Lab 2: Implement and analyze Bubble Sort, Selection Sort and Insertion Sort.

Theory:

Algorithm for Bubble Sort

```
BubbleSort(A, n)
{
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
            if(a[j]>a[j+1])
                swap(a[j], a[j+1];
}
```

Analysis:

Time Complexity: O (n^2)

Space Complexity: O (n)

Algorithm for Selection Sort

```
SelectionSort(A,n)
{
    for(i=0;i<n;i++)
    {
        least=A[i];
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if (A[j]<least)
            {
                least=A[j];
                pos=j;
            }
        }
        swap(A[i], A[pos]);
    }
}
```

Analysis:

Time Complexity: O (n^2)

Space Complexity: O (n)

Algorithm for Insertion Sort

```
iSort(A,n)
{
    i,j,temp;
    for(i=0;i<n;i++)
    {
      temp=A[i];
      j=i-1;
      while((temp<A[j] &&j>=0))
      {
          A[j+1]=A[j];
          j--;
      }
      A[j+1]=temp;
    }
}
```

Analysis:

Time Complexity: O (n^2)

Space Complexity: O (n)

Source Code

```
#include <iostream>
#include <chrono>
#define MAX 10000
using namespace std;
void swapp(int *p, int *q)
{
    int temp;
    temp = *p;
    *p = *q;
    *q = temp;
}
void display(int A[], int n)
{
    int i;
    for(i=0;i<n;i++)
        cout<<A[i]<<"\t";
    cout<<endl;
}
void bSort(int A[], int n)
{
    int i,j;
    for(i=0;i<n-1;i++)
        for(j=0;j<n-i-1;j++)
            if(A[j]>A[j+1])
                swapp(&A[j], &A[j+1]);
}
void sSort(int A[],int n)
{
    int i,j,least,pos;
    for(i=0;i<n;i++)
    {
        least=A[i];
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if (A[j]<least)
            {
                least=A[j];
                pos=j;
            }
        }
        swapp(&A[i], &A[pos]);
    }
```

```
}
void iSort(int A[],int n)
{
    int i,j,temp;
    for(i=0;i<n;i++)
    {
      temp=A[i];
      j=i-1;
      while((temp<A[j] &&j>=0))
      {
          A[j+1]=A[j];
          j--;
      }
      A[j+1]=temp;
    }
}
int main()
{
    int A[MAX], i, n, choice;
    do
    {
        cout<<"1.GENERATE\n2.BUBBLE SORT\n3.SELECTION SORT\n";
        cout<<"4.INSERTION SORT\n5.EXIT\n";
        cout<<"> ";
        cin>>choice;
        switch(choice)
        {
            case 1:
            {
            cout<<"How many elements? ";
            cin>>n;
            for(i=0;i<n;i++)
                A[i] = rand();
            cout<<n<<" elements generated!"<<endl;
            break;
            }

        case 2:
            {
                display(A,n);
                auto start = chrono::high_resolution_clock::now();
                bSort(A,n);
                auto stop = chrono::high_resolution_clock::now();
                auto duration = chro-
no::duration_cast<chrono::microseconds>(stop-start);
```

```
                display(A,n);
                cout<<"Time = "<<duration.count()<<endl;
                break;
            }
        case 3:
            {
                display(A,n);
                auto start = chrono::high_resolution_clock::now();
                sSort(A,n);
                auto stop = chrono::high_resolution_clock::now();
                auto duration = chro-
no::duration_cast<chrono::microseconds>(stop-start);
                display(A,n);
                cout<<"Time = "<<duration.count()<<endl;
                break;
            }

        case 4:
            {
                display(A,n);
                auto start = chrono::high_resolution_clock::now();
                iSort(A,n);
                auto stop = chrono::high_resolution_clock::now();
                auto duration = chro-
no::duration_cast<chrono::microseconds>(stop-start);
                display(A,n);
                cout<<"Time = "<<duration.count()<<endl;
                break;
            }

        case 5:
            cout<<"BYE"<<endl;
            break;
        }
    }while(choice!=5);
    return 0;
}
```

Conclusion: Hence, in this lab, we successfully implemented the iterative algorithms for sorting namely bubble sort, selection sort and insertion sort. We also analyzed the time and space complexity of these algorithms.