

**Write an
Article**

Login

Quick Links for
Sorting

Sorting
Terminology

Stability in
sorting
algorithms

Time
Complexities of
all Sorting
Algorithms

External Sorting

Sorting
Algorithms

Bubble Sort,
Recursive Bubble
Sort

Insertion Sort,
Recursive
Insertion Sort

Selection Sort

Merge Sort,
Iterative Merge
Sort

3 - Way Merge
Sort

Quick Sort,
Iterative Quick
Sort

3 - Way Quick
Sort

Heap Sort

Counting Sort

Radix Sort



Bucket Sort
Shell Sort
Tim Sort
Bitonic Sort
Binary Insertion Sort
Comb Sort
Cycle Sort
Pancake Sorting
Pigeonhole Sort
Gnome Sort
Tree Sort
Explore More...
Library Implementations of Sorting Algorithms :
Comparator function of qsort() in C
std::sort() in C++ STL
C qsort() vs C++ sort()
Arrays.sort() in Java with examples
Collections.sort() in Java with Examples
Know Your Sorting Algorithm :
Sorting Weapons used by Programming Languages
Introsort- C++'s Sorting Weapon



Misc :
Hoare's vs Lomuto partition scheme in QuickSort
Serial Sort v/s Parallel Sort in Java
Lower bound for comparison based sorting algorithms
Which sorting algorithm makes minimum number of memory writes?
Where is Heap Sort used practically?
Find memory conflicts among multiple threads
Explore More...
Searching
Linear Search
Binary Search
Fibonacci Search
More...
Arrays
Segregate 0s and 1s in an array
Rearrange positive and negative numbers in $O(n)$ time and $O(1)$ extra space
Maximum Product Sub-array
More...
Strings



Return maximum occurring character in an input string
Check whether two strings are anagram of each other
Remove all duplicates from a given string
More...
Linked List
Detect loop in a linked list
Find the middle of a given linked list
Reverse a linked list
More...
Stack
Implement two stacks in an array
Infix to Postfix Conversion using Stack
The Stock Span Problem
More...
Queue
Implement Queue using Stacks
Implementation of Deque using circular array
How to efficiently implement k Queues in a single array?
More...



Binary Tree
Print level order traversal line by line Set 1
Diagonal Traversal of Binary Tree
Boundary Traversal of binary tree
More...
Binary Search Tree
Lowest Common Ancestor in a Binary Search Tree
Check if each internal node of a BST has exactly one child
Merge Two Balanced Binary Search Trees
More...
Heap
Binomial Heap
Fibonacci Heap
More...
Graph
Detect Cycle in an Undirected Graph
Longest Path in a Directed Acyclic Graph
More...
Hashing
Count distinct elements in every window of size k



Largest subarray
with equal
number of 0s and
1s

More...

Radix Sort

The **lower bound for Comparison based sorting algorithm** (Merge Sort, Heap Sort, Quick-Sort .. etc) is $\Omega(n \log n)$, i.e., they cannot do better than $n \log n$.

3.3

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 $O(d \cdot (n+b))$ 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 $O((n+b) \cdot \log_b(k))$. Which looks more than the time complexity of comparison based sorting algorithms for a large k . Let us first limit k . Let $k \leq n^c$ 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.

Recommended: Please try your approach on [{IDE}](#) first, before moving on to the solution.

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;
}

```

[Run on IDE](#)

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 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
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 */

```

[Run on IDE](#)

Python

```

# Python program for implementation of Radix Sort

# A function to do counting sort of arr[] according to
# the digit represented by exp.
def countingSort(arr, exp1):

    n = len(arr)

    # The output array elements that will have sorted arr
    output = [0] * (n)

    # initialize count array as 0
    count = [0] * (10)

    # Store count of occurrences in count[]
    for i in range(0, n):
        index = (arr[i]/exp1)
        count[ (index)%10 ] += 1

    # Change count[i] so that count[i] now contains actual
    # position of this digit in output array
    for i in range(1,10):
        count[i] += count[i-1]

    # Build the output array
    i = n-1
    while i>=0:
        index = (arr[i]/exp1)
        output[ count[ (index)%10 ] - 1 ] = arr[i]
        count[ (index)%10 ] -= 1
        i -= 1

    # Copying the output array to arr[],
    # so that arr now contains sorted numbers
    i = 0
    for i in range(0,len(arr)):
        arr[i] = output[i]

# Method to do Radix Sort
def radixSort(arr):

    # Find the maximum number to know number of digits
    max1 = max(arr)

    # 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
    exp = 1
    while max1/exp > 0:
        countingSort(arr,exp)
        exp *= 10

# Driver code to test above
arr = [ 170, 45, 75, 90, 802, 24, 2, 66]

```



```
radixSort(arr)

for i in range(len(arr)):
    print(arr[i]),

# This code is contributed by Mohit Kumra
```

[Run on IDE](#)

Output:

2 24 45 66 75 90 170 802

连接超时

www.youtube.com 的服务器响应时间过长。

- 此站点暂时无法使用或者太过忙碌。请过几分钟后再试。
- 如果您无法载入任何网页，请检查您计算机的网络连接状态。
- 如果您的计算机或网络受到防火墙或者代理服务器的保护，请确认 Firefox 已被授权访问网络。

[重试](#)

Snapshots:



Consider this input array

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

First consider the one's place

Consider this input array

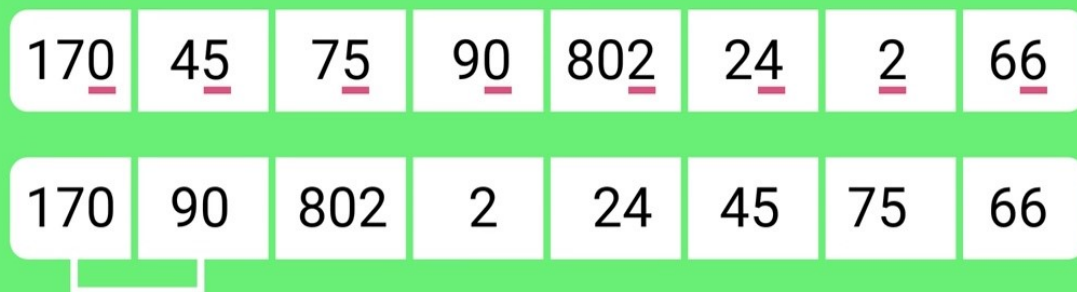
17 <u>0</u>	4 <u>5</u>	7 <u>5</u>	9 <u>0</u>	80 <u>2</u>	24 <u> </u>	<u>2</u>	6 <u>6</u>
-------------	------------	------------	------------	-------------	-------------	----------	------------

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.



Array is Now sorted

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

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

GATE CS Corner Company Wise Coding Practice

Sorting

[Login to Improve this Article](#)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Counting Sort

Bucket Sort

QuickSort

Heap Sort

Lower bound for comparison based sorting algorithms

Sort the given string using character search

Pairs such that one is a power multiple of other

Loop Invariant Condition with Examples of Sorting Algorithms

Sort on the basis of number of factors using STL

Insertion sort to sort even and odd positioned elements in different orders

(Login to Rate)

3.3

Average Difficulty : 3.3/5.0
Based on 71 vote(s)



Add to TODO List



Mark as DONE

Basic

Easy

Medium

Hard

Expert

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!



A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
GBlog
Videos

@geeksforgeeks, Some rights reserved

