K. J. Somaiya College of Engineering
(A Constituent College of Somaiya Vidyavihar University)

Department of Computer Engineering

Batch: B4	**Roll No.:** 16010122805
Experiment No.__01__	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation of selection sort/ Insertion sort

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 algortihtms",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 case, the priori analysis can helps the engineer to choose the best algorithm.

New Concepts to be learned:
Space complexity, time complexity, size of input, order of growth.

Topic: Divide and Conquer

Theory: Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, $1< k \leq n$, 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.

Control Abstraction:

```
 Type DAndC(Problem P)
{
if small (P) return S(P);
else   {
          divide P into smaller instances P1, P2, …. ,Pk, k ≥1;
          Apply  DAndC to each of these sub problems;
          Return combine(DAndC(P1), DAndC(P2),…., DAndC(Pk));
       }
}
```

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
```

The space complexity of Insertion sort: n / O(1)

The space complexity of Selection sort: n / O(1)

Time complexity for Insertion sort:

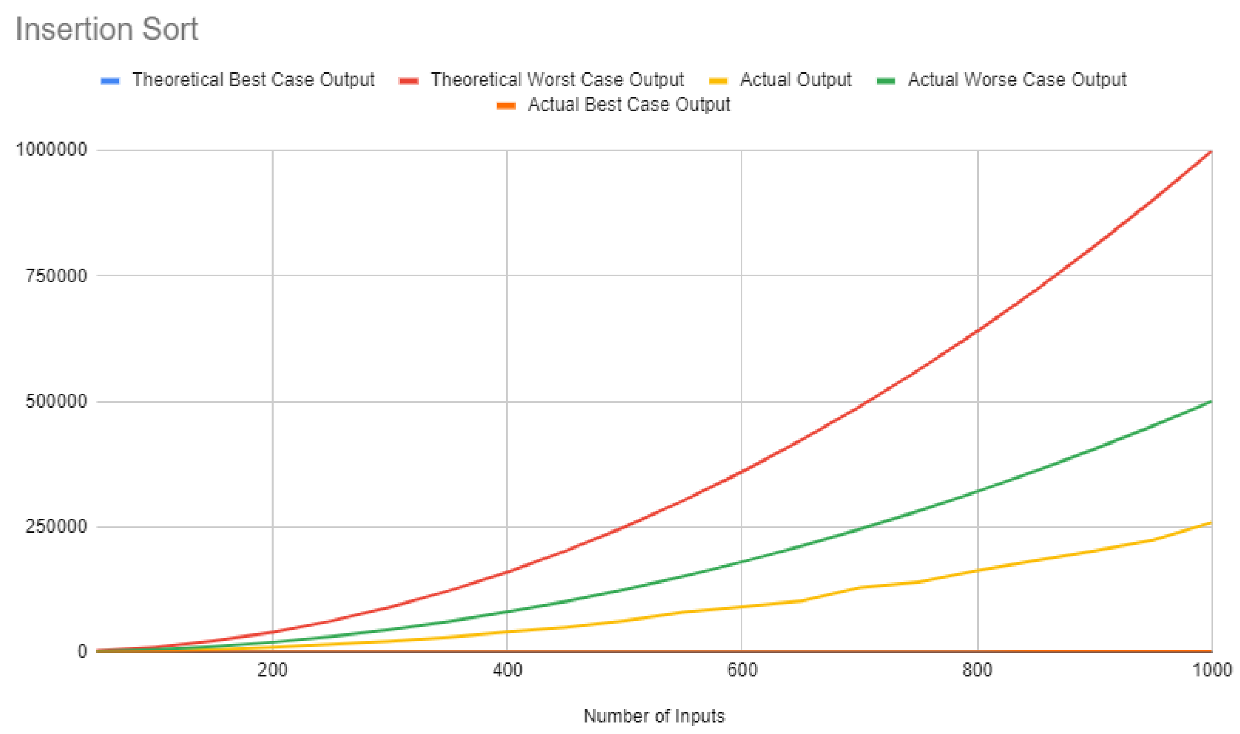
- Best Case: n / Ω(n)
- Worst Case: n^2 / O(n)

Time complexity for selection sort:

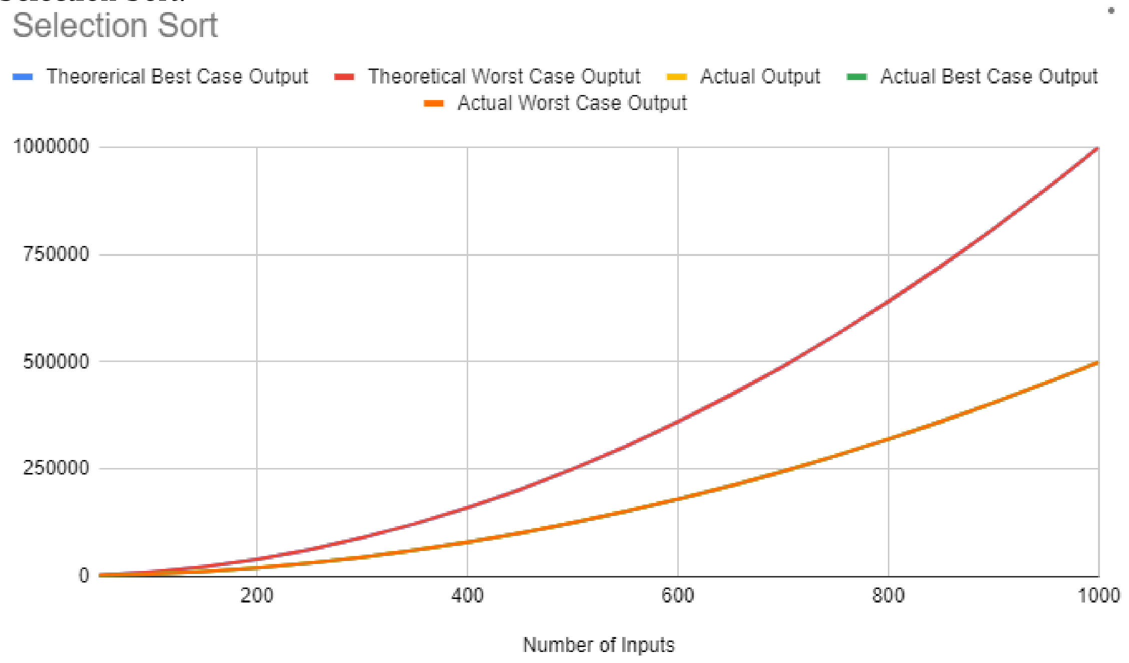
- Best Case: n^2 / $\Omega(n^2)$
- Worst Case: n^2 / $O(n^2)$
- Average Case: n^2 / $\theta(n^2)$

Graphs for varying input sizes: (Insertion Sort & Selection sort)--

Inserstion Sort:



Selection Sort:



NOTE: Best case and Worst Case outputs are overlapping as they are the same

Conclusion:

We performed analysis on two sorting algorithms, Selection sort and Insertion sort, and found out the following details:

Time Complexity for Selection Sort:

- Best Case: n^2 / $\Omega(n^2)$
- Worst Case: n^2 / $O(n^2)$
- Average Case: n^2 / $\theta(n^2)$

Time Complexity for Insertion Sort:

- Best Case: n / $\Omega(n)$
- Worst Case: n^2 / $O(n)$

Through this we can figure out that insertion sort is faster than selection sort in the best case scenario.