Radix Sort

QuickSort, MergeSort, HeapSort are comparison based sorting algorithms. CountSort is not comparison based algorithm. It has the complexity of O(n+k), where k is the maximum element of the input array. So, if k is O(n) ,CountSort becomes linear sorting, which is better than comparison based sorting algorithms that have O(nlogn) time complexity. The idea is to extend the CountSort algorithm to get a better time complexity when k goes $O(n^2)$. Here comes the idea of Radix Sort.

Algorithm:

```
For each digit where varies from the least significant digit to the most significant digit of a number
    Sort input array using countsort algorithm according to i-th digit.
```

We used count sort because it is a stable sort.

Example: Assume the input array is: `10, 21, 17, 34, 44, 11, 654, 123`

```
Based on the algorithm, we will sort the input array according to the one's digit (least significant digit).
0: 10
1: 21 11
2:
3: 123
4: 34 44 654
5:
6:
7: 17
8:
9:

So, the array becomes 10,21,11,123,24,44,654,17
Now, we'll sort according to the ten's digit:
0:
1: 10 11 17
2: 21 123
3: 34
4: 44
5: 654
6:
7:
8:
9:

Now, the array becomes : 10,11,17,21,123,34,44,654
Finally , we sort according to the hundred's digit (most significant digit):
0: 010 011 017 021 034 044
1: 123
2:
3:
4:
5:
6: 654
7:
8:
9:

The array becomes : 10,11,17,21,34,44,123,654 which is sorted. This is how our algorithm works.
```

What is the running time of Radix Sort?

Let there be d digits in input integers. Radix Sort takes O(d*(n+b)) time where b is the base for representing numbers, for example, for decimal system, b is 10. What is the value of d? If k is the maximum possible value, then d would be $O(\log_b(k))$. So overall time complexity is $O((n+b) * \log_b(k))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k. Let us first limit k. Let $k <= n^c$ where c is a constant. In that case, the complexity becomes $O(n\text{Log}_b(n))$.

But it still doesn’t beat comparison based sorting algorithms.
What if we make value of b larger?. What should be the value of b to make the time complexity linear? If we set b as n, we get the time complexity as O(n). In other words, we can sort an array of integers with range from 1 to n^c if the numbers are represented in base n (or every digit takes $\log_2(n)$ bits).

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?

If we have $\log_2 n$ bits for every digit, the running time of Radix appears to be better than Quick Sort for a wide range of input numbers. The constant factors hidden in asymptotic notation are higher for Radix Sort and Quick-Sort uses hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.