Radix Sort

The lower bound for the Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is $\Omega(n\text{Log}n)$, 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 n^2?

We can't use counting sort because counting sort will take $O(n^2)$ 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

Do the following for each digit I where I varies from the least significant digit to the most significant digit. Here we will be sorting the 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 $O(\log_b(k))$. 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 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 n^c if the numbers are represented in base n (or every digit takes $\log_2(n)$ bits).

Applications of Radix Sort:

- In a typical computer, which is a sequential random-access machine, where the records are keyed by multiple fields radix sort is used. For eg., you want to sort on three keys month, day and year. You could compare two records on year, then on a tie on month and finally on the date. Alternatively, sorting the data three times using Radix sort first on the date, then on month, and finally on year could be used.
- It was used in card sorting machines with 80 columns, and in each column, the machine could punch a hole only in 12 places. The sorter was then programmed to sort the cards, depending upon which place the card had been punched. This was then used by the operator to collect the cards which had the 1st row punched, followed by the 2nd row, and so on.

Implementation

C++Java

```cpp
// C++ implementation of Radix Sort
#include <iostream>
using namespace std;

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)
{
    int output[n]; // output array
    int i, count[10] = { 0 };
    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    // Change count[i] so that count[i] now contains actual
    //  position of this digit in output[]
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // Build the output array
    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)
{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}

// A utility function to print an array
void print(int arr[], int n)
```

```
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()
{
    int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
    int n = sizeof(arr) / sizeof(arr[0]);

      // Function Call
      radixsort(arr, n);
    print(arr, n);
    return 0;
}
```

Output

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