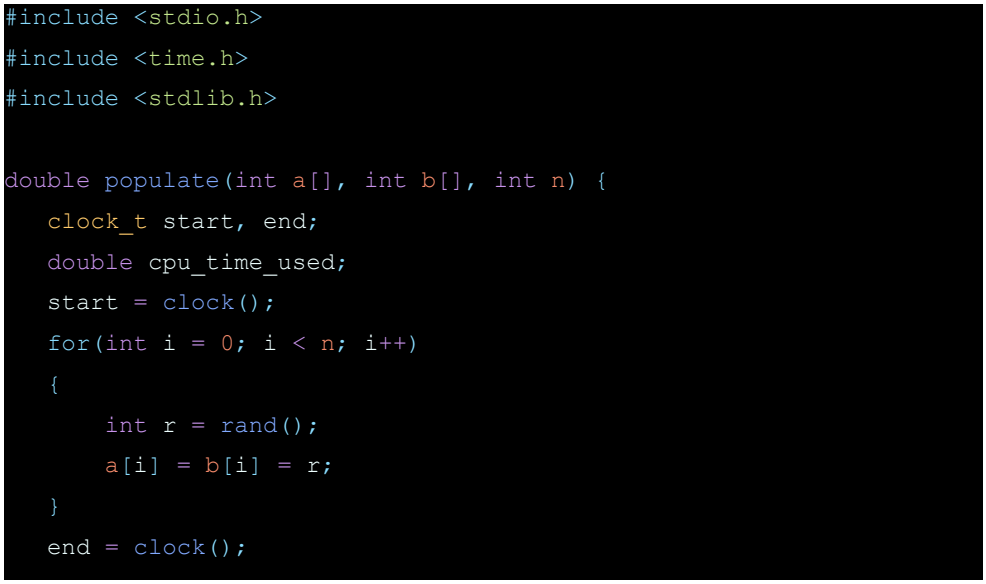
Name	Stephen David Vaz
UID no.	2021700070
Experiment No.	1

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 by 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 algorithm uses 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.
ALGORITHM/ THEORY:	**Theory:** The understanding of running time of algorithms is explored by implementing two basic sorting algorithms namely Insertion and Selection sorts. These algorithms work as follows. Insertion sort– It works similar to the sorting of playing cards in hands. It is assumed that the first card is already sorted in the card game, and then we select an unsorted card. If the selected unsorted card is greater than the first card, it will be placed at the right side; otherwise, it will be placed at the left side. Similarly, all unsorted cards are taken and put in their exact place.

	Selection sort– It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and the unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted. **Algorithm:** Insertion Sort:- 1. Start comparison of elements from index 1 of the array 2. If the element is smaller than it's previous element then we swap it 3. Keep swapping in the backwards direction until the element is no longer smaller than it's previous element or you've reached the first element Selection Sort:- 1. Assume the first element of the unsorted array to be the minimum 2. Loop through the unsorted array to find a new minimum 3. Swap the new minimum with the first element of the unsorted array 4. The new unsorted array is one index ahead of what it was previously 5. Keep repeating steps 1 to 4 until you reach the end of the unsorted array.

PROGRAM:	
	`#include <stdio.h>`
	`#include <time.h>`
	`#include <stdlib.h>`
	
	`double populate(int a[], int b[], int n) {`
	`clock_t start, end;`
	`double cpu_time_used;`
	`start = clock();`
	`for(int i = 0; i < n; i++)`
	`{`
	`int r = rand();`
	`a[i] = b[i] = r;`
	`}`
	`end = clock();`

```
    FILE *fp = fopen("./random.txt", "w+");
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    return cpu_time_used;
}



void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}



double selection(int a[], int n) {
    FILE *fp = fopen("./selection.csv", "w+");
    // printf("File opened\n");
    double totalTime = 0;
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    fprintf(fp, "n, time\n");
    for (int i = 100; i <= n; i+=100)
    {
        // printf("%d\n", i);
        clock_t start, end;
        double cpu_time_used;
        start = clock();
        for(int j = 0; j < i; j++) {
            int min = j;
            for(int k = j+1; k < i; k++) {
                if(a[k] < a[min]) {
                    min = k;
                }
```

```
            }
            swap(&a[j], &a[min]);
        }
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        totalTime += cpu_time_used;
        fprintf(fp, "%d, %f\n", i, cpu_time_used);
        printf("Sorted from 0 to %d in %.2fs\n", i, cpu_time_used);

        // for(int z = 0; z < i; z++) {
        //     printf("%d\n", a[z]);
        // }
        // getchar();
    }
    fclose(fp);
    fp = fopen("./selection.txt", "w+");
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    fclose(fp);
    return totalTime;
}

double insertion(int a[], int n) {
    FILE *fp = fopen("./insertion.csv", "w+");
    // printf("File opened\n");
    double totalTime = 0;
    if(!fp) {
        printf("Error opening file\n");
        return -1;
    }
    fprintf(fp, "n, time\n");
    //insertion sort
    //first sort from 0 to 100 the 0 to 200 and so on upto n
    for (int i = 100; i <= n; i+=100)
    {
        // printf("%d\n", i);
        clock_t start, end;
        double cpu_time_used;
        start = clock();
        for(int j = 1; j < i; j++)
```

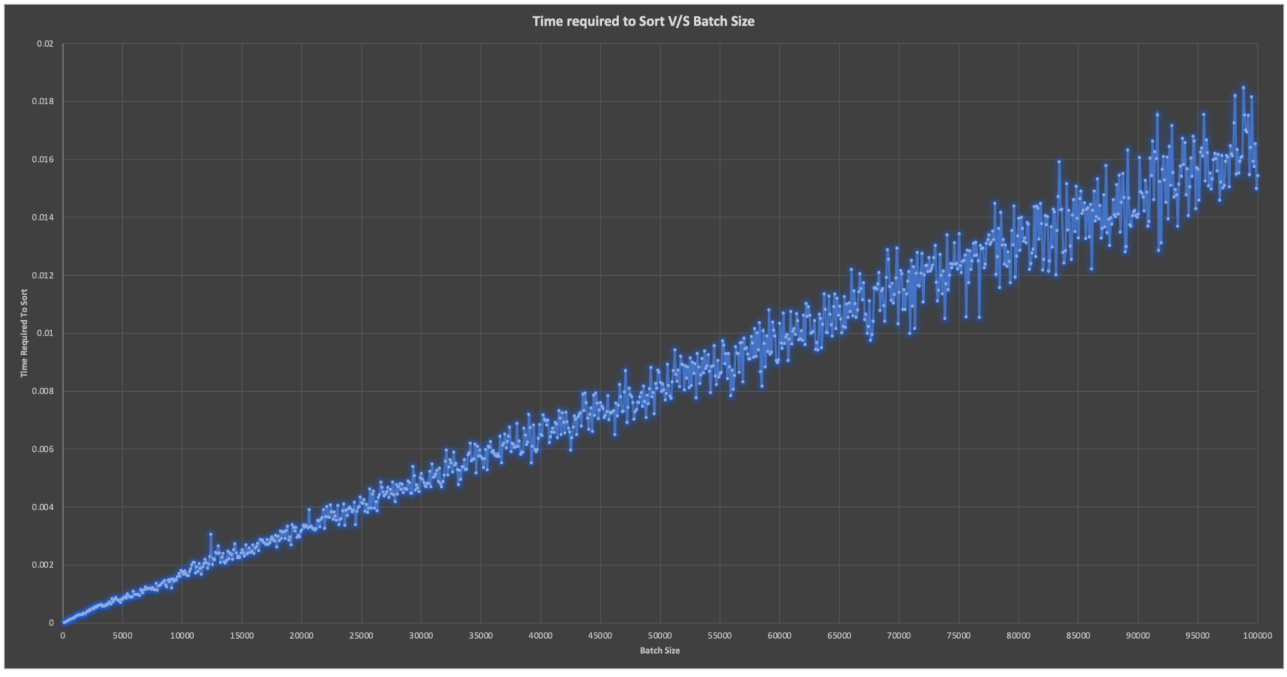
```
        {
            int k = j;
            // printf("%d\n", i);
            while(k > 0 && a[k] < a[k-1])
            {
                swap(&a[k], &a[k-1]);
                k--;
            }
        }
        end = clock();
        cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
        totalTime += cpu_time_used;
        fprintf(fp, "%d, %f\n", i, cpu_time_used);

        printf("Sorted from 0 to %d in %.2fs\n", i, cpu_time_used);

        // for(int z = 0; z < i; z++) {
        //     printf("%d\n", a[z]);
        // }
        // getchar();
    }
    fclose(fp);
    fp = fopen("./insertion.txt", "w+");
    for(int i = 0; i < n; i++) {
        fprintf(fp, "%d\n", a[i]);
    }
    fclose(fp);
    return totalTime;
}

int main()
{
    int n = 100000;
    int a[n],b[n];
    double timeToPopulate = populate(a, b, n);
    printf("Time taken to populate: %f\nSorting...\n",
timeToPopulate);
    //first sort from 0 to 100 the 0 to 200 and so on upto n
    double timeToSortI = insertion(a, n);
    double timeToSortS = selection(b, n);
    printf("Array sorted by insertion sort in %.2f\n", timeToSortI);
```

```
    printf("Array sorted by selection sort in %.2f\n", timeToSortS);
    printf("Total time taken to sort: %f\n", timeToSortI +
timeToSortS);
    return 0;
}
```

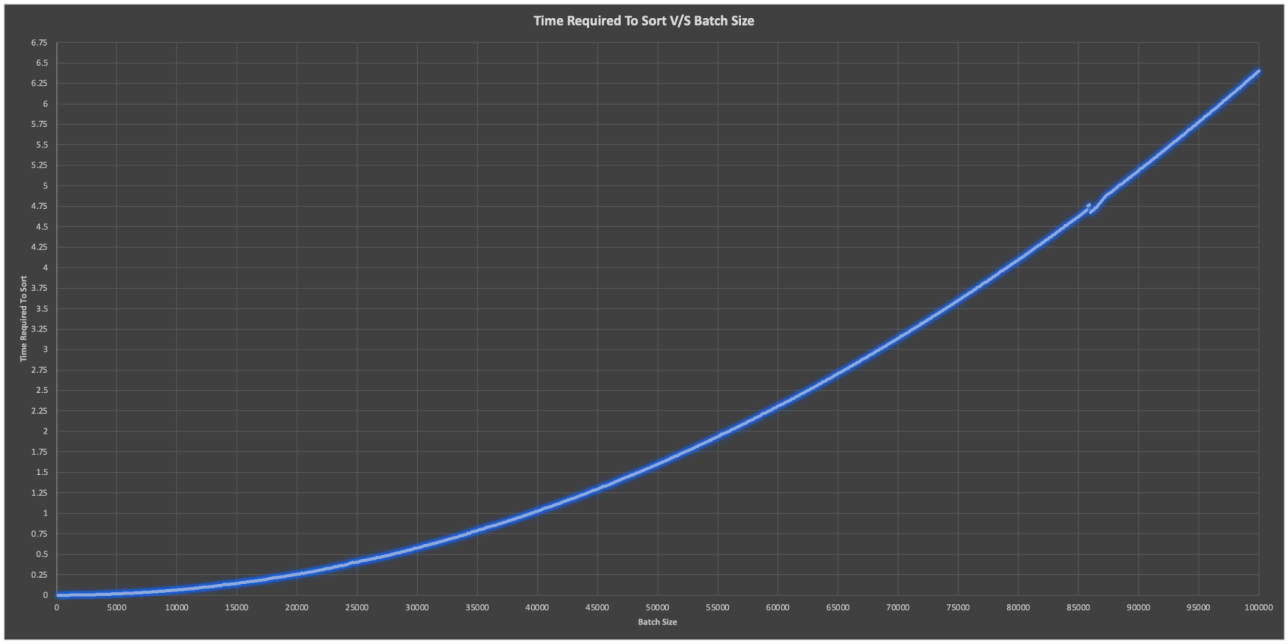
RESULT:

```
Sorted from 0 to 99200 in 6.29s
Sorted from 0 to 99300 in 6.30s
Sorted from 0 to 99400 in 6.32s
Sorted from 0 to 99500 in 6.33s
Sorted from 0 to 99600 in 6.34s
Sorted from 0 to 99700 in 6.36s
Sorted from 0 to 99800 in 6.37s
Sorted from 0 to 99900 in 6.38s
Sorted from 0 to 100000 in 6.39s
Array sorted by insertion sort in 8.24
Array sorted by selection sort in 2138.56
Total time taken to sort: 2146.801899
 *  Terminal will be reused by tasks, press any key to close it.
```

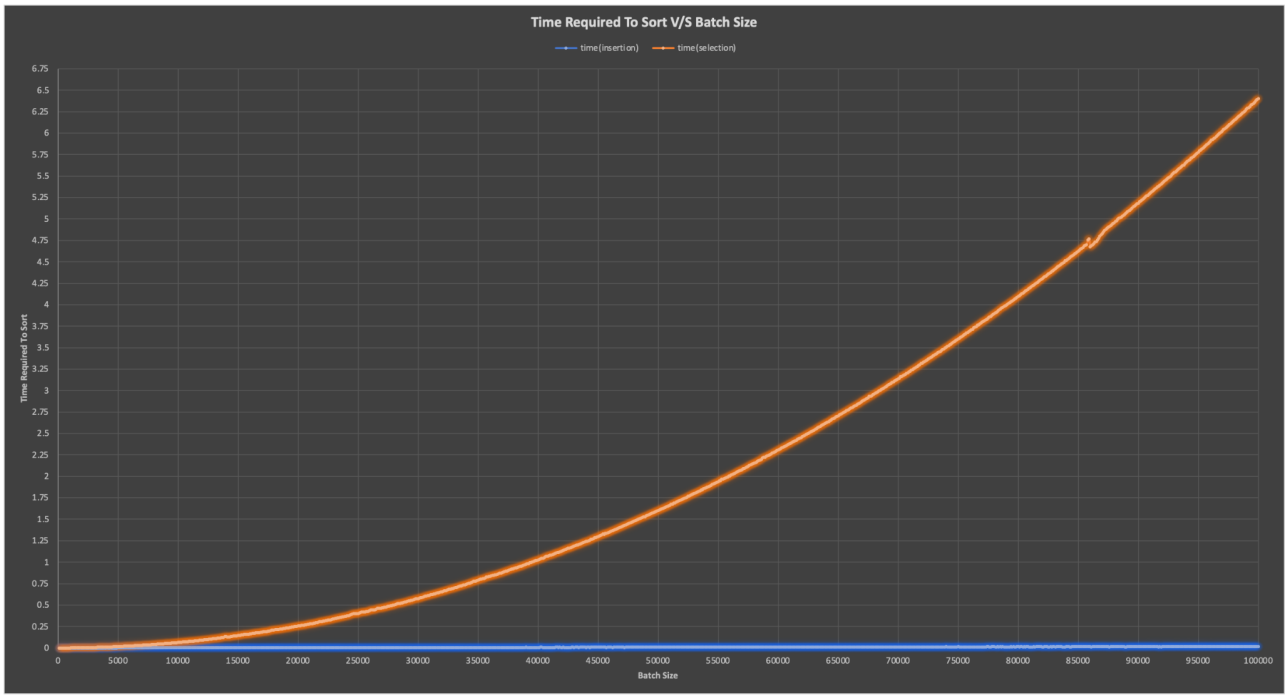
Truncated Terminal Output



Insertion Sort



Selection Sort



Comparison of insertion sort and selection sort

CONCLUSION:	Successfully understood calculation of running time of algorithms by implementing insertion and selection sort in C. Also observed the two sorting methods for 1,00,000 randomly generated numbers graphically.