Radix sort

# Radix sort

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.

LSD radix sorts typically use the following sorting order: short keys come before longer keys, and then keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as "b, c, d, e, f, g, h, i, j, ba" would be lexicographically sorted as "b, ba, c, d, e, f, g, h, i, j".

The idea is to extend the Count sort algorithm to get a better time complexity when k goes O(n2).

# Algorithm

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.

Radix-Sort (list, n)

shift = 1

for loop = 1 to keysize do

for entry = 1 to n do

bucketnumber = (list[entry].key / shift) mod 10

append (bucket[bucketnumber], list[entry])

list = combinebuckets()

shift = shift \* 10

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

# Least significant digit (LSD) radix sorts

A Least significant digit (LSD) Radix sort is a fast stable sorting algorithm which can be used to sort keys in integer representation order.

Time complexity O(nw) time

n is the number of keys

w is the average key length

# 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.

# 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(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 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 nc if the numbers are represented in base n (or every digit takes log2(n) bits).

# Implementation of Radix Sort

// 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;

}

Output:

2 24 45 66 75 90 170 802

# Advantages

1. Fast when the keys are short i.e. when the range of the array elements is less.
2. Used in suffix array construction algorithms like Manber's algorithm and DC3 algorithm.

# Disadvantages

1. Since Radix Sort depends on digits or letters, Radix Sort is much less flexible than other sorts. Hence, for every different type of data it needs to be rewritten.
2. The constant for Radix sort is greater compared to other sorting algorithms.
3. It takes more space compared to Quicksort which is in-place sorting.

The Radix Sort algorithm is an important sorting algorithm that is integral to suffix -array construction algorithms. It is also useful on parallel machines.

# Quiz

1. If we use Radix Sort to sort n integers in the range (nk/2,nk], for some k>0 which is independent of n, the time taken would be?

A Θ(n)

B Θ(kn)

**C Θ(nlogn) ☑**

D Θ(n2)

Radix sort time complexity = O(wn)

for n keys of word size= w => w = log(nk)

O(wn)=O(klogn.n) => kO(nlogn)

1. If there are n integers to sort, each integer has d digits, and each digit is in the set {1, 2, ..., k}, radix sort can sort the numbers in:

A O(k(n + d))

**B O(d(n + k)) ☑**

C O((n + k)logd)

D O((n + d)logk)