# Experiment No. – 03

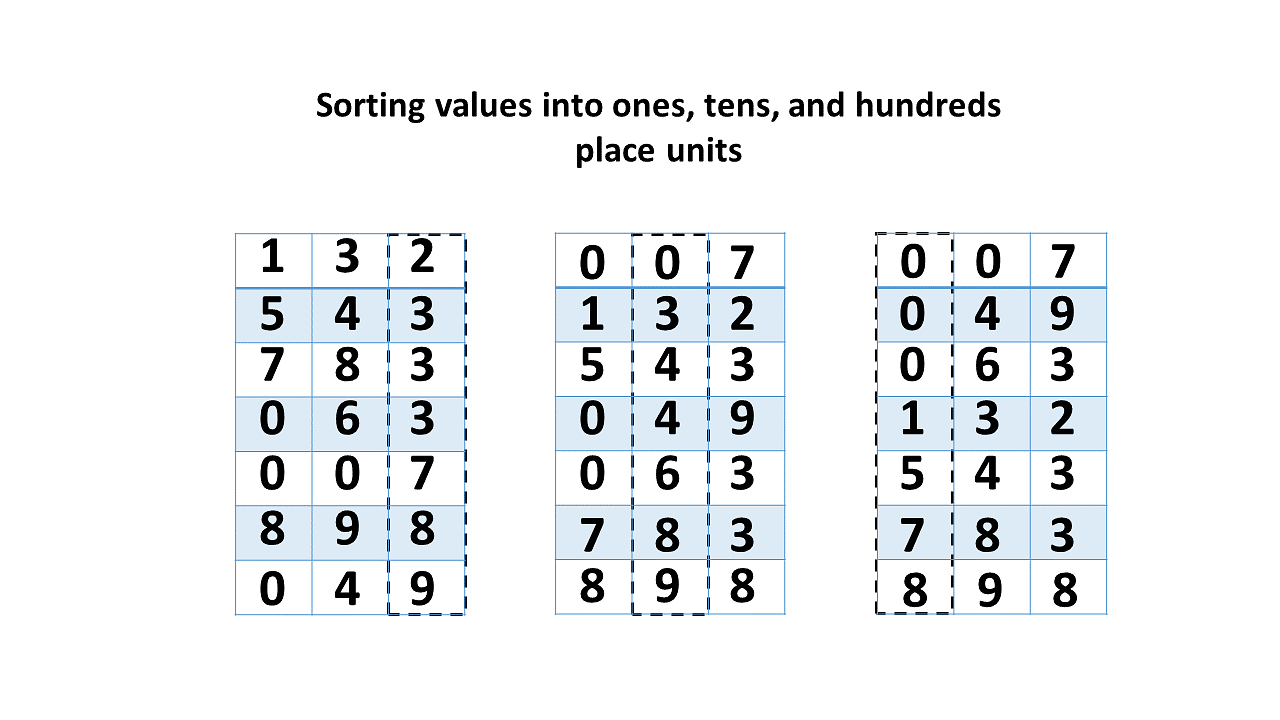
**Aim:** Study of Radix Sort Algorithm using CUDA.

**Objective:** The primary objective of this experiment is to implement and analyze the parallel version of the Radix Sort algorithm using CUDA. The focus is on understanding how parallelization can exploit the parallel processing power of GPUs to improve the sorting performance, especially for large datasets.

**Theory:**

**Radix Sort Algorithm**

* Radix sort algorithm is a non-comparative sorting algorithm in computer science. It avoids comparison by creating and categorizing elements based on their radix. For elements with more than one significant digit, it repeats the bucketing process for each digit while preserving the previous step's ordering until all digits have been considered.
* Radix Sort is a linear sorting algorithm. Radix Sort's time complexity of O(nd), where n is the size of the array and d is the number of digits in the largest number.
* It is not an in-place sorting algorithm because it requires extra space.
* Radix Sort is a stable sort because it maintains the relative order of elements with equal values.
* Radix sort algorithm may be slower than other sorting algorithms such as merge sort and Quicksort if the operations are inefficient. These operations include sub-inset lists and delete functions, and the process of isolating the desired digits.
* Because it is based on digits or letters, radix sort is less flexible than other sorts. If the type of data changes, the Radix sort must be rewritten.



After defining the radix sort algorithm, you will look at how it works with an example.

**Working of Radix Sort Algorithm:**

* The Radix sort algorithm works by ordering each digit from least significant to most significant.
* In base 10, radix sort would sort by the digits in the one's place, then the ten's place, and so on.
* To sort the values in each digit place, Radix sort employs counting sort as a subroutine.
* This means that for a three-digit number in base 10, counting sort will be used to sort the 1st, 10th, and 100th places, resulting in a completely sorted list. Here's a rundown of the counting sort algorithm.

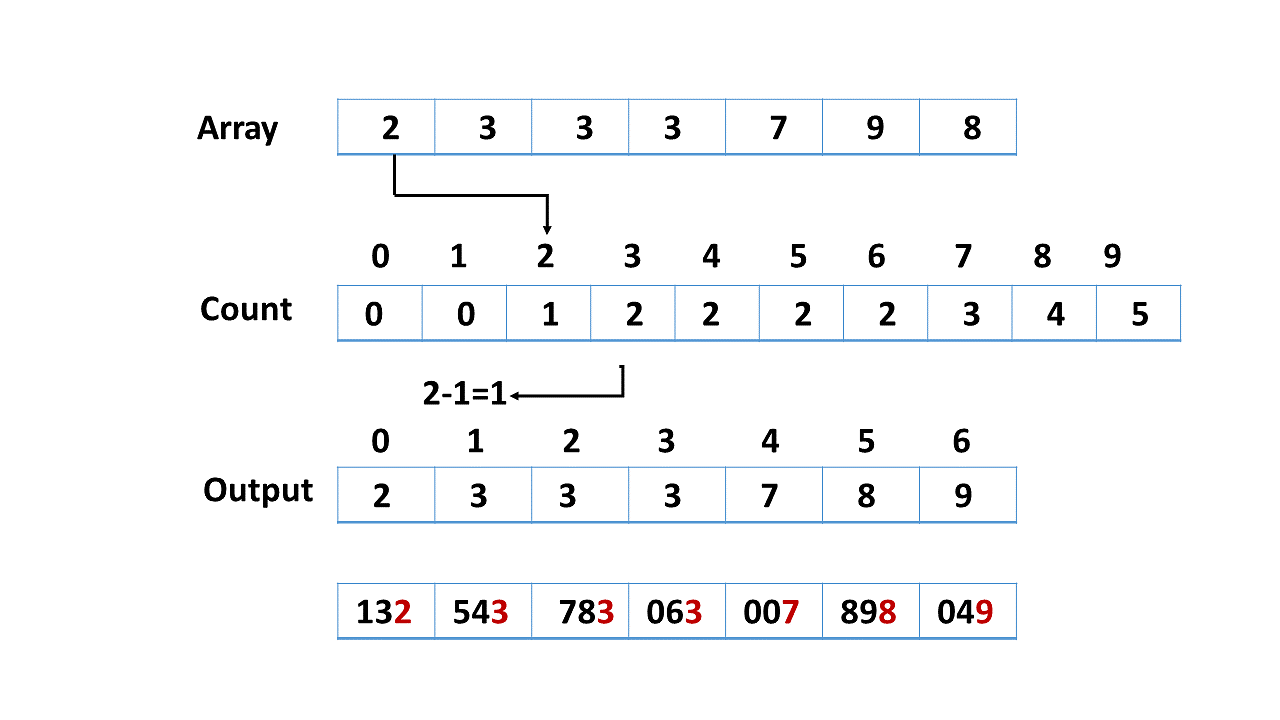
Assume you have an 8-element array. First, you will sort the elements by the value of the unit place. It will then sort the elements based on the value of the tenth position. This process is repeated until it reaches the last significant location.

Let's start with [132, 543, 783, 63, 7, 49, 898]. It is sorted using radix sort, as illustrated in the figure below.

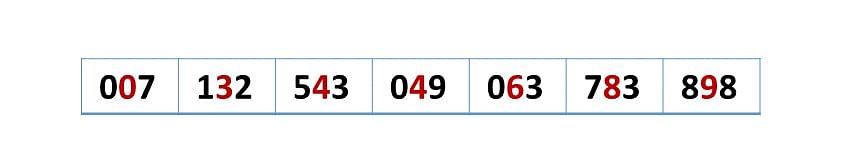
* Find the array's largest element, i.e., maximum. Consider A to be the number of digits in maximum. A is calculated because we must traverse all of the significant locations of all elements.

The largest number in this array [132, 543, 783, 63, 7, 49, 898] is 898. It has three digits. As a result, the loop should be extended to hundreds of places (3 times).

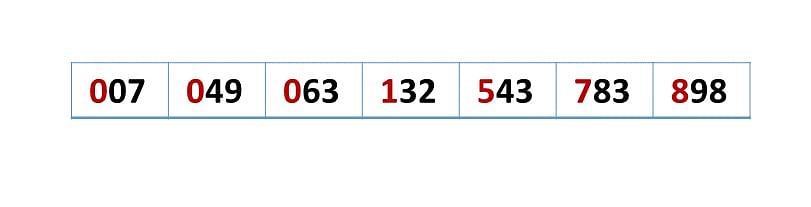
* Now, go through each significant location one by one. Sort the digits at each significant place with any stable sorting technique. You must use counting sort for this. Sort the elements using the unit place digits (A = 0).



* Sort the elements now by digits in the tens place.



* Finally, sort the elements by digits in the hundreds place.



## **Code Implementation of Radix Sort Algorithm:**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int Max\_value(int Array[], int n) // This function gives maximum value in array[]

{

int i;

int maximum = Array[0];

for (i = 1; i < n; i++)

{

if (Array[i] > maximum)

maximum = Array[i];

}

return maximum;

}

void radixSortalgorithm(int Array[], int n) // Main Radix Sort sort function

{

int i, digitPlace = 1;

int result\_array[n]; // resulting array

int largest = Max\_value(Array, n); // Find the largest number to know number of digits

while (largest / digitPlace > 0)

{

int count\_array[10] = {0};

for (i = 0; i < n; i++) // Store the count of "keys" or digits in count[]

count\_array[(Array[i] / digitPlace) % 10]++;

for (i = 1; i < 10; i++)

count\_array[i] += count\_array[i - 1];

for (i = n - 1; i >= 0; i--) // Build the resulting array

{

result\_array[count\_array[(Array[i] / digitPlace) % 10] - 1] = Array[i];

count\_array[(Array[i] / digitPlace) % 10]--;

}

for (i = 0; i < n; i++) // numbers according to current digit place

Array[i] = result\_array[i];

digitPlace \*= 10; // Move to next digit place

}

}

void displayArray(int Array[], int n) // Function to print an array

{

int i;

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

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

printf("\n");

}

int main()

{

int array1[] = {20, 30, 40, 90, 60, 100, 50, 70};

int n = sizeof(array1) / sizeof(array1[0]);

printf("Unsorted Array is : ");

displayArray(array1, n);

radixSortalgorithm(array1, n);

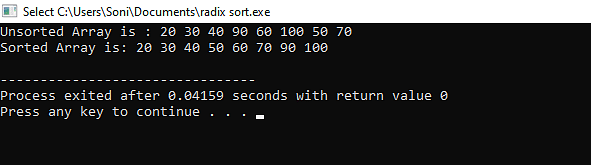
printf("Sorted Array is: ");

displayArray(array1, n);

return 0;

}

**Output:**



**Conclusion:** In conclusion, implementing Radix Sort in CUDA can provide significant performance gains when sorting large datasets compared to traditional CPU-based implementations. The parallel processing power of NVIDIA GPUs allows for efficient processing of multiple elements simultaneously, making Radix Sort an attractive algorithm for CUDA programming.