DAA EXPERIMENT NO. 1-B

NAME: Glen Dsouza CSE DS (BATCH A)

UID: 2022701002

AIM: Experiment on finding the running time of an algorithm.

Problem Definition & Assumptions – 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:

Selection Sort Function:

Step 1: Take an array arr of size size.

Step 2: For each element i in the array arr, from 0 to size-1, do the following:

a. Initialize a variable min with i.
b. For each element j in the array arr, from i+1 to size-1, do the following:
    i. If `arr[j]` is less than `arr[min]`, set `min` to `j`.
c. Swap the element at index min with the element at index i.

Step 3: The array arr is now sorted in ascending order.

Insertion Sort Function:

Step 1: Take an array arr of size n.

Step 2: For each element i in the array arr, from 1 to n-1, do the following:

a. Initialize a variable key with arr[i].
b. Initialize a variable j with i-1.
c. While j is greater than or equal to 0 and arr[j] is greater than key, do the following:
    i. Set `arr[j+1]` to `arr[j]`.
    ii. Decrement `j` by 1.
d. Set arr[j+1] to key.

Step 3: The array arr is now sorted in ascending order.

Main Function:

Step 1: Include the required libraries stdio.h, stdlib.h, time.h, and limits.h.

Step 2: Define two sorting functions selection_sort and insertion_sort to sort the array elements.

Step 3: In the main function, open a file "exp1b.txt" for writing and initialize the random number generator with srand((unsigned int) time(NULL)).

Step 4: Generate 1000 blocks of 100 random numbers each and store them in the file.

Step 5: Close the file after writing.

Step 6: Open the file "exp1b.txt" for reading.

Step 7: For each block of 100 elements, read the elements from the file into two arrays arr and arr1.

Step 8: Sort the elements in the arr using the selection_sort function.

Step 9: Measure the time taken for sorting using the clock() function and store it in the time_taken_selection_sort variable.

Step 10: Sort the elements in the arr1 using the insertion_sort function.

Step 11: Measure the time taken for sorting using the clock() function and store it in the time_taken_insertion_sort variable.

Step 12: Print the block number, time taken for selection sort, and time taken for insertion sort.

Step 13: Repeat the process for 500 blocks.

Step 14: Close the file after reading.

CODE:

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

void selection_sort(int arr[],int size) {
    for(int i=0;i<size-1;i++) {
        int min=i;
        for(int j=i+1;j<size;j++)
            if(arr[j]<arr[min])
                min = j;
        int temp = arr[min];
        arr[min] = arr[i];
        arr[i] = temp;
    }
}

void insertion_sort(int arr[],int n) {
    int i,key,j;
    for(int 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 main() {
    FILE *fp;
    fp  = fopen ("exp1b.txt", "w");
    srand((unsigned int) time(NULL));
    for(int block=0;block<1000;block++) {
        for(int i=0;i<100;i++) {
            int number = (int)(((float) rand() / (float)(RAND_MAX))*100000);
            fprintf(fp,"%d ",number);
        }
        fputs("\n",fp);
    }
    fclose (fp);

   fp = fopen("exp1b.txt", "r");
   printf("Block\tSelection_sort\tInsertion_sort\n");
   for(int block=0;block<500;block++) {
```

```
    clock_t t,t1;

    int arr[(block+1)*100];
    int arr1[(block+1)*100];
    for(int i=0;i<(block+1)*100;i++){
        fscanf(fp, "%d", &arr[i]);
        arr1[i] = arr[i];
    }
    fseek(fp, 0, SEEK_SET);
    t = clock();
    selection_sort(arr,(block+1)*100);
    t = clock() - t;
    t1 = clock();
    insertion_sort(arr1,(block+1)*100);
    t1 = clock() - t1;
    double time_taken_selection_sort = ((double)t)/CLOCKS_PER_SEC;
    double time_taken_insertion_sort = ((double)t1)/CLOCKS_PER_SEC;
    printf("%d\t%f\t%f\n",(block+1),time_taken_selection_sort,time_taken_insertion_sort);
   }
    fclose(fp);
}
```

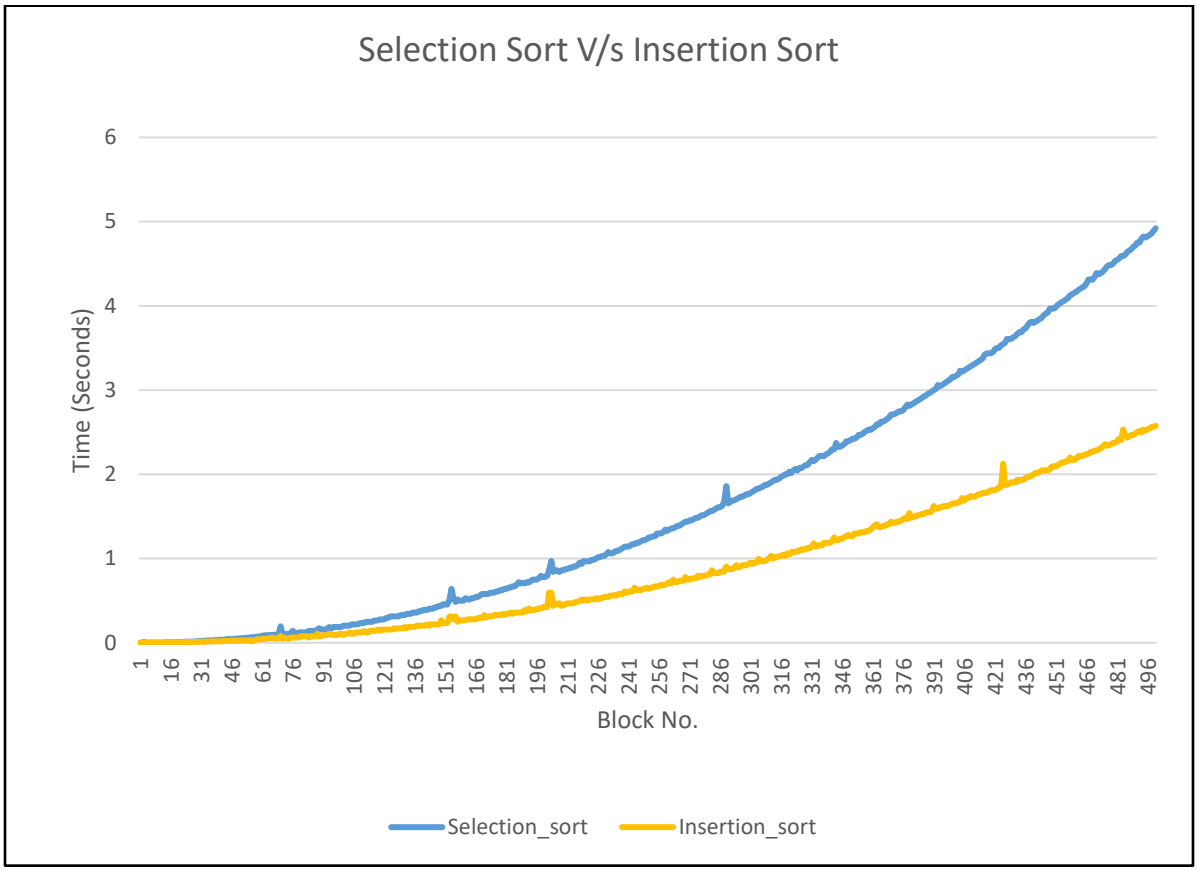
OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                   Code

PS D:\Documents\Desktop\Glen\S.P.I.T\2nd Year\SEM IV\DAA\PRACS> cd "d:\Documents\Desktop\Glen\S.P.I.T\2nd Year\SEM IV\DAA\PRACS\" ; if ($?) { gcc EXP1B.c
XP1B } ; if ($?) { .\EXP1B }
Block   Selection_sort  Insertion_sort
1       0.000000        0.000000
2       0.000000        0.000000
3       0.011000        0.000000
4       0.001000        0.000000
5       0.001000        0.000000
6       0.001000        0.000000
7       0.001000        0.001000
8       0.002000        0.000000
9       0.002000        0.001000
10      0.002000        0.002000
11      0.003000        0.001000
12      0.003000        0.002000
13      0.003000        0.004000
14      0.005000        0.003000
15      0.006000        0.004000
16      0.006000        0.004000
17      0.006000        0.004000
18      0.006000        0.004000
19      0.008000        0.004000
20      0.010000        0.004000
21      0.011000        0.004000
22      0.011000        0.006000
23      0.012000        0.008000
24      0.014000        0.008000
25      0.014000        0.009000
26      0.015000        0.009000
27      0.015000        0.010000
28      0.017000        0.009000
29      0.018000        0.010000
30      0.020000        0.012000
```

```
470     4.343000        2.280000
471     4.390000        2.281000
472     4.374000        2.296000
473     4.390000        2.312000
474     4.405000        2.328000
475     4.435000        2.359000
476     4.467000        2.344000
477     4.483000        2.343000
478     4.484000        2.359000
479     4.499000        2.374000
480     4.530000        2.374000
481     4.546000        2.390000
482     4.561000        2.421000
483     4.592000        2.406000
484     4.593000        2.531000
485     4.608000        2.469000
486     4.639000        2.437000
487     4.655000        2.453000
488     4.671000        2.468000
489     4.702000        2.468000
490     4.718000        2.484000
491     4.749000        2.500000
492     4.749000        2.515000
493     4.796000        2.499000
494     4.823000        2.531000
495     4.812000        2.515000
496     4.827000        2.531000
497     4.842000        2.547000
498     4.859000        2.562000
499     4.890000        2.561000
500     4.921000        2.577000
```

RESULT:



RESULT ANALYSIS:

The following graph is representation of amount of time (in seconds) required to sort block of integers using Selection sort & Insertion sort algorithm.

In the above graph, time values of sorting algorithm are plotted on y-axis against no. of blocks on x-axis. The maximum no. of block is 500 on X-axis.

Maximum amount of time required to sort 500^{th} block using selection sort is approx. 4.92 seconds and using insertion sort is 2.57 seconds.

No sudden major spikes were observed (2-3 small spikes are visible which are negligible).

As the no. of integers in blocks increases both the lines for selection sort and insertion sort grow exponentially and not linearly.

From the above graph it can be derived that amount of time required to sort a block using selection sort increases quickly compared to insertion sort as no. of integers in block increases.

CONCLUSION: In this experiment run time of selection sort and insertion sort was found and plotted on a graph.