# MSD Radix Sort Algorithms

**Ankita Senapati**
**Chandrakanth Chittappa**
**Venkata Raghu Teja Kumar Gollapudi**

**Northeastern University**

**INFO6205 Program Structure and Algorithms**

**Instructor*:* Robin Hillyard**

**Date*:* 12/05/2021**

## Abstract

The purpose of this project is to implement MSD radix sort for a natural language which uses Unicode characters. Any natural language can be used to implement the various sorting methods - Timsort, Dual-pivot Quicksort, Huskysort, and LSD radix sort.

Our team has used Hindi language string to implement and analyse the sorting methods.

This paper will demonstrate the details of implementing the MSD radix sort for Hindi words.

## Introduction

Radix sort is a sorting algorithm that sorts the elements by grouping the individual characters of the same index value and then sorting the elements based on their Unicode/ASCII values.

The most common forms of radix sorts are Most Significant Digit sorting (MSD) and Least Significant Digit (LSD) sorting techniques. Radix sort is considered to be one of the best methods for string sorting applications.

As part of this project, we have successfully demonstrated that the runtime for MSD sort, for data up to 4 million, is better than other sorting algorithms- Tim sort, Dual Pivot quicksort, LSD sort and Huskysort.

## Sorting Method Analysis

**Radix Sort-** Earlier, it was considered to be inefficient due to misconception that it uses an inordinate amount of time and space. In their paper, they describe an efficient way of combining the advantages of traditional LSD and MSD radix sort: Forward radix sort.

**MSD Radix Sort**- This sorting method quickly divides the array to be sorted into small subarrays. But it is important to carefully implement this method in order to avoid too many tiny subarrays. This gets more typical with its implementation for Unicode. Therefore, switch to insertion sort for small subarrays is a must for MSD string sort.

**Implementation of MSD Radix Sort:**

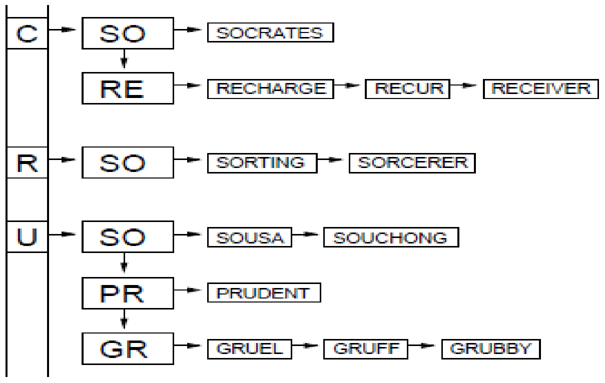
<u>Arne Andersson and Stefan Nilsson:</u>

They mention in their paper that in order to implement a string sorting algorithm efficiently, the strings should not be moved but only the pointers to them. This ensures constant time for string movement.

Another practical improvement discussed in their paper is switching to a simple comparison-based method for small subarrays.

Forward Radixsort- This algorithm, combines the advantages of LSD and MSD radixsort. This algorithm starts with the most significant digit, performs bucketing only once for each horizontal strip and inspects only the significant characters. The algorithm maintains the invariant that after the *i*th pass, the strings are sorted according to the first *i* characters.

Its implementation involves a linked list in order keep track of the groups of strings sorted after the *i*-passes. The use of linked list becomes a huge overhead but the overall running time is reduced by about 40% by swapping to Insertion sort for small subarrays.



**Parallel In-place radixsort algorithm**

<u>Rolland He</u>

In his paper, he discussed few innovations that have helped parallelize in-place radix sort, which in 2014 led to PARADIS sort.

In-place MSD radix sort: The need to reduce the amount of extra memory required from O(n) to O(1), the need to swap elements arise. So now instead of bucketizing each element, a histogram is built to count the number of elements that belong to each group. The parallelization of the radix sort as demonstrated by Rolland is a huge gain over the Forward radix sort in terms of both the runtime efficiency and the space complexity.

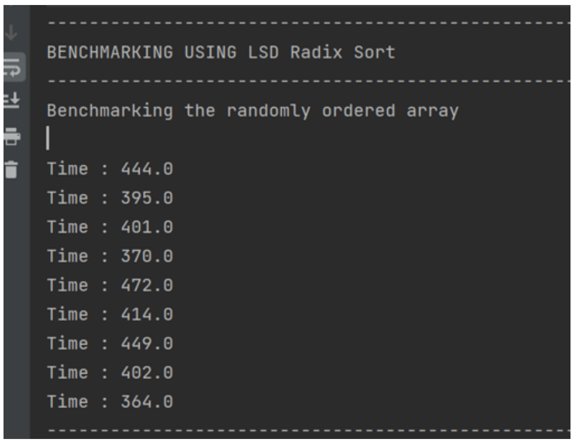
**Radix Sort Implementation for a natural language:**

As part of the final project, we implemented the two variations of radix sort- MSD and LSD for Hindi language.

Typically, for sorting of a natural language using Unicode, the Java Collator class is required. However, during the implementation, we found out that the Hindi language was successfully sorted even without the Collator class. The reason behind the same is that the lexical order of Devnagari script that is used by Hindi is in line with the order of the Unicode values.

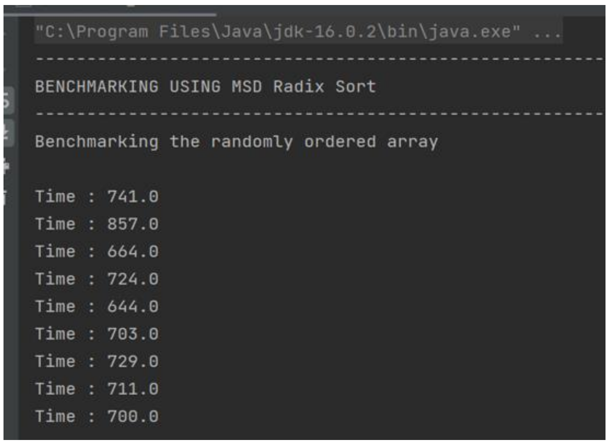
We used a HindiWords generator utility class to create text files containing hindi words. The implementation is quite easy and results were quite interesting. We had expected better performance from the MSD sorting algorithm, however the running time for LSD sort was much better than the former.

Below is the runtime comparison in milliseconds for LSD and MSD sort for 1 million Hindi words:



```
-------------------------------------------------
BENCHMARKING USING LSD Radix Sort
-------------------------------------------------
Benchmarking the randomly ordered array

Time : 444.0
Time : 395.0
Time : 401.0
Time : 370.0
Time : 472.0
Time : 414.0
Time : 449.0
Time : 402.0
Time : 364.0
-------------------------------------------------
```



```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" ...
-------------------------------------------------
BENCHMARKING USING MSD Radix Sort
-------------------------------------------------
Benchmarking the randomly ordered array

Time : 741.0
Time : 857.0
Time : 664.0
Time : 724.0
Time : 644.0
Time : 703.0
Time : 729.0
Time : 711.0
Time : 700.0
```

## Conclusion:

For a natural language with its lexical order same as its Java Unicode order, LSD radix sort proves to be an efficient sorting mechanism for strings of almost same length. When compared with other sorting mechanisms, the performance is as below:

LSD > TimSort > Dual-Pivot Quicksort > MSD > HuskySort

The above relation implies that the LSD radix sort has the best runtime performance compared to others.

## References

1. Andresson. A & Nisson. S (n.d.). Implementing Radixsort. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.4536&rep=rep1&type=pdf

2. *Introduction - Stanford University*. (n.d.). Retrieved December 6, 2021, from https://stanford.edu/~rezab/classes/cme323/S16/projects_reports/he.pdf.

3. Algorithhms 4th Edition by Robert Sedgewick, Kevin Wayne