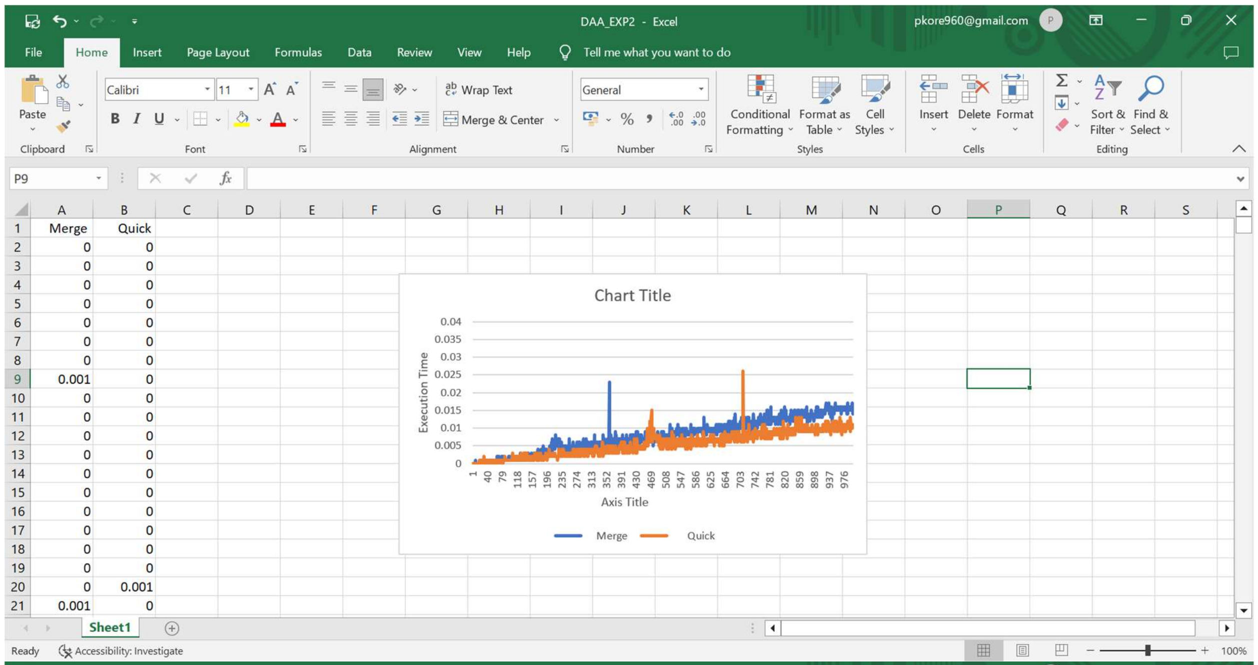
NAME:	Pranav Sadanand Kore
UID:	2021300065
SUBJECT	Design and Analysis of Algorithm
EXP No.	2
AIM:	Execution times of Merge and Quick Sort

ALGORITHM:	• **MergeSort function**
	```if left < right```
	``` mid= (left+right)/2```
	``` MergeSort(array, left, mid)```
	``` MergeSort (array, mid+1, right)```
	``` Merge(array, left, mid, right)```
	• **Merge function**
	```n1= mid-left+1```
	```n2= right-q```
	```create arrays L[n1+1] and R[n2+1]```
	```for i from 0 to n1-1```
	``` do L[i]=array[p+i]```
	```for i from 0 to n2-1```
	``` do R[i]=array[q+i+1]```
	```L[n1]= aprox. Infinity```
	```R[n2]= aprox. Infinity```
	```Declare k, i=1 and j=1```
	```For k from left to right```
	``` Do if L[i]<=R[j]```
	``` Then array[k]=L[i]```
	``` i++```
	``` else```
	``` array[k]=R[j]```
	``` j++```

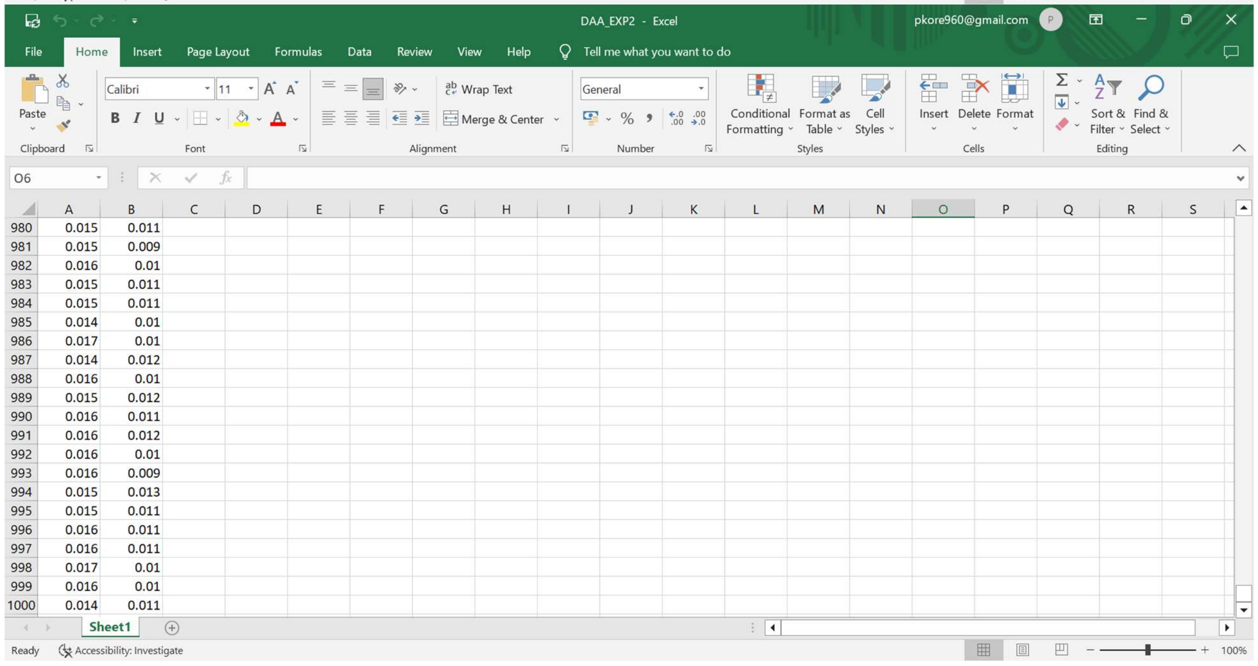
PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

void Printarr(int arr[],int n)
{
    for(int i=0; i<n; i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
}

void Merge(int arr[],int p,int q,int r)
{
    int n1,n2;
    n1=q-p+1;
    n2=r-q;
    int L[n1+1], R[n2+1];
    for(int i=0; i<n1; i++)
    {
        L[i]=arr[p+i];
    }
    for(int i=0; i<n2; i++)
    {
        R[i]=arr[q+i+1];
    }
    L[n1]=999999;
    R[n2]=999999;

    int i=0,j=0;
    for(int k = p ; k <= r ; k++)
    {
        if(L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
    }
    return;
}

void MergeSort(int arr[],int p,int r)
{
    int q;
    if(r>p)
    {
        int q = (p+r)/2;
        MergeSort(arr,p,q);
        MergeSort(arr,q+1,r);
        Merge(arr,p,q,r);
    }
    return;
}

void QuickSort(int number[25],int first,int last){
    int i, j, pivot, temp;
    if(first<last){
        pivot=first;
        i=first;
        j=last;
        while(i<j){
            while(number[i]<=number[pivot]&&i<last)
                i++;
            while(number[j]>number[pivot])
                j--;
            if(i<j){
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
            }
        }
        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        QuickSort(number,first,j-1);
        QuickSort(number,j+1,last);
    }
}

void Addnnum(int arr[],int n){
    for(int i=0;i<n;i++){
        arr[i]=rand()%10000+1;
    }
}

int main()
{
    int arr[100000],a=1,b;
    double start,end,time;

    printf("Enter 1:Merge sort    2:Quick sort\n");
    scanf("%d",&b );
    if(b==1){
        printf("Merge Sort:\n");
    }
    else{
        printf("Quick Sort:\n");
    }

    for(int i=1;i<1000;i++){
        a=i*100;
        Addnnum(arr,a);
        if(b==1){
            start=(double)clock();
            MergeSort(arr,0,a-1);
            end=(double)clock();
        }
        else{
            start=(double)clock();
            QuickSort(arr,0,a-1);
            end=(double)clock();
        }
        time=(double)((end-start)/CLOCKS_PER_SEC);
        printf("%lf\n",time);
    }

    return 0;
}
```

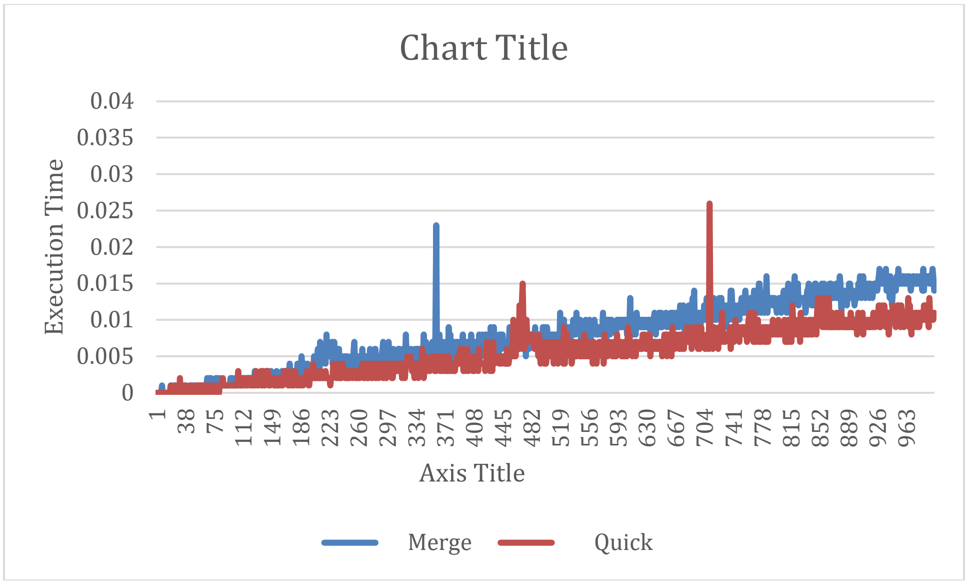
RESULT (SNAPSHOT)



	A	B
1	Merge	Quick
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0.001	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0
20	0	0.001
21	0.001	0



	A	B
980	0.015	0.011
981	0.015	0.009
982	0.016	0.01
983	0.015	0.011
984	0.015	0.011
985	0.014	0.01
986	0.017	0.01
987	0.014	0.012
988	0.016	0.01
989	0.015	0.012
990	0.016	0.011
991	0.016	0.012
992	0.016	0.01
993	0.016	0.009
994	0.015	0.013
995	0.015	0.011
996	0.016	0.011
997	0.016	0.011
998	0.017	0.01
999	0.016	0.01
1000	0.014	0.011



CONCLUSION:	The experiment demonstrated the efficiency and scalability of the merge and Quick sort algorithm. The results showed that the both algorithm had a linear time complexity of O(n log n), making it an ideal choice for sorting large data sets. The results were consistent with the expected results, and the algorithm was able to sort the data sets efficiently and accurately.