Date-

Assignment No. :

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:

- Simple implementation: Jon Bentley shows a three-line C version, and a five-line optimized version
- 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

Algorithm:

Input specification: An unsorted array say **a[].**

Output specification: Sorted input array **a[].**

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.b For i=1 to i<n
   a. Repeat For j=0 to j<i
      i. If(a[i]<a[j]) Then
         1. Set t=a[i]
         2. Set a[i]=a[j]
         3. Set a[j]=t
      ii. j=j+1
   b. i=i+1
Step 6 Repeat Step 6.i to Step 6.ii For i=0 to i<n
      i. Print a[i]
      ii. 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++){//Controlling the unsorted part
		for(j=0;j<i;j++){//Controlling the sorted part
			if(a[i]<a[j]){
				t=a[i];
				a[i]=a[j];
```

```
                        a[j]=t;
                    }
                }
        }
        for(i=0;i<n;i++){
                printf("%d\n",*(a+i));
        }
        return 0;
}
```

Input & Output:

```
Enter the number of elements of the array: 5
Enter the element no. 1: 11
Enter the element no. 2: 78
Enter the element no. 3: 54
Enter the element no. 4: 87
Enter the element no. 5: 2
The sorted array is:
2
11
54
78
87
```

Discussion:

1. 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. Stable; i.e., does not change the relative order of elements with equal keys
3. In-place; i.e., only requires a constant amount O(1) of additional memory space
4. Online; i.e., can sort a list as it receives it
5. Complexity:
    a. Best case: O(n)
    b. Worst case: $O(n^2)$
    c. Average case: $O(n^2)$