Name	Aditi Nilesh Bhutada
UID no.	2021700009
Experiment No.	1 B

AIM:	Finding the running time of an algorithm. The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts.
PROGRAM	
THEORY:	**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.
INPUT:	We have to generate random 100000 numbers using rand() function and use this input as 1000 blocks of 100 integer numbers to Insertion and Selection sorting algorithms.
ALGORITHM:	• Selection sort: Step 1: Initialize minimum value(min) to location 0. Step 2: Traverse the array to find the minimum element in the array. Step 3: While traversing if any element smaller than min is found then swap

	both the values. Step 4: Then, increment min to point to the next element. Step 5: Repeat until the array is sorted. • Insertion sort Step 1: If the element is the first element, assume that it is already sorted. Return 1. Step 2: Pick the next element, and store it separately in a key. Step 3: Now, compare the **key** with all elements in the sorted array. Step 4: If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right. Step 5: Insert the value and repeat until the array is sorted • Main function Step 1: Now in the main function, initialize the array with size 1 lakh and use rand() function to generate 100000 random numbers • Now import the file where the output of 100000 numbers is stored as an input for both the algorithms • Initialize the block size from 1 to 1000 and size of the block=100 so that with every increment in block, 100 numbers are added to it i.e. 100,200,300…1000. • Use clock() function and CLOCKS_PER_SEC to calculate the runtime of every block

PROGRAM:	```#include <stdio.h>
	#include<stdlib.h>
	#include<time.h>
	#include <math.h>
	
	void swap(int *x, int *y)
	{
	    int temp = *x;
	    *x = *y;
	    *y = temp;
	}
	
	void insertionSort(int arr[], int n)
	{
	    int i, key, j;
	    for (i = 1; i < n; i++)```

```
    {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}


void selectionSort(int arr[],int n)
{

    for(int i=0;i<n-1;i++)
    {
        int minIndex=i;
        for(int j=i+1;j<n;j++)
        {
            if(arr[j]<arr[minIndex])
            {
                minIndex=j;
            }
        }
        swap(&arr[minIndex],&arr[i]);
    }


}

int main()
{
   FILE* ptr;

    // file in reading mode
    ptr = fopen("inputRandom.txt", "r");

    if (NULL == ptr)
    {
        printf("Error in opening file!\n");
    }
```

```
    int block=1, size=100;
    while(block<=1000)
    {
      int data[size];
      for(int i=0;i<size;i++)
     {
        fscanf(ptr,"%d ",&data[i]);
     }

     clock_t t;
     t = clock();
     selectionSort(data,size);

    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("\n  %d    %f",block,time_taken);

    size=size+100;
    block++;
    fseek(ptr,0,SEEK_SET); //moving cursor again to start pointer
of txt file

    }

     fclose(ptr);

}
```

OUTPUT:

Total runtime taken for Selection Sort: 4435.913 Seconds
Total Runtime taken for Insertion sort: 2810.567 seconds

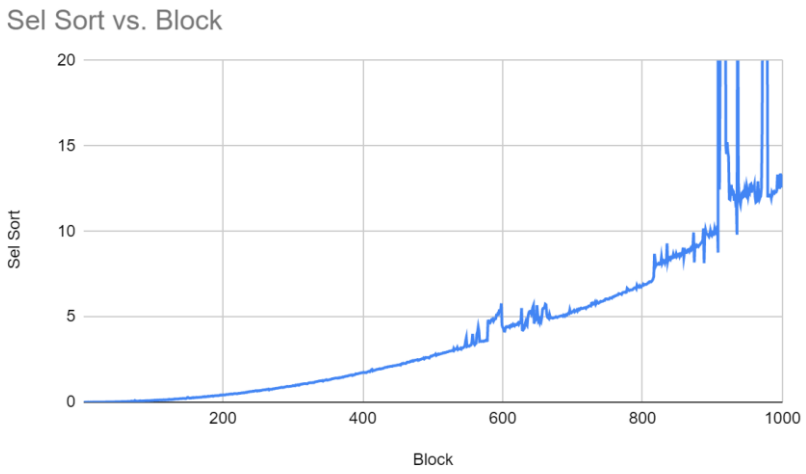
Block	Selection Sort	Insertion Sort
1	0.000000	0.000000
2	0.000000	0.000000
3	0.000000	0.000000
4	0.000000	0.000000
5	0.001000	0.000000
6	0.000000	0.000000
7	0.002000	0.000000

8	0.001000	0.001000
9	0.001000	0.001000
10	0.001000	0.001000
11	0.002000	0.001000
12	0.001000	0.002000
13	0.001000	0.001000
14	0.002000	0.001000
15	0.002000	0.002000
16	0.002000	0.003000
17	0.004000	0.003000
18	0.003000	0.003000
19	0.004000	0.003000
20	0.004000	0.003000
21	0.005000	0.003000
22	0.004000	0.000000
23	0.006000	0.008000
24	0.006000	0.000000
25	0.007000	0.000000
26	0.006000	0.008000
27	0.008000	0.008000
28	0.007000	0.008000
29	0.008000	0.008000
30	0.009000	0.008000
31	0.009000	0.000000
32	0.011000	0.008000
33	0.012000	0.009000
34	0.013000	0.000000
35	0.011000	0.015000
36	0.013000	0.000000
37	0.014000	0.015000
38	0.015000	0.016000
39	0.016000	0.000000
40	0.016000	0.016000
41	0.018000	0.017000
42	0.020000	0.015000
43	0.021000	0.015000
44	0.018000	0.016000
45	0.020000	0.016000
46	0.022000	0.015000
47	0.023000	0.024000
48	0.024000	0.016000
49	0.025000	0.016000
50	0.025000	0.016000

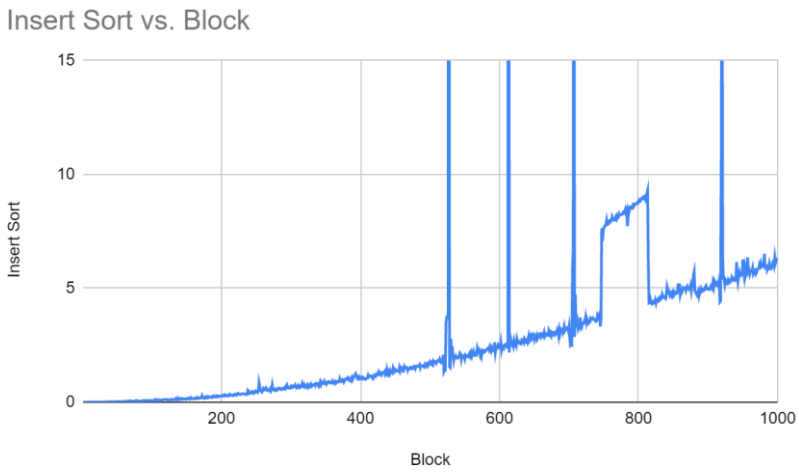
978	26.350000	5.960000
979	12.063000	5.959000
980	12.059000	5.956000
981	12.123000	6.515000
982	12.130000	5.952000
983	12.130000	5.995000
984	11.951000	5.991000
985	12.280000	5.796000
986	12.184000	5.878000
987	12.299000	5.909000
988	12.341000	6.067000
989	12.250000	5.860000
990	12.301000	5.874000
991	12.301000	5.860000
992	12.392000	5.962000
993	13.323000	6.149000
994	12.772000	6.194000
995	13.166000	6.037000
996	12.480000	6.356000
997	13.387000	6.101000
998	13.009000	6.198000
999	12.568000	6.290000
1000	13.336000	6.307000

GRAPHS:

- Selection sort:



- Insertion Sort



RESULT ANALYSIS:	Here we see that the total runtime taken by the Selection sort algorithm is around 4435.913 seconds and that by Insertion sort algorithm is around 2810.567 seconds. Time taken by both the algorithms to sort the blocks was comparatively same till block number 550, and then the selection sort algorithm took much more

	time tha insertion sort, almost double time to sort till block number 1000 Runtime of insertion sort is very less as compared to selection sort, almost by 50% less. Therefore the insertion sort algorithm is faster, stable and more efficient than selection sort algorithm • Time complexity: Selection: Worst case = Average Case = Best Case = O(n2) Insertion: Worst case = Average Case =O(n2) and Best Case=O(n) • Space Complexity: Selection sort: Has space complexity O(1) because we are not using any extra data structure apart from input array Insertion Sort: Since we use only a constant amount of additional memory apart from the input array, the space complexity is O(1).
CONCLUSION:	In this experiment I understood about two sorting algorithms i.e. selection and insertion sort and their implementation in c programming language. Also I learned about a new function which is high_resolution_clock::now() to calculate the running time.