Experiment - 5

Student Name: Diksha **UID:** 23BCS10994
Branch: BE-CSE **Section/Group:** KRG-2B
Semester: 5th **Date of Performance:** 9/10/25
Subject Name: Design and Analysis of Algorithms
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. **Objective:** To understand and implement quick sort.

3. **Procedure/Algorithm:**

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.

QUICKSORT

Quicksort is a sorting algorithm whose worst-case running time is ($n2$) on an input array of n numbers. In spite of this slow worst-case running time, quicksort is often the best practical choice for sorting because it is remarkably efficient on the average: its expected running time is (n lg n), and the constant factors hidden in the (n lg n) notation are quite

small. It also has the advantage of sorting in place (see page 3), and it works well even in virtual memory environments.

The following procedure implements quicksort.

5. Code and

Output-#include

<bits/stdc++.h> using

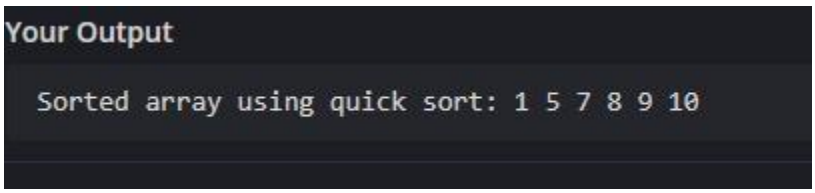
namespace std;

```
int partition(vector<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(vector<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() { vector<int> arr = {10, 7,
    8, 9, 1, 5}; quickSort(arr, 0,
    arr.size() - 1); for (int x : arr) cout
    << x << " ";
}
```



6. **Time Complexity:**

Best case	Average case	Worst case
O (nlogn)	O (nlogn)	O (n2)