Name	SINGH AKKSHIT JEEVAN PRAKASH
UID no.	2022300115
Experiment No.	1-B

AIM:	Experiment on finding the running time of an algorithm.
Program 1	
PROBLEM STATEMENT :	Details – The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts.
ALGORITHM:	Insertion Sort : Insertion Sort is a simple sorting algorithm that builds the final sorted array one element at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, it performs well for small datasets or partially sorted datasets. Algorithm: 1. Start with the assumption that the first element in the array is already sorted. 2. Iterate through the unsorted portion of the array. 3. For each element, compare it with the elements in the sorted portion and insert it at the correct position in the sorted portion. 4. Repeat this process until the entire array is sorted. Selection Sort: Selection Sort is another simple sorting algorithm that divides the input list into a sorted and an unsorted region. It repeatedly selects the smallest (or largest, depending on sorting order) element from the unsorted region and swaps it with the first element of the unsorted region. The process is repeated until the entire list is sorted. Algorithm: Divide the array into a sorted and an unsorted region. Find the smallest element in the unsorted region. Swap the smallest element with the first element in the unsorted region. Expand the sorted region to include the newly sorted element. Repeat steps 2-4 until the entire array is sorted.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
FILE *file;

void selection_sort(int *pArr, int size)
{
    int arr[size];
    clock_t start, end;
    double exec_time = 0;
    for(int i = 0; i < size; ++i)
    {
        arr[i] = pArr[i];
    }
    int min, temp;
    start = clock();
    for(int j = 0; j < size-1; ++j)
    {
        min = arr[j];
        int index;
        for(int k = j+1; k < size; ++k)
        {
            if(arr[k] < min)
            {
                min = arr[k];
```

```
                index = k;
            }
        }
        temp = arr[j];
        arr[j] = min;
        arr[index] = temp;
    }
    end = clock();

    exec_time = ((double)(end-start))/(CLOCKS_PER_SEC);
    fprintf(file, "%lf,", exec_time);
    printf("%lf ", exec_time);
}

void insertion_sort(int *pArr, int size)
{
    int arr[size];
    clock_t start, end;
    double exec_time = 0;
    for(int i = 0; i < size; ++i)
    {
        arr[i] = pArr[i];
    }
    int min, temp;
    start = clock();
    for(int i = 0: i < size-1: ++i)
```

```
    {
        int j = i+1;
        int temp;
        while(j > 0 && arr[j-1] > arr[j])
        {
            temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            --j;
        }
    }
    end = clock();
    exec_time = ((double)(end-start))/(CLOCKS_PER_SEC);

    fprintf(file, "%lf\n", exec_time);
    printf("%lf\n", exec_time);
}

int main()
{
    if(fopen("input.txt","r") == NULL)
    {
        srand(time(NULL));
        file = fopen("input.txt","w");
        for(int i = 0; i < 100000; ++i)
        {
```

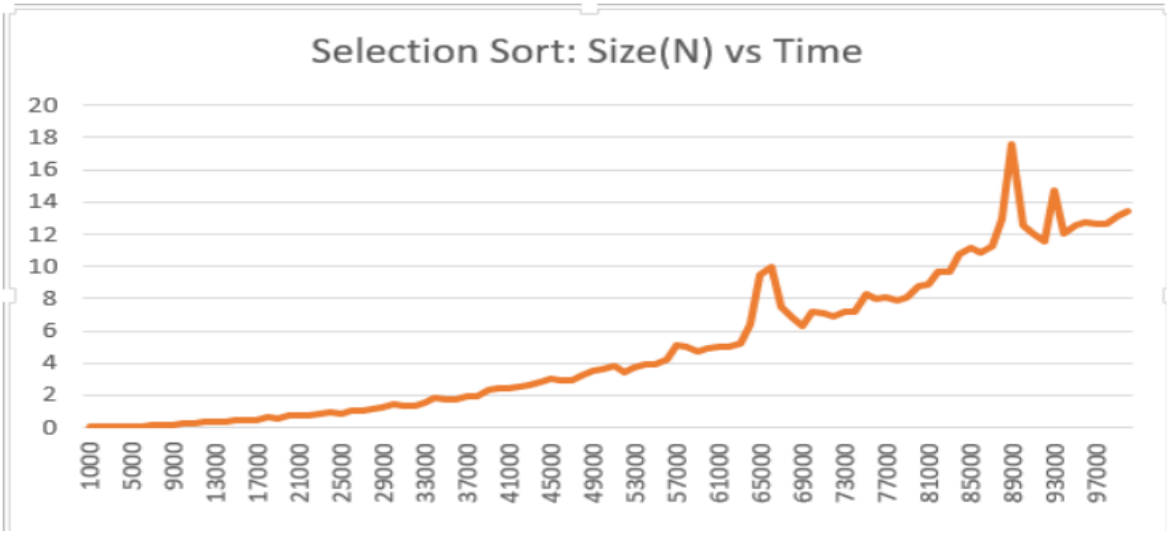
```
        }
        fclose(file);
    }
    file = fopen("input.txt","r");
    int arr[100000];
    for(int i = 0; i < 100000; ++i){
        fscanf(file, "%d ", &arr[i]);
    }
    fclose(file);
    file = fopen("Exp1b.csv", "w");
    for(int i = 1000; i <= 100000; i += 1000)
    {
        fprintf(file, "%d,", i);
        printf("%d ", i);
        selection_sort(arr, i);
        insertion_sort(arr, i);
    }
    fclose(file);
    return 0;
}
```

OUTPUT:

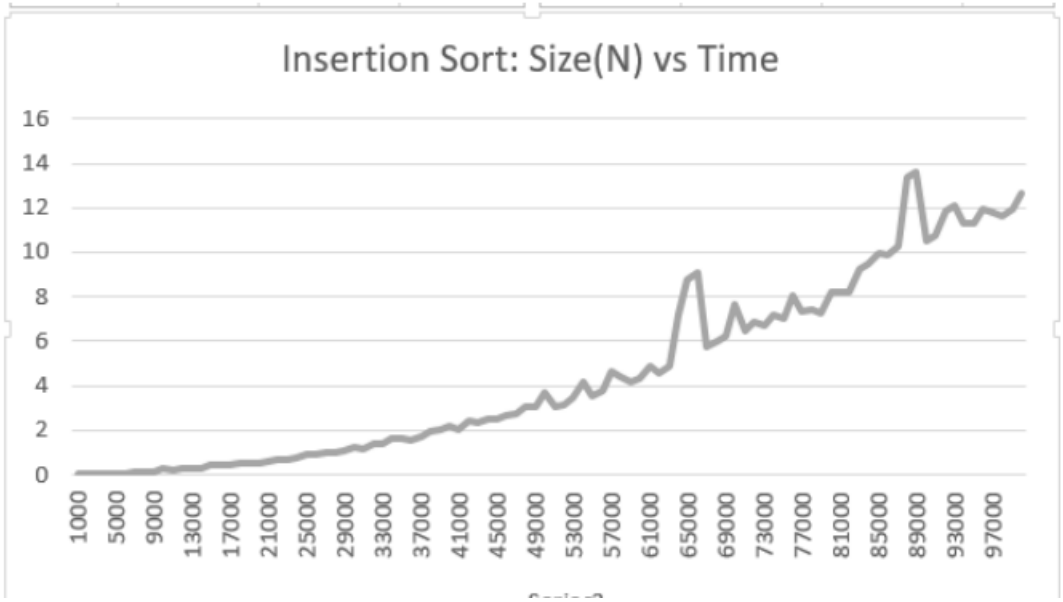
```
1000 0.000908 0.000755
2000 0.003126 0.003610
3000 0.006974 0.007786
4000 0.010626 0.016228
5000 0.015665 0.018827
6000 0.022480 0.032391
7000 0.030723 0.037210
8000 0.043776 0.049972
9000 0.055528 0.065049
10000 0.064209 0.076911
11000 0.131348 0.099297
12000 0.096072 0.108254
13000 0.114278 0.159290
14000 0.169096 0.166730
15000 0.151179 0.180967
16000 0.183397 0.206507
17000 0.216549 0.253061
18000 0.246222 0.250052
19000 0.272400 0.280560
20000 0.304821 0.310470
21000 0.318717 0.336444
22000 0.357625 0.373568
23000 0.395375 0.419515
24000 0.374565 0.467199
25000 0.474719 0.514361
26000 0.460450 0.513272
27000 0.473761 0.616796
28000 0.527384 0.625078
29000 0.680924 0.703279
30000 0.709554 0.690471
31000 0.636646 0.727623
32000 0.719946 0.846687
33000 0.779133 0.924630
34000 0.867789 0.994941
35000 0.877594 1.018254
36000 0.933933 1.122009
37000 1.001801 1.168405
38000 1.055377 1.203043
39000 1.137173 1.290671
40000 1.137328 1.341185
41000 1.173456 1.473739
42000 1.215749 1.465604
43000 1.323887 1.491371
44000 1.321119 1.487921
45000 1.411187 1.557231
46000 1.507203 1.614406
47000 1.403561 1.693199
48000 1.496585 1.796269
49000 1.619998 1.947272
50000 1.679187 1.999231
51000 1.804352 2.038177
```

	52000 1.742603 2.068044 53000 1.772352 2.155451 54000 1.871120 2.291755 55000 1.976136 2.343464 56000 2.065407 2.433174 57000 2.091364 2.561964 58000 2.158199 2.606557 59000 2.171967 2.643152 60000 2.249978 2.749769 61000 2.321841 2.904031 62000 2.499312 2.992481 63000 2.553716 3.124639 64000 2.648421 3.150813 65000 2.770380 3.265053 66000 2.781895 3.425667 67000 3.092477 3.624162 68000 3.099874 3.667735 69000 3.060823 3.772560 70000 3.347371 3.832635 71000 3.233856 3.954085 72000 3.445189 4.122563 73000 3.543212 4.169219 74000 3.696550 4.862852 75000 3.891628 4.573472 76000 4.246436 4.840965 77000 4.266723 4.965737 77000 4.266723 4.965737 78000 4.191983 5.077748 79000 4.427206 5.228092 80000 4.424204 5.287904 81000 4.577462 5.259495 82000 4.633811 5.429223 83000 4.683326 5.775010 84000 4.703893 5.602127 85000 5.211196 5.769873 86000 5.209520 5.849634 87000 5.492299 6.249032 88000 5.587462 6.435427 89000 5.468702 6.394370 90000 5.466674 6.690457 91000 5.945568 6.921987 92000 5.799971 6.789255 93000 6.008420 6.846794 94000 6.178999 7.304361 95000 6.113273 7.330443 96000 6.252469 7.097406 97000 6.304593 7.323462 98000 6.178972 7.493280 99000 6.476576 7.660903 100000 6.731687 7.939516

1. **Selection Sort**



2. **Insertion Sort**



RESULT: while Insertion Sort and Selection Sort are simple and have their merits, they may not be the optimal choices for large-scale applications with extensive datasets. The understanding of their strengths and limitations is essential when selecting sorting algorithms based on specific use cases and performance requirements.

Program 2	
PROBLEM STATEMENT :	
RESULT:	

Program 3	
PROBLEM STATEMENT:	
ALGORITHM:	
FLOWCHART:	
PROGRAM:	
RESULT:	
Program 4	
PROBLEM STATEMENT:	
ALGORITHM:	

FLOWCHART:	
PROGRAM:	
RESULT:	
Program 5	
PROBLEM STATEMENT:	
ALGORITHM:	
FLOWCHART:	

PROGRAM:	
RESULT:	
CONCLUSION:	