

Yuhan Yang (001094267) Luo Chen (001564677) Zhijie Li (001563872)

Program Structures & Algorithms

Fall 2021

Final Project

◎ **Task (List down the tasks performed in the Assignment)**

Your task is to implement MSD radix sort for a natural language which uses Unicode characters. You may choose your own language or (Simplified) Chinese. Additionally, you will complete a literature survey of relevant papers and you will compare your method with Timsort, Dual-pivot Quicksort, Huskysort, and LSD radix sort.

◎ **Conclusion:**

We compare the following sorting algorithms: MSD radix sort(with two cutoff value. First one is to determine the stop length of array partition. The second one is to determine the length of comparison key), Timsort, Dual-pivot Quicksort(using Introsort with cutoff for recursion depth), Huskysort, and LSD radix sort(with cutoff for comparison key length).

Findings:

The fastest sorting algorithm to sort Simplified Chinese is Huskysort. The slowest sorting algorithm to sort Simplified Chinese is Dual-pivot Quicksort.

When the array length is small, like 250K, the difference between LSD radix sort and MSD radix sort is not obvious. As the length grows bigger, MSD radix sort seems to have a better efficiency than LSD radix sort.

Meanwhile, we try to find a best cutoff value for MSD radix sort and LSD radix sort. It seems that the cutoff value of 5 is not a very good choice since the difference of efficiency between MSD radix sort and LSD radix sort with or without cutoff is ambiguous. So, finding the appropriate value for cutoff may be the one of the future work of our final project.

Recommendations:

The reason why non-comparison sorting algorithms are significantly faster than the comparison sorting algorithms is that non-comparison sorting algorithms use Collator

only one time to convert the incomparable Chinese characters into comparable bytes array, whose time complexity is $O(n)$. While comparison sorting algorithms need to use Collator every time when it is doing the compare operation.

Meanwhile, huskysort encode only first 7 bytes of the characters, after the first time sorting, huskysort using Timsort to sort the “partial ordered” array. However, radix sort need to encode the whole characters, which is 22 bytes for three words Chinese, leads to the worse performance in efficiency.

Based on the analyze of the reason that causing the efficiency difference, we came up with some optimizations:

For radix sort, we can try to find an appropriate cutoff value, which means how many bytes we use to go through first sorting. If the cutoff value is less than the length of the characters, we need to use Timsort to do the second sorting in order to make sure that the elements are completely ordered. How to find the appropriate cutoff value still need some future work.

For the comparison sorting algorithms, we can construct a key array which store the encode value of each character. We only run the sorting on the key array, but we need to swap both key array and value array when we doing the swap operation.

Actually, we can abstract this concept of building key-value array when sorting incomparable elements as KeySort. Huskysort and our optimizations on radix sort are both based on this concept.

🕒 Evidence to support the conclusion:

Experiment environment:

Server environment:

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.xlarge	4	16	EBS only	-	Moderate	Yes
<input type="checkbox"/>	t2	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
<input type="checkbox"/>	t3	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes

System: ubuntu 20

VM settings: Max. Heap Size (Estimated): 1.73G

Note:

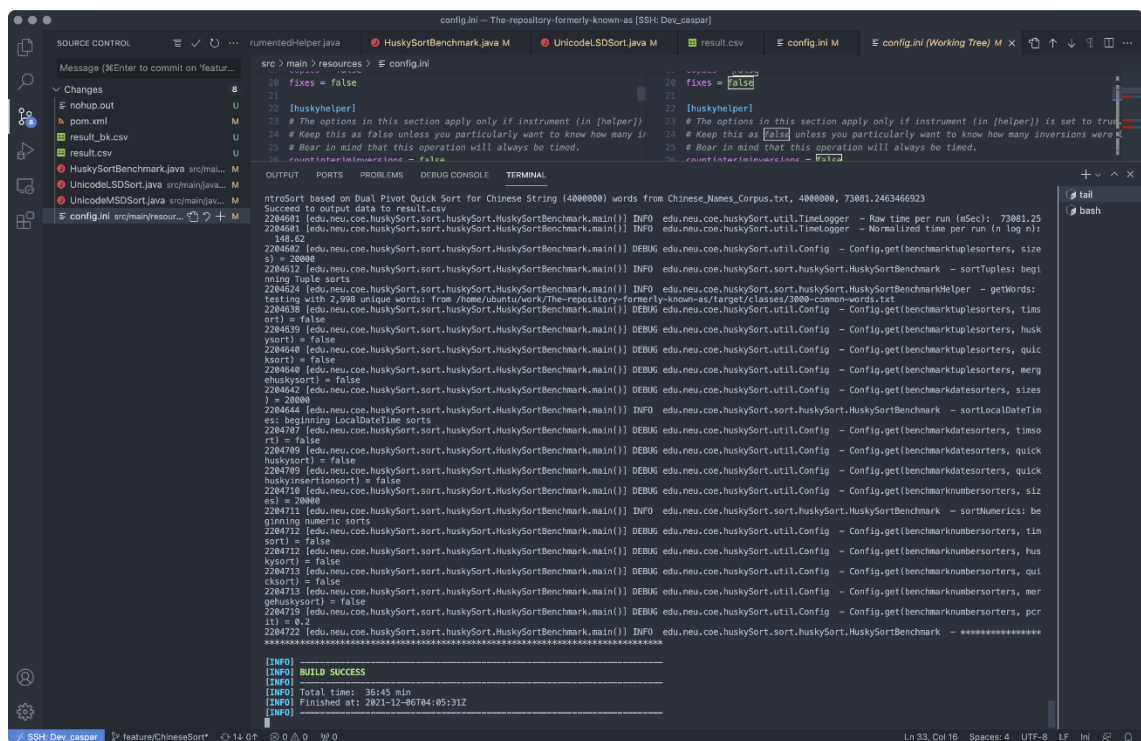
1. When the array length reach 4M, the cutoff value(for partition length) for MSD radix sort need to be set from 15 to 31 in order to prevent stack overflow.
2. We use the Introsort, which is based on Dual-pivot Quicksort, in order to prevent stack overflow.

Relative expression:

By using Class java.text.Collator, we are able to compare two Chinese characters in the specific rule. For comparison sort, we override the compare function. As a result, we can sort the Chinese characters in the order of Pinyin.

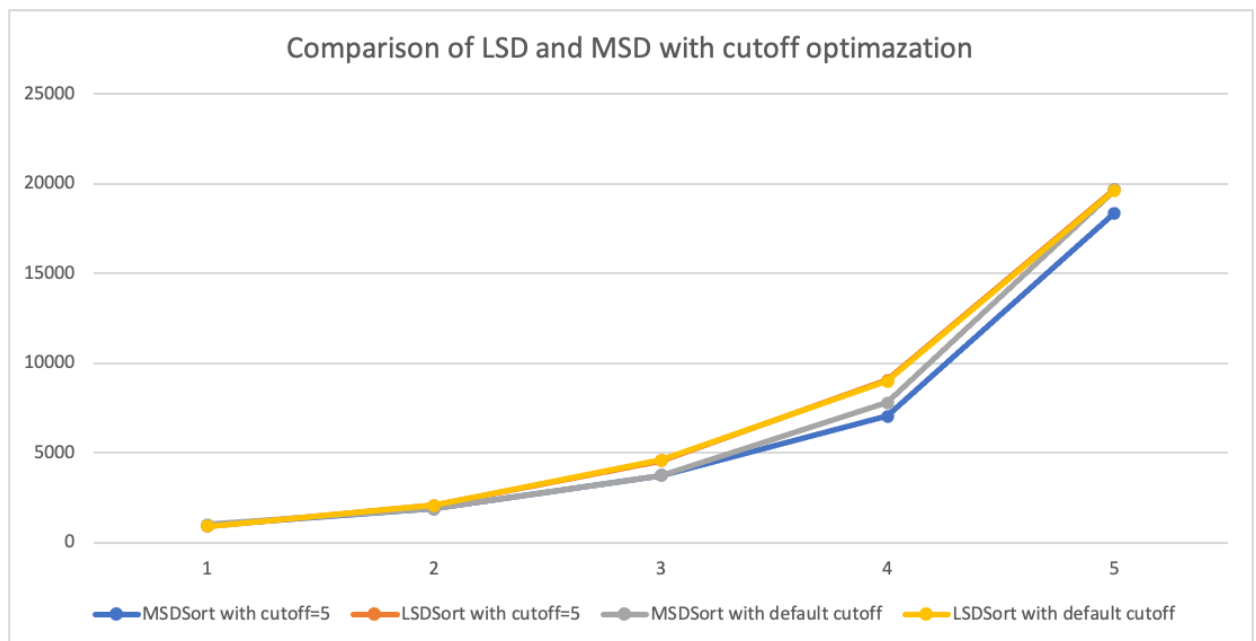
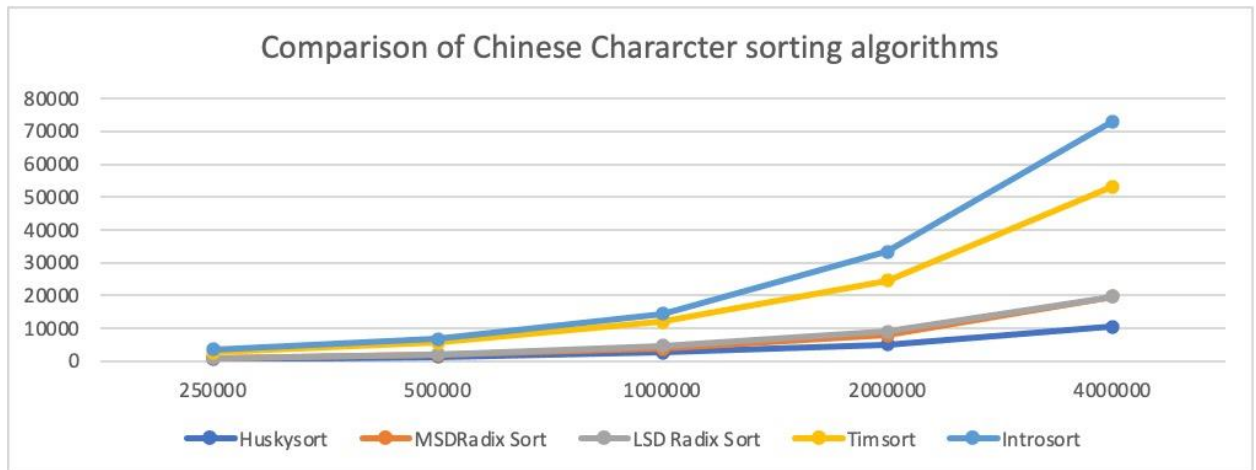
For radix sort, Collator can convert one Chinese character into a key, which combines two bytes. We can do radix sort for these bytes. When doing swap operation for the keys, we have to swap the original Chinese characters as well. The main difference between MSD radix sort and LSD radix sort is that MSD radix sort recursively sort the bytes from left to right while LSD radix sort recursively sort the bytes from right to left. In addition, the length of these byte array must be the same for LSD radix sort.

1. Output



```
config.ini - The repository formerly known as [SSH: Dev_caspar]
src > main > resources > config.ini
20 fixes = false
21
22 [huskyhelper]
23 # The options in this section apply only if instrument [in [helper]]
24 # Keep this as false unless you particularly want to know how many in
25 # Bear in mind that this operation will always be timed.
26 countInterInversions = false
27
28 OUTPUT PORTS PROBLEMS DEBUG CONSOLE TERMINAL
ntroSort based on Dual Pivot Quick Sort for Chinese String (4000000) words from Chinese_Names_Corpus.txt, 4000000, 73081.2463466923
Succed to output data to result.csv
2284681 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.util.TimerLogger - Raw time per run (mSec): 73081.25
2284681 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.util.TimerLogger - Normalized time per run (n log n):
146.62
2284682 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkTuplesorters, size
s) = 20000
2284682 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark - sortTuples: begi
ning Tuple sorts
2284684 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmarkHelper - getWords:
testing with 2,998 unique words: /home/dunhu/work/The-repository-formerly-known-as/target/classes/2000-common-words.txt
2284638 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkTuplesorters, time
ort) = false
2284639 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkTuplesorters, husk
ysort) = false
2284640 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkTuplesorters, quic
ksort) = false
2284640 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkTuplesorters, merg
ehuskySort) = false
2284642 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkDatesorters, sizes
) = 20000
2284644 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark - sortLocalDateTim
es: beginning LocalDateTime sorts
2284787 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkDatesorters, time
rt) = false
2284789 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkDatesorters, quick
huskySort) = false
2284789 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkDatesorters, quick
huskyInsertionsort) = false
2284710 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, siz
es) = 20000
2284711 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark - sortNumerics: be
ginning numeric sorts
2284712 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, tin
sort) = false
2284712 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, husk
ysort) = false
2284712 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, qui
ckhuskySort) = false
2284713 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, mer
gehuskySort) = false
2284719 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] DEBUG edu.neu.coe.huskySort.util.Config - Config.get(benchmarkNumbersorters, pcr
t) = 0.2
2284722 [edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark.main()] INFO edu.neu.coe.huskySort.sort.huskySort.HuskySortBenchmark - *****
*****
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 36:45 min
[INFO] Finished at: 2021-12-06T04:05:31Z
[INFO]
```

2. Graphical Representation



◉ **Unit tests result:(Snapshot of successful unit test run)**

