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 $O(n\log_b(n))$. 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.

Implementation of Radix Sort

Following is a simple C++ implementation of Radix Sort. For simplicity, the value of d is assumed to be 10. We recommend you to see Counting Sort for details of countSort() function in below code.

C/C++

```
// 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 program to test above functions
int main()
{
    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
    int n = sizeof(arr)/sizeof(arr[0]);
    radixsort(arr, n);
    print(arr, n);
    return 0;
}
```

Java

```
// Radix sort Java implementation
import java.io.*;
import java.util.*;

class Radix {
```

```
// A utility function to get maximum value in arr[]
static 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.
static void countSort(int arr[], int n, int exp)
{
    int output[] = new int[n]; // output array
    int i;
    int count[] = new int[10];
    Arrays.fill(count,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 curent digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
static 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
    static void print(int arr[], int n)
    {
        for (int i=0; i<n; i++)
            System.out.print(arr[i]+" ");
    }


    /*Driver function to check for above function*/
    public static void main (String[] args)
    {
        int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
        int n = arr.length;
        radixsort(arr, n);
        print(arr, n);
    }
}
/* This code is contributed by Devesh Agrawal */
```

Output:

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

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Counting Sort
- Bucket Sort
- ShellSort

References:

http://en.wikipedia.org/wiki/Radix_sort
http://alg12.wikischolars.columbia.edu/file/view/RADIX.pdf
MIT Video Lecture
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above