Assignment Number

Problem Statement

Program in C to implement Insertion sort in ascending order.

Theory

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages:

- Efficient for (quite) small data sets, much like other quadratic sorting algorithms
- More efficient in practice than most other simple quadratic (i.e., $O(n^2)$) algorithms such as selection sort or bubble sort
- Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is $O(nk)$ when each element in the input is no more than k places away from its sorted position
- Stable; i.e., does not change the relative order of elements with equal keys
- In-place; i.e., only requires a constant amount O(1) of additional memory space
- Online; i.e., can sort a list as it receives it

Complexity:

- Best case: O(n)
- Worst case: $O(n^2)$
- Average case: $O(n^2)$

Algorithm

Input : An unsorted array, say **a[].**
Output : Elements of the input array **a[]** sorted in ascending order**.**
Steps :
Step 1 : Print "Enter the number of elements of the array: "
Step 2 : Input n
Step 3 : Repeat Step 3.a to Step 3.b For i=0 to i<n
    a. Print "Enter the element no. "i+1
    b. Input a[i]
Step 4 : Print "The sorted array is: "
Step 5 : Repeat Step 5.a to Step 5.7 For i=1 to i<n
    a. Set key=a[i]
    b. Set j=i-1
    c. Repeat Step 5.c.i while(j>=0 AND a[j]>key)
        i. Set a[j+1]=a[j-1]
        ii. Set j = j - 1
Step 6 : a[j+1]=key
Step 7 : Set i=i+1
Step 8 : Repeat Step 6.i to Step 6.ii For i=0 to i<n
        i. Print a[i]
        ii. Set i=i+1

Source Code

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *a,i,j,min,t,n,k;
    printf("Enter the number of elements of the array: ");
    scanf("%d",&n);
    a=(int*)malloc(n*sizeof(int));
    for(i=0;i<n;i++){
        printf("Enter the element no. %d: ",i+1);
```

```
        scanf("%d",a+i);
    }
    printf("The sorted array is: \n");

    for (i=1;i<n;i++){
        key=a[i];
        j=i-1;
                /* Move elements of a[0..i-1], that are
                    greater than key, to one position ahead
                of their current position */
        while(j>=0&&a[j]>key){
                a[j+1]=a[j--];
        }
        a[j+1]=key;
    }
    for(i=0;i<n;i++){
        printf("%d\n",*(a+i));
    }
    return 0;
}
```

Input and Output

Set 1 :

```
Enter the number of elements of the array: 5
Enter the element no. 1: -381
Enter the element no. 2: 382
Enter the element no. 3: 481
Enter the element no. 4: 0
Enter the element no. 5: 38
The sorted array is:
-381
0
38
382
```

481

Set 2 :
Enter the number of elements of the array: 5
Enter the element no. 1: 382
Enter the element no. 2: 39
Enter the element no. 3: 981
Enter the element no. 4: 28
Enter the element no. 5: 30
The sorted array is:
28
30
39
382
981

Discussion

1. Insertion sort is adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is $O(nk)$ when each element in the input is no more than k places away from its sorted position
2. Insertion sort is stable; i.e., does not change the relative order of elements with equal keys
3. Insertion sort can be done in-place; i.e., requiring no additional memory space
4. Insertion sort is online; i.e., can sort a list as it receives it