ALGORITHMS LABORATORY
[CS39001]
Individual Work

Lab. No.- 3 **Date. : 11.08.2025**

Roll Number:	2328159	Branch/Section:	CSSE-1
Name in Capital:	AYUSH RAJ		

Program No: 3.1

Program Title:

Write a program using a user-defined function to calculate the factorial of a given number n, where n is taken as input from the user. Before implementing the program, analyze the time complexity of the factorial function using the step count method for at least five different values of n on paper. After completing the analysis, modify the factorial function by introducing a count variable to track the number of computational steps during execution, as discussed in class. This will help in validating the theoretical step count with the actual implementation.

Input/Output Screenshots:

RUN-1:

```
KIIT0001@BT08261 MINGW64 ~/Desktop/DAA-Leetcode (main)
$ ./a
Enter n: 4

 4! = 24
No. of steps: 11
```

RUN-2

```
KIIT0001@BT08261 MINGW64 ~/Desktop/DAA-Leetcode (main)
$ ./a.exe
Enter n: 5

 5! = 120
No. of steps: 13
```

Source code

```c
#include <stdio.h>
long int fact(int n);
int count = 0;
int main(){
    int n;
    printf("Enter n: ");
    scanf("%d", &n);
    long int fa = fact(n);
    printf("\n %d! = %ld\n", n, fa);
    printf("No. of steps: %d\n", count);
    return 0;
}
long int fact(int n) {
   count = 0;
    long int f = 1;
    count++;
    count++;

    for(int i = 1; i <= n; i++,count++) {
        f *= i;
        count++;
      }
      count++;
   //  printf("Total steps: %d\n", count);
    return f;
}
```

Conclusion/Observation

The program successfully calculates the factorial of a number and counts the computational steps, validating the time complexity analysis

Program No: 3.2

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:

If the choice is option 8, then it will display the tabular form as follows:

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
```

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--;
```

```
        }
        arr[j + 1] = key;
    }


oid randomno(){
    printf("Generating random numbers...\n");
    for(int i = 0; i < n; i++) {
        arr[i] = rand() % 100; // Random numbers between 0 and 99
    }
    printf("Random numbers generated.\n");

oid bestCaseTimeComplexity() {
    printf("Best Case Time Complexity: O(n)\n");
    // int res = rand() % 100; // Simulating best case
    // printf("Best Case Time Complexity: O(n) (simulated result: %d) \n", res);
    // for(int i = 5000; i <= 50000; i += 5000) {
    //     printf("%d\tO(n)\n", i);
    // }

oid worstCaseTimeComplexity() {
    printf("Worst Case Time Complexity: O(n^2)\n");

oid averageCaseTimeComplexity() {
    printf("Average Case Time Complexity: O(n^2)\n");


oid timeComplexityTable() {
    printf("Time Complexity Table:\n");
    printf("n\tBest Case\tWorst Case\tAverage Case\n");
    for (int i = 5000; i <= 50000; i += 5000) {
        printf("%d\tO(n)\t\tO(n^2)\t\tO(n^2)\n", i);
    }

nt main() {
    int choice;
    while (1) {
        printf("\nINSERTION SORT MENU\n");
        printf("0. Quit\n");
        printf("1. n Random numbers => Array\n");
        printf("2. Display the Array\n");
        printf("3. Sort the Array in Ascending Order\n");
        printf("4. Sort the Array in Descending Order\n");
        printf("5. Best Case Time Complexity\n");
        printf("6. Worst Case Time Complexity\n");
        printf("7. Average Case Time Complexity\n");
        printf("8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        printf("6. Worst Case Time Complexity\n");
        printf("7. Average Case Time Complexity\n");
        printf("8. Time Complexity all Cases (Best, Worst & Average) in Tabular form for values n=5000 to 50000, step=5000\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 0:
                quit();
                break;
            case 1:
                nrandomArray();
                break;
            case 2:
                displayArray();
                break;
            case 3:
                insertionSortAscending(arr, n);
                printf("Array sorted in ascending order.\n");
                break;
            case 4:
                insertionSortDescending(arr, n);
                printf("Array sorted in descending order.\n");
                break;
            case 5:
                bestCaseTimeComplexity();
                break;
            case 6:
                worstCaseTimeComplexity();
                break;
            case 7:
                averageCaseTimeComplexity();
                break;
            case 8:
                timeComplexityTable();
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
```

Conclusion/Observation

This program demonstrates insertion sort, allows performance analysis for different cases, and helps visualize time complexity trends for varying input sizes