## Experiment – 5

**Student Name:** Ayush Ranjan
**UID:** 23BCS10187
**Branch:** BE-CSE
**Section/Group:** KRG-2-B
**Semester:** 5th
**Date of Performance:** 07/8/25
**Subject Name:** DAA
**Subject Code:** 23CSH-301

## 1. Aim:
Sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted. The elements can be read from a file or can be generated using the random number generator.

## 2. Procedure:
Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick Sort that pick pivot in different ways.

- Always pick first element as pivot.
- Always pick last element as pivot (implemented below)
- Pick a random element as pivot.
- Pick median as pivot.

The key process in quick Sort is partition (). Target of partition() is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

## 3. Code:

```
#include <iostream>
using namespace std;

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}
```

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) cin >> arr[i];
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    return 0;
  }
```

## 4. Output:

```
Enter number of elements: 6
Enter elements: 45 12 89 33 22 10
Sorted array: 10 12 22 33 45 89
```

## 5. Learning Outcomes:

- Gained understanding of the divide and conquer strategy used in sorting algorithms
- Learned how partitioning helps in efficient data arrangement.
- Developed skills in recursive problem-solving using C++.
- Understood time complexity implications of Quick Sort.
- Gained practical experience in implementing and testing sorting algorithms.