NAME:	Shubham Vishwakarma
UID No.	2021700071
BRANCH:	S.Y CSE-DS
BATCH:	D
SUBJECT	Design and Analysis of Algorithms
EXPERIMENT No.	1B
Date of Performance	30/01/2023
Date of Submission	06/01/2023

AIM:	**Experiment on finding the running time of an algorithm.**
Program 1	
PROBLEM STATEMENT :	For this experiment, you need to implement two sorting algorithms namely Insertion and Selection sort methods. Compare these algorithms based on time and space complexity. Time required to sorting algorithms can be performed using high_resolution_clock::now() under namespace std::chrono. You have togenerate1,00,000 integer numbers using C/C++ Rand function and save them in a text file. Both the sorting algorithms uses these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100 integers numbers with array indexes numbers A[0..99], A[0..199], A[0..299],..., A[0..99999]. You need to use high_resolution_clock::now() function to find the time required for 100, 200, 300.... 100000 integer numbers. Finally, compare two algorithms namely Insertion and Selection by plotting the time required to sort 100000 integers using LibreOffice Calc/MS Excel. The x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot representsthe tunning time to sort 1000 blocks of 100,200,300,...,100000 integer numbers. Note – You have to use C/C++ file processing functions for reading and writing randomly generated 100000 integer numbers.
ALGORITHM/ THEORY:	**Sorting Algorithms** Sorting Algorithms are a class of algorithms which are used to arrange the elements of a list or anarray in a particular order. There are a lot of sorting algorithms including bubble sort, selection sort, quick sort, insertion sort,heap sort, etc. This experiment focuses on insertion and selection sort. For

our examples, we will consider sorting in ascending order.

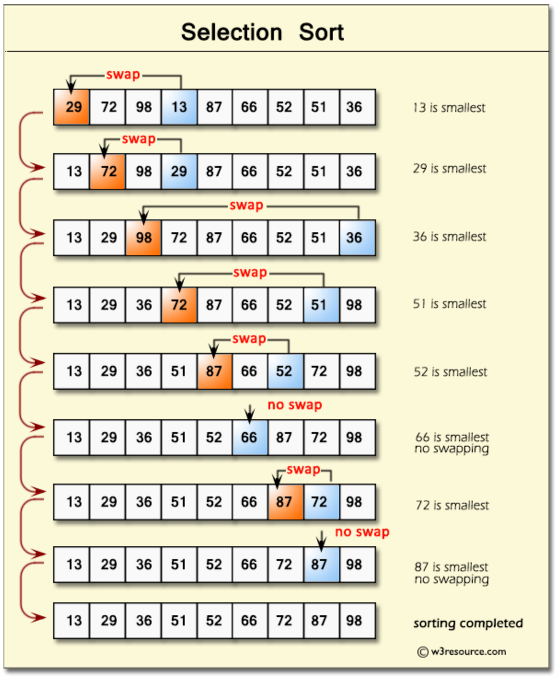
Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (consideringascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from theunsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:



The time complexities for selection sort are given below:

Best	Average	Worst
Ω(n^2)	θ(n^2)	O(n^2)

Since it always has an n^2 time complexity, it is inefficient

for large lists.We will discuss its algorithm later.

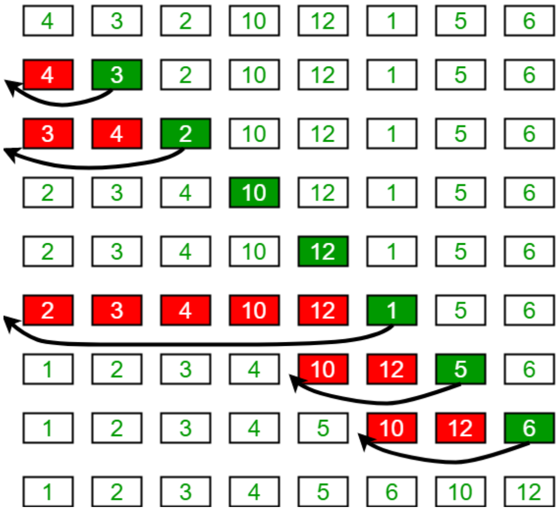
Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsortedpart are picked and placed at the correct position in the sorted part.

To sort an array of size n in ascending order:

a. Iterate from arr[1] to arr[n] over the array.
b. Compare the current element (key) to its predecessor.
c. If the key element is smaller than its predecessor, compare it to the elements before. Movethe greater elements one position up to make space for the swapped element.

Insertion Sort Execution Example



The time complexities for insertion sort are given below:

Best	Average	Worst
Ω(n)	θ(n^2)	O(n^2)

Since it always has a best-case time complexity of n, it is more efficient compared to selection sort.

Algorithms:

Selection Sort:

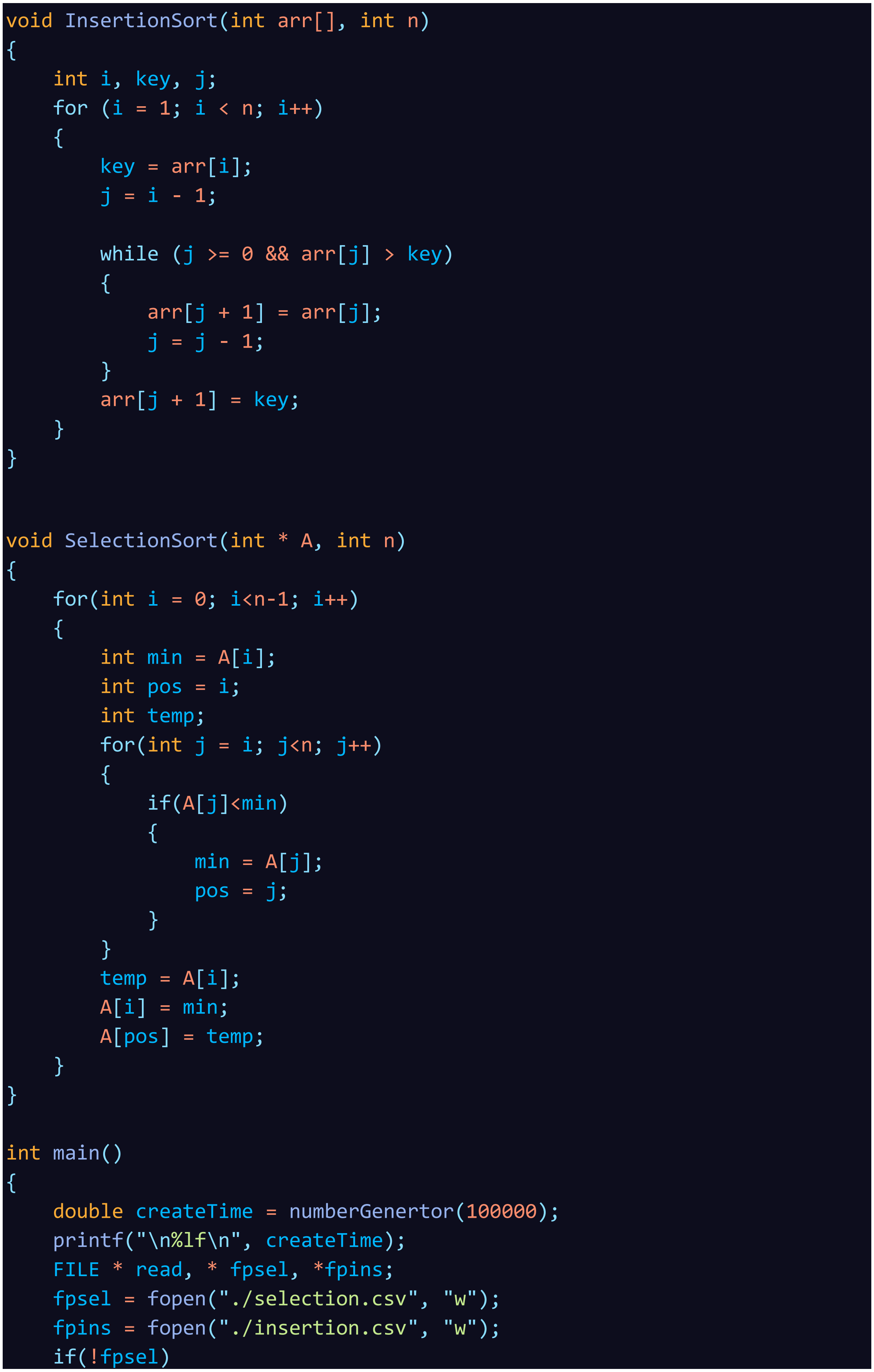
```
1)  START
2)  For i = 0 to n-2:
        a.  minIndex = i
        b.  For j = i+1 to n-1
                i.  If arr[minIndex] > arr[j]
   1. minIndex = j
        c.  Swap elements of array at positions i and minIndex
3)  END
```

Insertion Sort:

```
1) START
2) For i = 1 to n-1
    a. Key = arr[i]
    b. j = i – 1
    c. While j >= 0 and arr[j] > Key:
        i. arr[j+1] = arr[j]
        ii. j = j – 1
    d. arr[j+1] = Key
3) STOP
```

PROGRAM:

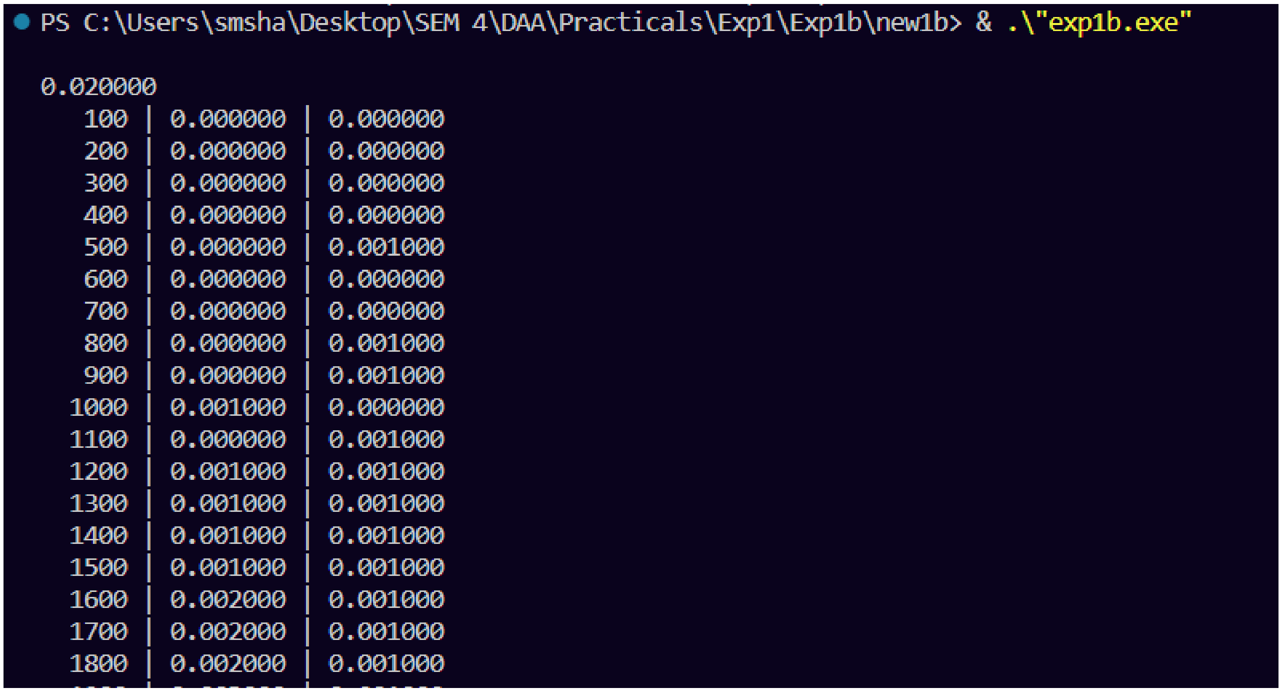
```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void print(int * A, int n)
{
    for(int i = 0; i<n; i++)
    {
        printf("%d ", A[i]);
    }
}

double numberGenertor(int n)
{
    FILE *fp = fopen("./num.txt", "w");
    clock_t start, end;
    double cpu_time_used;
    int p;
    start = clock();
    for(int i = 0; i < n; i++)
    {
        p = (rand() + (rand()*133));
        fprintf(fp," %d\n", p);

    }
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    fclose(fp);
    return cpu_time_used;
}
```



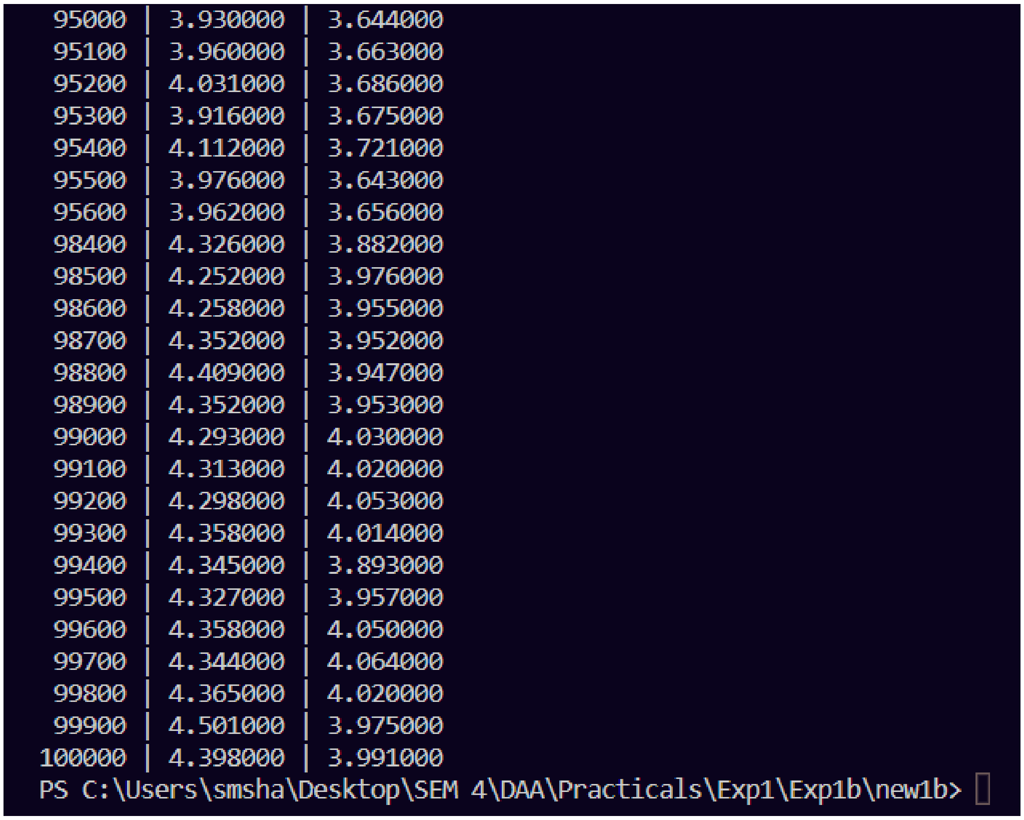
```c
void InsertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

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


void SelectionSort(int * A, int n)
{
    for(int i = 0; i<n-1; i++)
    {
        int min = A[i];
        int pos = i;
        int temp;
        for(int j = i; j<n; j++)
        {
            if(A[j]<min)
            {
                min = A[j];
                pos = j;
            }
        }
        temp = A[i];
        A[i] = min;
        A[pos] = temp;
    }
}

int main()
{
    double createTime = numberGenertor(100000);
    printf("\n%lf\n", createTime);
    FILE * read, * fpsel, *fpins;
    fpsel = fopen("./selection.csv", "w");
    fpins = fopen("./insertion.csv", "w");
    if(!fpsel)
```

```c
        return 0;
    if(!fpins)
        return 0;
    fprintf(fpsel, "Blocks, time\n");
    fprintf(fpins, "Blocks, time\n");

    for(long int x = 100; x<=100000; x+=100)
    {
        read = fopen("num.txt", "r");
        int A[x], B[x];
        clock_t start1, end1, start2, end2;
        // if(x % 1000 == 0)
        // {
        //     printf("x = %ld.\n", x);
        // }

        for(long int i = 0; i<x; i++)
        {
            fscanf(read, "%d\n", &A[i]);
            B[i] = A[i];
        }

        start1 = clock();
        SelectionSort(A, x);
        end1 = clock();

        start2 = clock();
        InsertionSort(B, x);
        end2 = clock();

        double t1 = (double) (end1 - start1) / CLOCKS_PER_SEC;
        double t2 = (double) (end2 - start2) / CLOCKS_PER_SEC;
        printf("%6d | %lf | %lf\n", x, t1, t2);

        fprintf(fpsel, "%ld, %f\n",x,t1);
        fprintf(fpins, "%ld, %f\n",x,t2);


        fclose(read);
    }
    fclose(fpsel);
    fclose(fpins);
}
```

RESULT:

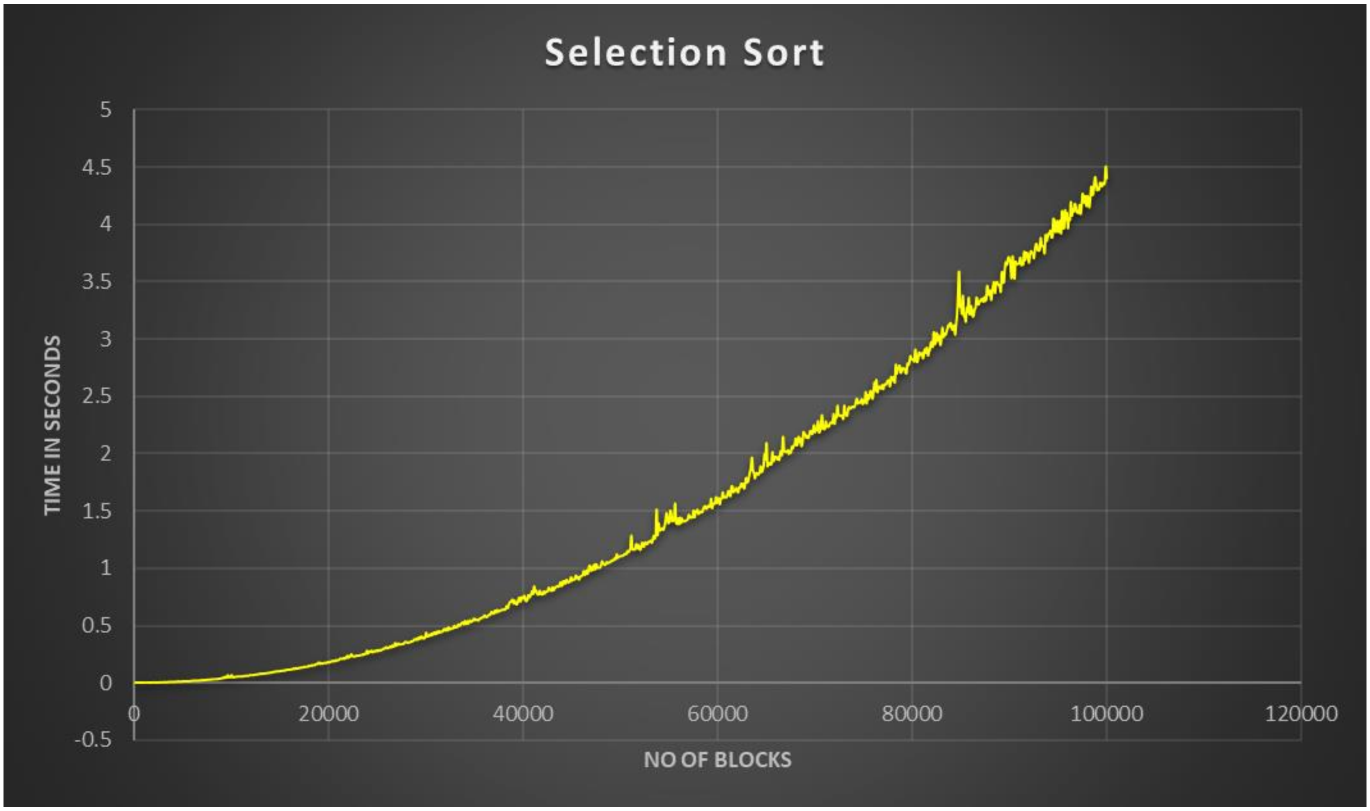


```
● PS C:\Users\smsha\Desktop\SEM 4\DAA\Practicals\Exp1\Exp1b\new1b> & .\"exp1b.exe"

 0.020000
    100 | 0.000000 | 0.000000
    200 | 0.000000 | 0.000000
    300 | 0.000000 | 0.000000
    400 | 0.000000 | 0.000000
    500 | 0.000000 | 0.001000
    600 | 0.000000 | 0.000000
    700 | 0.000000 | 0.000000
    800 | 0.000000 | 0.001000
    900 | 0.000000 | 0.001000
   1000 | 0.001000 | 0.000000
   1100 | 0.000000 | 0.001000
   1200 | 0.001000 | 0.001000
   1300 | 0.001000 | 0.001000
   1400 | 0.001000 | 0.001000
   1500 | 0.001000 | 0.001000
   1600 | 0.002000 | 0.001000
   1700 | 0.002000 | 0.001000
   1800 | 0.002000 | 0.001000
```

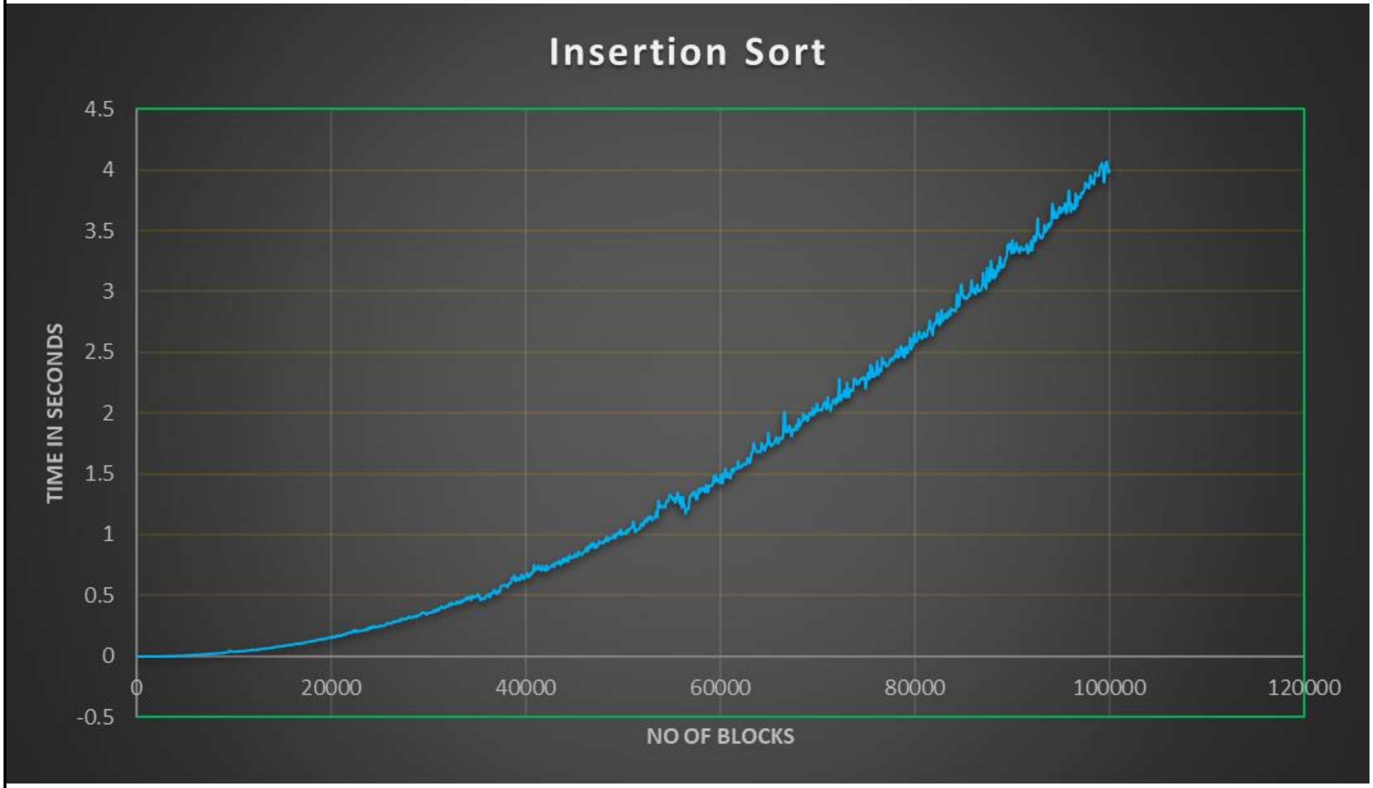


```
  95000 | 3.930000 | 3.644000
  95100 | 3.960000 | 3.663000
  95200 | 4.031000 | 3.686000
  95300 | 3.916000 | 3.675000
  95400 | 4.112000 | 3.721000
  95500 | 3.976000 | 3.643000
  95600 | 3.962000 | 3.656000
  98400 | 4.326000 | 3.882000
  98500 | 4.252000 | 3.976000
  98600 | 4.258000 | 3.955000
  98700 | 4.352000 | 3.952000
  98800 | 4.409000 | 3.947000
  98900 | 4.352000 | 3.953000
  99000 | 4.293000 | 4.030000
  99100 | 4.313000 | 4.020000
  99200 | 4.298000 | 4.053000
  99300 | 4.358000 | 4.014000
  99400 | 4.345000 | 3.893000
  99500 | 4.327000 | 3.957000
  99600 | 4.358000 | 4.050000
  99700 | 4.344000 | 4.064000
  99800 | 4.365000 | 4.020000
  99900 | 4.501000 | 3.975000
 100000 | 4.398000 | 3.991000
PS C:\Users\smsha\Desktop\SEM 4\DAA\Practicals\Exp1\Exp1b\new1b> 
```

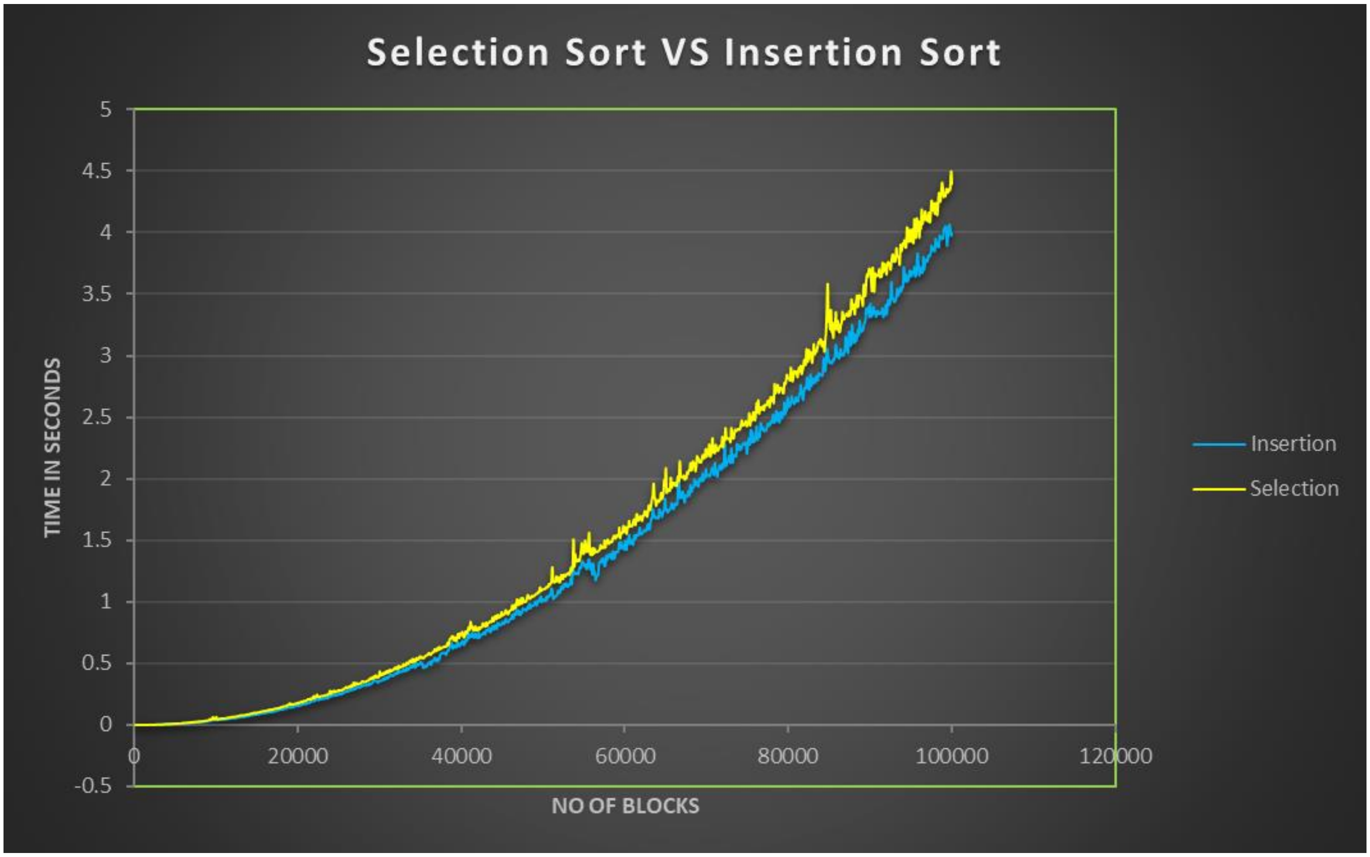
GRAPH:

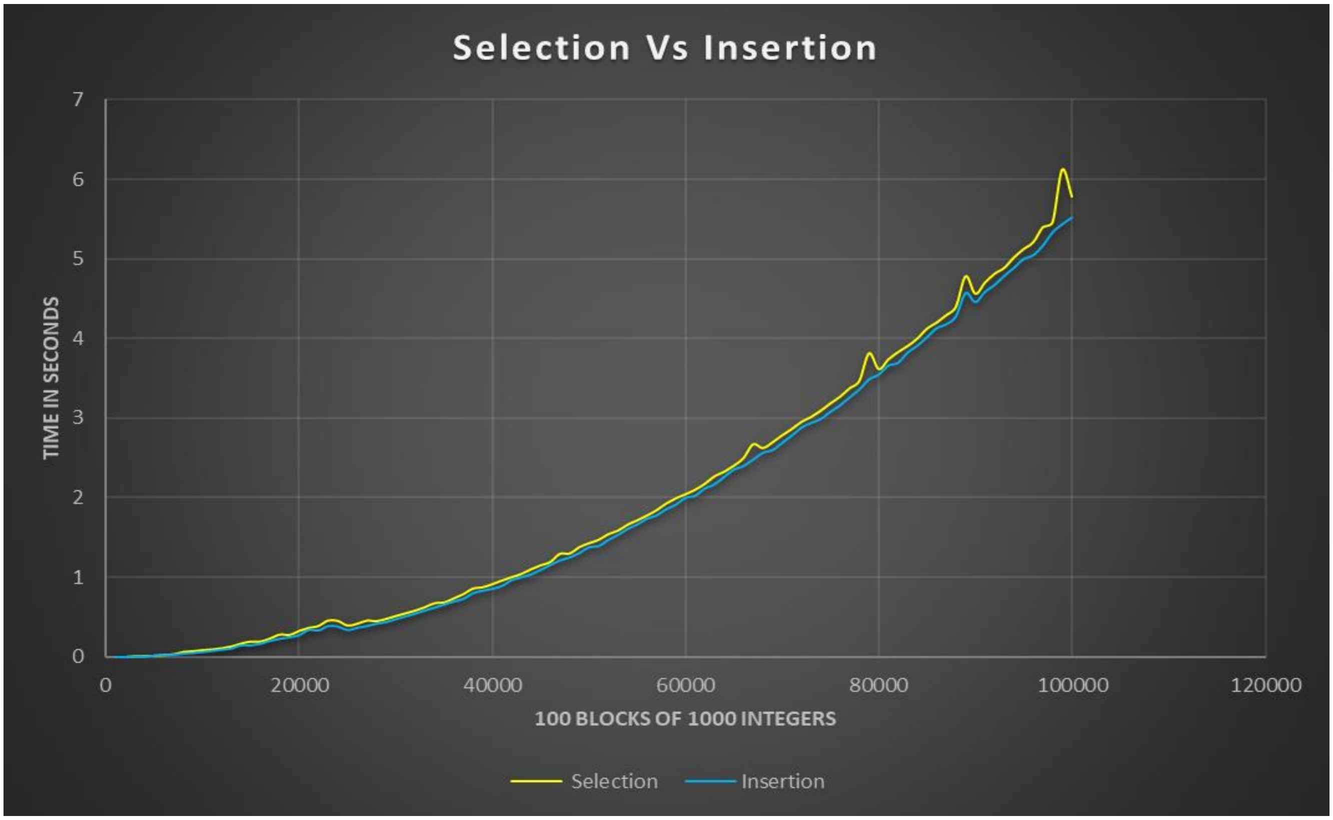


Selection Sort
TIME IN SECONDS
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
20000
40000
60000
80000
100000
120000
NO OF BLOCKS



Insertion Sort
TIME IN SECONDS
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
20000
40000
60000
80000
100000
120000
NO OF BLOCKS


Selection Sort VS Insertion Sort
TIME IN SECONDS
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
20000
40000
60000
80000
100000
120000
NO OF BLOCKS
Insertion
Selection


Selection Vs Insertion
TIME IN SECONDS
7
6
5
4
3
2
1
0
0
20000
40000
60000
80000
100000
120000
100 BLOCKS OF 1000 INTEGERS
Selection
Insertion

From the graph we can see that insertion sort have better running time than selection sort
The spikes in the graph due to following reasons

1. Due to a greater number of swaps at that given range
2. Due to CPU over heating (Machine Dependent)
3. Due to nature of random number (large difference between consecutive numbers)
4. Due to other process running in the background during the execution of program

	Best	**Average**	**Worst**
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)

The time complexity for all 3 cases of selection sort is n^2. Meanwhile in insertion sort, the average and worst-case time complexity is the same – n^2 but the best-case time complexity is linear (in the case where the input is already sorted).

Hence, we can say that in general, insertion sort is more efficient compared to selection sort.

CONCLUSION :	In this experiment, we wrote a program which given 100000 integers in 1000 blocks of 100 numbers, uses the selection andinsertion sort algorithms to sort them in ascending order. In selection sort, during every iteration, we move the minimum element of the unsorted subarray tothe end of the sorted subarray. In insertion sort, during every iteration, we move the first element of the unsorted subarray to the appropriate position in the sorted subarray. We compared the running time of insertion and selection sort by comparing the running time graph of 2 algorithms. We also analyzed the 2 algorithms in which we found that while the time complexity of selection sort is n^2, the best-case time complexity of insertion sort is linear (in the case where the input is already sorted). Hence, we concluded that in general, insertion sort is

	more efficient compared to selection sort. Thus, we successfully accomplished the aim of this experiment.