Insertion_sort

We take an element, compare its value with left-side element,
if subsequent left side element is greater than selected element,
we shift the element to right,
we will compare and shift right until any left subsequent element is less than selected element or we have reach zeroth element, and then insert the element at that position.

eg:

4	2	7	1	3

↑

we start to loop through (pass through) from 1^{st} index.

we will store the index position and element value.

position = index

temp_value = arr[index]

we will compare with left subsequent value.

position = 1

temp_value = 2

4	2	7	1	3

is 4 > 2 ? yes

then, shift '4' to right, and update the position.

position = 0

temp_val = 2

4	4	7	1	3

Here we have reached left most position and nothing to compare on left side, So insert the temp-val at the position.

2	4	7	1	3

→ we can say that these element are sorted in order.

Now In 2^{nd} passthrough,

2	4	7	1	3

↑

position = 2 , temp_val = 7

we will compare the left subsequent value is 4 > 7 ? No , Hence pass-through ends

2	4	7	1	3
↑			↑	

→ after 2nd passthrough, we can say elements are sorted till 2nd index.

In 3rd passthrough,

2	4	7	1	3
			↑	

position = 3
temp_val = 1

is 7 > 1 ? Yes

we will shift 7 to right, and update the position.

2	4	7	7	3
		↑		

position = 2
temp_val = 1

is 4 > 1 ? Yes

shift 4 to right , update position.

2	4	4	7	3

position = 1

↑

temp_val = 1

is 2 > 1 ? yes

shift 2 to right, update position.

2	2	4	7	3

↑

position = 0

temp_val = 1

Nothing to compare on left, Hence insert temp_val in the position.

1	2	4	7	3

→ after 3rd passthrough, we can say, elements are sorted till 3rd index.

In 4th passthrough,

1	2	4	7	3

↑

position = 4

temp_val = 3

is 7 > 3 ? yes

shift 7 to right, update position

1	2	4	7	7

position = 3

↑ temp_val = 3

is 4 > 3 ? yes

shift 4 to right , update position

1	2	4	4	7
		↑		

position = 2

temp_val = 3

is 2 > 3 ? NO , so we will insert temp_val at position.

1	2	3	4	7

So, after 4 passthrough we can say array is sorted.

∴ For N elements, it will take (N-1) pass through

∴ After each passthrough, we can say elements till passthrough index are sorted.

Implementation :

```
func InsertionSort (arr)
 {
    for ( index = 1 → len(arr) -1 ) {
       position = index
    temp_val = arr[index]

   while( position > 0 &&
     arr[position] > temp_val )
    {
      arr[position] = arr[position-1]
       position--;
    }

   arr[position] = temp_val;

}
```

<u>Efficiency</u> :

There are 3 types of steps that occur, Comparison, shifting & Insertion.

(i). comparison - In worst case scenario, we will be comparing all the left-side elements

$\therefore \frac{N(N-1)}{2}$ steps.

(ii). shifting - in worst case, we will be shifting all the left-side element to right for each passthrough

$\therefore \quad \frac{N(N-1)}{2}$ steps

(III). Insertion - for each shift, insertion happens twice

$\therefore \quad 2(N-1)$ steps.

$$(I) + (II) + (III) \Rightarrow \frac{N(N-1)}{2} + \frac{N(N-1)}{2} + 2(N-1)$$

$$\Rightarrow N^2 - \cancel{N} + N^2 - \cancel{N} + 2\cancel{N} - 2$$

$$\Rightarrow N^2 - 2 \quad \text{(approximately)}$$

$\therefore$ Insertion sort is Big $O(N^2)$.

$O(N^2)$ also known as quadratic times

Another major rule.

Big O Notation takes into account the highest order of N.

eg: $N^4 + N^3 + N^2 + N$

N^4 will be taken into account has all other is less significant compare to N^4.
∴ It is take Big $O(N^4)$

The average case:

- In worst case: selection sort > Insertion sort

- The cases that occurs most frequently are average scenario.
- worst and best case occurs rarely.
- Insertion sort
  - worst-case : $O(N^2)$
  - Average-case : $O(N^2/2)$
  - Best-case : $O(N)$
- selection sort

$$\left.\begin{array}{l}\cdot\ \text{wort - case}\\ \cdot\ \text{Anerage - case}\\ \cdot\ \text{Best - case}\end{array}\right\}\ O(N^2)$$

- selection sort – Don't have any mechanism to end passthrough early at any point. Each passthrough compares value to its right no matter what.

- Insertion sort - when the values of left-side is sorted, no shifting is done, thus shifting steps are reduced.

→ what is best ?

It depends,

- when array is randomly sorted then insertion sort.
- when array is sorted in reverse order then selection sort.
- when no idea, how data will be present, then on average case both will work fine.