Insertion Sort.

Insertion sort is simple sorting algorithm which is used to sort smaller lists. But, it is very inefficient if list are larger. For larger lists we use Merge sort, Heap Sort or Quick sort.

Lean [Bubble](http://data-structure-learning.blogspot.com/2015/05/bubble-sort.html) and [Selection](http://data-structure-learning.blogspot.com/2015/05/selection-sort.html) sort.

Let us discuss few advantages of Insertion sort:

Efficient for smaller data sets: Insertion sort is better for small data set and also better when data set is almost sorted.

Online: Means we can sort list when it is getting populated.

In-Place: Insertion sort has space complexity of O(1).

Stable: Insertion sort maintains relative order of elements that have same value.

Simple Implementation: Insertion sort shifts the elements in array. So code is fairly easy to implement and concise in size.

Let us now discuss the Insertion sort working.

One good thing about insertion sort is that it finds the location of element where it belongs in sorted list. This is absolute thing that insertion sort provides.

So the sorting algorithm iterates, taking one element in each repetition. Now that element is placed in proper location where it would belong in sorted list.

We start out outer loop from array index 1. The reason behind selecting index from 1 is that we will compare all the preceding elements and shift them.

**for** (**int** j = 1; j < n; j++) {

Take a key for which we will compare with other elements.

**int** key = arr[j];

Now take the preceding index of the key.

**int** i = j - 1;

Check the underflow condition (index should be greater than 0) and if previous values are greater than the key then shift them all.

**while** ((i > -1) && (arr[i] > key)) {

arr[i + 1] = arr[i];

i--;

}

Now place the key in proper index where it was supposed to be in sorted array.

arr[i + 1] = key;

Insertion sort is now done.

Time complexity:

If the array is sorted then time complexity is O(1) as there won’t be any shifting.

But if the array is reversed sorted and then we try to sort it will take O(n2), as there will be shifting done for each element in list.

Following is code for insertion sort method. It takes 2 parameters array and length of array.

**public** **static** **void** insertionSort(**int**[] arr, **int** n) {

**for** (**int** j = 1; j < n; j++) {

**int** key = arr[j];

**int** i = j - 1;

**while** ((i > -1) && (arr[i] > key)) {

arr[i + 1] = arr[i];

i--;

}

arr[i + 1] = key;

}

}

Output per pass is as follows:

Pass 1.0 : 6 6 4 3 2 1

Pass output 5 6 4 3 2 1

Pass 2.1 : 5 6 6 3 2 1

Pass 2.0 : 5 5 6 3 2 1

Pass output 4 5 6 3 2 1

Pass 3.2 : 4 5 6 6 2 1

Pass 3.1 : 4 5 5 6 2 1

Pass 3.0 : 4 4 5 6 2 1

Pass output 3 4 5 6 2 1

Pass 4.3 : 3 4 5 6 6 1

Pass 4.2 : 3 4 5 5 6 1

Pass 4.1 : 3 4 4 5 6 1

Pass 4.0 : 3 3 4 5 6 1

Pass output 2 3 4 5 6 1

Pass 5.4 : 2 3 4 5 6 6

Pass 5.3 : 2 3 4 5 5 6

Pass 5.2 : 2 3 4 4 5 6

Pass 5.1 : 2 3 3 4 5 6

Pass 5.0 : 2 2 3 4 5 6

Pass output 1 2 3 4 5 6

Following is code that prints all the passes.

**package** com.Sorting;

**class** InsertionSort {

**public** **static** **void** main(String args[]) {

**int** arr[] = { 6, 5, 4, 3, 2, 1 };

**int** n = arr.length;

*insertionSort*(arr, n);

}

**public** **static** **void** insertionSort(**int**[] arr, **int** n) {

**for** (**int** j = 1; j < n; j++) {

**int** key = arr[j];

**int** i = j - 1;

**while** ((i > -1) && (arr[i] > key)) {

arr[i + 1] = arr[i];

System.***out***.print("Pass " + j + "."+i+" : ");

*display*(arr);

i--;

}

arr[i + 1] = key;

System.***out***.print("Pass output ");

*display*(arr);

System.***out***.println();

}

}

**static** **void** display(**int**[] arr){

**for** (**int** k = 0; k < arr.length; k++) {

System.***out***.print(arr[k] + " ");

}

System.***out***.println();

}

}

That’s all. In next post we will learn Merge and Quick Sort.