K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: EXCP B-1 **Roll No.:** 16014022096

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: Sorting Algorithms

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

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]}
$i \leftarrow j - 1$
WHILE $i > 0$ and $A[i] >$ key
DO $A[i+1] \leftarrow A[i]$
$i \leftarrow i - 1$
$A[i + 1] \leftarrow$ key

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
int count_i = 0;

void insertionSort(int A[], int n) {
    int i, j, key;

    for (j = 1; j < n; j++) {
        count_i++;
        key = A[j];
        i = j - 1;
        while (i >= 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i - 1;

        }
        A[i + 1] = key;
    }

}

int main() {
      int  n;
    printf("Enter value of n: ");
    scanf("%d", &n);

    int arr[n];
    printf("Original Array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10;
```

```
        printf("%d ", arr[i]);
    }
    printf("\n");

    insertionSort(arr, n);

    printf("Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Count : %d \n", count_i);

    return 0;
}
```

OUTPUT:

```
Enter value of n: 100
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9
Sorted Array: 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5
    5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9
Count : 99
```

```
Enter value of n: 500
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
    7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9 4 8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8 4 8 1
    0 3 0 4 4 4 4 7 6 3 1 7 5 9 6 2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0 4 8 8 8 9 7 7 6 4 3 0 3 0 9
    2 5 4 0 5 9 4 6 9 2 2 4 7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1 1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8 1 3
    1 1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7 5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3 2 1 5 9 9 1 4 9 1 0 7 5 8
    7 0 4 8 0 4 2 9 6 1 0 4 2 2 2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2 5 4 4 9 9 3 6 0 5 0 2 9 4 3 5 1 7 4 3 1 4 6 9 4
    2 2 6 4 1 2 8 8 9 2 8 8 8 6 8 3 8 3 3 3 8 0 4 7 6 8 9 0 6 8 7 9 0 3 3 3 7 3 2 6 5 2 6 5 8 7 9 6 0 4 1 0 4 8 7 0 8
    6 2 4 7 9 3 9 2 8 3 0 1 7 8 9 1 5 4 9 2 5 7 4 9 9 4 5 9 3 5 7 0 8 1 9 9 7 8 2 5 3 4 9 0 2 0 1 9 6 2
Sorted Array: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2
    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
    3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4
    4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5
    5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
    6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
    8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
Count : 499
```

```
Enter value of n: 700
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
    7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9 4 8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8 4 8 1
    0 3 0 4 4 4 4 7 6 3 1 7 5 9 6 2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0 4 8 8 8 9 7 7 6 4 3 0 3 0 9
    2 5 4 0 5 9 4 6 9 2 2 4 7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1 1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8 1 3
    1 1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7 5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3 2 1 5 9 9 1 4 9 1 0 7 5 8
    7 0 4 8 0 4 2 9 6 1 0 4 2 2 2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2 5 4 4 9 9 3 6 0 5 0 2 9 4 3 5 1 7 4 3 1 4 6 9 4
    2 2 6 4 1 2 8 8 9 2 8 8 8 6 8 3 8 3 3 3 8 0 4 7 6 8 9 0 6 8 7 9 0 3 3 3 7 3 2 6 5 2 6 5 8 7 9 6 0 4 1 0 4 8 7 0 8
    6 2 4 7 9 3 9 2 8 3 0 1 7 8 9 1 5 4 9 2 5 7 4 9 9 4 5 9 3 5 7 0 8 1 9 9 7 8 2 5 3 4 9 0 2 0 1 9 6 2 1 2 0 7 3 1 1
    9 0 5 6 7 7 4 0 6 4 7 4 8 5 8 2 6 0 6 6 4 6 2 8 9 6 0 7 0 1 0 1 3 7 7 2 6 4 5 2 0 2 9 0 9 9 4 5 1 0 3 7 8 8 6 0 4
    6 7 6 0 9 7 5 9 7 8 5 3 3 8 3 7 9 3 7 8 7 4 1 9 0 9 8 8 5 8 4 3 7 1 3 8 0 9 7 9 9 3 2 4 3 7 1 2 2 0 2 1 7 5 1 7 4
    1 7 1 1 1 7 0 4 0 8 5 1 6 6 2 1 9 6 4 8 0 8 1 0 2 2 9 7 5 6 4 6 3 7 9 7 4 9 1 7 0 6 0 8 5 3 9 4 1 5 4 1 5 5 4 9 8
    3 8 5 2 2 2 7 0 3 4 6 3 6 3 3 4 4 3 9 7 2 5
Sorted Array: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
    3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4
    4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
    4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
    5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
    6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
Count : 699
```

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 $i \leftarrow 1$ **TO** n-1 **DO**
  min $j \leftarrow i$;
  min $x \leftarrow A[i]$
  FOR $j \leftarrow i + 1$ to n do
    IF $A[j] <$ min x then
      min $j \leftarrow j$
      min $x \leftarrow A[j]$
  $A[\text{min } j] \leftarrow A[i]$
  $A[i] \leftarrow$ min x

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int count_i = 0;

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;

        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
            count_i++;
        }

        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;

    }
}

int main() {

    int  n;
    printf("Enter value of n: ");
```

```
    scanf("%d", &n);

    int arr[n];
    printf("Original Array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10;
        printf("%d ", arr[i]);
    }
    printf("\n");

    selectionSort(arr, n);

    printf("Sorted Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("Count : %d \n", count_i);

    return 0;
}
```

OUTPUT:

```
Enter value of n: 120
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
    7 2 2 6 1 0 6 1 5 9 4 9
Sorted Array: 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4
    4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 9 9
    9 9 9 9 9 9 9 9 9 9
Count : 7140
```

```
Enter value of n: 375
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
    7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9 4 8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8 4 8 1
    0 3 0 4 4 4 4 7 6 3 1 7 5 9 6 2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0 4 8 8 8 9 7 7 6 4 3 0 3 0 9
    2 5 4 0 5 9 4 6 9 2 2 4 7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1 1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8 1 3
    1 1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7 5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3 2 1 5 9 9 1 4 9 1 0 7 5 8
    7 0 4 8 0 4 2 9 6 1 0 4 2 2 2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2 5 4 4 9 9 3
Sorted Array: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
    4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6
    6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7
    7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
Count : 70125
```

```
Enter value of n: 750
Original Array: 3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3
    6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
    7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9 4 8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8 4 8 1
    0 3 0 4 4 4 4 7 6 3 1 7 5 9 6 2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0 4 8 8 8 9 7 7 6 4 3 0 3 0 9
    2 5 4 0 5 9 4 6 9 2 2 4 7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1 1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8 1 3
    1 1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7 5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3 2 1 5 9 9 1 4 9 1 0 7 5 8
    7 0 4 8 0 4 2 9 6 1 0 4 2 2 2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2 5 4 4 9 9 3 6 0 5 0 2 9 4 3 5 1 7 4 3 1 4 6 9 4
    2 2 6 4 1 2 8 8 9 2 8 8 8 6 8 3 8 3 3 3 8 0 4 7 6 8 9 0 6 8 7 9 0 3 3 3 7 3 2 6 5 2 6 5 8 7 9 6 0 4 1 0 4 8 7 0 8
    6 2 4 7 9 3 9 2 8 3 0 1 7 8 9 1 5 4 9 2 5 7 4 9 9 4 5 9 3 5 7 0 8 1 9 9 7 8 2 5 3 4 9 0 2 0 1 9 6 2 1 2 0 7 3 1 1
    9 0 5 6 7 7 4 0 6 4 7 4 8 5 8 2 6 0 6 6 4 6 2 8 9 6 0 7 0 1 0 1 3 7 7 2 6 4 5 2 0 2 9 0 9 9 4 5 1 0 3 7 8 8 6 0 4
    6 7 6 0 9 7 5 9 7 8 5 3 3 8 3 7 9 3 7 8 7 4 1 9 0 9 8 8 5 8 4 3 7 1 3 8 0 9 7 9 9 3 2 4 3 7 1 2 2 0 2 1 7 5 1 7 4
    1 7 1 1 1 7 0 4 0 8 5 1 6 6 2 1 9 6 4 8 0 8 1 0 2 2 9 7 5 6 4 6 3 7 9 7 4 9 1 7 0 6 0 8 5 3 9 4 1 5 4 1 5 5 4 9 8
    3 8 5 2 2 2 7 0 3 4 6 3 6 3 3 4 4 3 9 7 2 5 8 9 0 2 4 7 6 5 7 1 3 3 3 8 5 1 0 8 7 6 3 5 0 8 0 4 1 1 3 5 9 3 4 1 5
    0 8 3 5 6 5 9 9 0 7 6 3 7 6 1
Sorted Array: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
    3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
    3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
    4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
    5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
    6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
    6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
    7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
    9 9 9 9 9 9 9 9 9 9 9 9 9 9
Count : 280875
```

The space complexity of Insertion sort:

The space complexity of the insertion sort algorithm is O(1), which means it requires constant space regardless of the size of the input array.

The space complexity of Selection sort:

The space complexity of the selection sort algorithm is O(1), which means it requires constant space regardless of the size of the input array.

Time complexity for Insertion sort:

The time complexity of insertion sort algorithm are:
Worst-case: O(n^2)
Best-case: O(n)
Average-case: O(n^2)

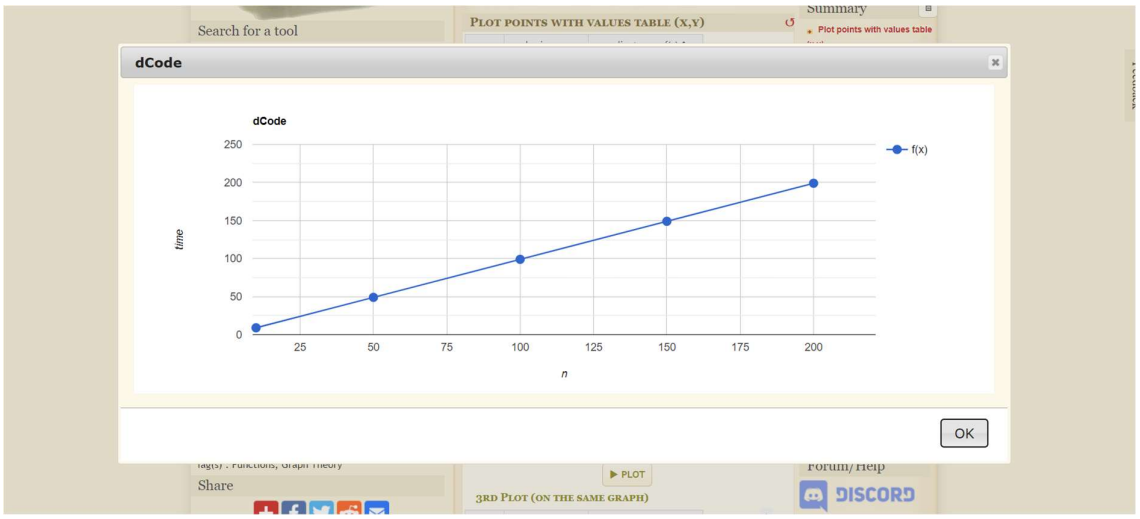
Time complexity for selection sort:

The time complexity of selection sort algorithm are:
Worst-case: O(n^2)
Best-case: O(n)
Average-case: O(n^2)

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

SELECTION SORT:

INSERTION SORT:



CONCLUSION:

We have successfully analysed asymptotic running time and space complexity of insertion sort and selection sort algorithm.