BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	1B

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 sort 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 use these 1,00,000 integer numbers as input as follows. Each sorting algorithm sorts a block of 100 integer 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 the 2-D plot represents the block no. of 1000 blocks. The y-axis of the 2-D plot represents the running 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. Input – 1) Each student have to generate random 100000 numbers using rand() function and use this input as 1000 blocks of 100,200,300,...,100000 integer numbers to Insertion and Selection sorting algorithms. Output – 1) Store the randomly generated 100000 integer numbers to a text file. 2) Draw a 2D plot of both sorting algorithms such that the x-axis of 2-D plot represents the block no. of 1000 blocks. The y-axis of 2-D plot represents the running time to sort 1000 blocks of 100,200,300,...,100000

integer numbers.
3) Comment on Space complexity for two sorting algorithms.

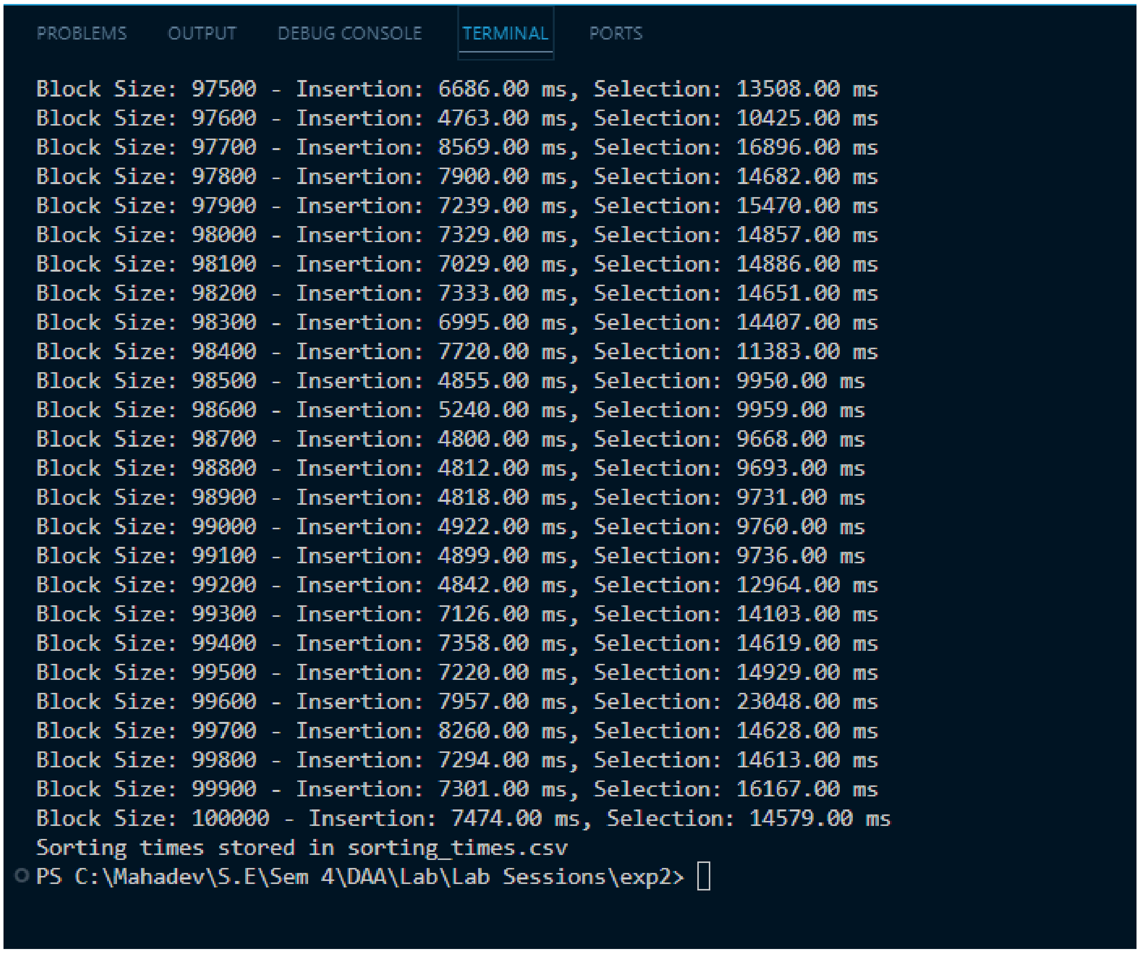
Average Case

PROGRAM:

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

#define DATA_FILE "random_numbers.txt"
#define TIME_FILE "sorting_times.csv"
#define NUM_COUNT 100000

void generateRandomNumbers()
{
    FILE *outFile = fopen(DATA_FILE, "w");
    if (outFile == NULL)
    {
        printf("Error opening file!\n");
        return;
    }
    srand(time(0));
    for (int i = 0; i < NUM_COUNT; i++)
    {
        fprintf(outFile, "%d ", rand() % 1000000);
    }
    fclose(outFile);
}

void readNumbers(int *numbers, int count)
{
    FILE *inFile = fopen(DATA_FILE, "r");
    if (inFile == NULL)
    {
        printf("Error opening file!\n");
        return;
    }
    for (int i = 0; i < count; i++)
    {
        fscanf(inFile, "%d", &numbers[i]);
    }
    fclose(inFile);
}

void insertionSort(int *arr, int n)
{
    for (int i = 1; i < n; i++)
    {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        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;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```

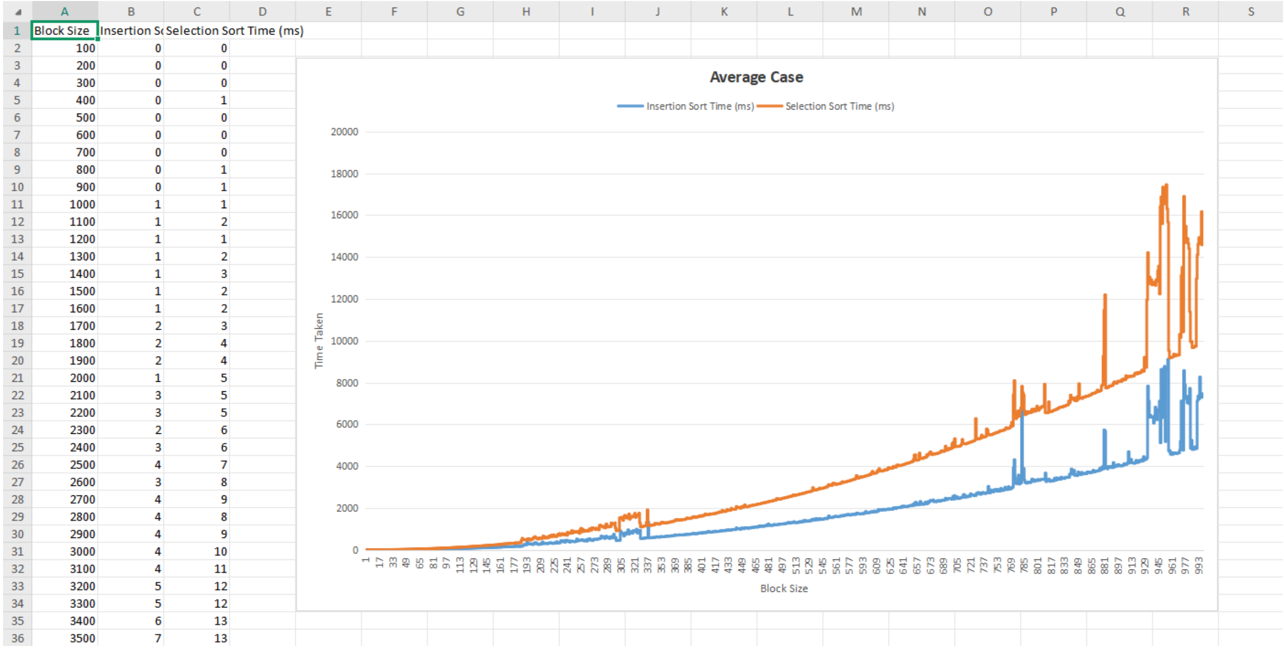
```
void measureSortingTime()
{
    FILE *timeFile = fopen(TIME_FILE, "w");
    if (timeFile == NULL)
    {
        printf("Error opening file!\n");
        return;
    }
    fprintf(timeFile, "Block Size,Insertion Sort Time (ms),Selection Sort
Time (ms)\n");

    int *numbers = (int *)malloc(NUM_COUNT * sizeof(int));
    if (numbers == NULL)
    {
        printf("Memory allocation failed!\n");
        return;
    }
    readNumbers(numbers, NUM_COUNT);

    for (int blockSize = 100; blockSize <= NUM_COUNT; blockSize +=
100)
    {
        int *tempInsertion = (int *)malloc(blockSize * sizeof(int));
        int *tempSelection = (int *)malloc(blockSize * sizeof(int));
        if (tempInsertion == NULL || tempSelection == NULL)
        {
            printf("Memory allocation failed!\n");
            return;
        }

        for (int i = 0; i < blockSize; i++)
        {
            tempInsertion[i] = numbers[i];
            tempSelection[i] = numbers[i];
        }

        clock_t start = clock();
        insertionSort(tempInsertion, blockSize);
        clock_t stop = clock();
        double insertionTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC; // Convert to milliseconds

        start = clock();
        selectionSort(tempSelection, blockSize);
        stop = clock();
        double selectionTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC;

        fprintf(timeFile, "%d,%.2f,%.2f\n", blockSize, insertionTime,
selectionTime);
        printf("Block Size: %d - Insertion: %.2f ms, Selection: %.2f ms\n",
blockSize, insertionTime, selectionTime);

        free(tempInsertion);
        free(tempSelection);
    }

    fclose(timeFile);
    free(numbers);
}

int main()
{
    generateRandomNumbers();

    measureSortingTime();

    printf("Sorting times stored in %s\n", TIME_FILE);
    return 0;
}
```

RESULT:



```
Block Size: 97500 - Insertion: 6686.00 ms, Selection: 13508.00 ms
Block Size: 97600 - Insertion: 4763.00 ms, Selection: 10425.00 ms
Block Size: 97700 - Insertion: 8569.00 ms, Selection: 16896.00 ms
Block Size: 97800 - Insertion: 7900.00 ms, Selection: 14682.00 ms
Block Size: 97900 - Insertion: 7239.00 ms, Selection: 15470.00 ms
Block Size: 98000 - Insertion: 7329.00 ms, Selection: 14857.00 ms
Block Size: 98100 - Insertion: 7029.00 ms, Selection: 14886.00 ms
Block Size: 98200 - Insertion: 7333.00 ms, Selection: 14651.00 ms
Block Size: 98300 - Insertion: 6995.00 ms, Selection: 14407.00 ms
Block Size: 98400 - Insertion: 7720.00 ms, Selection: 11383.00 ms
Block Size: 98500 - Insertion: 4855.00 ms, Selection: 9950.00 ms
Block Size: 98600 - Insertion: 5240.00 ms, Selection: 9959.00 ms
Block Size: 98700 - Insertion: 4800.00 ms, Selection: 9668.00 ms
Block Size: 98800 - Insertion: 4812.00 ms, Selection: 9693.00 ms
Block Size: 98900 - Insertion: 4818.00 ms, Selection: 9731.00 ms
Block Size: 99000 - Insertion: 4922.00 ms, Selection: 9760.00 ms
Block Size: 99100 - Insertion: 4899.00 ms, Selection: 9736.00 ms
Block Size: 99200 - Insertion: 4842.00 ms, Selection: 12964.00 ms
Block Size: 99300 - Insertion: 7126.00 ms, Selection: 14103.00 ms
Block Size: 99400 - Insertion: 7358.00 ms, Selection: 14619.00 ms
Block Size: 99500 - Insertion: 7220.00 ms, Selection: 14929.00 ms
Block Size: 99600 - Insertion: 7957.00 ms, Selection: 23048.00 ms
Block Size: 99700 - Insertion: 8260.00 ms, Selection: 14628.00 ms
Block Size: 99800 - Insertion: 7294.00 ms, Selection: 14613.00 ms
Block Size: 99900 - Insertion: 7301.00 ms, Selection: 16167.00 ms
Block Size: 100000 - Insertion: 7474.00 ms, Selection: 14579.00 ms
Sorting times stored in sorting_times.csv
PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> 
```

EXCEL OUTPUT :



	A	B	C
1	Block Size	Insertion Sc	Selection Sort Time (ms)
2	100	0	0
3	200	0	0
4	300	0	0
5	400	0	1
6	500	0	0
7	600	0	0
8	700	0	0
9	800	0	1
10	900	0	1
11	1000	1	1
12	1100	1	2
13	1200	1	1
14	1300	1	2
15	1400	1	3
16	1500	1	2
17	1600	1	2
18	1700	2	3
19	1800	2	4
20	1900	2	4
21	2000	1	5
22	2100	3	5
23	2200	3	5
24	2300	2	6
25	2400	3	6
26	2500	4	7
27	2600	3	8
28	2700	4	9
29	2800	4	8
30	2900	4	9
31	3000	4	10
32	3100	4	11
33	3200	5	12
34	3300	5	12
35	3400	6	13
36	3500	7	13

Best and Worst Cases

PROGRAM:

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

#define DATA_FILE "random_nos.txt"
#define TIME_FILE "times.csv"
#define NUM_COUNT 100000

void generateRandomNumbers() {
    FILE *outFile = fopen(DATA_FILE, "w");
    if (outFile == NULL) {
        printf("Error opening file!\n");
        return;
    }
    srand(time(0));
    for (int i = 0; i < NUM_COUNT; i++) {
        fprintf(outFile, "%d ", rand() % 1000000);
    }
    fclose(outFile);
}
```

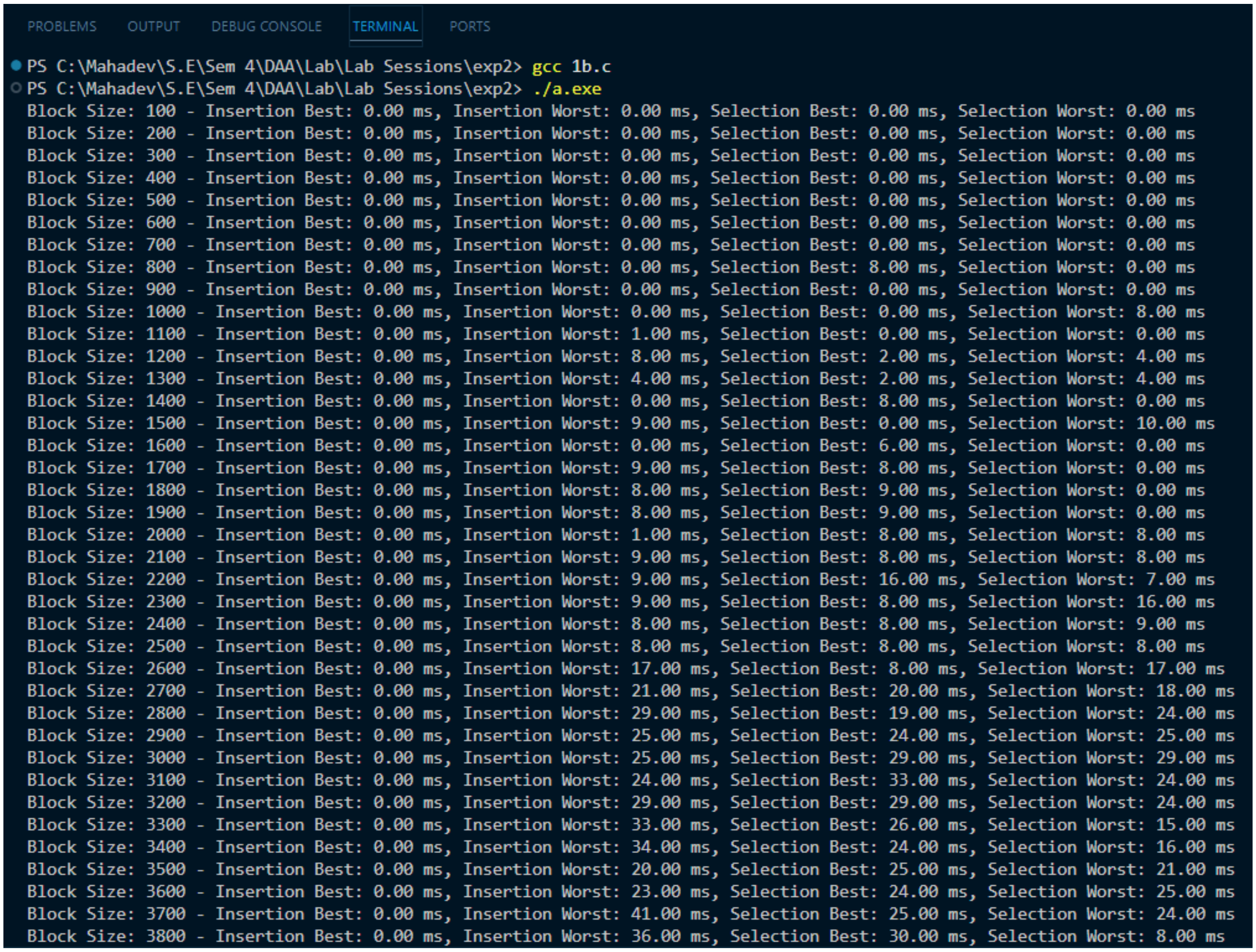
```
void readNumbers(int *numbers, int count) {
    FILE *inFile = fopen(DATA_FILE, "r");
    if (inFile == NULL) {
        printf("Error opening file!\n");
        return;
    }
    for (int i = 0; i < count; i++) {
        fscanf(inFile, "%d", &numbers[i]);
    }
    fclose(inFile);
}

void insertionSort(int *arr, int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        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;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}
```

```
void generateBestCaseInsertion(int *arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = i;
    }
}

void generateWorstCaseInsertion(int *arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = n - i;
    }
}

void generateWorstCaseSelection(int *arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = n - i;
    }
}

void measureSortingTime() {
    FILE *timeFile = fopen(TIME_FILE, "w");
    if (timeFile == NULL) {
        printf("Error opening file!\n");
        return;
    }
    fprintf(timeFile, "Block Size,Insertion Sort Best Case (ms),Insertion Sort Worst Case (ms),Selection Sort Best Case (ms),Selection Sort Worst Case (ms)\n");

    int *numbers = (int *)malloc(NUM_COUNT * sizeof(int));
    if (numbers == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    readNumbers(numbers, NUM_COUNT);

    for (int blockSize = 100; blockSize <= NUM_COUNT; blockSize += 100) {
```

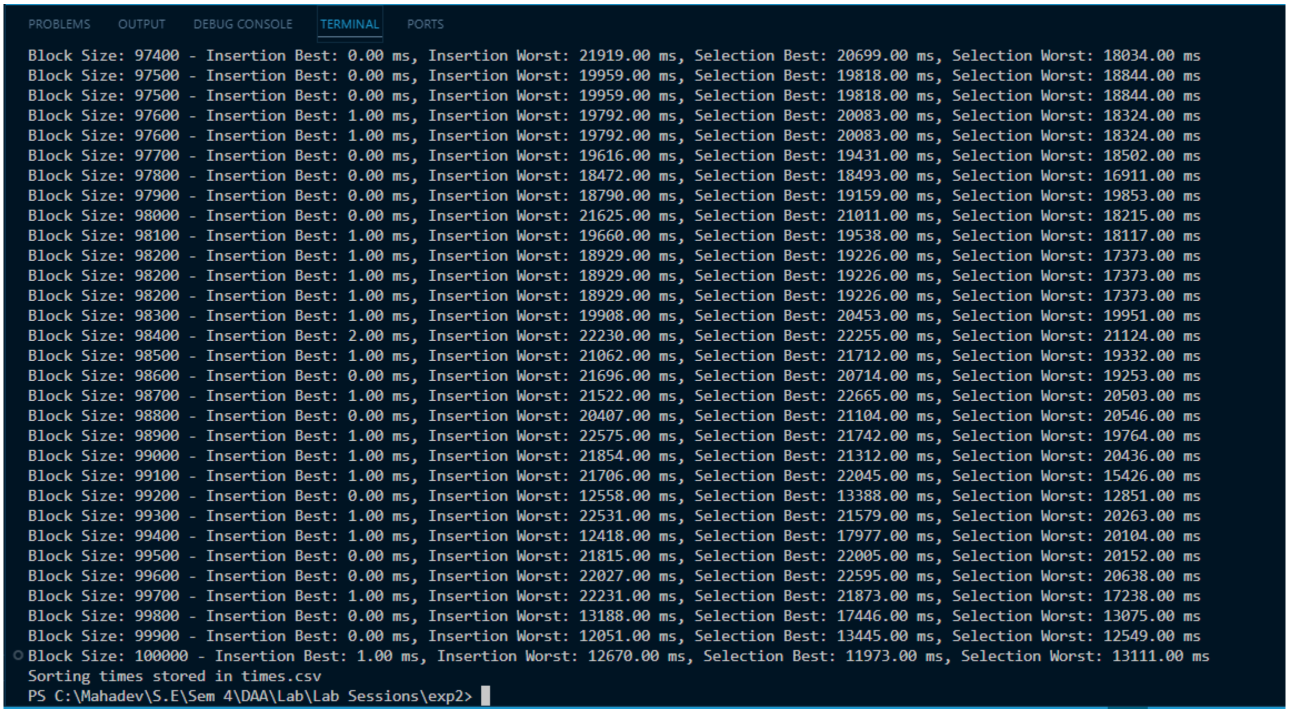
```
    int *tempInsertionBest = (int *)malloc(blockSize * sizeof(int));
    int *tempInsertionWorst = (int *)malloc(blockSize * sizeof(int));
    int *tempSelectionBest = (int *)malloc(blockSize * sizeof(int));
    int *tempSelectionWorst = (int *)malloc(blockSize * sizeof(int));
    if (tempInsertionBest == NULL || tempInsertionWorst == NULL ||
tempSelectionBest == NULL || tempSelectionWorst == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    generateBestCaseInsertion(tempInsertionBest, blockSize);
    generateWorstCaseInsertion(tempInsertionWorst, blockSize);
    generateBestCaseInsertion(tempSelectionBest, blockSize);
    generateWorstCaseSelection(tempSelectionWorst, blockSize);

    clock_t start = clock();
    insertionSort(tempInsertionBest, blockSize);
    clock_t stop = clock();
    double insertionBestTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC;

    start = clock();
    insertionSort(tempInsertionWorst, blockSize);
    stop = clock();
    double insertionWorstTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC;

    start = clock();
    selectionSort(tempSelectionBest, blockSize);
    stop = clock();
    double selectionBestTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC;

    start = clock();
    selectionSort(tempSelectionWorst, blockSize);
    stop = clock();
    double selectionWorstTime = (double)(stop - start) * 1000 /
CLOCKS_PER_SEC;
```

```
        fprintf(timeFile, "%d,%.2f,%.2f,%.2f,%.2f\n", blockSize, insertionBestTime, insertionWorstTime, selectionBestTime, selectionWorstTime);
        printf("Block Size: %d - Insertion Best: %.2f ms, Insertion Worst: %.2f ms, Selection Best: %.2f ms, Selection Worst: %.2f ms\n", blockSize, insertionBestTime, insertionWorstTime, selectionBestTime, selectionWorstTime);

        free(tempInsertionBest);
        free(tempInsertionWorst);
        free(tempSelectionBest);
        free(tempSelectionWorst);
    }

    fclose(timeFile);
    free(numbers);
}

int main() {
    generateRandomNumbers();
    measureSortingTime();
    printf("Sorting times stored in %s\n", TIME_FILE);
    return 0;
}
```

RESULT:

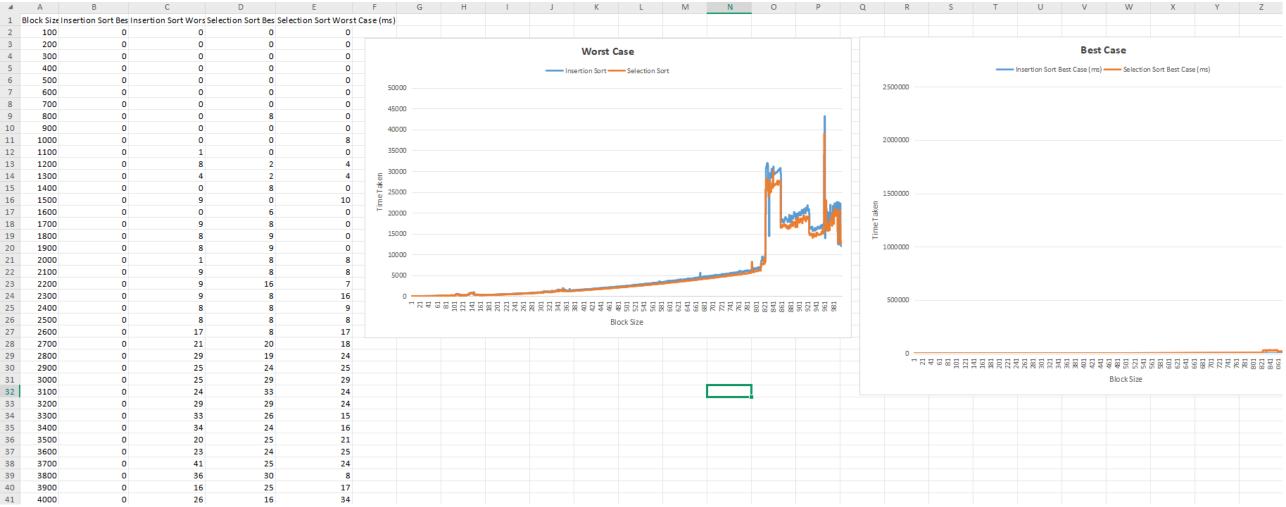


```
● PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> gcc 1b.c
○ PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> ./a.exe
  Block Size: 100 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 200 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 300 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 400 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 500 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 600 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 700 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 800 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 8.00 ms, Selection Worst: 0.00 ms
  Block Size: 900 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 1000 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 0.00 ms, Selection Worst: 8.00 ms
  Block Size: 1100 - Insertion Best: 0.00 ms, Insertion Worst: 1.00 ms, Selection Best: 0.00 ms, Selection Worst: 0.00 ms
  Block Size: 1200 - Insertion Best: 0.00 ms, Insertion Worst: 8.00 ms, Selection Best: 2.00 ms, Selection Worst: 4.00 ms
  Block Size: 1300 - Insertion Best: 0.00 ms, Insertion Worst: 4.00 ms, Selection Best: 2.00 ms, Selection Worst: 4.00 ms
  Block Size: 1400 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 8.00 ms, Selection Worst: 0.00 ms
  Block Size: 1500 - Insertion Best: 0.00 ms, Insertion Worst: 9.00 ms, Selection Best: 0.00 ms, Selection Worst: 10.00 ms
  Block Size: 1600 - Insertion Best: 0.00 ms, Insertion Worst: 0.00 ms, Selection Best: 6.00 ms, Selection Worst: 0.00 ms
  Block Size: 1700 - Insertion Best: 0.00 ms, Insertion Worst: 9.00 ms, Selection Best: 8.00 ms, Selection Worst: 0.00 ms
  Block Size: 1800 - Insertion Best: 0.00 ms, Insertion Worst: 8.00 ms, Selection Best: 9.00 ms, Selection Worst: 0.00 ms
  Block Size: 1900 - Insertion Best: 0.00 ms, Insertion Worst: 8.00 ms, Selection Best: 9.00 ms, Selection Worst: 0.00 ms
  Block Size: 2000 - Insertion Best: 0.00 ms, Insertion Worst: 1.00 ms, Selection Best: 8.00 ms, Selection Worst: 8.00 ms
  Block Size: 2100 - Insertion Best: 0.00 ms, Insertion Worst: 9.00 ms, Selection Best: 8.00 ms, Selection Worst: 8.00 ms
  Block Size: 2200 - Insertion Best: 0.00 ms, Insertion Worst: 9.00 ms, Selection Best: 16.00 ms, Selection Worst: 7.00 ms
  Block Size: 2300 - Insertion Best: 0.00 ms, Insertion Worst: 9.00 ms, Selection Best: 8.00 ms, Selection Worst: 16.00 ms
  Block Size: 2400 - Insertion Best: 0.00 ms, Insertion Worst: 8.00 ms, Selection Best: 8.00 ms, Selection Worst: 9.00 ms
  Block Size: 2500 - Insertion Best: 0.00 ms, Insertion Worst: 8.00 ms, Selection Best: 8.00 ms, Selection Worst: 8.00 ms
  Block Size: 2600 - Insertion Best: 0.00 ms, Insertion Worst: 17.00 ms, Selection Best: 8.00 ms, Selection Worst: 17.00 ms
  Block Size: 2700 - Insertion Best: 0.00 ms, Insertion Worst: 21.00 ms, Selection Best: 20.00 ms, Selection Worst: 18.00 ms
  Block Size: 2800 - Insertion Best: 0.00 ms, Insertion Worst: 29.00 ms, Selection Best: 19.00 ms, Selection Worst: 24.00 ms
  Block Size: 2900 - Insertion Best: 0.00 ms, Insertion Worst: 25.00 ms, Selection Best: 24.00 ms, Selection Worst: 25.00 ms
  Block Size: 3000 - Insertion Best: 0.00 ms, Insertion Worst: 25.00 ms, Selection Best: 29.00 ms, Selection Worst: 29.00 ms
  Block Size: 3100 - Insertion Best: 0.00 ms, Insertion Worst: 24.00 ms, Selection Best: 33.00 ms, Selection Worst: 24.00 ms
  Block Size: 3200 - Insertion Best: 0.00 ms, Insertion Worst: 29.00 ms, Selection Best: 29.00 ms, Selection Worst: 24.00 ms
  Block Size: 3300 - Insertion Best: 0.00 ms, Insertion Worst: 33.00 ms, Selection Best: 26.00 ms, Selection Worst: 15.00 ms
  Block Size: 3400 - Insertion Best: 0.00 ms, Insertion Worst: 34.00 ms, Selection Best: 24.00 ms, Selection Worst: 16.00 ms
  Block Size: 3500 - Insertion Best: 0.00 ms, Insertion Worst: 20.00 ms, Selection Best: 25.00 ms, Selection Worst: 21.00 ms
  Block Size: 3600 - Insertion Best: 0.00 ms, Insertion Worst: 23.00 ms, Selection Best: 24.00 ms, Selection Worst: 25.00 ms
  Block Size: 3700 - Insertion Best: 0.00 ms, Insertion Worst: 41.00 ms, Selection Best: 25.00 ms, Selection Worst: 24.00 ms
  Block Size: 3800 - Insertion Best: 0.00 ms, Insertion Worst: 36.00 ms, Selection Best: 30.00 ms, Selection Worst: 8.00 ms
```



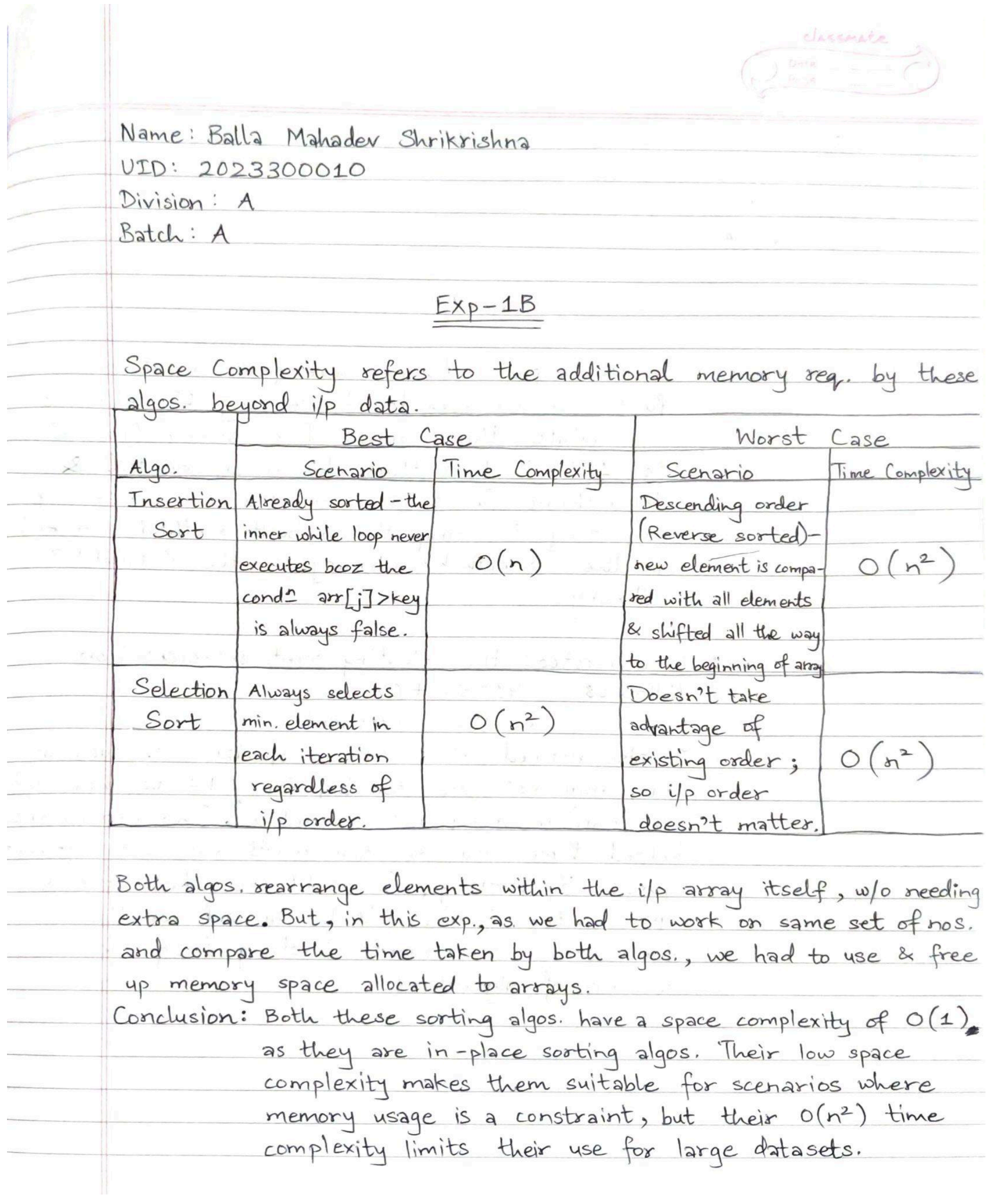
```
Block Size: 97400 - Insertion Best: 0.00 ms, Insertion Worst: 21919.00 ms, Selection Best: 20699.00 ms, Selection Worst: 18034.00 ms
Block Size: 97500 - Insertion Best: 0.00 ms, Insertion Worst: 19959.00 ms, Selection Best: 19818.00 ms, Selection Worst: 18844.00 ms
Block Size: 97500 - Insertion Best: 0.00 ms, Insertion Worst: 19959.00 ms, Selection Best: 19818.00 ms, Selection Worst: 18844.00 ms
Block Size: 97600 - Insertion Best: 1.00 ms, Insertion Worst: 19792.00 ms, Selection Best: 20083.00 ms, Selection Worst: 18324.00 ms
Block Size: 97600 - Insertion Best: 1.00 ms, Insertion Worst: 19792.00 ms, Selection Best: 20083.00 ms, Selection Worst: 18324.00 ms
Block Size: 97700 - Insertion Best: 0.00 ms, Insertion Worst: 19616.00 ms, Selection Best: 19431.00 ms, Selection Worst: 18502.00 ms
Block Size: 97800 - Insertion Best: 0.00 ms, Insertion Worst: 18472.00 ms, Selection Best: 18493.00 ms, Selection Worst: 16911.00 ms
Block Size: 97900 - Insertion Best: 0.00 ms, Insertion Worst: 18790.00 ms, Selection Best: 19159.00 ms, Selection Worst: 19853.00 ms
Block Size: 98000 - Insertion Best: 0.00 ms, Insertion Worst: 21625.00 ms, Selection Best: 21011.00 ms, Selection Worst: 18215.00 ms
Block Size: 98100 - Insertion Best: 1.00 ms, Insertion Worst: 19660.00 ms, Selection Best: 19538.00 ms, Selection Worst: 18117.00 ms
Block Size: 98200 - Insertion Best: 1.00 ms, Insertion Worst: 18929.00 ms, Selection Best: 19226.00 ms, Selection Worst: 17373.00 ms
Block Size: 98200 - Insertion Best: 1.00 ms, Insertion Worst: 18929.00 ms, Selection Best: 19226.00 ms, Selection Worst: 17373.00 ms
Block Size: 98200 - Insertion Best: 1.00 ms, Insertion Worst: 18929.00 ms, Selection Best: 19226.00 ms, Selection Worst: 17373.00 ms
Block Size: 98300 - Insertion Best: 1.00 ms, Insertion Worst: 19908.00 ms, Selection Best: 20453.00 ms, Selection Worst: 19951.00 ms
Block Size: 98400 - Insertion Best: 2.00 ms, Insertion Worst: 22230.00 ms, Selection Best: 22255.00 ms, Selection Worst: 21124.00 ms
Block Size: 98500 - Insertion Best: 1.00 ms, Insertion Worst: 21062.00 ms, Selection Best: 21712.00 ms, Selection Worst: 19332.00 ms
Block Size: 98600 - Insertion Best: 0.00 ms, Insertion Worst: 21696.00 ms, Selection Best: 20714.00 ms, Selection Worst: 19253.00 ms
Block Size: 98700 - Insertion Best: 1.00 ms, Insertion Worst: 21522.00 ms, Selection Best: 22665.00 ms, Selection Worst: 20503.00 ms
Block Size: 98800 - Insertion Best: 0.00 ms, Insertion Worst: 20407.00 ms, Selection Best: 21104.00 ms, Selection Worst: 20546.00 ms
Block Size: 98900 - Insertion Best: 1.00 ms, Insertion Worst: 22575.00 ms, Selection Best: 21742.00 ms, Selection Worst: 19764.00 ms
Block Size: 99000 - Insertion Best: 1.00 ms, Insertion Worst: 21854.00 ms, Selection Best: 21312.00 ms, Selection Worst: 20436.00 ms
Block Size: 99100 - Insertion Best: 1.00 ms, Insertion Worst: 21706.00 ms, Selection Best: 22045.00 ms, Selection Worst: 15426.00 ms
Block Size: 99200 - Insertion Best: 0.00 ms, Insertion Worst: 12558.00 ms, Selection Best: 13388.00 ms, Selection Worst: 12851.00 ms
Block Size: 99300 - Insertion Best: 1.00 ms, Insertion Worst: 22531.00 ms, Selection Best: 21579.00 ms, Selection Worst: 20263.00 ms
Block Size: 99400 - Insertion Best: 1.00 ms, Insertion Worst: 12418.00 ms, Selection Best: 17977.00 ms, Selection Worst: 20104.00 ms
Block Size: 99500 - Insertion Best: 0.00 ms, Insertion Worst: 21815.00 ms, Selection Best: 22005.00 ms, Selection Worst: 20152.00 ms
Block Size: 99600 - Insertion Best: 0.00 ms, Insertion Worst: 22027.00 ms, Selection Best: 22595.00 ms, Selection Worst: 20638.00 ms
Block Size: 99700 - Insertion Best: 1.00 ms, Insertion Worst: 22231.00 ms, Selection Best: 21873.00 ms, Selection Worst: 17238.00 ms
Block Size: 99800 - Insertion Best: 0.00 ms, Insertion Worst: 13188.00 ms, Selection Best: 17446.00 ms, Selection Worst: 13075.00 ms
Block Size: 99900 - Insertion Best: 0.00 ms, Insertion Worst: 12051.00 ms, Selection Best: 13445.00 ms, Selection Worst: 12549.00 ms
○ Block Size: 100000 - Insertion Best: 1.00 ms, Insertion Worst: 12670.00 ms, Selection Best: 11973.00 ms, Selection Worst: 13111.00 ms
Sorting times stored in times.csv
PS C:\Mahadev\S.E\Sem 4\DAA\Lab\Lab Sessions\exp2> 
```

EXCEL OUTPUT :



	A	B	C	D	E
1	Block Size	Insertion Sort Bes	Insertion Sort Wors	Selection Sort Bes	Selection Sort Worst Case (ms)
2	100	0	0	0	0
3	200	0	0	0	0
4	300	0	0	0	0
5	400	0	0	0	0
6	500	0	0	0	0
7	600	0	0	0	0
8	700	0	0	0	0
9	800	0	0	8	0
10	900	0	0	0	0
11	1000	0	0	0	8
12	1100	0	1	0	0
13	1200	0	8	2	4
14	1300	0	4	2	4
15	1400	0	0	8	0
16	1500	0	9	0	10
17	1600	0	0	6	0
18	1700	0	9	8	0
19	1800	0	8	9	0
20	1900	0	8	9	0
21	2000	0	1	8	8
22	2100	0	9	8	8
23	2200	0	9	16	7
24	2300	0	9	8	16
25	2400	0	8	8	9
26	2500	0	8	8	8
27	2600	0	17	8	17
28	2700	0	21	20	18
29	2800	0	29	19	24
30	2900	0	25	24	25
31	3000	0	25	29	29
32	3100	0	24	33	24
33	3200	0	29	29	24
34	3300	0	33	26	15
35	3400	0	34	24	16
36	3500	0	20	25	21
37	3600	0	23	24	25
38	3700	0	41	25	24
39	3800	0	36	30	8
40	3900	0	16	25	17
41	4000	0	26	16	34

CONCLUSION:



Name: Balla Mahadev Shrikrishna
UID: 2023300010
Division: A
Batch: A

Exp-1B

Space Complexity refers to the additional memory req. by these algos. beyond i/p data.

	Best Case		Worst Case	
Algo.	Scenario	Time Complexity	Scenario	Time Complexity
Insertion Sort	Already sorted - the inner while loop never executes bcoz the cond^n arr[j]>key is always false.	$O(n)$	Descending order (Reverse sorted)- new element is compared with all elements & shifted all the way to the beginning of array	$O(n^2)$
Selection Sort	Always selects min. element in each iteration regardless of i/p order.	$O(n^2)$	Doesn't take advantage of existing order; so i/p order doesn't matter.	$O(n^2)$

Both algos. rearrange elements within the i/p array itself, w/o needing extra space. But, in this exp., as we had to work on same set of nos. and compare the time taken by both algos., we had to use & free up memory space allocated to arrays.

Conclusion: Both these sorting algos. have a space complexity of $O(1)$, as they are in-place sorting algos. Their low space complexity makes them suitable for scenarios where memory usage is a constraint, but their $O(n^2)$ time complexity limits their use for large datasets.