Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

Name	Tejas Jadhav
UID No.	2022301006
Class	COMPS A (B batch)
Experiment No.	01 B

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

Algorithm:

```
InsertionSort(A)
    For i = 2 to A.length
        key = A[i]
        j = i -1
        while j > 0 and A[j] > key
            A[j + 1] = A[j]
            j  = j – 1
        A[j + 1] = key
```

```
SelectionSort(A)
    For i = 1 to A.length – 1
        i_min = i
        for j = i +1 to A.length
            if (A[j] < A[i_min])
                i_min = j
        if i != i_min
            swap(A[i], A[i_min])
```

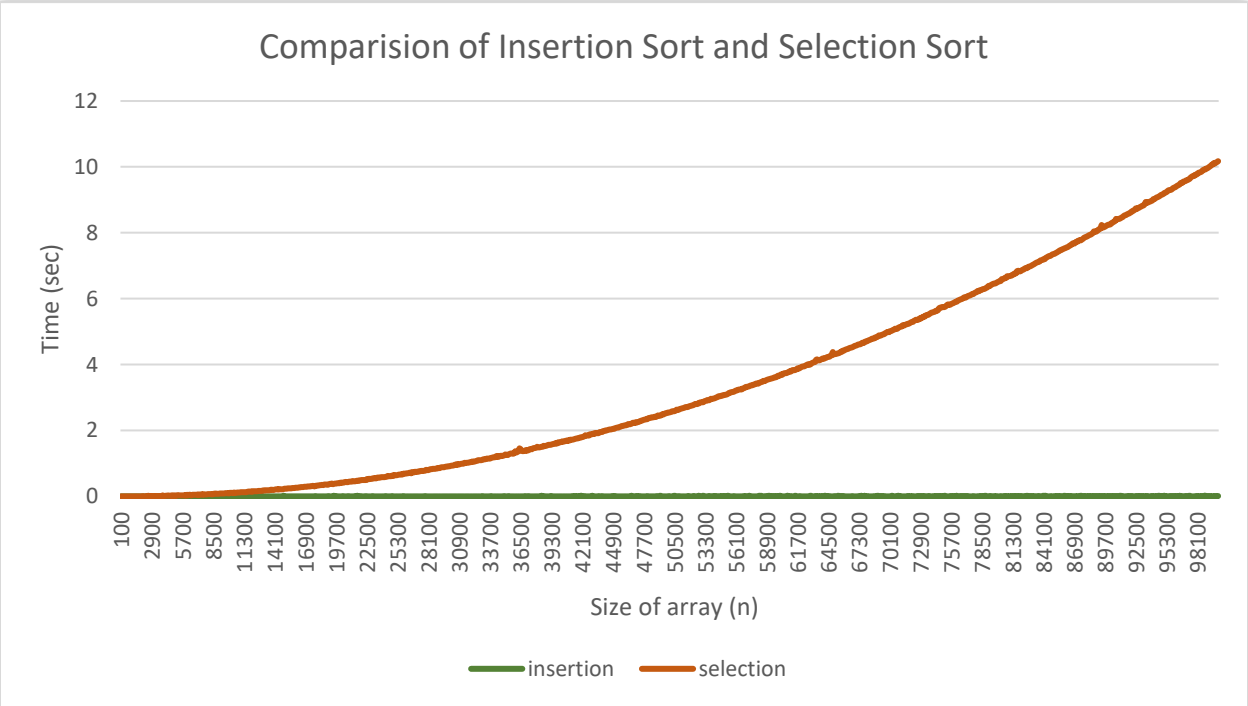
Code:

```
#include <chrono>
#include <fstream>
#include <iostream>

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

void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

void selection_sort(int* arr, int size) {
    for (int i = 0; i < size - 1; i++) {
        int i_min = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[i_min]) {
                i_min = j;
            }
        }
        if (i != i_min) {
            swap(arr[i], arr[i_min]);
        }
    }
}

int main() {
    int* arr_ins = new int[100000];
    int* arr_sel = new int[100000];
```

```
    std::ifstream nums("random_numbers.txt");
    std::ofstream output("../csv/sort_analysis.csv");
    output << "block_size,insertion,selection\n";

    for (int i = 1; i <= 100000; i++) {
        int val = 0;
        nums >> val;
        arr_ins[i] = val;
        arr_sel[i] = val;
    }

    // 1000 blocks of 100 numbers
    for (int i = 1; i <= 1000; i++) {
        auto ins_start = std::chrono::high_resolution_clock::now();
        insertion_sort(arr_ins, i * 100);
        auto ins_end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> ins_time = (ins_end - ins_start);

        auto sel_start = std::chrono::high_resolution_clock::now();
        selection_sort(arr_sel, i * 100);
        auto sel_end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> sel_time = (sel_end - sel_start);

        // std::cout << "\nBlock : " << i * 100 << "\n";
        // std::cout << "insertion sort : " << ins_time.count() << "\t"
        //           << "selection sort : " << sel_time.count() << "\n";

        output << i * 100 << "," << ins_time.count() << "," << sel_time.count()
<< "\n";
    }

    std::cout << "Sorting completed !";

    return 0;
}
```

Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Chart:



Observations:

1) Initially, both sorting techniques take the same time.
2) As the size of blocks increase, I observe that selection sort begins to take more time. However, insertion sort takes almost the same amount of time at every block
3) The graph observed for selection sort resembles a scaled down quadratic curve.
4) The graph observed for insertion sort resembles a line.
5) Insertion sort works by maintaining two subarrays, one is always sorted while other is not. This problem statement mimics the behavior of insertion sort as in the only the new 100 elements are completely random while the all the previous elements are partially sorted.
6) Selection sort works by finding the smallest element in the unsorted subarray and replacing it with current element. In order to determine if an element is truly the smallest element, it is necessary to check all elements.

Conclusion:

- Insertion sort works by maintaining two subarrays, one of which always contain sorted elements. If an element from unsorted array finds its appropriate position in sorted array, we go on to the next iteration.
- The current problem statement, that goes on adding 100 unsorted elements on already sorted array, greatly favors in how insertion sort works.
- Selection sort works by again maintaining two sorted arrays. However, in selection sort we find the smallest element in unsorted array and swap it with current element.
- In order to find the smallest element, it is necessary to traverse the entire unsorted subarray. Hence each addition of 100 elements as per problem statement, just increases the number of comparisons for selection sort.

After conducting this experiment and analyzing the time for both sorting techniques, I conclude that the nature of problem statement and initial state of data affects greatly on the runtime on algorithm even if they have the same worst case time complexity.