**RADIX SORT UNRAVELED**

**Abstract:** This research focuses on the usage of radix sort in daily applications. Radix sort is a fascinating algorithm with many intricacies to explore and implement. The goal of this research is to thoroughly investigate Radix Sort, clarifying its underlying ideas, practical applications, and performance attributes. Radix sort is appropriate for sorting because, under certain circumstances, it can handle a wide range of data formats and achieve linear time complexity by sorting items efficiently by processing individual digits or characters from the least significant to the most significant. A practical issue that can be resolved with this technique is explored and the algorithm paradigm is described. In conclusion, this article offers a thorough review of radix sort.

**Introduction:** Radix sort, first conceptualized by Herman Hollerith in 1887, became a practical sorting method for punched cards by 1923. It operates by processing individual digits or characters of numbers or strings from least to most significant, sorting elements based on their radix (the base of the numbering system used). Radix sort handles elements of varying lengths independently and can be more efficient than comparison-based algorithms in certain scenarios, despite its linear time complexity and space complexity . It can also maintain stability, preserving the relative order of elements with equal keys, which is important for specific applications.

**Rational:**  Radix sort is preferred for sorting hospital records due to its efficiency, stability, adaptability, and space complexity. It excels in handling variable-length keys, such as patient IDs and medical codes, by sorting them digit by digit or character by character. Radix sort requires less additional space compared to other sorting algorithms like counting sort and bucket sort, making it more memory efficient, especially in scenarios with large datasets. Its stable sorting nature preserves the relative order of equal elements, which is crucial for maintaining the integrity of medical records. Additionally, radix sort can be easily adapted to sort hospital records based on different criteria, such as patient name, age, admission date, or medical condition. Overall, the combination of these factors makes radix sort a suitable and preferred choice for sorting hospital records efficiently and effectively

**Substitute Use of the Algorithm *-*** Radix sort finds applications beyond sorting hospital records and can be effectively used in various scenarios where keys are represented in a fixed-length format or can be processed digit by digit. One such application is in sorting large volumes of numerical data, particularly in data analysis and scientific computing.

In data analysis tasks, datasets often contain numerical values, such as sensor readings, financial transactions, or scientific measurements. Radix sort can efficiently sort these datasets based on numerical values, providing valuable insights and facilitating further analysis.

Runtime Complexity:

* Radix sort has a linear time complexity of , where d is the maximum number of digits in the numerical values, n is the number of elements to be sorted, and k is the range of the input.
* For numerical data with a fixed number of digits or a limited range, radix sort can achieve optimal performance, often outperforming comparison-based sorting algorithms like quicksort or merge sort.

Space Complexity:

* Radix sort typically has a space complexity of , where n is the number of elements to be sorted and k is the range of the input.
* In the context of sorting numerical data, the space complexity remains efficient and scales well with the size of the dataset.

In conclusion, radix sort's efficiency and scalability make it a valuable algorithm for sorting numerical data in various applications, including data analysis, scientific computing, and financial analysis. Its linear runtime complexity and space efficiency make it well-suited for handling large volumes of numerical data effectively.

**Methodology:**

Below explained how the functions radix sort and count sort works in the code

Radix Sort Methodology:

The `radix\_sort\_Patient\_ID` and `radix\_sort\_Patient\_age` methods utilize counting sort (`count\_sort\_numbers`) to numerically sort records by patient IDs or ages. They determine the maximum value of the attribute (`maximum\_ID` or `maximum\_age`) to set the number of sorting iterations per digit. During each iteration, records are categorized and rearranged based on the current digit's value using counting sort logic. For string sorting, the `radix\_sort\_string` method normalizes string attributes for consistent length and applies counting sort (`count\_sort\_strings`) to sort records lexicographically. This iterative process continues until all character positions are sorted, resulting in a fully sorted array of records based on string attributes, demonstrating the efficiency of radix sort in managing and sorting hospital records with stability and predictability.

Count sort methodology:

The counting sort methodology involves iterating through an array of records (`arr`) and categorizing them based on a specific digit value (`arr\_index = arr[i][index] // exp % 10`). Using a counting array (`arr\_count`), it tallies the occurrences of each digit and organizes records in the output array (`output`) according to their sorted values. After sorting all digits, the original array (`arr`) is updated with the sorted records from `output`. The `count\_sort\_strings` function similarly normalizes strings to a uniform length before categorizing records based on characters at specific positions (`char\_val = ord(arr[i][index][char\_index])`). It counts occurrences of each character value and sorts records into the output array (`output`) based on these counts. Finally, the original array (`arr`) is updated with the sorted records from `output` after sorting all characters for the current position.

These methodologies collectively enable the radix sort algorithm to efficiently manage and sort hospital records based on various attribute types (numerical or string-based) within the provided hospital record management system, leveraging the stability and predictability of radix sort for sorting fixed-length keys or strings effectively.

The radix sort methodology utilizes counting sort (`count\_sort\_numbers`) within the `radix\_sort\_Patient\_ID` and `radix\_sort\_Patient\_age` methods to sort patient records numerically by IDs or ages. These methods calculate the maximum value (`maximum\_ID` or `maximum\_age`) to determine sorting iterations per digit, categorizing and rearranging records based on digit values using counting sort logic. For string sorting, the `radix\_sort\_string` method standardizes string attributes for consistent length and applies counting sort (`count\_sort\_strings`) to achieve lexicographical sorting. This iterative process continues until all character positions are sorted, resulting in efficiently managed and predictable hospital record sorting.

In the counting sort methodology, records in an array (`arr`) are sorted by specific digit values (`arr\_index = arr[i][index] // exp % 10`) using a counting array (`arr\_count`) to tally occurrences and organize records in the output array (`output`). After sorting all digits, the original array (`arr`) is updated with the sorted records from `output`. The `count\_sort\_strings` function normalizes strings and categorizes records by characters (`char\_val = ord(arr[i][index][char\_index])`), counting character occurrences to sort records into `output` based on these counts. Finally, the original array (`arr`) is updated with sorted records after processing all characters for the current position, demonstrating an efficient sorting method for various record attributes.

The radix sort methodology employs counting sort (`count\_sort\_numbers`) in `radix\_sort\_Patient\_ID` and `radix\_sort\_Patient\_age` to numerically sort patient records by IDs or ages. These methods determine sorting iterations per digit based on the maximum value (`maximum\_ID` or `maximum\_age`), categorizing records using counting sort logic. For string sorting, `radix\_sort\_string` normalizes string attributes for consistent length and uses `count\_sort\_strings` to achieve lexicographical sorting. In the counting sort approach, records are sorted by specific digit values with a counting array (`arr\_count`) tallying occurrences. After sorting digits, the original array (`arr`) is updated. `count\_sort\_strings` normalizes strings, categorizes by characters, and sorts based on counts into `output`. This process efficiently manages and predicts hospital record sorting

**Alternative solution:** Quick sort is a versatile and efficient sorting algorithm that can be used to sort hospital records based on various attributes, including numeric and string fields. Unlike radix sort, quick sort does not require specific properties of the data like fixed-length keys and can handle diverse data types effectively. With an average time complexity of quick sort is often faster than radix sort for larger datasets, making it suitable for scenarios where performance is critical. Additionally, quick sort can be implemented as an in-place sorting algorithm, minimizing additional memory usage compared to radix sort, which may require extra storage proportional to the range of key values. Overall, quick sort's adaptability, average-case performance, and efficient use of memory make it a practical choice for sorting hospital records, particularly when dealing with diverse or random data types.

Hospital records can be sorted using quick sort, an adaptable and effective sorting algorithm, based on a variety of criteria, including string and numeric fields. Quick sort can effectively handle a variety of data types and is not limited by specific properties of the data, such as fixed-length keys, as radix sort is. Quick sort is appropriate for situations where performance is crucial because, on average, it is faster than radix sort for larger datasets, with an O(n log n) time complexity. Furthermore, quick sort is an in-place sorting algorithm that uses less memory than radix sort, which might need extra storage in proportion to the range of key values.