Sorting

Counting Sort

Counting sort can sort n integers in the range 0 to k in $O(n + k)$ time. Say the unsorted n integers are stored in array A. Counting sort works as follows:

1. Initialize counting array C, where $C[i]$ will contain the number of times the element i occurs in A. At initialization, $C[i]$ = 0 for all i. Also initialize sorted array B, where B will contain all the elements in A in sorted order.

2. Iterate through A, incrementing $C[i]$ by 1 for each value i seen in A. At the end of this step, $C[i]$ = number of times element i was found in A

3. Iterate through C, setting $C[i]$ = $C[i-1]$ + $C[i]$ for each i in C. At the end of this step, $C[i]$ = number of elements less than or equal to i that were found in A

4. Iterate through A backwards, placing element $A[i]$ into $B[C[A[i]]]$ and decrementing $C[A[i]]$ by 1 for each i in A. At the end of this step, B will contain all the elements in A in sorted order

Radix Sort

Radix sort can sort n integers in base k with at most d digits in $O(d(n + k))$ time. It does this by using counting sort to sort the n integers by digits, starting from the least significant digit (i.e. ones digit for integers) to the most significant digit. Each counting sort will take $O(n + k)$ time since there are n elements and the elements are all integers in the range 0 to k since we're in base k. Since the maximum number of digits in these n integers is d, we will have to execute counting sort d times to finish the algorithm. This is how we get a $O(d(n + k))$ running time for radix sort.

The running time of radix sort depends on the base k that the integers are represented in. Large bases result in slower counting sorts, but fewer counting sorts since the number of digits in the elements decrease. On the other hand, small bases result result in faster counting sorts, but more digits and consequently more counting sorts.

Let's find the optimal base k for radix sort. Say we are sorting n integers in the range 0 to $u-1$. The maximum number of digits in an element will be $\log_k u$ for some base k. To minimize running time, we will want to minimize $O((n + k) \log_k u)$. It turns out that to minimize running time, the best k to choose is $k = n$, in which case the running time of radix sort would be $O(n \log_n u)$. Note that if $u = n^{O(1)}$, the running time of radix sort turns out to be $O(n)$, giving us a linear time sorting algorithm if the range of integers we're sorting is polynomial in the number of integers we're sorting.