Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value. Radix sort can be implemented to start at either the most significant digit (MSD) or least significant digit (LSD).

LSD radix sorts typically use the following sorting order: short keys come before longer keys, and the keys of the same length are sorted lexicographically.

MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations.

We introduce LSD here:

Each key is first figuratively dropped into one level of buckets corresponding to the value of the rightmost digit. Each bucket preserves the original order of the keys as the keys are dropped into the bucket. There is a one-to-one correspondence between the buckets and the values that can be represented by a digit plus one bucket for the empty digit, which signifies that the string is exhausted and which is not required if all strings are of same length. Then the process repeats with the next neighbouring more significant digit until there are no more digits to process.

1. Take the least significant digit (or group of bits, both being examples of radices) of each key.
2. 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).
3. Repeat the grouping process with each more significant digit.

The sort in step 2 is usually done using bucket sort or counting sort, which are efficient in this case since there are usually only a small number of digits.

```
def radix_sort(array, base = 10):
    def list_to_buckets(array, base, iteration):
        buckets = [[] for x in range(base)]
        for number in array:
            # isolate the base-digit from the number
            digit = (number // (base ** iteration)) %
base
            # drop the number into the correct bucket
            buckets[digit].append(number)
        return buckets
    def buckets_to_list(buckets):
        numbers = []
        for bucket in buckets:
            # append the numbers in a bucket
            # sequentially to the returned array
            for number in bucket:
                numbers.append(number)
```

```
        return numbers

    maxval = max(array)

    it = 0
    # iterate, sorting the array by each base-digit
    while base ** it <= maxval:
        array = buckets_to_list(list_to_buckets(array,
base, it))
        it += 1
    return array
```

Complexity Analysis:

Let there be d digits in input integers. Radix Sort takes $O(d * (n + b))$ time where b is the base for representing numbers, for example, for decimal system, b is 10. If k is the maximum possible value, d would be $O(log_b k)$. So overall time complexity is $O((n + b) * log_b k)$.

If we have $log_2 n$ bits for every digit, the running time of Radix appears to be better than Quick Sort for a wide range of input numbers. The constant factors hidden in asymptotic notation are higher for Radix Sort and Quick-Sort uses hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.