ALGORITHMS LABORATORY
[CS39001]
Individual Work

Lab. No.- 4 **Date. : 11.08.2025**

Roll Number:	2328145	Branch/Section:	CSSE-1
Name in Capital:	ADARSH UPADYAY		

Program No: 3.1

Program Title:

Write a menu (given as follows) driven program to sort a given set of elements using the Insertion sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

INSERTION SORT MENU
0. Quit
1. n Random numbers=>Array
2. Display the Array
3. Sort the Array in Ascending Order
4. Sort the Array in Descending Order
5. Best Case Time Complexity
6. Worst Case Time Complexity
7. Average Case Time Complexity
8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000
Enter your choice:

Input/Output Screenshots:

RUN-1:

```
INSERTION SORT MENU
0. Quit
1. n Random numbers => Array
2. Display the Array
3. Sort the Array in Ascending Order
4. Sort the Array in Descending Order
5. Best Case Time Complexity
6. Worst Case Time Complexity
7. Average Case Time Complexity
8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000
Enter your choice: 1
Enter n: 4
Random array generated.

INSERTION SORT MENU
0. Quit
1. n Random numbers => Array
2. Display the Array
3. Sort the Array in Ascending Order
4. Sort the Array in Descending Order
5. Best Case Time Complexity
6. Worst Case Time Complexity
7. Average Case Time Complexity
8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000
Enter your choice: 6
Worst Case Time Complexity: O(n^2)

INSERTION SORT MENU
0. Quit
1. n Random numbers => Array
2. Display the Array
3. Sort the Array in Ascending Order
4. Sort the Array in Descending Order
5. Best Case Time Complexity
6. Worst Case Time Complexity
7. Average Case Time Complexity
8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000
```

RUN-2:

```
KIIT0001@BT08261 MINGW64 ~/Desktop/DAA-Leetcode (main)
$ ./a.exe
Enter array size: 6

INSERTION SORT MENU
0. Quit
1. n Random numbers => Array
2. Display the Array
3. Sort the Array in Ascending Order
4. Sort the Array in Descending Order
5. Best Case Time Complexity
6. Worst Case Time Complexity
7. Average Case Time Complexity
8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000

Enter choice: 8
Sl       n       Best(s)        Worst(s)       Average(s)
1       5000    0.000000        0.039000        0.017000
2       10000   0.000000        0.117000        0.059000
3       15000   0.000000        0.254000        0.150000
4       20000   0.000000        0.458000        0.218000
5       25000   0.000000        0.718000        0.358000
6       30000   0.000000        1.024000        0.527000
7       35000   0.000000        1.359000        0.712000
8       40000   0.000000        1.805000        0.918000
9       45000   0.000000        2.258000        1.152000
10      50000   0.000000        2.793000        1.460000

Enter choice:
```

Source code

```c
#include <stdio.h>
#include<stdlib.h>
#include <time.h>
#define MAX_SIZE 100000
int arr[MAX_SIZE];
int n = 0;
void quit() {
    printf("Exiting the program.\n");
    exit(0);
}
void nrandomArray(){
    printf("Enter n: ");
    scanf("%d", &n);
    if (n > MAX_SIZE) n = MAX_SIZE;
    srand(time(NULL));
    for(int i = 0; i < n; i++) {

        arr[i] = rand() % 100;
    }
    printf("Random array generated.\n");

}
void displayArray(){
    printf("Current array elements are: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
void insertionSortAscending(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void insertionSortDescending(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] < key) {
            arr[j + 1] = arr[j];
            j--;
```

Conclusion/Observation

This program effectively demonstrates the required concept and achieves the intended result as per the problem statement. The implementation is clear and allows for easy understanding of the underlying logic. By running and analyzing the output, users can validate the correctness and efficiency of the approach used.