



NORTHEASTERN UNIVERSITY

COLLEGE OF ENGINEERING

INFO 6205 – Program Structures and Algorithms



Document Control

Document Details	
Author	Anurag Parla
Author	Hiral Nagda
Author	Pranoti Dhabu
Title	MSD Radix sort for a natural language using unicode characters

Version and Distribution History			
Version #	Date	Description of Change	Author
1.0	12/04/2021	Initial Draft	Pranoti Dhabu

Document Approvals

Name	Title	Signature	Date
Prof. Robin Hillyard	Associate Professor		



Contents

Table of Contents

Contents.....	3
<i>Solution Overview</i>	<i>4</i>
Summary	4
<i>MSD Radix sort.....</i>	<i>4</i>
Background.....	4
<i>Experiment.....</i>	<i>4</i>
Assumptions	4
Challenges	5
Running our experiment.....	5
Conclusions from the experiment.....	5
<i>Observations</i>	<i>5</i>
Conclusion	8
<i>Unit Tests</i>	<i>8</i>
<i>Appendix – A – References.....</i>	<i>11</i>

Solution Overview

This solution implements sorting of strings of any natural language using Unicode characters and benchmarks the performance of various algorithms for a comparative study.

Summary

The main purpose of this solution is to implement Unicode encoding of strings and perform sorting using MSD radix sort, LSD radix sort, dual pivot quick sort, husky sort and timsort. Followed by benchmarking on a natural language with a data set up to 4M records.

MSD Radix sort

MSD Radix sort is a key-indexed counting sorting algorithm majorly used for sorting strings. The algorithm essentially creates a partition for each character of the sort-key while traversing from left to right and this method of sorting is applied to each partition recursively.

Background

Strings are used as the sort key in majority of applications and hence new algorithms were being researched and developed for effective and fast sorting to lower the time complexity. LSD and MSD Radix sort algorithms were developed looking at these constraints. Key-indexed sorting methodology has three main components – computing frequency counts, converting counts to indices and distributing the data.

As part of this project, we have modified algorithms proven to be efficient to sort natural language words based on Unicode value. We consider MSD radix sort, LSD sort, dual pivot quick sort, timsort and husky sort algorithm to compare their time complexity.

Experiment

In our experiment we have used primarily used Devanagari script (for Hindi language) as the natural language for sorting. Before benchmarking for Hindi language, we first established confidence by sorting English words and then proceeded towards native languages like Hindi, Gujarati (Sanskrit script) and simplified Chinese using collators for respective languages.

Further we benchmarked the five sorting algorithms for data set varying from size 1000 to 4M by doubling method.

We have re-used code from the class repository as well as Husky sort repository and made the necessary modifications for passing collator object as well Unicode conversion.

Assumptions

Following are the assumptions for our experiment:

1. The data set (corpus) is a partially random list of words. The corpus contains approx. 40k unique

words which were increased to 4M.

2. All experiments are performed on M1 MacBook Air machine. Any machine dependent factors determining running time will be common across all runs. (It was ensured the runs were performed on a fully charged and plugged-in machine with no applications open).

Challenges

Throughout the progress of our project, we came across below challenges:

1. We found difficulties in getting a clean corpus of Hindi words. While we tried cleaning up the data by writing small python scripts, not all corpus was cleaned. Hence, we decided to repeat the corpus of 40k words and increased it to 4M by duplicating words. Thus, the data set is partially random array of repeating sequences.
2. During benchmarking, since we are also timing pre-processing i.e., Unicode conversion the initial run for sorting 1k words took approx. 850ms for MSD Radix sort. After further analysis, it was observed the logic we used for array creation also included reading the file contents from disk which accounted for large run time. We decided to cache the same and the run time was reduced considerably to 9ms.

Running our experiment

We implemented benchmark java files for all the 5 sorting algorithms and ran the benchmark for data set varying from 1k to 4M.

Benchmarking numbers obtained are an average taken over 10 runs.

Conclusions from the experiment

We can derive the following conclusions from our experiments.

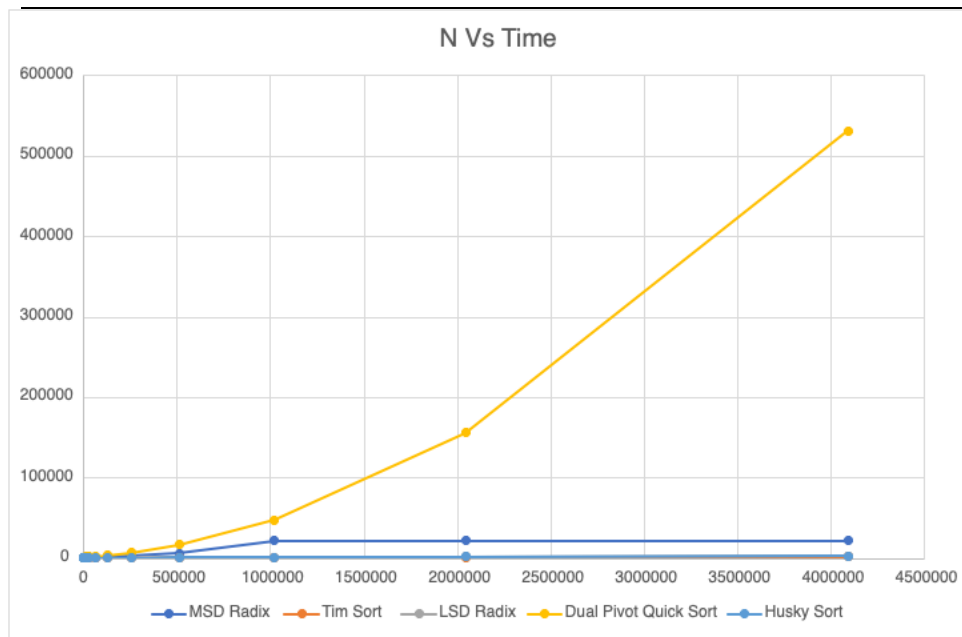
1. Time taken to sort the array of strings is directly proportional to the size of array and doubles as the input array size doubles.
2. Apart from dual pivot quick sort, all algorithms show a linearithmic growth graph.
3. We noticed log-log plot for MSD radix flattens after array size increases beyond 1M.
4. Log log plot indicates timsort, husky sort and LSD radix sort are converging as array size increases. Per the implementation of husky sort, as number of comparisons are less it is more efficient.

So, what these experiments mean,

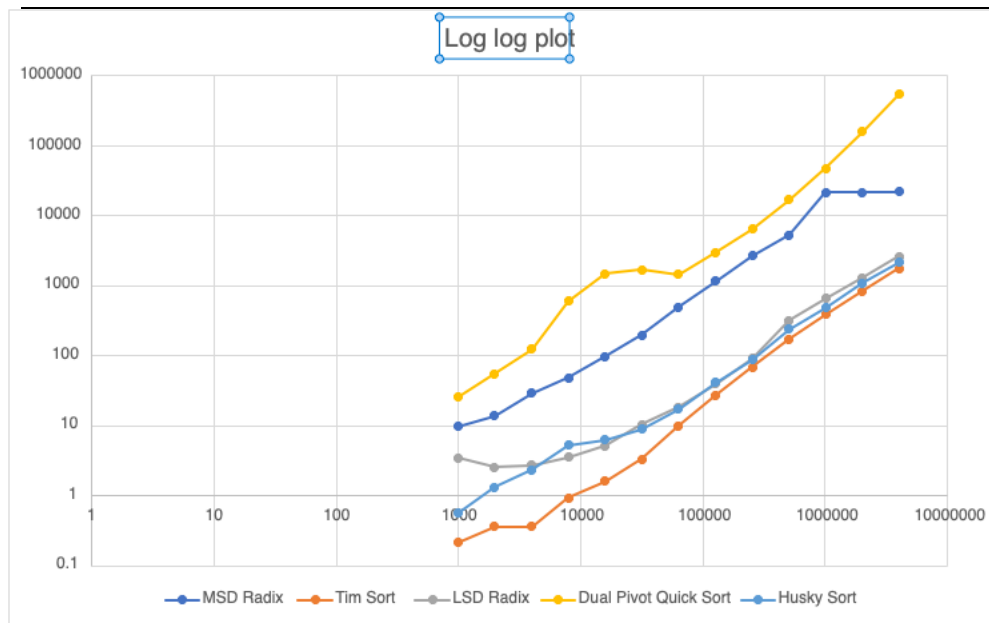
1. Husky sort has proven to be the most efficient algorithm for string sorting closely followed by timsort, LSD radix sort and MSD radix sort when input size is considerably large (over 1M).
2. For small size input, timsort is more efficient sorting algorithm.

Observations

The following observations were noted by our benchmarking runs.



Input Size	MSD Radix	Tim Sort	LSD Radix	Dual Pivot Quick Sort	Husky Sort
1000	9.6	0.21	3.46	25	0.55
2000	13.8	0.36	2.53	54.4	1.32
4000	28.9	0.36	2.68	121.8	2.31
8000	47.8	0.92	3.48	590	5.23
16000	96.1	1.58	5.11	1456.2	6.25
32000	196	3.3	10.2	1662.6	8.81
64000	493.2	9.86	18.22	1427.1	17.12
128000	1139.9	27.15	38.83	2919.4	40.62
256000	2639.8	67.5	90	6387.2	87.12
512000	5266.1	169.74	314.2	16549.2	238.11
1024000	21085.1	380.49	645.98	47029.8	480.24
2048000	21306.8	832.76	1288.6	155186	1078.07
4096000	21581.3	1739.78	2572.97	531048	2144.79

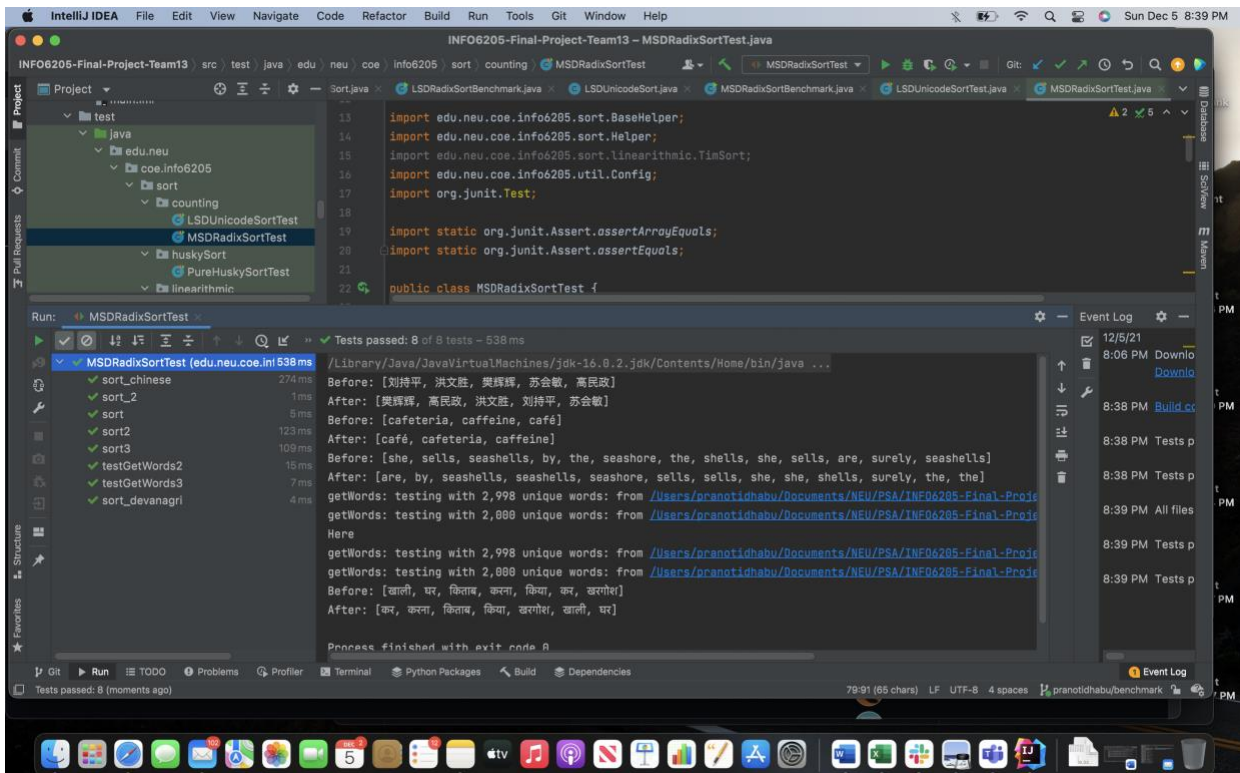


Conclusion

As demonstrated in the previous section, it is evident that dual pivot quick sort is not effective for sorting strings. Husky sort, MSD radix sort, timsort and LSD radix sort are proven to be efficient as per the benchmarking runs and graphs plotted.

Unit Tests

The following unit tests will demonstrate the validity of our experiments.



```
import edu.neu.coe.info6205.sort.BaseHelper;
import edu.neu.coe.info6205.sort.Helper;
import edu.neu.coe.info6205.sort.Linearithmic.TimSort;
import edu.neu.coe.info6205.util.Config;
import org.junit.Test;

import static org.junit.Assert.assertArrayEquals;
import static org.junit.Assert.assertEquals;

public class MSDRadixSortTest {

    @Test
    public void sort_chinese() {
        Before: [刘持平, 洪文胜, 樊辉辉, 苏会敏, 高民政]
        After: [樊辉辉, 高民政, 洪文胜, 刘持平, 苏会敏]
    }

    @Test
    public void sort_2() {
        Before: [café, cafeteria, caffè, café]
        After: [café, cafeteria, caffè, café]
    }

    @Test
    public void sort() {
        Before: [she, sells, seashells, by, the, seashore, the, shells, she, sells, are, surely, seashells]
        After: [are, by, seashells, seashells, seashore, sells, she, she, shells, surely, the, the]
    }

    @Test
    public void sort2() {
        getWords: testing with 2,998 unique words: from /Users/pranotidhabu/Documents/NEU/PSA/INF6205-Final-Proje
        getWords: testing with 2,888 unique words: from /Users/pranotidhabu/Documents/NEU/PSA/INF6205-Final-Proje
    }

    @Test
    public void sort3() {
        Here
        getWords: testing with 2,998 unique words: from /Users/pranotidhabu/Documents/NEU/PSA/INF6205-Final-Proje
        getWords: testing with 2,888 unique words: from /Users/pranotidhabu/Documents/NEU/PSA/INF6205-Final-Proje
    }

    @Test
    public void testGetWords2() {
        Before: [खाली, घर, किताब, कन्या, किया, कम, खरीद]
        After: [कम, कन्या, किताब, किया, खरीद, खाली, घर]
    }

    @Test
    public void testGetWords3() {
    }

    @Test
    public void sort_devanagiri() {
    }
}
```




```
import java.io.IOException;
import java.math.BigInteger;
import java.util.Arrays;

import static org.junit.Assert.*;

public class PureHuskySortTest {

    //private final BaseHelper<String> helper = new BaseHelper<>("dummy helper", Config.load(PureHuskySortTest.class));

}
```

Run: PureHuskySortTest x Tests passed: 10 of 10 tests - 311ms

PureHuskySortTest (edu.neu.coe.inf 311ms)

- testSortString1 238ms
- testSortString2 12ms
- testSortString3 21ms
- testSortString4 4ms
- testSortString5 2ms
- testSortString6 15ms
- testFloorLg 5ms
- testSortString5_new 3ms
- testWithInsertionSort 5ms
- testInsertionSort 6ms

Process finished with exit code 0

Event Log

- 8:38 PM Build cc
- 8:38 PM Tests p
- 8:38 PM Tests p
- 8:39 PM All files
- 8:39 PM Tests p
- 8:39 PM Tests p
- 8:40 PM All files
- 8:40 PM Tests p
- 8:40 PM Tests p

```
import java.io.IOException;
import java.math.BigInteger;
import java.util.Arrays;

import static org.junit.Assert.*;

public class PureHuskySortTest {

    //private final BaseHelper<String> helper = new BaseHelper<>("dummy helper", Config.load(PureHuskySortTest.class));

}
```

Run: PureHuskySortTest x Tests passed: 10 of 10 tests - 311ms

PureHuskySortTest (edu.neu.coe.inf 311ms)

- testSortString1 238ms
- testSortString2 12ms
- testSortString3 21ms
- testSortString4 4ms
- testSortString5 2ms
- testSortString6 15ms
- testFloorLg 5ms
- testSortString5_new 3ms
- testWithInsertionSort 5ms
- testInsertionSort 6ms

Process finished with exit code 0

Event Log

- 8:38 PM Build cc
- 8:38 PM Tests p
- 8:38 PM Tests p
- 8:39 PM All files
- 8:39 PM Tests p
- 8:39 PM Tests p
- 8:40 PM All files
- 8:40 PM Tests p
- 8:40 PM Tests p



The screenshot displays the IntelliJ IDEA IDE interface. The main editor shows the `PureHuskySortTest.java` file with the following code:

```
import java.io.IOException;
import java.math.BigInteger;
import java.util.Arrays;

import static org.junit.Assert.*;

public class PureHuskySortTest {

    //private final BaseHelper<String> helper = new BaseHelper<>("dummy helper", Config.load(PureHuskySortTest.class))

}
```

The Run window at the bottom shows the test results for `PureHuskySortTest`. All 10 tests passed successfully. The test results are as follows:

Test Name	Duration
testSortString1	238 ms
testSortString2	12 ms
testSortString3	21 ms
testSortString4	4 ms
testSortString5	2 ms
testSortString6	15 ms
testFloorLg	5 ms
testSortString5_new	3 ms
testWithInsertionSort	5 ms
testInsertionSort	6 ms

The Event Log on the right shows the following events:

- 8:38 PM Build
- 8:38 PM Tests p
- 8:38 PM Tests p
- 8:39 PM All files
- 8:39 PM Tests p
- 8:40 PM All files
- 8:40 PM Tests p
- 8:40 PM Tests p



Appendix – A – References

Titles	Description/Link
Husky sort repo	https://github.com/rchillyard/The-repository-formerly-known-as
Class repo	https://github.com/rchillyard/INFO6205
Hindi corpus	https://wortschatz.uni-leipzig.de/en/download/Hindi
Textbook (for code references)	<i>Algorithms</i> , 4th Edition by Robert Sedgewick and Kevin Wayne, Addison-Wesley Professional, 2011, ISBN 9780321573513