Name	SMRUTI SONEKAR
UID no.	2021700064
Experiment No.	1B

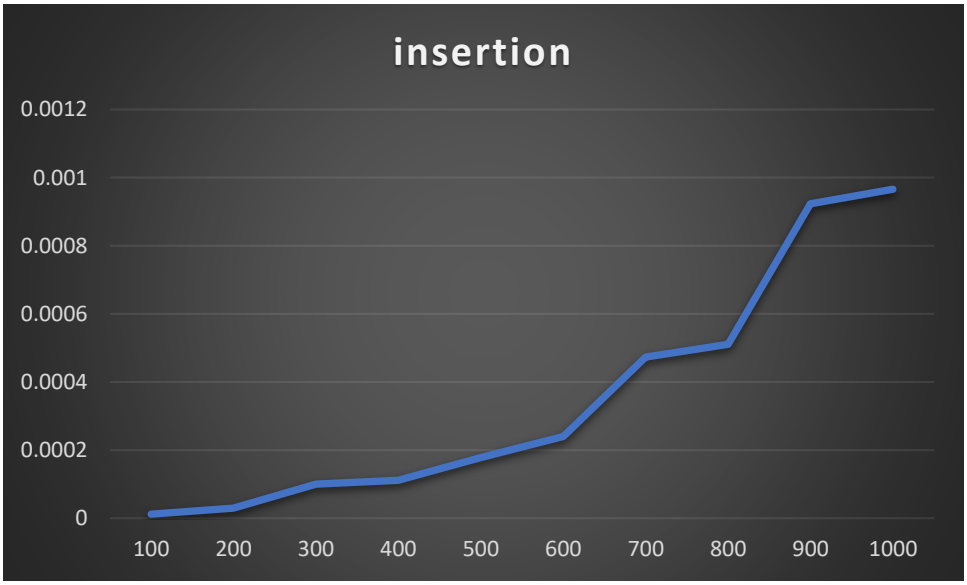
AIM:	**To implement the sorting functions e.g. Insertion sort, Selection sort.**
Program 1	
PROBLEM STATEMENT :	INSERTION SORT
ALGORITHM/ THEORY:	It works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place 1. **In insertion sort , I and j are declared and i=1 and j starts from 0.** 2. **We run loop from i=1 to till size of array passed .** 3. **Declare j=i-1.Then we put while condition where a[i] and a[j] are checked is a[j] is grater than a[i] then it is swapped or replaced.** 4. **In main function , I have used rand() function to generate random numbers till 1000 and sort those random numbers.** 5. **Clock() function is also used to calculation**

	the time taken to sort by various array sizes. 6. Insertion sorting is simple and fast method and it sorts element as we pass or give input the numbers. 7. he outer for loop starts at index '1' and runs for 'n-1' iterations, where 'n' is the length of the array. 8. The inner while loop starts at the current index i of the outer for loop and compares each element to its left neighbor. If an element is smaller than its left neighbor, the elements are swapped. 9. The inner while loop continues to move an element to the left as long as it is smaller than the element to its left. 10. Once the inner while loop is finished, the element at the current index is in its correct position in the sorted portion of the array. 11. The outer for loop continues iterating through the array until all elements are in their correct positions and the array is fully sorted. Time complexity is size of array and The Big O notation is a function that is defined in terms of the input. • Best case (all input in ascending order)= O(n). • Worst case(input in descending

order)=O(n^2) .

- Average case:O(n^2).

GRAPH:



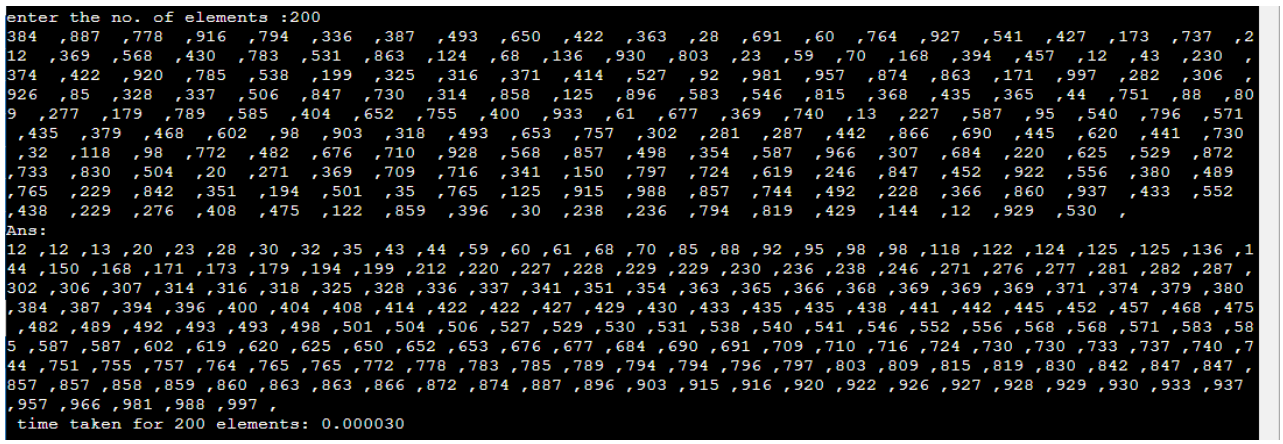
As size of array increases , insertion sort time increases gradually , it also depends on The input as random numbers are generated and it sorts as soon as input is given.

PROGRAM:

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

//#define CLOCKS_PER_SEC ((clock_t)(1000))

void insertionsort(int a[],int n);
void insert1(int a[],int n);
void display(int a[],int n);
int main()
{

    int n,k=0;
    clock_t start,end;
    printf("\nenter the no. of elements :");
    scanf("%d",&n);
    int a[n],num;
    for(k=0;k<n;k++)
    {
        a[k]= rand() % 1000 + 1;
        printf("%d  ,",a[k]);
    }
     double time1;
    start=clock();
    insertionsort(a,n);
    end=clock();
    display(a,n);

    time1= ((double)(end - start))/CLOCKS_PER_SEC;

    printf("\n time taken for %d elements: %lf\n ",n,time1);
    return 0;
}

void display(int a[],int n)
{
     int i;
     printf("\nAns:\n");
    for(i=0;i<n;i++)
    {
        printf("%d ,",a[i]);
    }
}
```

```c
void insertionsort(int a[],int n)
{
    int temp,i,j;
    for(i=1;i<n;i++)
    {
        temp=a[i];
        j=i-1;

        while(j>=0 && a[j]>temp)
        {
           a[j+1]=a[j];
            j=j-1;
        }
       a[j+1]=temp;
    }
}

void insert1(int a[], int n)
{
    int i,j,temp;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(a[i] > a[j] )
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
    }
}
```

RESULT:



Program 2	
PROBLEM STATEMENT :	SELECTION SORT.
ALGORITHM/ THEORY:	• **In the selection sort, first of all, we set the initial element as a minimum.** • **Now we will compare the minimum with the second element. If the second element turns out to be smaller than the minimum, we will swap them, followed by assigning to a minimum to the third element.** • **Else if the second element is greater than the minimum, which is our first element, then we will do nothing and move on to the third element and then compare it with the minimum.** • **We will repeat this process until we reach the last element.**

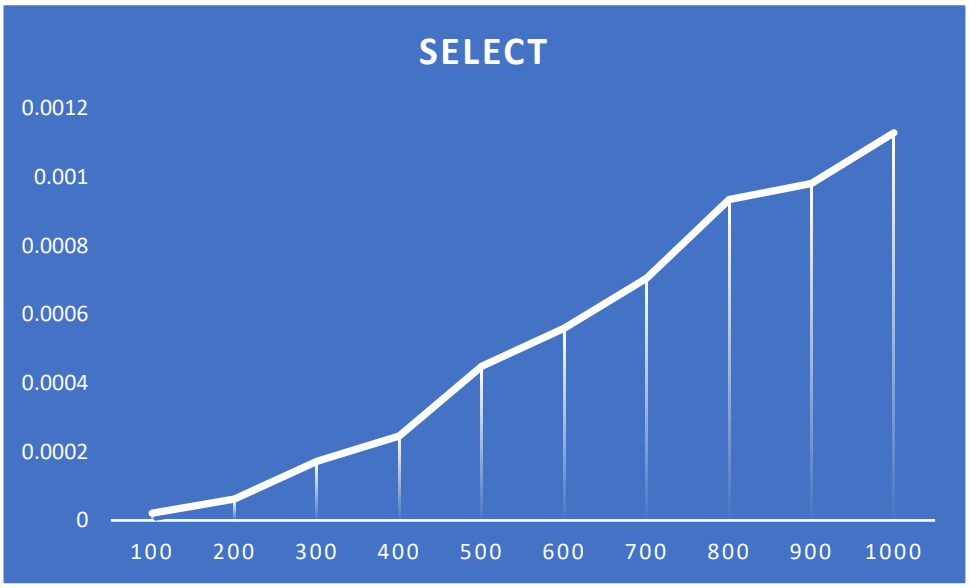
- **After the completion of each iteration, we will notice that our minimum has reached the start of the unsorted list.**
- **For each iteration, we will start the indexing from the first element of the unsorted list. We will repeat the Steps from 1 to 4 until the list gets sorted or all the elements get correctly positioned.**
- **Consider the following example of an unsorted array that we will sort with the help of the Selection Sort algorithm.**
- **ALGORITHM:**
- 1.k ← length [A]
- 2. **for** j ←1 to n-1
- 3. smallest ← j
- 4. **for** I ← j + 1 to k
- 5. **if** A [i] < A [smallest]
- 6. then smallest ← i
- 7. exchange (A [j], A [smallest])

Time Complexity:

- **Best case (all input in ascending order)= O(n^2).**
- **Worst case(input in descending order)=O(n^2) .**
- **Average case:O(n^2).**

- This is because, in each step, we are required to find minimum elements so that it can be placed in the correct position. Once we trace the complete array, we will get our minimum element.

GRAPH:



This graph indicates that time is taken more than insertion sort as at each step minimum element is to found and replaced. So the line at 1000 input is at 0.00012 as in insertion sort of 1000 array size was 0.000966.

PROGRAM:

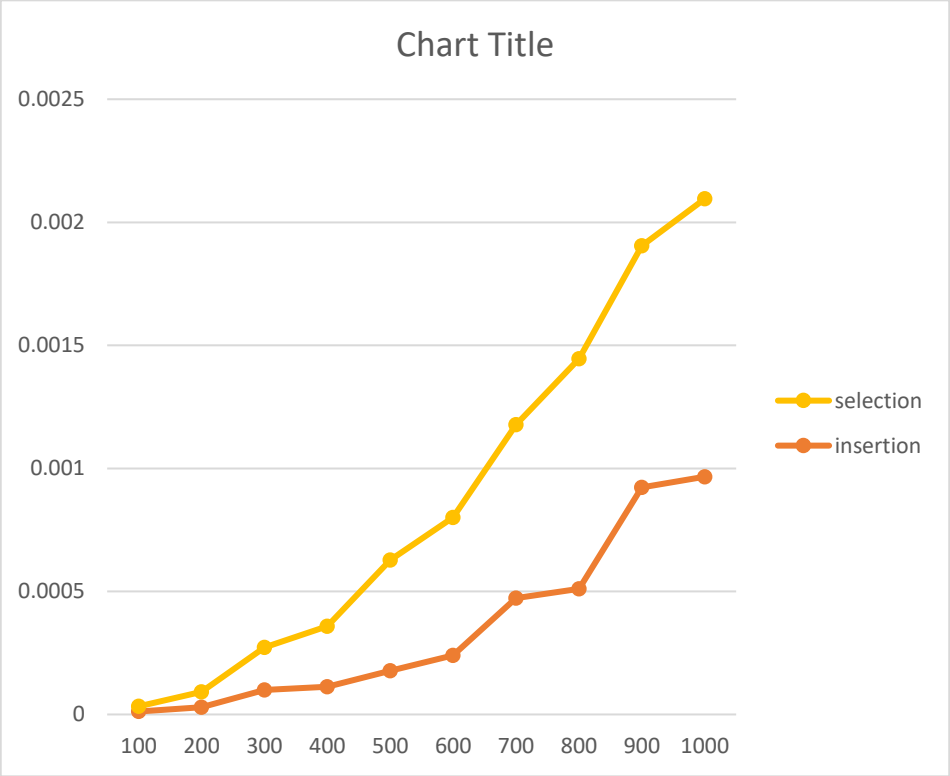
```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
//#define CLOCKS_PER_SEC ((clock_t)(1000))


void selection(int a[],int n);
void display(int a[],int n);
void swap(int *x,int *y);
int main()
{

    int n,i;  clock_t start,end;
    printf("enter number of elements:");
    scanf("%d",&n);
    int a[n];
    for(i=0;i<n;i++)
    {
        a[i]= rand() % 1000 + 1;
        printf("%d  ,",a[i]);
    }
      double time1;
    start=clock();
   selection(a,n);
    end=clock();
   display(a,n);

    time1= ((double)(end - start))/CLOCKS_PER_SEC;

    printf("\n time taken for %d elements: %lf\n ",n,time1);



    return 0;
}

void display(int a[],int n)
{
    int i;

    printf("\nANS:\n");
    for(i=0;i<n;i++)
```

```
    {
        printf("%d ,",a[i]);
    }
}
void swap (int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
void selection(int a[],int n)
{
    int i,j;
    int min;int temp;
    for(i=0;i<n-1;i++)
    {
        min=i;
        for(j=i+1;j<n;j++)
        {
          if(a[min] > a[j])
         {

            min=j;
         }

        }
       swap(&a[min],&a[i]);


    }
}
```

RESULT:

```
enter number of elements:200
884  ,887  ,778  ,916  ,794  ,336  ,387  ,493  ,650  ,422  ,363  ,28  ,691  ,60  ,764  ,927  ,541  ,427  ,173  ,737  ,2
12  ,369  ,568  ,430  ,783  ,531  ,863  ,124  ,68  ,136  ,930  ,803  ,23  ,59  ,70  ,168  ,394  ,457  ,12  ,43  ,230  ,
874  ,422  ,920  ,785  ,538  ,199  ,325  ,316  ,371  ,414  ,527  ,92  ,981  ,957  ,874  ,863  ,171  ,997  ,282  ,306  ,
926  ,85  ,328  ,337  ,506  ,847  ,730  ,314  ,858  ,125  ,896  ,583  ,546  ,815  ,368  ,435  ,365  ,44  ,751  ,88  ,80
9  ,277  ,179  ,789  ,585  ,404  ,652  ,755  ,400  ,933  ,61  ,677  ,369  ,740  ,13  ,227  ,587  ,95  ,540  ,796  ,571
,435  ,379  ,468  ,602  ,98  ,903  ,318  ,493  ,653  ,757  ,302  ,281  ,287  ,442  ,866  ,690  ,445  ,620  ,441  ,730
,32  ,118  ,98  ,772  ,482  ,676  ,710  ,928  ,568  ,857  ,498  ,354  ,587  ,966  ,307  ,684  ,220  ,625  ,529  ,872
733  ,830  ,504  ,20  ,271  ,369  ,709  ,716  ,341  ,150  ,797  ,724  ,619  ,246  ,847  ,452  ,922  ,556  ,380  ,489
765  ,229  ,842  ,351  ,194  ,501  ,35  ,765  ,125  ,915  ,988  ,857  ,744  ,492  ,228  ,366  ,860  ,937  ,433  ,552
438  ,229  ,276  ,408  ,475  ,122  ,859  ,396  ,30  ,238  ,236  ,794  ,819  ,429  ,144  ,12  ,929  ,530  ,
ANS:
12 ,12 ,13 ,20 ,23 ,28 ,30 ,32 ,35 ,43 ,44 ,59 ,60 ,61 ,68 ,70 ,85 ,88 ,92 ,95 ,98 ,98 ,118 ,122 ,124 ,125 ,125 ,136 ,1
44 ,150 ,168 ,171 ,173 ,179 ,194 ,199 ,212 ,220 ,227 ,228 ,229 ,229 ,230 ,236 ,238 ,246 ,271 ,276 ,277 ,281 ,282 ,287 ,
302 ,306 ,307 ,314 ,316 ,318 ,325 ,328 ,336 ,337 ,341 ,351 ,354 ,363 ,365 ,366 ,368 ,369 ,369 ,369 ,371 ,374 ,379 ,380
384 ,387 ,394 ,396 ,400 ,404 ,408 ,414 ,422 ,422 ,427 ,429 ,430 ,433 ,435 ,435 ,438 ,441 ,442 ,445 ,452 ,457 ,468 ,475
,482 ,489 ,492 ,493 ,493 ,498 ,501 ,504 ,506 ,527 ,529 ,530 ,531 ,538 ,540 ,541 ,546 ,552 ,556 ,568 ,568 ,571 ,583 ,58
5 ,587 ,587 ,602 ,619 ,620 ,625 ,650 ,652 ,653 ,676 ,677 ,684 ,690 ,691 ,709 ,710 ,716 ,724 ,730 ,730 ,733 ,737 ,740 ,7
44 ,751 ,755 ,757 ,764 ,765 ,765 ,772 ,778 ,783 ,785 ,789 ,794 ,794 ,796 ,797 ,803 ,809 ,815 ,819 ,830 ,842 ,847 ,847 ,
857 ,857 ,858 ,859 ,860 ,863 ,863 ,866 ,872 ,874 ,887 ,896 ,903 ,915 ,916 ,920 ,922 ,926 ,927 ,928 ,929 ,930 ,933 ,937
957 ,966 ,981 ,988 ,997 ,
time taken for 200 elements: 0.000093
```

RESULT GRAPH



CONCLUSION:

Through insertion sort and selection sort time complexity concept is very clearly understood and from analysis we can say that insertion sort is better than selection sort as time taken by insertion sort was much lesser than selection sort.