Experiment-5

Student Name: Tejinderpal Singh **UID:** 23BCS80343
Branch: BE-CSE **Section/Group:** 23BCS_KRG_3A
Semester: 5th **Subject Code:** 23CSH-301
Subject Name: DAA

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. **Objective:** To understand and implement the Quick Sort algorithm in C++ for sorting a set of elements. The objective is to help students learn the divide-and-conquer strategy, efficient in-place sorting, pivot selection, recursive function implementation, and analysis of time complexity in best, average, and worst-case scenarios, thereby strengthening their understanding of sorting algorithms and their practical applications in data handling.

3. **Procedure:**
   1. Start.
   2. Define a swap() function to exchange two integers.
   3. Define partition() function:
      - Choose the last element as pivot.
      - Initialize index i = low - 1.
      - Traverse from low to high - 1:
      - If element < pivot, increment i and swap with array[i].
      - Swap array[i + 1] with pivot.
      - Return i + 1 as pivot index.

   4. Define quickSort() function:
      - If low < high:
      - Call partition() to get pivot index pi.
      - Recursively call quickSort() for subarray before pivot (low to pi - 1).
      - Recursively call quickSort() for subarray after pivot (pi + 1 to high).

   5. In main(), initialize the array, call quickSort(), and display the sorted array.
   6. End.

4. Code:

```
#include <iostream>
using namespace std;
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}
int partition(int array[], int low, int high) {
    int pivot = array[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (array[j] < pivot) {
            i++;
            swap(array[i], array[j]);
        }
    }
    swap(array[i + 1], array[high]);
    return i + 1;
}

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

int main() {
    int data[] = {5, 2, 9, 1, 7, 3};
    int n = sizeof(data) / sizeof(data[0]);
    quickSort(data, 0, n - 1);
    for (int i = 0; i < n; i++) {
        cout << data[i] << " ";
    }
    return 0;
}
```

5. Observations:

```
1 2 3 5 7 9
c:\Users\singh\Desktop\VS Code\CPP>
```

6. Time Complexity:

Case	Time Complexity
Best Case	O(n log n)
Average Case	O(n log n)
Worst Case	O(n^2)

7. Learning Outcome:

- ❖ Learned how to implement the Quick Sort algorithm using C++ functions.
- ❖ Understood the divide-and-conquer strategy and the role of partitioning in sorting.
- ❖ Gained practical experience in recursively sorting subarrays around a pivot.
- ❖ Strengthened understanding of array manipulation and in-place sorting techniques.
- ❖ Learned to analyze the time complexity of sorting algorithms in best, average, and worst cases.