**Date-**

**Assignment No. :**

**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), [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)
* [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
* [Stable](https://en.wikipedia.org/wiki/Stable_sort); i.e., does not change the relative order of elements with equal keys
* [In-place](https://en.wikipedia.org/wiki/In-place_algorithm); i.e., only requires a constant amount O(1) of additional memory space
* [Online](https://en.wikipedia.org/wiki/Online_algorithm); i.e., can sort a list as it receives it

**Complexity:**

Best case: O(n)

Worst case: O(n2)

Average case: O(n2)

**Algorithm:**

**Input specification:** An array (Sorted for linear/Binary search or Unsorted for linear search) say **a[]**, where the search will be done,

The element which need to be searched, say **find** and the number of elements in the array a[], say **n**.

**Output specification:** Success message of the search with the position of the element or appropriate failure message.

**Steps:**

Algorithm for method linear\_search(a[], n, find):

1. For (c = 0 to n-1 )
2. If (a[c] == find) Then
3. Print search" is present at location " c+1
4. Exit
5. End If
6. Set c=c+1
7. End For
8. Print search" is not present in array."

Algorithm for method binary\_search(a[], n, find):

1. Set first=0
2. Set last = n-1
3. Set middle = (first+last)/2
4. While (first <= last) Then
5. If (a[middle] < find) Then
6. Set first = middle + 1
7. Else If (a[middle] == find) Then
8. Print search" is present at location "middle+1
9. Else
10. Set last = middle - 1
11. Set middle = (first + last)/2
12. If (first > last) Then
13. Print search" is not present in array."

**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);

}

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++){

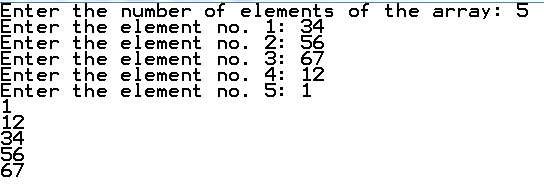
printf("%d\n",\*(a+i));

}

return 0;

}

**Input & Output:**



**Discussion:**

1. Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to linear search.
2. A linear search scans one item at a time, without jumping to any item .
   1. The worst case complexity is  O(n), sometimes known an O(n) search.
   2. Time taken to search elements keep increasing as the number of elements are increased.
3. A binary search however, cut down your search to half as soon as you find middle of a sorted list.
   1. The middle element is looked to check if it is greater than or less than the value to be searched.
   2. Accordingly, search is done to either half of the given list.
4. Input data needs to be sorted in Binary Search and not in Linear Search
5. Linear search does the sequential access whereas Binary search access data randomly.
6. Time complexity of linear search- O(n) , Binary search has time complexity-O(log n).
7. Linear search performs equality comparisons and Binary search performs ordering comparisons.