2 Insertion Sort

[5, 3, 1, 4, 7, 2]

j   i key:3

[5, 5, 1, 4, 7, 2]

j

[3, 5, 1, 4, 7, 2]

j   i, key:1

[3, 5, 5, 4, 7, 2]

j

[3, 3, 5, 4, 7, 2]

j

[1, 3, 5, 4, 7, 2]

j

$\downarrow$

[1, 2, 3, 4, 5, 7]

j   i, key:7

Invariant: Insertion sort

For any given value of i in the main loop of insertion sort, at the beginning of the iteration, the following invariant holds:

1. $array[1..i-1]$ is sorted

LI: arr[1...0] is sorted

for (i=2, i<=n, i++):

LI: arr[1... i-1] is sorted

OR

LI: arr[1... j] is sorted

key = arr[i]

j = i-1

while arr[j] > key and j > 0:

arr[j+1] = arr[j]
j -= 1

arr[j+1] = key LI: arr[j+1 ... i] is sorted

OR

LI: arr[1 ... i] is sorted

LI: arr[1 ... i] is sorted
i = n, so arr[1 ... n] is sorted

Code:

```
def insertionSort(arr, n):

    for i in range( 1, n ):
        key = arr[ i ]
        j = i-1

        while arr[ j ] > key and j >= 0:
            arr[ j+1 ] = arr[ j ]
            j -= 1

        arr[ j+1 ] = key

    return arr
```

Time:

If input arr in reverse order: $T(n) = n + (n-1) + (n-2) + \dots + 1$

$$T(n) = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$$

Worst-Case: $O(n^2)$

If input arr is alread sorted: $T(n) = n$

Best-case: $O(n)$

Space: $O(1)$ In-place: Yes, as aux space is $O(1)$

Stable: Yes, because arr[j] > key comparison makes sure that the last element is the sorted sub-array is greater than equal to (>=) the first item in unsorted sub-array OR $arr[j] \geq arr[j+1]$ OR $arr[i-1] \geq arr[i]$. This maintains the relative order of items with same value.

Online: Yes, because the algo. can process its input array

sequentially without knowing it all in advance.
Insertion sort can start sorting even if new items are being added at the end of input array.