Course Name:	**Analysis of Algorithms**	**Semester:**	**IV**
Date of Performance:	**21 / 01 / 2024**	**Batch No:**	**EXCP B1**
Faculty Name:	**Prof. Payal Varangoankar**	**Roll No:**	**16014022096**
Faculty Sign & Date:		**Grade/Marks:**	

Experiment No: 1

Title: Implementation of Insertion sort.

Aim and Objective of the Experiment:
To analyze performance of sorting methods

COs to be achieved:
CO1: Analyze the asymptotic running time and space complexity of algorithms.

Apparatus / Software tools used:

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 subproblems. These sub-problems must be solved and then a method must be found to combine sub-solutions into a solution of the whole. The divide-and-conquer strategy can be reapplied if the sub-problems are still relatively large. Often the sub-problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases, a recursive algorithm naturally expresses the reapplication of the divide-and-conquer principle. Now smaller and smaller subproblems of the same kind are generated until eventually subproblems that are small enough to be solved without splitting are produced.

Code:

INSERTION SORT:

```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;
}
```

SELECTION SORT:

```
#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;
}
```

Stepwise-Procedure / Algorithm:

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

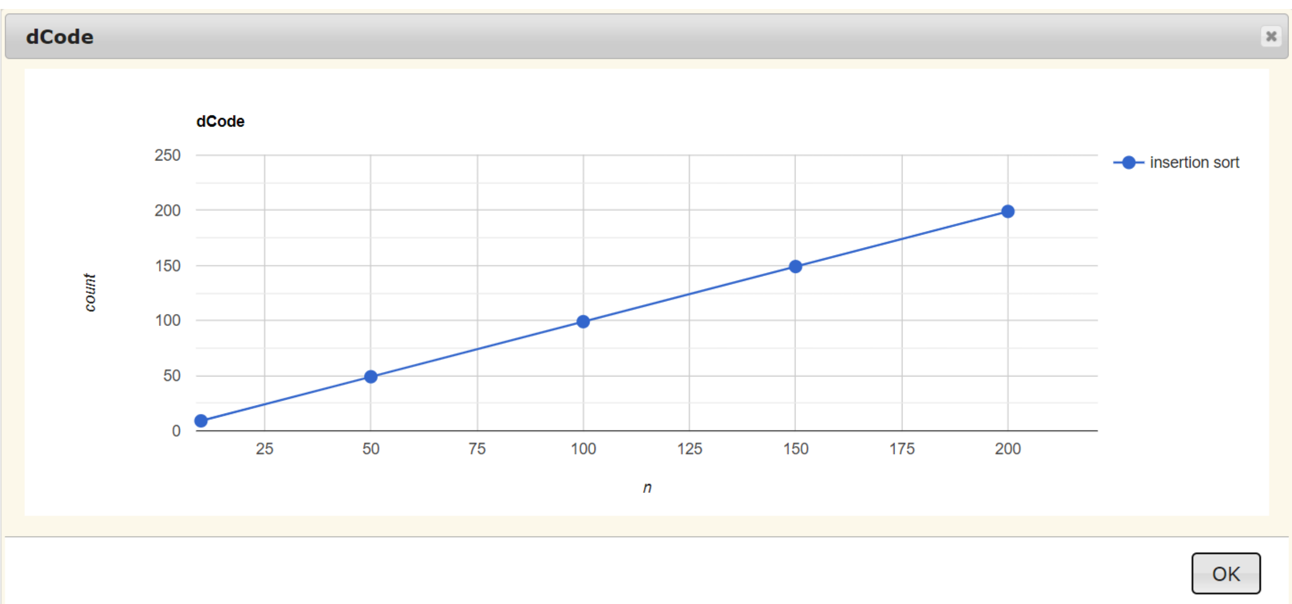
```
FOR j ← 2 TO length[A]
    DO  key ← A[j]
          {Put A[j] into the sorted sequence A[1 . . j − 1]}
        i ← j − 1
        WHILE i > 0 and A[i] > key
              DO A[i +1] ← A[i]
                    i ← i − 1
        A[i + 1] ← key
```

Observation Table:

Graphs for varying input sizes of Insertion Sort

	abscissa x ⇢	ordinate y or f(x) ⇡
1	10	9
2	50	49
3	100	99
4	150	149
5	200	199
6	...	...

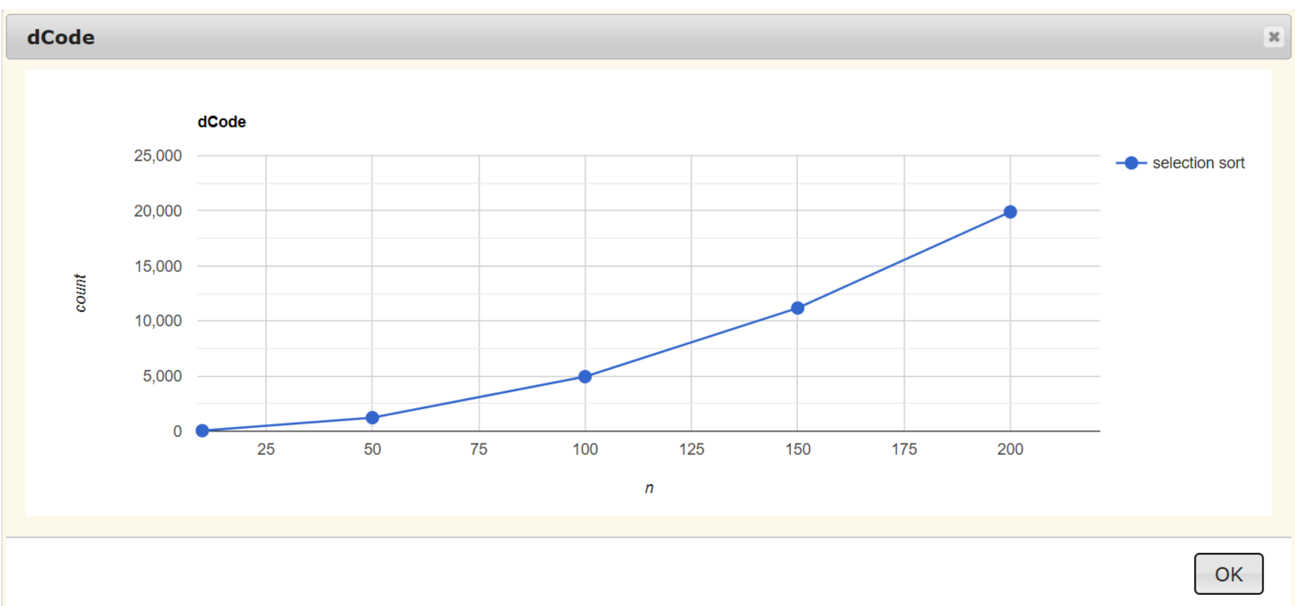
★ ORIGINS (●) AUTOMATICALLY CALCULATED
( ) SET TO (0,0)
★ HORIZONTAL X-AXIS NAME: n
★ VERTICAL Y-AXIS NAME: count
★ LEGEND: insertion sort
★ DOT SIZE: 10



Graphs for varying input sizes of Selection Sort:

	abscissa x →	ordinate y or f(x) ↑
1	10	45
2	50	1225
3	100	4950
4	150	11175
5	200	19900
6	...	...

- Origins: (selected) Automatically calculated / Set to (0,0)
- Horizontal x-axis name: n
- Vertical y-axis name: count
- Legend: selection sort
- Dot size: 10



Output:

INSERTION SORT:

```
Enter value of n: 5
Original Array: 3 6 7 5 3
Sorted Array: 3 3 5 6 7
Count : 4
```

```
Enter value of n: 10
Original Array: 3 6 7 5 3 5 6 2 9 1
Sorted Array: 1 2 3 3 5 5 6 6 7 9
Count : 9
```

```
Enter value of n: 10
Original Array: 83 86 77 15 93 35 86 92 49 21
Sorted Array: 15 21 35 49 77 83 86 86 92 93
Count : 9
```

SELECTION SORT:

```
Enter value of n: 10
Original Array: 3 6 7 5 3 5 6 2 9 1
Sorted Array: 1 2 3 3 5 5 6 6 7 9
Count : 45
```

```
Enter value of n: 10
Original Array: 83 86 77 15 93 35 86 92 49 21
Sorted Array: 15 21 35 49 77 83 86 86 92 93
Count : 45
```

Calculation:

The Time and Space complexity of Insertion sort:

PAGE No.
DATE / /

16014022096

11/03/24 23:14

INSERTION SORT (int A[ ], int n)

```
FOR j ← 2 to length[A]                n times
    DO key ← A[j]
       i ← j-1                         n(1) times
       WHILE i > 0 and A[i] > key      n(n) times
           DO A[i+1] ← A[i]
              i ← i-1
       A[i+1] ← key
```

time complexity = $O(n+n^2) \equiv O(n^2)$ average / best

In best case A[ ] is already sorted hence $O(n)$

space complexity = $O(1)$

Shot on OnePlus
By w.vrishank

Post Lab Subjective/Objective type Questions:
Solve the problem theoretically which was implemented during practical

16014022096

INSERSTION SORT

3	6	7	5	3

first two elements compared
no swap
compared 2nd and 3rd
no swap
compare 3rd and 4th

3	6	5	7	3

compare 2nd and ~~3rd~~ key

3	5	6	7	3

compare 4th and 5th

3	5	6	3	7

compare 3rd and 4th (key)

3	5	3	6	7

compare 2nd and 3rd (key)

3	3	5	6	7

Conclusion:

We have successfully implemented Insertion sort and Selection sort and derived following analysis

Insertion Sort:

- Best Case Time Complexity: O(n)
- Average Case Time Complexity: O(n^2)
- Worst Case Time Complexity: O(n^2)
- Space Complexity: O(1)

Selection Sort

- Best Case Time Complexity: O(n^2)
- Average Case Time Complexity: O(n^2)
- Worst Case Time Complexity: O(n^2)
- Space Complexity: O(1)

Signature of faculty in-charge with Date: