Radix sort: No comparisons required

Sorting (arranging data in a particular sequence or order) is a very important operation in computer science, and as such, it is very rare to talk about computer algorithms without mentioning sorting algorithms. Practically speaking, there are so many ways in which data can be sorted, which is why so many sorting algorithms exist — merge sort, quicksort, insertion sort, heap sort, etc.

The radix sort algorithm starts the grouping into buckets with either the least or most significant digit of each item of the data set, and then collapses the items in the buckets into a new data set containing items that are sorted based on the digit at the start position — this is the first iteration. The process is repeated for the other digits in each item until the data set is completely sorted.

Uses 10 buckets

On each iteration, 10 buckets are used because we are dealing with decimal (base 10) numbers. The buckets map to their corresponding digits in sequential order (0–9). Therefore, the number of buckets to be used depends on the radix (base) of the number system used for the items.

It is also important to notice that some buckets are empty for some iterations, which means that memory was allocated but never used to store anything — good optimization starting point.

- Take the least significant digit (or group of bits, both being examples of radices) of each key.
- Group the keys based on that digit, but otherwise keep the original order of keys. (This is what makes the LSD radix sort a stable sort).
- Repeat the grouping process with each more significant digit.

Original, unsorted list:

170, 45, 75, 90, 802, 2, 24, 66

- Sorting by least significant digit (1s place) gives:

  170, 90, 802, 2, 24, 45, 75, 66

**** Don't change the order of the key values in the list

Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.

- Sorting by next digit (10s place) gives:

  802, 2, 24, 45, 66, 170, 75, 90

- Sorting by most significant digit (100s place) gives:

  2, 24, 45, 66, 75, 90, 170, 802

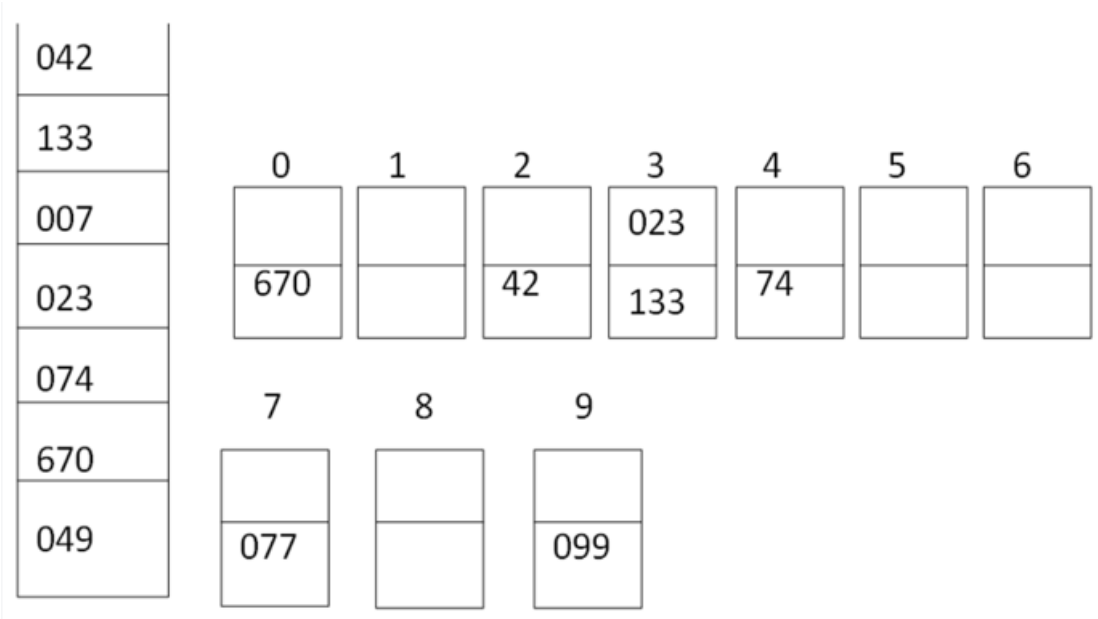
Iterative version using queues

A simple version of an LSD radix sort can be achieved using queues as buckets. The following process is repeated for a number of times equal to the length of the longest key:

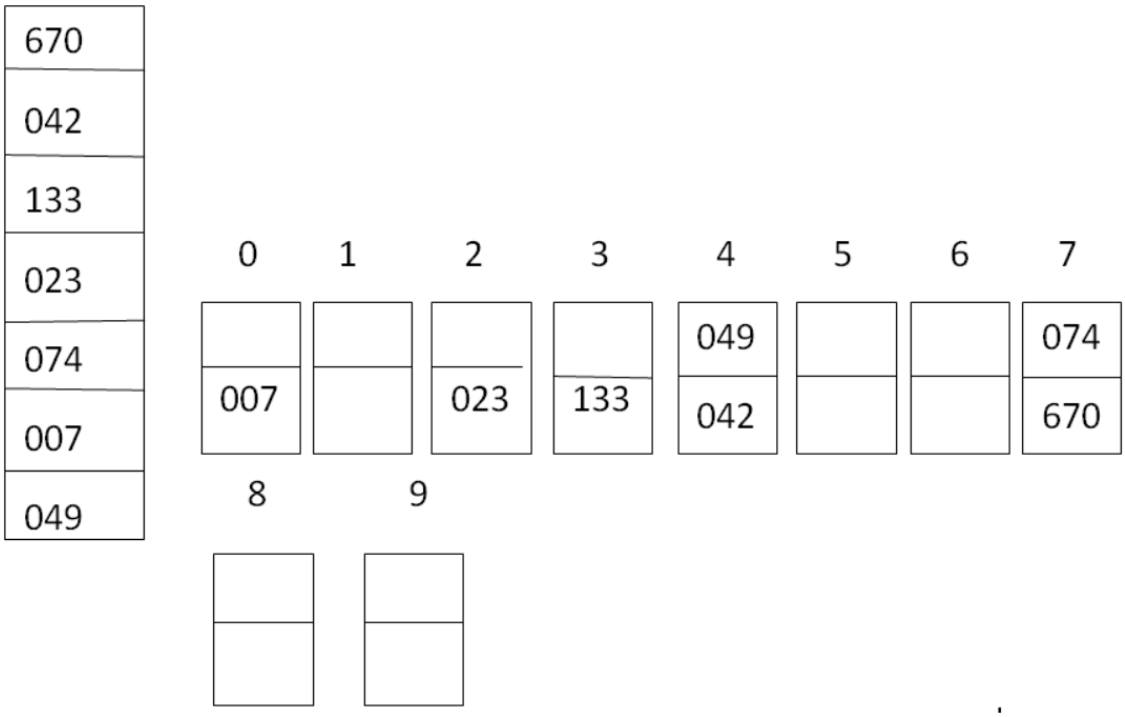
To illustrate the radix sort consider the following array with 7 elements

42, 133, 7, 23, 74, 670, 49

In this array, the biggest elements are 670 and the number of digits is 3, so 3 passes are required to sort the array. Read the elements and compare the first position (2 is in the first position of 42) digits with the digits of the buckets and place it

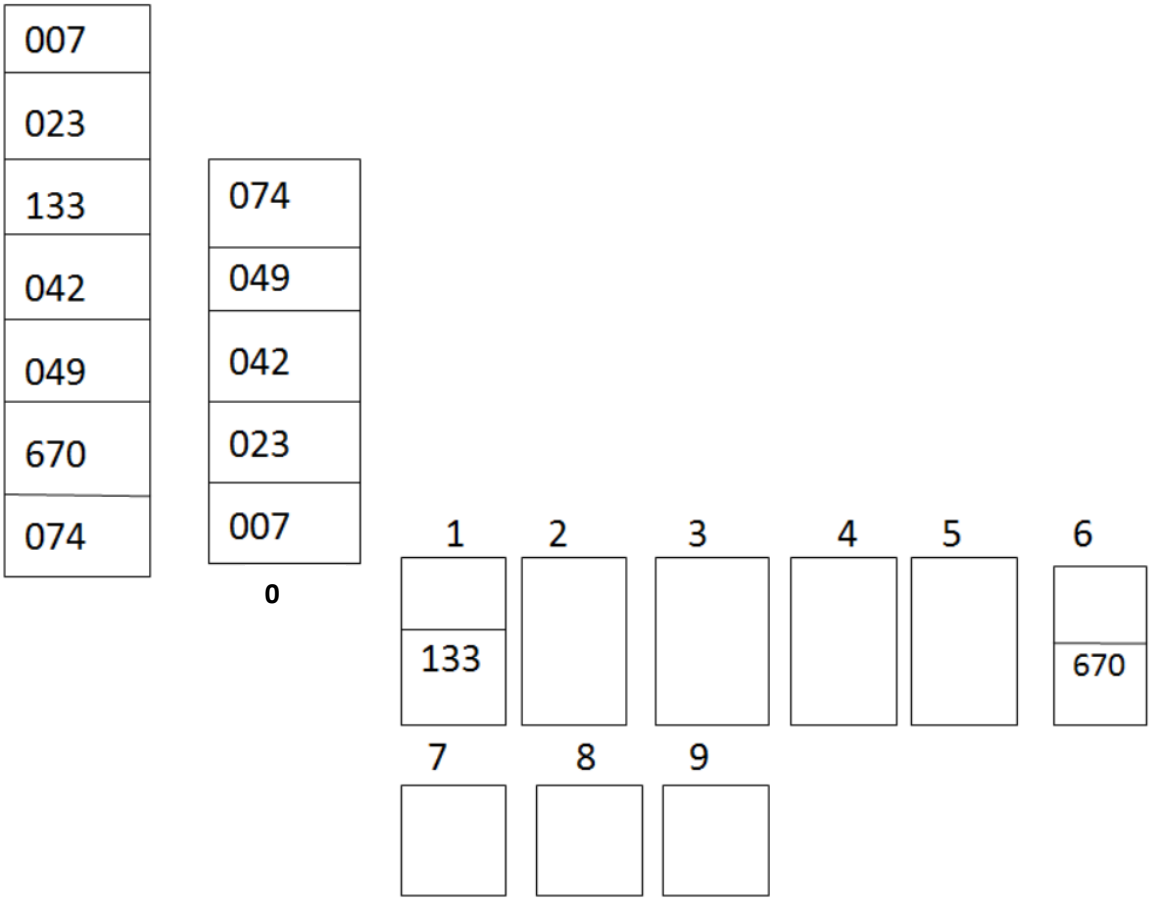


Now read the elements from left to right and the bottom to the top of the bucket and place it in an array for the next pass. Read the array elements and compare the second position (4 is in the second position of the elements 042) digit with a number of the bucket and place it.



Again read the element from left to right and from bottom to top to get an array for the third pass. (0 is in the first position of 042) compare the

third position digit in each element with the budget digit and place it wherever it matches.



Now the sorted array is 7 , 23, 42, 49, 74, 133, 670.

Complexity Analysis of Radix Sort:

Time Complexity:

- Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping the keys by the individual digits which share the same significant position and value. It has a time complexity of **O(d * (n + b))**, where d is the number of digits, n is the number of elements, and b is the base of the number system being used.
- In practical implementations, radix sort is often faster than other comparison-based sorting algorithms, such as quicksort or merge sort, for large datasets, especially when the keys have many digits. However, its time complexity grows linearly with the number of digits, and so it is not as efficient for small datasets.