**Date:-**

**ASSIGNMENT NUMBER :-**

**PROBLEM STATEMENT:-**

**Program in C to implement Insertion sort in ascending order.**

**THEORY:-**

**Insertion sort** is a simple [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) that builds the final [sorted array](https://en.wikipedia.org/wiki/Sorted_array) (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as [quicksort](https://en.wikipedia.org/wiki/Quicksort" \o "Quicksort), [heapsort](https://en.wikipedia.org/wiki/Heapsort" \o "Heapsort), or [merge sort](https://en.wikipedia.org/wiki/Merge_sort). However, insertion sort provides several advantages:

* Simple implementation: [Jon Bentley](https://en.wikipedia.org/wiki/Jon_Bentley_(computer_scientist)) shows a three-line [C](https://en.wikipedia.org/wiki/C_(programming_language)) version, and a five-line [optimized](https://en.wikipedia.org/wiki/Program_optimization) 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](https://en.wikipedia.org/wiki/Big_O_notation)(*n*2)) algorithms such as [selection sort](https://en.wikipedia.org/wiki/Selection_sort) or [bubble sort](https://en.wikipedia.org/wiki/Bubble_sort)

**ALGORITHM:**

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

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

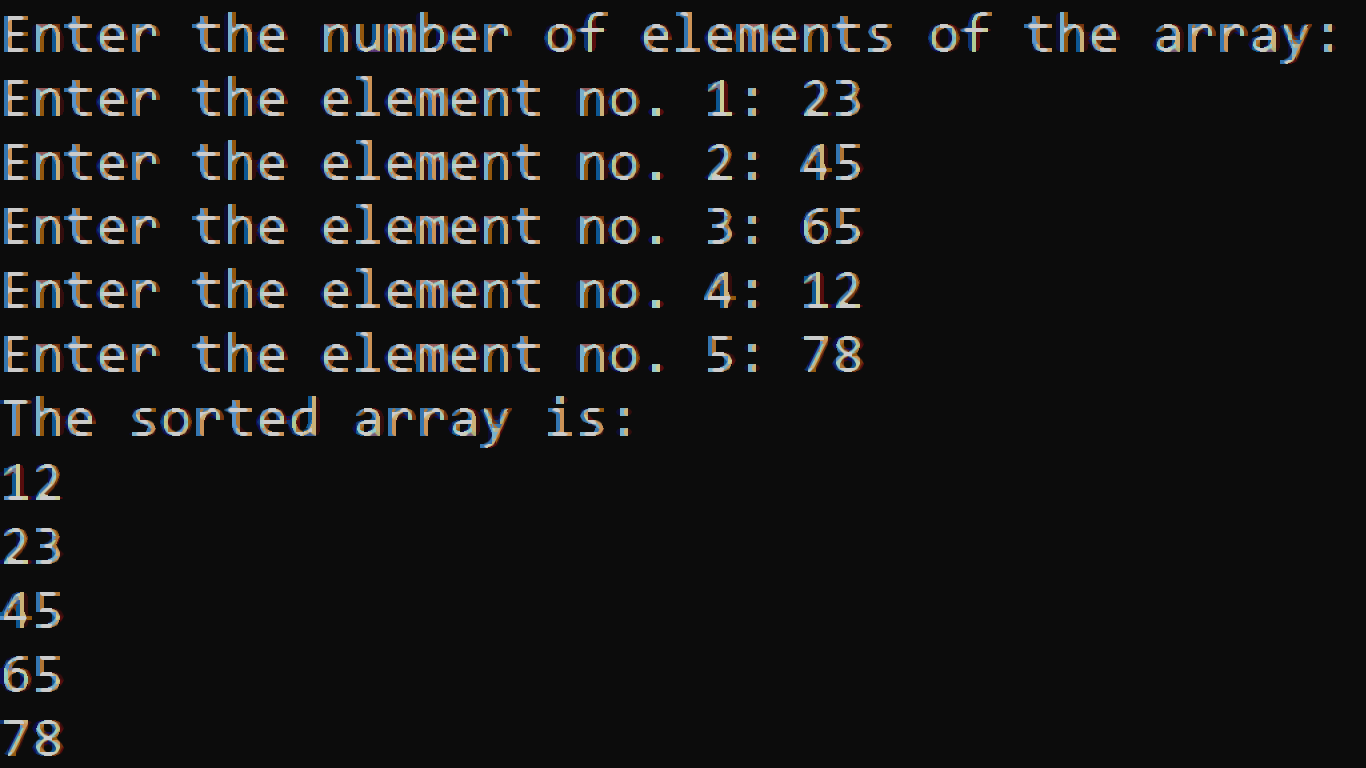
**Steps:**

1. Print "Enter the number of elements of the array: "
2. Input n
3. Repeat Step 3.a to Step 3.b For i=0 to i<n
   1. Print "Enter the element no. "i+1
   2. Input a[i]
4. Print "The sorted array is: "
5. Repeat Step 5.a to Step 5.b For i=1 to i<n
   1. Repeat For j=0 to j<i
      1. If(a[i]<a[j]) Then
         1. Set t=a[i]
         2. Set a[i]=a[j]
         3. Set a[j]=t
      2. j=j+1
   2. i=i+1
6. Repeat Step 6.i to Step 6.ii For i=0 to i<n
   * 1. Print a[i]
     2. i=i+1

**SOURCE CODE:**

|  |
| --- |
|  |
|  | #include<stdio.h>  #include<stdlib.h> |
|  | int main() |
|  | { |
|  | int \*a,i,j,n; |
|  | 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++){ |
|  | int 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]; |
|  | j--; |
|  | } |
|  | a[j+1]=key; |
|  | } |
|  | for(i=0;i<n;i++){ |
|  | printf("%d\n",\*(a+i)); |
|  | } |
|  | return 0; |
|  | } |

**INPUT & OUTPUT:**-



**DISCUSSION:**

1. [Adaptive](https://en.wikipedia.org/wiki/Adaptive_sort), i.e., efficient for data sets that are already substantially sorted: the [time complexity](https://en.wikipedia.org/wiki/Time_complexity) is *O*(*nk*) when each element in the input is no more than *k* places away from its sorted position
2. [Stable](https://en.wikipedia.org/wiki/Stable_sort); i.e., does not change the relative order of elements with equal keys
3. [In-place](https://en.wikipedia.org/wiki/In-place_algorithm); i.e., only requires a constant amount O(1) of additional memory space
4. [Online](https://en.wikipedia.org/wiki/Online_algorithm); i.e., can sort a list as it receives it
5. Complexity:
   1. Best case: O(n)
   2. Worst case: O(n2)
   3. Average case: O(n2)

----------------------------------