of using the default Collator class in Java, we use the java library icu4j. The International Components for Unicode (ICU) library provides robust and full-featured Unicode services on a wide variety of platforms. We use **com.ibm.icu.text.Collator** to sort (support Chinese characters sorted according to pinyin) with higher efficiency and better compatibility.

Precisely, there are some characters not in Pinyin order while sorting with the default java Collator. For characters "啊 这 都 能 梵", the character "梵" will be the third in the sequence with pinyin order, but the sorting result of default Collator principle is at last . The reason for it is

mainly because the default Collator order is locale-sensitive in specified language, the same data might process inconsistently under different conditions. And the rule using default Chinese sorting is not only based on Pinyin order, but also on strokes and uncommon characters *(*for example, 梵).

Therefore, our code is based on ICU library which is strictly Pinyin order for Chinese characters.

3.2 Discussion of algorithm

There is a theory says that Radix sort is the best algorithm to sort Strings. What's more, among the best-known, simplest and fastest string sorting algorithms is the MSD (Most Significant Digit first) radix sort [3]. But will it perform perfectly all the time? There are some kinds of reasons that we do not think the fundamental MSD sort will perform really well.

The problem we assume is based on the sorting scheme of MSD sort. While sorting with this algorithm, we put elements in different buckets according to their radix.

The cost of the number of buckets will obviously drag down the performance of MSD sort. With Unicode (R = 65536) the sort might be thousands of times slower [4]. The total length of the distinguishing prefixes of the strings is the minimum number of characters that need to be inspected, and provides a lower bound for the problem complexity. The best theoretical variants of radix sorting have time complexity O (D + σ) [5] when D is the prefix length and σ is the alphabet number. The slowing down thing is the space cost through the recursive process. MSD, different from LSD sorting method, every loop we sort one byte, we have to search for the same radix and sort the rest recursively. In that case, the space cost will be large compared to its time complexity because the memory required depends on the number of bits on every pass, in another word, radix.

Therefore, we assume that the Most Significant Digit first radix sort will not doing really good on this experiment. We implement the other 4 sorts for the same Pinyin order with the same scheme icu.text.Collator and run them with benchmark to prove that.

Additionally, we have to point out that the test source is a list of Chinese names. It is quite a unique string sample and will actually affect the performance of LSD sort.

3.3 Discussion of implement

Compared to the normal implements of MSD sort, the Chinese pinyin order's change is about the sequence of every character's code. The original Unicode order which depends on the radicals of Chinese is not the required sequence. So, the modification we did on the Radix sorts is to change the default #charAt() method and make the encode of characters in order. With comparison-based algorithms we directly change the compare method to implement the sorting.

Please notice that all the tests and comparison we do on of every sorting algorithm are based on sorting Chinese pinyin order that we have done the modify implement but not the original method to sort English characters.

4 Implementation

For comparison sort like Tim sort, Dual-pivot quick sort and insertion sort. We user compare method in Collator.

```
protected Collator collator;

{
    collator = Collator.getInstance(Locale.CHINA);
}



@Override
public int compare(X v, X w) {
    return collator.compare(v, w);
}
```

For non-comparison sort like MSD sort, LSD sort. We use #getCollationKey().toByteArray method to create a new order for Chinese character.

```
阿 [41, 1, 5, 1, 5, 0]

苏 [136, 225, 1, 5, 1, 5, 0]

何 [101, 157, 1, 5, 1, 5, 0]

中 [160, 67, 1, 5, 1, 5, 0]

辰 [86, 254, 1, 5, 1, 5, 0]
```

After some experiments of #toByteArray() method, It turns out that bytes[0] plus bytes[1] will decide the order of the Chinese character, and the max number of each byte value is 255. But we cannot simply multiply bytes[0] and bytes[1], since bytes[0] firstly decides the order and then it's bytes[1] turn. So we must give bytes[0] priority, multiplying 255 is a way to solve the priority problem. Finally we convert the byte number to int number to fit MSD/LSD sort method, and then replace #charAt() method. This method will generate 65281 numbers, similar to Unicode numbers.

```
private static int ChineseCharAt(String s, int d) {
    if (d < s.length()) {
        byte[] bytes = collator.getCollationKey(String.valueOf(s.charAt(d))).toByteArray();
        if (bytes.length < 7) {
            return (bytes[0] & 0xFF) * 255;
        } else {
            return (bytes[0] & 0xFF) * 255 + (bytes[1] & 0xFF);
        }
    } else return -1;
}
```

For husky sort, we use the same method as MSD sort to acquire the sequence of Chinese character and replace #charAt() method and then convert the character to long type. As for Tim sort and insertion sort in the sort method, we use compare method above.

In particular, although Huskysort is actually based on comparison, the modification we did to Huskysort still includes the same ChineseCharAt() method with MSD/LSD sort since the important part of Huskysort is encoding [6]. We make changes to the processes including the string-to-long encoding part and the underlying compare sorting scheme using the scheme above.

Please find the code on GitHub for more implement details.

5 Benchmarks Result

5.1 System Environment

Model Name:	MacBook Pro
Model Identifier:	MacBookPro15,2
Processor Name:	Quad-Core Intel Core i5
Processor Speed:	2.3 GHz
Number of Processors:	1
Total Number of Cores:	4
L2 Cache (per Core):	256 KB
L3 Cache:	6 MB
Hyper-Threading Technology:	Enabled
Memory:	8 GB

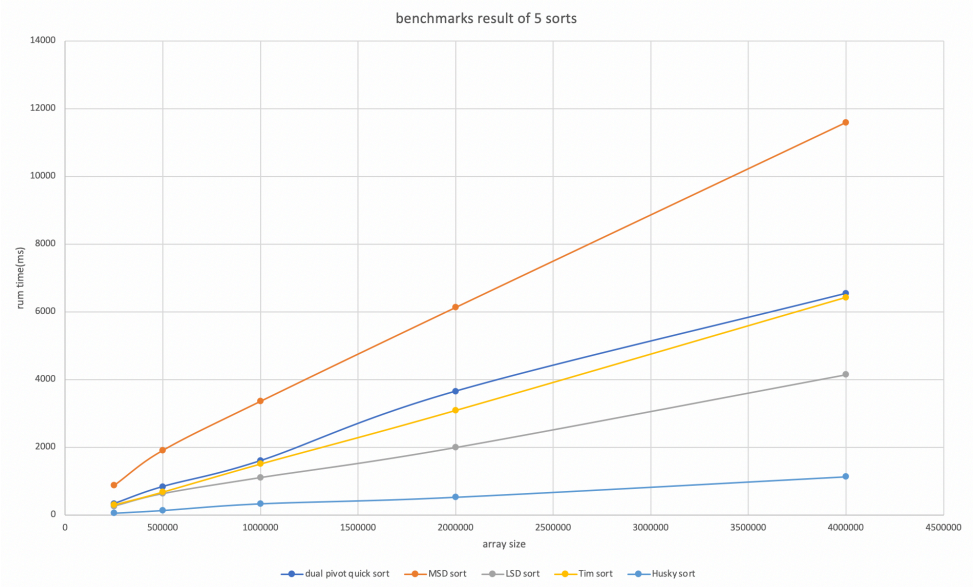
Figure 2. System Environment

5.2 graph

After experiments on different array sizes from 250k to 4M, the average run time(ms):

array size	dual pivot quick sort	MSD sort	LSD sort	Tim sort	Husky sort
250000	328.1868106	867.2820422	256.1526946	295.2662282	52.67957
500000	832.1055478	1904.487389	625.330445	675.6229574	133.21832
1000000	1596.007458	3352.816691	1099.470063	1504.238901	329.79776
2000000	3650.253493	6125.18087	1988.880726	3085.868225	523.04269
4000000	6541.859975	11589.54949	4142.186014	6423.562164	1123.6951

Figure 3. Benchmark result table



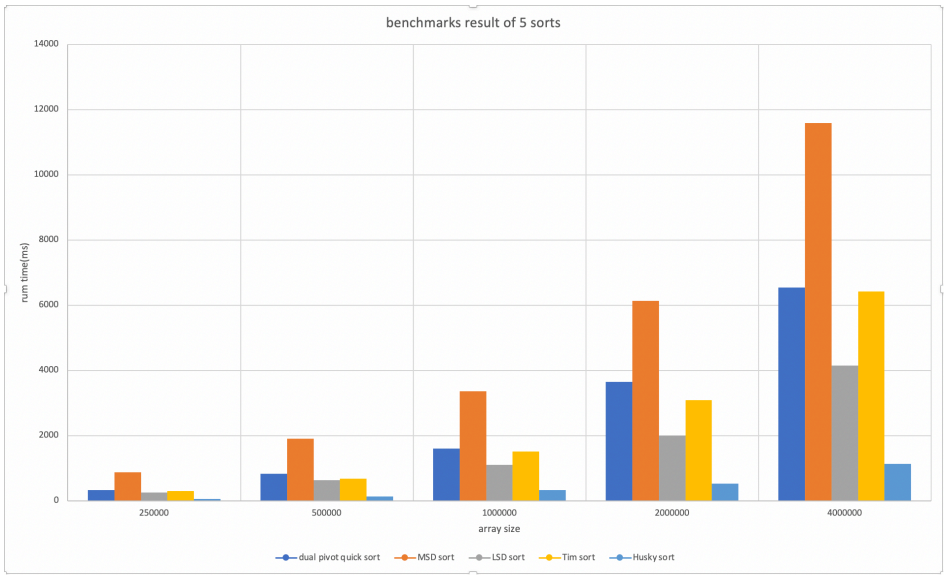


Figure 3. Benchmark result chart

5.3 summary

According to Figure 3, the best performance is husky sort, LSD sort is better than Tim sort and Dual pivot quick sort which have similar performance, MSD sort behaves worse than other sorts when sorting Chinese words.

6 Conclusion and Analysis

We have demonstrated that the performance of MSD Radix sort has its disadvantage while sorting Chinese in pinyin order with Unicode. The performance using our implement method is

Huskysort > LSD sort > Timsort ~ Dual pivot quick sort > MSD sort

This result is quite predictable according to the discussion before. Husky sort rank the best because it reduces process time when sorting Objects to linear phase. What we have to mention here about LSD sort having a fast performance is because of the testing samples: Chinese names. In common case Chinese names have 2-3 characters (sometimes up to four), and our sample is made up of common names. Strings with the same or similar numbers of characters suit LSD algorithm well and this test source erases the advantage of MSD sort.

References

[1] C.A.R. Hoare, Quicksort, *Computer Journal* 5(1) (1962) 10–16.

[2] J.W. Williams, Algorithm 232: Heapsort, *Communications of the ACM* 7(6) (1964) 347–8.

[3] J. Kärkkäinen and T. Rantala, Engineering radix sort for strings, Lect. *Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5280 LNCS*, pp. 3–14, 2008.

[4] Robert Sedgewick, Kevin Wayne, *Algorithms 4th*, ISBN-13: 978-0-321-57351-3

[5] Paige, R., Tarjan, R.E.: Three partition refinement algorithms. *SIAM Journal on Computing 16(6)*, 973–989 (1987)

[6] R.C. Hillyard, Yunlu Liaozheng, Sai Vineeth K.R, Husky Sort, arXiv:2012.00866

[7] Kärkkäinen J., Rantala T, Engineering Radix Sort for Strings. In: Amir A., Turpin A., Moffat A. (eds) *String Processing and Information Retrieval. SPIRE 2008. Lecture Notes in Computer Science*, vol 5280.

[8] Al-Darwish N. Formulation and analysis of in-place MSD radix sort algorithms. *Journal of Information Science.* 2005;31(6):467-481.