NAME:	Pranav Sadanand Kore
UID:	2021300065
SUBJECT	Design and Analysis of Algorithm
EXPERIMENT NO :	1b
AIM:	Finding the running time of an algorithm

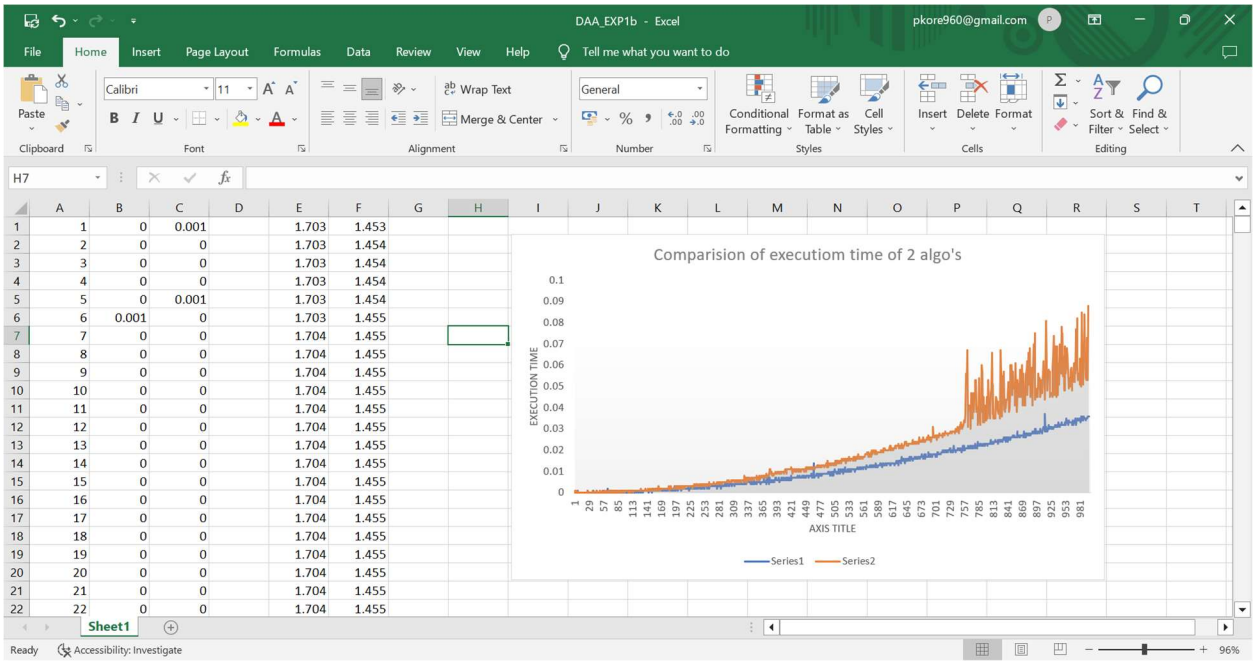
ALGORITHM:	**1.Insertion Sort**
	procedure insertionSort(A: list of sortable items)
	n = length(A)
	for i = 1 to i<n do
	j = i-1
	while j >= 0 and A[j] > A[i] do
	swap(A[j+1], A[j])
	j = j - 1
	end while
	A[j + 1] = A[i];
	end for
	end procedure
	
	2.Selection Sort
	• Initialize minimum value(min_idx) to location 0.
	• Traverse the array to find the minimum element in the array.
	• While traversing if any element smaller than min_idx is found then swap both the values.
	• Then, increment min_idx to point to the next element.
	• Repeat until the array is sorted**.**

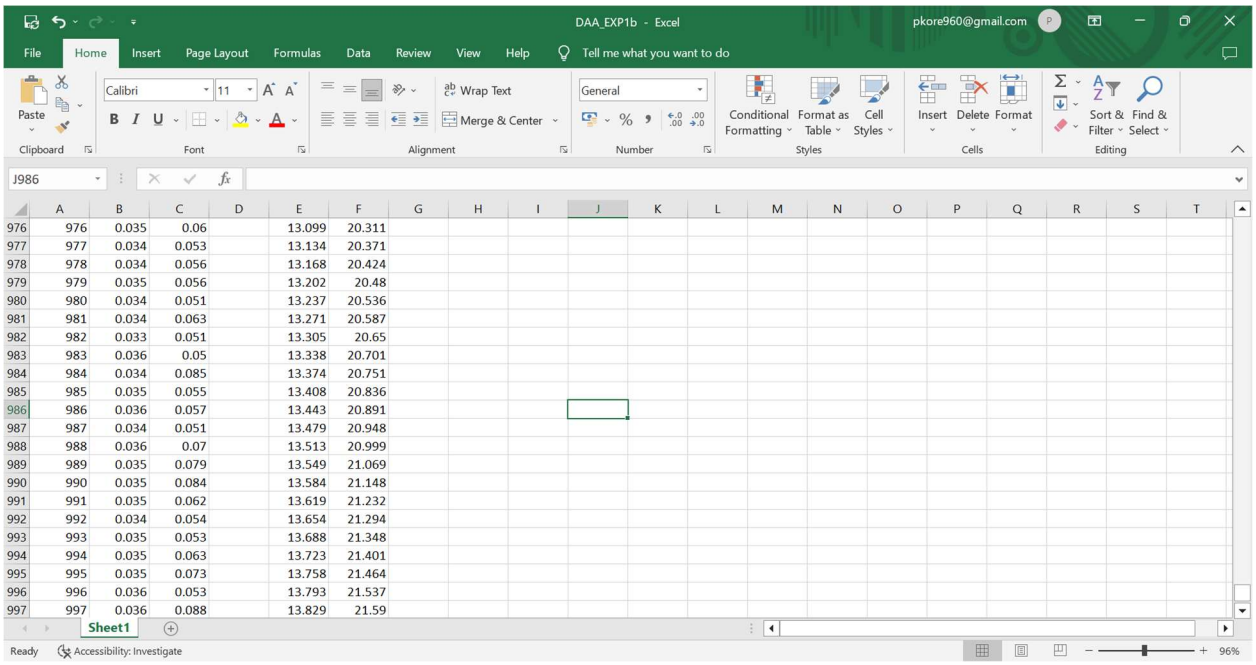
PROGRAM:

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

void Insertionsort(int A[],int n){
  int j,key;
   for(int i=1;i<n;i++){
       key=A[i];
       j=i-1;
       while(j>=0 && key<A[j]){
           A[j+1]=A[j];
           j--;
       }
       A[j + 1] = key;
   }
}

void Selectionsort(int A[],int n){
    int min_ind,temp;
    for(int i=0;i<n;i++){
        min_ind=i;
        for(int j=i;j<n;j++){
            if(A[min_ind]>A[j]){
                  min_ind=j;
            }
        }
        if(min_ind!=i){
            temp=A[i];
            A[i]=A[min_ind];
            A[min_ind]=temp;
        }
    }
}

int main(){

    int A[100000],a,b;
    double start,end,time;

    printf("Enter 1:Insertion sort    2:Selection sort\n");
    scanf("%d",&b );

    for(int i=1;i<1000;i++){
        a=i*10;
```

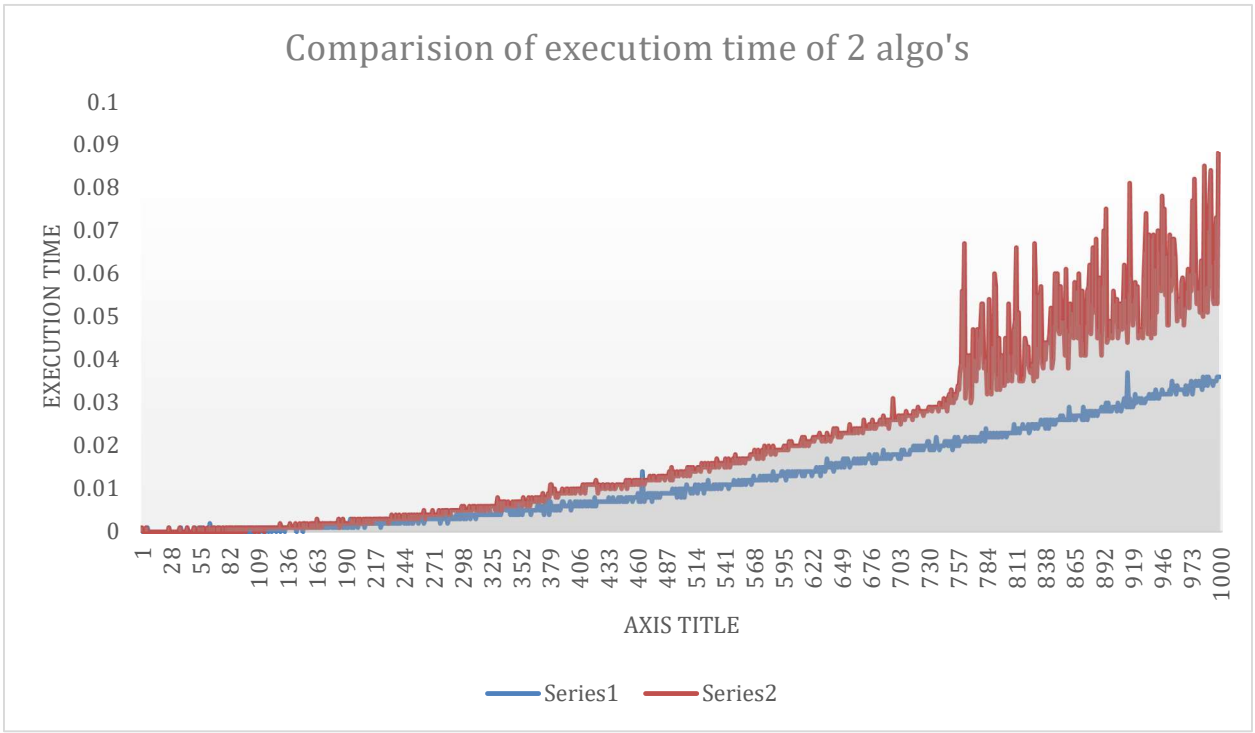
```
        for(int i=0;i<a;i++){
            A[i]= rand() % 100000 +1;
        }
        start=(double)clock()/CLOCKS_PER_SEC;
        if(b==1){
            Insertionsort(A,a);
        }
        else{
            Selectionsort(A,a);
        }
        end=(double)clock()/CLOCKS_PER_SEC;
        time=end-start;
        printf("%lf\n",time);
    }

    return 0;
}
```

RESULT (SNAPSHOT)





	A	B	C	D	E	F
976	976	0.035	0.06		13.099	20.311
977	977	0.034	0.053		13.134	20.371
978	978	0.034	0.056		13.168	20.424
979	979	0.035	0.056		13.202	20.48
980	980	0.034	0.051		13.237	20.536
981	981	0.034	0.063		13.271	20.587
982	982	0.033	0.051		13.305	20.65
983	983	0.036	0.05		13.338	20.701
984	984	0.034	0.085		13.374	20.751
985	985	0.035	0.055		13.408	20.836
986	986	0.036	0.057		13.443	20.891
987	987	0.034	0.051		13.479	20.948
988	988	0.036	0.07		13.513	20.999
989	989	0.035	0.079		13.549	21.069
990	990	0.035	0.084		13.584	21.148
991	991	0.035	0.062		13.619	21.232
992	992	0.034	0.054		13.654	21.294
993	993	0.035	0.053		13.688	21.348
994	994	0.035	0.063		13.723	21.401
995	995	0.035	0.073		13.758	21.464
996	996	0.036	0.053		13.793	21.537
997	997	0.036	0.088		13.829	21.59



Series1: Insertion Sort

Series2: Selection Sort

As the no. of elements increases for sorting the time for the same also increases in both cases and in an exponential order.

CONCLUSION:	From the result of graphs , the thing can be concluded that both algorithms are good enough in case of smaller amount of numbers. As graph of both algorithms growing exponentially, they are not good in case of large amount of numbers. In case of only these two algo's insertion sort is better.