

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN
(Autonomous)**

(Approved by AICTE, New Delhi and Permanently Affiliated to Andhra University,
Visakhapatnam) Madhurawada :: Visakhapatnam – 530 048

Department of Computer Sciences and Engineering

Make use of Radix sort algorithm to sort an array by individual digits, starting with the least significant digit.



A REPORT SUBMITTED

BY Team – 05

N. Harika ---- 324103210148

N. Pavithra ---- 324103210149

P. Thanuja ---- 324103210150

P. Laxmi Prasanna --- 324103210151

P. Raahithya --- 324103210152

Under the esteemed guidance

of

Mrs. D. Sarvani

Assistant Professor

CSE Department

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN
(Autonomous)**

Approved by AICTE NEW DELHI, Affiliated to Andhra University, Accredited by National Board of Accreditation (NBA) for B.Tech. CSE, ECE & IT - valid from 2019 - 22 and 2022 - 25 Accredited by National Assessment and Accreditation Council (NAAC) with A Grade - valid from 2022 - 2027 Kommadi, Madhurawada, Visakhapatnam – 530048.

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN
(Autonomous)**

(Approved by AICTE, New Delhi and Permanently Affiliated to Andhra University,
Visakhapatnam) Madhurawada :: Visakhapatnam – 530 048

Department of Computer Sciences and Engineering

2025 - 2026



This is to certify that the application report titled "**Make use of Radix sort algorithm to sort an array by individual digits, starting with the least significant digit.**" is a bonafide work of following II B. Tech I Semester students in the **Department of Computer Sciences And Engineering**, Gayatri Vidya Parishad College of Engineering for Women (Autonomous) during the academic year 2025 - 26.

BY

Team – 05

N. Harika ---- 324103210148

N. Pavithra ---- 324103210149

P. Thanuja ---- 324103210150

P. Laxmi Prasanna --- 324103210151

P. Raahithya --- 324103210152

Signature of the Guide

Mrs. D. Sarvani

Assistant Professor

Department of CSE

Signature of the External

HOD of CSE Department

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN
(Autonomous)**

(Approved by AICTE, New Delhi and Permanently Affiliated to Andhra University,
Visakhapatnam) Madhurawada :: Visakhapatnam – 530 048

Department of Computer Sciences and Engineering

2025 - 2026

INDEX

Sno.	TITLE	Page No.
1.	Aim: Make use of Radix sort algorithm to sort an array by individual digits, starting with the least significant digit.	4
2.	Description : Introduction to Radix Sort :	4
3.	1. Objectives of Radix Sort : 1.1 Primary Objectives 1.2 Additional Educational and Project Objectives	4 – 5
4.	Flowchart: STEP-BY-STEP FLOW EXPLANATION : Flowchart: LSD Radix Sort	5 -- 7
5.	Algorithm : Algorithm in Simple Steps :	8 -- 9
6.	Data Structures Used in Radix Sort:	9
7.	Source Code:	9 -- 13
8.	Outputs :	14 -- 15

AIM : Make use of Radix sort algorithm to sort an array by individual digits, starting with the least significant digit.

DESCRIPTION :

Introduction to Radix Sort :

Radix Sort is a stable, non-comparative, digit-wise sorting technique.

Instead of comparing elements directly (like in Quick Sort, Merge Sort, etc.), Radix Sort groups numbers by individual digits, using a stable sort at each stage.

There are two variants:

1. LSD Radix Sort (Least Significant Digit First) → Most common, covered here.
2. MSD Radix Sort (Most Significant Digit First) → Used in strings, words, etc.

In your project, you are implementing LSD Radix Sort, where sorting begins from the units place, then tens, then hundreds...

Example ordering of digits:

```
step 1: Sort by 1's place
Step 2: Sort by 10's place
Step 3: Sort by 100's place
...
...
```

1. Objectives of Radix Sort

When implementing Radix Sort in a project, clearly stating your objectives adds strong value. Below are detailed and enriched objectives:

1.1 Primary Objectives

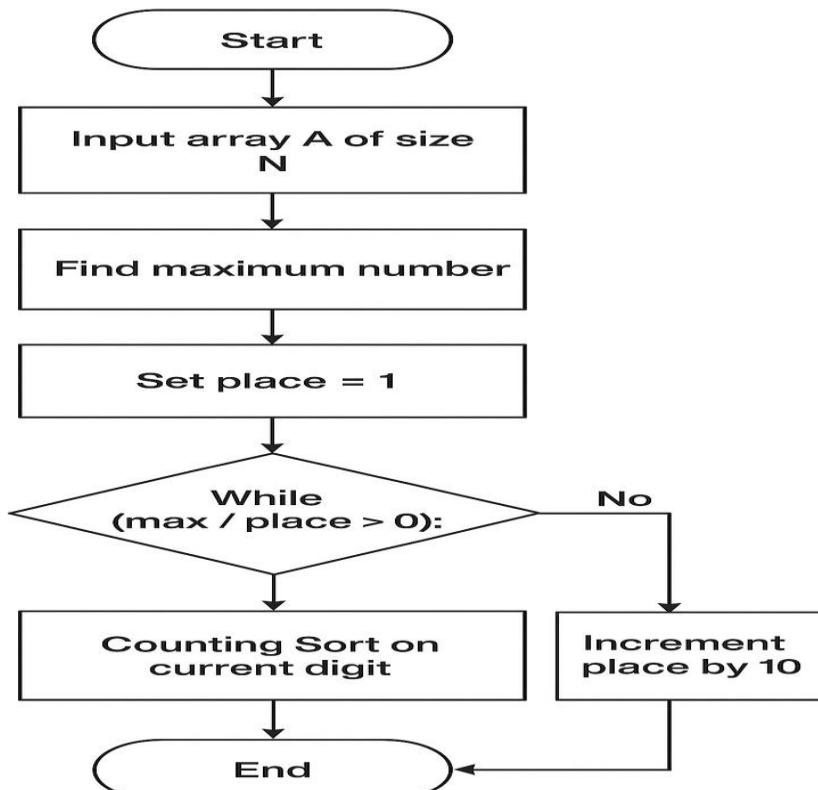
- To sort a list of integers digit by digit, starting from the Least Significant Digit (LSD) to the Most Significant Digit (MSD).
- To compare numbers without direct comparison operators, but by processing digits in each positional place.
- To demonstrate how stable sorting (typically Counting Sort) is used internally in each digit pass.

- To achieve more efficient sorting for large datasets of uniformly sized integers than traditional comparison-based algorithms.
- To highlight the role of base representation (radix) such as base-10 for decimal numbers or base-2 for binary integers.

1.2 Additional Educational and Project Objectives

- To study and implement a non-comparison based sorting algorithm.
- To understand digit extraction using mathematical operations.
- To analyze time complexity $O(d \times (n + k))$, where
 - d = number of digits
 - n = number of elements
 - k = range of digits (0–9 for decimal)
- To visually demonstrate sorting using flowcharts and diagrams.
- To explore real-time applications such as:
 - Sorting postal PIN codes
 - Sorting phone numbers
 - Internal use in digital hardware/processor operations

FLOW CHART:



STEP-BY-STEP FLOW EXPLANATION :

1. Start

The algorithm begins execution.

2. Input the Array

The user enters an array of integers.

Example:

[170, 45, 75, 90, 802, 24, 2, 66]

3. Find the Largest Number

To know how many digits we must process, the algorithm finds the maximum element.

Example: largest number = 802

→ It has 3 digits, therefore we will process 3 passes (units, tens, hundreds).

4. Set Position = 1

position = 1 represents the units (1's place).

Later:

position = 10 → tens

position = 100 → hundreds

and so on.

5. Repeat the Following Until $(\text{max} / \text{position}) > 0$

This loop keeps running until all digit positions are processed.

Step	Purpose
Find largest	Know total digit passes
position = 1	Start at least significant digit
Counting sort on each digit	Ensures stability
position *= 10	Move to next digit
Repeat until all digits processed	Guarantee full sorting

Flowchart: LSD Radix Sort

1. Start
2. Input Array A of size N

Example: $A = [170, 45, 75, 90, 802, 24, 2, 66]$

3. Find Maximum Number

Used to determine the number of digits (d).

$\max = 802 \rightarrow d = 3$ digits

4. Set place = 1 (units place)

5. Repeat While ($\max / \text{place} > 0$):

This loop processes each digit.

How Radix Sort Works Internally

Radix Sort relies on a helper stable algorithm, usually Counting Sort, applied digit-wise.

Digit Extraction Formula:

digit = (number / place) % 10

Where place = 1, 10, 100, 1000...

Example:

number = 276

place = 10

digit = $(276 / 10) \% 10 = 27 \% 10 = 7$

SMALL WORKED EXAMPLE:

Let the array be:

[170, 45, 75, 90]

PASS 1 → Units place

Sort by units:

[90, 170, 45, 75]

PASS 2 → Tens place

Sort by tens:

[170, 75, 45, 90]

PASS 3 → Hundreds place

Sort by hundreds:

[45, 75, 90, 170]

Final Output:

[45, 75, 90, 170]

Algorithm in Simple Steps :

RadixSort(array)

1. max = maximum number in array

2. place = 1

3. while max/place > 0:

 CountingSort(array, place)

 place = place * 10

Counting Sort for each digit:

CountingSort(array, place)

1. create count[10] = {0}

2. for each element:

 digit = (element/place)%10

 count[digit]++

3. Make count cumulative

4. Build output array (reverse order)

5. Copy output back to array

Radix" means **base** of a number system.

Example:

- Decimal numbers → base 10 (digits 0–9)
- Binary numbers → base 2 (digits 0–1)

Radix sort works for any base.

Radix sort never compares numbers directly.

Instead, it groups them by digits:

1's place → 10's place → 100's place → ...

☆ Advantages of Radix Sort

- ✓ Very fast for integers
 - ✓ Linear time for fixed digit size
 - ✓ Stable sorting
 - ✓ Works great with large datasets
-

☆ Disadvantages

- ✗ Needs extra memory (for counting).
- ✗ Works only with integers or fixed-length strings.
- ✗ More complex than simple sorting algorithms.

Real-Life Examples of Radix Sort:

1. **Sorting Phone Numbers** – Organizing contacts by number efficiently.
2. **Employee/Student IDs** – Arranging ID numbers in ascending order.
3. **Library Book Codes** – Sorting books using code numbers or ISBNs.
4. **Digital Dates** – Ordering dates in day/month/year format.
5. **Zip Codes** – Sorting mail items for postal delivery.

Data Structures Used in Radix Sort:

1. **Arrays** – To store elements and buckets for each digit.
2. **Queues** – To maintain the order of elements within each digit bucket for stability.

□ Arrays

- arr[] → Holds the original input numbers.
- positive[] and negative[] → Separate arrays for positive and negative numbers.
- output[] → Temporary array in countingSort to store sorted results by each digit.
- count[10] → Array for counting occurrences of digits (0–9) in counting sort.

□ Implicit Use of Indexing

- The program uses array indices to simulate the behavior of queues for stability when sorting digits.

So mainly: **Arrays** are the primary data structure, used for buckets, temporary storage, and maintaining stable ordering.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

int getMax(int arr[], int n) {
    int max = abs(arr[0]);
    for (int i = 1; i < n; i++) {
        if (abs(arr[i]) > max)
            max = abs(arr[i]);
    }
    return max;
}

void countingSort(int arr[], int n, int place) {
    int output[n];
    int count[10] = {0};
    int i;
    for (i = 0; i < n; i++) {
        int digit = (abs(arr[i]) / place) % 10;
        count[digit]++;
    }
    for (i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    for (i = n - 1; i >= 0; i--) {
        int digit = (abs(arr[i]) / place) % 10;
        output[count[digit] - 1] = arr[i];
        count[digit]--;
    }
    for (i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
```

```

    }

// LSD Radix Sort - Supports positive and negative numbers

void radixSort(int arr[], int n) {

    int positive[n], negative[n];

    int p = 0, q = 0;

    // Separate positive and negative numbers

    for (int i = 0; i < n; i++) {

        if (arr[i] >= 0)

            positive[p++] = arr[i];

        else

            negative[q++] = -arr[i]; // convert negative to positive form

    }

    if (p > 0) {

        int max = getMax(positive, p);

        for (int place = 1; max / place > 0; place *= 10) {

            printf("\nSorting positives at place %d:\n", place);

            countingSort(positive, p, place);

            for (int i = 0; i < p; i++)

                printf("%d ", positive[i]);

            printf("\n");

        }

    }

    if (q > 0) {

        int max = getMax(negative, q);

        for (int place = 1; max / place > 0; place *= 10) {

            printf("\nSorting negatives at place %d:\n", place);

            countingSort(negative, q, place);

            for (int i = 0; i < q; i++)

                printf("%d ", negative[i]);

        }

    }

}

```

```

printf("\n");
}

}

int index = 0;

for (int i = q - 1; i >= 0; i--) {
    arr[index++] = -negative[i];
}

for (int i = 0; i < p; i++) {
    arr[index++] = positive[i];
}

}

int main() {

    int choice, n, i;

    while (1) {

        printf("      RADIX SORT PROGRAM      \n");
        printf("1. Sort your own array\n");
        printf("2. Use sample test array\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1) {

            printf("\nEnter number of elements: ");
            scanf("%d", &n);

            int arr[n];

            printf("Enter elements (positive or negative):\n");
            for (i = 0; i < n; i++) {
                scanf("%d", &arr[i]);
            }

            printf("\nOriginal array: ");
            for (i = 0; i < n; i++)

```

```

        printf("%d ", arr[i]);
    radixSort(arr, n);
    printf("\n\nFinal Sorted Array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

else if (choice == 2) {
    int arr[] = {170, 45, 75, -90, 802, 24, -2, 66, -500};
    n = sizeof(arr) / sizeof(arr[0]);
    printf("\nUsing Sample Array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    radixSort(arr, n);
    printf("\n\nFinal Sorted Array: ");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

else if (choice == 3) {
    printf("\nExiting program. Thank you!\n");
    break;
}

else
    printf("\nInvalid choice! Try again.\n");
}

return 0;
}

```

OUTPUT :

```
C:\2nd year assignments\Data Structures>gcc radixsort.c -o radix.exe
C:\2nd year assignments\Data Structures>.\radix.exe
=====
                    RADIX SORT PROGRAM
=====
1. Sort your own array
2. Use sample test array
3. Exit
Enter your choice: 1

Enter number of elements: 10
Enter elements (positive or negative):
33
456
89
-345
678
213
-98
-67
765
119

Original array: 33 456 89 -345 678 213 -98 -67 765 119
```

```
Sorting positives at place 1:
33 213 765 456 678 89 119
```

```
Sorting positives at place 10:
213 119 33 456 765 678 89
```

```
Sorting positives at place 100:
33 89 119 213 456 678 765
```

```
Sorting negatives at place 1:
345 67 98
```

```
Sorting negatives at place 10:
345 67 98
```

```
Sorting negatives at place 100:
67 98 345
```

```
Final Sorted Array: -345 -98 -67 33 89 119 213 456 678 765
```

```
=====
          RADIX SORT PROGRAM
=====
1. Sort your own array
2. Use sample test array
3. Exit
Enter your choice: 2

Using Sample Array:
170 45 75 -90 802 24 -2 66 -500
Sorting positives at place 1:
170 802 24 45 75 66

Sorting positives at place 10:
802 24 45 66 170 75

Sorting positives at place 100:
24 45 66 75 170 802

Sorting negatives at place 1:
90 500 2

Sorting negatives at place 10:
500 2 90

Sorting negatives at place 100:
2 90 500

Final Sorted Array: -500 -90 -2 24 45 66 75 170 802
```

```
=====
          RADIX SORT PROGRAM
=====
1. Sort your own array
2. Use sample test array
3. Exit
Enter your choice: 3

Exiting program. Thank you!
```

```
C:\2nd year assignments\Data Structures>gcc radixsort.c -o radix.exe
C:\2nd year assignments\Data Structures>.\radix.exe
=====
          RADIX SORT PROGRAM
=====
1. Sort your own array
2. Use sample test array
3. Exit
Enter your choice: 4

Invalid choice! Try again.
```