INFO 6205 - Program Structures & Algorithms
Final Project

Member Name	**NUID**	**Email ID**
Sutiksh Verma	002122052	verma.su@northeastern.edu
Daisy Quadros	001568823	quadros.d@northeastern.edu
Riddhi Bhatti	001502713	bhatti.r@northeastern.edu

1. **Abstract & Methodology of MSD radix sort:** We used Most Significant Digit(MSD) Radix Sort with Dual Pivot QuickSort to sort our data of Hindi words and compared this approach with Least Significant Digit (LSD) String Sort, Merge Husky Sort, Dual Pivot QuickSort, and Tim Sort to find which algorithm performs better with sorting Hindi words. We compared all the algorithms with different amounts of data, i.e., 500K, 1M, 2M, and 4M Hindi Words. In this approach, we found out that Tim Sort performs better for sorting 1M Hindi words and gives the lowest relative time. We decided to use Hindi as our language of choice, as we will be able to check if the words are getting sorted properly or not and Hindi comes from Devanagari Scriptures which follow Unicode, thus Hindi follows Unicode as well. To collect our data we did the following; we referred to Hindi newspapers, magazines, dictionaries, and scraped the words from these sources, which gave us around 1.3M words, and for the rest of the data we took this data multiple times to increase the sample size. After getting the data, we ran a python script to remove any English characters, punctuations, and blank spaces to clean up the data. To analyze the data, we benchmarked each run of sorting in nanoseconds and averaged it to milliseconds to give us a precise and accurate time. The result is the average of 50 runs performed by a specific algorithm. After getting results from each algorithm, we compared all of them and chose the one which had the lowest time taken to sort.

2. **Keywords**: Most Significant Digit (MSD) Radix Sort, MSD Radix Exchange Sort, Radix Sort

3. **Abstract of relevant research papers**

    a. **Sequential & Parallel hybrid approach for non-recursive most significant digit radix sort:** The goal of this paper is to come up with an alternate sequential MSD radix sort algorithm that performs better than the traditional recursive solution. The second goal is to implement sequential and parallel versions of a hybrid combination of MSD radix sort with QuickSort algorithm. Instead of the recursive version, an efficient sequential non-recursive MSD radix sort version is implemented. The results presented show that the non-recursive MSD radix sort is significantly superior to the traditional recursive MSD radix sort. In addition, implement two versions of the sequential and corresponding parallel hybrid MSD radix versions using MSD non-recursive radix sort and efficient quicksort methods. Performance benchmarks for the sequential and parallel hybrid MSD radix sort algorithms show that both parallel hybrid versions operate faster than the corresponding sequential versions and significantly improve performance over quicksort.

    b. **Implementing Radixsort:** Presents and evaluates some new optimization and implementation techniques for String Sorting. In particular, the recently published radix sort algorithm Forward Radixsort obviously has good worst-case behavior. Our experimental results show that the radix is Sorting is much faster than comparison-based sorting (often more than twice as fast). Method. This also applies to small input sequences. It also shows that it is possible to Implement radix sort by worst-case execution time without forgoing the average case Power. Our implementation is competitive with the best strings ever published Sorting algorithm. Code, test data, and test results are available on the World Wide Web. They investigated the performance of some string sorting algorithms. Proven radix sort algorithm is much faster than the more commonly used comparison-based algorithms. On average, Adaptive Radixsort was the fastest algorithm. However, the forward radix sort is a bit slow and guaranteed to work in the worst case. Forward radix sort is Large alphabets such as the recently proposed 16-bit Unicode character set

    c. **A Novel Approach of Implementing Radix Sort by Avoiding Zero Bits Based on Divide and Conquer Technique:** Abstract Sorting plays an important role in theoretical calculations and facilitates data analysis. Existing algorithms work well in terms of time and space consumption, but there are new technologies still manufactured. Efficient non-comparative individual digit clustering, integer sorting algorithms, and radix sorting appear to be using $\boldsymbol{O}\ (\boldsymbol{n})$. However, in general, you need $\boldsymbol{O}\ (\boldsymbol{n}\mathbf{log}\ (\boldsymbol{n}))$ as the best performance comparison based algorithms do. Introducing an exciting technique based on radix sort to achieve a better performance algorithm than existing algorithms. This technique divides a set of decimal integers into two groups with respect to the co-located bits of the binary representation. This step is repeated within each subgroup and the process continues until the list is sorted. The most significant non-zero bit with the maximum value is used as the pivot bit for the partition of each step to prevent the leading zero bit from passing through. This will take $O(cn)$ time. Here, c is significantly smaller than $\log n$.

d. **A Fast Radix Sort:** Almost all computers regularly sort data. Many different sort algorithms have therefore been proposed, and the properties of these algorithms studied in great detail. It is known that no sort algorithm based on key comparisons can sort N keys in less than O(N log N) operations, and that many perform 0(N2) operations in the worst case. The radix sort has the attractive feature that it can sort N keys in O(N) operations, and it is therefore natural to consider methods of implementing such a sort efficiently. In this paper one efficient implementation of a radix sort is presented, and the performance of this algorithm compared with that of Quicksort. Empirical results are presented which suggest that this implementation of a radix sort is significantly faster than Quicksort, and that it therefore has wide applicability.

4. References & Citations

[1] A Novel Approach of Implementing Radix Sort by Avoiding Zero Bits Based on Divide and Conquer Technique - Link to the research paper
[2] Sequential & Parallel Hybrid Approach for Non-Recursive Most Significant Digit Radix Sort - Link to the research paper
[3] Implementing Radix Sort - Link to the research paper
[4] Fast Radix Sort - Link to the research paper
[4] Dual pivot quicksort
https://algs4.cs.princeton.edu/23quicksort/QuickDualPivot.java.html
[5] MSD Radix sort - Algorithms 4th Edition by Robert Sedgewick, Kevin Wayne.pdf

5. **Conclusion** : The below order is based on the time it took to sort the array. Tim sort performs best for the input array size ranging between 100 to 4M

Tim < Husky < Dual pivot quick < LSD radix < MSD radix