INSERTION SORT

Algorithm: Insertion sort (A)

```
1  for j = 0  to  A.length
2      key = A[j]
3      i = j - 1
4      while (i >= 0 & A[i] > key)
5          A[i+1] = A[i]
6          i = i - 1
7          A[i+1] = end while
8      A[i+1] = key
9  end for
```

Cost of line	No. of times Run
c_1	n
c_2	$n-1$
c_3	$n-1$
c_4	$\sum_{i=1}^{n-1} (t_i)$
c_5	$\sum_{i=1}^{n-1} (t_i - 1)$
c_6	$\sum_{i=1}^{n-1} (t_i - 1)$
c_7	$n-1$

$$T(n) = c_1 n + (c_2 + c_3)(n-1) + \sum_{i=1}^{n-1} (t_i) c_4 + (c_5 + c_6) \sum_{i=1}^{n-1} (t_i - 1) + c_8 (n-1)$$

$$\Rightarrow c_n \left(n + 3(n-1) + \frac{n(n-1)}{2} \right)$$

$$\Rightarrow (n^2 - 1 + n^2 - n) \Rightarrow O(n^2)$$

Optimizing further

- We can optimize the searching by using binary search.

which will improving the searching complexity from O(n) to O(log n) for one element and to n* O(log n) or O(n log n) for n elements.

But since it will take O(n) for one element to be placed at the correct position, n elements will take n* O(n) or $O(n^2)$ time for being placed at their right places hence the overall complexity remains $O(n^2)$.

- We can optimize the swapping by Doubly link list.

Instead of Array that will improve the complexity of swapping from O(n) to O(1) as we can insert an element in a linked list by changing pointers (without shifting rest of element) But since the complexity to search remains $O(n^2)$ as we cannot use binary search in link list Hence overall complexity remain same.