Let's say we start out with an unsorted list, and after 3 steps we sort it out:

In each step of Radix Sort, there is a Counting Sort using each digit as the key:

| | | | | | |
|---|---|---|---|---|---|
| 53 | 89 | 150 | 36 | 633 | 233 |
| 150 | 53 | 633 | 233 | 36 | 89 |
| 633 | 233 | 36 | 150 | 53 | 89 |
| 36 | 53 | 89 | 150 | 233 | 633 |

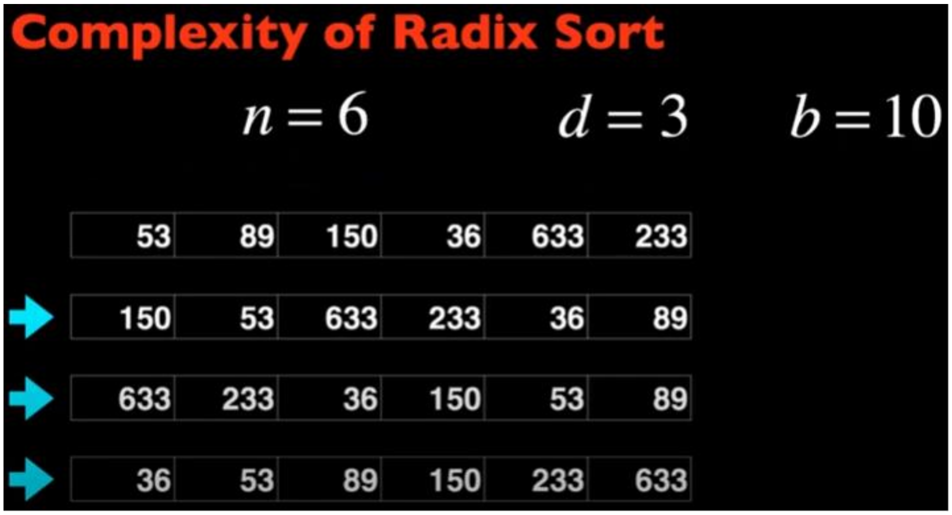
… to explain the variables:

**n** = 6 (ie, the number of elements in the list).

**d** = 3 (ie, the digits of numbers in our list).

**b** = 10 (ie, the base of 10 comprises of numbers 0 through 9).

Remember: the time complexity of a Counting Sort is **O(n+k)**

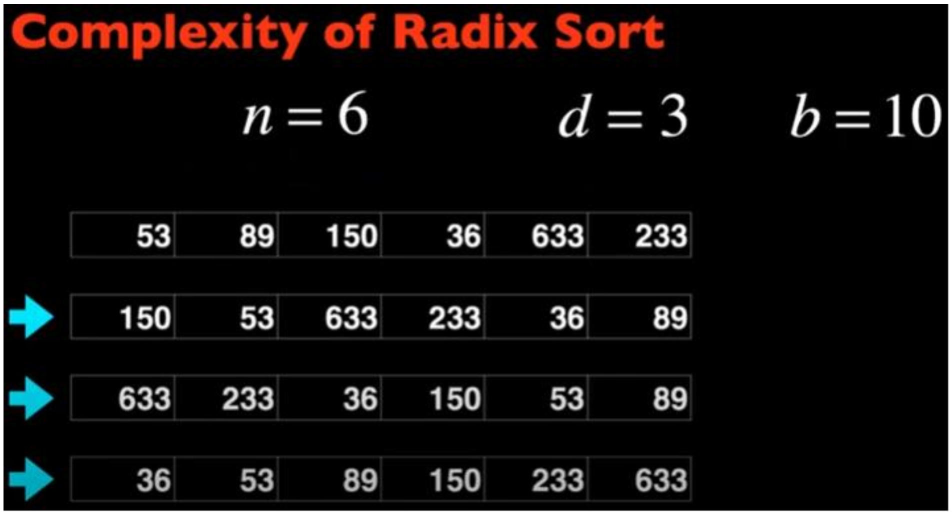



**k** is the range of keys for each number (ie, 0 through 9)… so **k** is the same as **b**…

...thus, each step takes **O(n+b)**, and in this case it was repeated 3 times (or **d** times).

… so to sum it all up, the time complexity for the algorithm of the Radix Sort will be: **O(d(n+b))**

Depending on the nature of input, a Radix Sort can outperform a Quick Sort or Merge Sort **O(n log n)**.




… for simplicity purpose, we choose base of 10 with digit of 3.

If b was larger than 10, like say, 1,000, it would imply a smaller d, meaning smaller digits, so we would need less sorting steps, so d would be comparatively smaller.

Likewise, if b was smaller, it would imply a larger d, meaning larger digits, so we may need more sorting steps, so d would be comparatively larger than...

…what does this all mean to us?... it means there is a trade-off between space and time.

...basically, if b is larger, it occupies more space, but it would save us time since less steps are required.... and likewise, if b was smaller and occupied less space, this would require more time for the extra steps.