NOTES

Bachelor of Science (First Year)
CSC-102-Data Structures
Title of the Unit : Searching and Sorting
Module Name : Insertion Sort
Module Number : 35

An insertion sort is one that sorts a set of records by inserting records into an existing sorted file. Insertion sort 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 quick sort, heap sort, or merge sort.

To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to its predecessor.

3: If the key element is smaller than its predecessor, compare it to the elements efore. Move the greater elements one position up to make space for the swapped element.

Consider the following array:

[25] [57 48 37 12 92 86 33]

The first element **x[0] ie 25 is considered as a sorted list of 1 item.**

The remaining array from **x[1] to x[7] ie 57 to 33 are considered as unsorted list.**

The first element from the unsorted list, x[1] ie 57, is compared with its predessesor x[0] ie 25. Since 57 >25 , 57 is added to the sorted list on the right side of 25.

After the first pass, sorted list has two elements x[0] and x[1] as 25 and 57 while the unsorted list ranges from x[2] to x[7] ie 48 to 33.

[25 57] [48 37 12 92 86 33]

After the first pass,

[25 57] [48 37 12 92 86 33]

The first element of unsorted list x[2] ie 48 is now compared with its predecessor ie 57, since 48<57, 48 is also compared with 57s predecessor 12 and we find 48>12.

The sorted list from number 57 onwards, is shifted by one position to the right thereby making place for the number 48 to be added before 57.

After the second pass, the sorted list is 25 48 57

[25 48 57] [37 12 92 86 33]

The process continues till all elements from unsorted list are inserted properly in the sorted list.

The complete set of passes are the following:

Pass 0--> **[25] [57 48 37 12 92 86 33]**

Pass 1--> **[25 57] [48 37 12 92 86 33]**

Pass 2--> **[25 48 57] [37 12 92 86 33]**

Pass 3--> **[25 37 48 57] [12 92 86 33]**

Pass 4--> [12 25 37 48 57] [92 86 33]
Pass 5--> [12 25 37 48 57 92] [86 33]
Pass 6--> [12 25 37 48 57 86 92] [33]
Pass 7--> [12 25 37 48 57 86 92 33]

Here is the algorithm for Insertion Sort

Start InsertionSort(array)

Take the first element, it is already sorted. return 1

Pick next element

Compare with all elements in the sorted sub-list

Shift all the elements in the sorted sub-list that is greater than the value to be sorted

Insert the value

Repeat until array is sorted

Return array

End InsertionSort

Let us now discuss the efficiency of Insertion sort. If the initial file is sorted, only one comparison is made on each pass, so that the sort is 0(n) for best case input. If the file is initially sorted in the reverse order ie worst case input, the sort is $0(n^2)$, since the total number of comparisons is (n-1)+(n-2)+(n-3)+….+1 comparisons ie n*(n-1)/2 comparisons, which is **$0(n^2)$.** However, the simple insertion sort is still usually better than the bubble sort. The closer the file is to sorted order, the more efficient the simple insertion sort becomes.