RADIX SORT

USING THE RADIX TO SORT

ALGORITHM RADIX SORT

```
1       RADIX-SORT( L )
2               for i = 1 to N
3                       use a stable sort to sort List L on digit i
```

STABLE SORT: COUNTING SORT

- A stable_sort preserves the relative order of the elements with equivalent values.
- PSEUDOCODE FOR COUNTING SORT

    Initialize counting array to all zeros.

    Count the number of times each radix value occurs in the input.

    Modify the counting array to give the number of values smaller than index

    Transfer numbers from input to output array at positions provided by counting array

COUNT SORT ALGORITHM

```
1    Algorithm countSort(arr,n,exp)
2        output = new int[ n ]
3        count = new int[ 10 ] = 0
4        for i = 0 to n
5            count[ (arr[i] / exp) % 10 ] += 1
6        for i = 1 to n
7            count[ i ] += count[ i - 1 ];
8        for i = n - 1 to 0
9            output[ count[(arr[i] / exp) % 10 ] - 1 ]  =  arr[i]
10           count[ (arr[i] / exp) % 10 ] -= 1
11       for i = 0 to n
12           arr[i] = output[i]
```

ORIGINAL LIST

List

{ 493, 812, 710, 195, 437, 582, 340, 385 }

*Notice that the numbers were added onto the list in the order that they were found

GROUP BY 1RADIX

[0]	710, 340
[1]	
[2]	812, 582
[3]	493
[4]	
[5]	715, 195, 385
[6]	
[7]	437
[8]	
[9]	

REORDER 1'S

List

{ 340, 710, 812, 582,493, 715, 195, 385, 437 }

GROUP BY 10'S RADIX

[0]	
[1]	710, 812, 715
[2]	
[3]	437
[4]	340
[5]	
[6]	
[7]	
[8]	582, 385
[9]	493, 195

REORDER 10'S

List

{ 710, 812, 715, 437, 340, 582, 385, 493, 195 }

GROUP BY 100'S RADIX

[0]	
[1]	195
[2]	
[3]	340, 385
[4]	437, 495
[5]	582
[6]	
[7]	710, 715
[8]	812
[9]	

REORDER 100'S

List

{ 195, 340, 385, 437, 493, 582, 710, 715, 812 }

ALGORITHM RADIX SORT

```
RADIX-SORT (L)
1       for i = 1 to N
2               use a [stable] sort to sort List L on digit i
```

In Radix sort, first sort the elements based on the last digit [the least significant digit]. These results are again sorted by second digit [the next to least significant digit]. Continue this process for all digits until we reach the most significant digits.

Use some stable sort to sort them by last digit. Then stable sort them by the second least significant digit, then by the third, etc. If we use Counting sort as the stable sort, the total time is

$$O(nd) \approx O(n)$$