DIGITAL ANALYSIS AND ALGORITHM

EXPERIMENT - 01

NAME :- MANTHAN AYALWAR

UID :- 2021700003

BATCH :- D1

PART – 1B

Aim :- Experiment on finding the running time of an algorithm.

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 to generate1,00,000 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.

Theory :-

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 unsorted part are picked and placed at the correct position in the sorted part.

Algorithm:
To sort an array of size n in ascending order:

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

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

- The subarray is already sorted.
- The remaining subarray is unsorted.

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

Code :-

```c
#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("./random10.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;
}
```

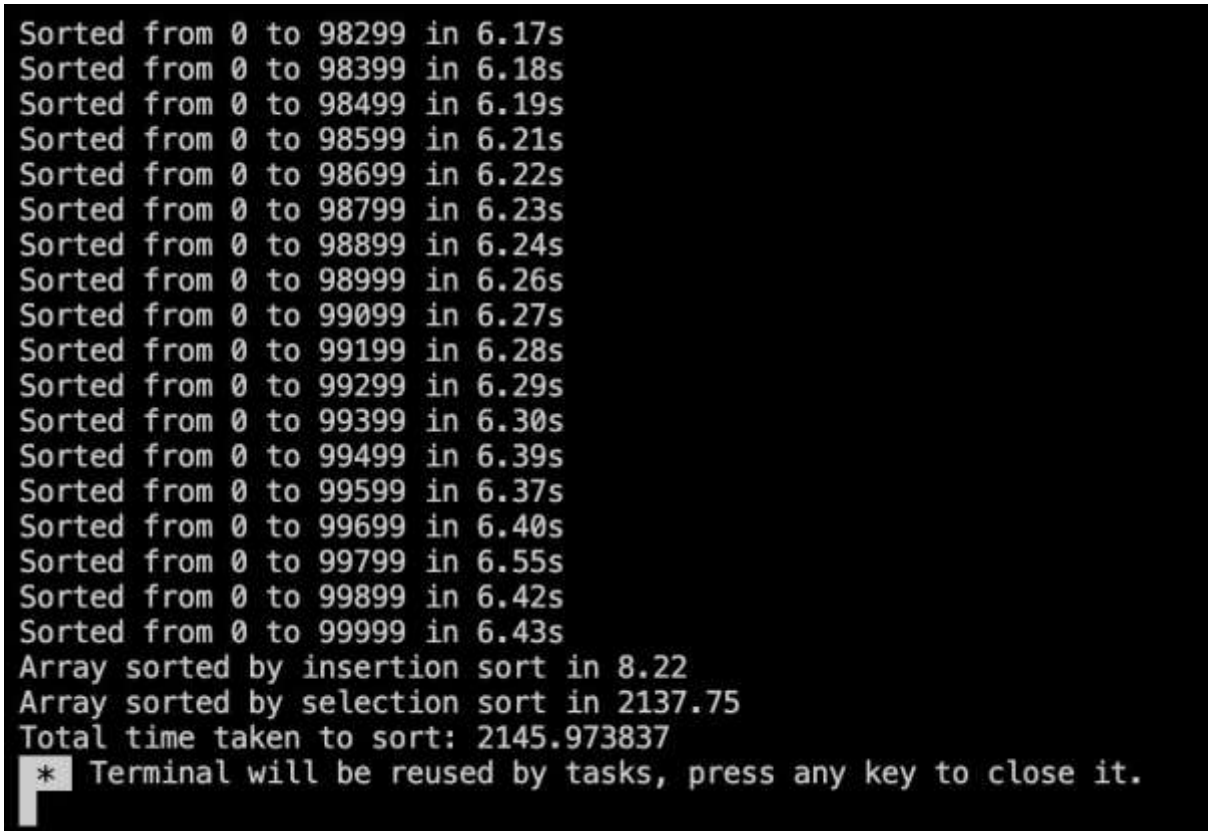
```c
void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}


double selection(int a[], int n) {
    FILE *fp = fopen("./selection10.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 = 9999; i <= n; i+=10000)
    {
        // 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+1, 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("./selection10.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("./insertion10.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 = 9999; i <= n; i+=10000)
    {
        // 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+1, 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("./insertion10.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;
}
```

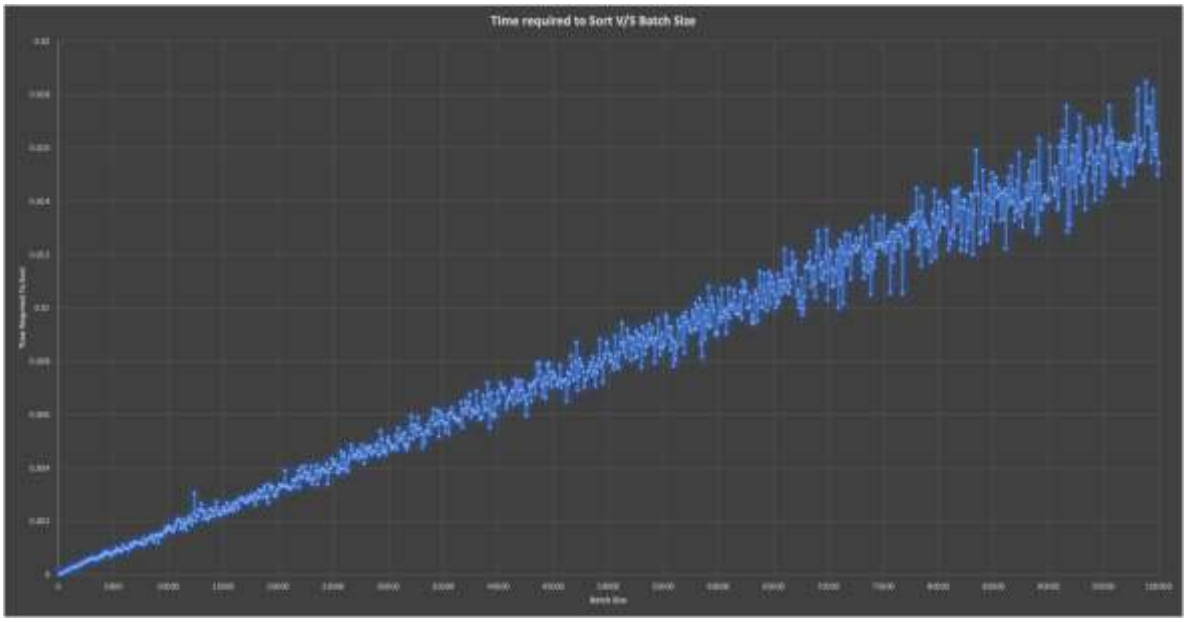
Input :-

Ouput :-

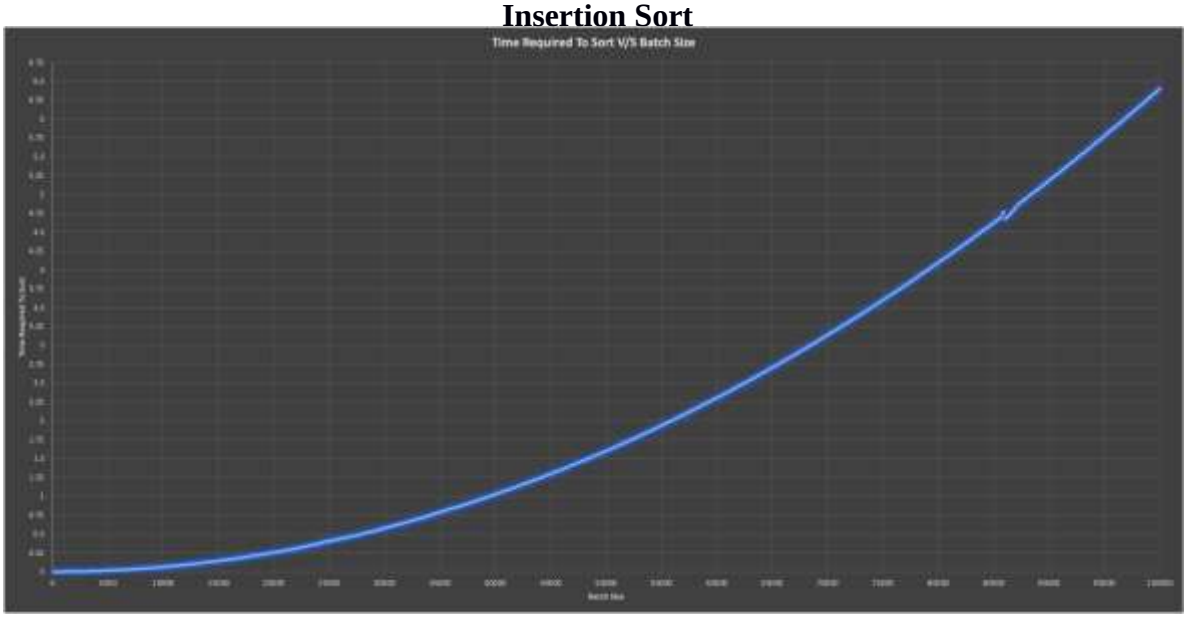

Sorted from 0 to 98299 in 6.17s
Sorted from 0 to 98399 in 6.18s
Sorted from 0 to 98499 in 6.19s
Sorted from 0 to 98599 in 6.21s
Sorted from 0 to 98699 in 6.22s
Sorted from 0 to 98799 in 6.23s
Sorted from 0 to 98899 in 6.24s
Sorted from 0 to 98999 in 6.26s
Sorted from 0 to 99099 in 6.27s
Sorted from 0 to 99199 in 6.28s
Sorted from 0 to 99299 in 6.29s
Sorted from 0 to 99399 in 6.30s
Sorted from 0 to 99499 in 6.39s
Sorted from 0 to 99599 in 6.37s
Sorted from 0 to 99699 in 6.40s
Sorted from 0 to 99799 in 6.55s
Sorted from 0 to 99899 in 6.42s
Sorted from 0 to 99999 in 6.43s
Array sorted by insertion sort in 8.22
Array sorted by selection sort in 2137.75
Total time taken to sort: 2145.973837
* Terminal will be reused by tasks, press any key to close it.

……..

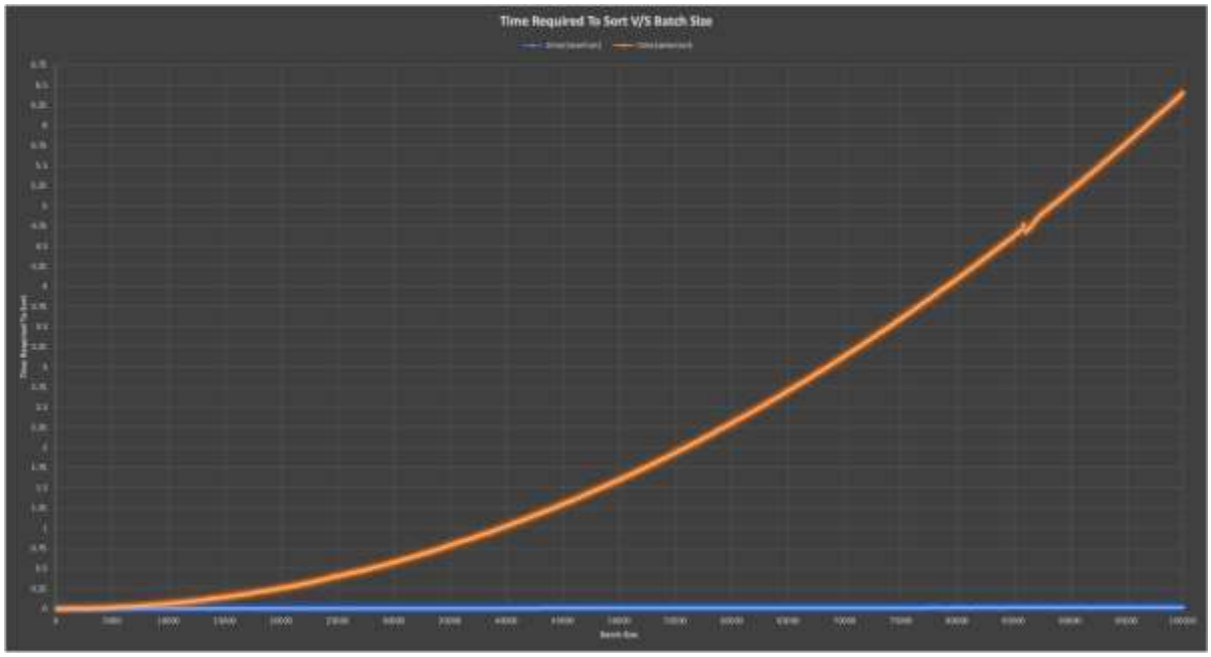
Graph :-



Insertion Sort



Selection Sort



Obsrvation :-

Factors affecting run time of sorting algorithm:-

1. The order of numbers to be sorted plays an important role in the running time.
2. CPU Utilization - If CPU is already utilized by some other processes then the running time of the algorithm will increase.
3. Recursion can cause a lot of overhead, if the structure of function permits, then use *tail recursion*, to avoid the stack overhead.
4. Choice of Language- C is faster when compared to Java because it is a low level language. Thus many network security applications are written in C, where speed matters.

Main memory usage- Higher main memory usage can lead to slower processing.

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. The array of random numbers used for sorting was the same for both sorting methods yet insertion sort on the same computer takes much less time when compared to selection sort. This can be confirmed from the graph as well as the run time observed