Insertion Sort (Insertion)

Insertion Sort

Insertion sort works by taking each element of the input array and *inserting* it into its correct position relative to all the elements that have been inserted so far.

It does this by partitioning the array into a sorted part at the front and an unsorted part at the end.

Initially the sorted part is just the first cell of the array and the unsorted part is the rest.

In each pass it takes the first element of the unsorted part and inserts it into its correct position in the sorted part, simultaneously growing the sorted part and shrinking the unsorted part by one cell.

Example of an Insertion Sort run

1. $5 \mid \underline{12}\ ,\ 6,\ 3,\ 11,\ 8,\ 4$
2. $5,\ 12 \mid \underline{6}\ ,\ 3,\ 11,\ 8,\ 4$
3. $5,\ 6,\ 12, \mid \underline{3}\ ,\ 11,\ 8,\ 4$
4. $3,\ 5,\ 6,\ 12 \mid \underline{11}\ ,\ 8,\ 4$
5. $3,\ 5,\ 6,\ 11,\ 12 \mid \underline{8}\ ,\ 4$
6. $3,\ 5,\ 6,\ 8,\ 11,\ 12 \mid \underline{4}$
7. $3,\ 4,\ 5,\ 6,\ 8,\ 11,\ 12 \mid$

Insertion Sort Pseudo-Code

```
1  insertionsort(a, n) {
2    for ( i = 1 ; i < n ; i++ ) {
3      j = i
4      t = a[j]
5      while ( j > 0 && t < a[j-1] ) {
6        a[j] = a[j-1]
7        j--
8      }
9      a[j] = t
10   }
11 }
```

Insertion Sort Complexity

The outer loop is iterated $n-1$ times

In the worst case, the inner loop is iterated 1 time for the first outer loop iteration, 2 times for the 2nd outer iteration, etc. Thus, in the worst case, the number of comparisons is:

$$\begin{aligned}\sum_{i=1}^{n-1}\sum_{j=1}^{i} 1 &= \sum_{i=1}^{n-1} i \\ &= 1+2+\cdots+(n-1) \\ &= \frac{n(n-1)}{2}\end{aligned}$$

In the average case, it is half that (because on average the correct position for the insertion in each inner loop will be in the middle of the sorted part), i.e. $\frac{n(n-1)}{4}$

Hence average and worst case complexity is $O(n^2)$

Insertion Sort Stability

Consider what happens when two elements with the same value are in the array to be sorted.

Since the value inserted in each outer loop is the first value in the unsorted part of the array, the first occurrence of two copies of the same value will be taken for insertion before the second.

Further, in the inner loop, we walk down from the end of the sorted part of the array until we find the first location that is not strictly greater than the value to be inserted before inserting there. That means that we will not insert a later copy of a value before an earlier copy that has already been inserted.

Hence insertion sort is *stable*.