Radix Sort

Carol Zander

The radix sort is very different from the other sorting algorithms that we have looked at so far, since **no comparisons are ever done between array elements**. The basic idea is to **group the elements into sets**, where the sets can be placed in a known order.

An example is sorting 3-digit numbers between 000 and 999. All numbers that start with a 0 can be placed in a group, all numbers starting with 1 in another, and so. After examining the first digit, the order to place the sets in is known, but the items must still be sorted within each set. This is performed using the same process on the second digit of the number (recursively or iteratively). In practice, this is usually done in the reverse order of the digits, so that the most important digit is examined last. However, for each step, this means to keep the elements with the same digit at the current position in the same order that they were in from the previous step to make sure that the numbers stay sorted correctly. Typically queues are used:

```
void radixSort(int a[], int size, int d) {      // d is the number of digits

      for (j = d; j > 0; j--){
            initialize 10 empty queues
            for (i = 0; i < size; i++){
                  k = jth digit of a[i]
                  insert a[i] into queue k
            }
            Replace the items in a with all the items in group 0, group 1, etc.
      }
}
```

The data to be sorted -- stored somewhere, possibly a data file:

```
 64   8   216   512   27   729   1   0   343   125   713   525   537   347   382
```

Pass 1 – sort by the least significant digit (put numbers into the queue, enqueue)

0	1	2	3	4	5	6	7	8	9
0	1	512	343	64	125	216	27	8	729
		382	713		525		537		
							347		

Pass 2 – sort by the next significant digit (dequeue, then enqueue; need counts of how many are in each queue because you may insert into a queue from a later pass before you have removed all the numbers from an earlier pass)

0	1	2	3	4	5	6	7	8	9
0	512	125	537	343		64		382	
1	713	525		347					
8	216	27							
		729							

Pass 3 – sort by the third significant digit

0	1	2	3	4	5	6	7	8	9
0	125	216	343		512		713		
1			347		525		729		
8			382		537				
27									
64									

Now empty out the queues starting at zero, and the numbers are sorted.