Insertion sort

In this sorting technique, we consider two parts - one is sorted part and another is unsorted one. And we make the whole array sorted by placing elements from the unsorted part to sorted part.

Algorithm :-

Step-1 Start

Step-2 Take input for an array

Step-3 Consider 0th index element as sorted

Step-4 Compare the elements and Loop through array from 1 to n-1 (where n is no. of elements / indices)

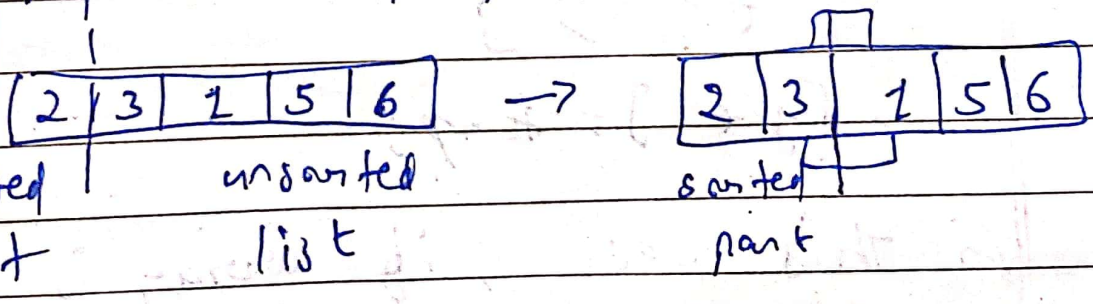
Step-5 Put the unsorted elements into sorted list in proper order i.e. ascending or descending

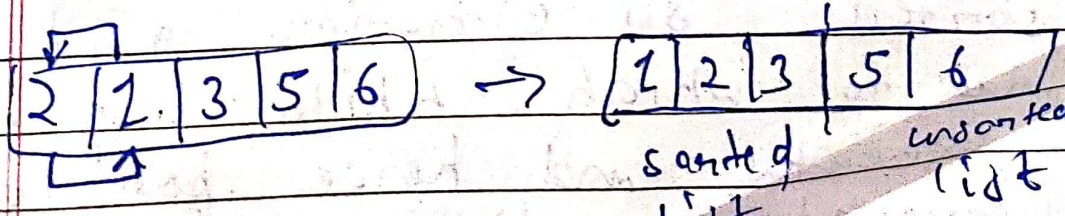
Step-6 Display sorted array

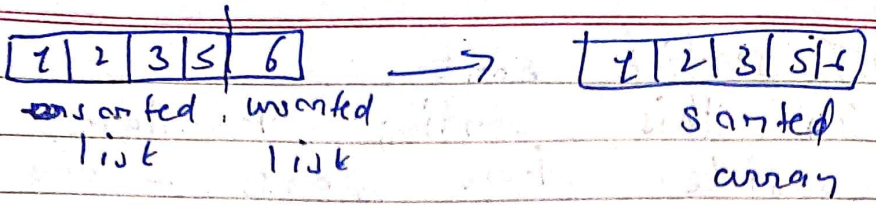
Step-7 End

Example:

Input → 2, 3, 1, 5, 6







Complexity :
Worst case - $O(n^2)$
Average case - $O(n^2)$
Best case - $O(n)$

For the best case, array is already sorted and hence only 1 main loop iteration takes place.

Program for insertion sort

```
for (i = 1 ; i<n; i++) // n is no. of elements
{ temp = a[i]          // temp is used to
  j = i - 1;             put the sorted
                         element at app. place
while (j >= 0 && a[j] > temp)
    { a[j+1] = a[j];
      j--; }
  a[j+1] = temp;
```

In this case, if array is already sorted, so everytime $a[j] < temp$ due to which while loop never executes and hence for loop executes n times. Therefore,

time complexity of insertion sort (best case) is $O(n)$.

Optimization
Searching and swapping are 2 main func.

① We can optimise swapping by using LL (Doubly linked list) which will reduce complexity of swapp. n elements from $O(n)$ to $O(1)$.
Although we can insert an element in it but still searching remain of the order of complexity $O(n^2)$ as we cannot use binary search in linked list. Thus, overall complexity is $O(n^2)$

② Searching by using BST reduce search complexity from $O(n)$ to $O(\log n)$ for one element and for n element $O(n \log n)$.