Insertion Sort

Vivek K. S., Deepak G.

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2017

Introduction

- The insertion sort has a different way of working although it still only offers $O(n^2)$.
- It maintains a sorted sublist in the lower positions of the list.
- Each new item is then inserted back into this sublist such that the sorted sublist grows by one item at a time.
- In the beginning, we start with a list of one item at position 0, which is already sorted.
- Every pass, through items from position 1 through n-1, compares the current item against those in the sorted sublist.
- The items that are greater are shifted to the right.
- When an item that is smaller than the current item is encountered, it can be inserted.

How does it work?

- Imagine a list [12,24,43,45,64,81] which as we can see is already sorted. This is the sorted sublist that we have.
- Lets say we want to insert 44 into this list.
- The current item 44 will be compared against each item of the sorted sublist.
- Initially, 81 will be moved to the right as it is larger than 44.
- The same happens with 45 and 64.
- When it encounters 44, which is smaller than 45, it performs an insertion.
- Animation - http://www.cs.armstrong.edu/liang/animation/web/InsertionSort.htr

Contd

- There are n-1 passes to be made for a list of n items again.
- The iterations starts at 1 and moves through position n-1. Note that position 0 is already sorted when we start.
- The maximum number of comparisons for an insertion sort is the sum of the first n-1 integers.
- Again, this is O(n^2).
- It is well worth noticing that here, we are shifting instead of exchanging.
- In general, shifting requires only a third of the processing power required for exchanging since only one assignment is performed.

Python Implementation

```python
def insertion_sort(a_list):
    for index in range(1, len(a_list)):
        current_value = a_list[index]
        position = index
        while position > 0 and
        a_list[position - 1] > current_value:
            a_list[position] = a_list[position - 1]
            position = position - 1
        a_list[position] = current_value
```

Notice that we are not swapping the items but simply assigning them to the target position.

Summary

- Insertion Sort has a different approach towards sorting as compared to Bubble sort.
- There is no significant performance gain as it still depends on passes and insertion of item into a sorted sublist.
- Thereby it still offers only O(n^2).