Batch: B2 **Roll No.:** 16010124107

Experiment No. 1

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Study, Implementation, and Performance Comparison of Selection Sort, Insertion Sort to analyze sorting strategies.

Objective: To analyse performance of sorting methods

CO to be achieved:

CO 1 Analyze the asymptotic running time and space complexity of algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algorithms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://en.wikipedia.org/wiki/Insertion_sort**
4. **http://www.sorting-algorithms.com/insertion-sort**
5. **http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html**
6. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**
7. **http://en.wikipedia.org/wiki/Selection_sort**
8. **http://www.sorting-algorithms.com/selection-sort**
9. **http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**
10. **http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.html**

Pre Lab/ Prior Concepts:

Data structures, sorting techniques.

Historical Profile:

There are various methods to sort the given list. As the size of input changes, the performance of these strategies tends to differ from each other. In such a case, the priori analysis can help the engineer to choose the best algorithm.

New Concepts to be learned:

Space complexity, time complexity, size of input, order of growth.

Topic: Sorting Algorithms

Theory: Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and- conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller sub problems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

Algorithm Insertion Sort

INSERTION_SORT (*A,n*)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR j ← 2 TO length[A]
    DO  key ← A[j]
        {Put A[j] into the sorted sequence A[1 . . j − 1]}
                i ← j − 1
                WHILE i > 0 and A[i] > key
            DO A[i +1] ← A[i]
                i ← i − 1
                A[i + 1] ← key
```

Algorithm Selection Sort

SELECTION_SORT (A,n)
//The algorithm takes as parameters an array *A*[1.. *n*] and the length *n* of the array.
//The array *A* is sorted in place: the numbers are rearranged within the array
// A[1..n] of eletype, n: integer

```
FOR i ← 1 TO n-1 DO
  min j ← i;
  min x ← A[i]
  FOR j ← i + 1 to n do
     IF A[j] < min x then
        min j ← j
        min x ← A[j]
  A[min j] ← A [i]
  A[i] ← min x
```

Code for insertion sort:

B_Hourglass.cpp ass.cpp

ass.cpp > insertionSort(int [], int)

```cpp
#include <bits/stdc++.h>
using namespace std;

void print(const int v[], int n){
    for(int i = 0; i < n; i++)
        cout << v[i] << " ";
    cout << endl;
}

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

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int a[] = {1,2,3,8,7,2,1,4,982};
    int n = sizeof(a) / sizeof(a[0]);
    print(a,n);
    insertionSort(a, n);
    print(a, n);
}
```

output:

```
PS C:\Users\syeda\OneDri
1 2 3 8 7 2 1 4 982
1 1 2 2 3 4 7 8 982
PS C:\Users\syeda\OneDri
```

code for selection sort:

B_Hourglass.cpp   ass.cpp

ass.cpp > main()

```cpp
#include <bits/stdc++.h>
using namespace std;

void print(const int v[], int n){
    for(int i = 0; i < n; i++)
        cout << v[i] << " ";
    cout << endl;
}

void selectionSort(int a[], int n){
    for(int i = 0; i < n - 1; i++){
        int minIdx = i;
        for(int j = i + 1; j < n; j++){
            if(a[j] < a[minIdx])
                minIdx = j;
        }
        swap(a[i], a[minIdx]);
    }
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int a[] = {1,2,3,3,5,1,2,3,8,2};
    int n = sizeof(a) / sizeof(a[0]);

    print(a,n);
    selectionSort(a, n);
    print(a, n);
}
```

output:

```
PS C:\Users\syeda\One
1 2 3 3 5 1 2 3 8 2
1 1 2 2 2 3 3 3 5 8
PS C:\Users\syeda\One
```

The space complexity of Insertion sort:

Insertion sort has O(1) space complexity. It sorts the array in place, therefore no extra memory is required.

The space complexity of Selection sort:

Selection sort has **O(1)** space complexity. It also sorts the array in place, therefore no extra memory is required

Time complexity for Insertion sort:

Best case (already sorted):- $\Omega(n)$

$(n-1) + (n(n-1))/2 = (n2+n+2)/2=n^2$

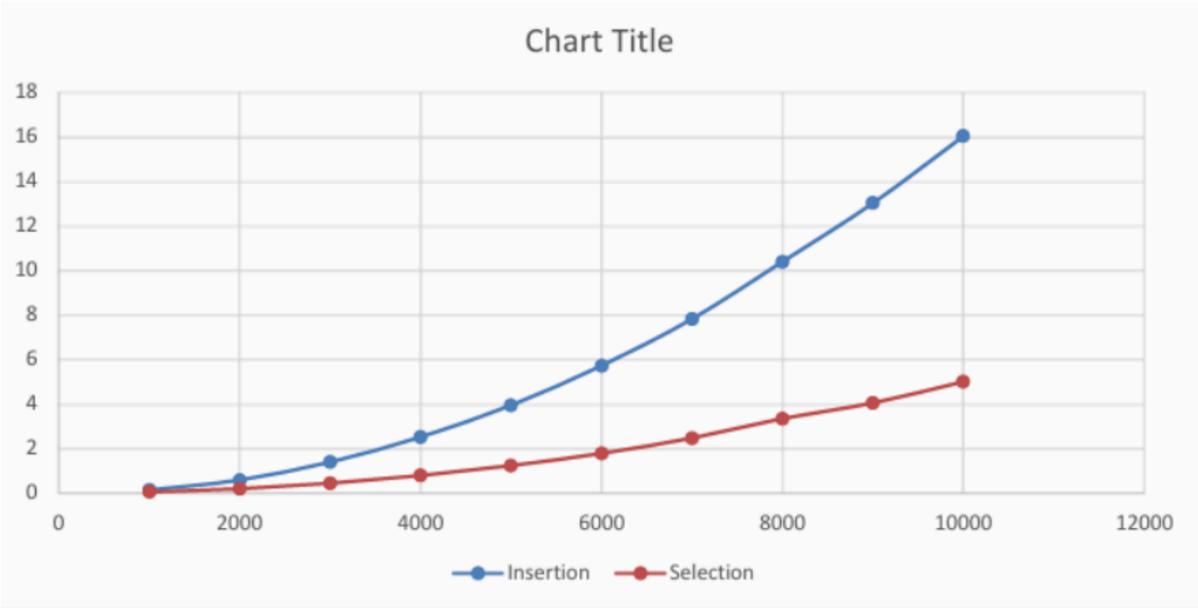
Insertion sort has $O(n^2)$ time complexity in worst case. Average case time complexity is $O(n^2)$.

Time complexity for selection sort:

$(n-1) + (n(n-1))/2 = (n2+n+2)/2$
Best case (already sorted):- $\Omega(n^2)$

Selection sort has $O(n^2)$ time complexity. It is same for best, average, and worst cases.

Graphs for varying input sizes: (Insertion Sort & Selection sort)
The graph shows how execution time changes with input size in both sorting algorithms



CONCLUSION:

Thus, the experiment verifies theoretical time complexities