Aim: Sorting Techniques.

Objectives: The main objective of this assignment is to understand and implement various sorting techniques such as bubble sort, insertion sort, selection sort, quick sort, radix sort, merge sort in C++. The goal is to understand how different sorting algorithms can be used to sort the given array.

Tools Used: Online C++ Compiler.

Concept:

Bubble sort is a sorting algorithm that repeatedly steps through the array, compares adjacent elements, and swaps them if they are in the wrong order. This process repeats until the array is sorted.

Process:

1.) Start from the first element of the array.

2.) Compare each pair of adjacent elements.

3.) If the first is greater than the second, swap them.

4.) Continue this process for each pair, moving the largest unsorted element to the end.

5.) Repeat the process for the remaining unsorted elements.

6.) Stop when no swaps are needed.

Example:

Array: [5, 2, 9, 1, 5]

1.) Compare 5 and 2 → Swap → [2, 5, 9, 1, 5]

2.) Compare 5 and 9 → No swap → [2, 5, 9, 1, 5]

3.) Compare 9 and 1 → Swap → [2, 5, 1, 9, 5]

4.) Compare 9 and 5 → Swap → [2, 5, 1, 5, 9] The largest element (9) is now in place.

Repeat the process for the remaining elements until sorted.

Insertion sort is a sorting algorithm that builds a sorted array one element at a time by comparing each new element to the already sorted elements and inserting it into its correct position.

Process:

1.) Start from the second element (index 1).

2.) Compare it to the previous elements.

3.) Move it to the correct position by shifting larger elements to the right.

4.) Repeat for each element until the entire array is sorted.

Example:

Array: [4, 2, 7, 1]

1.) Compare 2 with 4 → Insert 2 before 4 → [2, 4, 7, 1]

2.) Compare 7 with 4 → No change → [2, 4, 7, 1]

3.) Compare 1 with 7, 4, and 2 → Insert 1 at the beginning → [1, 2, 4, 7]

The array is now sorted.

Selection sort is a sorting algorithm that repeatedly selects the smallest (or largest) element from the unsorted portion of the array and swaps it with the first unsorted element.

Process:

1.) Start with the first element.

2.) Find the smallest element in the remaining unsorted part.

3.) Swap it with the first unsorted element.

4.) Move to the next element and repeat the process for the remaining unsorted array.

5.) Continue until the array is fully sorted.

Example:

Array: [3, 1, 5, 2]

1.) Find the smallest element (1) → Swap with the first element → [1, 3, 5, 2]

2.) Find the smallest in the remaining array (2) → Swap with the second element → [1, 2, 5, 3]

3.) Find the smallest in the remaining array (3) → Swap with the third element → [1, 2, 3, 5]

The array is now sorted.

Quick sort is a sorting algorithm that uses a divide-and-conquer approach. It selects a "pivot" element, partitions the array into two sub-arrays (elements smaller than the pivot and elements larger), and then recursively sorts the sub-arrays.

Process:

1.) Choose a pivot element from the array.

2.) Partition the array so that elements smaller than the pivot are on the left, and elements larger are on the right.

3.) Recursively apply the same process to the left and right sub-arrays.

4.) Repeat until the sub-arrays contain only one element or are empty.

Example:

Array: [45, 23, 12, 89, 34, 67]

1.) Choose 45 as the pivot.

Partition → [23, 12, 34] (smaller than 45), [89, 67] (larger than 45) → [34, 23, 12, 45, 89, 67]

2.) Recursively apply quicksort to [34, 23, 12]

Pivot 34 → [23, 12] (smaller), no larger → [12, 23, 34]

3.) Recursively apply quicksort to [23, 12]

Pivot 12 → [12, 23]

4.) Recursively apply quicksort to [89, 67]

Pivot 89 → [67], no larger → [67, 89]

5.) Combine →

[12, 23, 34, 45, 67, 89]

The array is now sorted.

Radix sort is a non-comparative sorting algorithm that sorts numbers digit by digit, starting from the least significant digit to the most significant digit, using a stable sorting algorithm like counting sort at each step.

Process:

1.) Find the maximum number in the array to determine the number of digits.

2.) Sort the numbers based on the least significant digit (ones place).

3.) Move to the next significant digit (tens place) and repeat the sorting.

4.) Continue this process until all digits are sorted.

Example:

Array: [170, 45, 75, 90, 802]

1.) Sort by ones place → [170, 90, 802, 45, 75]

2.) Sort by tens place → [802, 45, 75, 170, 90]

3.) Sort by hundreds place → [45, 75, 90, 170, 802]

The array is now sorted.

Merge sort is a divide-and-conquer sorting algorithm that recursively splits the array into two halves, sorts each half, and then merges the sorted halves back together.

Process:

1.) Divide the array into two halves until each subarray contains only one element.

2.) Recursively merge the subarrays by comparing their elements and combining them into a sorted array. 3.) Repeat this merging process until the entire array is merged and sorted.

Example:

Array: [38, 27, 43, 3, 9]

1.) Divide into [38, 27] and [43, 3, 9]

2.) Divide further: [38], [27], [43], [3], [9]

3.) Merge: [27, 38], [3, 9, 43]

4.) Merge the two halves: [3, 9, 27, 38, 43]

The array is now sorted.

Problem Statement:

1.) Implement bubble sort.

2.) Implement insertion sort.

3.) Implement selection sort.

4.) Implement quick sort.

5.) Implement radix sort.

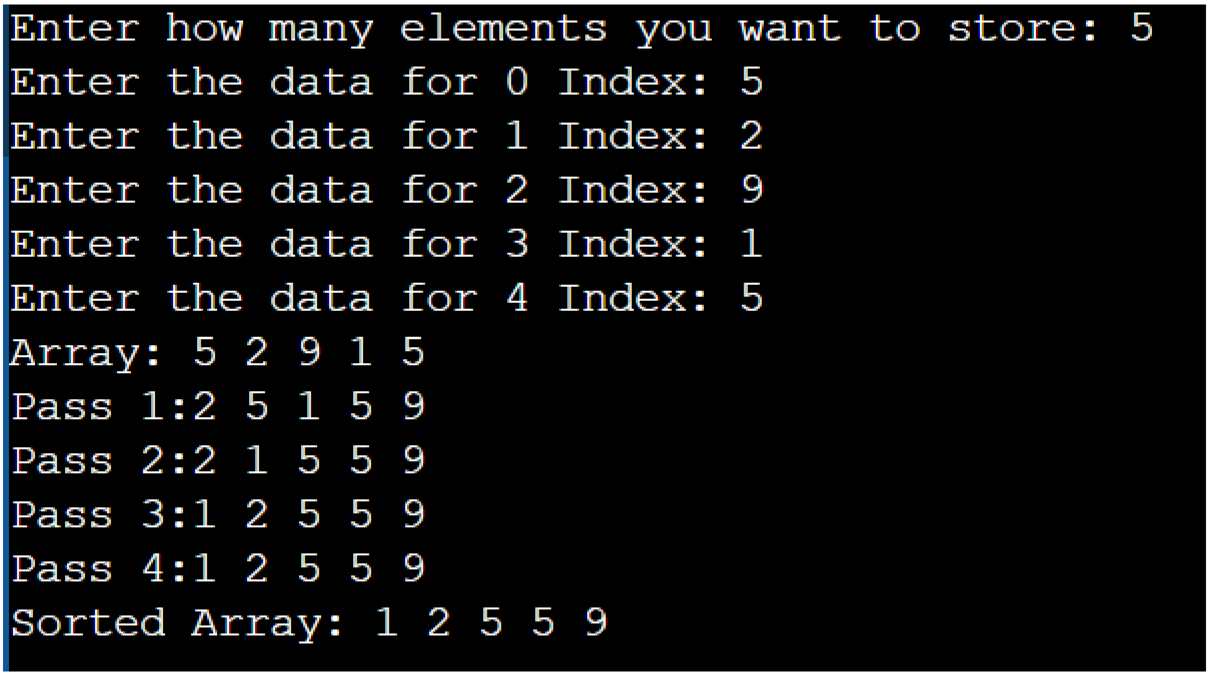
6.) Implement merge sort.

Solution:

1.) Bubble Sort.

```
#include <iostream>
#define max 100
using namespace std;
int n,arr[max],pass=1;

class BubbleSort {
public:
        void input()
        {
                cout<<"Enter how many elements you want to store: ";
                cin>>n;
                for (int i=0; i<n; i++)
                {
                    cout<<"Enter the data for "<<i <<" Index: ";
                    cin>>arr[i];
                }
                cout << "Array: ";
                for(int i = 0; i < n; i++) {
                    cout << arr[i] << " ";
                }
                cout << endl;
        }
        void bsort() {
                for(int i = 0; i < n-1; i++) {
                    for(int j = 0; j < n-i-1; j++) {
                        if(arr[j] > arr[j+1]) {
                            int temp = arr[j];
                            arr[j] = arr[j+1];
                            arr[j+1] = temp;
                        }
                    }
                    cout << "Pass " <<  pass << ":";
                    for(int i = 0; i< n ; i++) {
                        cout << arr[i] << " ";
                    }
                    pass++;
                    cout << endl;


                }
        }
        void display() {
```

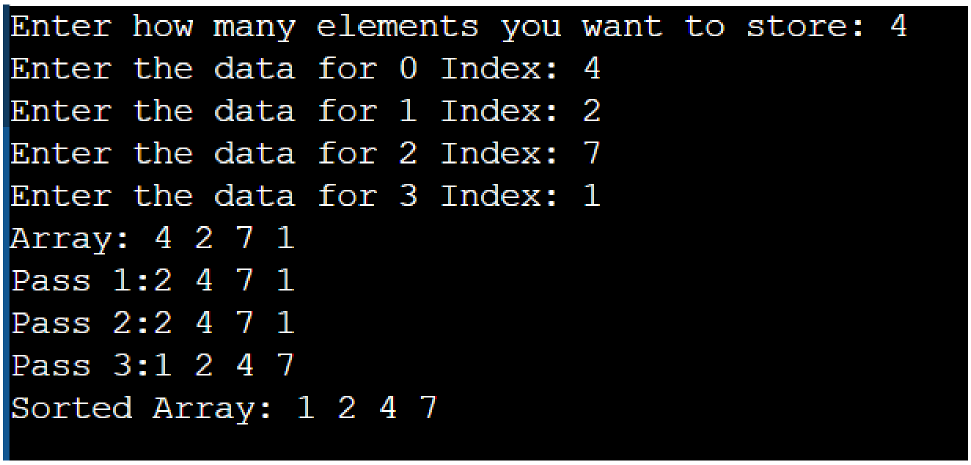
```
                cout << "Sorted Array: ";
                for(int i = 0; i < n; i++) {
                        cout << arr[i] << " ";
                }
                cout << endl;
        }
};
int main()
{
        BubbleSort bs;
        bs.input();
        bs.bsort();
        bs.display();
        return 0;
}
```



```
Enter how many elements you want to store: 5
Enter the data for 0 Index: 5
Enter the data for 1 Index: 2
Enter the data for 2 Index: 9
Enter the data for 3 Index: 1
Enter the data for 4 Index: 5
Array: 5 2 9 1 5
Pass 1:2 5 1 5 9
Pass 2:2 1 5 5 9
Pass 3:1 2 5 5 9
Pass 4:1 2 5 5 9
Sorted Array: 1 2 5 5 9
```

2.) Insertion Sort

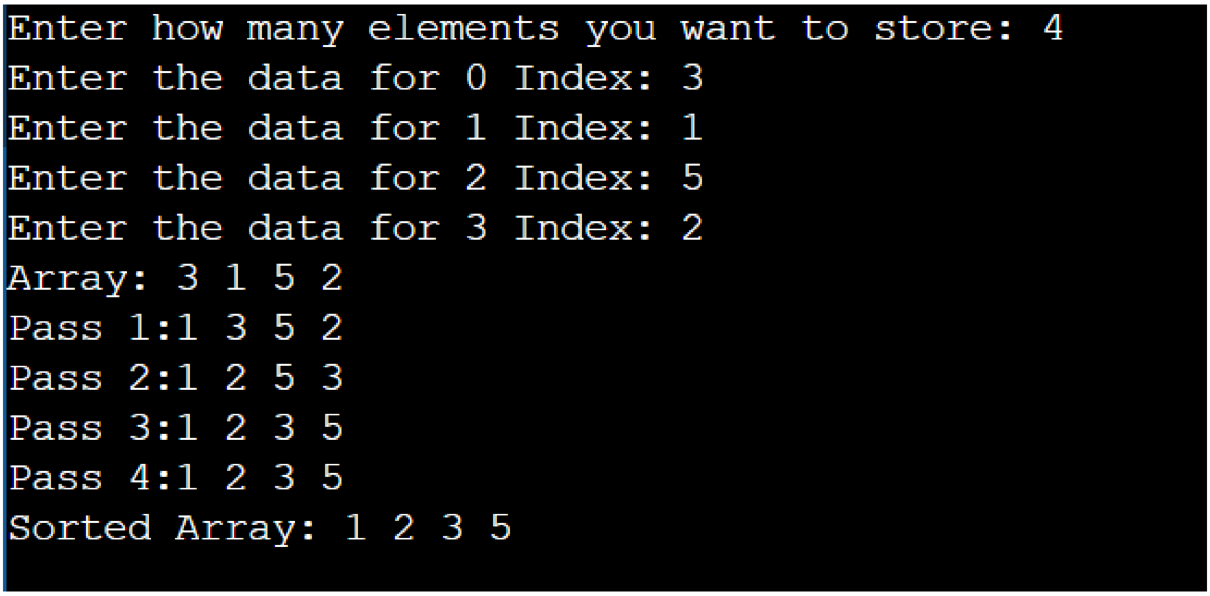
```
#include <iostream>
#define max 100
using namespace std;
int n,arr[max],pass=1;

class InsertionSort {
public:
   void input()
   {
      cout<<"Enter how many elements you want to store: ";
      cin>>n;
      for (int i=0; i<n; i++)
      {
         cout<<"Enter the data for "<<i <<" Index: ";
         cin>>arr[i];
      }
      cout << "Array: ";
      for(int i = 0; i < n; i++) {
         cout << arr[i] << " ";
      }
      cout << endl;
   }
   void isort() {
      for(int i = 1; i < n; i++) {
         for(int j = i; j > 0; j--) {
            if(arr[j] < arr[j-1]) {
               int temp = arr[j];
               arr[j] = arr[j-1];
               arr[j-1] = temp;
            }
            }
            cout << "Pass " <<  pass << ":";
            for(int i = 0; i< n ; i++) {
                cout << arr[i] << " ";
            }
            pass++;
            cout << endl;
        }
    }
    void display() {
        cout << "Sorted Array: ";
        for(int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main()
{
    InsertionSort is;
    is.input();
    is.isort();
    is.display();
    return 0;
}
```



```
Enter how many elements you want to store: 4
Enter the data for 0 Index: 4
Enter the data for 1 Index: 2
Enter the data for 2 Index: 7
Enter the data for 3 Index: 1
Array: 4 2 7 1
Pass 1:2 4 7 1
Pass 2:2 4 7 1
Pass 3:1 2 4 7
Sorted Array: 1 2 4 7
```

3.) Selection Sort

```
#include <iostream>

#define max 100

using namespace std;

int n,arr[max],pass=1;


class SelectionSort {

public:

    void input()

    {

        cout<<"Enter how many elements you want to store: ";

        cin>>n;

        for (int i=0; i<n; i++)

        {

            cout<<"Enter the data for "<<i <<" Index: ";

            cin>>arr[i];

        }

        cout << "Array: ";

        for(int i = 0; i < n; i++) {

            cout << arr[i] << " ";

        }

        cout << endl;

    }

    void ssort() {

        for(int i = 0; i < n; i++) {

            int minIndex = i;

            for(int j = i+1; j < n; j++) {

                if(arr[j] < arr[minIndex]) {

                    minIndex = j;

                }
            }
            if (minIndex != i) {
                int temp = arr[minIndex];
                arr[minIndex] = arr[i];
                arr[i] = temp;
            }
            cout << "Pass " <<  pass << ":";
            for(int i = 0; i< n ; i++) {
                cout << arr[i] << " ";
            }
            pass++;
            cout << endl;
        }
    }
    void display() {
        cout << "Sorted Array: ";
        for(int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main()
{
    SelectionSort ss;
    ss.input();
    ss.ssort();
    ss.display();

    return 0;
}
```



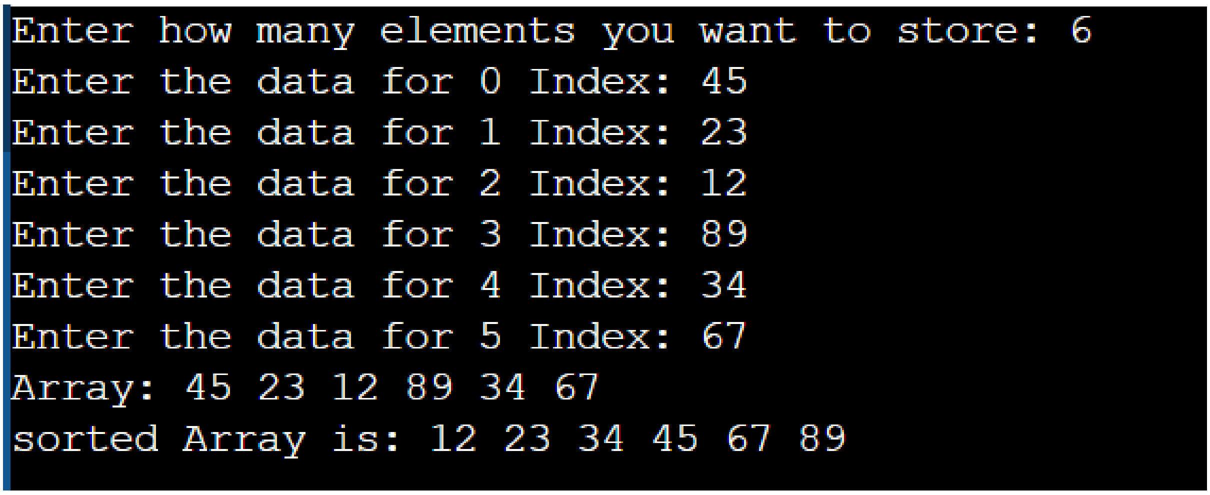
```
Enter how many elements you want to store: 4
Enter the data for 0 Index: 3
Enter the data for 1 Index: 1
Enter the data for 2 Index: 5
Enter the data for 3 Index: 2
Array: 3 1 5 2
Pass 1:1 3 5 2
Pass 2:1 2 5 3
Pass 3:1 2 3 5
Pass 4:1 2 3 5
Sorted Array: 1 2 3 5
```

4.) Quick Sort

```
#include<iostream>

#define max 100

using namespace std;

int n,arr[max];


class QuickSort{

   public:


   void input()

      {

      cout<<"Enter how many elements you want to store: ";

      cin>>n;

      for (int i=0; i<n; i++)

      {

         cout<<"Enter the data for "<<i <<" Index: ";

         cin>>arr[i];

      }

      cout << "Array: ";

      for(int i = 0; i < n; i++) {

         cout << arr[i] << " ";

      }

      cout << endl;

       }

   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);
  }
}
int partition(int arr[], int low, int high)
{
     int pivot = low;
     int i = low +1;
     int j = high;
     while(i<=j){
       while(i <= j && arr[pivot] > arr[i]){
         i++;
       }
       while(i <= j && arr[pivot] < arr[j]){
         j--;
       }
       if(i<j){
         int temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
       }
     }
     int temp = arr[j];
     arr[j] = arr[pivot];
     arr[pivot] = temp;

     return j;
  }
  void output(){
    cout << "sorted Array is: ";
    for(int k = 0; k < n; k++){
      cout << arr[k] << " ";
```

```
            }
        }
};
int main() {
    QuickSort qs;
    qs.input();
    qs.quickSort(arr,0,n-1);
    qs.output();
    return 0;
}
```

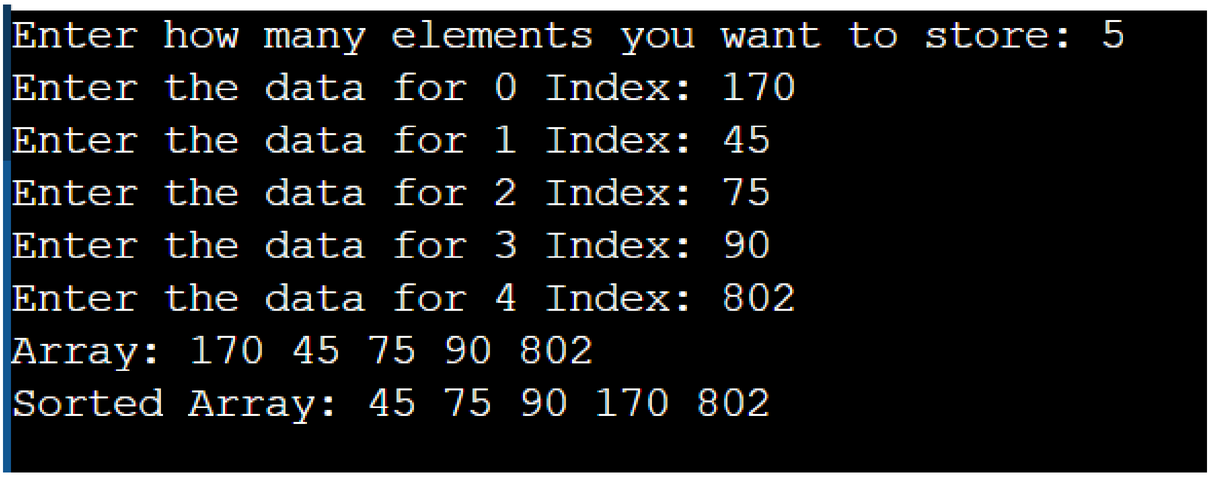


```
Enter how many elements you want to store: 6
Enter the data for 0 Index: 45
Enter the data for 1 Index: 23
Enter the data for 2 Index: 12
Enter the data for 3 Index: 89
Enter the data for 4 Index: 34
Enter the data for 5 Index: 67
Array: 45 23 12 89 34 67
sorted Array is: 12 23 34 45 67 89
```

5.) Radix Sort

```
#include <iostream>
using namespace std;
const int MAX = 100;
int n, arr[MAX];

class RadixSort {
public:
   void input() {
      cout<<"Enter how many elements you want to store: ";
      cin>>n;
      for (int i=0; i<n; i++)
      {
         cout<<"Enter the data for "<<i <<" Index: ";
         cin>>arr[i];
      }
      cout << "Array: ";
      for(int i = 0; i < n; i++) {
         cout << arr[i] << " ";
      }
      cout << endl;
   }
   int getMax() {
      int max = arr[0];
      for(int i = 1; i < n; i++) {
         if(arr[i] > max) {
            max = arr[i];
         }
      }
      return max;
   }
```

```
void countSort(int div) {
    int output[MAX];
    int count[10] = {0};
    for (int i = 0; i < n; i++) {
        count[(arr[i] / div) % 10]++;
    }
    for (int i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / div) % 10]-1] = arr[i];
        count[(arr[i] / div) % 10]--;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
void radixSort() {
    int m = getMax();
    cout << "Max element: " << m << endl;
    for(int div = 1; m/div > 0; div = div *10) {
        countSort(div);
    }
}
void display() {
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout<< arr[i]<<" ";
    }
    cout<<endl;
    }
};
int main() {
    RadixSort rs;
    rs.input();
    rs.radixSort();
    rs.display();
    return 0;
}
```

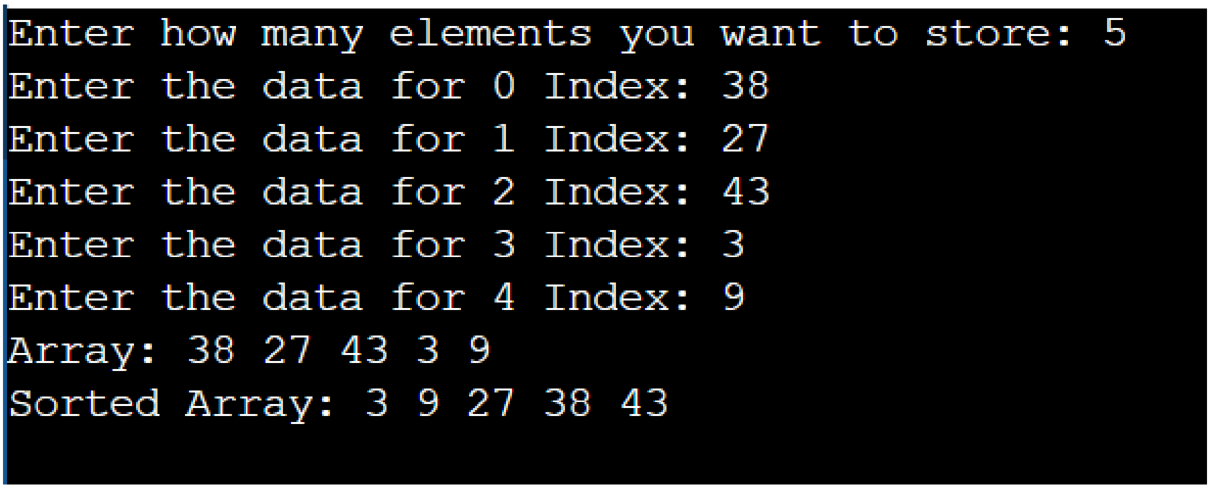


```
Enter how many elements you want to store: 5
Enter the data for 0 Index: 170
Enter the data for 1 Index: 45
Enter the data for 2 Index: 75
Enter the data for 3 Index: 90
Enter the data for 4 Index: 802
Array: 170 45 75 90 802
Sorted Array: 45 75 90 170 802
```

6.) Merge Sort

```
#include <iostream>
using namespace std;
const int MAX = 100;
int n, arr[MAX];

class MergeSort {
public:
   void input() {
      cout<<"Enter how many elements you want to store: ";
      cin>>n;
      for (int i=0; i<n; i++)
      {
         cout<<"Enter the data for "<<i <<" Index: ";
         cin>>arr[i];
      }
      cout << "Array: ";
      for(int i = 0; i < n; i++) {
         cout << arr[i] << " ";
      }
      cout << endl;
   }
   void merge(int low, int mid, int high) {
      int l1 = mid - low + 1;
      int l2 = high - mid;
      int a1[l1], a2[l2];
      for (int i = 0; i < l1; i++) {
         a1[i] = arr[low + i];
      }
      for (int j = 0; j < l2; j++) {
         a2[j] = arr[mid + 1 + j];
```

```
        }
        int i = 0, j = 0, k = low;
        while (i < l1 && j < l2) {
            if (a1[i] < a2[j]) {
                arr[k] = a1[i];
                i++;
            } else {
                arr[k] = a2[j];
                j++;
            }
            k++;
        }
        while (i < l1) {
            arr[k] = a1[i];
            i++;
            k++;
        }
        while (j < l2) {
            arr[k] = a2[j];
            j++;
            k++;
        }
    }
    void mergeSort(int low, int high) {
        if (low < high) {
            int mid = (low + high) / 2;
            mergeSort(low, mid);
            mergeSort(mid + 1, high);
            merge(low, mid, high);
        }
    }
    void sort() {
        mergeSort(0, n - 1);
    }
    void display() {
        cout << "Sorted Array: ";
        for (int i = 0; i < n; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main() {
    MergeSort ms;
    ms.input();
    ms.sort();
    ms.display();
    return 0;
}
```



```
Enter how many elements you want to store: 5
Enter the data for 0 Index: 38
Enter the data for 1 Index: 27
Enter the data for 2 Index: 43
Enter the data for 3 Index: 3
Enter the data for 4 Index: 9
Array: 38 27 43 3 9
Sorted Array: 3 9 27 38 43
```

Observation: In this practical session I learned about different sorting techniques such as bubble sort, insertion sort, selection sort, quick sort, radix sort, merge sort.

Bubble Sort: Simple but inefficient for large datasets; repeatedly swaps adjacent elements. Best for small or simple tasks.

Insertion Sort: Builds a sorted array one element at a time; efficient for small or nearly sorted datasets. Slower for large unsorted arrays.

Selection Sort: Finds the minimum element and places it at the beginning; easy to understand but inefficient for larger arrays. Best for small datasets.

Quick Sort: It works by selecting a "pivot" element and partitioning the array into two halves: one with elements less than the pivot and one with elements greater than it. This process is repeated recursively, making it particularly effective for large datasets.

Radix Sort: Sorts numbers digit by digit; non-comparative and efficient for integers. Limited to fixed-length data types.

Merge Sort: Divides and conquers by merging sorted subarrays; it is much more faster but requires extra space. Great for large datasets requiring stable sorting.