Bucket Sort & Radix Sort:

<u>Bucket sort</u>:

simple Mailroom sort: allocate a sufficient number of "boxes", called buckets, and put each element in the corresponding bucket. This is called bucket sort.

If the elements are letters and need to be sorted according to states, we need 50 buckets but if it is according to zipcodes (5 digits), we need 100000 boxes.

Thus bucket sort works very well only for elements from a small, simple range known in advance.

Say if we have n elements ranging from 1 to m. We need m buckets and for each i, we put x_i in the bucket corresponding to its value in $O(n)$. At the end we scan the buckets in $O(m)$ to collect all elements.

$O(n+m)$ time and $O(m)$ space which is not good for $m \gg n$.

<u>Radix sort algorithm</u> (least significant digit radix sort)

1) take the least significant digit of each key (say the number of records to be sorted)

2) group the keys based on that digit, but otherwise keep the original order of key

3) Repeat the grouping process with each more significant digit.

* radix sort is a <u>stable sort</u>, i.e. it maintains the relative order of records with equal keys.

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

The running time of the algorithm is $O(nK)$ where K is the maximum key length, since we need $O(K)$ repetition of the loop (each step of the loop requires just a single pass over data

An example:

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

Sort by least significant digit.

17<u>0</u>, 9<u>0</u>, 80<u>2</u>, <u>2</u>, 2<u>4</u>, 4<u>5</u>, 7<u>5</u>, 6<u>6</u>

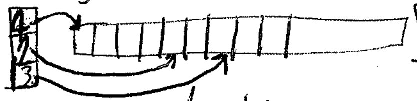
next

8<u>0</u>2, <u>2</u>, <u>2</u>4, <u>4</u>5, <u>6</u>6, 1<u>7</u>0, <u>7</u>5, <u>9</u>0

sort by last digit

2, 24, 45, 66, 75, 90, <u>1</u>70, <u>8</u>02

We can implement this by a linked list, however by first counting the number of keys that belong to each bucket before moving keys into these buckets, and store them in an array we can indeed implement this in an array as well (using pointers that we move after each insert)



Pointers are ~~integers.~~ indices

The proof is by induction: We know how to sort elements with $<K$ digits.

lexicographic sort (Most Significant digit sorts)
It can be used to sort keys (esp. strings) in lexicographic order (dictionary order), e.g.
b, ba, c, d, da.

lexicographic sort (most significant digit radix sort)

1. Take the most significant digit of each key
2. Sort the list of elements based on that digit, grouping elements with the same digit into one bucket.
3. Recursively sort each bucket, starting with the next digit to the right.
4. Concatenate (append) the buckets together in order.

Implementation:
Again a two-pass method can be used to first find out how big each bucket needs to be and then place each key into the appropriate bucket using pointers. A single-pass can also be used by link lists.

Example:

~~170, 045, 075, 090, 002, 026~~
~~a ab bb c cd cab~~

cab, ab, bb, cd, a, c, cda
<u>a</u>b, <u>a</u>, <u>b</u>b, <u>c</u>ab, <u>c</u>d, c, cda
a, a<u>b</u>, b<u>b</u>, c, c<u>a</u>b, c<u>d</u>, c<u>d</u>a
a, ab, bb, c, ca<u>b</u>, cd, cd<u>a</u>

we could sort the integer if we pad them with zeros
170, 045, 090, 002, 802, 066.
Again the running time is $O(nk)$

The current implementation of lexicographic sort is not, but there are some in-place implementation.