Radix Sort

The lower bound for the 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 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*

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.

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?

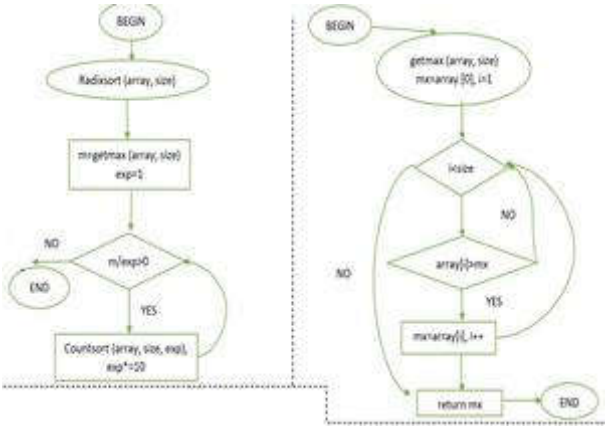
hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.

Key points about Radix Sort:

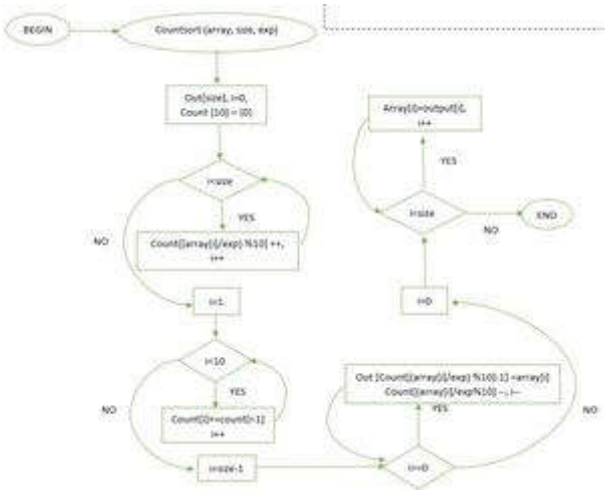
Some key points about radix sort are given here

1. It makes assumptions about the data like the data must be between a range of elements.
2. Input array must have the elements with the same radix and width.
3. Radix sort works on sorting based on an individual digit or letter position.
4. We must start sorting from the rightmost position and use a stable algorithm at each position.
5. Radix sort is not an in-place algorithm as it uses a temporary count array.

FLowcharts



Flowchart radix sort, getmax



Following is a simple 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 the below code.

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++)
```

```
// 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;
}
```

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)
```

```
// 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)
{
```

```
    public static void main(String[] args)
    {
        int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
        int n = arr.length;

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

Python3

```
# 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
```

```
# 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 >= 1:
        countingSort(arr, exp)
        exp *= 10


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

# Function Call
radixSort(arr)

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

# This code is contributed by Mohit Kumra
# Edited by Patrick Gallagher
```

Javascript

```
// Radix sort Javascript implementation

// A utility function to get maximum value in arr[]
function getMax(arr,n)
{
    let mx = arr[0];
        for (let 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.
function countSort(arr,n,exp)
{
    let output = new Array(n); // output array
        let i;
```

```
        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            let x = Math.floor(arr[i] / exp) % 10;
            count[x]++;

        // 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[x] - 1] = arr[i];
            count[x]--;
        }

        // 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
function radixsort(arr,n)
{
    // Find the maximum number to know number of digits
        let 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 (let exp = 1; Math.floor(m / exp) > 0; exp *= 10)
            countSort(arr, n, exp);
}

// A utility function to print an array
function print(arr,n)
{
    for (let i = 0; i < n; i++)
            document.write(arr[i] + " ");
}

/*Driver Code*/
let arr=[170, 45, 75, 90, 802, 24, 2, 66];
let n = arr.length;

// Function Call
```

C#

```
// C# implementation of Radix Sort
using System;

class GFG {
    public 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.
    public static void countSort(int[] arr, int n, int exp)
    {
        int[] output = new int[n]; // output array
        int i;
        int[] count = new int[10];

        // initializing all elements of count to 0
        for (i = 0; i < 10; i++)
            count[i] = 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++)
```

```
    // Radix Sort
    public 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
    public static void print(int[] arr, int n)
    {
        for (int i = 0; i < n; i++)
            Console.Write(arr[i] + " ");
    }

    // Driver Code
    public static void Main()
    {
        int[] arr = { 170, 45, 75, 90, 802, 24, 2, 66 };
        int n = arr.Length;

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

    // This code is contributed by DrRoot_
}
```

PHP

```
<?php
// PHP implementation of Radix Sort


// A function to do counting sort of arr[]
// according to the digit represented by exp.
function countSort(&$arr, $n, $exp)
{
    $output = array_fill(0, $n, 0); // output array
    $count = array_fill(0, 10, 0);
```

```
        // 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
    function radixsort(&$arr, $n)
    {

        // Find the maximum number to know
        // number of digits
        $m = max($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
        for ($exp = 1; $m / $exp > 0; $exp *= 10)
            countSort($arr, $n, $exp);
    }

    // A utility function to print an array
    function PrintArray(&$arr,$n)
    {
        for ($i = 0; $i < $n; $i++)
            echo $arr[$i] . " ";
    }

    // Driver Code
    $arr = array(170, 45, 75, 90, 802, 24, 2, 66);
    $n = count($arr);
```

```
// This code is contributed by rathbhupendra
?>
```

Output

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

Following is another way of the implementation of the radix sort while using the bucket sort technique, it might not look simple while having a look at the code but if you give it a shot it's quite easy, one must know Bucket Sort to deeper depth.

C++

```
#include <bits/stdc++.h>
using namespace std;

// structure for a single linked list to help further in the
// sorting
struct node {
    int data;
    node* next;
};

// function for creating a new node in the linked list
struct node* create(int x)
{
    node* temp = new node();
    temp->data = x;
    temp->next = NULL;

    return temp;
}

// utility function to append node in the linked list
// here head is passed by reference, to know more about this
// search pass by reference
void insert(node*& head, int n)
{
    if (head == NULL) {
        head = create(n);
        return;
    }
```

```
}

// utility function to pop an element from front in the list
// for the sake of stability in sorting
int del(node*& head)
{
    if (head == NULL)
        return 0;
    node* temp = head;
    // storing the value of head before updating
    int val = head->data;

    // updation of head to next node
    head = head->next;

    delete temp;
    return val;
}

// utility function to get the number of digits in the
// max_element
int digits(int n)
{
    int i = 1;
    if (n < 10)
        return 1;

    while (n > (int)pow(10, i))
        i++;
    return i;
}

void radix_sort(vector<int>& arr)
{
    // size of the array to be sorted
    int sz = arr.size();

    // getting the maximum element in the array
    int max_val = *max_element(arr.begin(), arr.end());

    // getting digits in the maximum element
    int d = digits(max_val);

    // creating buckets to store the pointers
    node** bins;

    // array of pointers to linked list of size 10 as
    // integers are decimal numbers so they can hold numbers
```

```
    for (int i = 0; i < 10; i++)
        bins[i] = NULL;

    // first loop working for a constant time only and inner
    // loop is iterating through the array to store elements
    // of array in the linked list by their digits value
    for (int i = 0; i < d; i++) {
        for (int j = 0; j < sz; j++) // bins updation
            insert(bins[(arr[j] / (int)pow(10, i)) % 10],
                arr[j]);

        int x = 0, y = 0;
        // write back to the array after each pass

        while (x < 10) {
            while (bins[x] != NULL)
                arr[y++] = del(bins[x]);
            x++;
        }
    }
}

// a utility function to print the sorted array
void print(vector<int> arr)
{
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    vector<int> arr = { 573, 25, 415, 12, 161, 6 };

    // function call
    radix_sort(arr);
    print(arr);

    return 0;
}
```

Java

```
import java.util.LinkedList;

public class RadixSort {
```

```
        int i = 1;
        if (n < 10)
            return 1;

        while (n > Math.pow(10, i))
            i++;

        return i;
    }

    public static void radixSort(int[] arr)
    {
        // size of the array to be sorted
        int sz = arr.length;

        // getting the maximum element in the array
        int max_val = Integer.MIN_VALUE;
        for (int i = 0; i < sz; i++)
            max_val = Math.max(max_val, arr[i]);

        // getting digits in the maximum element
        int d = digits(max_val);

        // creating buckets to store the pointers
        LinkedList<Integer>[] bins = new LinkedList[10];
        for (int i = 0; i < 10; i++)
            bins[i] = new LinkedList<Integer>();

        // first loop working for a constant time only and
        // inner loop is iterating through the array to
        // store elements of array in the linked list by
        // their digits value
        for (int i = 0; i < d; i++) {
            for (int j = 0; j < sz; j++) {
                int digit
                    = (arr[j] / (int)Math.pow(10, i)) % 10;
                bins[digit].add(arr[j]);
            }

            int x = 0, y = 0;
            // write back to the array after each pass
            while (x < 10) {
                while (!bins[x].isEmpty()) {
                    arr[y] = bins[x].remove();
                    y++;
                }
                x++;
            }
        }
```

```
    {
        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    public static void main(String[] args)
    {
        int[] arr = { 573, 25, 415, 12, 161, 6 };

        // function call
        radixSort(arr);
        printArr(arr);
    }
}
```

Python3

```
# Python3 code for the approach

# structure for a single linked list to help further in the sorting


class Node:
    def __init__(self, x):
        self.data = x
        self.next = None

# function for creating a new node in the linked list


def create(x):
    temp = Node(x)
    return temp

# utility function to append node in the linked list
# here head is passed by reference, to know more about this
# search pass by reference


def insert(head, n):
    if head is None:
        head = create(n)
        return head

    t = head
    while t.next is not None:
```

```
    # utility function to pop an element from front in the list
    # for the sake of stability in sorting


    def delete(head):
        if head is None:
            return None
        temp = head
        # storing the value of head before updating
        val = head.data

        # updation of head to next node
        head = head.next

        del temp
        return val, head

    # utility function to get the number of digits in the max_element


    def digits(n):
        i = 1
        if n < 10:
            return 1

        while n > (10 ** i):
            i += 1
        return i


    def radix_sort(arr):
        # size of the array to be sorted
        sz = len(arr)

        # getting the maximum element in the array
        max_val = max(arr)

        # getting digits in the maximum element
        d = digits(max_val)

        # creating buckets to store the pointers
        bins = [None] * 10

        # first loop working for a constant time only and inner loop is iterating
        # through the array to store elements of array in the linked list by their
        for i in range(d):
            for j in range(sz):  # bins updation
                bins[(arr[j] // (10 ** i)) %
```

```
            while bins[x] is not None:
                arr[y], bins[x] = delete(bins[x])
                y += 1
            x += 1

# a utility function to print the sorted array


def print_arr(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


if __name__ == "__main__":
    arr = [573, 25, 415, 12, 161, 6]

    # function call
    radix_sort(arr)
    print_arr(arr)
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program {
    // structure for a single linked list to help further in
    // the sorting
    class Node {
        public int data;
        public Node next;
    }

    // function for creating a new node in the linked list
    static Node Create(int x)
    {
        Node temp = new Node();
        temp.data = x;
        temp.next = null;

        return temp;
    }

    // utility function to append node in the linked list
```

```
        if (head == null) {
            head = Create(n);
            return;
        }

        Node t = head;
        while (t.next != null)
            t = t.next;
        t.next = Create(n);
    }

    // utility function to pop an element from front in the
    // list for the sake of stability in sorting
    static int Del(ref Node head)
    {
        if (head == null)
            return 0;
        Node temp = head;
        // storing the value of head before updating
        int val = head.data;

        // updation of head to next node
        head = head.next;

        temp = null;
        return val;
    }

    // utility function to get the number of digits in the
    // max_element
    static int Digits(int n)
    {
        int i = 1;
        if (n < 10)
            return 1;

        while (n > (int)Math.Pow(10, i))
            i++;
        return i;
    }

    static void RadixSort(List<int> arr)
    {
        // size of the array to be sorted
        int sz = arr.Count;

        // getting the maximum element in the array
        int max_val = arr.Max();
```

```
        Node[] bins;

        // array of pointers to linked list of size 10 as
        // integers are decimal numbers so they can hold
        // numbers from 0-9 only, that's why size of 10

        bins = new Node[10];

        // initializing the hash array with null to all
        for (int i = 0; i < 10; i++)
            bins[i] = null;

        // first loop working for a constant time only and
        // inner loop is iterating through the array to
        // store elements of array in the linked list by
        // their digits value
        for (int i = 0; i < d; i++) {
            for (int j = 0; j < sz; j++) // bins updation
                Insert(
                    ref bins[(arr[j] / (int)Math.Pow(10, i))
                             % 10],
                    arr[j]);

            int x = 0, y = 0;
            // write back to the array after each pass

            while (x < 10) {
                while (bins[x] != null)
                    arr[y++] = Del(ref bins[x]);
                x++;
            }
        }
    }

    // a utility function to print the sorted array
    static void Print(List<int> arr)
    {
        for (int i = 0; i < arr.Count; i++)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    static void Main(string[] args)
    {
        List<int> arr
            = new List<int>{ 573, 25, 415, 12, 161, 6 };

        // function call
```

Javascript

```
// JavaScript code for the approach

// structure for a single linked list to help further in the sorting
class Node {
    constructor(x) {
        this.data = x;
        this.next = null;
    }
}

// function for creating a new node in the linked list
function create(x) {
    let temp = new Node(x);
    return temp;
}

// utility function to append node in the linked list
// here head is passed by reference, to know more about this
// search pass by reference
function insert(head, n) {
    if (head === null) {
        head = create(n);
        return head;
    }


    let t = head;
    while (t.next !== null) {
        t = t.next;
    }
    t.next = create(n);
    return head;

}

// utility function to pop an element from front in the list
// for the sake of stability in sorting
function deleteNode(head) {
    if (head === null) {
        return null;
    }
    let temp = head;
    // storing the value of head before updating
    let val = head.data;
```

```
    return [val, head];

}

// utility function to get the number of digits in the max_element
function digits(n) {
    let i = 1;
    if (n < 10) {
        return 1;
    }

    while (n > Math.pow(10, i)) {
        i += 1;
    }
    return i;

}

function radixSort(arr) {
    // size of the array to be sorted
    let sz = arr.length;


    // getting the maximum element in the array
    let max_val = Math.max(...arr);

    // getting digits in the maximum element
    let d = digits(max_val);

    // creating buckets to store the pointers
    let bins = new Array(10);
    bins.fill(null);

    // first loop working for a constant time only and inner loop is iterating
    // through the array to store elements of array in the linked list by thei
    for (let i = 0; i < d; i++) {
        for (let j = 0; j < sz; j++) {
            // bins updation
            bins[Math.floor(arr[j] / Math.pow(10, i)) % 10] = insert(bins[Math
        }

        let x = 0,
            y = 0;
        // write back to the array after each pass
        while (x < 10) {
            while (bins[x] !== null) {
                [arr[y], bins[x]] = deleteNode(bins[x]);
                y += 1;
```

```
}

// a utility function to print the sorted array
function printArr(arr) {
    console.log(arr.join(" "))
}


// Driver code
let arr = [573, 25, 415, 12, 161, 6];

// function call
radixSort(arr);
printArr(arr);
```

Output

```
6 12 25 161 415 573
```

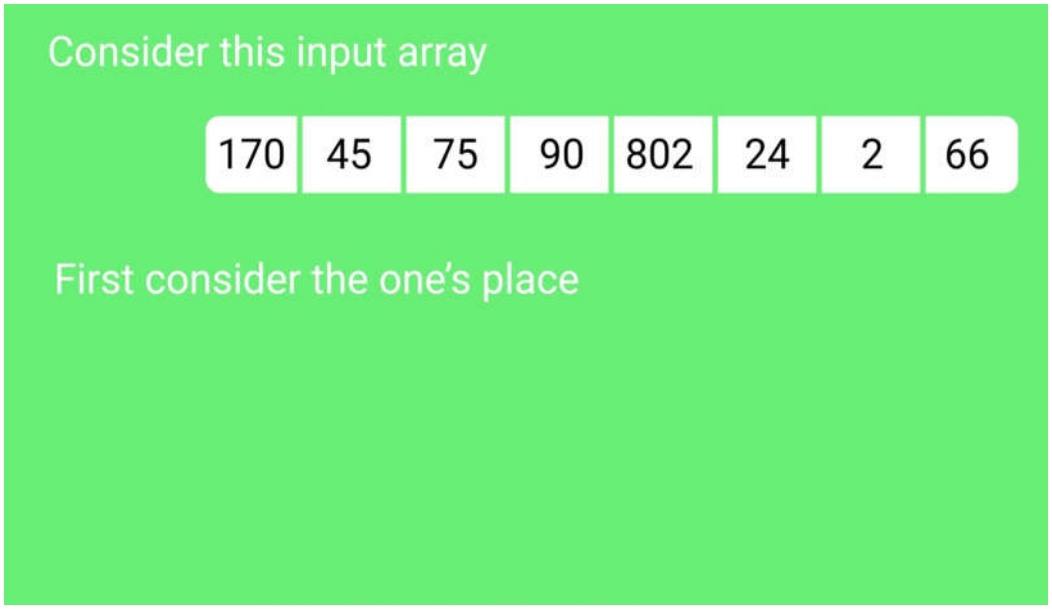
Time complexities remain the same as in the first method, it's just the implementation through another method.

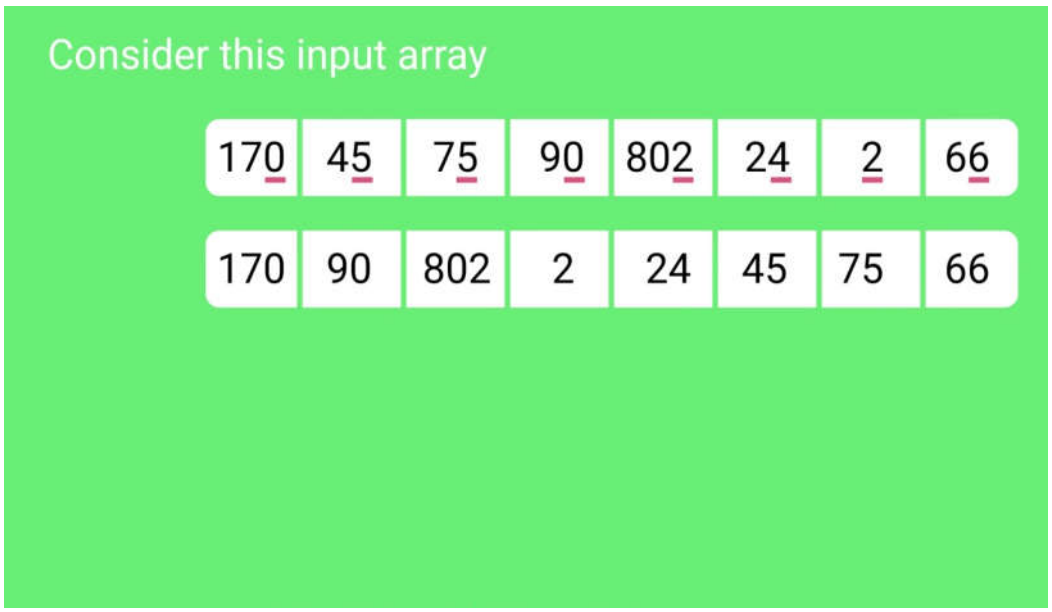
Radix Sort on Strings: Radix sort is mostly used to sort the numerical values or the real values, but it can be modified to sort the string values in lexicographical order. It follows the same procedure as used for numerical values.

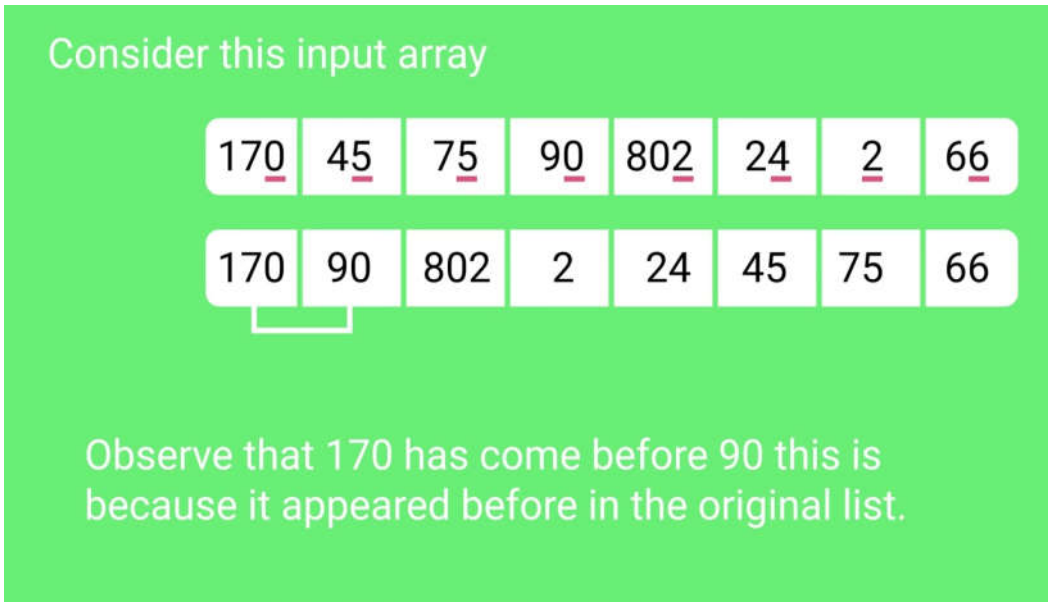
Please refer this IDE link for the implementation of the same.

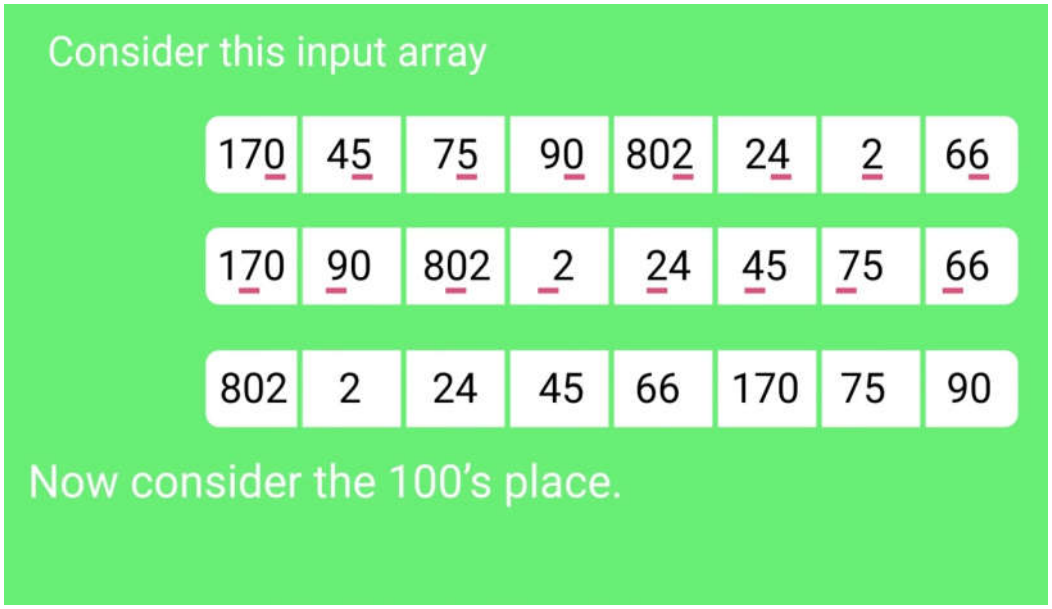
Output

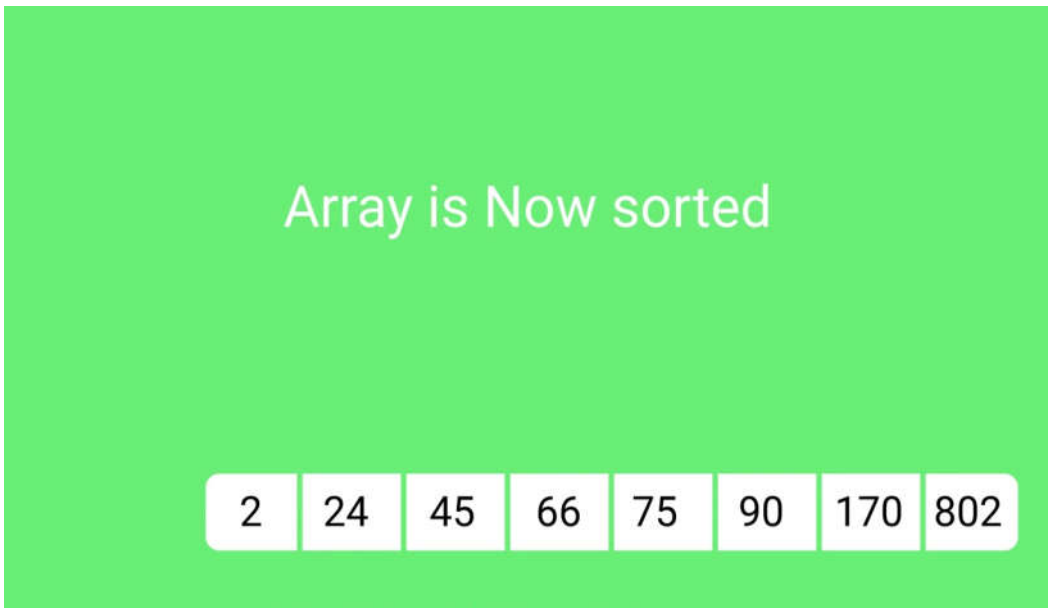
```
Input:[BCDEF, dbaqc, abcde, bbbbb]
Output:[abcde, bbbbb, BCDEF, dbaqc]
```











Complexity Analysis of Radix Sort:

Radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping the keys by the individual digits which share the same significant position and value. It has a time complexity of **O(d * (n + b))**, where d is the number of digits, n is the number of elements, and b is the base of the number system being used.

In practical implementations, radix sort is often faster than other comparison-based sorting algorithms, such as quicksort or merge sort, for large datasets, especially when the keys have many digits. However, its time complexity grows

from the need to create buckets for each digit value and to copy the elements back to the original array after each digit has been sorted.

Advantages:

- Radix sort has a linear time complexity, which makes it faster than comparison-based sorting algorithms such as quicksort and merge sort for large data sets.
- It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the sorted output.
- Radix sort is efficient for sorting large numbers of integers or strings.
- It can be easily parallelized.

Disadvantages:

- Radix sort is not efficient for sorting floating-point numbers or other types of data that cannot be easily mapped to a small number of digits.
- It requires a significant amount of memory to hold the count of the number of times each digit value appears.
- It is not efficient for small data sets or data sets with a small number of unique keys.
- It requires that the data being sorted can be represented in a fixed number of digits, which may not be the case for some types of data.

Quiz on Radix Sort

Check out DSA Self Paced Course

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- Quick Sort

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

Last Updated : 20 Apr, 2023

157

Recommended Problem

Radix Sort

Sorting Algorithms

Solve Problem

Submission count: 2.3K

Similar Reads

1. Radix Sort vs Bucket Sort
2. MSD(Most Significant Digit) Radix Sort
3. C Program For Radix Sort
4. Check if the number is even or odd whose digits and base (radix) is given
5. Comparison among Bubble Sort, Selection Sort and Insertion Sort
6. C/C++ Program for Odd-Even Sort / Brick Sort
7. Java Program for Odd-Even Sort / Brick Sort
8. Insertion sort to sort even and odd positioned elements in different orders
9. Sort an Array which contain 1 to N values in O(N) using Cycle Sort
10. sort() vs. partial_sort() vs. nth_element() + sort() in C++ STL

Related Tutorials

1. Learn Data Structures with Javascript | DSA Tutorial

3. Introduction to Set – Data Structure and Algorithm Tutorials

4. Introduction to Map – Data Structure and Algorithm Tutorials

5. What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : KOMAL Y, DrRoot_, rathbhupendra, abhijitjadhav1998, DishankJindal, m212076, vishwajeet0524, simranarora5sos, becharaerizk, sweetyty, unknown2108, kchandramani5265, sagar0719kumar, urvishajain50761, akashjha412, karthikns16, amartyaghoshgfg, yashmahajan3171, shreyasnaphad, guptavivek0503, phasing17, varunkansal21, snehalsalokhe, rutikb718, shivtejh2mx

Article Tags : DSA, Sorting

Practice Tags : Sorting

Improve Article | Report Issue

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Python

Python Tutorial

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

UPSC/SSC/BANKING

SSC CGL Syllabus

SBI PO Syllabus

IBPS PO Syllabus

UPSC Ethics Notes

UPSC Economics Notes

UPSC History Notes