



Experiment – 5

Student Name: Himanshu Gupta

Branch: BE-CSE

Semester: 5th

Subject Name: DAA

UID: 23BCS10889

Section/Group: KRG-2B

Date of Performance: 9/10/25

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 quick sort.

3. Input/Apparatus Used:

Quick sort concept is used.

4. 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 (n^2) 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—

```
import java.util.*;

public class QuickSort {

    static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);

            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    static 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++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
System.out.print("Enter number of elements: ");
int n = sc.nextInt();

int[] arr = new int[n];
System.out.print("Enter elements: ");
for (int i = 0; i < n; i++) {

    arr[i] = sc.nextInt();
}

quickSort(arr, 0, n - 1);

System.out.println("Sorted array: " + Arrays.toString(arr));
sc.close();
}
```

```
Enter number of elements: 5
Enter elements: 20 50 89 52 1
Sorted array: [1, 20, 50, 52, 89]
PS C:\Users\ASUS\Desktop\Sem 5\DAA_23BCS10889_KRG-2B>
```

6. Time Complexity:

Best case	Average case	Worst case
$O(n \log n)$	$O(n \log n)$	$O(n^2)$