Radix Sort

The lower bound for Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is Ω(nLogn), i.e., they cannot do better than nLogn.

Counting sort is a linear time sorting algorithm that sort in O(n+k) time when elements are in the range from 1 to k.

***What if the elements are in* the** ***range from 1 to n2? ***
We can’t use counting sort because counting sort will take O(n2) which is worse than comparison-based sorting algorithms. Can we sort such an array in linear time?

Radix Sort is the answer. The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

The Radix Sort Algorithm

1. Do following for each digit i where i varies from least significant digit to the most significant digit.
    - Sort input array using counting sort (or any stable sort) according to the i’th digit.

Example:

Original, unsorted list:
170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives:
[*Notice that we keep 802 before 2, because 802 occurred
before 2 in the original list, and similarly for pairs
170 & 90 and 45 & 75.]

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

Sorting by next digit (10s place) gives:
[*Notice that 802 again comes before 2 as 802 comes before
2 in the previous list.]

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

Sorting by the most significant digit (100s place) gives:
2, 24, 45, 66, 75, 90, 170, 802

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 the decimal system, b is 10. What is the value of d? If k is the maximum possible value, then d would be O(logb(k)). So overall time complexity is O((n+b) * logb(k)). Which looks more than the time complexity of comparison-based sorting algorithms for a large k. Let us first limit k. Let k <= nc where c is a constant.

In that case, the complexity becomes O(nLogb(n)). But it still doesn't beat comparison-based sorting algorithms.
What if we make the 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 a range from 1 to nc if the numbers are represented in base n (or every digit takes log2(n) bits).

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?
If we have log2n 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.

```
def countingSort(arr,exp=1):
    freqArr = [0]*10
    # Recording the frequency of each elements
    for ele in arr:
        idx = (ele//exp) %10
        freqArr[idx] = freqArr[idx]+1
    # Writing the prefix Sum in freq arr
    for i in range(1,len(freqArr)):
        freqArr[i] = freqArr[i-1]+freqArr[i]

#     Declaring a new array of size equal to arr
    ans = [0]*len(arr)
    for i in range(len(arr)-1,-1,-1):
        idxOfFreqArray = (arr[i]//exp)%10
        valInFreqArr = freqArr[idxOfFreqArray]
        idxInResultingArray = valInFreqArr-1
        ans[idxInResultingArray] = arr[i]
        freqArr[idxOfFreqArray] = freqArr[idxOfFreqArray]-1

    for i in range(len(ans)):
        arr[i] = ans[i]
    # print('After sorting on {} place,the array is==>'.format(exp))
    # print(arr)
```

```python
def radixSort(arr):
    maxi = max(arr)
    exp = 1
    while exp<=maxi:
        countingSort(arr,exp)
        exp = exp*10



arr = [86,15,5648,20,1,59]
radixSort(arr)
print(arr)
```