

**Comparison of MSD Radix Sort for Natural Languages**

**Fall 2021 INFO6205 Project**

**Instructor**

Prof. Robin Hillyard

**Document Control**

|  |  |
| --- | --- |
| **Document Details** | |
| **001565993** | **Ankur Chauhan** |
| **001009233** | **Aparna Shelar** |
| **001502719** | **Mohit Nagpal** |

**Document Approvals**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Title** | **Signature** | **Date** |
| **Prof. Robin Hillyard** | **Associate Professor** |  |  |

**Contents**

[Solution Overview 4](#_Toc69854015)

[Summary 4](#_Toc69854016)

[Radix Sort 4](#_Toc69854017)

[LSD Radix Sort 5](#_Toc69854020)

[Steps](#_Toc69854021)…………………………………………………………………………………………………………………………………5

[MSD Radix Sort 6](#_Toc69854027)

[Steps 6](#_Toc69854028)

[MSD Radix Sort for variable length strings 7](#_Toc69854029)

[MSD Radix Sort for Hindi Words 7](#_Toc69854030)

[Steps 8](#_Toc69854033)

[Unicode 9](#_Toc69854034)

Test Cases………………………………………………………………………………………………………………………….9

[Benchmarking 10](#_Toc69854034)

[MSD Radix sort for Chiense words 11](#_Toc69854031)

[Steps 12](#_Toc69854036)

[Test Cases 13](#_Toc69854037)

[Benchmarking 14](#_Toc69854038)

[Performance Evaluation 15](#_Toc69854039)

[Conclusion 16](#_Toc69854040)

[Appendix – A 16](#_Toc69854041)

**Solution Overview**

This solution provides implementation of MSD Radix sort for natural language using Unicode characters. Solution contains sorting implementation for Hindi as well as Chinese languages.

**Summary**

The main purpose of this solution is to provide implementation of MSD Radix sort for Natural languages such as Chinese and Hindi. Also, to compare it with other sorting techniques such as Tim Sort, Dual Quick Sort, Husky Sort and LSD Radix Sort.

**Radix Sort**

**Radix sort** is a non-comparative sorting algorithm which is an integer based that sorts of data with integer key by grouping the keys by individual digits that share the same significant position and value. Radix sort uses counting sort as a subroutine to sort an array of numbers. Having said that radix sort is not limited to integers. Because integers can be used to represent strings by hashing the strings to integers, radix sort works for data types other than integer as well.

The algorithm is named radix sort as it specifies the *radix* r to be used which changes how the sort is performed. The radix, or base, of the number system is the number of digits that represent a single position in the number; a radix of 2 is binary (0-1), 10 is decimal (0-9), 16 is hexadecimal (0-F) and so on. Since the radix determines the number of buckets in addition to the word size w used in the algorithm, changing it can drastically change how the sort plays out. The algorithm works for the integer value, to sort the string we need to find the index of each character by iterating through word.

**Radix sort can be implemented in two ways -**

LSD Radix Sort – Least Significant Digit radix sort

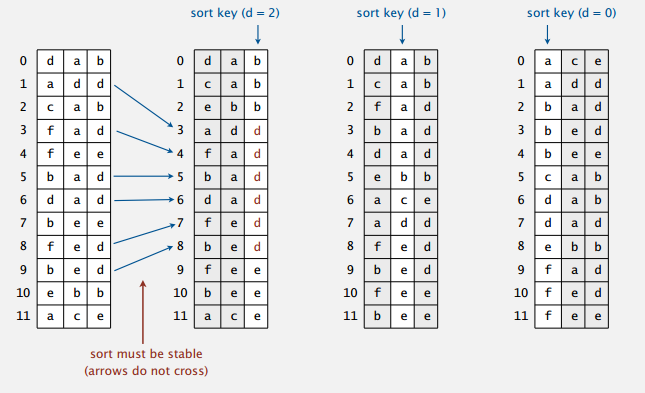
MSD Radix Sort – Most Significant Digit radix sort

LSD Radix Sort

LSD radix sort performs a counting sort on the provided list for each digit, starting from the least significant digit. LSD radix sort is stable, unlike the MSD variant as the relative order is retained after each sorting iteration.

**Steps:**

* Sort characters from right to left
* Stably sort using dth character as the key
* If the length of words varies, the character index for shortest word return -1
* After the first pass, strings are sorted by last dth characters.
* If the two strings have different sort key, then key-indexed sort puts them in proper relative order.
* If the strings have same sort key stability keeps them in proper relative order. For e.g., in below diagram 1st string and 3rd string have same character at (d==2) position, in this case they are placed in order which they appear in input array.
* Repeat the same for (d==1) and (d==0 keys.



**MSD Radix Sort**

MSD radix sort performs a counting sort on the provided list for each digit, starting from the most significant digit i.e first character. The biggest problem with LSD radix sort is that it starts at the digits that make the least difference. If we could start with the most significant digits, the first pass would go a long way toward sorting the entire range, and each pass after that would simply handle the details. The idea of MSD radix sorting is to divide all of the digits with an equal value into their own bucket, then do the same thing with all of the buckets until the array is sorted. Naturally, this suggests a recursive algorithm, but it also means that we can now sort variable length items and we don't have to touch all of the digits to get a sorted array. This makes MSD radix sort considerably faster and more useful.

Diagram

Description automatically generated

**Steps:**

* Sort array from left to right
* Partition array into R pieces according to first character
* After first pass, we will get all the strings starting with a followed b and so on.
* Do the same with each sub array
* Recursively sort all strings that start with each character

**MSD radix sort for variable length string:**

MSD radix sorts of element from most significant bit and character with an equal value into their own bucket, then do the same thing with all the buckets until the array is sorted. This means that we can now sort variable length items and we don't have to touch all the digits to get a sorted array. This is achieved by placing imaginary -1 at the end of each input string.

**![Text

Description automatically generated](data:image/png;base64,/9j/4AAQSkZJRgABAQEAkACQAAD/4RDcRXhpZgAATU0AKgAAAAgABAE7AAIAAAAGAAAISodpAAQAAAABAAAIUJydAAEAAAAMAAAQyOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE1vaGl0AAAFkAMAAgAAABQAABCekAQAAgAAABQAABCykpEAAgAAAAMxNQAAkpIAAgAAAAMxNQAA6hwABwAACAwAAAiSAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAyMToxMjowNSAyMzozODoyMQAyMDIxOjEyOjA1IDIzOjM4OjIxAAAATQBvAGgAaQB0AAAA/+ELGGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjEtMTItMDVUMjM6Mzg6MjEuMTUxPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPk1vaGl0PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAIkCaQMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APHqKKKACipbW3a7u4reMqHlcIpboCTitGXw5dRpPtntZJbcFpII5cyBR1OMf/XrOVSEXaTNI05zV4oyaK0LLRpr6ye7WaCKGOQRu0z7dvGc9OnamXulTWFxDHNJCyTAGOZHzGwzjOfaj2kOblvqHs58vNbQpUVuarBcQ6DZo09jPbRyMiSW2SxY8nJIFZrWEq6Wl+GRomlMRAJ3KwGefwpQqqSv52HKm4u3lcq0Vbl02eGG0c7Wa8GY41J3YzgZ+varcvh65jjlK3FrLLCpaWCKXc6AdePb2NN1YLdiVKb2Rk0Vu6Do1vfpLJc3NqR5EhWJpGDow6MQB0HWs6604288UUNzb3bynC/ZmLc5xjkCkq0HNw6oboyUFPoU6K1ZvD1zFHKVntZZYVLSwRS7pEA65Ht3xWVVRnGfwsmVOUPiQUU5EaWRY4xuZiFUDuTWpN4enhWUfarOSaFSzwJNl1x14xjj60SqRi7SYRpyn8KMmit3QdGt79JZLm5tSPIkKxNIwdGHRiAOg61nXWnm3niiiube7aU4H2di2DnGDkCpVaDm4dUU6MlBT6FOitaXw9cxxylbi1llhUtLBFLudAOvHt7Gp9B0a3v0lkubm1I8iQrE0jB0YdGIA6DrUyr01FyuVGhUclG1rmFRVu6sFtZooxe2s/mfxwuSqc9+BWjqsFxDoNmjT2M9tHIyJJbZLFjyckgVTqq8bdSVSet+hh0VesdJmvoHn82G3gjbaZZ32ru9B6mo7/T5tOmVJ9jB1DpJG25XX1BqvaQcuW+pPs5cvNbQq0UVseHLeZtQjntLmxSdW2JFdE/OSMcAD3oqT5IuQQjzyUTHoq19lluNW+y5QSyTeXkcKGLY/Krkvhy6ijnKz2ssluC0kMcuXCjqcY/+vSdWEbXZSpTlflWxk0Vu6Do1vfpLJc3NqR5EhWJpGDow6MQB0HWsy+shZMird21zuGc27lgv1yBSVWLm4LdA6UlBT6FWiitnR/DF3rFlNe/abOxsoXEbXN7L5aFyM7RwSTjnpVylGCvIzScnZGNRXR23hiWw8ZaTp2rpFPbXk0RV4Zd0c8TNjKsO3X0NWtW8DXMVxqsun3FjItm8kjWMdzunihDHBK+wxnkn15rJ4immlfc0VKbvoclRWxpHhq51ezlvPtVnY2cTiM3F7N5aFyM7RwSTjnpSX3hq+0/WbXTp2hZrzYbeeOTdFIrnAYMO2far9pDm5b6kckuXmtoZFFdVP8PtTha7hW802e8tAzSWUVxmYqvVguOmOcdcdq5WiFSFT4XccoSh8SCinwQyXNxHBAheWVgiKOrEnAFdFc+BdQt7e5aO9026uLRDJcWlvc75olH3iRjBx3wTTlUhD4mKMJS2RzVFdl4M8J2OswXdxf3+nEfYp2S3ed1khdfuyMAPujr1PHaue1fSV0mWNE1Ow1DzFJ3WUrOF9jlRzUKtBzdNbor2cuTn6FBI3k3eWjPtBZtozgDqabXU65cyaDo1lo2nnyRdWiXF7IvDTM/IUn+6B2qLSLDT4PC0+tX1g+pstyIPJErRrEu3O9ivPPT0rotq/L/hjK+i8zm6K2by20nUNTtV0edbGKePMq3bnZbvzkb8ZI44OO9bGu+EbK1trGS11XS4CbJZJA9w+Z35y6ZU8HoOnTpSs7XC+tjjqK2NN8N3Op6W+oJc2lvbRzeS73EpQKcZyeOnb1yelRaloNzpV9bwXU1uY7kBormOTdEyk43bsdB34oswujMorqPENre23hbTEkudLurKGSSOGWy3Fi3U7iQP0qhp/hi51DSRqX2uztrXzjCXuZSm1gAeePftk+1FtWgvoY1FaWo6DfabqkNhIqTS3AVoGhbcswbhSp7gmtN/A98r3EaX2nSy20Tyzxx3BZotoyQRjr29M96LO1wujmqKKfDDJcTxwwqXkkYIijuScAUtxjKK6C68IXVrHcY1DTZ7i2UtNaxXGZUA68EAEj0Brn6ACiiigAorY8OW8zahHPaXNik6tsSK6J+ckY4AHvVH7LLcat9lyglkm8vI4UMWx+VZ+0XM49jT2b5FLuVaK1pfDl1FHOVntZZLcFpIY5cuFHU4x/8AXqfQdGt79JZLm5tSPIkKxNIwdGHRiAOg61Eq9NRcrlqhUc1C1mzCoq1fWQsmRVu7a53DObdywX65AqrW0ZKSujGUXF2YUqqzsFQFmY4AAySa1dL8O3Op2Mt6bm0s7WNxH513LsVnxnaODk4q34l0uPQ/7INqvlXD2iyyyRyFt0m4/MDn2GMcVVrasm99jn2VkYq4KsDggjBFJXUanKde8Hpq92AdQtLkW00wGDMjLlS3qRjGa5ek9HYa1Vwp8UEs+7yYnk2jLbFJwPWmV1SXVzp/w0iazmktWudQdJGjJUyp5Y4yOop9GxdUjlaKKKQwooooAKKKKACiiigAooooAu6L/wAh2x/6+E/9CFdBDYT2OtalqVzsW0xPiXeCGLZAUd889K5myuPsd9Bc7d/kyK+3OM4OcZpt3P8Aarya427fNkZ9uc4yc4rmqUpTnptax00qsYRu907mnF/yJdx/1+p/6Aaz1kur0W1mpaQISsMfoWPQfU1XorWNO1353MpVLpLsrfjc6mXQdUPhWC2FnJ5y3TOU4yFKgZqnoMRv7e+0duHmUSRZ7Oh/qM/lWFRWSoy5XFvrfbrv3NXWjzJpbab9PuN29e1vPFkVvO+yyhdLcHOMIvHX3Oefetu0gnt7u5WTSrKzh8iURyIcu42nGDuOeO+K4eipnhuaKinpa3/B3KhiVGbk49b/APA2NjwzhtQniyA81rLHGCcbmI4FR29pPouuWL6nE0AEqud2PuhuTxWXRWrptybvozFVFyKNtU7/AJf5HZXLX1rNeXKafpcEO1yt2c/vFPoQxySD6VxtFFKjS9mv6/4JVat7Qkt9wuYtkgjbeNrscBTnqa6uaxkuhcya5p1tCojZxfwPt3Njg4zhs/SuQooq0nNpp2/r1ClVVO91f+vQ2PDOG1CeLIDzWsscYJxuYjgVFFpL2msWlvrUbW8UrjduIHy59e1ZlFN03zNp7kqouRRa2f8Al/kdxaQT293crJpVlZw+RKI5EOXcbTjB3HPHfFc/4Zw2oTxZAea1ljjBONzEcCseiso4dqMk3v8A13ZtLEJyi0tnf8vJdi5caRf2t1FbXFsyTTf6tCRlucVuy6Dqh8KwWws5POW6ZynGQpUDNctRWk6dSVtVp5f8EzhOEW9Hr5/8A3rS1l1HwwbKzAe5huzI0O4BipXGRnrzWfqdnJYNBBPciWUR5aMNkQkn7uc4zVGinGnKMr309BSqRcbW123Ctvwzpl5caraXcNuz28c675BjAxgmsSiqqRcouKdrkQkoyTaN/wDs28svFVrLdQNEk16ChOPm+f8A+vUunf8AI2an/u3P9a5uisnRclZvpbb/AIJsq6Urpdb7/wDANjwzhtQniyA81rLHGCcbmI4FUL3TrvTpFS9gaFmGVDEciq1FaqDU3JPcy506fK1t/wAAK7TTNPuPEXw5XTNICz31rqTTyW28K5jaMAMATyMjFcXRRUg5rR2aFCXK7nY2elTaN448N2d3frc3CzwtJAr7hakyD93nJGe5x61c0A5+I3iPPeG/z7/ergqKylQck03umtu5caqi00tmn9x3fhnSkm8Ii80fRbLW9UNyyXMd2+fIjAG0hNy8HnmrnijP9v8AgzfDbQsI4g0VocxIfO5C8ngfU15xRS+rt1Ody/q1u/6B7Vcjhbp+tzpPEeo3el/ETV7uwnaGdLyYK4wSASQevsa5uitrTfF+taRYraWFxDHCpJCtaROeTk8spP61pGDpwSirtJLtt95MpKc23om2yv4dtrG78SWFvq0nlWUk6rMxbbhc+vb0zXp2mWd5Y3+pR3PhvSdKtvsVysE0LbppRsOMHedwxyTivMdX8QalrpiOpzRy+TnZsgjjxnGfuKM9B1rNrKrQnWWrtp6/5GlOpGnK6V9v66nVfD7Ems39qGUTXemXEEKswG+Rl4XJ7msXV9C1PQZ44dXtHtZJF3orkHIzjPBrPorbkaqOaejMuZcnK/62/wAjpvF8bXMOkapCC1tNYxxbx0WRBhlPvR4csdZ+wm+8L35N35nlz2aMFYLjhiGOGH4cVj2ms39lY3Flb3BFrcqVlhYBlPvg9D7jmqNb395vuZW0SOl8aRxJc2JkS3j1JrfN+lvjaJMnGdvAbHXFXNb0a/1nRtHv9Mtzc21vpiRzOjD92yFiwIzniuOopX0aHY6SM/8AFsZ/+wsv/oo1kJLf6r9j05GafyyY7aLjjcckD6mqVFDd3cLaHe3XhPXX8CWFkumym4jvJZHjyMhSowetY84K/DKBTwRqr5H/AGzFc1RTbvfzt+Fv8gStb+u/+Z2l3dxWN94Nurg4ihtomc+gEh5q/p+izWOva3eyXNrJBcWd00BinVzKGBOcA5AA9a88roY/EWnWNrcLo+im1ubiBoGnku2lwrcNhdo5p3un8/xFazXy/A56p7EyDULcwzLBIJVKSucKhzwSfQVBRUp2dxvVHol1pUt7DfzeKNGsrULC8i6pay7PMkxwcbiH3H271wVnZXGoXS29nH5krAkLkDoM9TxUFXdN1E6b9paOPdLNCYlfdjy89T7moqOVm4LUumo8yU3oU2UqxU9QcHBzSUUVRJt+GdMvLjVbS7ht2e3jnXfIMYGME1J/Zt5ZeKrWW6gaJJr0FCcfN8//ANesCiueVObm3fRq23/BN1Uiocttb33/AOAdJp3/ACNmp/7tz/WqfhnDahPFkB5rWWOME43MRwKx6KHRumr7pL7h+395O2zb++3+RZvdOu9OkVL2BoWYZUMRyKrUUVur21MJWvodbYWFxrvgFLLS1Wa6tb5pZYd4DbGQANyeRkYqPxpA1tb6DBIys0enKrFG3DIZs4I61y1PhlaCeOVMbo2DDIyMg56Vbaf4fgrEJW/r5nRmNrD4blbkbH1C+V4UPBZEU5b6ZOK5mrepape6vdm51G4aeUjALcBR6ADgD2FVKT1dxrRWJba5ms7qO5tpDHNEwZHHVSO9dPrV7c6x4FstR1OQ3F2l88KzEAHZsBwce9cnWhY65qOm2k1rZ3JS3n/1kTKrKeMZwwODjuOaL6NBbVMz6KKKQwooooAKKKKACiiigAooooAe/wDqY/xplPf/AFMf40ygAooooAKKKKACinIjSSKicsxAHOOa1tUsp7PR7RLjTooCrsDcpKrmU9cHBPSolNRaj3LjByTfYx6K0tG0ltSuwJVlW3Cs7OiZzgZwD0zUepwWsJj+yW95DnO77Vjnp0wBS9pHn5FuP2cuTn6FGiitrwvp1te31xcaipezsLd7mWMHBk29Fz7kitUrmTdjFoq9qWoTazqAlMEUZwI4oLeMKqjPCgDr1+tbdh4HvlkuDrdtNbRJaSTIySITvVcgNjOM+hwaLO1wv0OWooopDCitKx0DUL+zku4ofLtIlJe4mYJHx2BPU+wrNoAKKVVZ3CIpZmOAAMkmtm68Ia7Z2klzcWDLHGu6TEiMyD1Kg5H4inZ2uF+hi0UVZ082Iux/aq3DW+DkWzKr57csCKQFaitzxLpVhpy6dNpZufJvbYT4uWVmX5iMfKAO1U9L0HUtZEh021aVY8b3LKir7bmIGfanZ3sK6tcz6KvSaLqMOrppc1q8d5IwRYmIG4npz0wfXOKtXPhTW7Oxku7iwdYYv9YwZSVHqQDnHvjFLpcfkY9FaOl6BqespK+m2pmSHG9yyoq57ZYgZ9qfJ4b1WK/trKa18u4ukLxI8irkDPUk4HQ9cU7MV0ZdFdbq/gLULO3tZbOLeGtFlufMuovkk53BeRkYA6Z+tU7HwtLqnhiK+06OWa7a7aFk3qqKgQHJJxjk45NHK9fL/hgujnqKs6hpt5pV41rqEDQTKASrenqCOCK3tSaLw5olrYW0ETXuoWomurmRAzKj9I0z93jqetHS4dbHMUVu+G9QgFwulanbRT2F5KFclB5kTHgOrdQRxx0rPvraXRdcntwwMtnOVVsZyVPBx+FFtgKVFdJqyQax4bj12G3jtrqO4+z3aQrtSQkZVwOx7GubpPR2HurhRTo42llSOMZd2CqPUmtCXw9qsMMkslowWLO/DKSMdTgHOPeolOMXaTsXGEpfCrmbRVuy0u81Hf8AY4DIE+8xIVR+JwKG0u8TUBYvAUuG6I7Bc8Z6k4o9pC9r6hyTte2hUorfv/Ct3BBbvbx7i0AebfPH8rc5A5GR+dYxtJhYi7KfuDJ5YfI+9jOMdelTCtTmrxZU6NSDtJENFWHsLlPs26PH2oAw/MPmGcfhz61al8ParDDJLJaMFizvwykjHU4Bzj3qnUgt2iVTm9kzNoq7aaPfX0AmtYPMj3lN25QAcZ5yePrUN5ZXFhcGC8iaKQDO0+nr70KcXLlT1Dkko81tCCiitrwvp1te31xcaipezsLd7mWMHBk29Fz7kitErmbdjFoq9qWoTazqCyNDDDwI4ooUCIi54A/Pqak1C1lttPtlmso4SpIMyyKxkPvg0ulxmbRVq30y7u4hLbw70LFd24DB/E8UjafdLei0eLbMeiswGfx6UAVqK17zw/cQxwtCm7MIeXdKnDc5A56fnU3hvUIBcLpWp20U9heShXJQeZEx4Dq3UEccdKaV3YTdlcwqKtapYtpmrXVjIdzW8rR7vXBxmqtJajCitHS9B1LWVkbTrUypFgO7OqKpPbLEDNQajpl5pN2bbUYGglwG2nByD3BHBHuKdmguVaKK0NL0HU9ZEh021aZYvvuWVFX23MQM0WbC9jPorXtNBnTxRZ6Tq8Mts00yI44DbWOMg8g/XkVNqfhLVbFbq5SylNjDIwEhZSwQEgMVHIGO+MUWdrivrYwqK0NM0LUtZEh061aVY8b3LKir7bmIGaZcaRf2mqLp1zbPFdOyqsbYG4k4GD0wfXpRZhdFKitqfwhrttazXE2nuI4M+Zh1JUA4J2g5xx1xil8LaPaavqE39pyyw2VvCZJZIiAw5CgZII6mhRbdgbSVzEorTuNDuU8ST6PbLvmjmeNA7qu4DPOSQOgzW1q/gLULO3tZbOLeGtFlufMuovkk53BeRkYA6Z+tKztzD62OSora8L6dbXt9cXGoqXs7C3e5ljBwZNvRc+5IqnqWoTazqAlMEUZwI4oLeMKqjPCgDr1+tO2wrlGiupsPA98slwdbtpraJLSSZGSRCd6rkBsZxn0ODVPw3qEAuF0rU7aKewvJQrkoPMiY8B1bqCOOOlPld7dQvpcwqKtapYtpmrXVjIdzW8rR7vXBxmibTbu3sYbyaErbzHEb5Hzf1rPmStd7l8rd7LYq0UUVRIUUUUAPf/Ux/jTKe/8AqY/xplABRRRQAUUUUAKqs7BUUsxOAAMk1vS2dyfB9vGLebeLxyV2HIG0c4rCjkeKRZImZHU5VlOCD6g1b/trVP8AoJXn/f8Ab/GsasZya5ehtSlCN+bqaXhe8ukvJYFuJliW3lYRhyFB2nnHrWNcXt1ebftdzNPt+75shbH0zTI55YpC8Uro7AgsrEEg9RTKcaaVRz7g6jdNQ9f0/wAgrpfCAM8OuWkfM0+myeWo6sQQcD3wK5qprO8uNPvIrqzlaGeJtyOvUGtl5mDL/hZ44/FmlvMQEF1Hkt0HzCup0nTtTtvE3iKa7t5442tbrdJIpCvnJGCevrXHarqR1a9+1PbW9vIygOLdNqu3dsepobWtUdQH1K8YKhQAzscKeCOvQ+lO+lvX8Qtrf0/ApVY0+5Sz1G3uZrdLlIpA7QyfdcA9DVeiknZ3G9dDrtXe38VK11Zaw6yxKWGnXzBNgA6REYU/TANcjRRSAs6c06apatZhWuFmQxBiAC2RjOfeu1k05NXutTln0rUdD1BYZZprhZSYGI5YNkcA+xNcDVybWNTuLb7PcajdywdPKedmX8icVV9LCtrcTTNKvdZvRaabAZ52BYICBwOpyeKrSxvDM8UqlXRirKexHUU+2urizmE1pPJBKOA8TlWH4io2YsxZiSScknvUjOq8Q2V1eaL4eltLaaeMWGwtFGWAIc5HHerekfZ5Ph6qSaTPqoj1BjLDbzGNkJQYY4UkjqK5W01jU7CIxWOo3dtGTkpDOyDP0BqK1v7uxkMlldT2zsMFoZChP4irurvz/wA7k2dl5f8ADHZTXEsvjPwzDNpb6b5HkpHFLcea5TfkZ4BH0PNRaNI8njTxCHYsHt7wNk/eHPWuR+13JuxdG4l+0Btwm3neD67uuaEvLmKaSaK4mSWUMJHVyGcN1BPfPele6+/8Qt+n4HTWdld6r8Ofs2lQyXMsOol54YRubaUAVsDkim+NopoIdChuv9dHpqK4JzghmGPw6Vj6NNp8Ekralc6hBlQENiFyfUHJHHSpfEOrW+pzWkdjHLHa2dstvF5xBdgCTuOOM80Sd1934KwR0f8AXe5p+KLO6udP0K4t7aaWFdKjDSIhZQQWyCR6VFI7D4XwKGIVtVbIz1/disWDV9StrVra21C6ht2BBijmZUOevAOKgNzObUWxmkMAfeIt52hsYzjpnHehu9/P/O40tvL/ACsdD4sJfTfDrscsdNUEnrgM2KPGgMk2lXacwz6bDsbsSowR9Qa56W5nnWNZ5pJFiXZGHckIvoM9B7VoW2vXMOjS6XNHDc2r5MazJkwMf4kPY/pQ3e/rf8/8xJWt6WIdEt5LvXrGCBSzvcIAB9RVnxXOlz4t1SWI7ka5fBHfBxUem69c6RZzxWEcMc82QbvZmVFIwVU9h+tZ8MgjnSR0WQKwYo3RsHoaTtoh92dFCpt/hnctL8v2rUUEWf4tqHJHtziuarR1fW7rWZYzcCOKKFdkNvCmyOJfRRWdQ9XcFoixp/8AyFLX/rsn/oQrS1HUJ7Dxbe3MLAusrp8/II5GD+FYysyMGQlWU5BBwQaWSR5ZGkldndjlmY5JPqTWMqalK72tY2jUcY2W90/uudFpvlP4SZXsZb7beEvHFIVK5UYY4ByOoqn4gld/sUcli1kIoMIjy72K5OM8Aj6GsyC5ntXL200kLEYLRuVP6UySR5pDJK7O7dWY5J/Gs40bVOb59f8AhjR1r0+T5dO9/U29Zt557DSZYYZJI1slBZFJAIJzzSRQy3fg0pbRtK8V7udUGSAUwDj0rLiv7yCEwwXc8cRzlEkIU59hTILme1cvbTSQsRgtG5U/pR7Kaja+zug9rHm5rdLP7rG7fo0N1oFvKNsscab0PVcvkA1X1HUJ7Dxbe3MLAusrp8/II5GD+FZDTStN5zSOZc7t5Y7s+uaSSR5ZGkldndjlmY5JPqTRGhb4tdH+LuEq1/h01X4KxrliPA4AJAN+cj1/d0a4S2naOzcn7JjJ9AxrJ86XyPJ8x/K3bvL3HbnpnHrQ80sqoskjusY2oGYkKPQelUqTUk/Nv7yfarlt5W/G4yul8IAzw65aR8zT6bJ5ajqxBBwPfArmqms7y40+8iurOVoZ4m3I69Qa6F5nOyJVLMFUEseAAOta81tOfDdsghk3CdyRsORwKralqsmpagL0QQ2k2BuNsuwM2fvY7E+1Rf2pqH/P9c/9/m/xpdB9SyxI8KoMkA3ZyP8AgNGrEm204nk/ZgM/iazjLIY/LLsU3btueM+uPWh5XkCh3ZggwoJztHoKbA1NUglkt9Pkjjd0FooLKpIGCc1X0S3ku9esYIFLO9wgAH1FV0vLmKExR3EqRnqiuQD+FXdN1650iznisI4Y55sg3ezMqKRgqp7D9aaaUrktXViTxXOlz4t1SWI7ka5fBHfBxWRRRUJWVi3qzp9Cknbw3Pb3Wiy6npjXQctbyFZIpduM8A8Y9RioPFmmR6bNYGGS68qe2EiW94cyW4yRtPt6cCse01C8092awu57VmGGMMpQn8jUc9xNdTNNdTSTSt1eRizH8TVt3JSsWrnRtQs9MttQubZo7W6/1MhI+b8M5H41uR21xffDWOPT4pJ2i1JmnSJSxAKDaSB2965uS8uZreO3luJXhiz5cbOSqZ64HQUtpfXdhIXsbqa2dhgtDIUJH1FF1qg10Z01nbapaeLPDcOsyZcNCYYmPzxR7+FYY4+hqfRpHk8aeIg7Mwkt7wNk/eHPWuR+13JuxdG4l+0Btwm3nfu9d3XNCXlzFNJNFcTJLKGEjq5DOG6gnvnvRfT7/wARW/T8DqNK06FPB8V5PZX+rrNdMptLadkjhIA+ZgAfmOfTpV/xAgTxB4TC2z2uIoR5Mj72jxL90nAyRXF2uo3tgGFjeXFsH+95MrJu+uDzTXvruR4XkupnaD/VM0hJj5z8vpzzxVcyun6fgLl0fzNzVdYu9K8eard2zBpPtE0REgyCpJXBH0/lVrR106x8DXMurvdRpqV0sSG1RWYrGNx+8Rxk/pXKSyyTzPLO7SSOxZ3c5LE9ST3NOe5nkt44JJpHhiz5cbOSqZ64HbNRF2Vinq7nReN1huNQs9VtC7Qahao4aQAMWX5GyBkZ+UfnUniizurnT9CuLe2mlhXSow0iIWUEFsgkelcy9zPJBHDJNI8UOfLjZiVTPJwO2ang1fUra1a2ttQuobdgQYo5mVDnrwDihtWfrf8AP/MErWNrwgDPDrlpHzNPpsnlqOrEEHA98Cs/ws8cfizS3mICC6jyW6D5hVCzvLjT7yK6s5Whnibcjr1BqfVdSOrXv2p7a3t5GUBxbptV27tj1NUpWal2/wCHE1dNdzsdJ07U7bxN4imu7eeONrW63SSKQr5yRgnr61x+iW8l3r1jBApZ3uEAA+opG1rVHUB9SvGCoUAM7HCngjr0PpU2m69c6RZzxWEcMc82QbvZmVFIwVU9h+tKLSafYGm0/Mk8Vzpc+LdUliO5GuXwR3wcVSmjvl06CScy/ZGJ8nc3y574FVa0dU1FLyGzggDLDawBAGGCW/iP4msXdOKSNo2tJtmdRRRWhmFFFFAD3/1Mf40ynv8A6mP8aZQAUUUUAFFFFABRSqrOwVAWZjgADJJqd7C8jiaSS0nWNDhnaMgKfQmk5JbjSb2K9FSwW0905S2gkmYDJWNCx/ShbW4a5+zrBIZ848oId2fp1o5kHK7XIqK1rzw7f2sNu6W1xL5sIkcCBv3R5+U/l7VmeTL5HneW/lFtvmbTtz1xn1qY1ITV4sqVOcHaSGUVIbeZfL3QyDzRmPKn5+3HrUj2F5HE0klpOsaHDO0ZAU+hNPmXcXLLsV6Kmis7q4UNBbTSqzbQUjJBOM4474qOWKSGQxzI0br1Vxgj8KfMr2FZ2uNooopiCipIIJLm4jghGZJGCqCccmtLUtFjsNNiuor5LotKYnEafKrAZOGzz+VVyu3MS5JOxk0U+GCW4lEdvE8sh6KilifwFNIKsVYEEHBB7VF1exdna4lFFFMQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQA9/8AUx/jTKe/+pj/ABplABRRRQAUUUUAWNP/AOQpa/8AXZP/AEIVtX+uXVl4nvS7tNBvkiMDOdhXkYx+tc/HI0UqSRnDowZTjoRTrieS6uJJ523SSMWZsAZJ+lYzpKc7y2tY2hVcIWjvdP7rnQabLb2/hJnnmu4Q94Vd7QDccKMA5I461V1y+WaSymtheI6Q7fPuF2vIMnDZB59M1QstTvNO3/Y5zGH+8pAZT+ByKiu7y4vpzNdytLIRjJ7D0HpWUaDVVye39f1uaOsnS5Fv/wAG/wDWhsa5eXK2mlbbiYb7JS2HPzHJ6+tREFvBK45235z7fJVWHW9Qgs/ssdx+5AICtGrYB6gEjIqOx1S804v9jnMYf7y4DA/gcihUpqNklo7+v4f5jdWLld32t+Fu/wDka14Cr+HVbhvKQ4PoZOKW/wBcurLxPel3aaDfJEYGc7CvIxj9axZr+6ub0Xc8zSTggh27Y6cdPwqO4nkuriSedt0kjFmbAGSfpRHD7c+uj/F3B17fBpt+CsbMVzNbeCs28rRFr4qShwcbM4zTPEDtPaaVPKS8slr87nq2GOM1l/a5jYi03/uBJ5uzA+9jGc9elE93PcRQxzPuSBdkYwBtGc4qlRanzebf4EusnDl8rfjchrQ0u70y2WT+1NMkvixGwpdGLZ69FOaz6K6r2OYs381rPeNJp9q1pAQMRNL5hHHPzYFaE3/Ik2v/AF+v/wCgismGZ7edJoiA6MGUkA4P0Nah8Vaw0ew3KbfT7PH/APE1cXFRafX/ADTM5KTat0KOn3V3Z3iy6ezLPghdi7ifXioZXeSZ3mJMjMS5PUnvWjoN9Dpt1PdytiWOBhAuCcueB/WswksxLHJJySe9c6+N6fM6H/DWvXYSiiitDMKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAUsSoB6DpSVLbPDHdRvdQmeFWBeMPsLj0zzitbxJplnYmwudNEqQX1sJxFKwZozkjGeMjin0uHWxiUUUUgCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArsdWOlr4b0Ua2l296bHMDWjKqCPcdgfdnJz1xiuTtvIN1H9sMgg3DzDEAW298Z4zWn4h1a11JrKDT4pY7WxtxBGZyC78k5OOB16U/si6mPRRRSGFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAf//Z)**

**MSD Radix sort to sort Hindi Words**

For this experiment we are providing set of 25K, 1M,2M Hindi words and applying MSD radix sort to sort them. Hindi words first need to convert into corresponding ASCII values and then sorted using Unicode.

Unicode:

Unicode is a standard maintained by Unicode Consortium, formally the Unicode standard, is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world’s writing system.

ASCII value for Hindi characters starts from 2309 and extended up to 2416.

Table

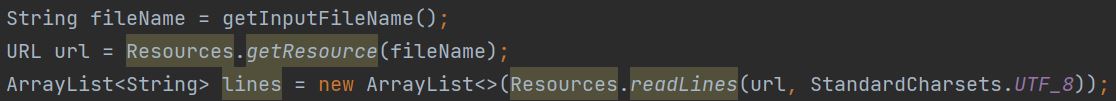
Description automatically generated

Table

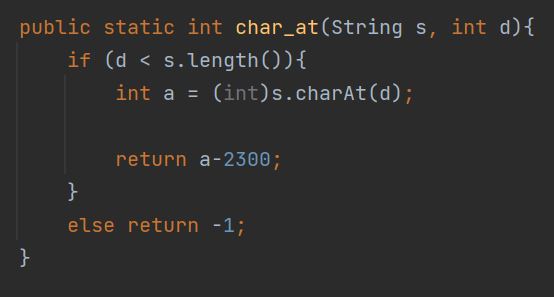
Description automatically generated

**Steps for implementation:**

1. Below block of code converts hind string to respective Unicode values.

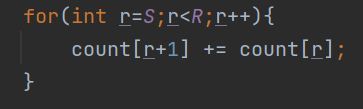


2. Once string is converted into Unicode MSD Radix sort is applied to get string sorted. Create an empty array count with the length of R. R is set to 400 considering Unicode value for Hindi words lies between 2300 and 2416. **To get the value in the range 0 to 400 we are subtracting 2300 from character’s index value.** Initially value of d is set to 0.

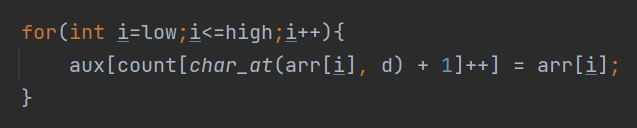


Get the value of char\_at for first character of all the strings provided as an input. After each iteration value of d will be increased. If this is the end of the string, then return -1. Increment value by 1 in count array for each matching key.

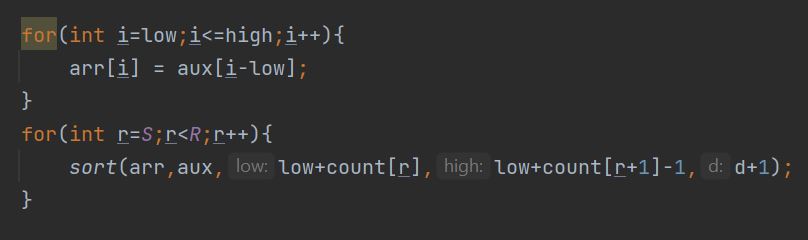
1. After reaching the end of the array of the string, run another iteration to count occurrence of each character at given value d. This let us know how many strings are there before given character.



1. Next iteration is for placing the each at correct position given by count array value. For this purpose, create an empty aux array.



1. Final iteration is needed to copy string from aux to original array for further sorting arrays recursively for each bucket created until all characters are sorted.



**Test Cases**

Text

Description automatically generated

**Benchmarking**

* We have implemented timer method to perform benchmarking of different sorting algorithms like MSD Radix sort, LSD Radix sort, Quick dual pivot Sort and Husky sort
* We have taken the input array size starting from 250K, 500K, 1M, 2M and 4M
* Input data might have some duplicate words
* Benchmarking is considered by taking the mean of 10 runs for each algorithm

Chart, bar chart

Description automatically generated

Fig Representing bar graph of multiple algorithms for Hindi words

Chart, line chart

Description automatically generated

Fig Representing line graph of multiple algorithms for Hindi words

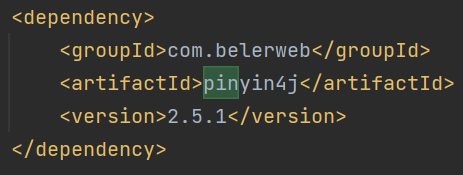
**MSD Radix sort to sort Chinese Words**

For this experiment we are providing set of 25K, 1M,2M Chinese words and applying MSD radix sort to sort them. We have taken pinyin order of Chinese words into consideration while sorting.

**Pinyin:**

Pinyin is a used to Romanize the Chinese characters for learning to speak the language faster. For this purpose, we have used Pinyin4j java library to convert Chinese character into Pinyin systems.

**Dependency for Pinyin4j:**

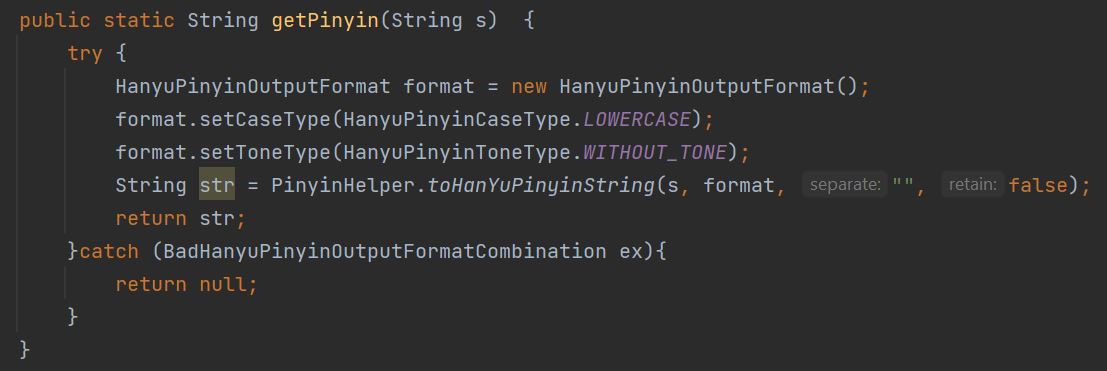


Steps for Implementation:

1. We have used dependency ibm.icu to get the collator instance in simplified Chinese.







A common function getPinyin() is implemented which converts string into equivalent Unicode characters.

We have implemented a class named Pair<K,V> for storing key and value where key is pinyin Chinese word and value is converted Unicode equivalent using function getPinyin() as stated above.

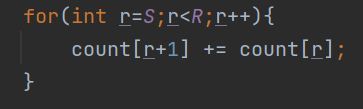
Graphical user interface, text, application, chat or text message

Description automatically generated

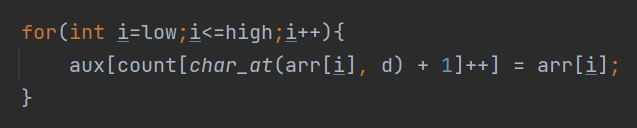
We are sorting these pair based on their value and reflecting the output of their respective keys.

Get the value of char\_at for first character of all the strings provided as an input. After each iteration value of d will be increased. If this is the end of the string, then return -1. Increment value by 1 in count array for each matching key.

1. After reaching the end of the array of the string, run another iteration to count occurrence of each character at given value d. This let us know how many string are there before given character.



1. Next iteration is for placing the each at correct position given by count array value. For this purpose, create an empty aux array.



1. Iteration is needed to copy string from aux to original array for further sorting arrays recursively for each bucket created until all characters are sorted.

**Benchmarking**

* We have implemented timer method to perform benchmarking of different sorting algorithms like MSD Radix sort, LSD Radix sort, Quick dual pivot Sort and Husky sort
* We have taken the input array size starting from 250K, 500K, 1M, 2M and 4M
* Input data might have some duplicate words
* Benchmarking is considered by taking the mean of 10 runs for each algorithm

Chart, bar chart

Description automatically generated

Fig Representing bar graph of multiple algorithms for Chinese words

Chart, line chart

Description automatically generated

Fig Representing line graph of multiple algorithms for Chinese words

**Test Cases**

Text

Description automatically generated

**Performance Evaluation:**

**Loop Invariant**: After the ith iteration of the loop, the elements are sorted by their last i digits

**Time Complexity**: n is the number of strings to sort and R is the Radix. MSD Radix sort best case time complexity is O(n) and the worst-case time complexity is O(n\*d).

**Space Complexity**: O(N + dB), where d = length of the longest string and B = size of radix (B=10 possible numbers or B=256 characters or B=2 for Binary).

Where d is the average length of the strings.

**Comparison with other Algorithms**

|  |  |  |
| --- | --- | --- |
| **Sorting Algorithm** | **Space Complexity** | **Time Complexity** |
| Tim Sort | O(n) | O(nlogn) |
| Quick Dual | O(n^2) | O(nlogn) |
| Husky | O(n ln n) | O(n ln n) |
| LSD Radix | O(n+d) | O(n\*d) |
| MSD Radix | O(n + db) | O(n\*d) |

**Conclusion**

**For Hindi input**

As the input size increase the time measurement for LSD radix has significantly increases, followed by Tim sort. The most efficient algorithm as input size increases are MSD and Husky Sort. LSD performance best when the length of the string in the array are equal. If the string length varies, LSD sort performance significantly decreases.

**For Chinese input**

Tim and quick sort complexity increase with the input size. Husky Sort is the most efficient followed by MSD Radix and LSD Radix.

# **Appendix – A – References**

|  |  |
| --- | --- |
| **Titles** | **Description/Link** |
| MSD and LSD | https://www.coursera.org/lecture/algorithms-part2/msd-radix-sort-gFxwG |
| Pinyin | https://en.wikipedia.org/wiki/Pinyin |
| Unicode | https://unicode.org/charts/PDF/U0900.pdf |
| ASCII | https://sites.psu.edu/symbolcodes/languages/southasia/devanagari/devanagarichart/ |
| Performance Evaluation | <https://algs4.cs.princeton.edu/home/> |
| Husky Sort | https://github.com/rchillyard/The-repository-formerly-known-as |
| Quick Dual | https://algs4.cs.princeton.edu/23quicksort/QuickDualPivot.java.html |
| Tim Sort | https://www.javatpoint.com/tim-sort |