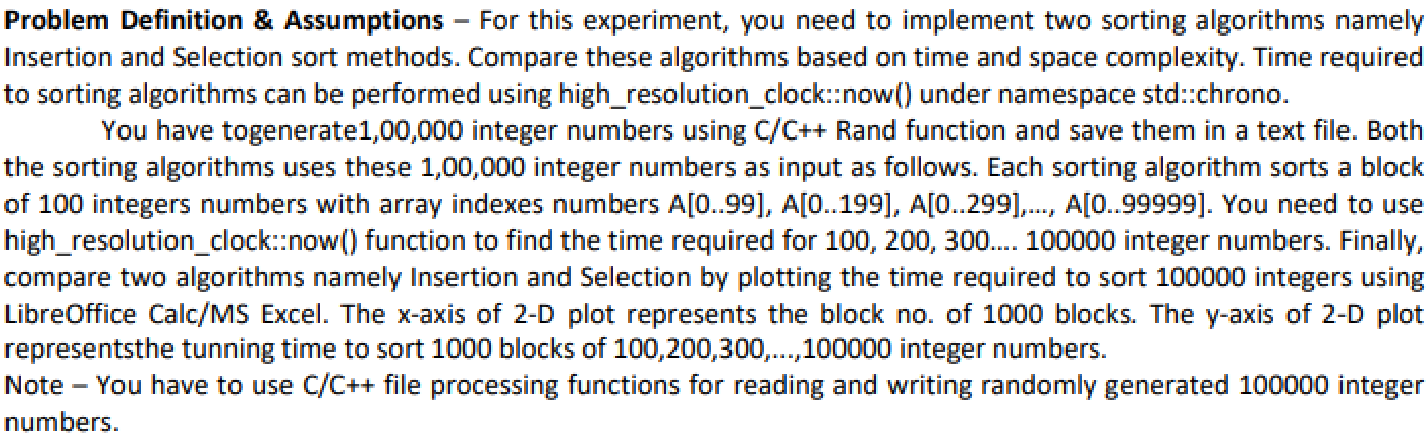
DAA EXPERIMENT NO. 1-B

NAME: Glen Dsouza CSE DS (BATCH A)

UID: 2022701002

AIM: Experiment on finding the running time, no. of comparisons and no. of swaps 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:

<u>Selection Sort Function:</u>

```
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.
```

<u>Insertion Sort Function:</u>

```
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 "random.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.

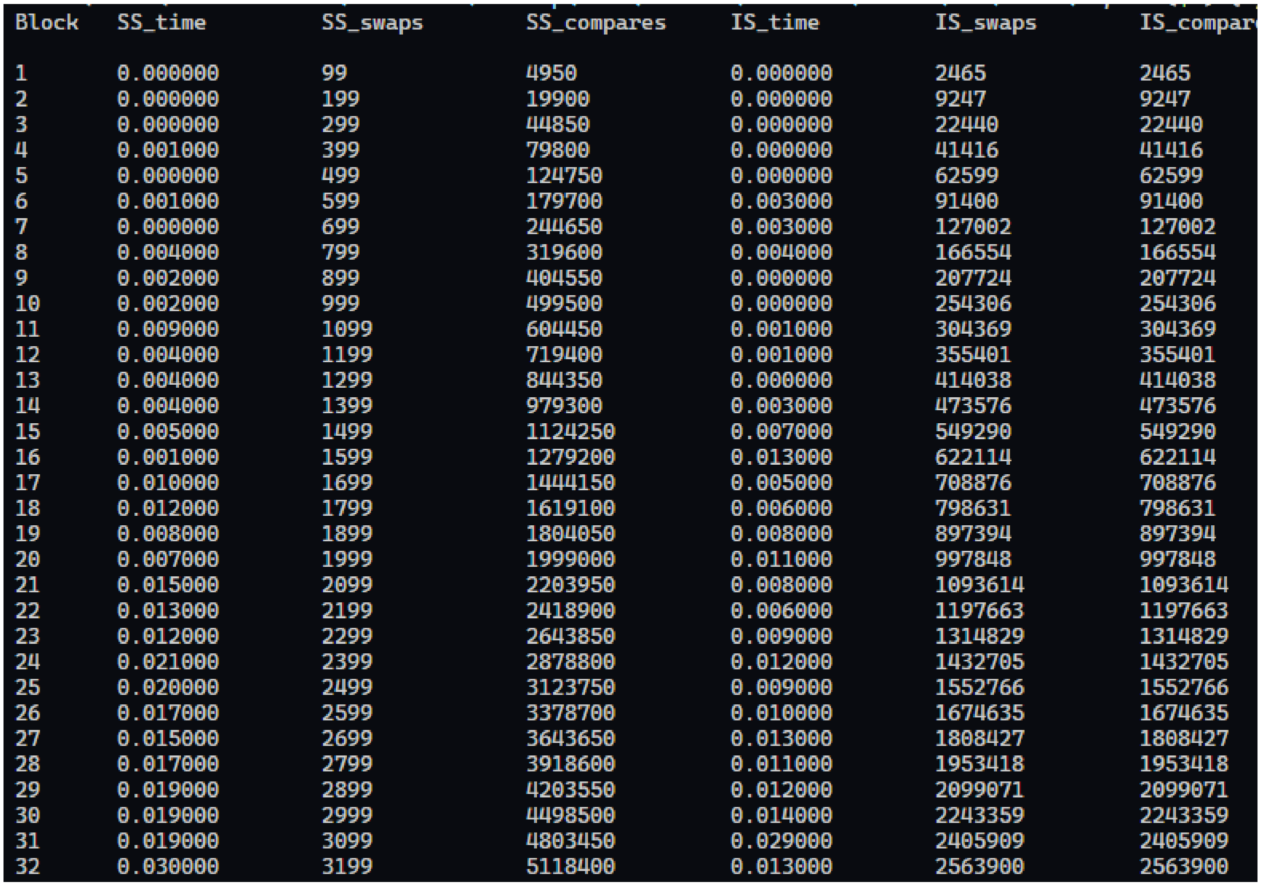
Step 13: Repeat the process for 500 blocks.

Step 14: Close the file after reading.

CODE:

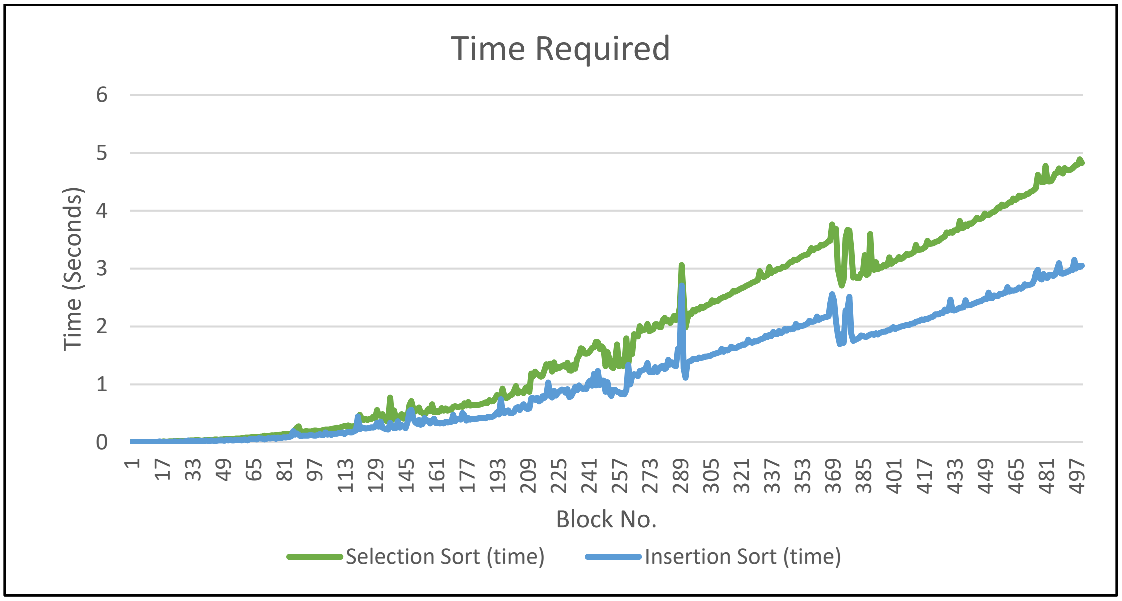
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<limits.h>

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

}

void insertion_sort(int arr[],int n, int *is_swaps, int *is_compares) {
    int i,key,j;
    for(int i=1;i<n;i++) {
        key = arr[i];
        j=i-1;
        while(j>=0 && arr[j]>key) {
            (*is_compares)++;
            arr[j+1] = arr[j];
            j=j-1;
            (*is_swaps)++;
        }
        arr[j+1] = key;
    }
}

void main() {
    FILE *fp;
    fp  = fopen ("random.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("random.txt", "r");
   printf("Block\t%-8s\t%-8s\t%-8s\t%-8s\t%-8s\t%-8s\n\n","SS_time", "SS_swaps",
"SS_compares","IS_time","IS_swaps","IS_compares");
   for(int block=0;block<500;block++) {
    int ss_swaps = 0, ss_compares = 0, is_swaps = 0, is_compares = 0;
    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, &ss_swaps, &ss_compares);
    t = clock() - t;
    t1 = clock();
    insertion_sort(arr1,(block+1)*100, &is_swaps, &is_compares);
    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%-8f\t%-8d\t%-8d\t%-8f\t%-8d\t%-8d\n",(block+1),time_taken_selection_sort, ss_swaps, ss_compares,
time_taken_insertion_sort, is_swaps, is_compares);
   }
    fclose(fp);
}
```

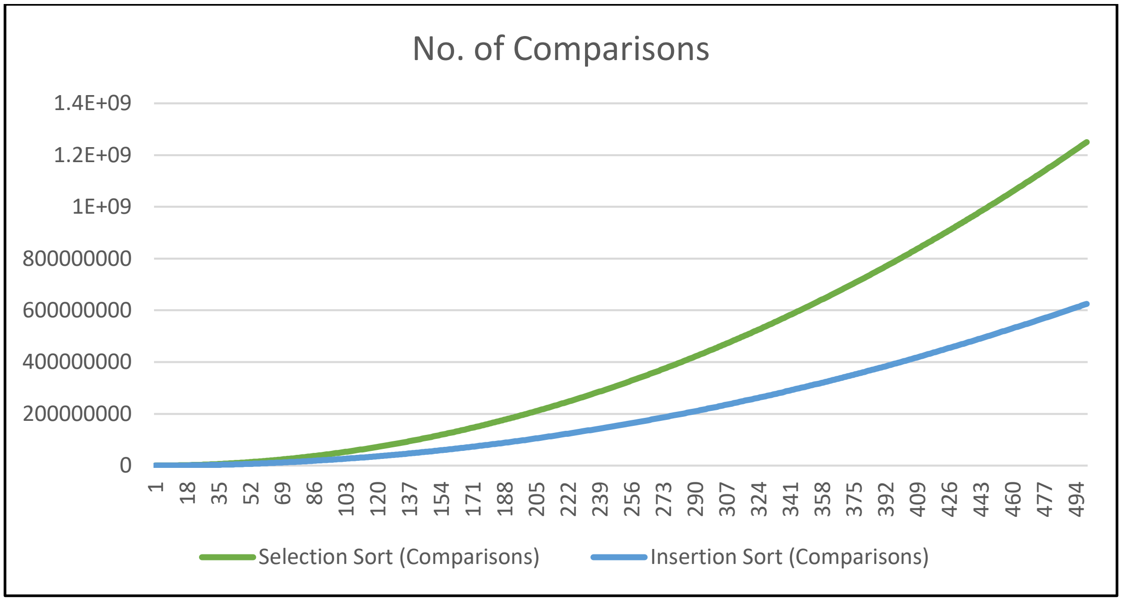
OUTPUT:



```
Block   SS_time         SS_swaps        SS_compares     IS_time         IS_swaps        IS_compar

1       0.000000        99              4950            0.000000        2465            2465
2       0.000000        199             19900           0.000000        9247            9247
3       0.000000        299             44850           0.000000        22440           22440
4       0.001000        399             79800           0.000000        41416           41416
5       0.000000        499             124750          0.000000        62599           62599
6       0.001000        599             179700          0.003000        91400           91400
7       0.000000        699             244650          0.003000        127002          127002
8       0.004000        799             319600          0.004000        166554          166554
9       0.002000        899             404550          0.000000        207724          207724
10      0.002000        999             499500          0.000000        254306          254306
11      0.009000        1099            604450          0.001000        304369          304369
12      0.004000        1199            719400          0.001000        355401          355401
13      0.004000        1299            844350          0.000000        414038          414038
14      0.004000        1399            979300          0.003000        473576          473576
15      0.005000        1499            1124250         0.007000        549290          549290
16      0.001000        1599            1279200         0.013000        622114          622114
17      0.010000        1699            1444150         0.005000        708876          708876
18      0.012000        1799            1619100         0.006000        798631          798631
19      0.008000        1899            1804050         0.008000        897394          897394
20      0.007000        1999            1999000         0.011000        997848          997848
21      0.015000        2099            2203950         0.008000        1093614         1093614
22      0.013000        2199            2418900         0.006000        1197663         1197663
23      0.012000        2299            2643850         0.009000        1314829         1314829
24      0.021000        2399            2878800         0.012000        1432705         1432705
25      0.020000        2499            3123750         0.009000        1552766         1552766
26      0.017000        2599            3378700         0.010000        1674635         1674635
27      0.015000        2699            3643650         0.013000        1808427         1808427
28      0.017000        2799            3918600         0.011000        1953418         1953418
29      0.019000        2899            4203550         0.012000        2099071         2099071
30      0.019000        2999            4498500         0.014000        2243359         2243359
31      0.019000        3099            4803450         0.029000        2405909         2405909
32      0.030000        3199            5118400         0.013000        2563900         2563900
```

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 500th block using selection sort is approx. 4.92 seconds and using insertion sort is 3.00 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, no. of comparisons and no. of swaps of selection sort and insertion sort was found and plotted on a graph.