K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: A3	**Roll. No.: 16010121045**
Experiment: 11	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of sorting algorithms

Objective: To understand various searching methods

Expected Outcome of Experiment:

CO	Outcome
CO3	Demonstrate sorting and searching methods.

Websites/books referred:

Abstract: -

(Define sorting as a concept,)

Sorting refers to ordering data in an increasing or decreasing fashion according to some linear relationship among the data items.

Sorting can be done on names, numbers and records. Sorting reduces the For example, it is relatively easy to look up the phone number of a friend from a telephone dictionary because the names in the phone book have been sorted into alphabetical order.

This example clearly illustrates one of the main reasons that sorting large quantities of information is desirable. That is, sorting greatly improves the efficiency of searching. If we were to open a phone book, and find that the names were not presented in any logical order, it would take an incredibly long time to look up someone's phone number.

Related Theory: *(Explain stable sort, in place sort, number of passes in a sorting algo)*

A stable sort is one which preserves the original order of the input set, *where the comparison algorithm does not distinguish between two or more items* .Consider a sorting algorithm that sorts cards by *rank*, but not by suit. The stable sort will guarantee that the original order of cards having the same rank is preserved; the unstable sort will not.

An in-place sorting algorithm uses constant space for producing the output (modifies the given array only). It sorts the list only by modifying the order of the elements within the list. For example, Insertion Sort and Selection Sorts are in-place sorting algorithms as they do not use any additional space for sorting the list and a typical implementation of Merge Sort is not in-place, also the implementation for counting sort is not an in-place sorting algorithm.

In algorithm, every iteration through each element of a list is called a pass. For a list with n elements, the bubble sort makes a total of n – 1 passes to sort the list. In each pass, the required pairs of adjacent elements of the list will be compared.

Assigned Sorting Algorithm: *(Bubble/insertion/counting)*

All Three

Code and output screenshots for assigned sorting algorithm:

```
#include <bits/stdc++.h>
using namespace std;
void bubbleSort(int *arr, int n)
{
    for (int i = n - 1; i >= 0; i--)
    {
        for (int j = 0; j < i; j++)
        {
            // comparing the two elements and swaping if
greater
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
void insertionSort(int *arr, int n)
{
    int j;
    for (int i = 1; i < n; i++)
    {
        int key = arr[i];
        j = i - 1;
        // This shifts elements till they are > key
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

```
void countingSort(int a[], int n) // function to perform
counting sort
{
    int max = a[0];
    // Max
    for(int i=1;i<n;i++)
        if(max<a[i])
            max=a[i];
    int count[max + 1];
    int output[n + 1];
    // Initialize with 0
    for (int i = 0; i <= max; ++i)
        count[i] = 0;
    // Hashing
    for (int i = 0; i < n; i++)
        count[a[i]]++;
    // Adding count values
    for (int i = 1; i <= max; i++)
        count[i] += count[i - 1];
    // Making sorted array
    for (int i = n - 1; i >= 0; i--)
    {
        output[count[a[i]] - 1] = a[i];
        count[a[i]]--;
    }
    // copy
    for (int i = 0; i < n; i++)
        a[i] = output[i];
}
int main()
{
    int arr[]={10,9,8,7,6,5,4,3,2,1};
    cout<<"Sorting using Bubble Sort"<<endl;
    bubbleSort(arr,10);
    for (int i = 0; i < 10; i++)
        cout << arr[i] << endl;
    cout<<"Sorting using Insertion Sort"<<endl;
```

```
    insertionSort(arr,10);
    for (int i = 0; i < 10; i++)
        cout << arr[i] << endl;
        cout<<"Sorting using Counting Sort"<<endl;
    countingSort(arr,10);
    for (int i = 0; i < 10; i++)
        cout << arr[i] << endl;
    return 0;
}
```

Output:

```
pargat@Router Code % cd "/Use
Sorting using Bubble Sort
1
2
3
4
5
6
7
8
9
10
Sorting using Insertion Sort
1
2
3
4
5
6
7
8
9
10
Sorting using Counting Sort
1
2
3
4
5
6
7
8
9
10
pargat@Router Code %
```

Conclusion: -

Successfully executed and performed the given sorting algorithms.

Post lab questions-

a. **Compare and contrast various sorting algorithms.**

Sorting Algorithms	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(1)
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n + k)
Bucket Sort	O(n + k)	O(n + k)	O(n^2)	O(n)

b. **Comment on the input (sorted in ascending order/descending order/random) and time required for execution of sorting algorithm.**

For sorting an array in either ascending or descending order the only parameter that changes in the whole algorithm is the comparator function, i.e the function which compares and decides where and how to place the given elements. The whole sorting part is depended on this comparator function. When it comes to the time required for the execution of the sorting, it depends upon the sorting algorithm used and the time complexity of that algorithm. Merge sort being the best out of the all.