NAME:	Utsav Avaiya
UID:	2021300005
SUBJECT	DAA
EXPERIMENT NO :	1.B
AIM:	Experiment on finding the running time of an algorithm
THEORY AND ALGORITHM:	Sorting refers to ordering data in an increasing or decreasing fashion according to some linear relationship among the data items. Sorting can be done on names, numbers and records. Sorting reduces the For example, it is relatively easy to look up the phone number of a friend from a telephone dictionary because the names in the phone book have been sorted into alphabetical order. This example clearly illustrates one of the main reasons that sorting large quantities of information is desirable. That is, sorting greatly improves the efficiency of searching. If we were to open a phone book, and find that the names were not presented in any logical order, it would take an incredibly long time to look up someone's phone number. **Insertion sort**– 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.

	Selection sort– It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted. **ALGORITHM:** Insertion Sort Function: ● Iterate from arr[1] to arr[N] over the array. ● Compare the current element (key) to its predecessor. ● If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element. Selection Sort Function: ● 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. GetInput Function: ● Function to make 100000 random numbers to sort and put into a text file Readfile Function: ● Function to read numbers from the text file Main Function: 1.Make a menu driven function and ask user his choice of

	sorting technique 2.If insertion sort, call the function and calculate the time interval at every 100 numbers getting sorted up to 1000 times/block. 3.If Selection sort, call the function and calculate the time interval at every 100 numbers getting sorted up to 1000 times/block. 4.Else if, invalid input.
PROGRAM:	

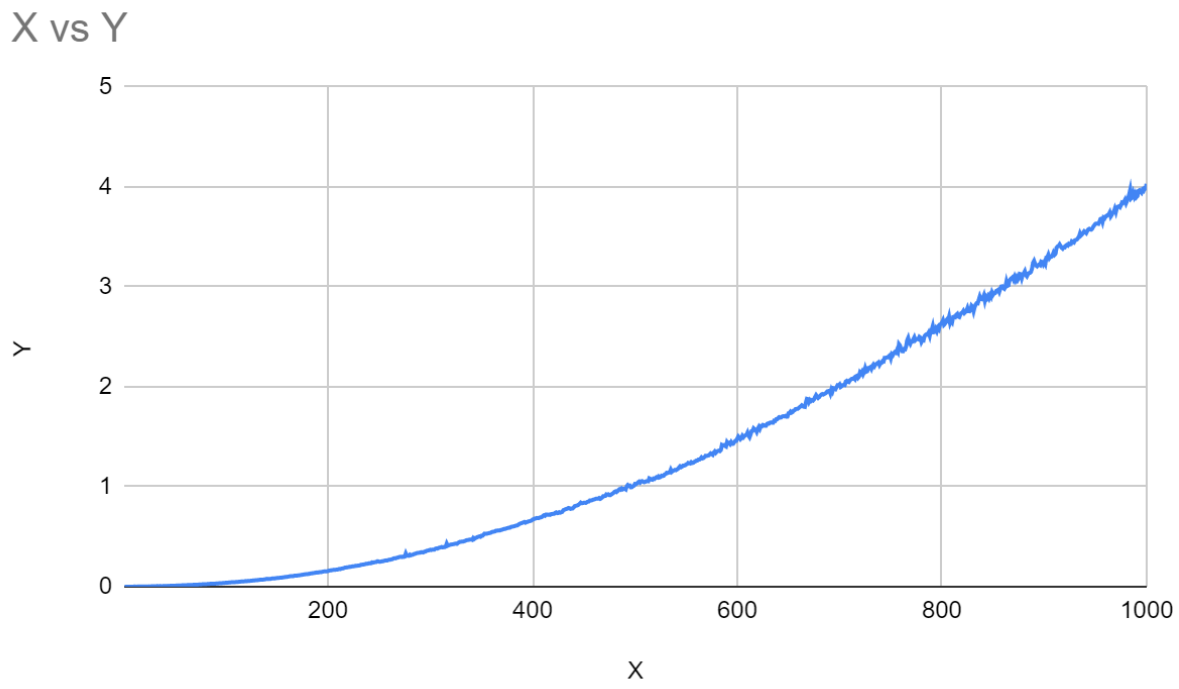
```c
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<time.h>
void selectionsort(int arr[],int n)
{
   for(int i=0;i<n;i++)
   {
      int min_ind=i;
      for(int j=i+1;j<n;j++)
      {
         if(arr[j]<arr[min_ind])
        min_ind=j;
      }
      if(min_ind!=i)
      {
         int t=arr[i];
         arr[i]=arr[min_ind];
     arr[min_ind]=t;
      }
   }
}
void insertionsort(int arr[],int n)
{
   for(int i=1;i<n;i++)
   {
      int j=i-1;
      int key=arr[i];
      while(j>=0 && arr[j]>key)
      {
            arr[j + 1] = arr[j];
            j = j - 1;
```

```
      }
      arr[j + 1] = key;
   }
}

void generate_numbers()
{

  FILE *ptr;
  ptr=fopen("number.txt","w");
  for(int i=0;i<100000;i++)
  {
     fprintf(ptr,"%d\n",rand() % 100000);
  }
  fclose(ptr);
}
void operation()
{
  FILE *ptr;
  ptr=fopen("number.txt","r");
  for(int j=0;j<100000;j+=100)
  {
      int arr1[j];
      int arr2[j];
      for(int i=0;i<j;i++)
      {
          fscanf(ptr,"%d\n",&arr1[i]);
      }
      for(int i=0;i<j;i++)
      {
         arr2[i]=arr1[i];
      }
      clock_t start_selection=clock();
      selectionsort(arr1,j);
      clock_t end_selection = clock();
      double currs=(double)(end_selection-
start_selection)/CLOCKS_PER_SEC;


      clock_t start_insertion=clock();
      insertionsort(arr2,j);
      clock_t end_insertion=clock();
      double curri=(double)(end_insertion-
start_insertion)/CLOCKS_PER_SEC;
```

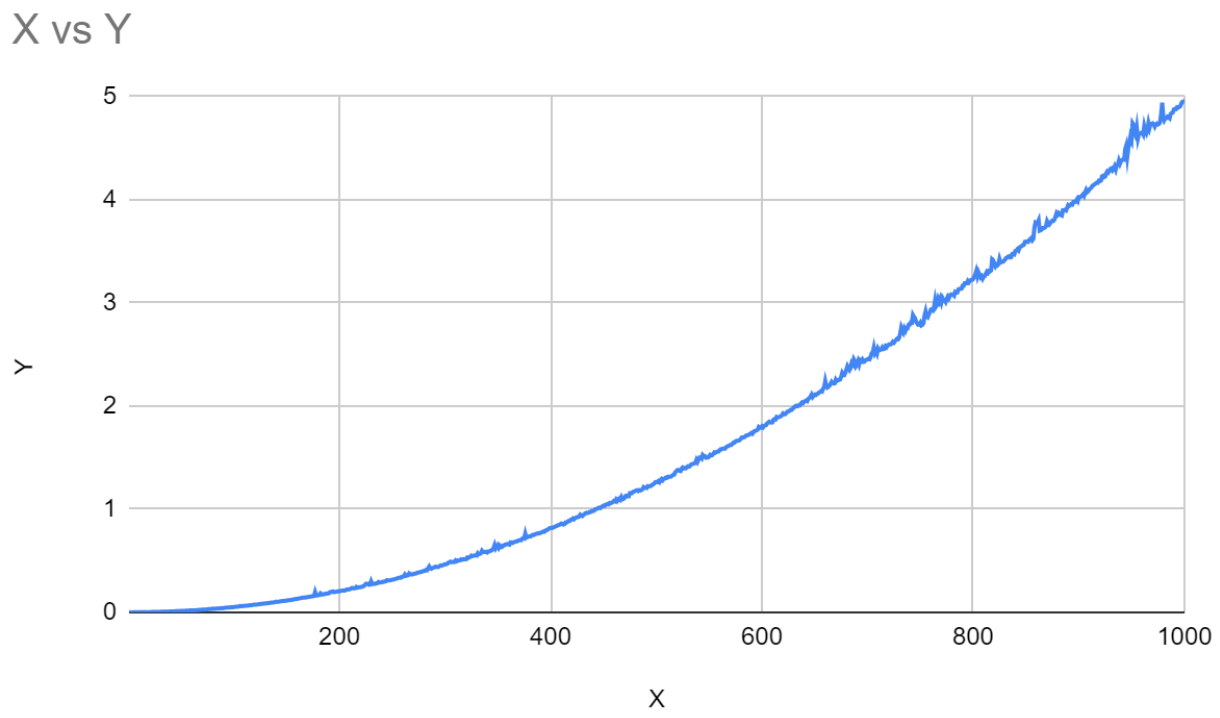
```
        printf("\n%d\t%f\t%f",j,currs,curri);
    }
}
int main()
{
    generate_numbers();
    operation();
    return 0;
}
```

RESULT (SNAPSHOT):
Graph for taken by Selection sort and Insertion sort:

1.INSERTION SORT:



2.SELECTION SORT:



CONCLUSION:	Selection sort takes more time than Insertion sort. So, Insertion sort is more efficient than selection sort. I understood the implementation of insertion sort and selection sort with their time complexity using graph plotting in MS excel.