

## Radix Sort

Last Updated : 30 Sep, 2025

Edit

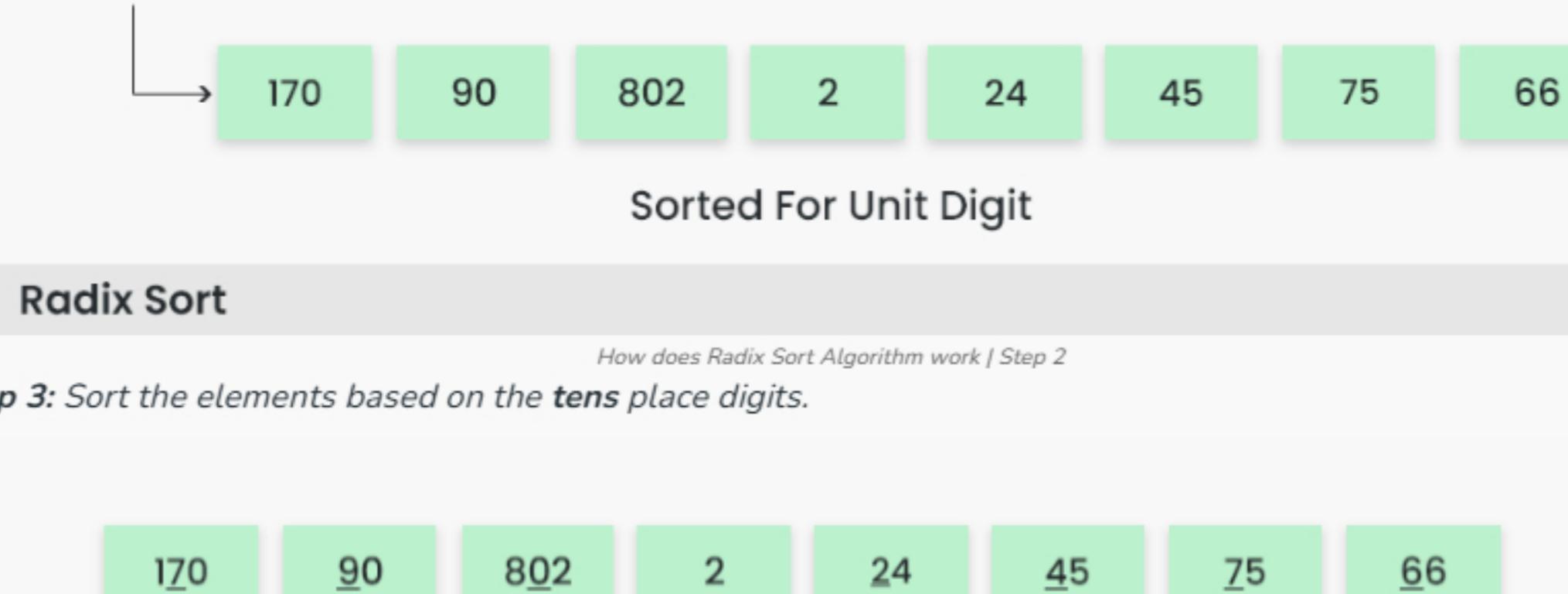
Radix Sort is a linear sorting algorithm (for fixed length digit counts) that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys.

- It repeatedly distributes the elements into buckets based on each digit's value. This is different from other algorithms like Merge Sort or Quick Sort where we compare elements directly.
- By repeatedly sorting the elements by their significant digits, from the least significant to the most significant, it achieves the final sorted order.
- We use a stable algorithm like Counting Sort to sort the individual digits so that the overall algorithm remains stable.

To perform radix sort on the array [170, 45, 75, 90, 802, 24, 2, 66], we follow these steps:

Step 1: Find the largest element, which is 802. It has three digits, so we will iterate three times.

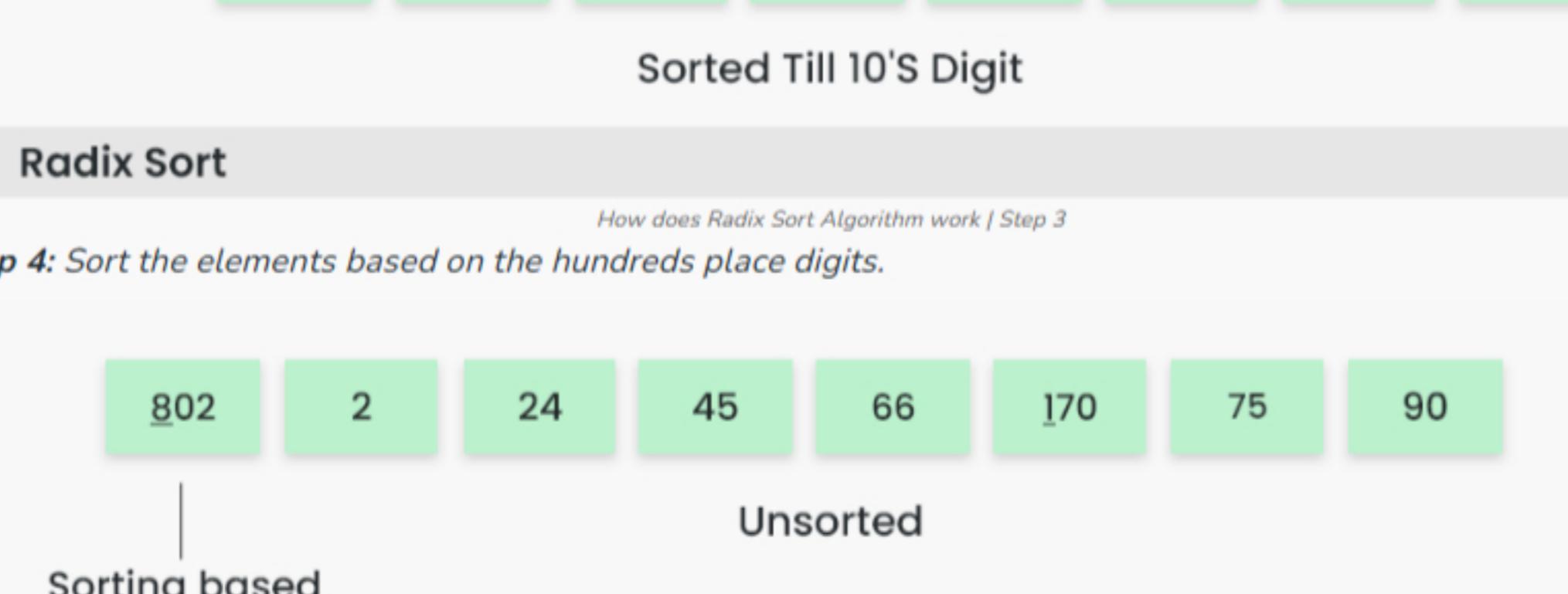
Step 2: Sort the elements based on the unit place digits (X=0).



## Radix Sort

How does Radix Sort Algorithm work | Step 2

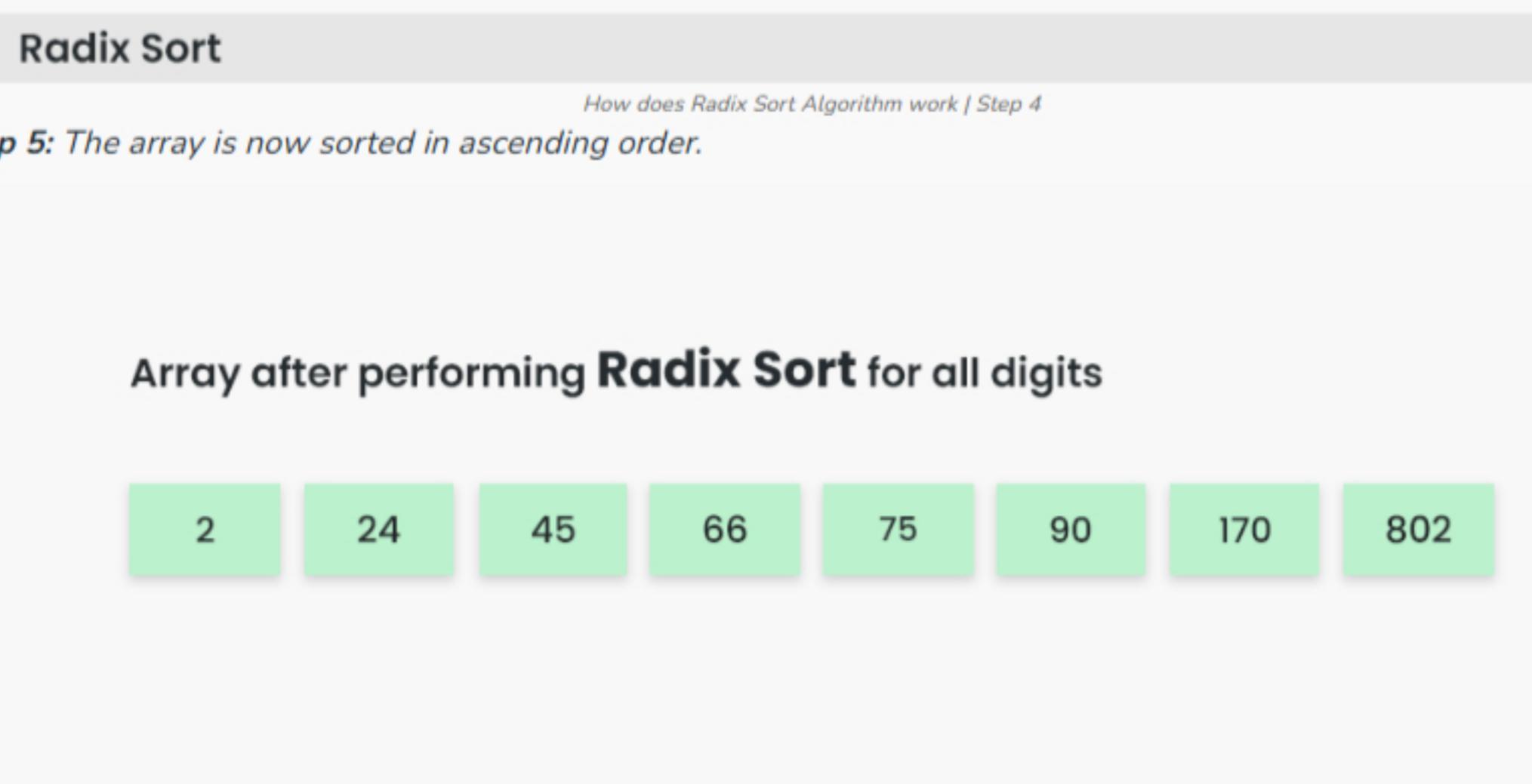
Step 3: Sort the elements based on the tens place digits.



## Radix Sort

How does Radix Sort Algorithm work | Step 3

Step 4: Sort the elements based on the hundreds place digits.



## Radix Sort

How does Radix Sort Algorithm work | Step 4

Step 5: The array is now sorted in ascending order.

## Array after performing Radix Sort for all digits



## Radix Sort

How does Radix Sort Algorithm work | Step 5

## Try it on GfG Practice

Below is the implementation for the above illustrations:

C++ C Java Python C# JavaScript PHP Dart

```
# 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[index % 10] = 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 >= 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 Kumar
# Edited by Patrick Gallagher
```

## Output

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

## Complexity Analysis of Radix Sort:

## Time Complexity:

- 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 linearly with the number of digits, and so it is not as efficient for small datasets.

## Auxiliary Space:

- Radix sort also has a space complexity of  $O(n + b)$ , where  $n$  is the number of elements and  $b$  is the base of the number system. This space complexity comes 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.

Please refer to [Complexity Analysis of Radix Sort](#) for more details.

Comment kartik + Follow

189

Article Tags : [Sorting](#) [DSA](#)

## Explore

## DSA Fundamentals

## Data Structures

## Algorithms

## Advanced

## Interview Preparation

## Practice Problem

## Tutorials

## Programming Languages

## AI, ML &amp; Data Science

## DevOps &amp; Cloud

## CS Core Subjects

## Interview Preparation

## Software and Tools

## Courses

## IBMs Platification

## Web and Development

## Programmung Languages

## DevOps &amp; Cloud

## Trending Technologies

## Software and Tools

## Interview Preparation

## Software and Tools

## Trending Technologies

## Software and Tools