Lab 4: Implement and analyze Merge Sort and Quick Sort using Divide and Conquer approach.

Theory: A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Algorithm for Merge Sort

```
merge(int A[],int l,int m,int r)
{
x=l; k=l; y=m;
while(x<m && y<=r)
{
        if (A[x]<A[y])
            b[k++]=A[x++];
        else
            b[k++]=A[y++];
    }
    for(;x<m;x++,k++)
        b[k]=A[x];
    for(;y<=r;y++,k++)
        b[k]=A[y];
    for(i=l;i<=r;i++)
        A[i]=b[i];
}
mSort(A, l, r)
{
    m;
    if(l<r)
    {
        m=(l+r)/2;
        mSort(A,l,m);
        mSort(A,m+1,r);
        merge(A,l,m+1,r);
    }
}
```

Analysis:

Time Complexity: O (n $\log_2$ n)

Space Complexity: O (n)

Algorithm for Quick Sort

```
partition(A,l,r)
{
    x=l;
    y=r;
    p=A[l];
    temp;
    while(x<y)
    {
        while(A[x]<=p)
            x++;
        while(A[y]>p)
            y--;
        if(x<y)
            swapp(&A[x], &A[y]);
    }
        A[l]=A[y];
        A[y]=p;
        return y;
}
qSort(A, l, r)
{
    p;
    if(l<r)
    {
        p=partition(A,l,r);
        qSort(A,l,p-1);
        qSort(A,p+1,r);
    }
}
```

Analysis:

Time Complexity: O (n^2)

Space Complexity: O (n)

Source Code

```
#include <iostream>
#include <chrono>
#define MAX 10000
int b[MAX];
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;
}
int partition(int A[],int l,int r)
{
    int x=l;
    int y=r;
    int p=A[l];
    int temp;
    while(x<y)
    {
        while(A[x]<=p)
            x++;
        while(A[y]>p)
            y--;
        if(x<y)
            swapp(&A[x], &A[y]);
    }
        A[l]=A[y];
        A[y]=p;
        return y;
}
```

```c
void qSort(int A[],int l,int r)
{
    int p;
    if(l<r)
    {
        p=partition(A,l,r);
        qSort(A,l,p-1);
        qSort(A,p+1,r);
    }
}
void merge(int A[],int l,int m,int r)
{
    int x=l;
    int k=l;
    int y=m;
    int i;
    while(x<m && y<=r)
    {
        if (A[x]<A[y])
            b[k++]=A[x++];
        else
            b[k++]=A[y++];
    }
    for(;x<m;x++,k++)
        b[k]=A[x];
    for(;y<=r;y++,k++)
        b[k]=A[y];
    for(i=l;i<=r;i++)
        A[i]=b[i];
}
void mSort(int A[],int l,int r)
{
    int m;
    if(l<r)
    {
        m=(l+r)/2;
        mSort(A,l,m);
        mSort(A,m+1,r);
        merge(A,l,m+1,r);
    }}
```

```
int main()
{
    int A[MAX], i, n, choice;
    do
    {
        cout<<"1.GENERATE\n2.QUICK SORT\n3.MERGE SORT\n4.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();
                qSort(A,0,n-1);
                auto stop = chrono::high_resolution_clock::now();
                auto duration = chrono::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();
                mSort(A,0,n-1);
                auto stop = chrono::high_resolution_clock::now();
                auto duration = chrono::duration_cast<chrono::microseconds>(stop-start);
                display(A,n);
                cout<<"Time = "<<duration.count()<<endl;
                break;
            }

        case 4:
            cout<<"BYE"<<endl;
            break;
        }
    }while(choice!=4);
    return 0;
}
```

Conclusion: Hence, in this lab, we successfully implemented the quick sort and merge sort. We also analyzed the time and space complexity of this algorithm.