geeksforgeeks.org

Radix Sort - GeeksforGeeks

6-8 minutes

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 range from 1 to k.

What if the elements are in 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
1) Do following for each digit i where i varies from least significant digit to the most significant digit.
………….**a)** 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 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 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(nLog$_b$(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++
- Java
- Python

C/C++

```
#include<iostream>
using namespace std;
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;
}
void countSort(int arr[], int n, int exp)
{
    int output[n];
    int i, count[10] = {0};
    for (i = 0; i < n; i++)
        count[ (arr[i]/exp)%10 ]++;
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];
    for (i = n - 1; i >= 0; i--)
    {
        output[count[ (arr[i]/exp)%10 ] - 1] =
arr[i];
        count[ (arr[i]/exp)%10 ]--;
    }
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}
void radixsort(int arr[], int n)
{
    int m = getMax(arr, n);
    for (int exp = 1; m/exp > 0; exp *= 10)
        countSort(arr, n, exp);
}
void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}
```

```
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

```
import java.io.*;
import java.util.*;
class Radix {
    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;
    }
    static void countSort(int arr[], int n, int exp)
    {
        int output[] = new int[n];
        int i;
        int count[] = new int[10];
        Arrays.fill(count,0);
        for (i = 0; i < n; i++)
            count[ (arr[i]/exp)%10 ]++;
        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];
        for (i = n - 1; i >= 0; i--)
        {
            output[count[ (arr[i]/exp)%10 ] - 1] =
arr[i];
            count[ (arr[i]/exp)%10 ]--;
        }
        for (i = 0; i < n; i++)
            arr[i] = output[i];
    }
    static void radixsort(int arr[], int n)
    {
        int m = getMax(arr, n);
        for (int exp = 1; m/exp > 0; exp *= 10)
            countSort(arr, n, exp);
    }
    static void print(int arr[], int n)
    {
        for (int i=0; i<n; i++)
            System.out.print(arr[i]+" ");
    }
    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);
    }
}
```

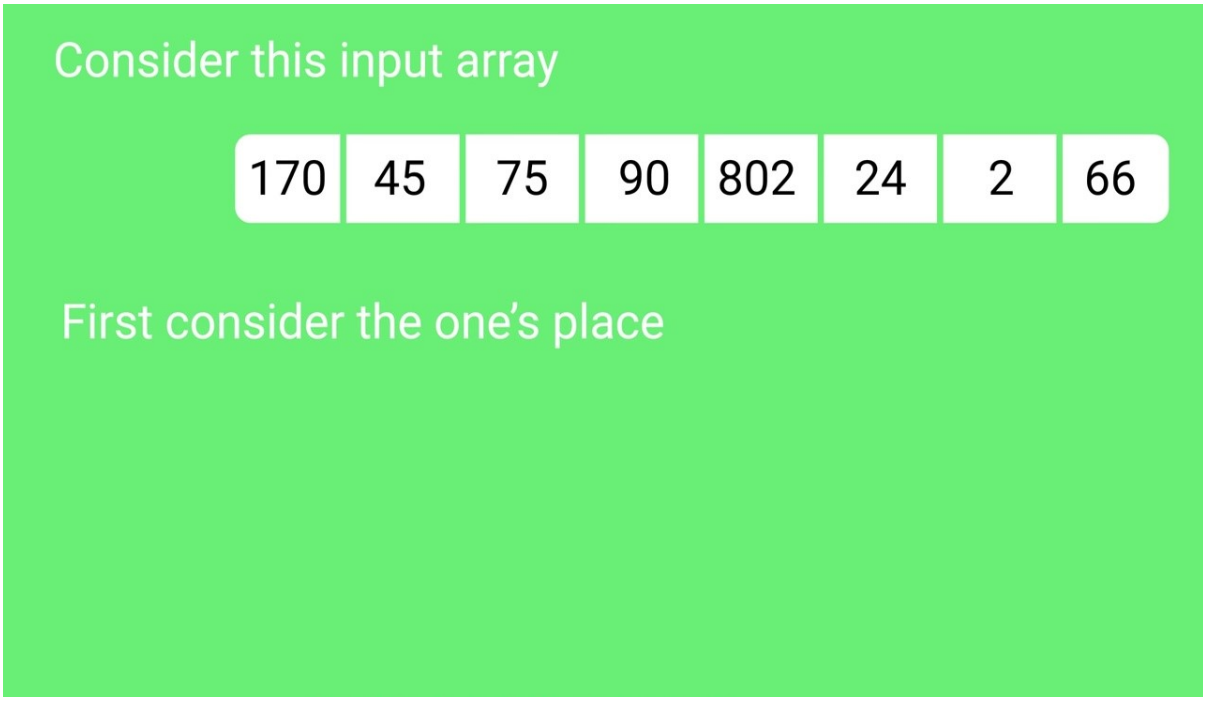
Python

```
def countingSort(arr, exp1):
    n = len(arr)
    output = [0] * (n)
    count = [0] * (10)
    for i in range(0, n):
        index = (arr[i]/exp1)
        count[ (index)%10 ] += 1
    for i in range(1,10):
        count[i] += count[i-1]
    i = n-1
    while i>=0:
        index = (arr[i]/exp1)
        output[ count[ (index)%10 ] - 1] = arr[i]
        count[ (index)%10 ] -= 1
        i -= 1
    i = 0
    for i in range(0,len(arr)):
        arr[i] = output[i]
def radixSort(arr):
    max1 = max(arr)
    exp = 1
    while max1/exp > 0:
        countingSort(arr,exp)
        exp *= 10
arr = [ 170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)
for i in range(len(arr)):
    print(arr[i]),
```

Output:

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

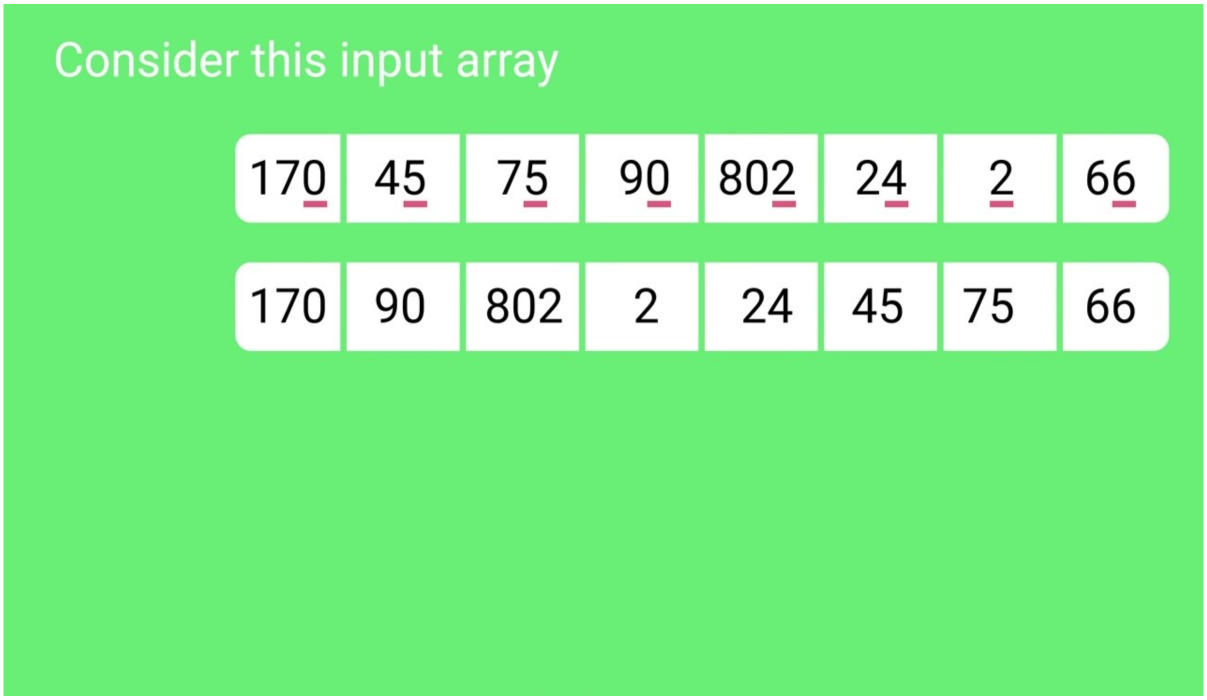
Snapshots:



Consider this input array

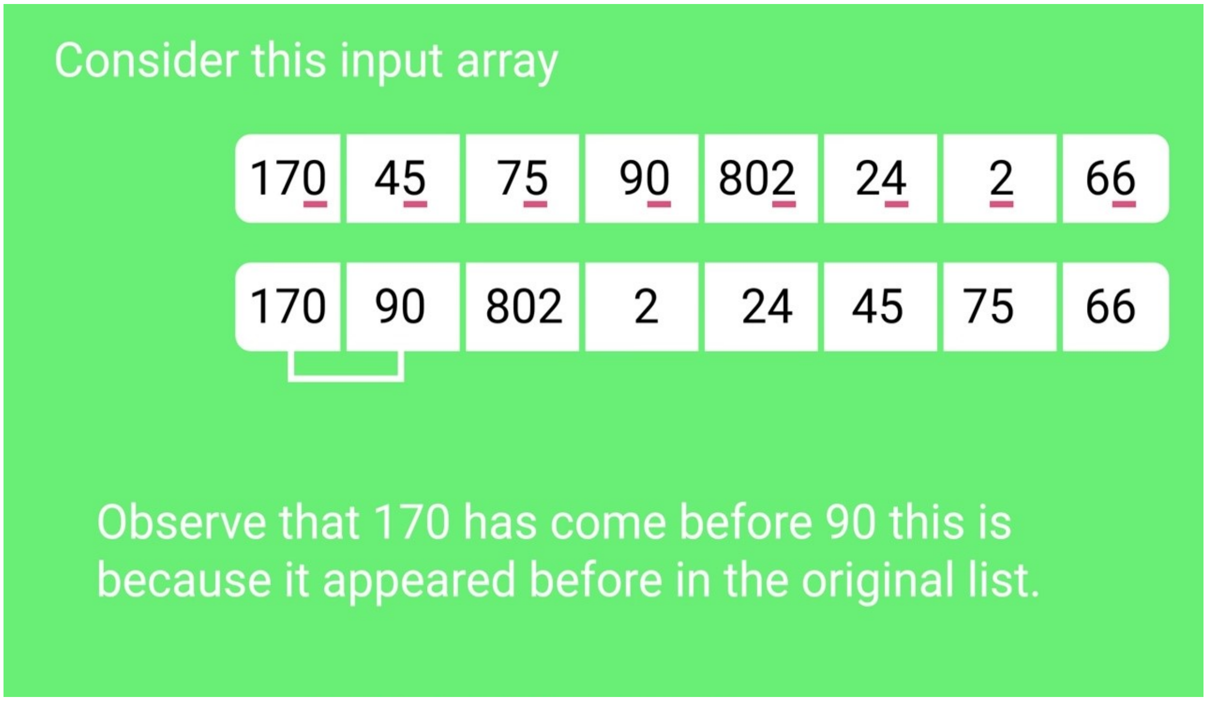
170	45	75	90	802	24	2	66

First consider the one's place



Consider this input array

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

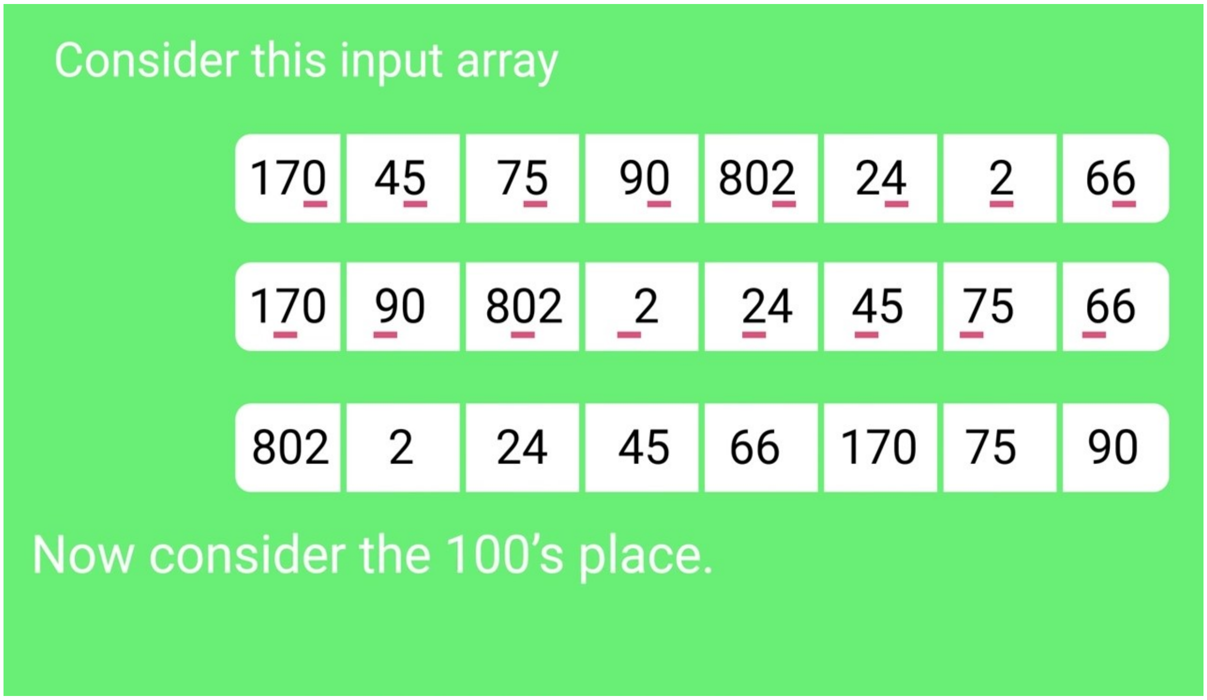


Consider this input array

170	45	75	90	802	24	2	66

170	90	802	2	24	45	75	66

Observe that 170 has come before 90 this is because it appeared before in the original list.



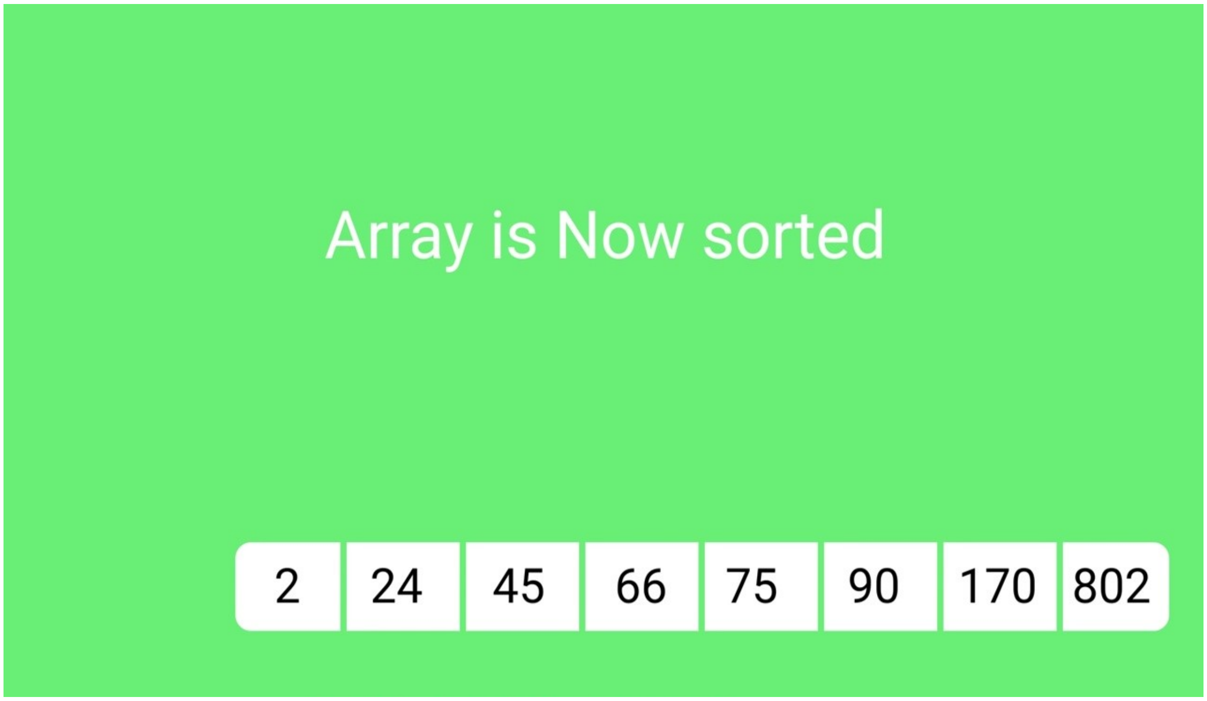
Consider this input array

170	45	75	90	802	24	2	66

170	90	802	2	24	45	75	66

802	2	24	45	66	170	75	90

Now consider the 100’s place.



Quiz on Radix Sort

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